

Curso Fundamentos Dynamo

DARCO

Dibujo Arquitectónico por Computadora S.A. de C.V.
Ejército Nacional 373-403 Col. Granada
Ciudad de México - CP 11520
Tel/Fax (+5255) 5545-3550



AUTODESK
Authorized Training Center
Certification Center

Contenido

Autodesk Dynamo	8
INTRODUCCIÓN.....	9
¿Qué es la Programación Visual?.....	10
Algoritmos en mano	10
Instrucciones textuales:.....	10
Instrucciones gráficas:.....	11
Programación Definida.....	11
Programación Visual Definida.....	11
Programa visual:.....	12
Programa Textual:.....	12
¿Qué es Dynamo?	13
La aplicación.....	13
El proceso.....	13
La plataforma	14
¡HOLA DYNAMO!.....	16
Instalación y lanzamiento de Dynamo	17
Descargando	17
Instalación	19
Lanzamiento	19
La interfaz de usuario Dynamo	21
Menús.....	22
Barra de herramientas.....	23
Biblioteca	24
Hojeada	24
Buscando.....	27
Configuraciones.....	28
Ayuda	29
El espacio de trabajo.....	30
Pestañas.....	31
Gráfico versus navegación de vista previa 3D.....	32
Zoom a Recenter.....	33
¡Hola, ratón!.....	35
Búsqueda en lienzo.....	36
Limpiar el diseño del nodo	36
Antes de la limpieza del nodo	37
Después de la limpieza del nodo	38
Empezando	38
Definición de objetivos y relaciones	38

Agregar nodos al espacio de trabajo.....	39
Conexión de nodos con cables	42
Ejecutando el programa	42
Agregar detalles	43
Agregar complejidad.....	44
Ajuste con manipulación directa.....	45
ANATOMÍA DE UN PROGRAMA VISUAL	48
Nodos	49
Anatomía de un nodo	49
Puertos	49
Estados.....	50
Alambres	52
Flujo de programa	52
Creando cables	52
Edición de cables	53
Vistas previas de cables.....	54
Biblioteca Dynamo	56
Biblioteca de Bibliotecas	56
El esquema organizacional	56
Convenciones de nombres.....	57
Nodos usados con frecuencia	59
Entrada	59
Reloj	59
Bloque de código.....	60
Administrar su programa	61
Alineación	61
Notas	62
Agrupamiento	64
Administrar sus datos con ajustes preestablecidos	66
Presets	66
Creando Presets.....	67
Restauración de ajustes preestablecidos	69
Eliminar Presets	69
LOS BLOQUES DE CONSTRUCCIÓN DE PROGRAMAS.....	71
Datos	72
¿Qué es Data?	72
Cuidado con los nulos	72
Estructuras de datos.....	73
Usar datos para hacer una cadena de cilindros.....	74

Matemáticas	83
Operadores aritméticos	83
Fórmula paramétrica	84
De la fórmula a la geometría	86
De Espiral a Nautilus	88
Del Nautilus al Patrón Phyllotaxis	90
Lógica	93
Booleanos	93
Declaraciones condicionales	93
Filtrando una lista	95
De la Lógica a la Geometría	96
Instrumentos de cuerda	101
Creando cadenas	101
Consulta de cadenas	102
Manipulación de cadenas	105
Trabajando con cadenas	107
Color	111
Creando colores	111
Consultar los valores de color	112
Gama de colores	113
Vista previa de color	114
Ejercicio de color	115
Color en las superficies	125
Ejercicio de color en superficies	125
GEOMETRÍA PARA EL DISEÑO COMPUTACIONAL	129
Resumen de Geometría	130
Los básicos	130
Pasando por la jerarquía	130
Geometría en Dynamo Studio	132
Yendo más allá con la geometría	134
Vectores, planos y sistemas de coordenadas	134
¿Qué es un Vector?	135
¿Qué es un plano?	138
¿Qué es un sistema de coordenadas?	140
Puntos	141
¿Qué es un punto?	142
Señalar como coordenadas	142
Curvas	143
¿Qué es una curva?	144

Líneas.....	144
Arcos, círculos, arcos de elipse y elipses	146
NURBS + Polycurves	146
Superficies.....	148
¿Qué es una superficie?	148
Superficies NURBS	150
Polisuperficies	151
Sólidos.....	153
¿Qué es un sólido?	153
Topología.....	154
Operaciones	155
Operaciones Booleanas	155
Congelación.....	157
Mallas	160
¿Qué es una malla?	160
Elementos de malla	161
Vértices + vértices normales	162
Caras	162
Mallas frente a superficies NURBS	163
Parametrización.....	163
Influencia local versus global	164
Juego de herramientas Mesh.....	164
Importación de geometría	165
Importación de geometría desde un archivo DWG	165
Obtener objetos importados	165
Filtro de objeto.....	166
Selección explícita de objetos	168
DISEÑANDO CON LISTAS.....	170
¿Qué es una lista?	171
Índices basados en cero	171
Entradas y salidas	172
Cordones	173
Archivo base.....	174
Lista más corta	175
La lista más larga	176
Producto cruzado	177
Trabajando con listas	178
Operaciones de lista.....	179
Ejercicio - Operaciones de lista.....	179

List.Count	180
Ejercicio - List.Count.....	180
List.GetItemAtIndex.....	181
Ejercicio - List.GetItemAtIndex	181
List.Reverse.....	182
Ejercicio - List.Reverse.....	182
List.ShiftIndices	183
Ejercicio - List.ShiftIndices	184
List.FilterByBooleanMask.....	185
Ejercicio - List.FilterByBooleanMask	185
Listas de listas	186
Jerarquía descendente.....	188
Ejercicio - Jerarquía descendente.....	188
Acoplar y List.Flatten	189
Ejercicio: aplanar	189
Picar	190
Ejercicio - List.Chop.....	190
List.Map y List.Combine	191
Ejercicio - Lista.Mapa	191
Ejercicio - List.Combine.....	194
List @ Level.....	197
Ejercicio List @ Level	198
Transponer	201
Ejercicio - List.Transpose	202
Code Block Creation.....	204
Consulta de bloque de código.....	204
Ejercicio: consultar e insertar datos	205
Listas n-dimensionales	211
Mapeo y Combinaciones.....	212
Ejercicio - Listas 2D – Básico.....	212
Ejercicio - Listas 2D – Avanzado.....	216
Ejercicio - Listas 3D.....	221
CODE BLOCKS Y DESIGNSCRIPT	233
¿Qué es un bloque de código?	234
Bloque de código: una breve descripción	234
Crear nodos de bloque de código	235
Números, cadenas y fórmulas.....	236
Sintaxis de DesignScript	237
Notación de puntos en el bloque de código	238

Nodos de llamada.....	238
Crear.....	239
Acción.....	240
Consulta.....	241
¿Qué tal el cordón?.....	242
Nodo para codificar.....	242
Ejercicio.....	243
Simplifica el gráfico con "Nodo al código".....	250
Taquigrafía.....	251
Sintaxis adicional.....	253
Rangos.....	253
Rangos avanzados.....	254
Haga listas y obtenga artículos de una lista.....	256
Ejercicio.....	257
Funciones de bloque de código.....	266
Padre.....	266
Niños.....	267
Ejercicio.....	268



DARCO
DESDE 1988

Autodesk Dynamo

Este manual es un proyecto en curso para compartir los fundamentos de la programación. Los temas incluyen trabajar con geometría computacional, mejores prácticas para el diseño basado en reglas, aplicaciones de programación interdisciplinarias y más con la plataforma Dynamo.

El poder de Dynamo se puede encontrar en una amplia variedad de actividades relacionadas con el diseño. Dynamo habilita una lista en expansión de formas fácilmente accesibles para que pueda comenzar:

- **Explore** la programación visual por primera vez
- **Conecte** flujos de trabajo en varios programas
- **Involucrar a** una comunidad activa de usuarios, contribuidores y desarrolladores
- **Desarrollar** una plataforma de código abierto para la mejora continua

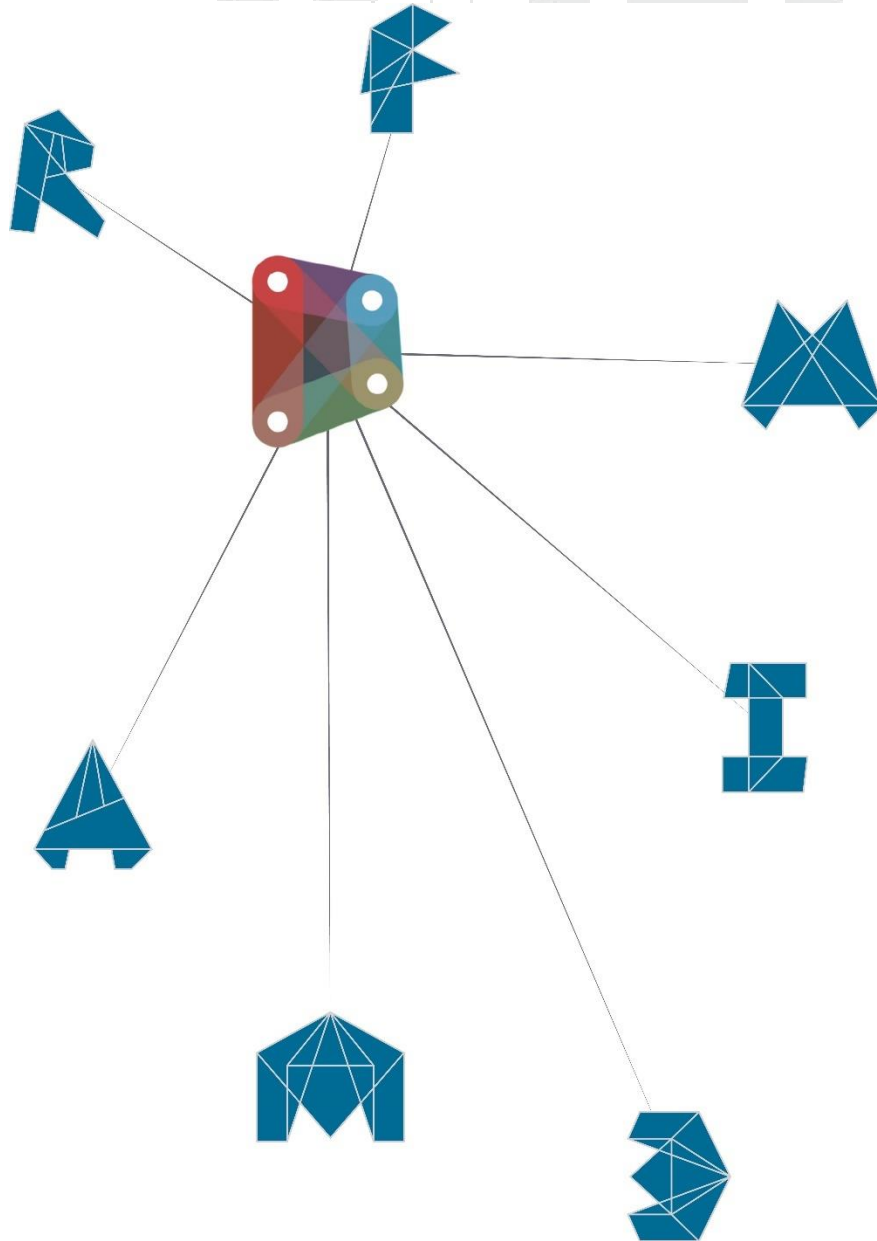
Estos capítulos se centran en los elementos esenciales que necesitará para comenzar a desarrollar sus propios programas visuales con Dynamo y puntos clave sobre cómo llevar Dynamo más lejos. Esto es lo que puede esperar aprender del manual:

- **Contexto:** ¿Qué es exactamente "Programación visual" y cuáles son los conceptos que necesito entender para sumergirme en Dynamo?
- **Comenzando** - ¿Cómo obtengo Dynamo y creo mi primer programa?
- **¿Qué hay en un programa?** ¿Cuáles son las partes funcionales de Dynamo y cómo los uso?
- **Building Blocks:** ¿qué son los "datos" y cuáles son algunos tipos fundamentales que puedo comenzar a utilizar en mis programas?
- **Geometry for Design** - ¿Cómo trabajo con elementos geométricos en Dynamo?
- **Listas:** ¿cómo puedo administrar y coordinar mis estructuras de datos?
- **Código en Nodos** - ¿Cómo puedo comenzar a extender Dynamo con mi propio código?
- **BIM computacional:** ¿cómo puedo usar Dynamo con un modelo de Revit?
- **Nodos personalizados:** ¿cómo puedo crear mis propios nodos?
- **Paquetes:** ¿cómo puedo compartir mis herramientas con la comunidad?

DARCO
DESDE 1988

INTRODUCCIÓN

Desde sus orígenes como complemento para Building Information Modeling en Revit, Dynamo ha madurado hasta convertirse en muchas cosas. Por encima de todo, es una plataforma que permite a los diseñadores explorar la programación visual, resolver problemas y crear sus propias herramientas. Comencemos nuestro viaje con Dynamo estableciendo un contexto: ¿qué es y cómo me acerco a usarlo?

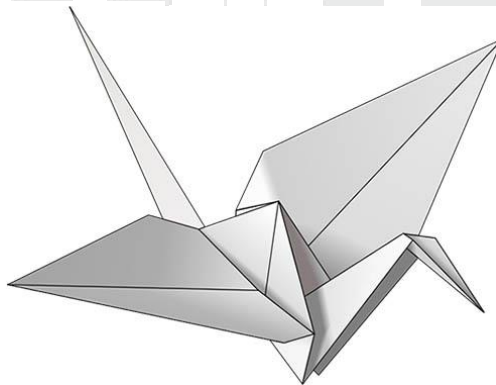


¿Qué es la Programación Visual?

Diseñar frecuentemente implica establecer relaciones visuales, sistémicas o geométricas entre las partes de un diseño. Más de las veces, estas relaciones se desarrollan mediante flujos de trabajo que nos llevan del concepto al resultado por medio de reglas. Quizás sin saberlo, estamos trabajando algorítmicamente, definiendo un conjunto de acciones paso a paso que siguen una lógica básica de entrada, procesamiento y salida. La programación nos permite seguir trabajando de esta manera, pero formalizando nuestros algoritmos.

Algoritmos en mano

Si bien ofrece algunas oportunidades de gran alcance, el término **Algoritmo** puede llevar algunos conceptos erróneos con él. Los algoritmos pueden generar cosas inesperadas, salvajes o geniales, pero no son mágicos. De hecho, son bastante simples, en sí mismos. Usemos un ejemplo tangible como una grulla de origami. Comenzamos con una hoja de papel cuadrada (entrada), seguimos una serie de pasos de plegado (acciones de procesamiento) y damos como resultado una grúa (salida).

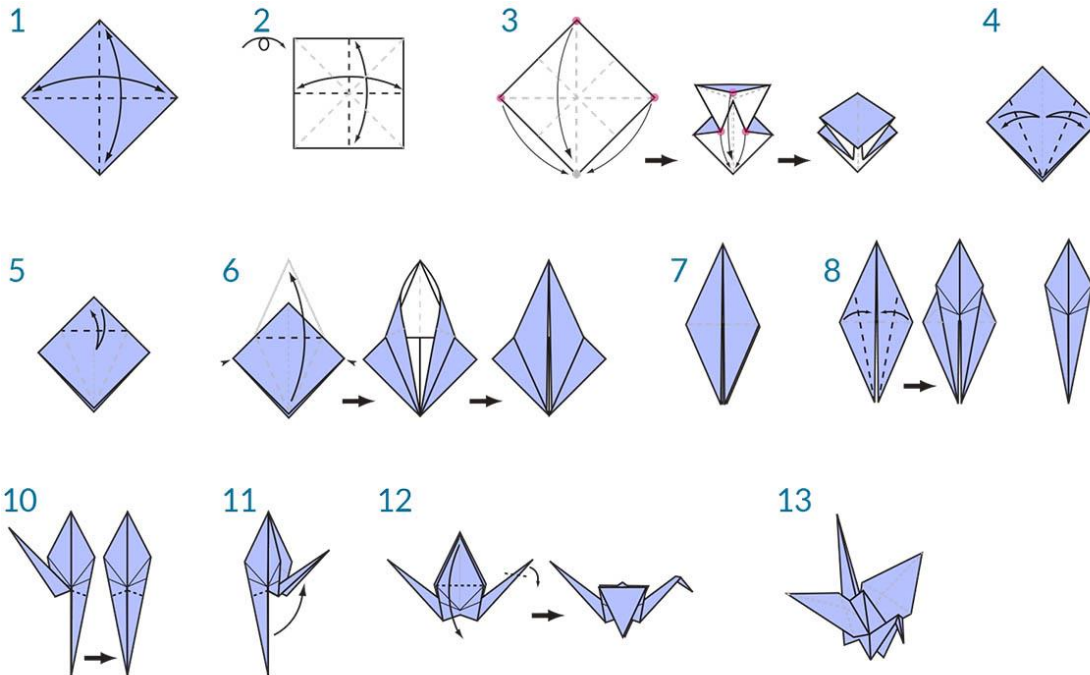


Entonces, ¿dónde está el algoritmo? Es el conjunto abstracto de pasos, que podemos representar de dos maneras: textual o gráficamente.

Instrucciones textuales:

1. Comience con un pedazo de papel cuadrado, con el lado de color hacia arriba. Doble por la mitad y ábralo. Luego doble por la mitad del otro lado.
2. Pase el papel al lado blanco. Doble el papel por la mitad, arrástralo bien y ábrelo, y luego vuelva a doblar en la otra dirección.
3. Usando los pliegues que ha hecho, lleve las 3 esquinas superiores del modelo a la esquina inferior. Aplana el modelo.
4. Doblar las aletas triangulares superiores en el centro y desplegar.
5. Pliegue la parte superior del modelo hacia abajo, doble bien y despliegue.
6. Abra la solapa superior del modelo, levantándola hacia arriba y presionando los lados del modelo hacia adentro al mismo tiempo. Aplanar, arrugar bien.
7. Dé la vuelta al modelo y repita los pasos 4 a 6 del otro lado.
8. Doble las solapas superiores hacia el centro.
9. Repita en el otro lado.
10. Doble las dos "patas" del modelo hacia arriba, arrástralas muy bien y luego despliega.
11. Reverso interior Doble las "piernas" a lo largo de los pliegues que acabas de hacer.
12. Reverso interior Doble un lado para formar una cabeza, luego doble las alas hacia abajo.
13. Ahora tienes una grúa.

Instrucciones gráficas:



Programación Definida

El uso de cualquiera de estos conjuntos de instrucciones debe dar como resultado una grúa, y si lo siguió, ha aplicado un algoritmo. La única diferencia es la forma en que leemos la formalización de ese conjunto de instrucciones y eso nos lleva a la **Programación**. La programación, frecuentemente acortada de *Programación de Computadora*, es el acto de formalizar el procesamiento de una serie de acciones en un programa ejecutable. Si convertimos las instrucciones anteriores para crear una grúa en un formato que nuestra computadora pueda leer y ejecutar, estamos programando.

La clave y el primer obstáculo que encontraremos en Programación es que tenemos que confiar en alguna forma de abstracción para comunicarnos efectivamente con nuestra computadora. Eso toma la forma de cualquier cantidad de lenguajes de programación, como Javascript, Python o C. Si podemos escribir un conjunto de instrucciones repetibles, como la grúa de origami, solo tenemos que traducirlo a la computadora. Estamos en camino de que la computadora pueda fabricar una grúa o incluso una multitud de grúas diferentes, donde cada una de ellas varía ligeramente. Este es el poder de la Programación: la computadora ejecutará repetidamente cualquier tarea, o conjunto de tareas, que le asignemos, sin demora y sin error humano.

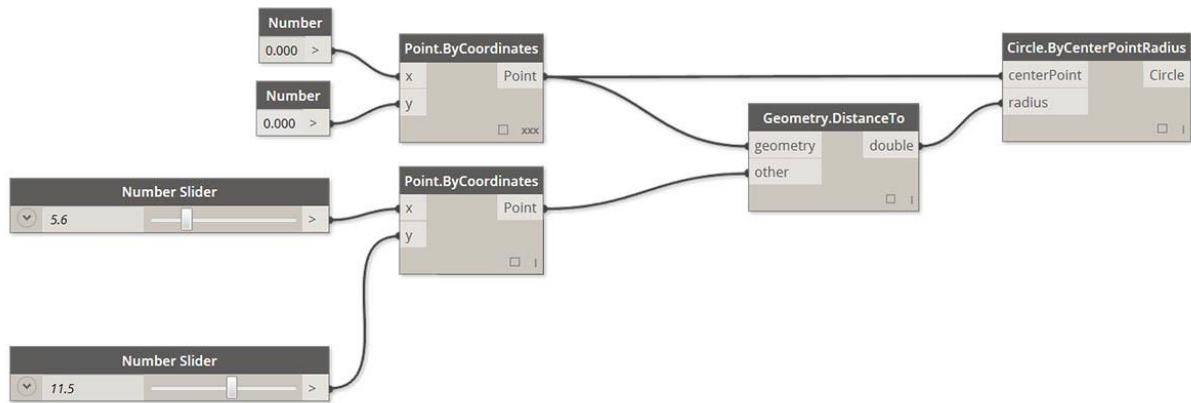
Programación Visual Definida

Descargue el archivo de ejemplo que acompaña a este ejercicio Programación visual - Circle Through Point.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

Si tuviera la tarea de escribir instrucciones para doblar una grúa de origami, ¿cómo lo haría? ¿Los harías con gráficos, texto o alguna combinación de los dos?

Si su respuesta contenía gráficos, entonces la **Programación Visual** es definitivamente para usted. El proceso es esencialmente el mismo tanto para Programación como para Programación Visual. Utilizan el mismo marco de formalización; sin embargo, definimos las instrucciones y relaciones de nuestro programa a través de una interfaz de usuario gráfica (o "Visual"). En lugar de escribir texto vinculado por sintaxis, conectamos nodos preempaquetados juntos. Aquí hay una comparación del mismo algoritmo - "dibujar un círculo a través de un punto" - programado con nodos versus código:

Programa visual:

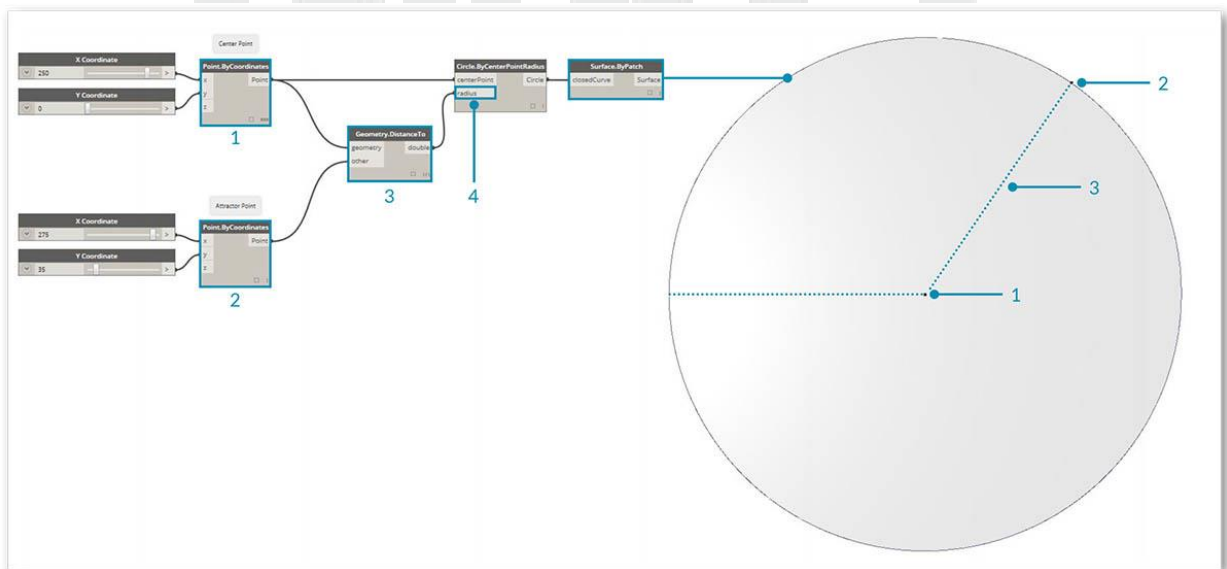


Programa Textual:

```

myPoint = Point.ByCoordinates(0.0,0.0,0.0);
x = 5.6;
y = 11.5;
attractorPoint = Point.ByCoordinates(x,y,0.0);
dist = myPoint.DistanceTo(attractorPoint);
myCircle = Circle.ByCenterPointRadius(myPoint,dist);
    
```

Los resultados de nuestro algoritmo:



La característica visual de programar de tal manera disminuye la barrera de entrada y frecuentemente habla a los diseñadores. Dynamo entra en el paradigma de Programación Visual, pero como veremos más adelante, aún podemos usar la programación textual en la aplicación también.

¿Qué es Dynamo?

Dynamo es, literalmente, lo que haces. Trabajar con Dynamo puede incluir el uso de la aplicación, ya sea en conexión con otro software de Autodesk o no, la participación de un proceso de Programación Visual o la participación en una amplia comunidad de usuarios y colaboradores.

La aplicación

Dynamo, la aplicación, es un software que se puede descargar y ejecutar en modo independiente "Sandbox" o como un complemento para otro software como Revit o Maya. Se describe como:

Una herramienta de programación visual que pretende ser accesible tanto para los no programadores como para los programadores. Les da a los usuarios la capacidad de guionizar visualmente el comportamiento, definir piezas de lógica personalizadas y secuencias de comandos usando varios lenguajes de programación textuales.

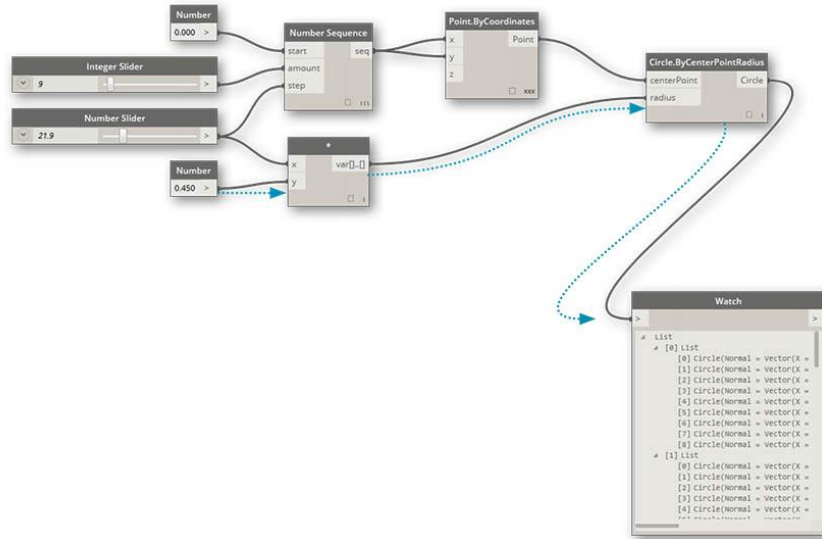
The screenshot shows the Dynamo website homepage. At the top, there is a navigation bar with links for 'Download', 'Learn', 'Blog', 'Gallery', and 'Community'. The main heading is 'Open source graphical programming for design'. Below this, there is a sub-heading: 'Dynamo extends building information modeling with the data and logic environment of a graphical algorithm editor.' and a button to 'Download 0.8.0'. To the right of the main text is a video player showing a 3D architectural model with a complex network of lines and nodes, with a blue arrow pointing to the video player. Below the video, there are three sections: 'Use Computational Design with BIM', 'Learn from a community of experts', and 'Get involved with Open Source', each with a corresponding image and a 'Learn' or 'Join' button.

1. Ver Dynamo en acción con Revit
2. Descargar el instalador

El proceso

Una vez que hayamos instalado la aplicación, Dynamo nos permitirá trabajar dentro de un proceso de Programación Visual donde conectaremos elementos para definir las relaciones y las secuencias de acciones que componen los algoritmos personalizados. Podemos utilizar nuestros

algoritmos para una amplia gama de aplicaciones, desde el procesamiento de datos hasta la generación de geometría, todo en tiempo real y sin escribir code.



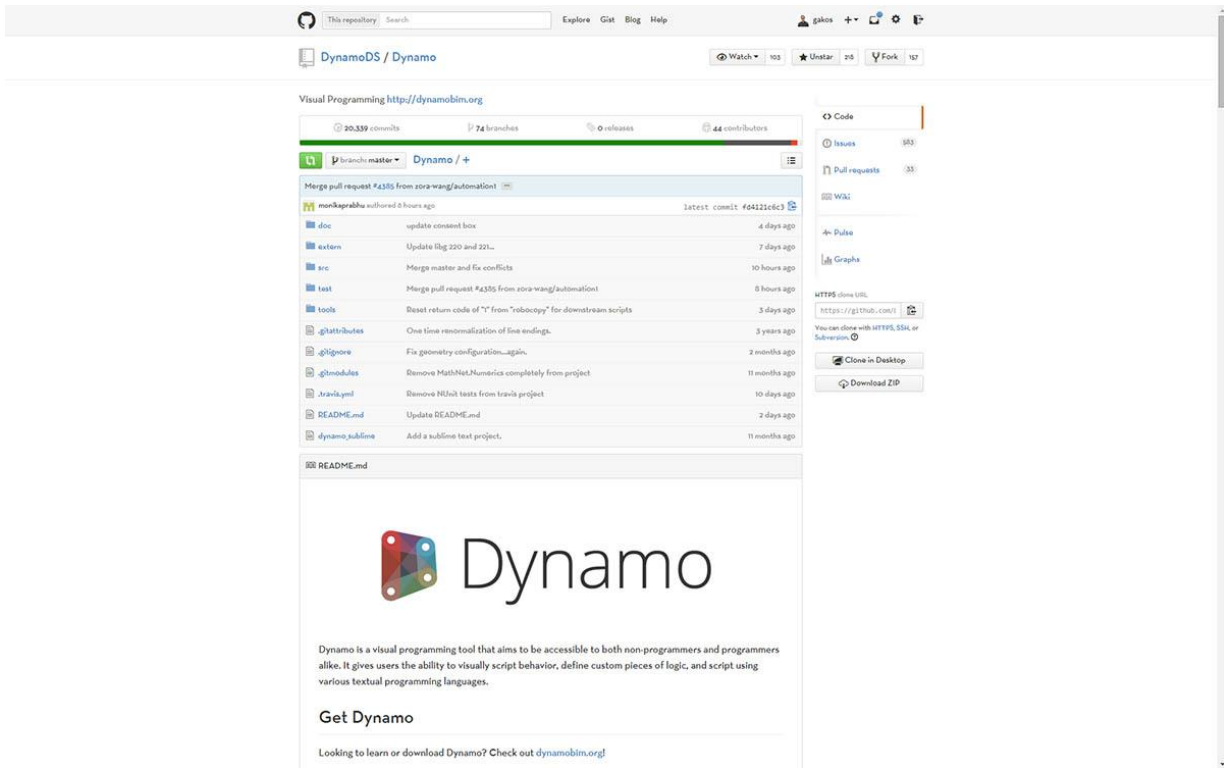
Agregue elementos, conéctese, y estamos listos para ejecutar Programas visuales.

La plataforma

Dynamo se concibe como una herramienta de programación visual para diseñadores, permitiéndonos hacer herramientas que hagan uso de bibliotecas externas o cualquier producto de Autodesk que tenga una API. Con Dynamo Studio podemos desarrollar programas en una aplicación de estilo "Sandbox", pero el ecosistema de Dynamo sigue creciendo.

El código fuente del proyecto es de código abierto, lo que nos permite extender su funcionalidad al contenido de nuestro corazón. Consulte el proyecto en Github y explore Works in Progress de usuarios que personalizan Dynamo.

DARCO
DESDE 1988



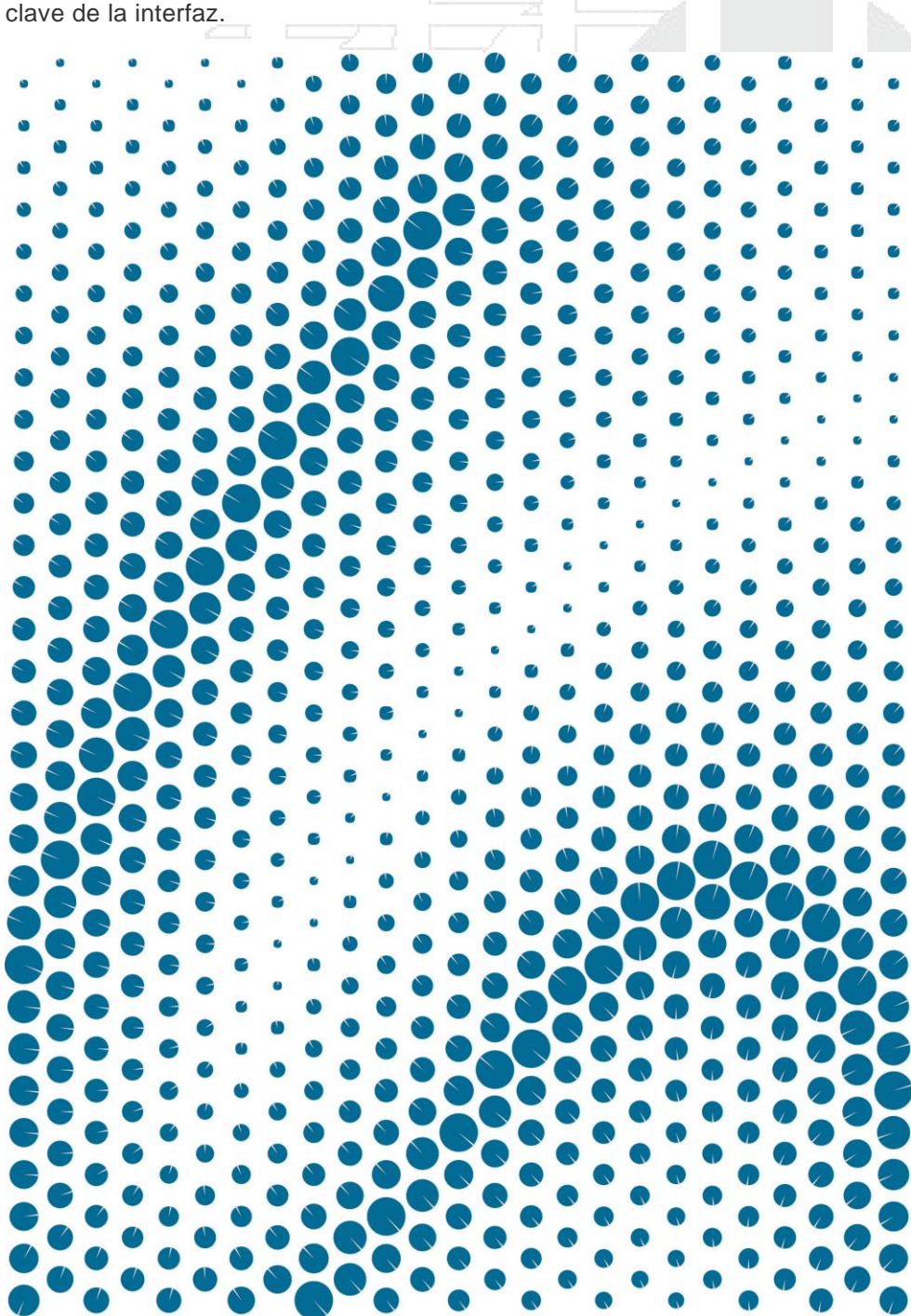
Navega, bifurca y comienza a extender Dynamo para tus necesidades



DARCO
DESDE 1988

¡HOLA DYNAMO!

En su núcleo, Dynamo es una plataforma para la Programación Visual: es una herramienta de diseño flexible y extensible. Debido a que puede funcionar como una aplicación independiente o como complemento de otro software de diseño, podemos usarlo para desarrollar una amplia gama de flujos de trabajo creativos. Instalemos Dynamo y comencemos por revisar las características clave de la interfaz.

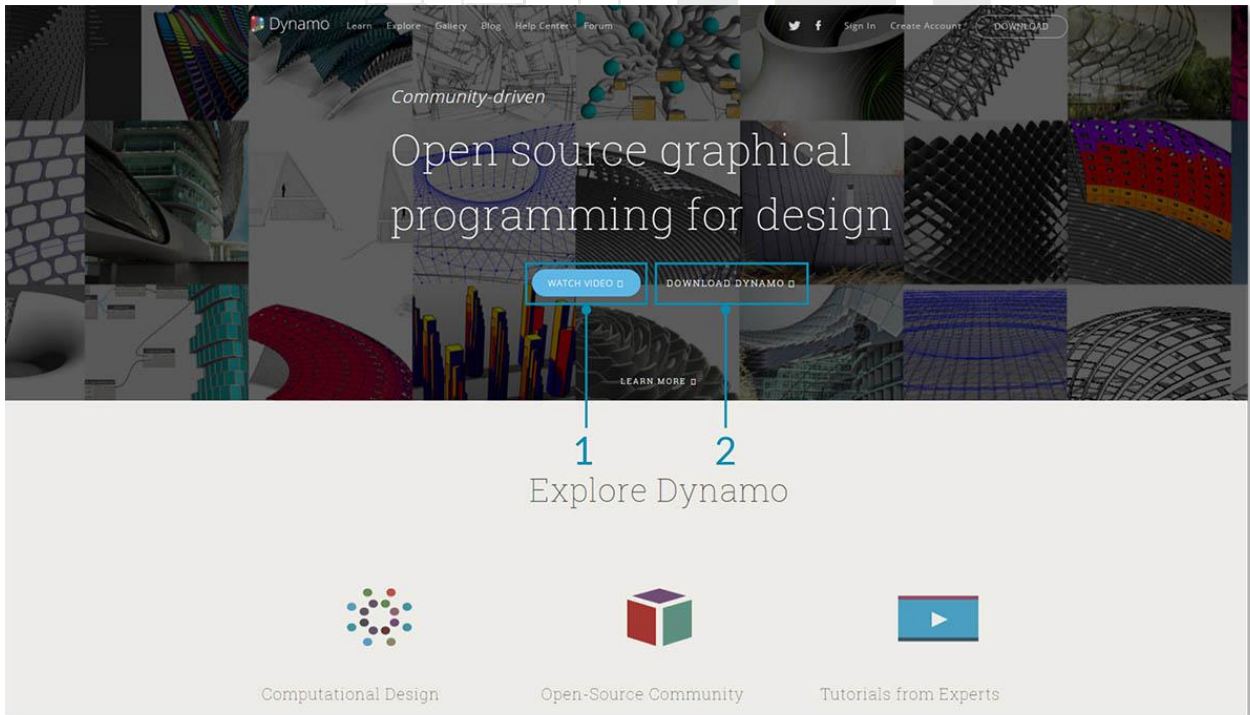


Instalación y lanzamiento de Dynamo

Dynamo es un proyecto de desarrollo de código abierto activo con instaladores descargables tanto para oficiales como para versiones preliminares, es decir, versiones "compilación diaria". Descargue el lanzamiento oficial para comenzar o contribuya a lo que Dynamo se convierte a través de las versiones diarias o el proyecto Github.

Descargando

Para descargar la versión oficial de Dynamo, visite el sitio web de Dynamo. Inicie la descarga de inmediato haciendo clic desde la página de inicio o navegue hasta la página de descarga dedicada.



1. Vea un video sobre Diseño Computacional con Dynamo para Arquitectura
2. O navega hacia la página de descarga

Aquí puede descargar las versiones de desarrollo "vanguardista" o ir al proyecto Dynamo Github.

This is Dynamo

Dynamo

Open-source Dynamo is a visual programming extension for Autodesk® Revit that allows you to manipulate data, sculpt geometry, explore design options, automate processes, and create links between multiple applications.

- Rapid design iteration and broad interoperability
 - Lightweight scripting interface
 - Works with Autodesk Revit
 - Free with Revit 2015, 2016, and 2017

1

[DOWNLOAD](#)

Version 1.2.0

DYNAMO STUDIO

Autodesk® Dynamo Studio is a visual programming platform that functions fully independently of any other application. Employ all the power of visual programming without buying another Revit license.

- Rapid design iteration and broad interoperability
 - Lightweight scripting interface
 - Direct access to cloud services
 - Includes advanced geometry engine

[BUY OR TRY](#)

Version 1.2.0 (with upgrade from 1.1.0)



Pre-Release Daily Builds

This is the bleeding edge of our development process, constantly getting new features and fixes. Help us improve it and check the ReadMe for known issues.

2

[DynamoInstall1.2.1.20161025T0057.exe](#)

[DynamoInstall1.2.1.20161024T0911.exe](#)

[DynamoInstall1.2.1.20161024T0503.exe](#)

Get Involved with Open Source

Dynamo is an open source tool, which means we need you to help us make it better!

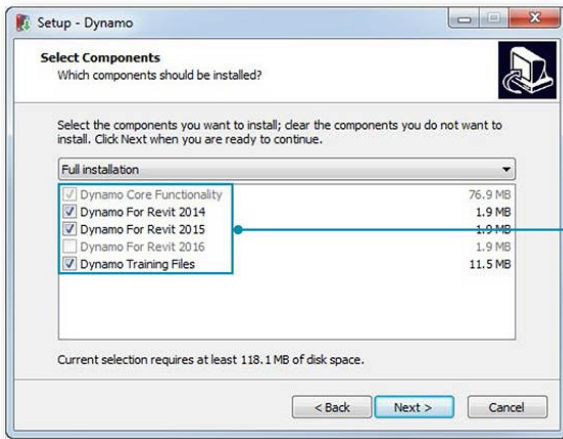
3

[JOIN THE COMMUNITY](#)

1. Descargue el instalador oficial de la versión
2. Descargue los instaladores de compilación diaria
3. Participe en el desarrollo de Dynamo en Github

Instalación

Busque el directorio del instalador descargado y ejecute el archivo ejecutable. Durante el proceso de instalación, la configuración le permite personalizar los componentes que se instalarán.

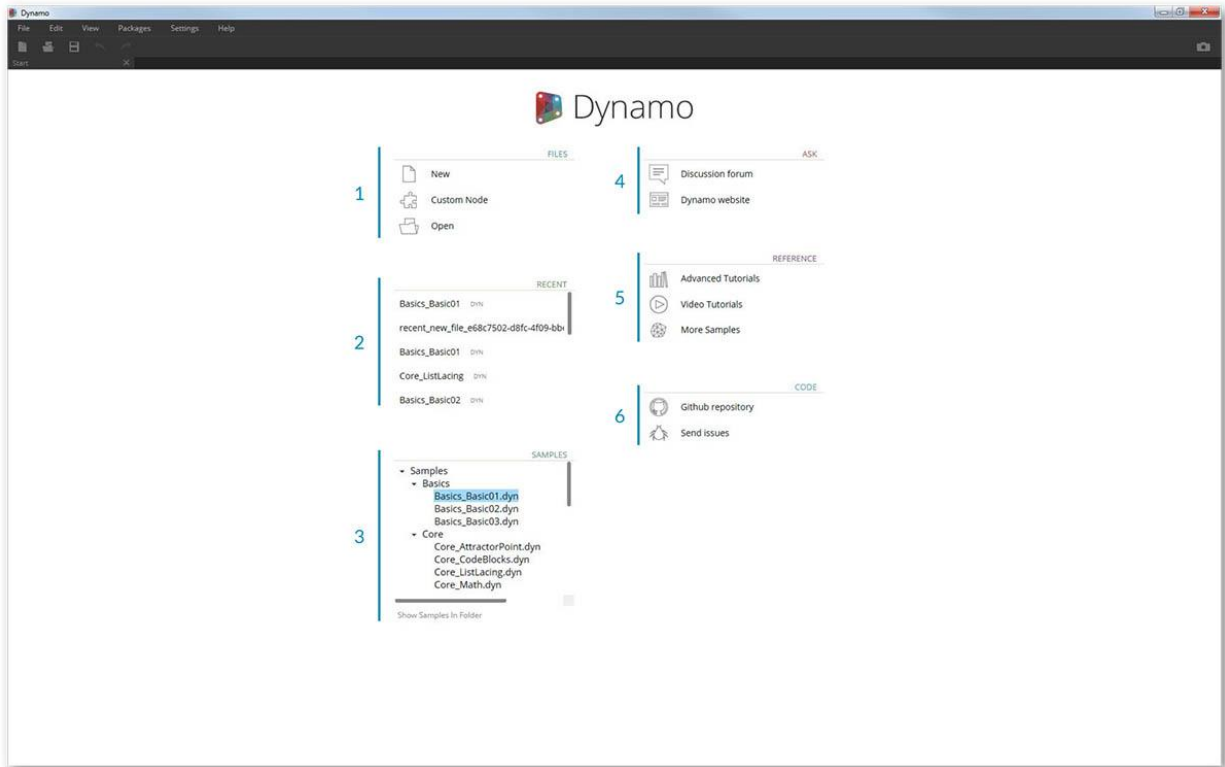


1. Seleccione los componentes que desea instalar

Aquí debemos decidir si queremos incluir los componentes que conectan a Dynamo con otras aplicaciones instaladas, como Revit.

Lanzamiento

Para iniciar Dynamo, vaya al Menú de Inicio de Windows> Dynamo> **Dynamo**. Esto abrirá la versión independiente y presentará la *Página de inicio* de Dynamo. En esta página, vemos los menús estándar y la barra de herramientas, así como una colección de atajos que nos permiten acceder a la funcionalidad del archivo o acceder a recursos adicionales.

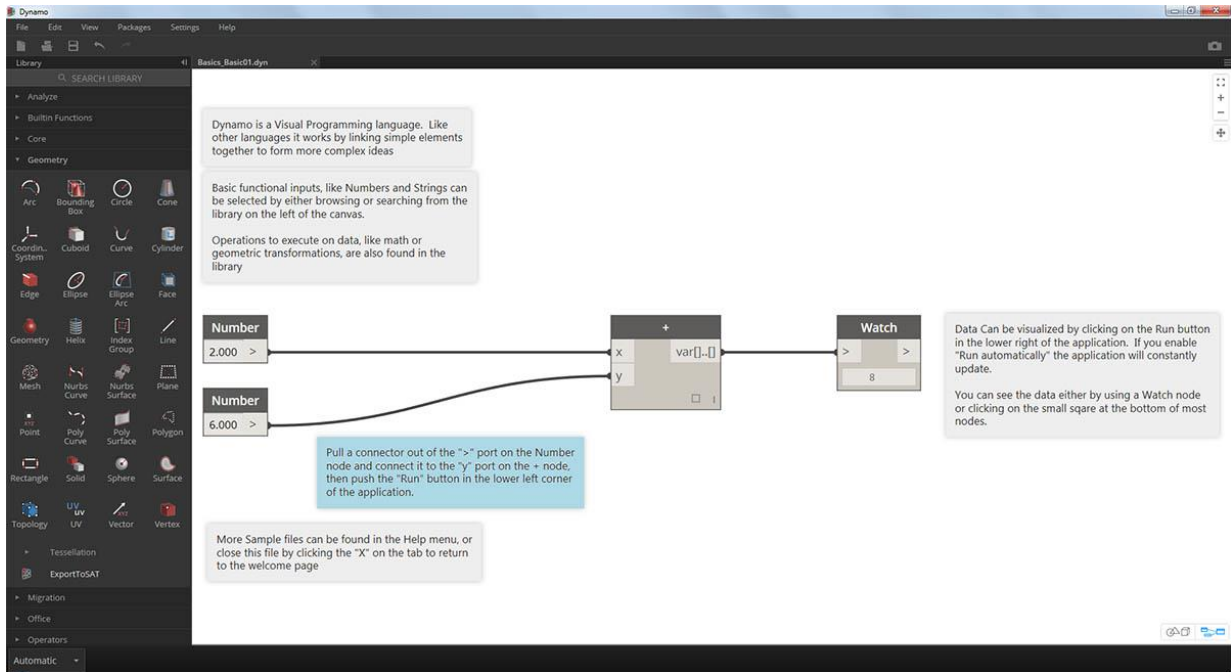


1. Archivos: comience un nuevo archivo o abra uno existente
2. Recientes: desplácese por sus archivos recientes
3. Muestras: echa un vistazo a los ejemplos que vienen con la instalación
4. Preguntar: obtenga acceso directo al Foro de usuarios o al sitio web de Dynamo.
5. Referencia: ir más allá con recursos de aprendizaje adicionales
6. Código: participe en el proyecto de desarrollo de código abierto

Abra el primer archivo de muestra para abrir su primer espacio de trabajo y confirme que Dynamo funciona correctamente. Haga clic en Muestras > Conceptos básicos > **Basics_Basic01.dyn**.

DARCO

DESDE 1988

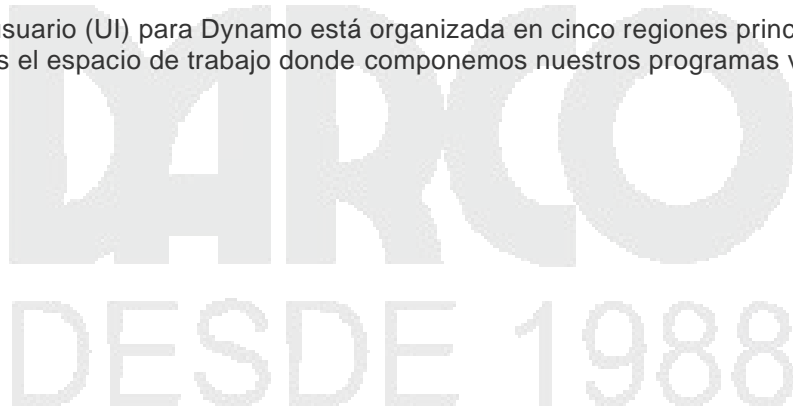


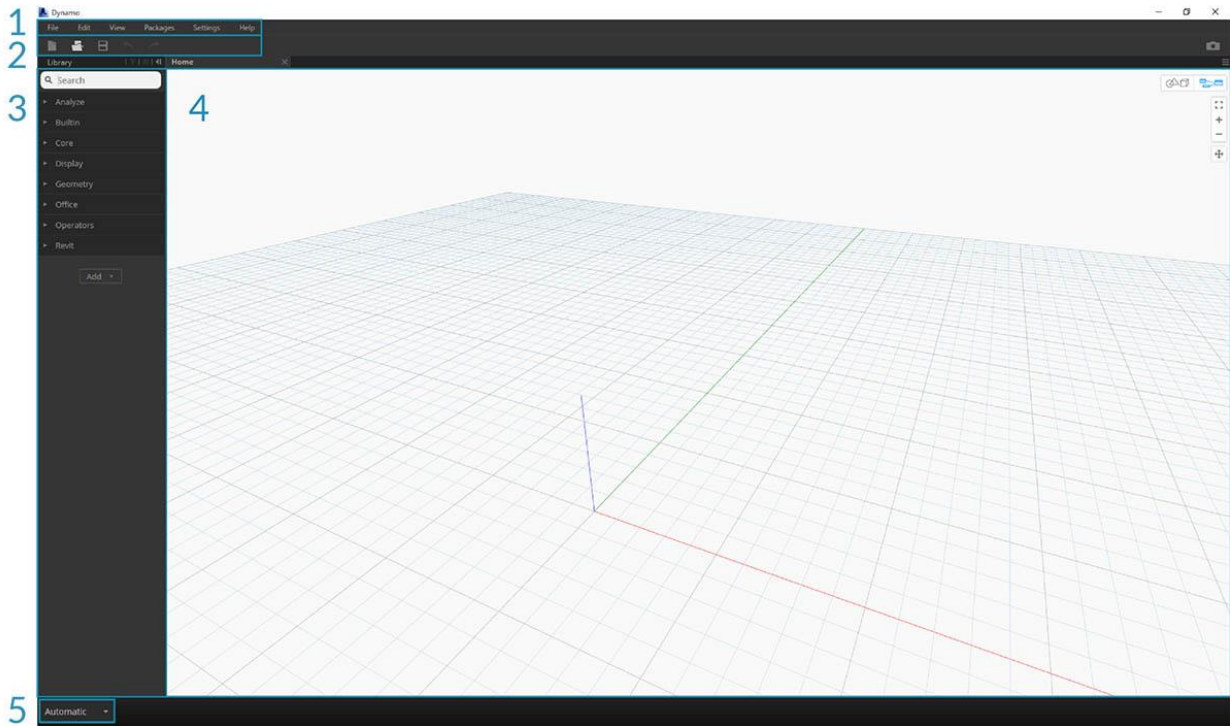
1. Confirme que la barra de ejecución dice "Automático" o haga clic en Ejecutar
2. Siga las instrucciones y conecte el **número** de nodos a la + Nodo
3. Confirme que este Nodo de vigilancia muestra un resultado

Si este archivo se carga correctamente, debería poder ejecutar su primer programa visual con Dynamo.

La interfaz de usuario Dynamo

La interfaz de usuario (UI) para Dynamo está organizada en cinco regiones principales, la mayor de las cuales es el espacio de trabajo donde componemos nuestros programas visuales.



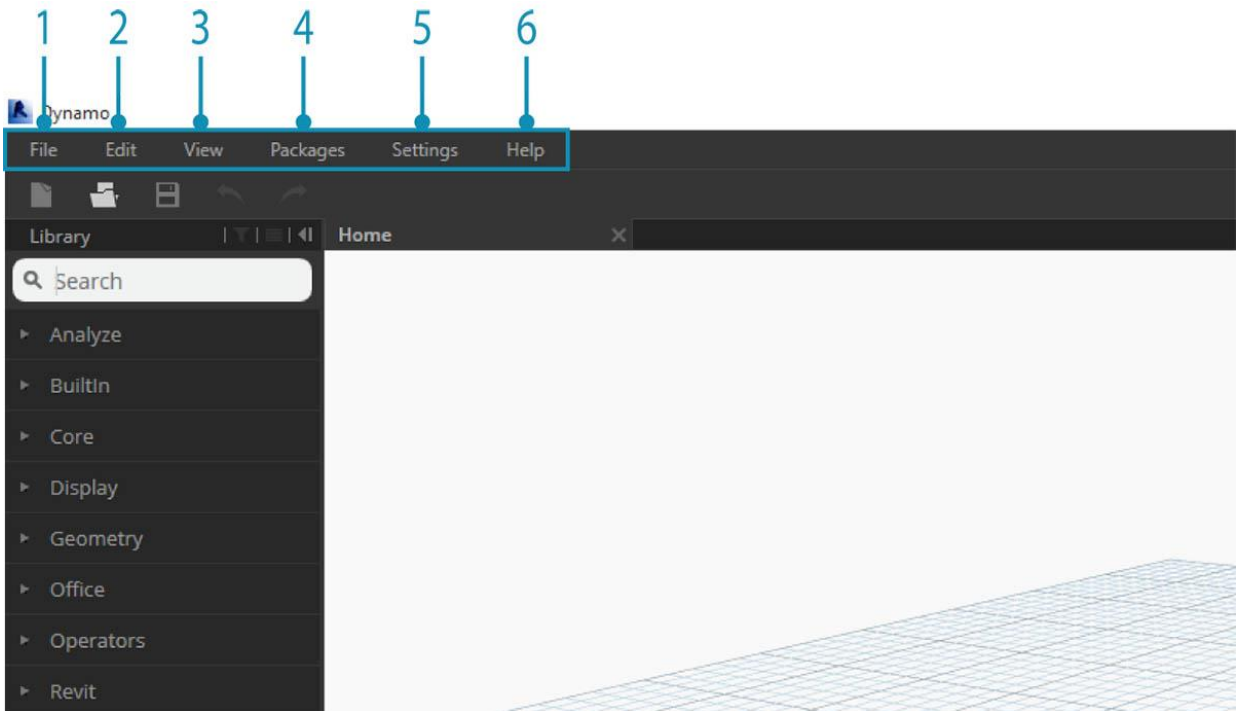


1. Menús
2. Barra de herramientas
3. Biblioteca
4. Espacio de trabajo
5. Barra de ejecución

Profundicemos en la interfaz de usuario y exploremos la funcionalidad de cada región.

Menús

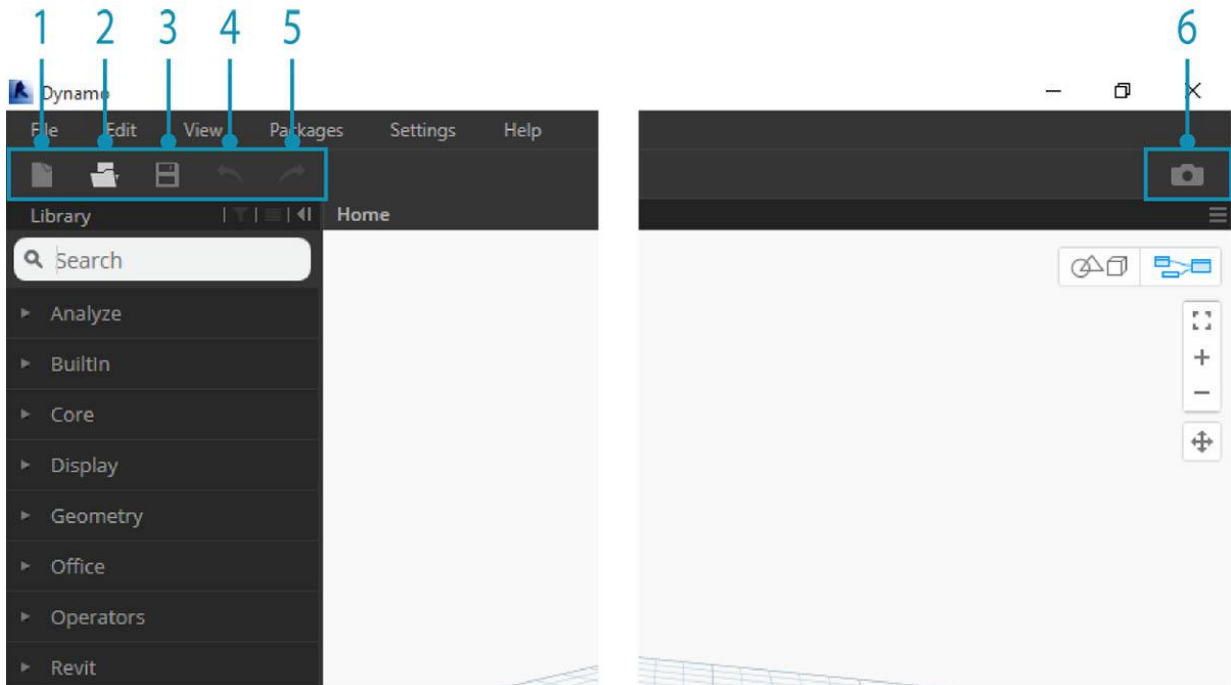
Los menús desplegables son un excelente lugar para encontrar algunas de las funciones básicas de la aplicación Dynamo. Al igual que la mayoría del software de Windows, las acciones relacionadas con la administración de archivos y operaciones para la selección y edición de contenido se encuentran en los primeros dos menús. Los menús restantes son más específicos para Dynamo.



1. Archivo
2. Editar
3. Ver
4. Paquetes
5. Configuraciones
6. Ayuda

Barra de herramientas

La barra de herramientas de Dynamo contiene una serie de botones para acceder rápidamente al trabajo con archivos, así como también comandos para deshacer [Ctrl + Z] y Rehacer [Ctrl + Y]. En el extremo derecho, hay otro botón que exportará una instantánea del espacio de trabajo, que es extremadamente útil para la documentación y el uso compartido.



1. Nuevo: cree un nuevo archivo .dyn
2. Abrir: abra un archivo .dyn (espacio de trabajo) existente o .dyf (nodo personalizado)
3. Guardar / Guardar como: guarde su archivo .dyn o .dyf activo
4. Deshacer: deshace tu última acción
5. Rehacer - Rehacer la siguiente acción
6. Exportar área de trabajo como imagen - Exportar el área de trabajo visible como un archivo PNG

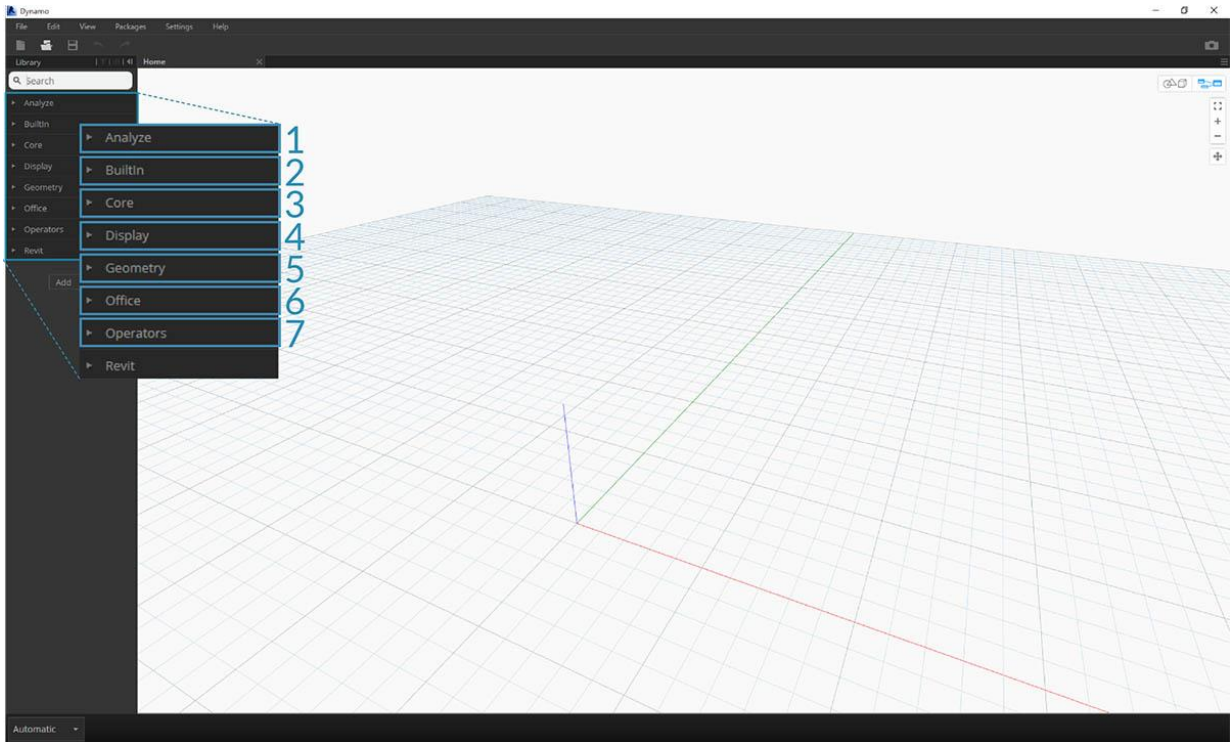
Biblioteca

La Biblioteca contiene todos los Nodos cargados, incluidos los Nodos predeterminados que vienen con la instalación, así como cualquier Nodos o Paquetes personalizados cargados adicionalmente. Los Nodos en la Biblioteca están organizados jerárquicamente dentro de bibliotecas, categorías y, cuando corresponda, subcategorías según si los Nodos **crean** datos, ejecutan una **Acción** o **consulta de** datos.

Hojeada

Por defecto, la **Biblioteca** contendrá ocho categorías de Nodos. **Core** y **Geometry** son excelentes menús para comenzar a explorar ya que contienen la mayor cantidad de Nodos. Navegar a través de estas categorías es la manera más rápida de comprender la jerarquía de lo que podemos agregar a nuestro Espacio de trabajo y la mejor manera de descubrir nuevos Nodos que no haya utilizado antes.

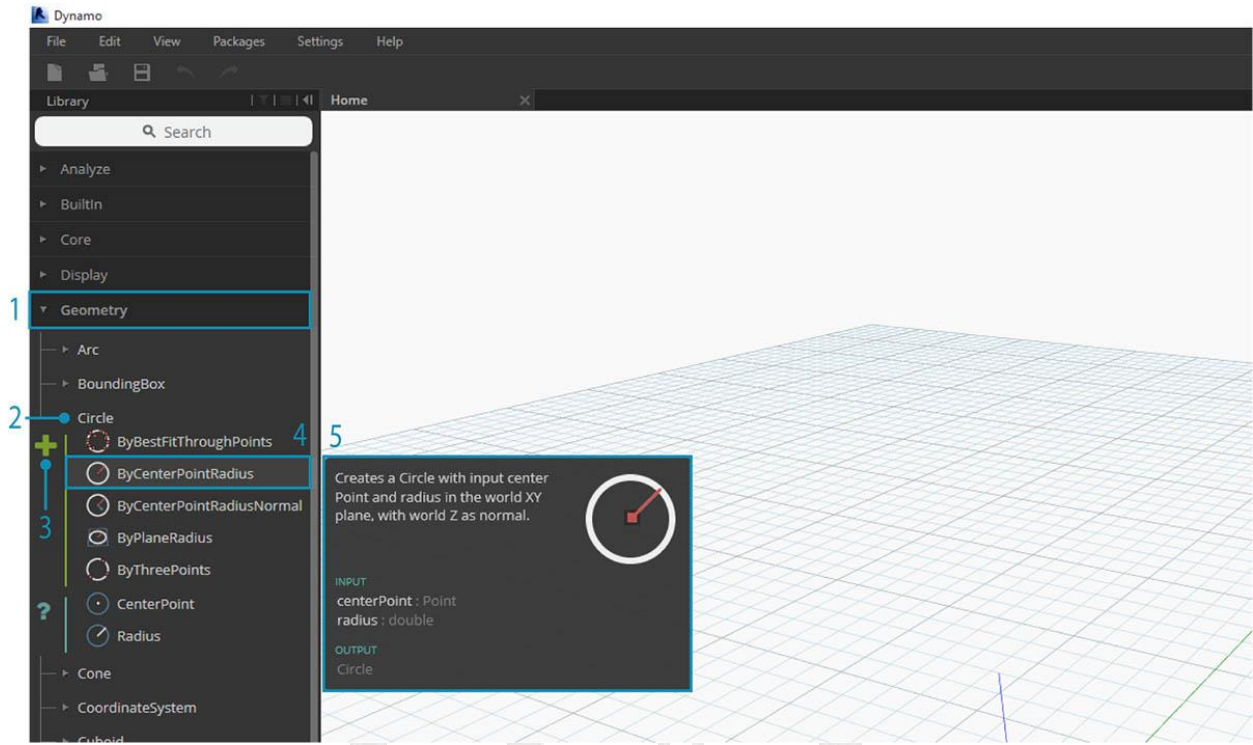
Ahora nos centraremos en la colección predeterminada de Nodos, pero tenga en cuenta que ampliaremos esta Biblioteca con Nodos personalizados, bibliotecas adicionales y el Administrador de paquetes más adelante.



1. Analizar
2. Funciones integradas
3. Núcleo
4. Geometría
5. Migración
6. Oficina
7. Operadores

Explore la Biblioteca haciendo clic en los menús. Haga clic en Geometría> Círculo. Tenga en cuenta la nueva porción del menú que se revela y específicamente las etiquetas **Crear** y **consultar**.

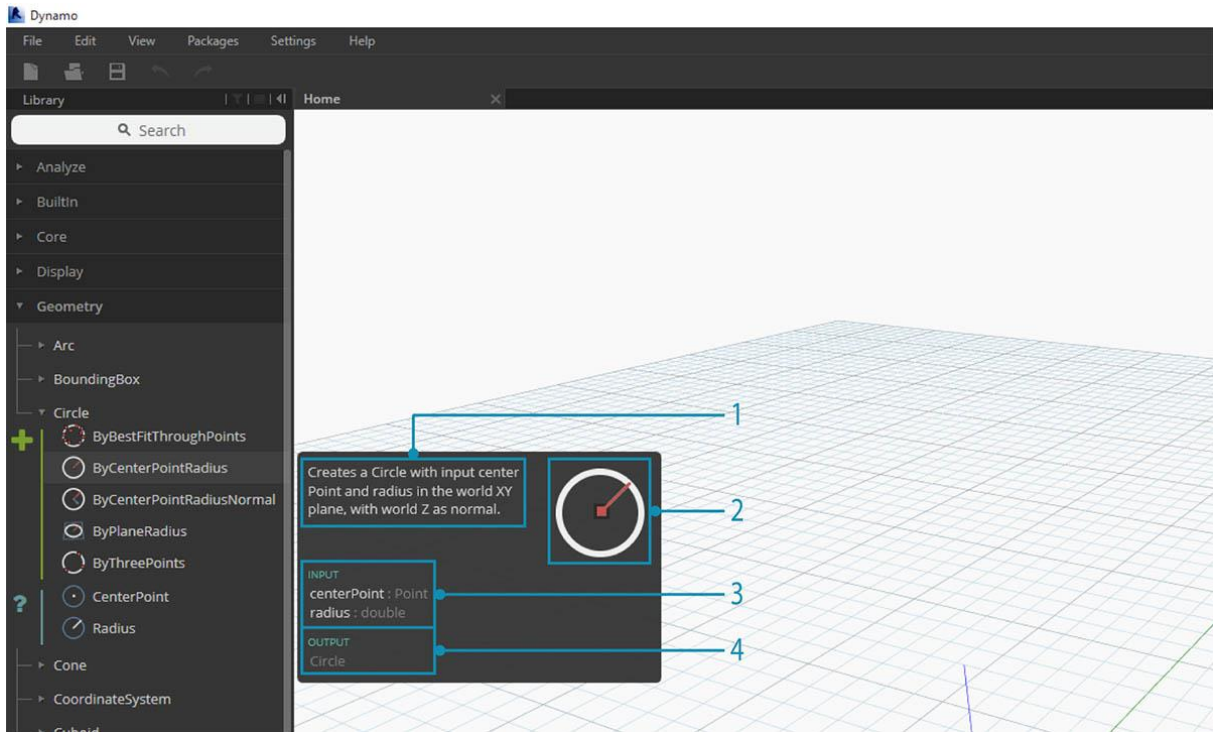
DARCO
DESDE 1988



1. Biblioteca
2. Categoría
3. Subcategoría: Crear / Acciones / Query
4. Nodo
5. Descripción y propiedades del nodo: aparece cuando se pasa el puntero sobre el icono del nodo.

Desde el mismo menú de Círculo, pase el mouse sobre **ByCenterPointRadius**. La ventana revela información más detallada sobre el Nodo más allá de su nombre e ícono. Esto nos ofrece una forma rápida de entender qué hace el Nodo, qué requerirá para las entradas y qué ofrecerá como resultado.

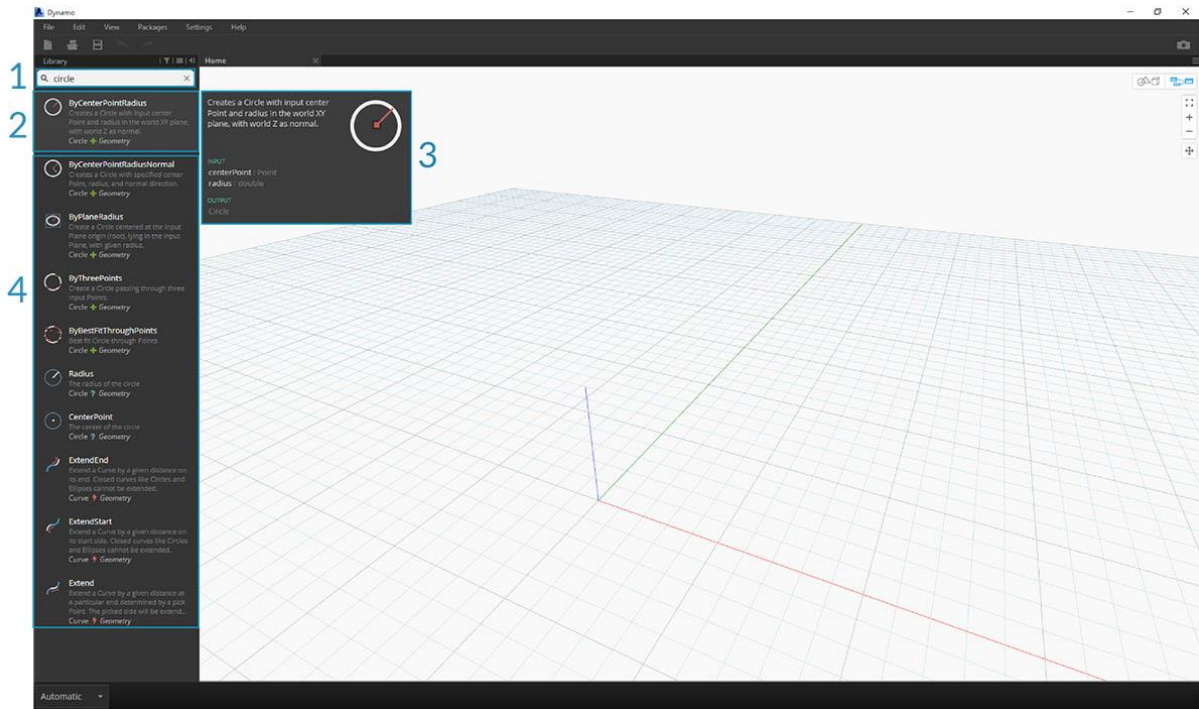
DARCO
DESDE 1988



1. Descripción - descripción en lenguaje sencillo del nodo
2. Icono - versión más grande del icono en el menú de la biblioteca
3. Entrada (s) - nombre, tipo de datos y estructura de datos
4. Salida (es) - tipo de datos y estructura

Buscando

Si sabe con relativa especificidad qué nodo desea agregar a su área de trabajo, el campo de **búsqueda** es su mejor amigo. Cuando no está editando configuraciones o especificando valores en el área de trabajo, el cursor siempre está presente en este campo. Si comienza a escribir, la Biblioteca de Dynamo revelará una coincidencia de mejor ajuste seleccionado (con rutas de exploración donde se puede encontrar en las categorías de Nodo) y una lista de coincidencias alternativas para la búsqueda. Cuando presiona Entrar, o hace clic en el elemento en el navegador truncado, el Nodo resaltado se agrega al centro del Espacio de trabajo.

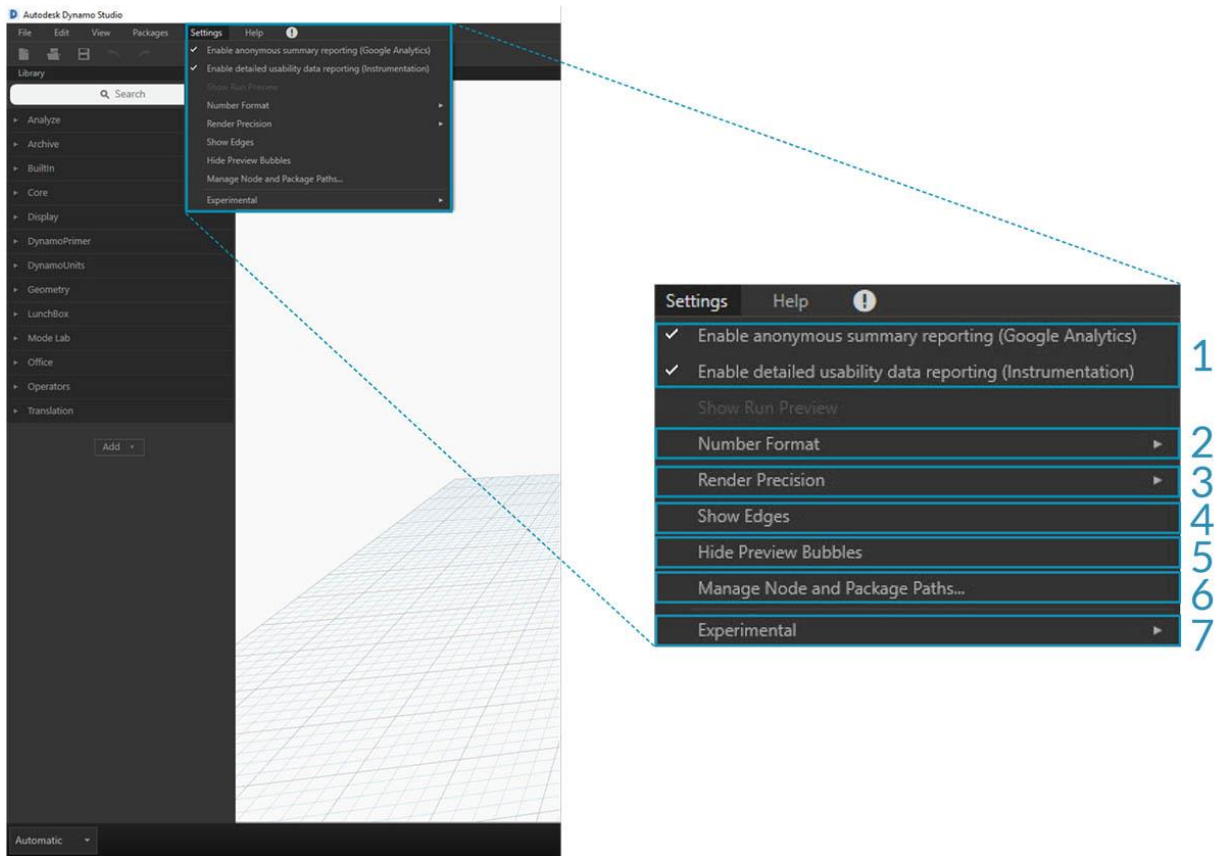


1. Campo de búsqueda
2. Resultado de mejor ajuste / seleccionado
3. Partidos alternativos

Configuraciones

Desde la configuración geométrica hasta la del usuario, estas opciones se pueden encontrar en el menú **Configuración**. Aquí puede activar o desactivar el uso compartido de sus datos de usuario para mejorar Dynamo, así como definir la precisión del punto decimal de la aplicación y la calidad de la representación geométrica.

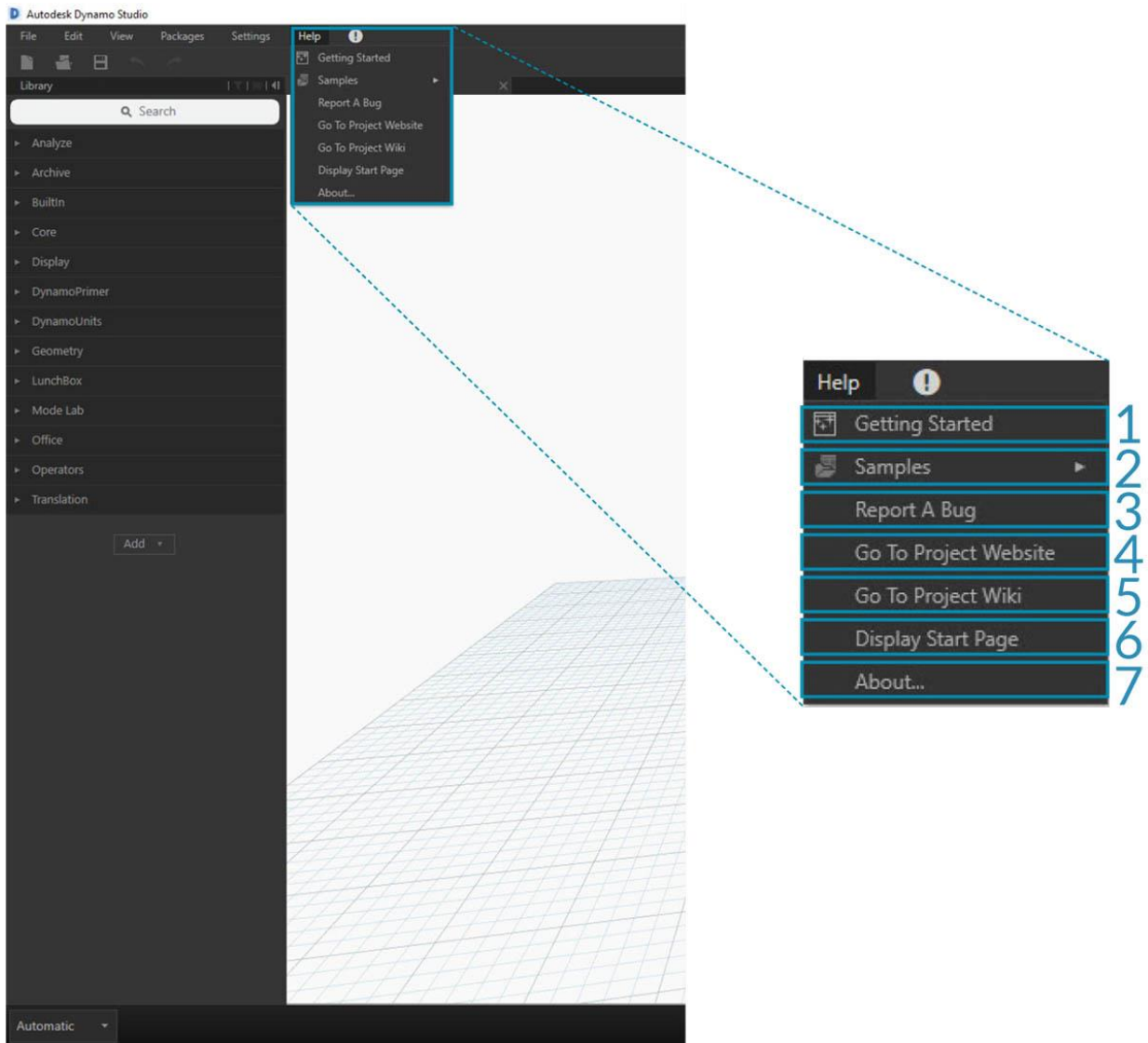
Nota: Recuerde que las unidades de Dynamo son genéricas.



1. Habilitación de informes: opciones para compartir datos de usuarios para mejorar Dynamo.
2. Opciones de formato numérico: cambie la configuración del documento para decimales.
3. Renderizar precisión: aumenta o reduce la calidad del procesamiento de documentos.
4. Mostrar / ocultar bordes de geometría - Conmuta bordes de geometría 3D.
5. Mostrar / Ocultar burbujas de vista previa - Alternar burbujas de vista previa de datos debajo de los nodos.
6. Administre rutas de paquetes y paquetes: gestione las rutas de archivos para hacer que los nodos y paquetes aparezcan en la biblioteca.
7. Habilitación de características experimentales: use funciones beta nuevas en Dynamo.

Ayuda

Si estás atascado, mira el Menú de **Ayuda**. Aquí puede encontrar los archivos de muestra que vienen con su instalación, así como acceder a uno de los sitios web de referencia de Dynamo a través de su navegador de Internet. Si es necesario, verifique la versión de Dynamo instalada y si está actualizada a través de la opción **Acerca de**.

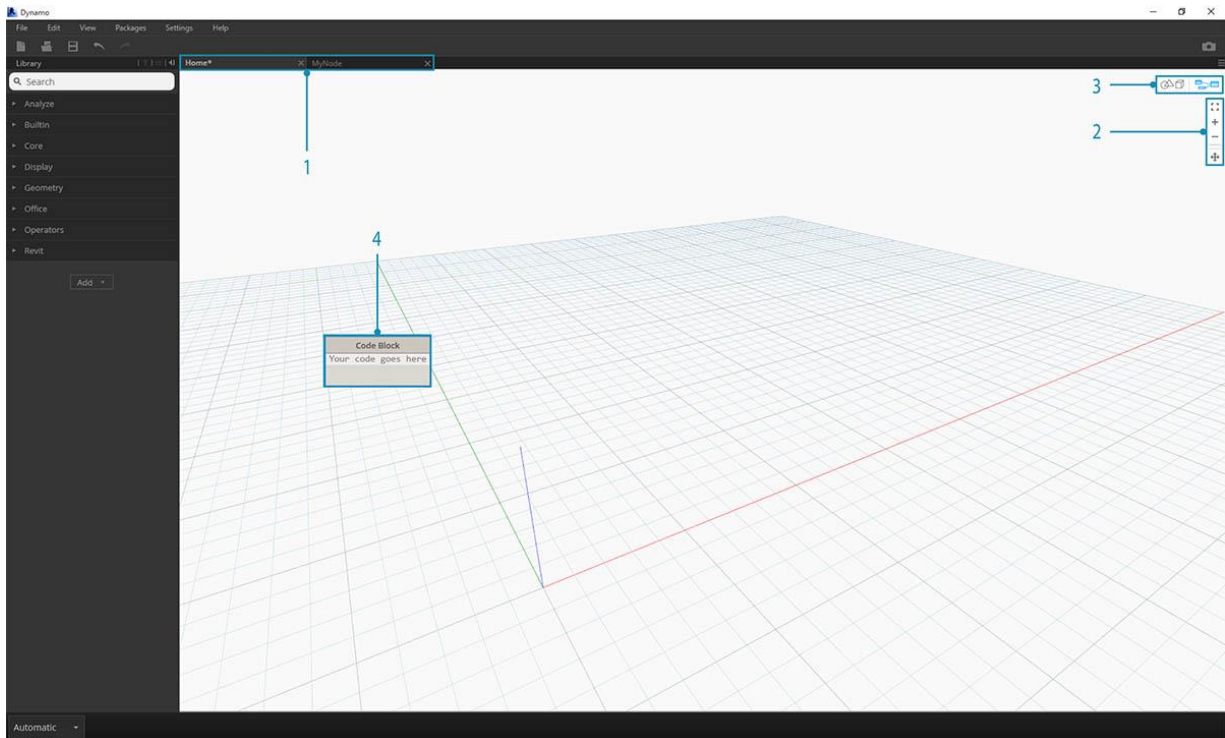


1. Primeros pasos: una breve introducción al uso de Dynamo.
2. Muestras - Archivos de ejemplo de referencia.
3. Informar un error: abra un problema en GitHub.
4. Ir al sitio web del proyecto: vea el Proyecto Dynamo en GitHub.
5. Ir al proyecto Wiki: visita la wiki para aprender sobre desarrollo usando la API de Dynamo, que admite bibliotecas y herramientas.
6. Mostrar página de inicio: vuelve a la página de inicio de Dynamo cuando está dentro de un documento.
7. Acerca de - Datos de la versión Dynamo.

El espacio de trabajo

El área de **trabajo de** Dynamo es donde desarrollamos nuestros programas visuales, pero también es donde hacemos una vista previa de cualquier geometría resultante. Ya sea que trabajemos en un Home Workspace o un Custom Node, podemos navegar con nuestro mouse o los botones en la parte superior derecha. Alternar entre los modos en la parte inferior derecha de los interruptores cuya vista previa navegamos.

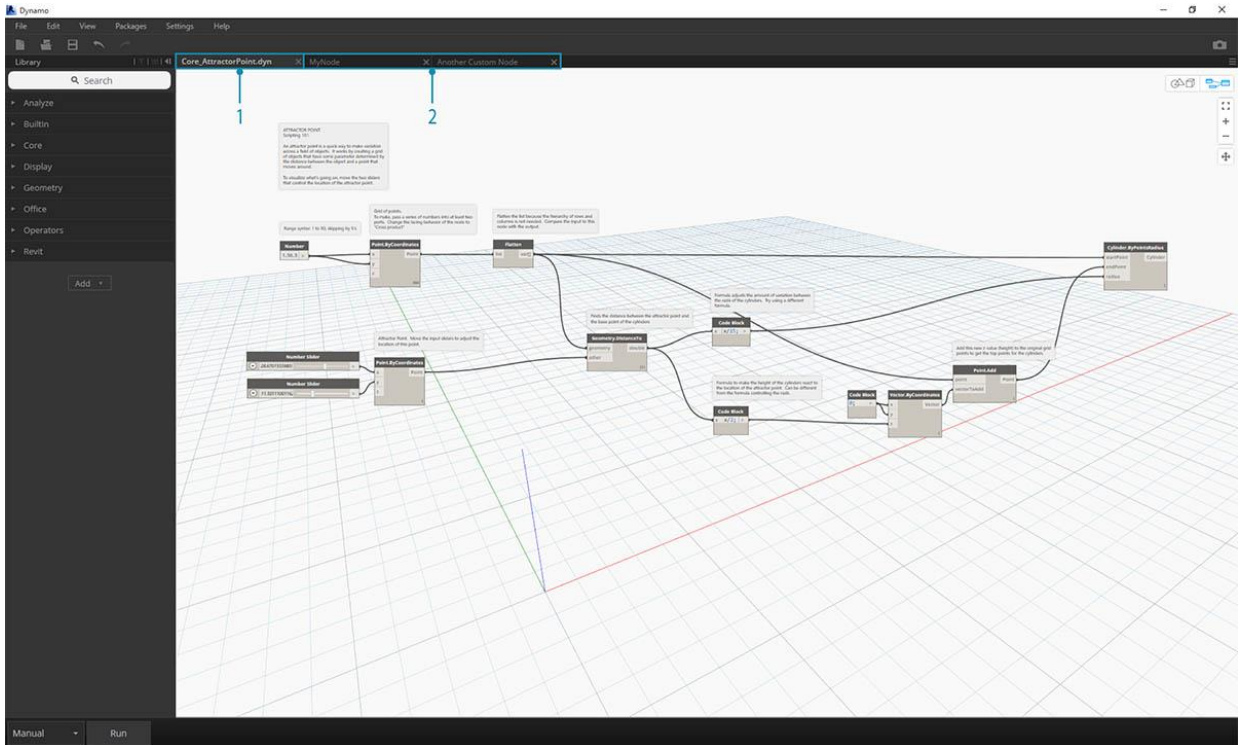
Nota: Los nodos y la geometría tienen un orden de extracción, por lo que puede tener objetos representados uno sobre el otro. Esto puede ser confuso cuando se agregan varios nodos en secuencia, ya que pueden representarse en la misma posición en el espacio de trabajo.



1. Pestañas
2. Botones Zoom / Pan
3. Modo de vista previa
4. Hacer doble clic en el espacio de trabajo

Pestañas

La pestaña del Área de trabajo activa le permite navegar y editar su programa. Cuando abre un nuevo archivo, de forma predeterminada está abriendo un nuevo espacio de trabajo **doméstico**. También puede abrir un nuevo espacio de trabajo de **nodo personalizado** desde el menú Archivo o mediante la opción *Nuevo nodo por selección* haciendo clic derecho cuando se seleccionan nodos (más sobre esta funcionalidad más adelante).

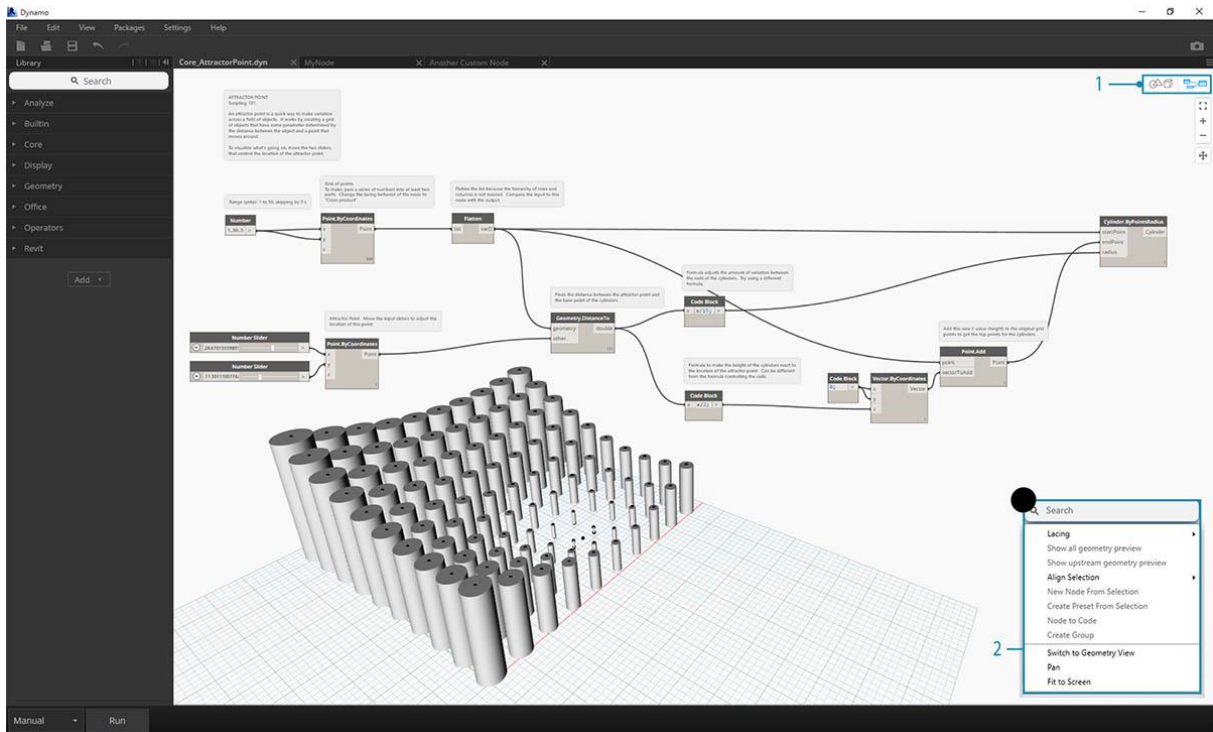


Nota: Puede tener solo un Home Workspace abierto a la vez; sin embargo, puede tener múltiples espacios de trabajo de nodo personalizado abiertos en pestañas adicionales.

Gráfico versus navegación de vista previa 3D

En Dynamo, el gráfico y los resultados 3D del gráfico (si estamos creando geometría) se representan en el espacio de trabajo. De forma predeterminada, el gráfico es la vista previa activa, por lo que, si utilizamos los botones de navegación o el botón central del mouse para desplazarnos y acercarnos, nos moverá a través del gráfico. Alternar entre vistas previas activas se puede lograr de tres maneras:



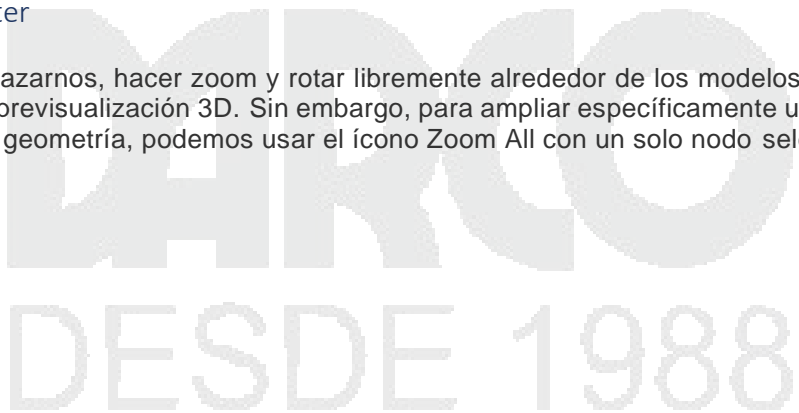


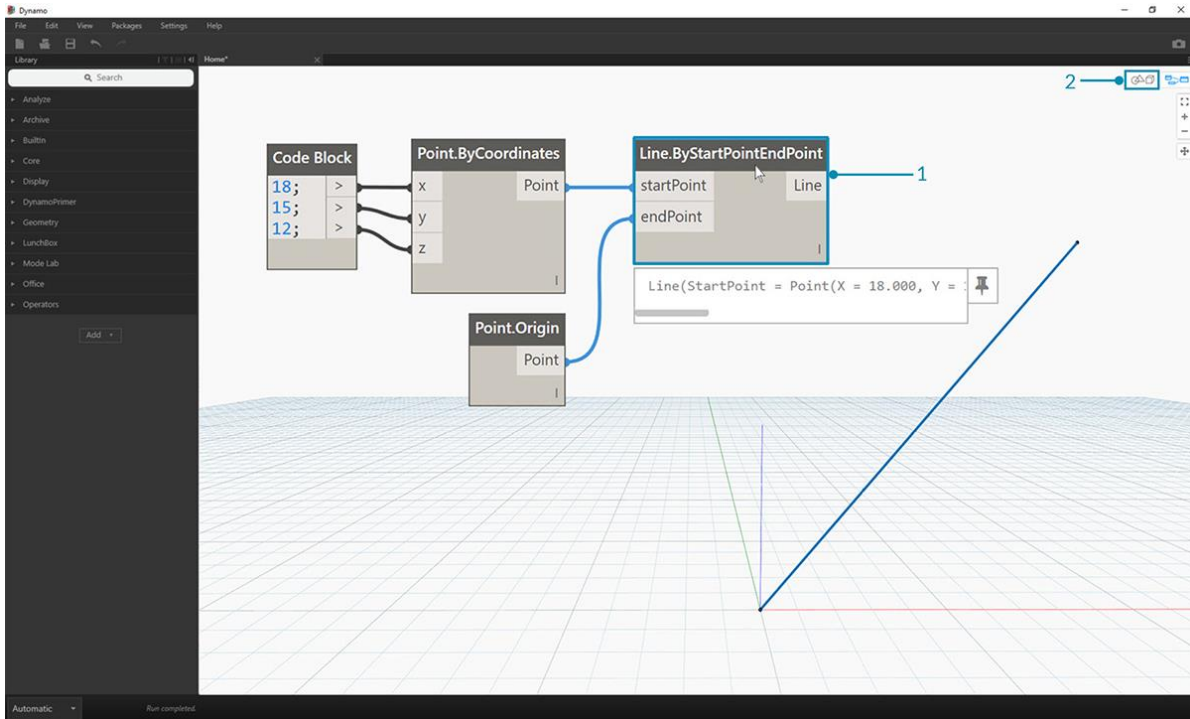
1. Vista previa Botones de alternancia en el espacio de trabajo
2. Al hacer clic con el botón derecho en el área de trabajo y seleccionar *Cambiar a ... Ver*
3. Método abreviado de teclado (Ctrl + B)

El modo de navegación de vista previa en 3D también nos da la capacidad de **manipulación directa** de puntos, ejemplificada en la sección Primeros pasos.

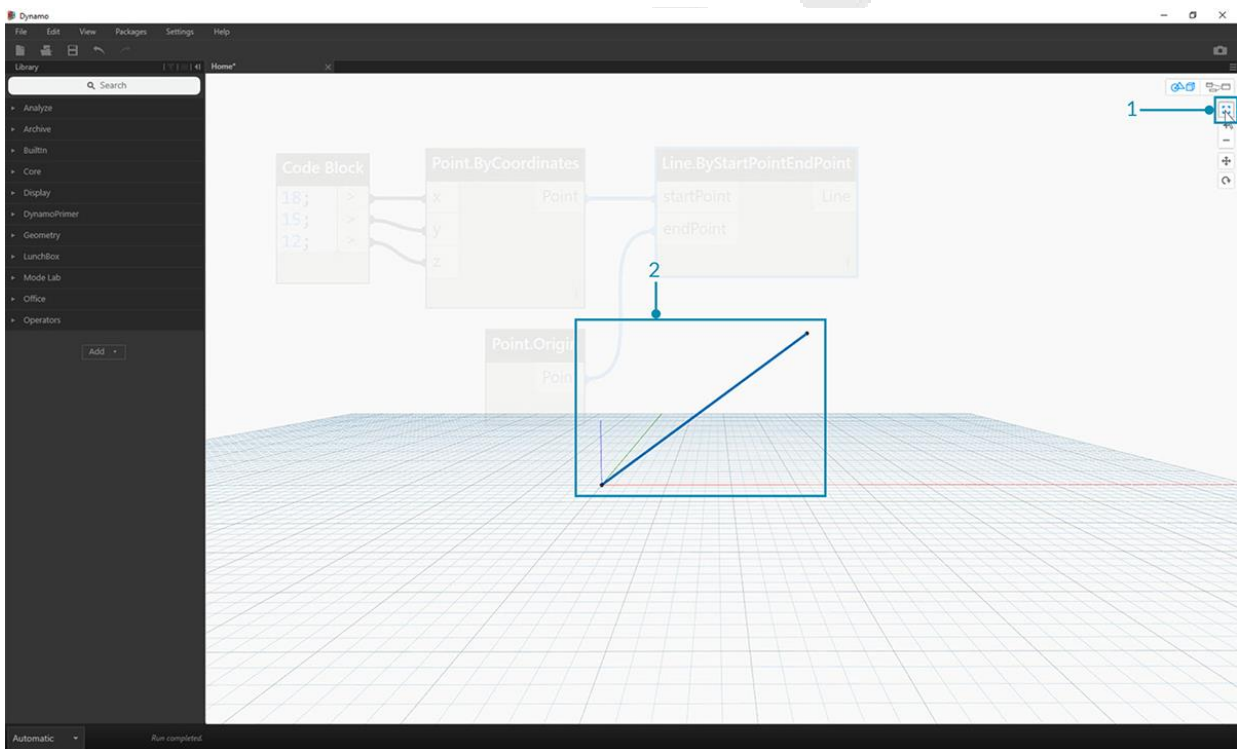
Zoom a Recenter

Podemos desplazarnos, hacer zoom y rotar libremente alrededor de los modelos en el modo de navegación de previsualización 3D. Sin embargo, para ampliar específicamente un objeto creado por un nodo de geometría, podemos usar el ícono Zoom All con un solo nodo seleccionado.





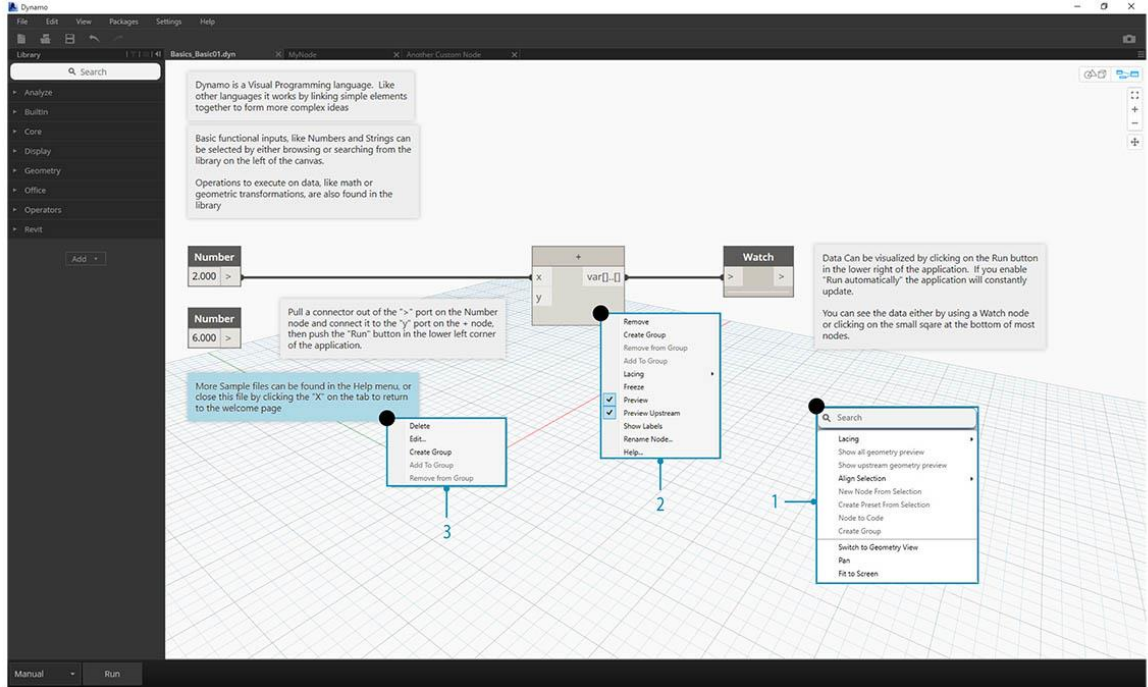
1. Seleccione el nodo correspondiente a la geometría que centrará la vista.
2. Cambia a la navegación de vista previa en 3D.



1. Haga clic en el ícono Zoom All en la parte superior derecha.
2. La geometría seleccionada se centrará dentro de la vista.

¡Hola, ratón!

Según el modo de vista previa que esté activo, los botones del mouse actuarán de manera diferente. En general, el clic izquierdo del mouse selecciona y especifica las entradas, el clic derecho del mouse da acceso a las opciones, y el clic del medio del mouse le permite navegar en el Espacio de trabajo. El clic derecho del mouse nos presentará opciones basadas en el contexto de donde estamos haciendo clic.



1. Haga clic derecho en el espacio de trabajo.
2. Haga clic derecho en un nodo.
3. Haga clic derecho en una nota.

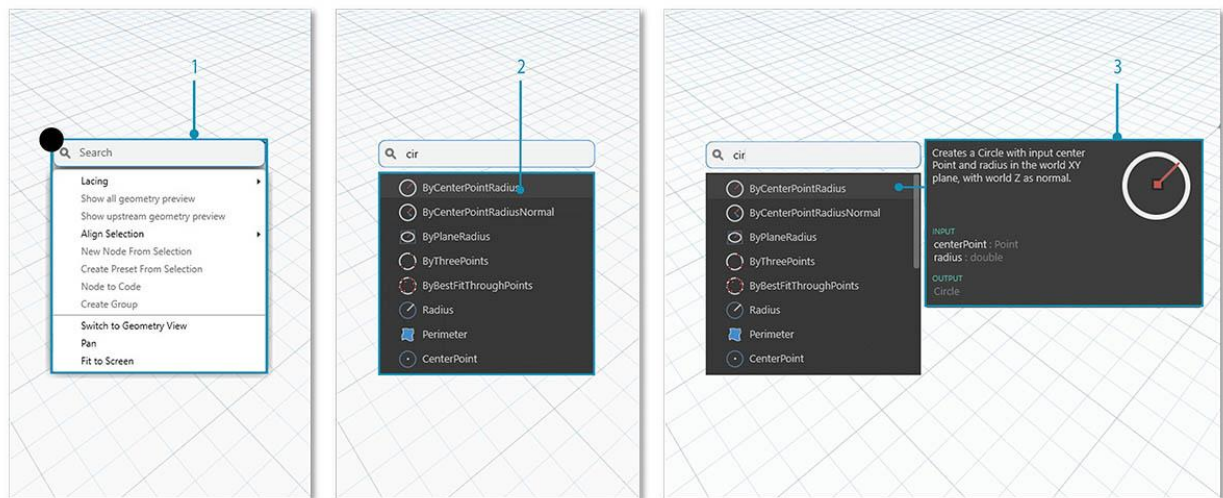
Aquí hay una tabla de las interacciones del mouse por Vista previa:

Acción del mouse	Vista previa del gráfico	Vista previa 3D
Click izquierdo	Seleccionar	N / A
Botón derecho del ratón	Menú de contexto	Opciones de Zoom
Medio clic	Pan	Pan
Voluta	Acercar / alejar	Acercar / alejar

Acción del mouse	Vista previa del gráfico	Vista previa 3D
Haga doble clic	Crear bloque de código	N / A

Búsqueda en lienzo

El uso de la "Búsqueda en Lienzo" agregará una gran velocidad a su flujo de trabajo de Dynamo al proporcionarle acceso a descripciones de nodos y sugerencias de herramientas sin alejarse de su lugar en el gráfico. Con solo hacer clic derecho, puede acceder a toda la funcionalidad útil de la "Búsqueda de la Biblioteca" desde donde quiera que trabaje en el lienzo.

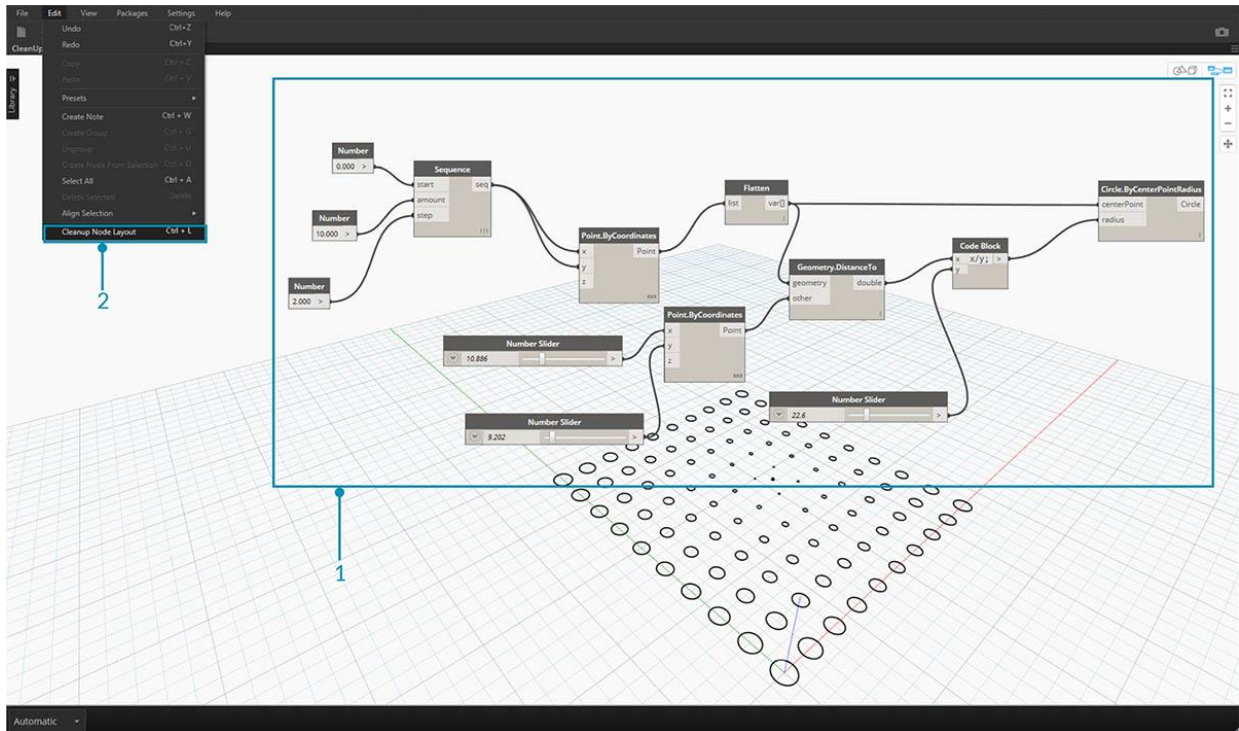


1. Haga clic derecho en cualquier parte del lienzo para que aparezca la función de búsqueda. Mientras la barra de búsqueda está vacía, el menú desplegable será un menú de vista previa.
2. A medida que escribe en la barra de búsqueda, el menú desplegable se actualizará continuamente para mostrar los resultados de búsqueda más relevantes.
3. Desplácese sobre los resultados de búsqueda para mostrar sus descripciones correspondientes y sugerencias de herramientas.

Limpiar el diseño del nodo

Mantener su lienzo de Dynamo organizado se vuelve cada vez más importante a medida que sus archivos crecen en complejidad. Aunque tenemos la herramienta **Align Selection** para trabajar con pequeñas cantidades de Nodos seleccionados, Dynamo también cuenta con la herramienta de **diseño Clean Node Layout** para ayudar con la limpieza general del archivo.

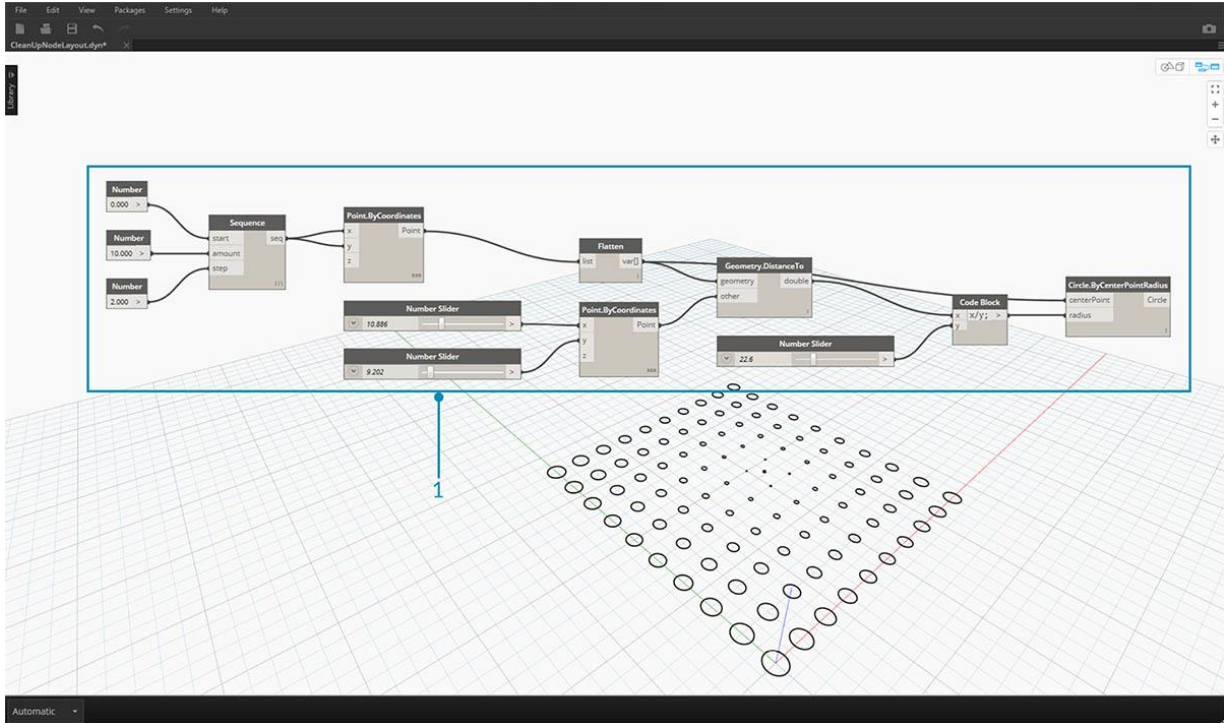
Antes de la limpieza del nodo



1. Seleccione los Nodos que se organizarán automáticamente o deje todo desmarcado para limpiar todos los nodos en el archivo.
2. La función Diseño de nodo de limpieza se encuentra en la pestaña Editar.

DARCO
DESDE 1988

Después de la limpieza del nodo



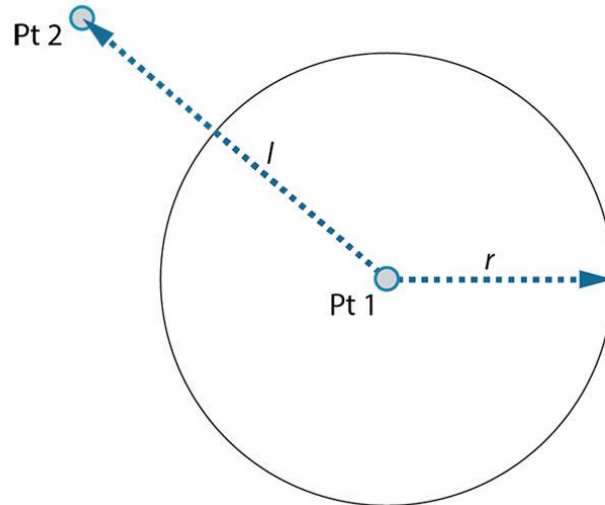
3. Los nodos se redistribuirán y alinearán automáticamente, limpiando los nodos escalonados o superpuestos y alineándolos con los nodos vecinos.

Empezando

Ahora que nos hemos familiarizado con el diseño de la interfaz y navegando por el espacio de trabajo, nuestro siguiente paso es comprender el flujo de trabajo típico para desarrollar un gráfico en Dynamo. Comencemos creando un círculo de tamaño dinámico y luego creamos una serie de círculos con radios variables.

Definición de objetivos y relaciones

Antes de agregar algo al espacio de trabajo de Dynamo, es clave que tengamos una comprensión sólida de lo que estamos tratando de lograr y cuáles serán las relaciones significativas. Recuerde que cada vez que conectemos dos Nodos, estamos creando un vínculo explícito entre ellos; podemos cambiar el flujo de datos más adelante, pero una vez conectados, nos hemos comprometido con esa relación. En este ejercicio, queremos crear un círculo (*Objetivo*) donde la entrada del radio se define por una distancia a un punto cercano (*Relación*).



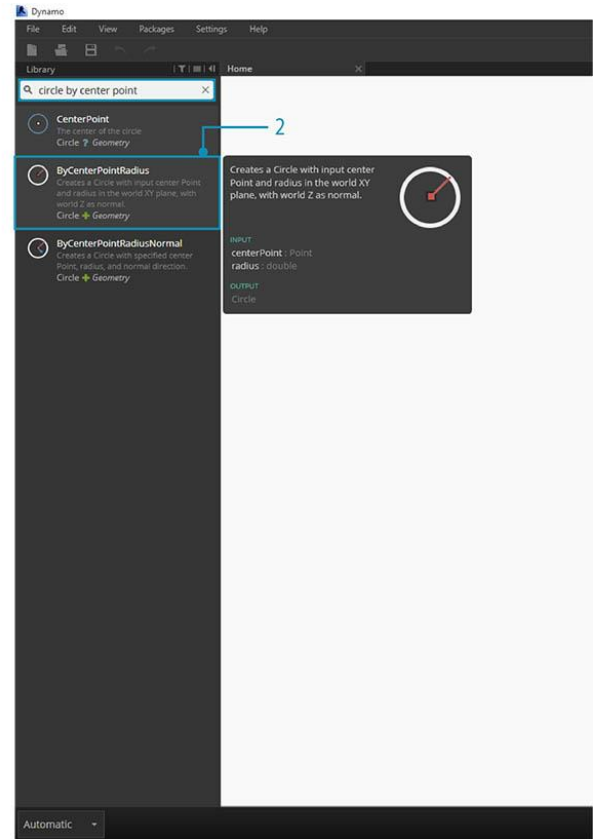
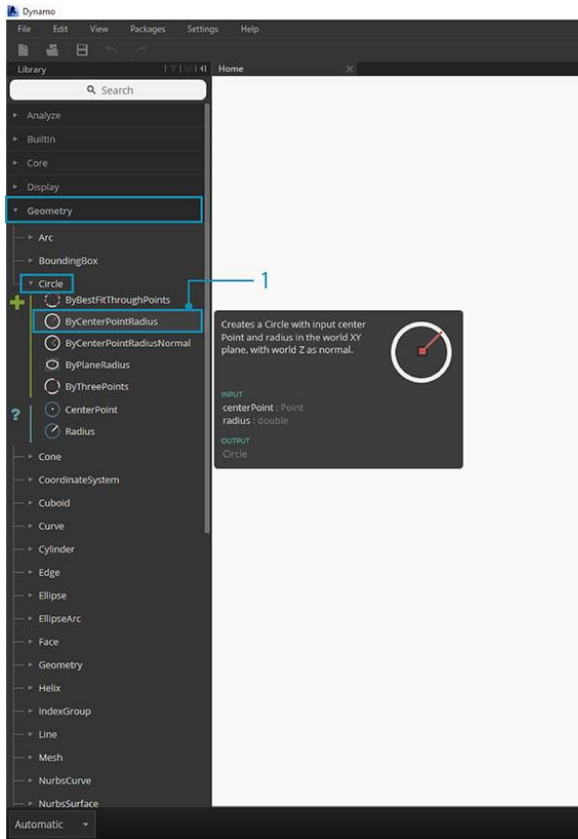
Length (l) \propto Radius (r)

Un punto que define una relación basada en la distancia se conoce comúnmente como "Atractor". Aquí la distancia a nuestro punto de atracción se usará para especificar qué tan grande debe ser nuestro círculo.

Agregar nodos al espacio de trabajo

Ahora que hemos bosquejado nuestros Objetivos y relaciones, podemos comenzar a crear nuestro gráfico. Necesitamos los Nodos que representarán la secuencia de acciones que Dynamo ejecutará. Como sabemos que estamos tratando de crear un círculo, comencemos por ubicar un Nodo que lo haga. Usando el campo de búsqueda o navegando por la biblioteca, descubriremos que hay más de una forma de crear un círculo.

DARCO
DESDE 1988

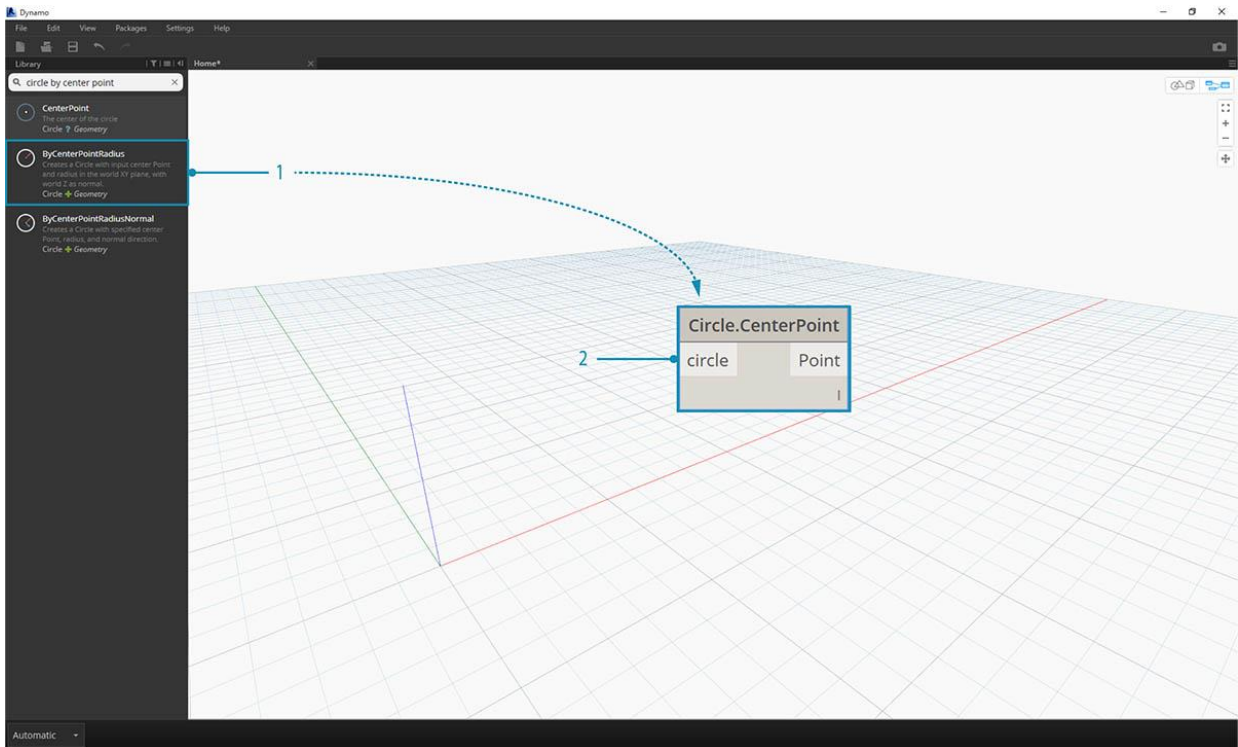


1. Busque Geometría> Círculo> **Circle.ByPointRadius**
2. Buscar> "Círculo por punto ..."

Vamos a añadir el **Circle.ByPointRadius** Nodo al área de trabajo haciendo clic en él en la Biblioteca - esto debe añadir el nodo al centro del espacio de trabajo.

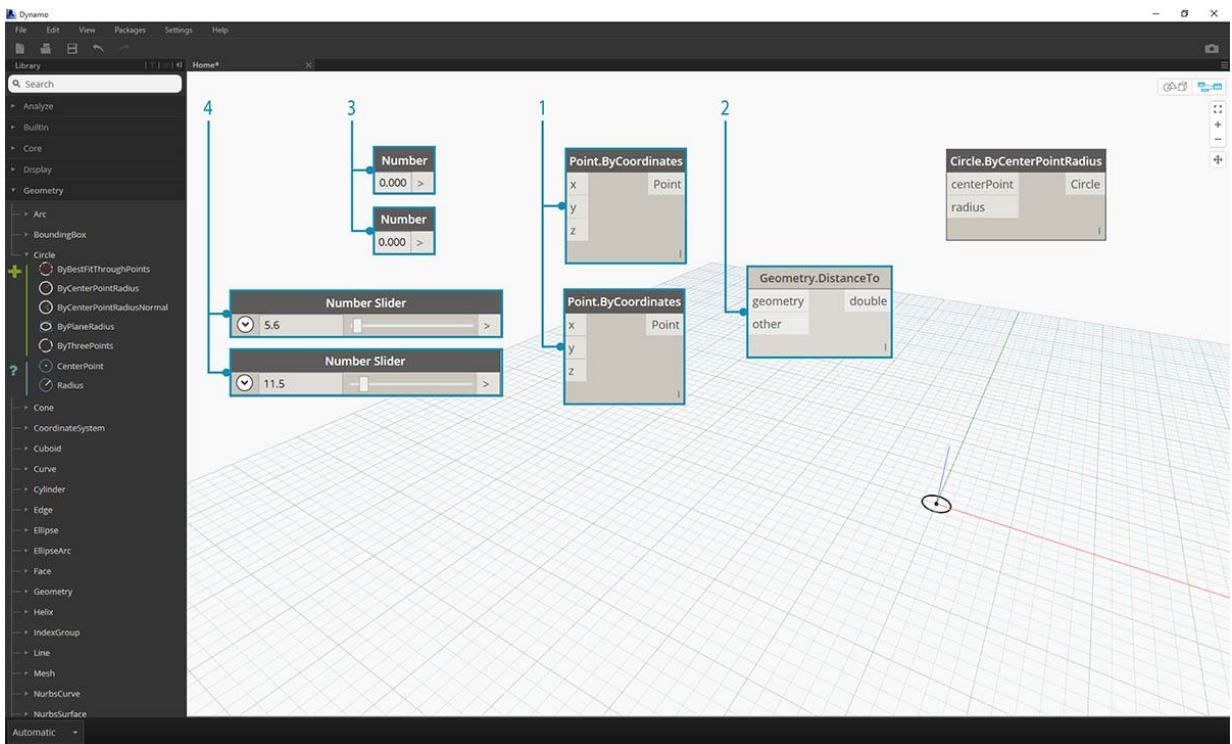
DARCO

DESDE 1988



1. The Circle.ByPointandRadius Node en la biblioteca
2. Al hacer clic en el Nodo en la Biblioteca, se agrega al Espacio de trabajo

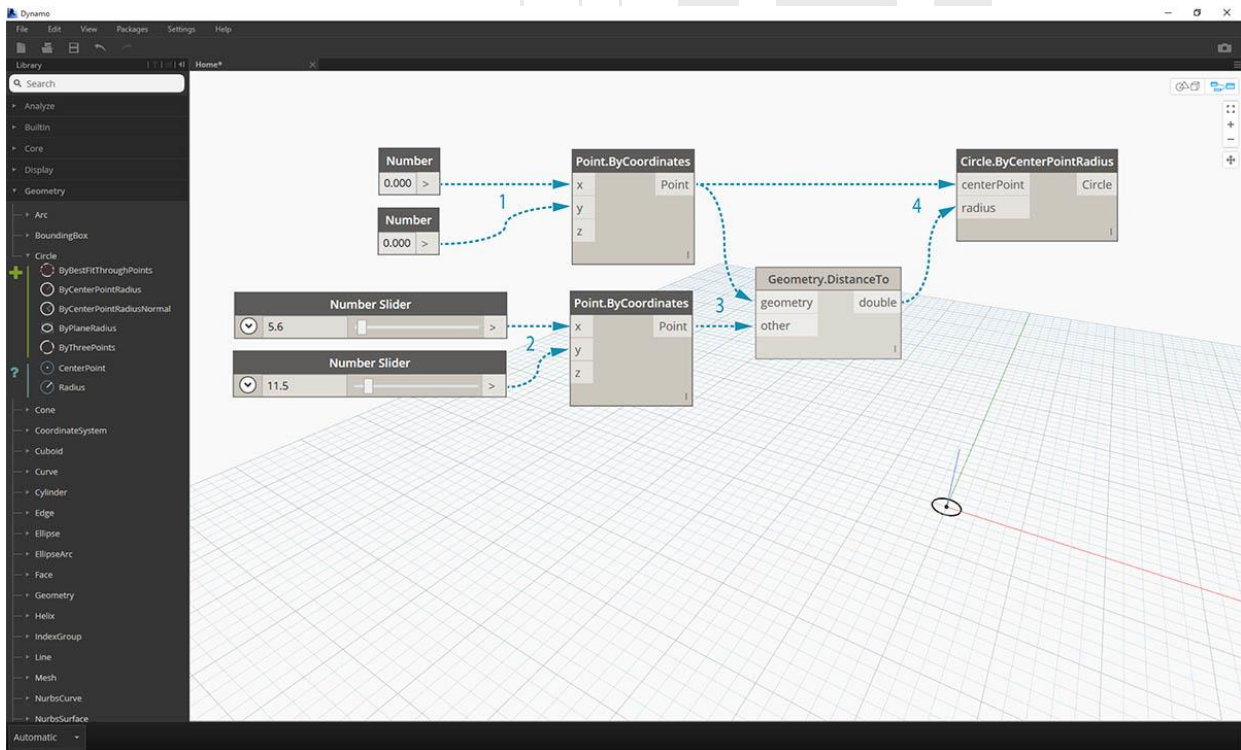
También necesitaremos **Point.ByCoordinates**, **Number Input** y **Number Slider** Nodes.



1. Geometry> Point> **Point.ByCoordinates**
2. Geometría> Geometría> **DistanciaTo**
3. Núcleo> Entrada> **Número**
4. Core> Input> **Number Slider**

Conexión de nodos con cables

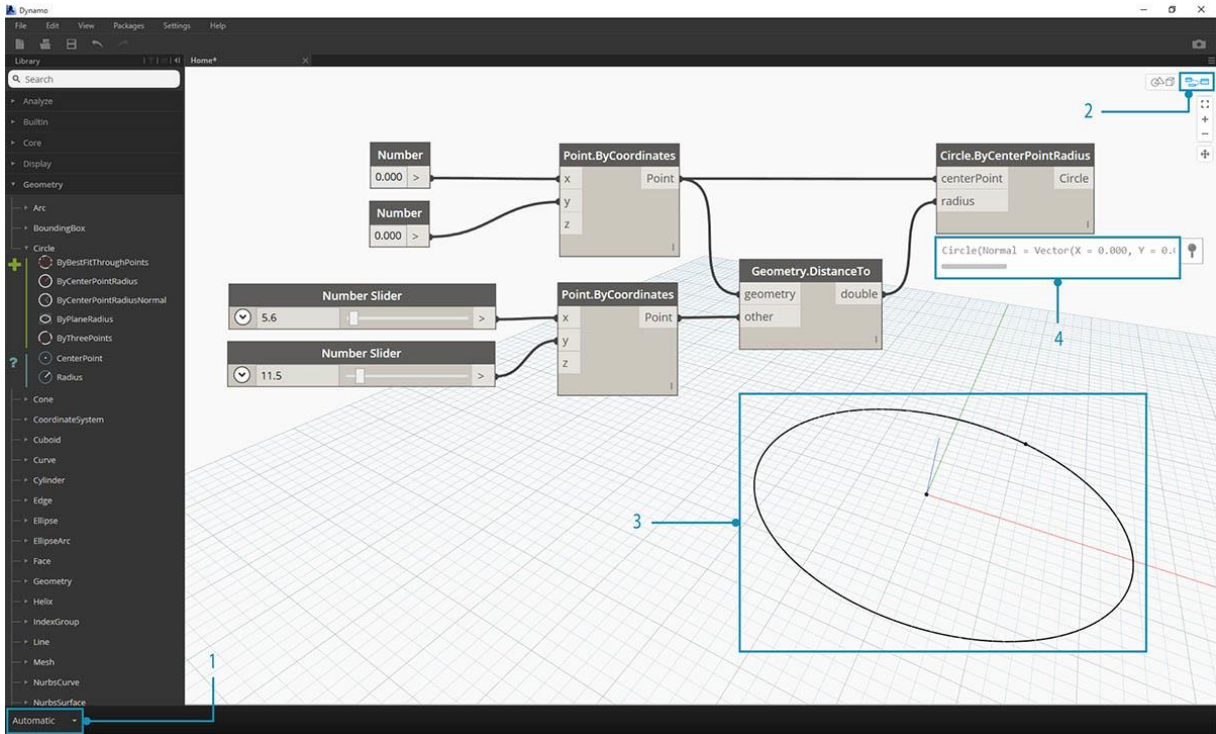
Ahora que tenemos algunos Nodos, necesitamos conectar los Puertos de los Nodos con Cables. Estas conexiones definirán el flujo de datos.



1. Número **a punto**. ByCoordinates
2. Número de controles deslizantes **a punto**. ByCoordinates
3. Point.ByCoordinates (**2**) a DistanceTo
4. Point.ByCoordinates y DistanceTo **a** Circle.ByCenterPointRadius

Ejecutando el programa

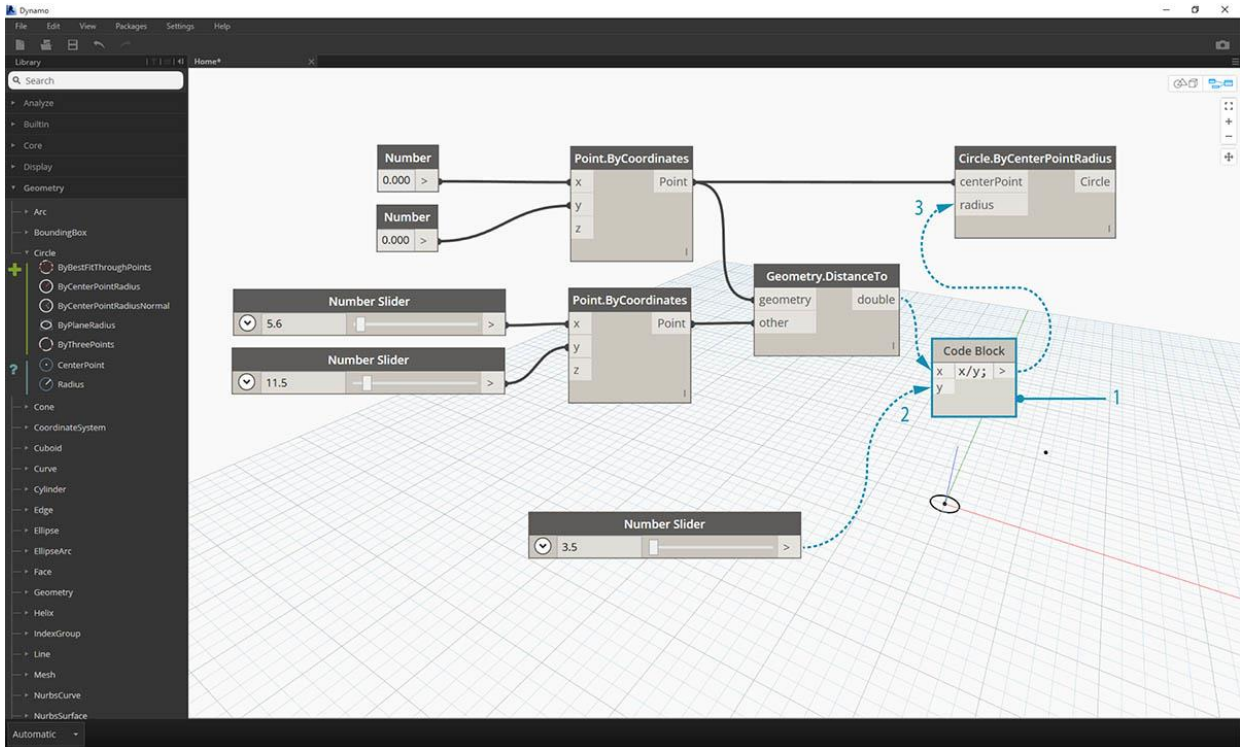
Con nuestro flujo de programa definido, todo lo que tenemos que hacer es decirle a Dynamo que lo ejecute. Una vez que se ejecuta nuestro programa (ya sea automáticamente o cuando hagamos clic en Ejecutar en modo manual), los datos pasarán por los cables, y deberíamos ver los resultados en la vista previa en 3D.



1. (Haga clic en Ejecutar): si la barra de ejecución está en modo manual, debemos hacer clic en Ejecutar para ejecutar el gráfico
2. Vista previa del nodo: al pasar el mouse sobre el recuadro en la esquina inferior derecha de un nodo, aparecerá una ventana emergente con los resultados.
3. Vista previa 3D: si alguno de nuestros nodos crea geometría, lo veremos en la vista previa 3D.

Agregar detalles

Si nuestro programa está funcionando, deberíamos ver un círculo en la vista previa en 3D que está pasando por nuestro punto de Atracción. Esto es genial, pero es posible que deseemos agregar más detalles o más controles. Ajustamos la entrada al Nodo del círculo para que podamos calibrar la influencia en el radio. Agregue otro **control deslizante numérico** al espacio de trabajo, luego haga doble clic en un área en blanco del espacio de trabajo para agregar un nodo de **bloque de código**. Edite el campo en el Bloque de Código, especificando X/Y.

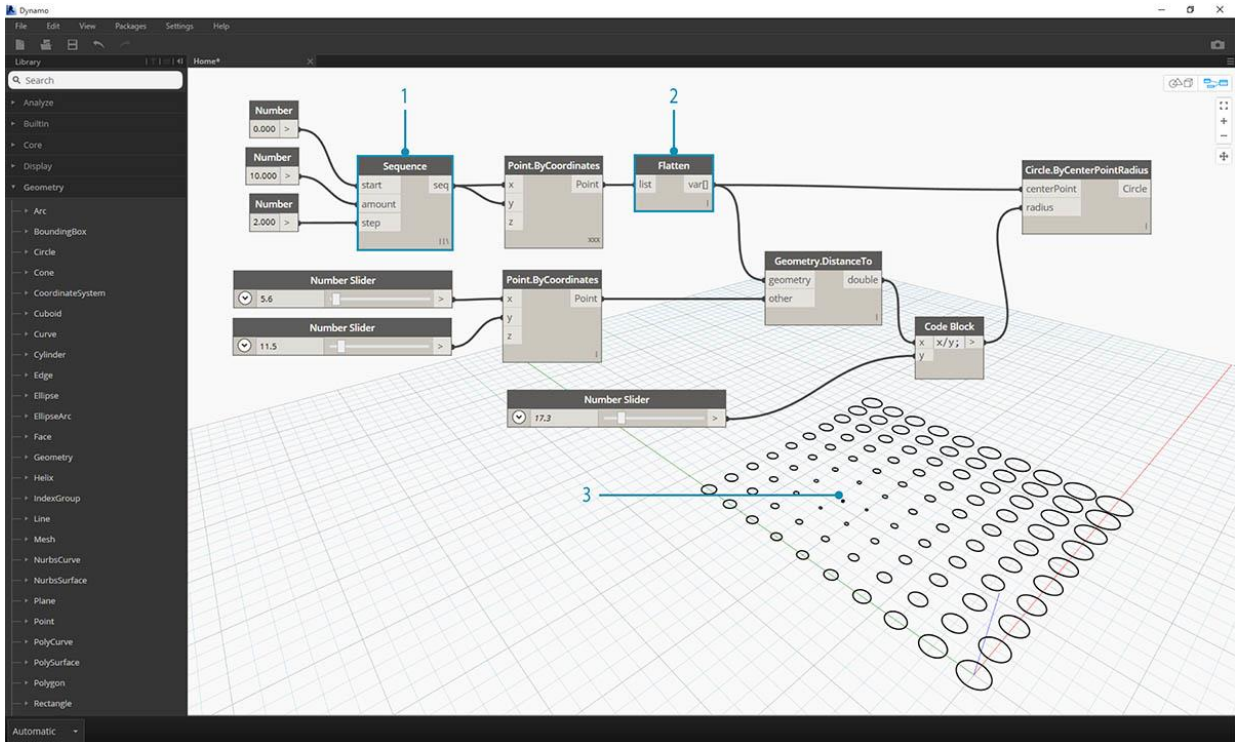


1. **Bloque de código**
2. **Control deslizante Distancia y número al bloque de código**
3. **Bloquear código a Circle.ByCenterPointRadius**

Agregando complejidad

Comenzar de manera simple y construir complejidad es una forma efectiva de desarrollar incrementalmente nuestro programa. Una vez que esté funcionando para un círculo, apliquemos el poder del programa a más de un círculo. En lugar de un punto central, si usamos una grilla de puntos y acomodamos el cambio en la estructura de datos resultante, nuestro programa creará ahora muchos círculos, cada uno con un valor de radio único definido por la distancia calibrada al Punto del Atractor.



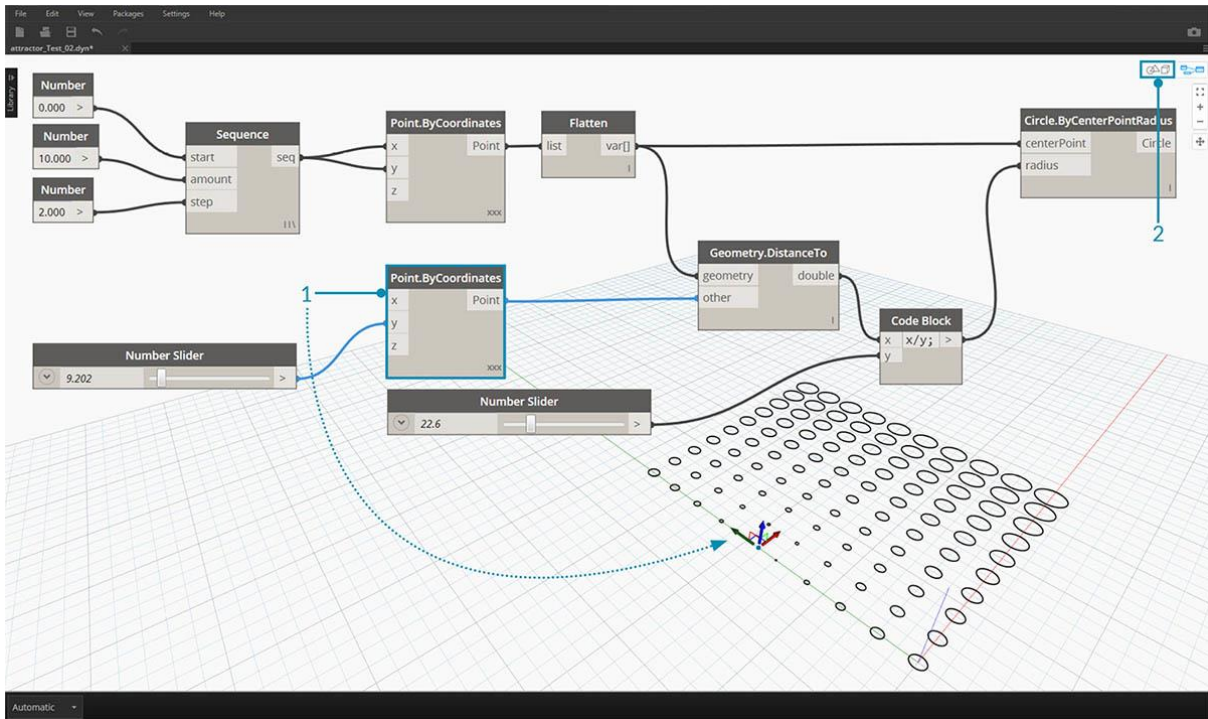


1. Agregue un **Nodo de secuencia numérica** y reemplace las entradas de **Point.ByCoordinates** : haga clic con el botón derecho en Point.ByCoordinates y seleccione **Cordón**> Referencia cruzada
2. Añadir un **Aplanar** Nodo después Point.ByCoordinates
3. La Vista previa en 3D se actualizará con una grilla de círculos

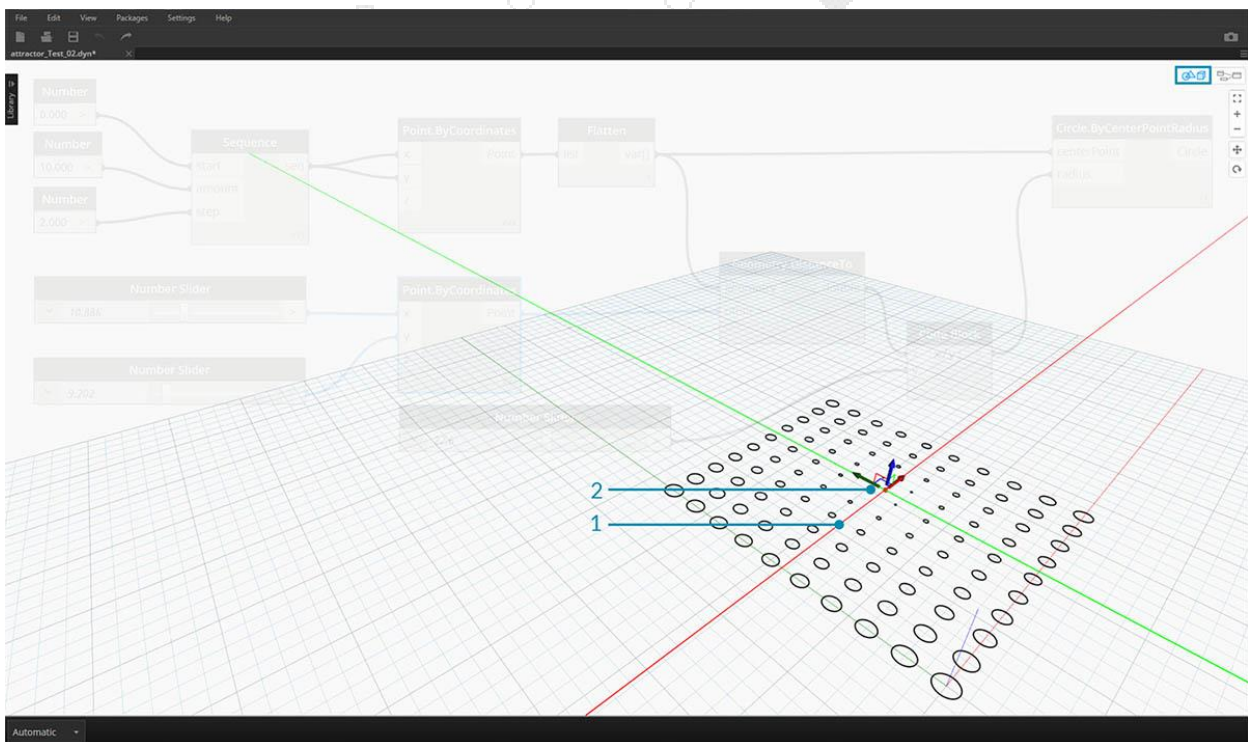
Ajuste con manipulación directa

A veces, la manipulación numérica no es el enfoque correcto. Ahora puede empujar y jalar manualmente la geometría de punto cuando navega en la vista previa 3D de fondo. También podemos controlar otra geometría que fue construida por un punto. Por ejemplo, **Sphere.ByCenterPointRadius** también es capaz de manipulación directa. Podemos controlar la ubicación de un punto a partir de una serie de valores X, Y y Z con **Point.ByCoordinates** . Sin embargo, con el enfoque Direct Manipulation, puede actualizar los valores de los controles deslizantes moviendo manualmente el punto en el modo de **navegación Vista previa en 3D** . Esto ofrece un enfoque más intuitivo para controlar un conjunto de valores discretos que identifican la ubicación de un punto.

DESDE 1988

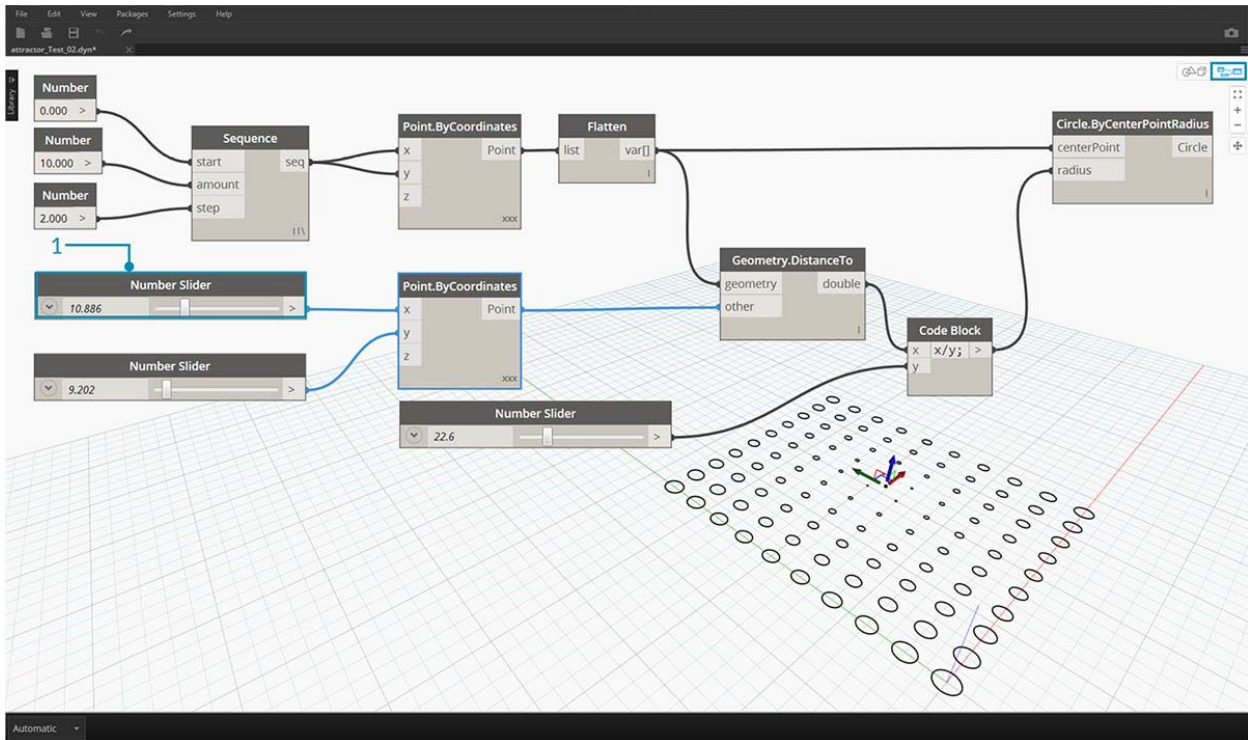


1. Para usar la **Manipulación directa**, seleccione el panel del punto que se va a mover: aparecerán flechas sobre el punto seleccionado.
2. Cambia al modo de **navegación de vista previa en 3D**.



1. Desplácese sobre el punto y aparecerán los ejes X, Y y Z.

- Haga clic y arrastre la flecha de color para mover el eje correspondiente, y los valores del **control deslizante numérico** se actualizarán en vivo con el punto movido manualmente.

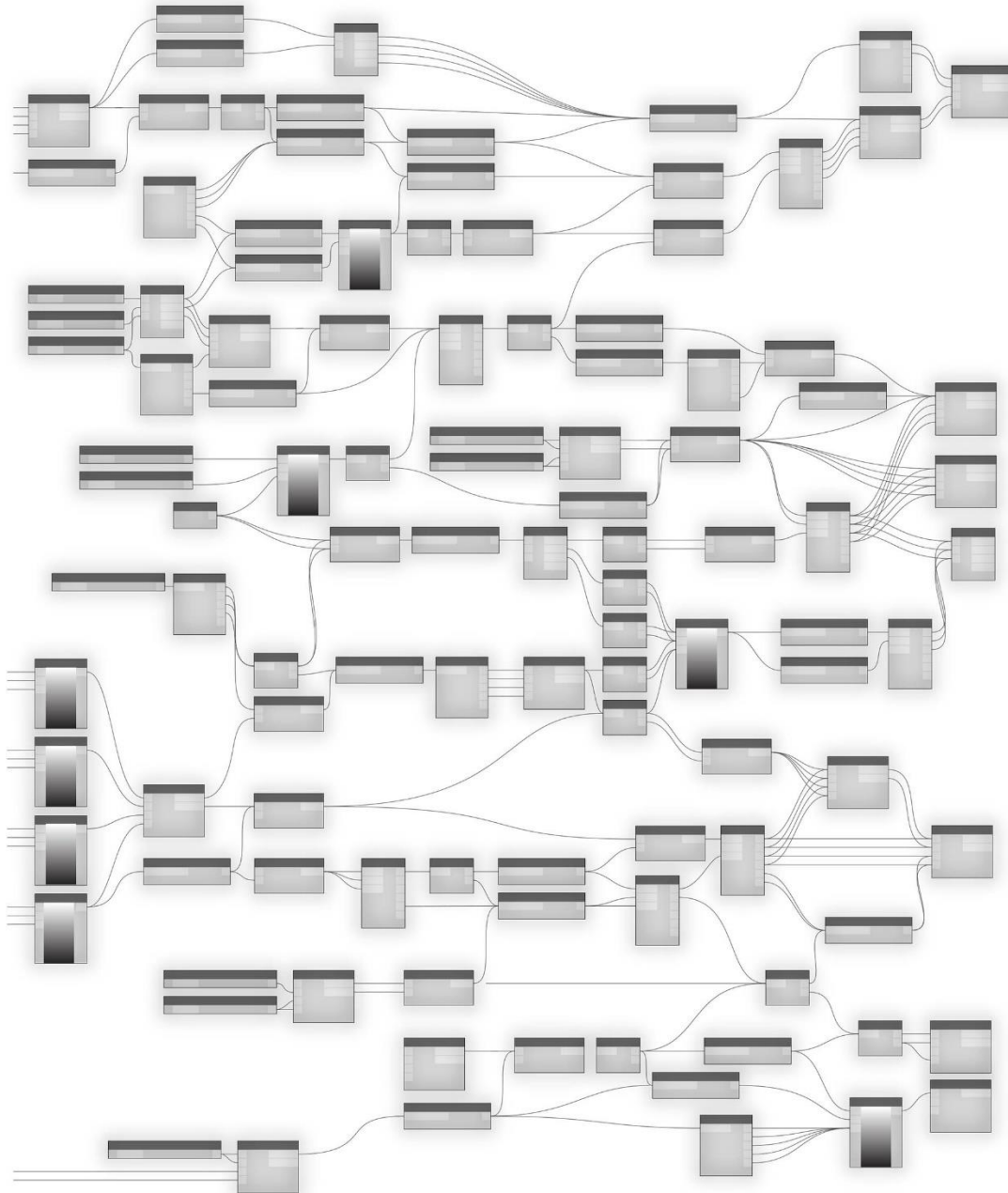


- Tenga en cuenta que antes de **Direct Manipulation** solo se conectaba un deslizador al componente **Point.ByCoordinates**. Cuando movemos manualmente el punto en la dirección X, Dynamo generará automáticamente un nuevo **Control deslizante numérico** para la entrada X.

DARCO
DESDE 1988

ANATOMÍA DE UN PROGRAMA VISUAL

Dynamo nos permite crear programas visuales en un área de trabajo conectando nodos con cables para especificar el flujo lógico del programa visual resultante. Este capítulo presenta los elementos de los Programas visuales, la organización de los Nodos disponibles en las Bibliotecas de Dynamo, las partes y estados de Nodos, y las mejores prácticas para sus Espacios de trabajo.

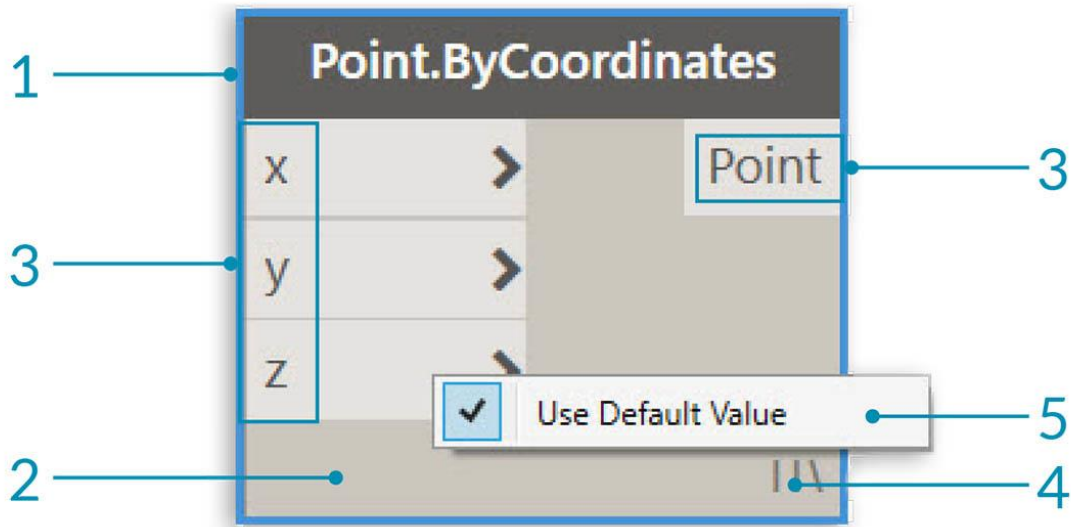


Nodos

En Dynamo, los **nodos** son los objetos que se conectan para formar un programa visual. Cada **nodo** realiza una operación, a veces puede ser tan simple como almacenar un número o puede ser una acción más compleja, como crear o consultar geometría.

Anatomía de un nodo

La mayoría de los Nodos en Dynamo están compuestos de cinco partes. Si bien hay excepciones, como los Nodos de entrada, la anatomía de cada nodo se puede describir de la siguiente manera:

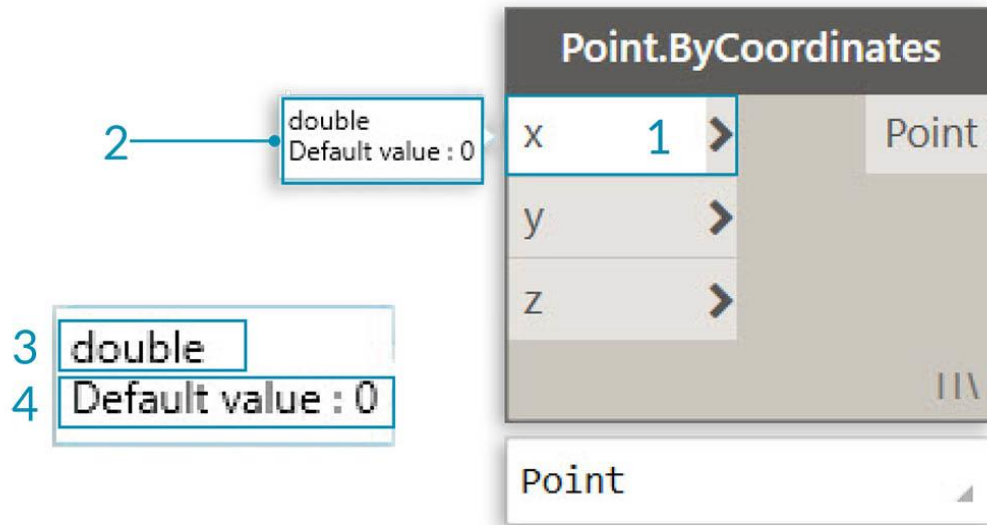


1. Nombre: el nombre del nodo con una convención de nomenclatura de Category.Name
2. Principal - El cuerpo principal del nodo - Hacer clic con el botón derecho aquí presenta opciones en el nivel de todo el nodo
3. Puertos (entrada y salida): los receptores para los cables que proporcionan los datos de entrada al nodo, así como los resultados de la acción del nodo.
4. Icono de cordón: indica la opción de cordón especificada para las entradas de la lista correspondiente (más sobre esto más adelante)
5. Valor predeterminado: haga clic con el botón derecho en un puerto de entrada; algunos nodos tienen valores predeterminados que pueden usarse o no usarse.

Puertos

Las Entradas y Salidas para Nodos se llaman Puertos y actúan como receptores para Alambres. Los datos ingresan al Nodo a través de Puertos a la izquierda y salen del Nodo después de que haya ejecutado su operación a la derecha. Los puertos esperan recibir datos de un cierto tipo. Por ejemplo, conectar un número como 2.75 a los Puertos en un Nodo de Coordenadas Punto por Punto dará como resultado la creación de un Punto; sin embargo, si suministramos "Rojo" al mismo Puerto, se producirá un error.

Consejo: Desplácese sobre un puerto para ver una información sobre herramientas que contiene el tipo de datos esperado.

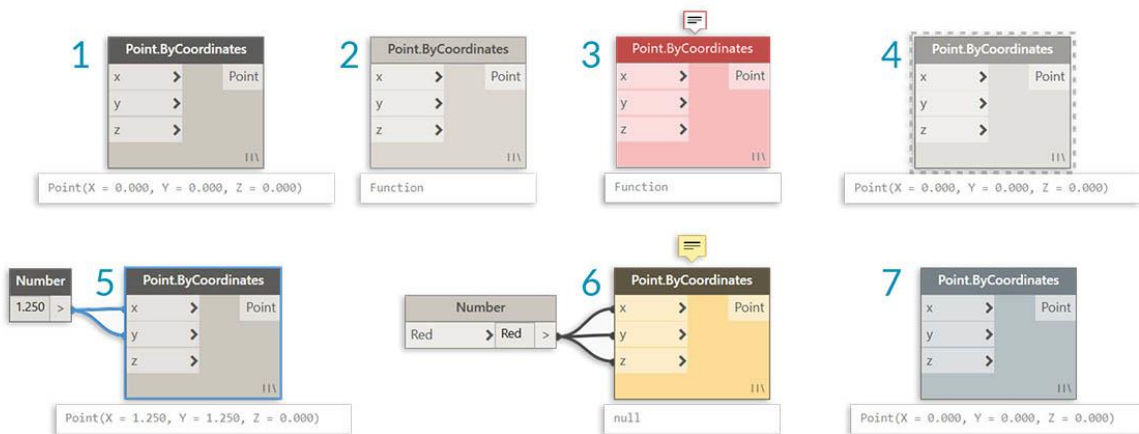


1. Etiqueta de puerto
2. Consejo de herramienta
3. Tipo de datos
4. Valor por defecto

Estados

Dynamo da una indicación del estado de la ejecución de su Programa Visual al renderizar Nodos con diferentes esquemas de color basados en el estado de cada Nodo. Además, al pasar el cursor o al hacer clic con el botón derecho sobre el Nombre o Puertos, se muestra información y opciones adicionales.

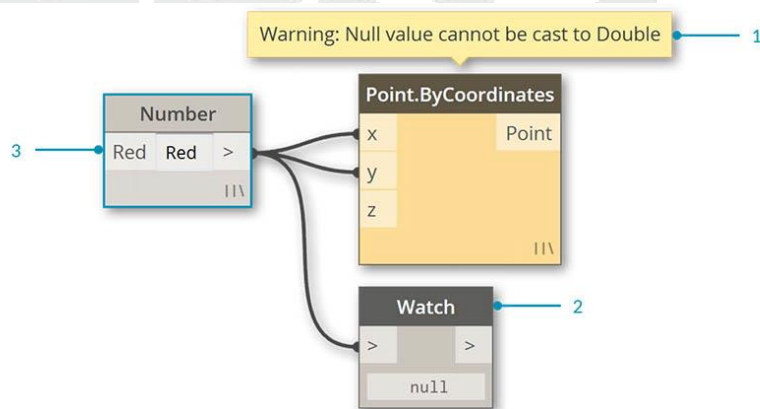
DARCO
DESDE 1988



1. Activo: los nodos con un fondo de nombre de gris oscuro están bien conectados y tienen todas sus entradas conectadas correctamente
2. Inactivo: los nodos grises están inactivos y deben conectarse con cables para formar parte del flujo de programas en el espacio de trabajo activo.
3. Estado de error - Rojo indica que el nodo está en un estado de error
4. Congelar: un nodo transparente tiene congelado encendido, suspendiendo la ejecución del nodo
5. Seleccionado - Los Nodos seleccionados actualmente tienen un resaltado Aqua en su borde
6. Advertencia: los Nodos amarillos están en estado de Advertencia, lo que significa que pueden tener tipos de datos incorrectos
7. Vista previa de fondo: gris oscuro indica que la vista previa de la geometría está desactivada

Si su Programa Visual contiene advertencias o errores, Dynamo proporcionará información adicional sobre el problema. Cualquier nodo que sea amarillo también tendrá una información sobre herramientas encima del nombre. Pase el mouse sobre la información sobre herramientas para expandirlo.

Consejo: Con esta información sobre información sobre herramientas en la mano, examine los Nodos ascendentes para ver si el tipo de datos o la estructura de datos requerida es errónea.



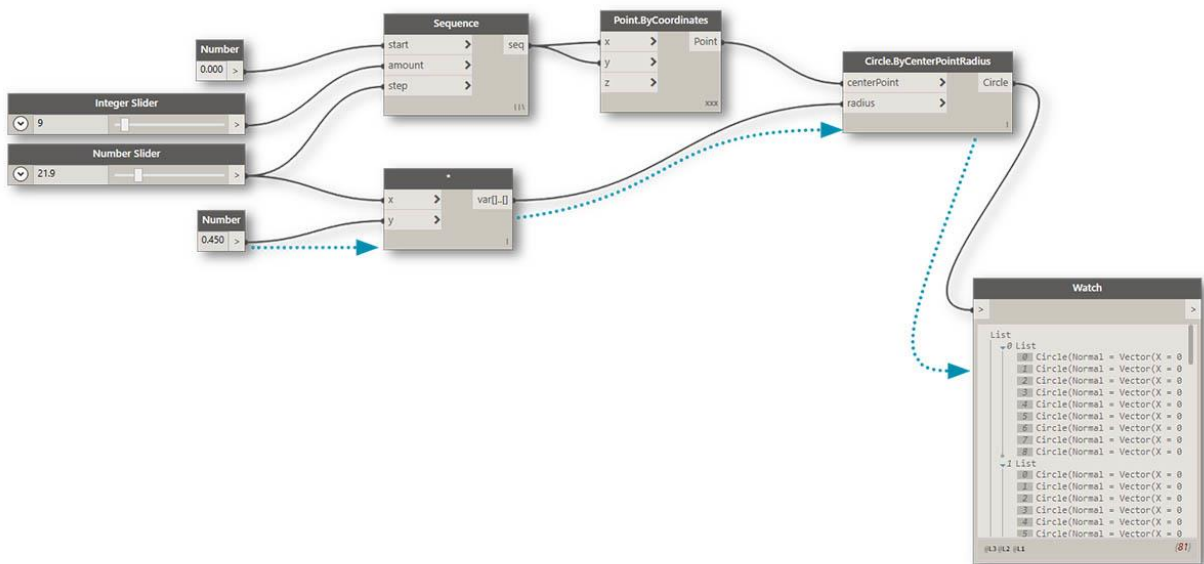
1. Warning Tooltip - "nulo" o sin datos no puede entenderse como un doble, es decir. un número
2. Use el Nodo de vigilancia para examinar los datos de entrada
3. Upstream the Number Node almacena "Red", no un número

Alambres

Los cables se conectan entre Nodos para crear relaciones y establecer el Flujo de nuestro Programa Visual. Podemos pensar en ellos literalmente como cables eléctricos que transportan pulsos de datos de un objeto al siguiente.

Flujo de programa

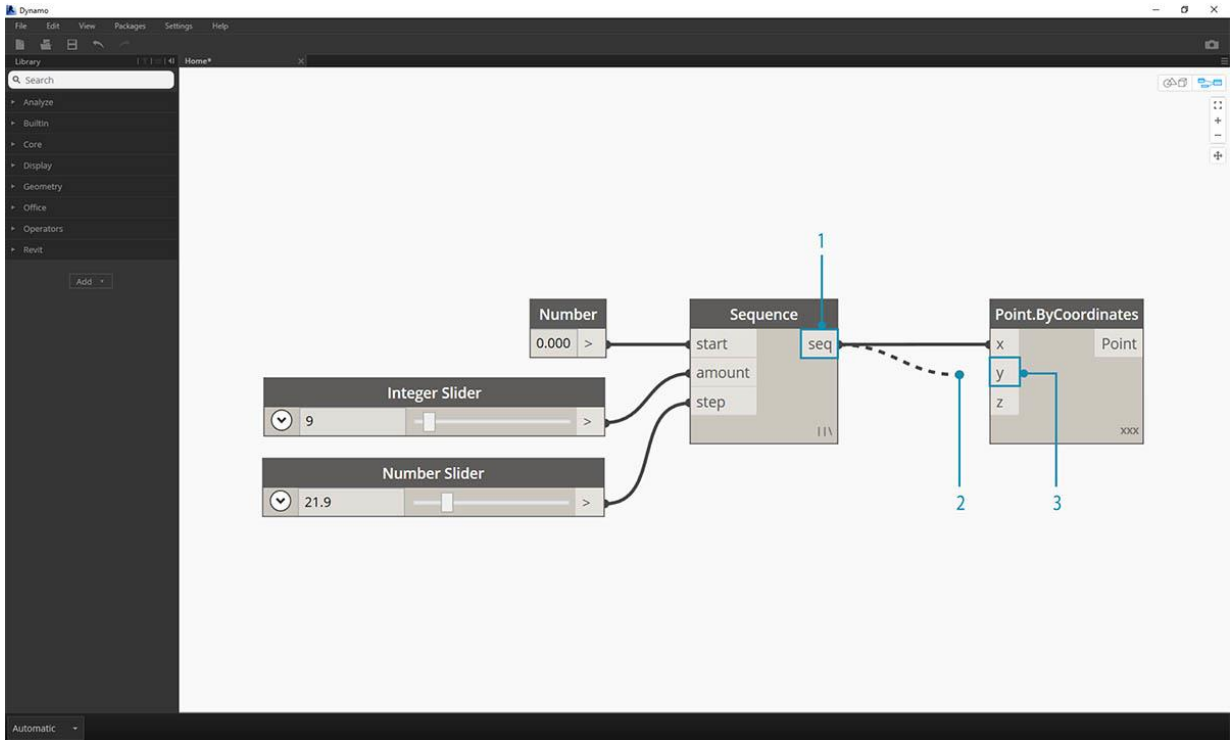
Los cables conectan el Puerto de salida desde un Nodo al Puerto de entrada de otro Nodo. Esta direccionalidad establece el **flujo de datos** en el programa visual. Aunque podemos organizar nuestros Nodos como deseamos en el Espacio de Trabajo, porque los Puertos de salida están ubicados en el lado derecho de los Nodos y los Puertos de entrada están en el lado izquierdo, generalmente podemos decir que el Flujo del Programa se mueve de izquierda a derecha.



Creando cables

Creamos un cable haciendo clic con el botón izquierdo del mouse en un puerto y luego haciendo clic izquierdo en el puerto de otro nodo para crear una conexión. Mientras estamos en el proceso de hacer una conexión, el cable aparecerá discontinuo y se ajustará para convertirse en líneas continuas cuando se conecte con éxito. Los datos siempre fluirán a través de este cable desde la salida a la entrada; sin embargo, podemos crear el cable en cualquier dirección en términos de la secuencia de hacer clic en los puertos conectados.

Consejo: Antes de completar la conexión con su segundo clic, permita que el Alambre encaje en un puerto y desplace el mouse hacia allí para ver la información sobre herramientas del Puerto.

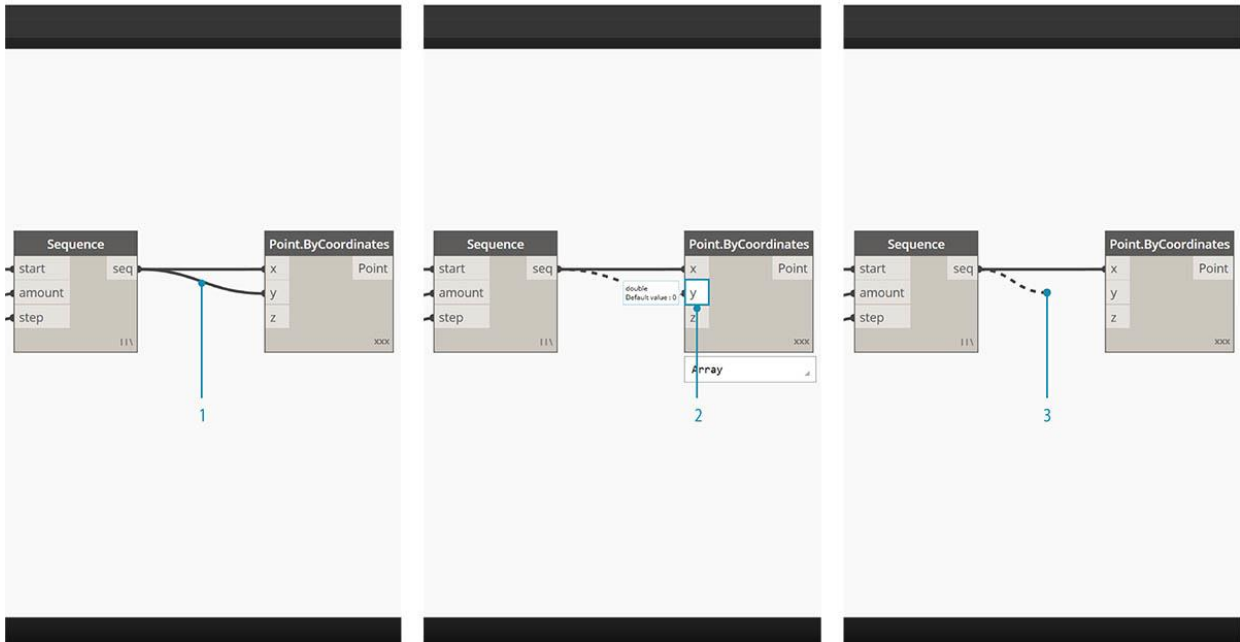


1. Haga clic en el sequeto de salida del nodo de secuencia numérica
2. A medida que mueve el mouse hacia otro puerto, el cable se corta
3. Haga clic en el ypuerto de entrada de Point.ByCoordiantes para completar la conexión

Edición de cables

Con frecuencia, querremos ajustar el flujo del programa en nuestro programa visual editando las conexiones representadas por los cables. Para editar un cable, haga clic izquierdo en el puerto de entrada del nodo que ya está conectado. Ahora tiene dos opciones:

DARCO
DESDE 1988



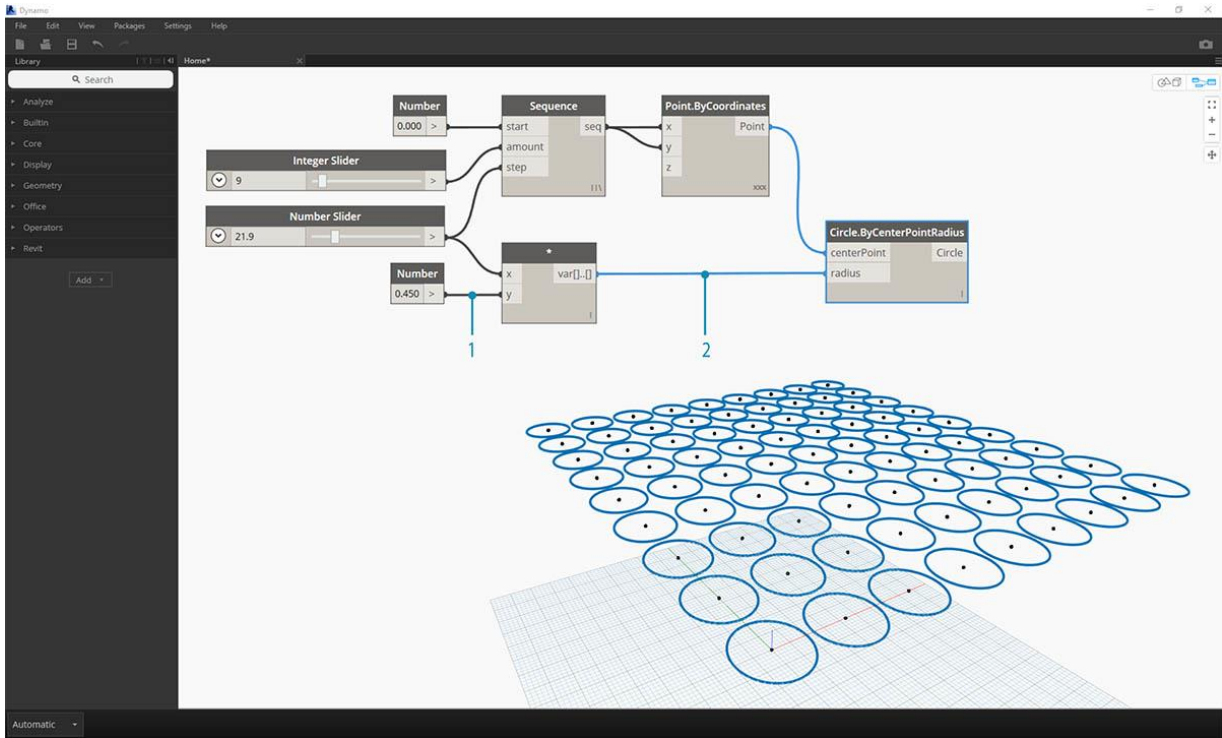
1. Cable existente
2. Para cambiar la conexión a un puerto de entrada, haga clic izquierdo en otro puerto de entrada
3. Para quitar el cable, retire el cable y haga clic izquierdo en el espacio de trabajo

Vistas previas de cables

Por defecto, nuestros Wires serán previsualizados con un trazo gris. Cuando se selecciona un Nodo, se renderizará cualquier Cable de conexión con el mismo resaltado acuático que el Nodo.

DARCO

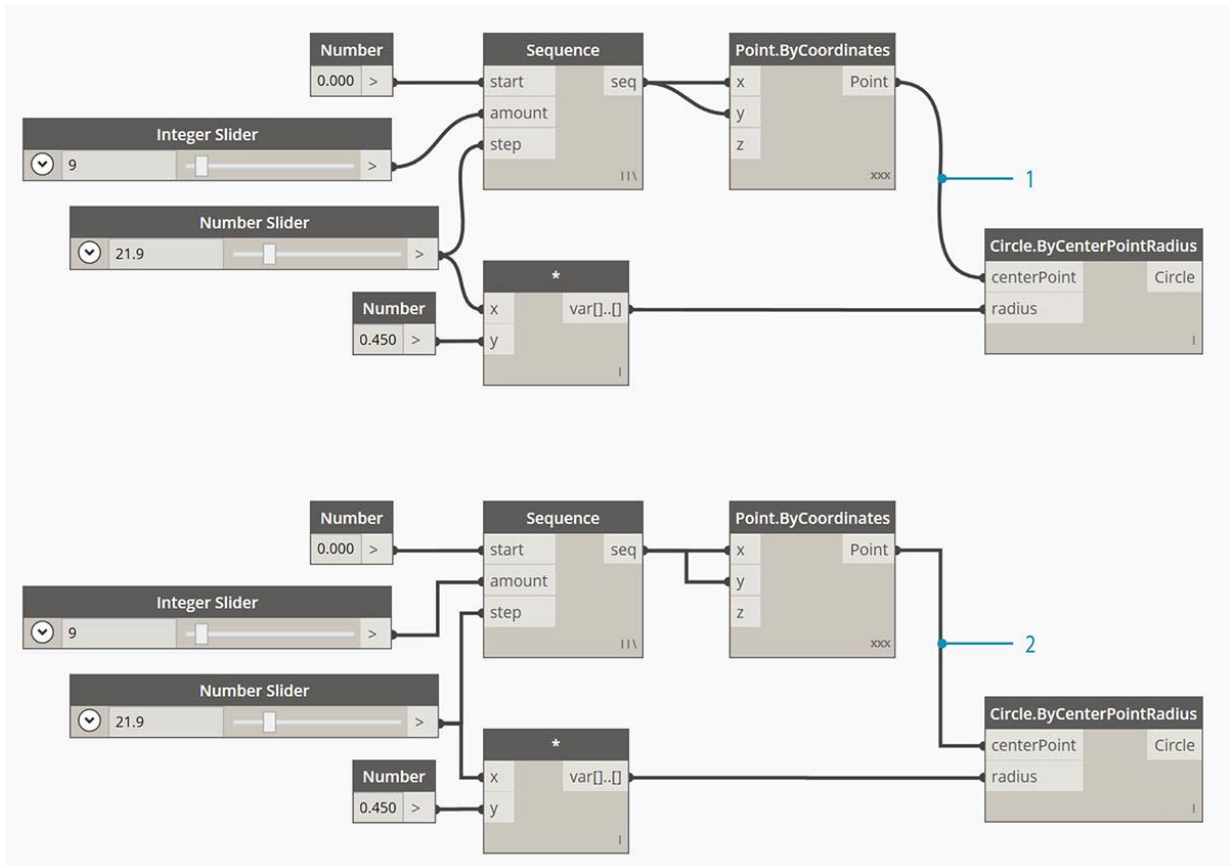
DESDE 1988



1. Cable predeterminado
2. Alambre resaltado

Dynamo también nos permite personalizar el aspecto de nuestros cables en el área de trabajo a través del menú Vista > Conectores. Aquí podemos alternar entre Curve o Polyline Wires o apagarlos todos juntos.

DARCO
DESDE 1988



1. Tipo de conector: curvas
2. Tipo de conector: Polylines

Biblioteca Dynamo

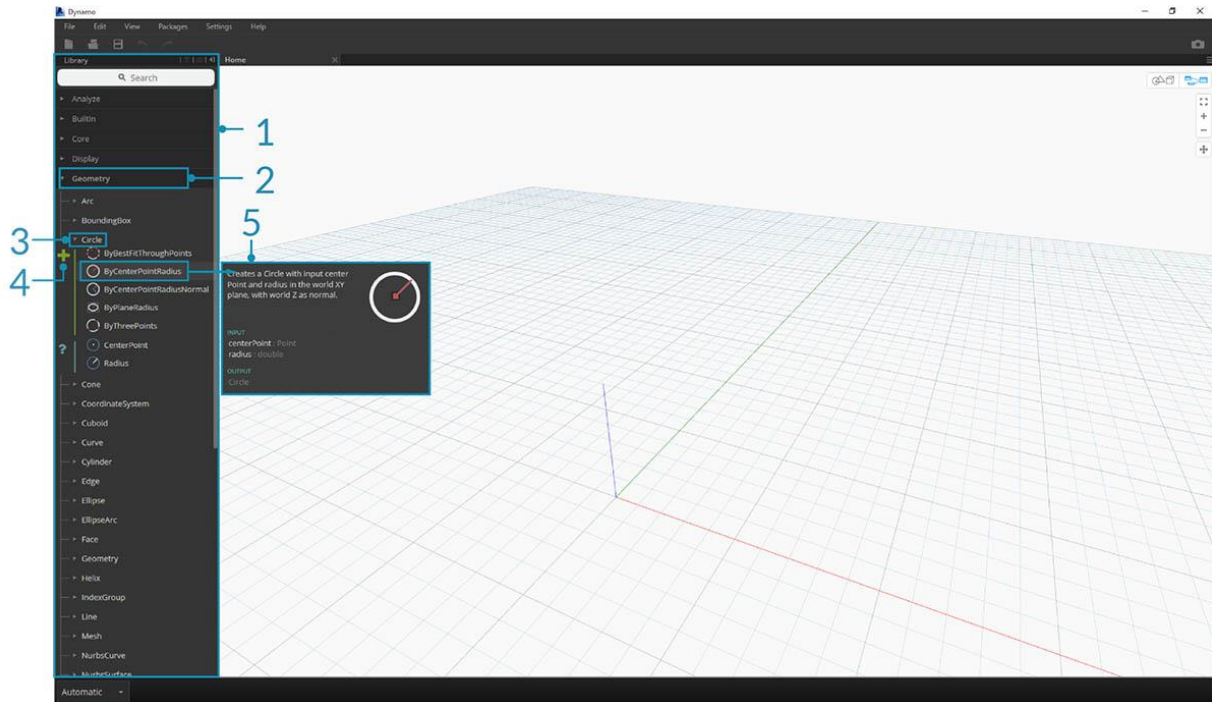
La **biblioteca de Dynamo** contiene los nodos que agregamos al área de trabajo para definir programas visuales para su ejecución. En la Biblioteca, podemos buscar o navegar a Nodos. Los nodos que se incluyen aquí (los nodos básicos instalados, los nodos personalizados que definimos y los nodos del administrador de paquetes que agregamos a Dynamo) están organizados jerárquicamente por categoría. Repasemos esta organización y exploremos los Nodos clave que usaremos con frecuencia.

Biblioteca de Bibliotecas

La **biblioteca de** Dynamo con la que interactuamos en la aplicación es en realidad una colección de bibliotecas funcionales, cada una con nodos agrupados por categoría. Si bien esto puede parecer obtuso al principio, es un marco flexible para organizar los Nodos que vienen con la instalación predeterminada de Dynamo, y es aún mejor en el futuro cuando comenzamos a ampliar esta funcionalidad básica con Nodos personalizados y Paquetes adicionales.

El esquema organizacional

La sección **Biblioteca** de Dynamo UI se compone de bibliotecas organizadas jerárquicamente. A medida que profundizamos en la Biblioteca, estamos navegando secuencialmente por una biblioteca, las categorías de la biblioteca y las subcategorías de la categoría para encontrar el Nodo.

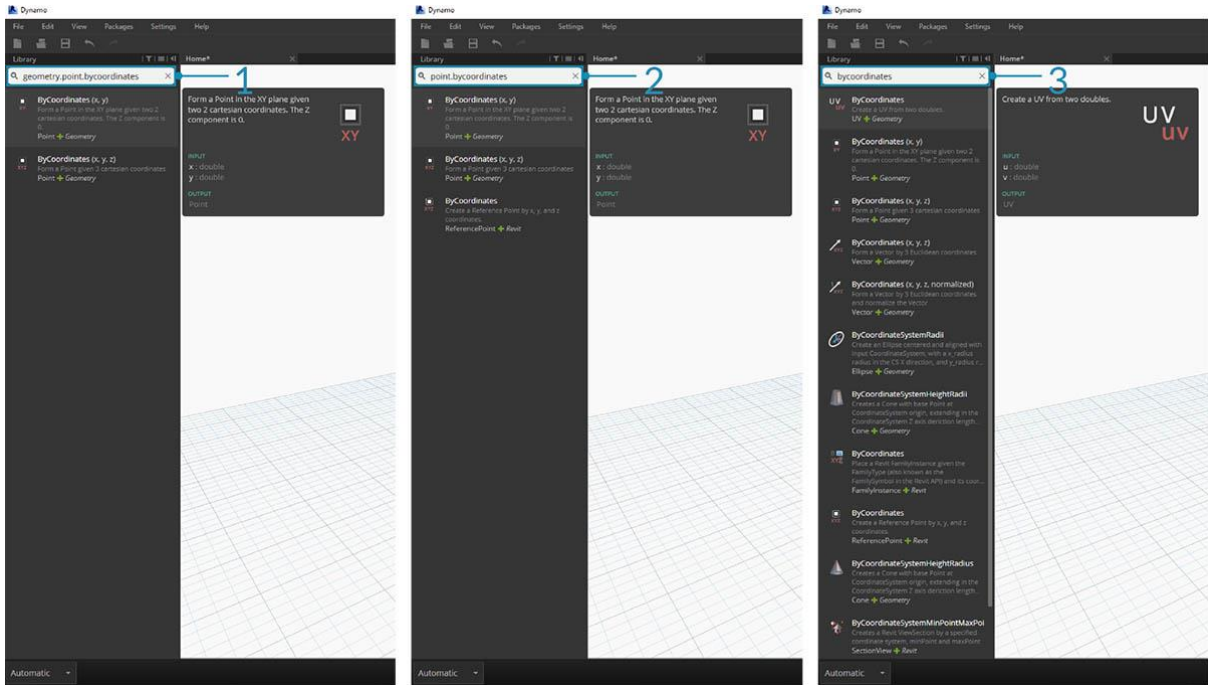


1. La biblioteca: la región de la interfaz Dynamo
2. Una biblioteca: una colección de categorías relacionadas, como la **geometría**
3. Una Categoría - Una colección de Nodos relacionados, como todo lo relacionado con **Círculos**
4. Una subcategoría: desglose de los nodos dentro de la categoría, generalmente por **crear, acción o consulta**
5. Un nodo: los objetos que se agregan al área de trabajo para realizar una acción

Convenciones de nombres

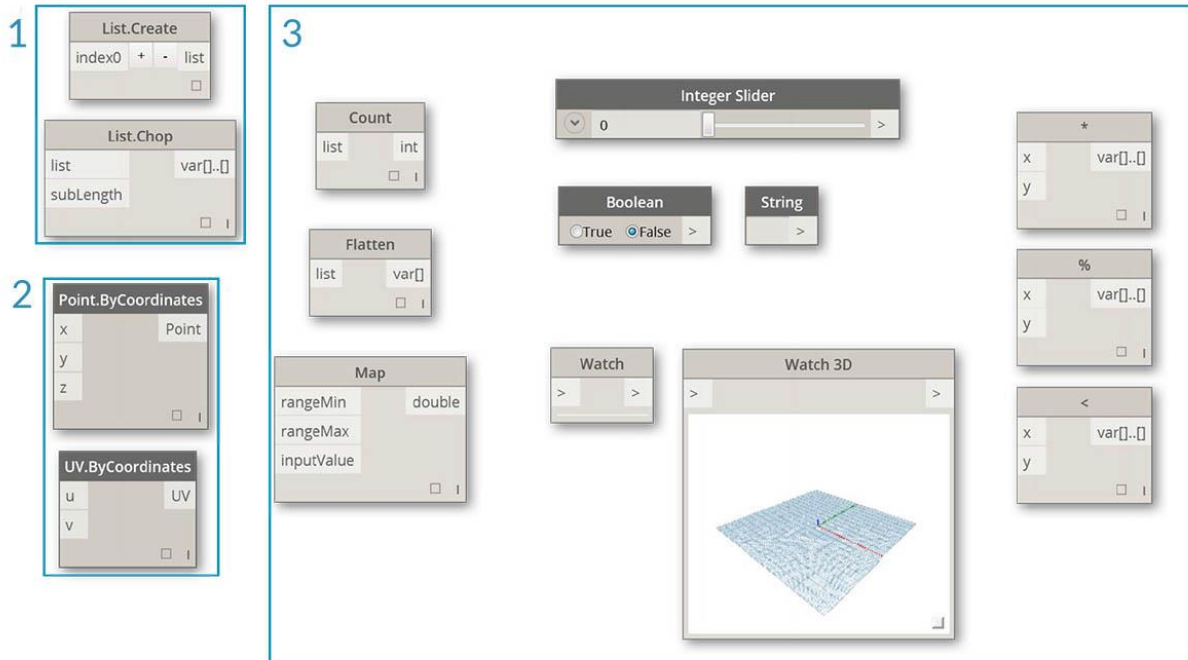
La jerarquía de cada biblioteca se refleja en el Nombre de los Nodos agregados al Espacio de trabajo, que también podemos usar en el Campo de búsqueda o con los Bloques de código (que usan el *lenguaje textual Dynamo*). Además de usar palabras clave para tratar de encontrar nodos, podemos escribir la jerarquía separada por un punto.

Escribir en diferentes partes del lugar del Nodo en la jerarquía de la Biblioteca en el `library.category.nodeNameformato` arroja resultados diferentes:



1. library.category.nodeName
2. category.nodeName
3. nodeName o keyword

Normalmente, el nombre del nodo en el espacio de trabajo se representará en el category.nodeNameformato, con algunas excepciones notables particularmente en las categorías de entrada y vista. Tenga cuidado con los Nodos nombrados de manera similar y tenga en cuenta la diferencia de categoría:



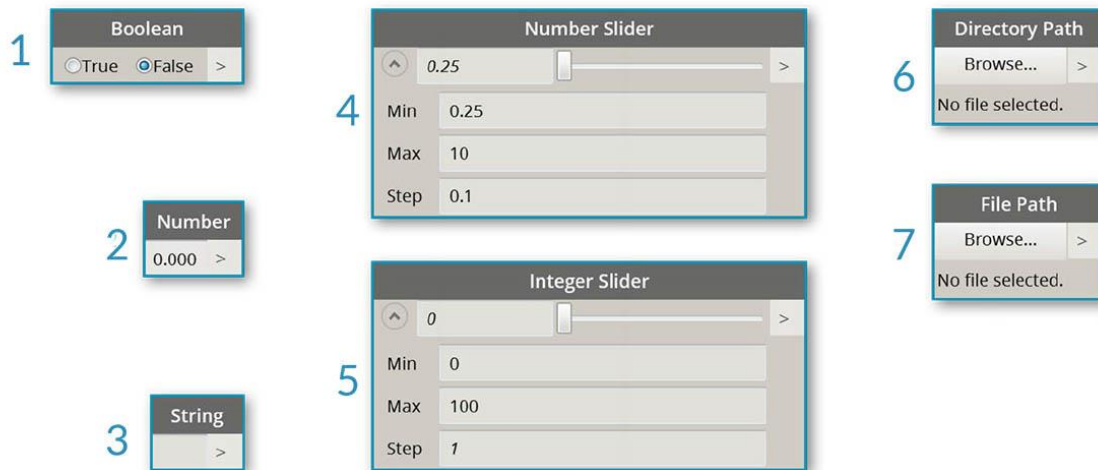
1. Point.ByCoordinates y UV.ByCoordinates tienen el mismo nombre pero provienen de diferentes categorías
2. Los nodos de la mayoría de las bibliotecas incluirán el formato de categoría
3. Excepciones notables incluyen funciones integradas, Core.Input, Core.View y Operators

Nodos usados con frecuencia

Con cientos de nodos incluidos en la instalación básica de Dynamo, ¿cuáles son esenciales para desarrollar nuestros programas visuales? Centrémonos en aquellos que nos permiten definir los parámetros de nuestro programa (Entrada), ver los resultados de la acción de un Nodo (Observar) y definir las entradas o la funcionalidad por medio de un atajo (Bloque de Código).

Entrada

Los Nodos de entrada son el medio principal para que el Usuario de nuestro Programa Visual sea usted mismo u otra persona para interactuar con los parámetros clave. Estos son los Nodos disponibles en la Categoría de Entrada de la Biblioteca Central:

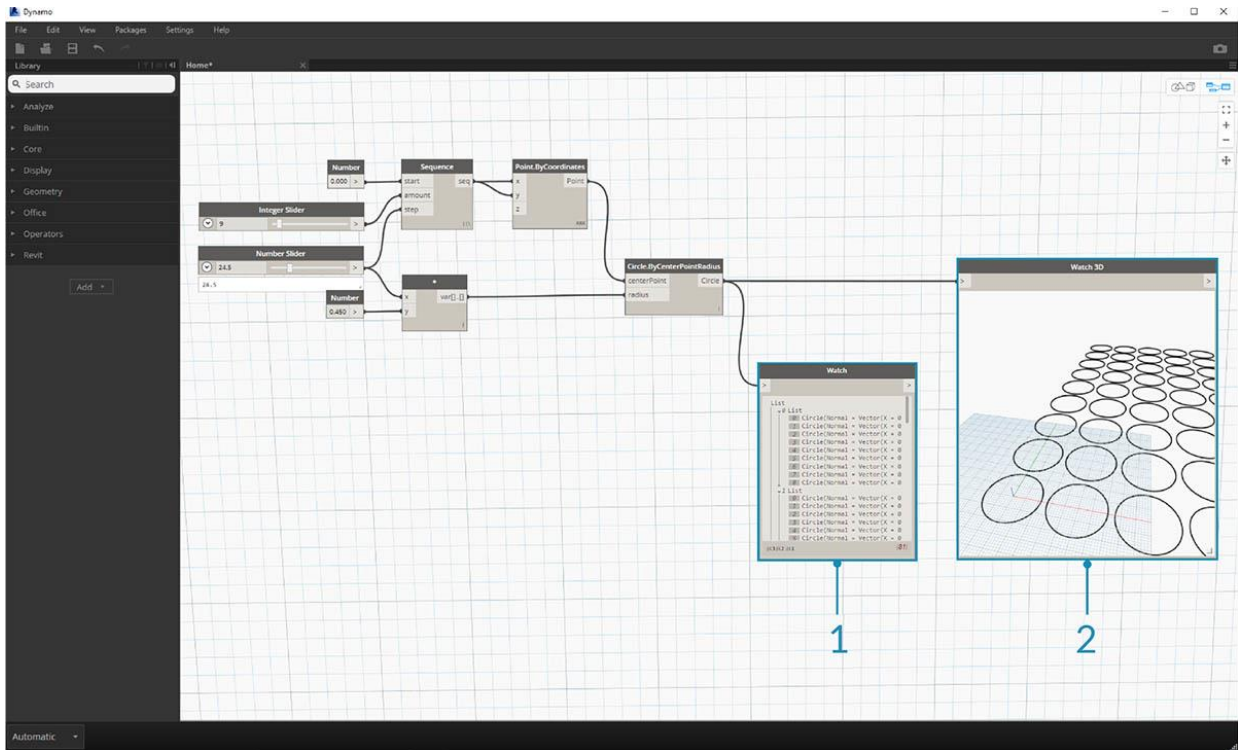


1. Booleano
2. Número
3. Cuerda
4. Deslizador de número
5. Control deslizante entero
6. Ruta de directorio
7. Ruta de archivo

Reloj

Los Nodos de Vigilancia son esenciales para administrar los datos que fluyen a través de su Programa Visual. Si bien puede ver el resultado de un Nodo a través de la vista previa de los datos del Nodo, es posible que desee mantenerlo revelado en un Nodo de **Vigilancia** o ver los resultados de la geometría a través de un Nodo de **Watch3D**. Ambos se encuentran en la Categoría de vista en la Biblioteca principal.

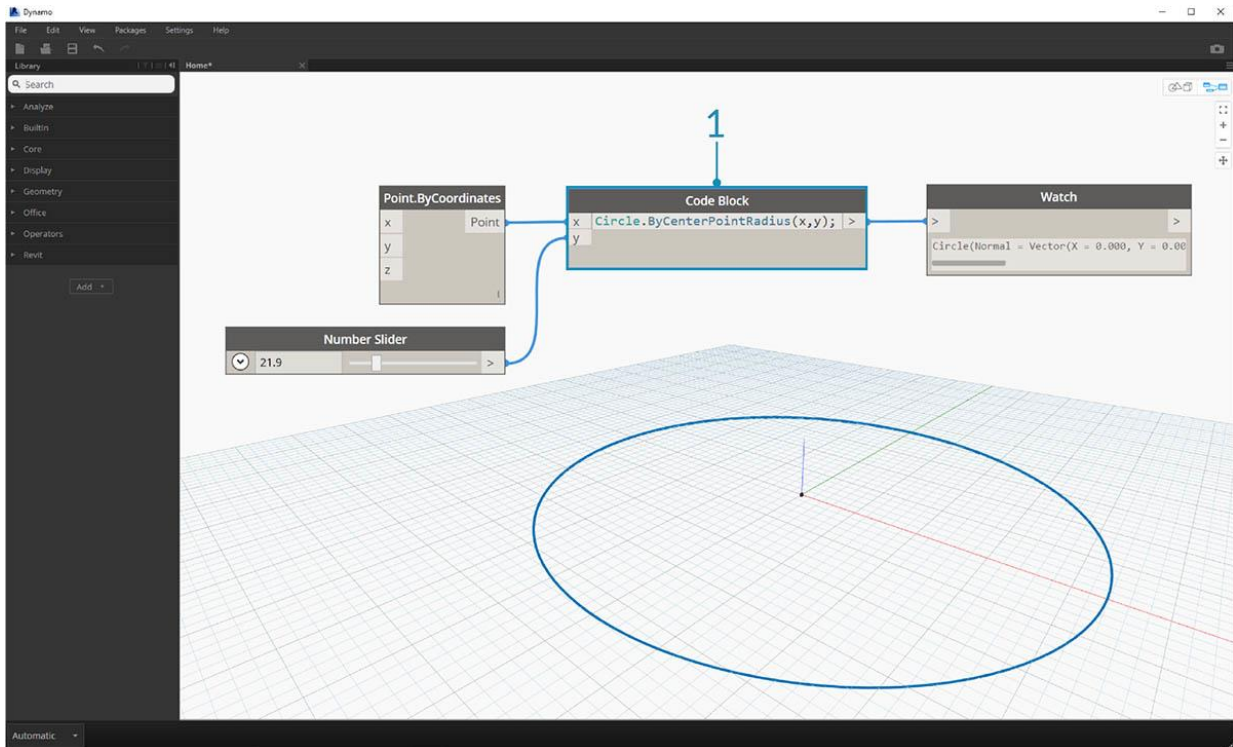
Consejo: Ocasionalmente, la vista previa 3D puede distraer cuando su programa visual contiene muchos nodos. Considere desmarcar la opción Mostrar vista previa en segundo plano en el menú de configuración y usar un nodo Watch3D para obtener una vista previa de su geometría.



1. Mirar: tenga en cuenta que cuando selecciona un elemento en el Nodo de vigilancia, se etiquetará en Watch3D y Previsualizaciones 3D.
2. Watch3D: agarra el agarre inferior derecho para cambiar el tamaño y navegar con el mouse de la misma manera que lo harías en la vista previa en 3D

Bloque de código

Los Nodos de **Bloque de Código** se pueden usar para definir un bloque de código con líneas separadas por punto y coma. Esto puede ser tan simple como X/Y. También podemos usar Bloques de Código como un atajo para definir una Entrada de Número o llamar a la funcionalidad de otro Nodo. La sintaxis para hacerlo sigue la Convención de nombres del lenguaje textual Dynamo, DesignScript, y se trata en la Sección 7.2. Tratemos de hacer un círculo con este atajo:



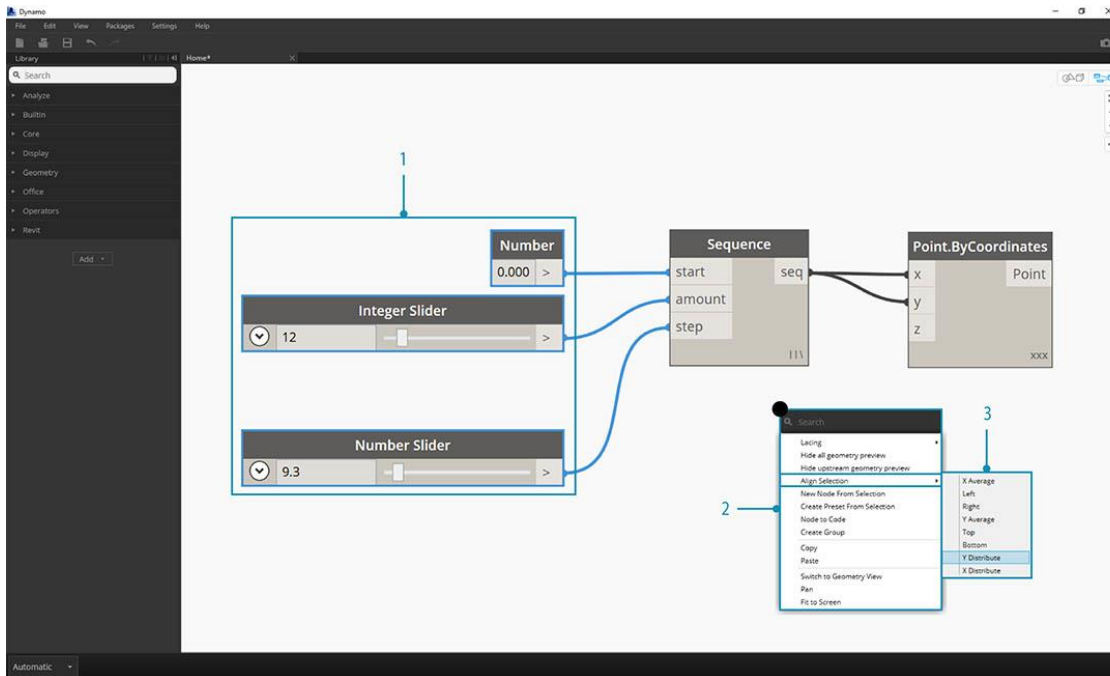
1. Haga doble clic para crear un nodo de **bloque de código**
2. Tipo `Circle.ByCenterPointRadius(x,y);`
3. Al hacer clic en el área de trabajo para borrar la selección, debe agregar xy hacer yentradas automáticamente
4. Cree un nodo **Point.ByCoordinates** y un **deslizador numérico** y luego conéctelos a las entradas del bloque de código
5. El resultado de ejecutar el programa visual debe ser un círculo en la vista previa en 3D

Administrar su programa

Trabajar dentro de un proceso de Programación Visual puede ser una actividad creativa poderosa, pero muy rápidamente el Flujo del Programa y las entradas del usuario clave pueden oscurecerse por la complejidad y / o el diseño del Espacio de Trabajo. Repasemos algunas de las mejores prácticas para administrar su programa.

Alineación

Una vez que hayamos agregado más de unos pocos nodos al espacio de trabajo, es posible que deseemos reorganizar el diseño de los Nodos para mayor claridad. Al seleccionar más de un nodo y hacer clic con el botón derecho en el área de trabajo, la ventana emergente incluye un menú **Alinear selección** con opciones de justificación y distribución en X e Y.

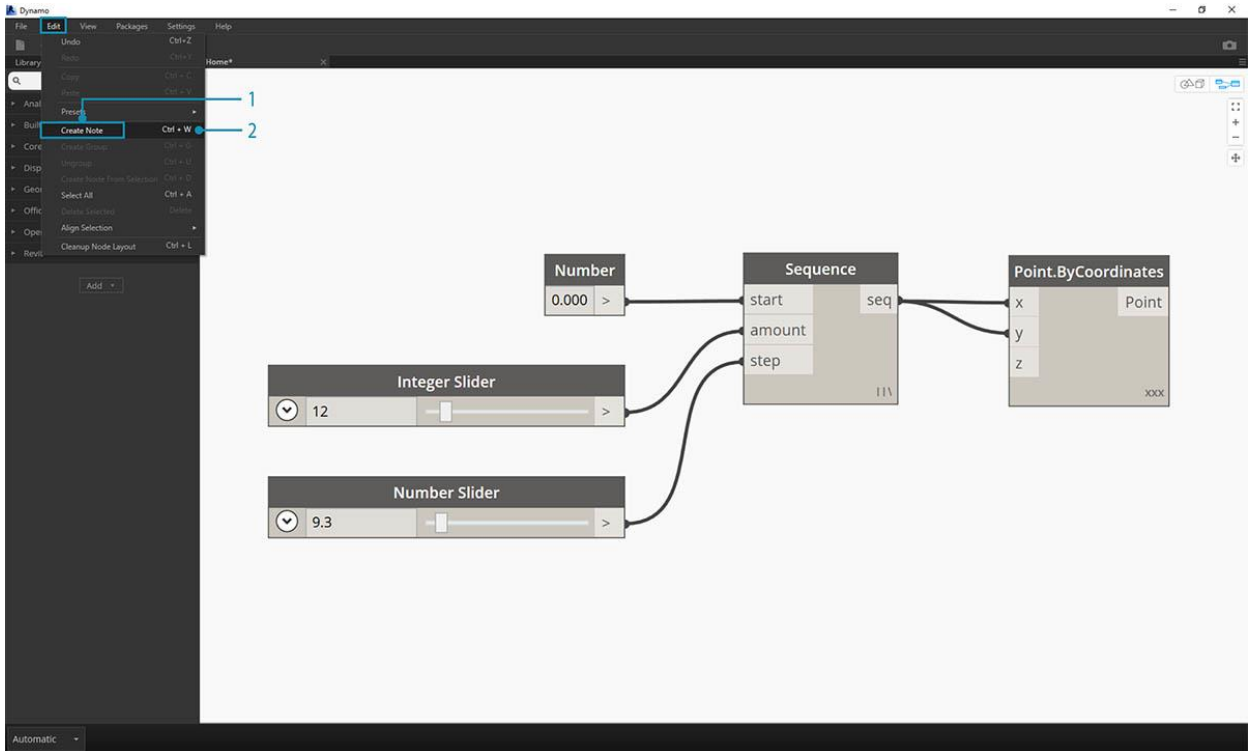


1. Seleccione más de un nodo
2. Haga clic derecho en el espacio de trabajo
3. Use las opciones **Align Selection**

Notas

Con un poco de experiencia, podremos "leer" el Programa Visual al revisar los Nombres de Nodo y seguir el Flujo del Programa. Para los usuarios de todos los niveles de experiencia, también es una buena práctica incluir etiquetas y descripciones en lenguaje sencillo. Dynamo tiene un nodo de **notas** con un campo de texto editable para hacerlo. Podemos agregar notas al espacio de trabajo de dos maneras:

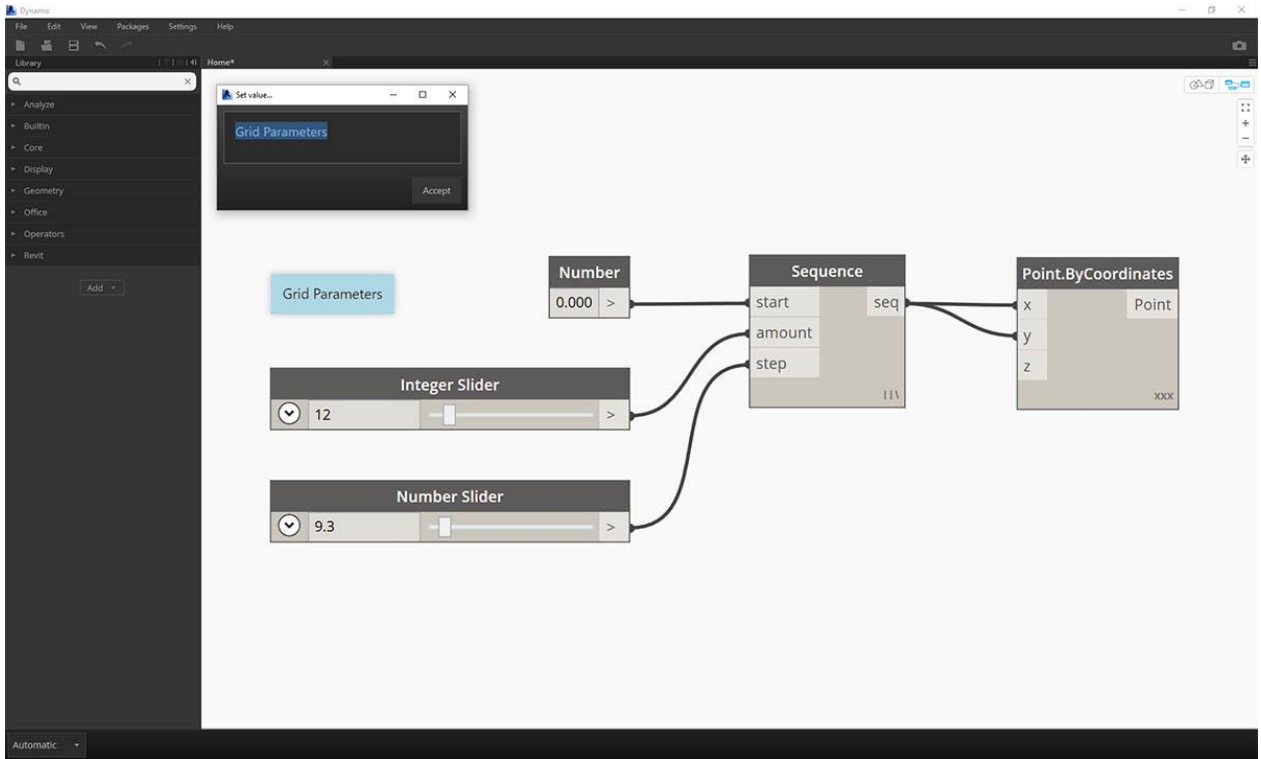
DARCO
DESDE 1988



1. Navegue al menú Archivo> Crear nota
2. Usa el atajo de teclado Ctrl + W

Una vez que la Nota se agrega al Espacio de trabajo, aparecerá un campo de texto que nos permitirá editar el texto en la Nota. Después de que se crean, podemos editar la Nota haciendo doble clic o haciendo clic con el botón derecho en el Nodo de Nota.

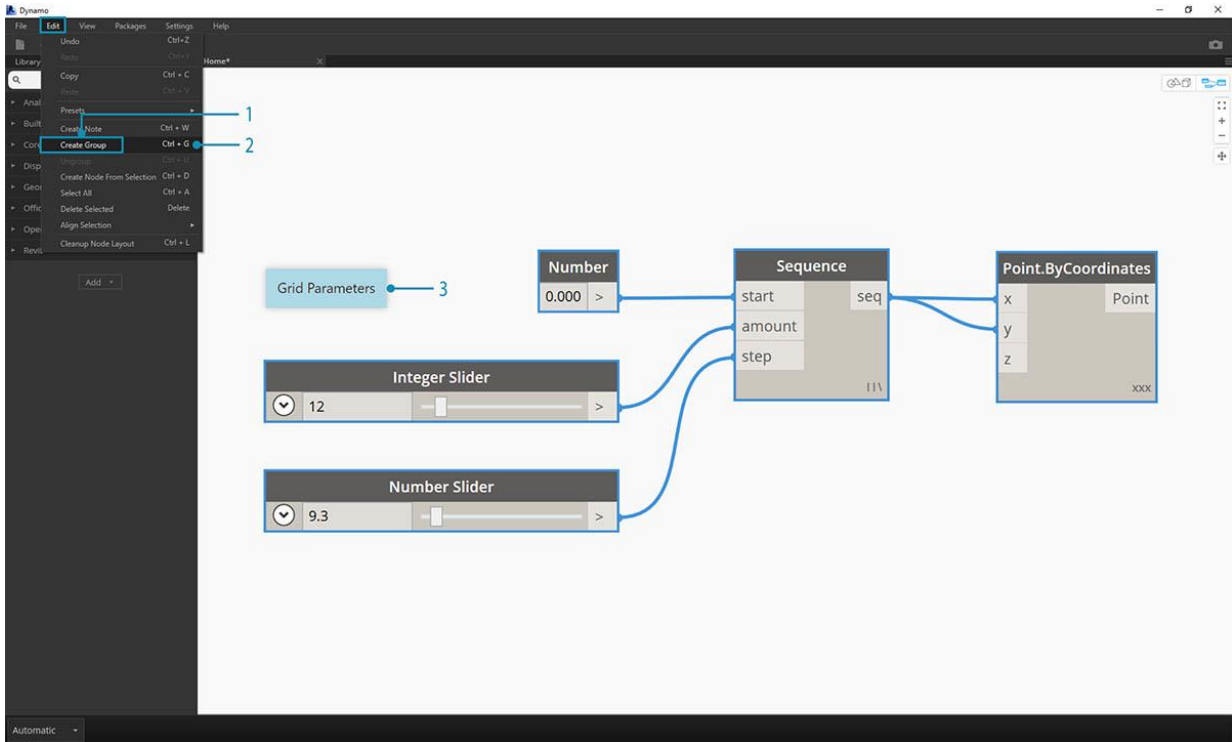
DARCO
DESDE 1988



Agrupamiento

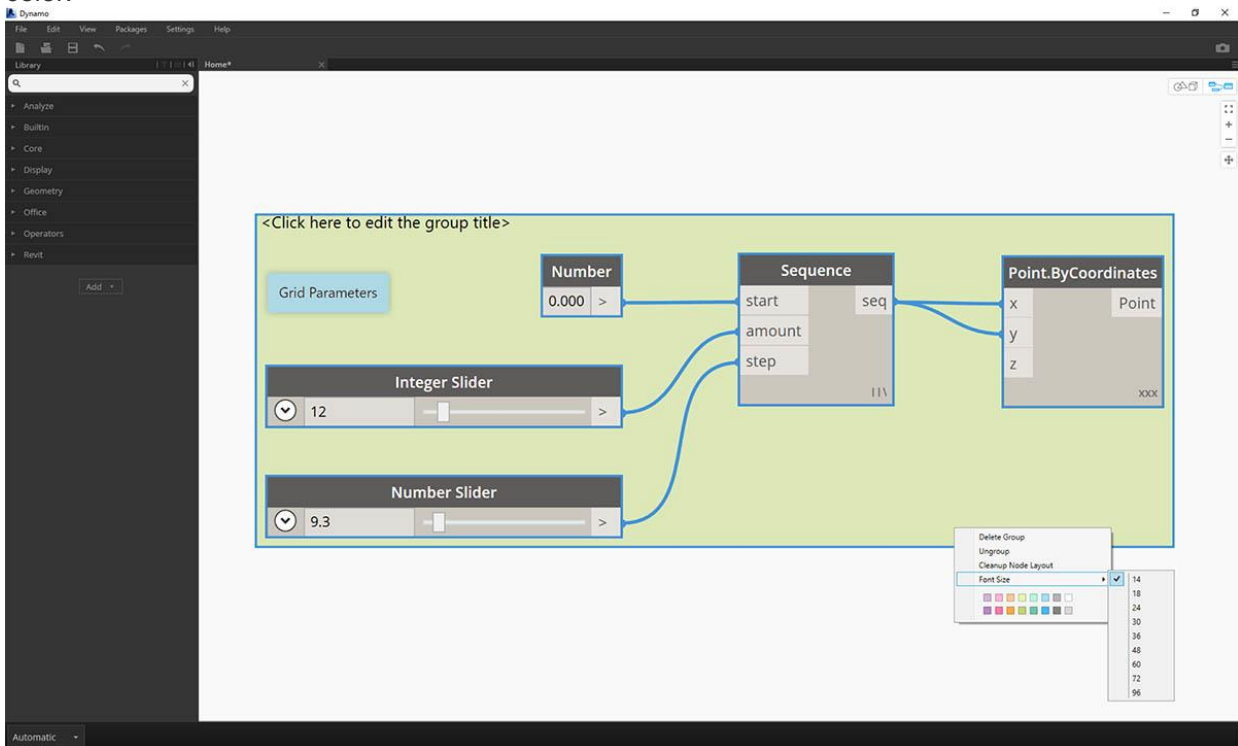
Cuando nuestro programa visual crece, es útil identificar los pasos más grandes que se ejecutarán. Podemos destacar colecciones más grandes de Nodos con un **Grupo** para etiquetarlos con un rectángulo de color en el fondo y un título. Hay tres formas de hacer un grupo con más de un nodo seleccionado:

DARCO
DESDE 1988



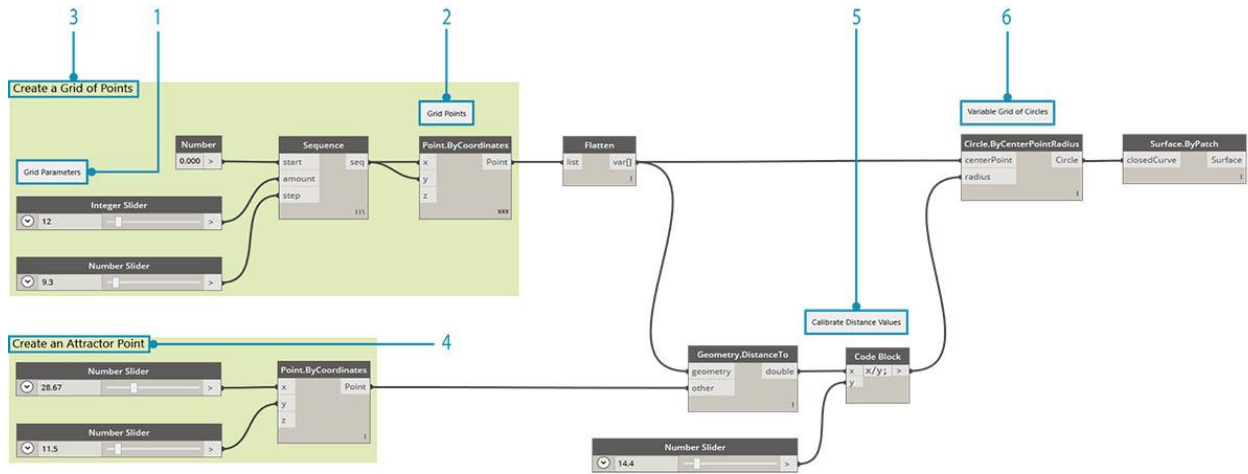
1. Navegue al menú Archivo > Crear grupo
2. Usa el atajo de teclado Ctrl + G
3. Haga clic derecho en el espacio de trabajo y seleccione "Crear grupo"

Una vez que se crea un grupo, podemos editar su configuración, como el título y el color.



Consejo: Usar notas y grupos es una forma efectiva de anotar su archivo y aumentar la legibilidad.

Aquí está nuestro programa de la Sección 2.4 con Notas y Grupos agregados:



1. Nota: "Parámetros de cuadrícula"
2. Nota: "Puntos de rejilla"
3. Grupo: "Crear una grilla de puntos"
4. Grupo: "Crear un punto atractor"
5. Nota: "Calibrar valores de distancia"
6. Nota: "Cuadrícula variable de círculos"

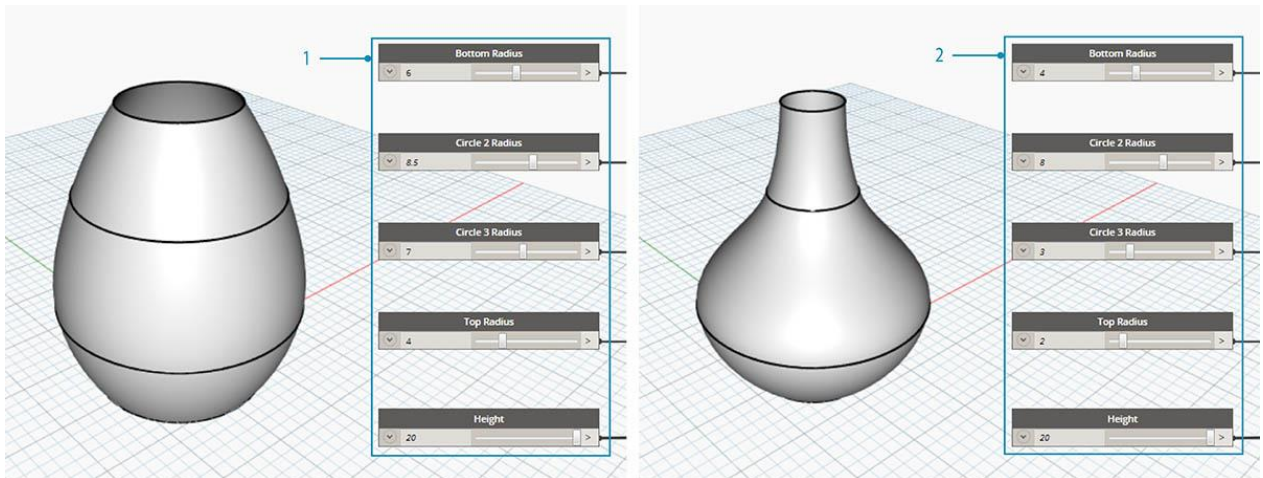
Administrar sus datos con ajustes preestablecidos

En la sección anterior analizamos la administración de programas al alinear, agrupar y anotar nodos para organizar el área de trabajo. Estas mejores prácticas ayudan a reducir parte de la complejidad visual de su gráfico. Ahora profundicemos y organicemos la complejidad del contenido. A menudo, un gráfico Dynamo tiene una amplia gama de parámetros que ofrecen innumerables iteraciones. Queremos organizar el rango de opciones para que podamos tomar decisiones de diseño reales, y aquí es donde entran en juego los Presets.

Imagine que ha creado un gráfico de Dynamo para compartirlo con un equipo, de modo que cada miembro del equipo pueda explorar el modelo paramétrico. Los miembros de su equipo tienen una amplia experiencia en programación visual, por lo que desea ofrecerles una dirección para que exploren diferentes esquemas de diseño. La manera más sencilla y fácil de usar para hacer esto es con los ajustes preestablecidos: puede establecer cualquier cantidad de parámetros para definir una iteración de diseño específica. Esto le permite volver a cargar los esquemas anteriores y trabajar con ellos paramétricamente.

Presets

Los ajustes preestablecidos son una forma de tomar el valor actual de una selección de nodos de entrada y guardarlos como un estado preestablecido. Estos estados se pueden restaurar a través del menú Edición > Ajustes preestablecidos. Los ajustes preestablecidos se pueden usar para crear y comparar iteraciones de diseño. Los ajustes preestablecidos se guardan con el archivo, lo que los convierte en una herramienta útil para compartir o solicitar comentarios. También permiten que otro usuario interactúe con el gráfico sin tener que buscar las entradas relevantes, o sintonizar un conjunto de valores que funcionen bien juntos desde una perspectiva de diseño.

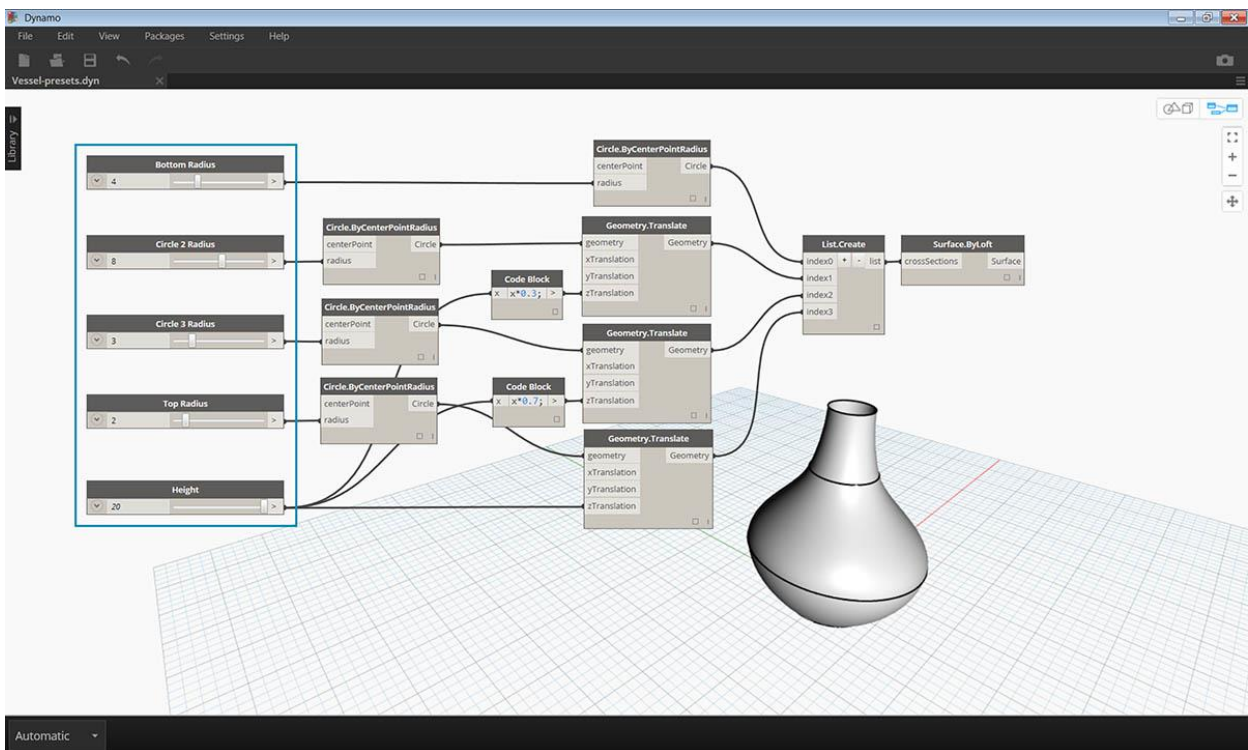


1. Preset 1
2. Preset 2

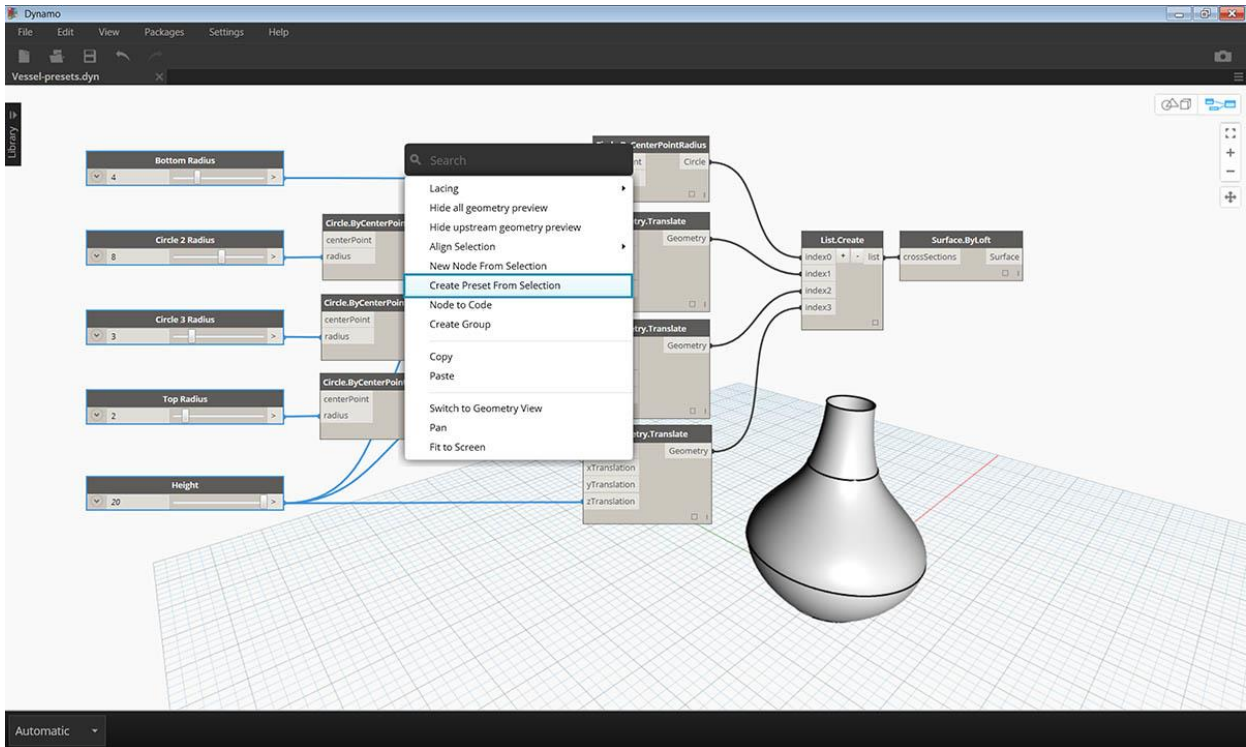
Creando Presets

Descargue el archivo de ejemplo que acompaña a este ejercicio Presets.dyn . Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

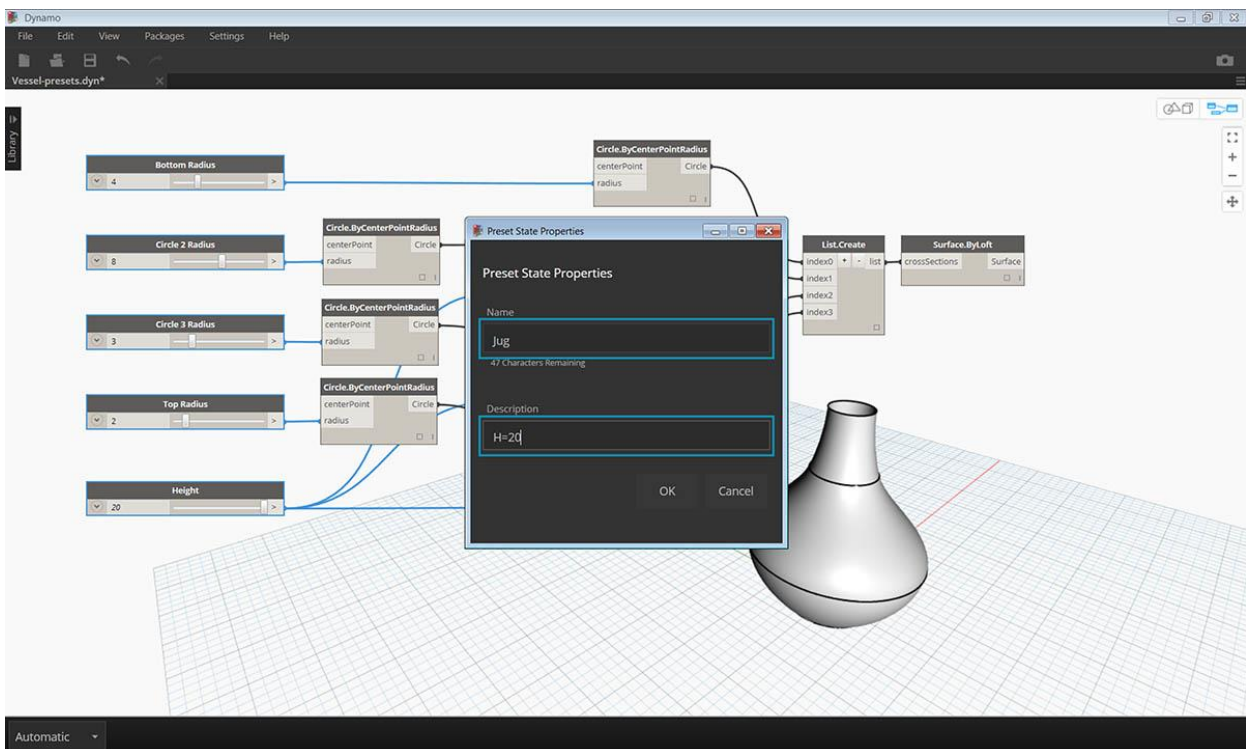
Para crear un preset, seleccione uno o más nodos de entrada. Haga clic derecho en el lienzo y seleccione "Crear preajuste de selección", o presione Control + T. Echemos un vistazo a un ejemplo. A continuación se muestra un gráfico simple que crea una superficie por medio de una serie de círculos.



1. Las entradas del gráfico son una serie de controles deslizantes que controlan la altura y los radios
- 67



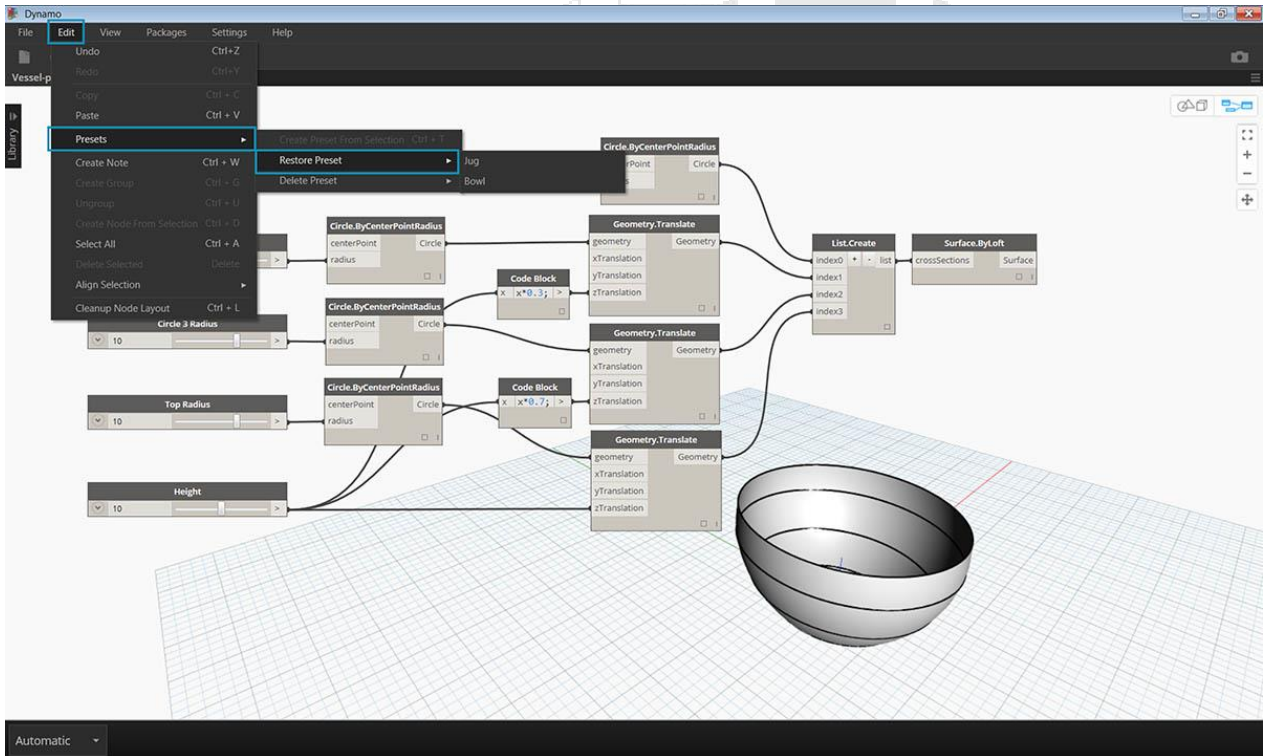
Seleccione los controles deslizantes de entrada y escriba Control + T



Ingrese un nombre y una descripción para el estado guardado en el cuadro de diálogo. Crea varios estados con diferentes valores de entrada.

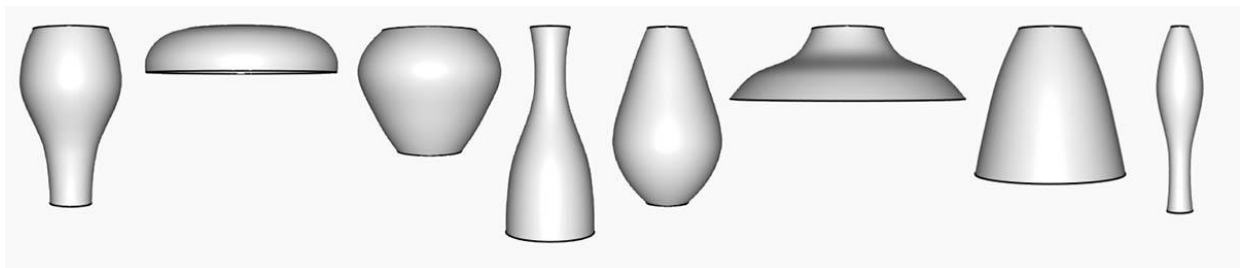
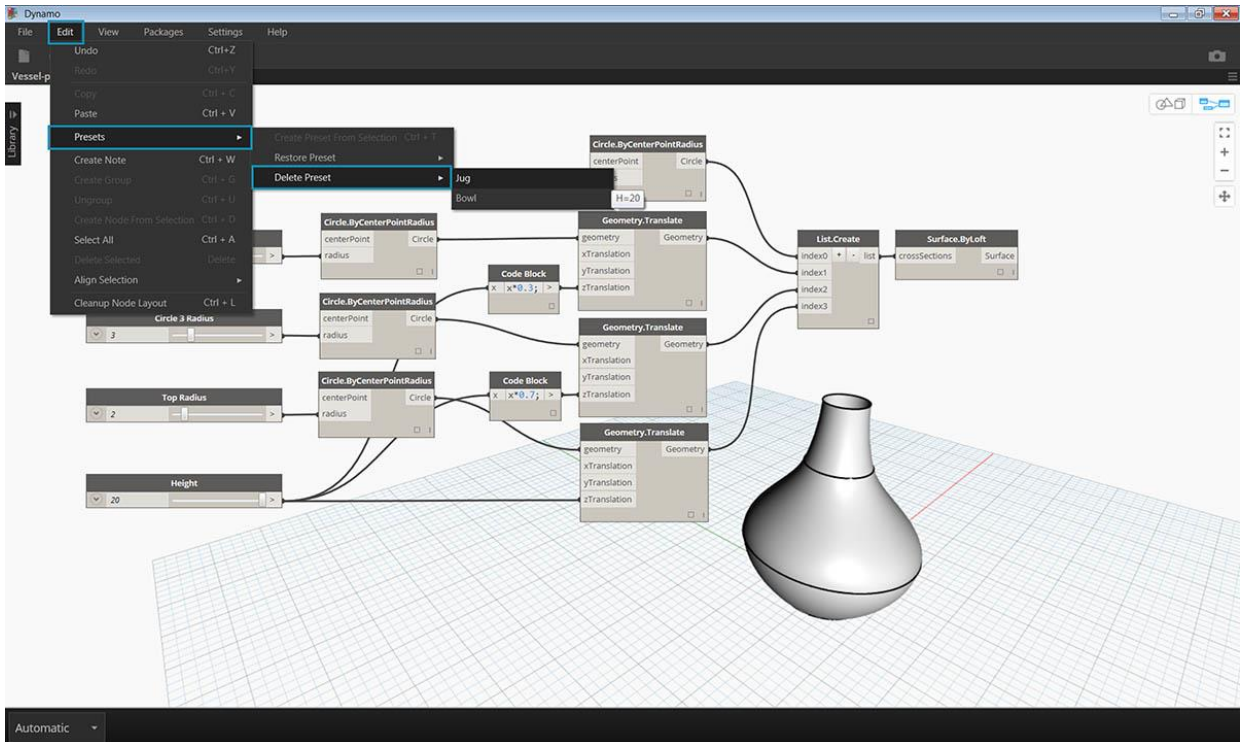
Restauración de ajustes preestablecidos

Para restablecer un ajuste preestablecido guardado, vaya a Edición > Ajustes preestablecidos > Restaurar ajuste preestablecido. Esto establecerá todos los nodos en ese estado a los valores guardados. Si un nodo en el estado ya no está presente en el gráfico (es decir, si se ha eliminado), se establecerán todos los otros nodos en el estado.



Eliminar Presets

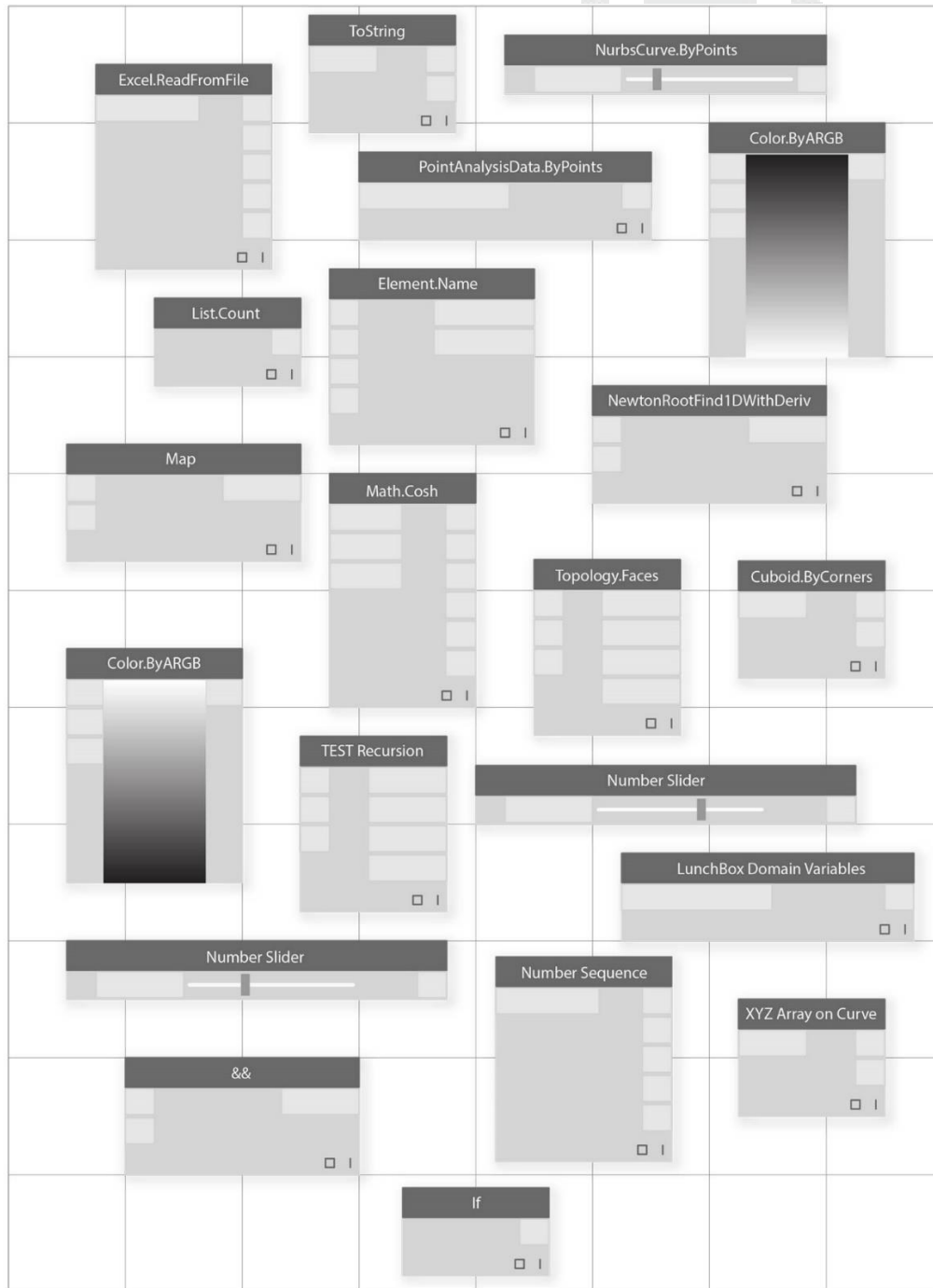
Para eliminar un ajuste preestablecido, vaya a Edición > Ajustes preestablecidos > Eliminar ajuste preestablecido. Esto eliminará un estado de la lista de estados guardados.



DARCO
DESDE 1988

LOS BLOQUES DE CONSTRUCCIÓN DE PROGRAMAS

Una vez que estemos listos para profundizar en el desarrollo de Programas Visuales, necesitaremos una comprensión más profunda de los componentes básicos que utilizaremos. Este capítulo presenta conceptos fundamentales en torno a los datos: lo que viaja a través del programa Wires of our Dynamo.

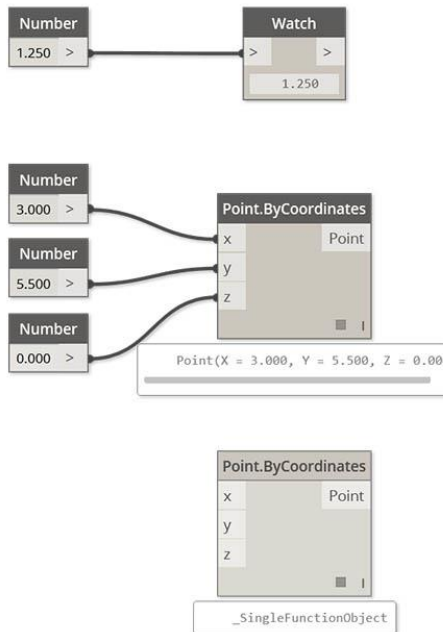


Datos

Los datos son parte de nuestros programas. Viaja a través de Wires, suministrando entradas para los nodos, donde se procesa en una nueva forma de datos de salida. Repasemos la definición de datos, cómo está estructurada y comencemos a utilizarla en Dynamo.

¿Qué es Data?


Los datos son un conjunto de valores de variables cualitativas o cuantitativas. La forma más simple de los datos es números tales como 0, 3.14o 17. Pero los datos también pueden ser de diferentes tipos: una variable que representa números cambiantes (height); personajes (myName); geometría (Circle); o una lista de elementos de datos (1,2,3,5,8, 13,...). Necesitamos datos para agregar a los Puertos de entrada de los Nodos del Dínamo: podemos tener datos sin acciones, pero necesitamos datos para procesar las acciones que representan nuestros Nodos. Cuando agregamos un Nodo al Área de trabajo, si no tiene ninguna entrada suministrada, el resultado será una función, no el resultado de la acción en sí.



1. Datos simples
2. Datos y acción (A Node) se ejecuta con éxito
3. La acción (A Node) sin entradas de datos devuelve una función genérica

Cuidado con los nulos

El 'null'tipo representa la ausencia de datos. Si bien este es un concepto abstracto, es probable que se encuentre con esto mientras trabaja con Visual Programming. Si una acción no crea un resultado válido, el nodo devolverá un valor nulo. La prueba de nulos y eliminación de nulos de la estructura de datos es una parte crucial para la creación de programas sólidos.

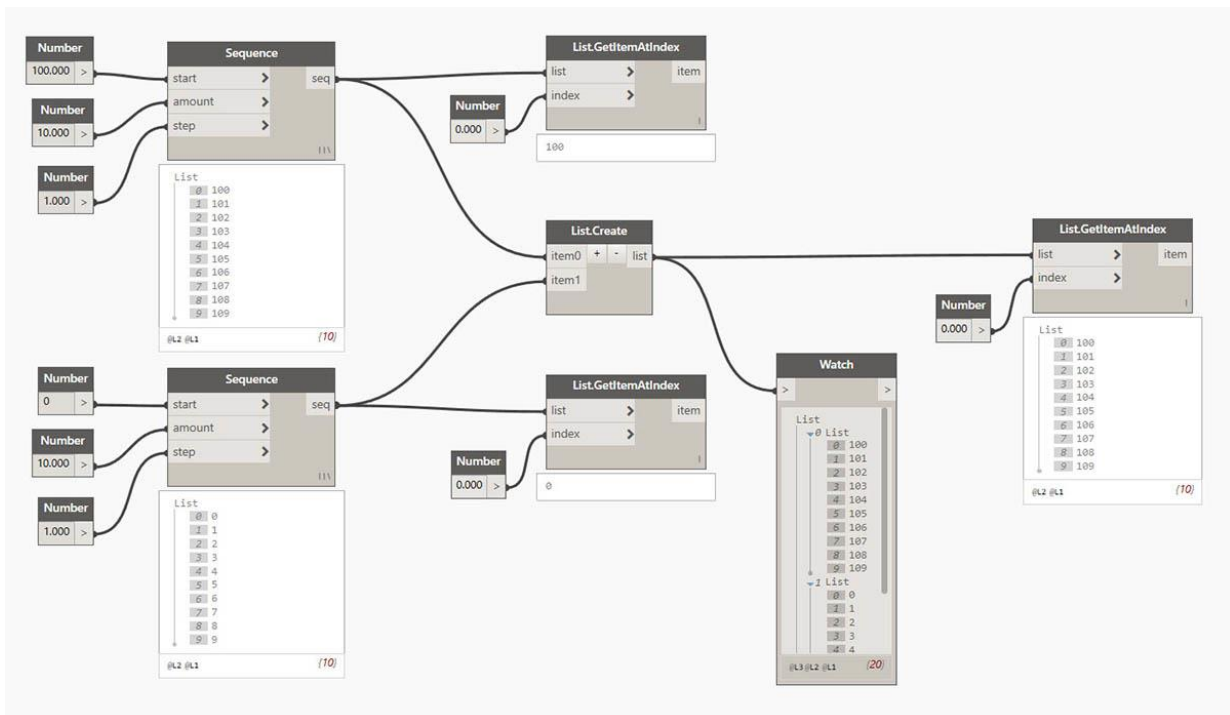
Icono	Nombre / Sintaxis	Entradas	Salidas
	Object.IsNotNull	obj	bool

Estructuras de datos

Cuando somos Programación Visual, podemos generar muy rápidamente una gran cantidad de datos y requerir un medio para administrar su jerarquía. Este es el papel de las estructuras de datos, los esquemas organizacionales en los que almacenamos los datos. Los detalles de las estructuras de datos y cómo usarlos varían del lenguaje de programación al lenguaje de programación. En Dynamo, agregamos jerarquía a nuestros datos a través de listas. Exploraremos esto en profundidad en capítulos posteriores, pero comencemos simplemente:

Una lista representa una colección de elementos colocados en una estructura de datos:

- Tengo cinco dedos (elementos) en mi mano (lista).
- Hay diez casas (artículos) en mi calle (lista).



1. Un nodo de **secuencia numérica** define una lista de números usando un *inicio*, *cantidad* y entrada de *paso*. Con estos nodos, hemos creado dos listas separadas de diez números, uno que va de 100 a 109 y otro que va de 0 a 9.
2. El nodo **List.GetItemAtIndex** selecciona un elemento en una lista en un índice específico. Al elegir 0, obtenemos el primer elemento de la lista (100 en este caso).

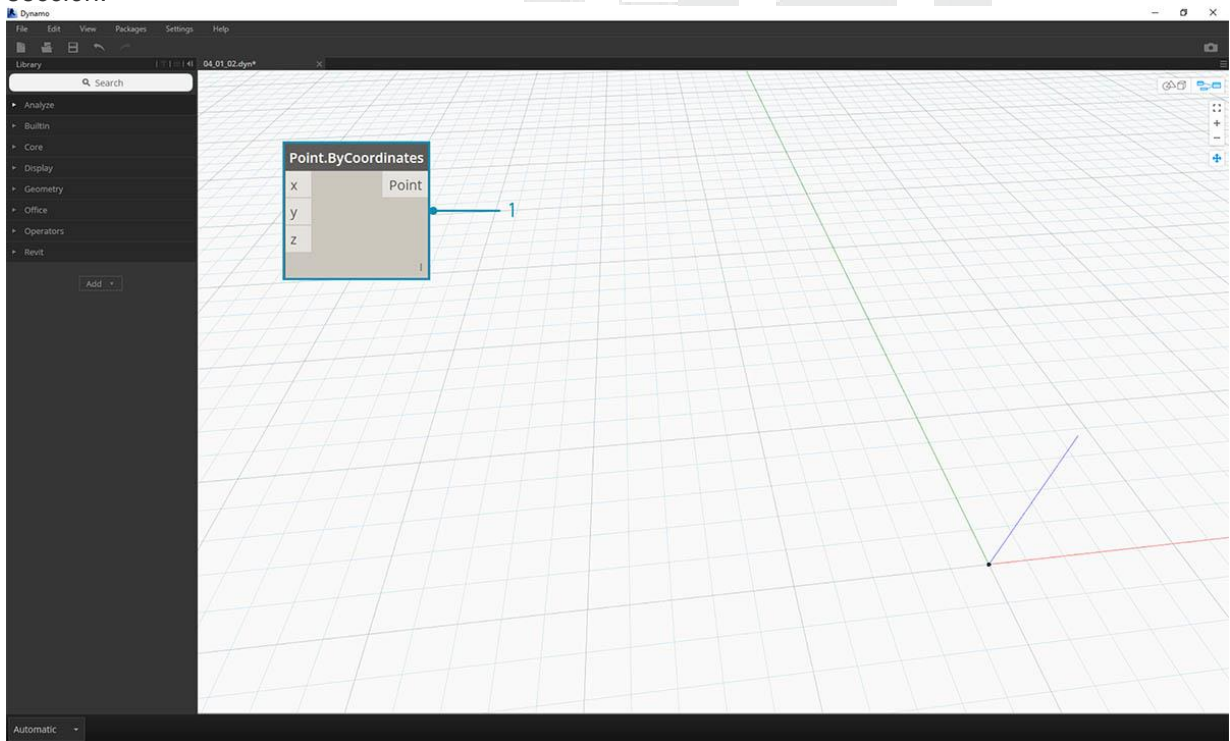
3. Aplicando el mismo proceso a la segunda lista, obtenemos un valor de 0, el primer elemento de la lista.
4. Ahora **fusionamos** las dos listas en una utilizando el nodo **List.Create**. Observe que el nodo crea una *lista de listas*. Esto cambia la estructura de los datos.
5. Al usar **List.GetItemAtIndex** nuevamente, con el índice establecido en 0, obtenemos la primera lista en la lista de listas. Esto es lo que significa tratar una lista como un elemento, que es algo diferente de otros lenguajes de scripting. Nos volveremos más avanzados con la manipulación de listas y la estructura de datos en capítulos posteriores.

El concepto clave para entender sobre la jerarquía de datos en Dynamo: **con respecto a la estructura de datos, las listas se consideran como elementos**. En otras palabras, Dynamo funciona con un proceso de arriba hacia abajo para comprender las estructuras de datos. ¿Qué significa esto? Veámoslo con un ejemplo.

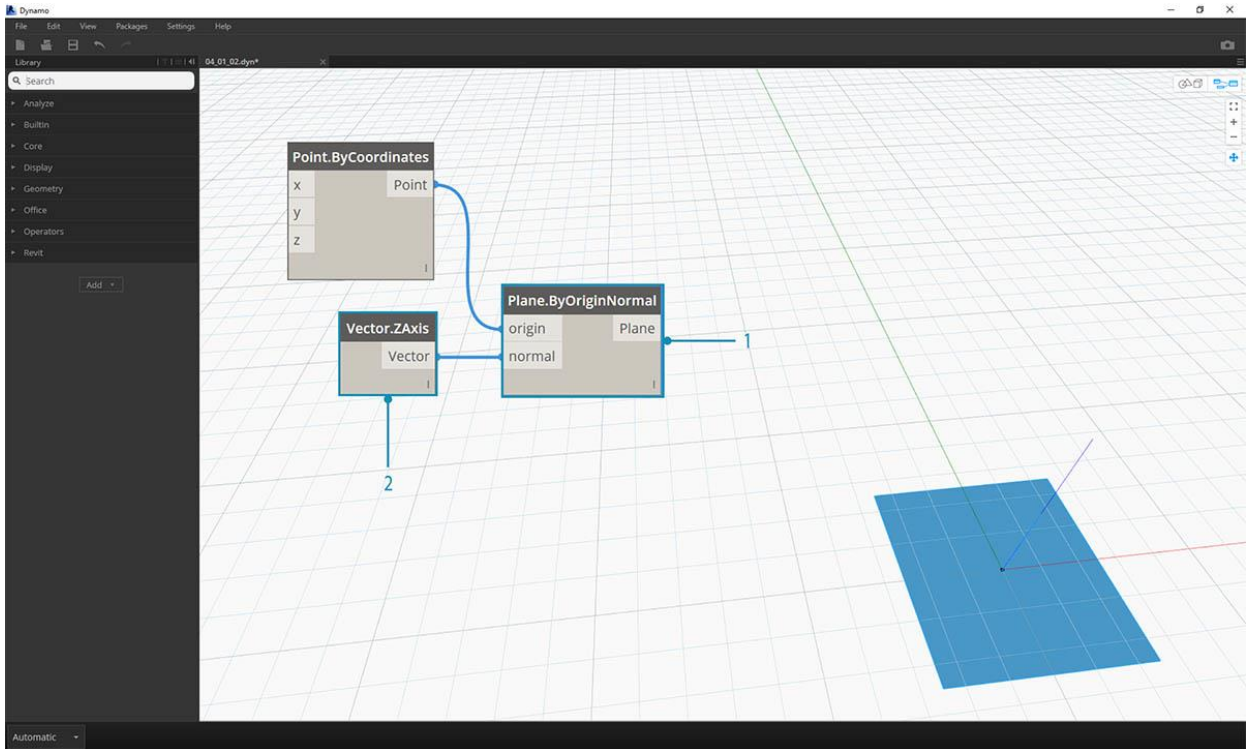
Usar datos para hacer una cadena de cilindros

Descargue el archivo de ejemplo que acompaña a este ejercicio Building Blocks of Programs - Data.dyn . Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

En este primer ejemplo, armamos un cilindro sin cáscara que recorre la jerarquía de geometría que se analiza en esta sección.

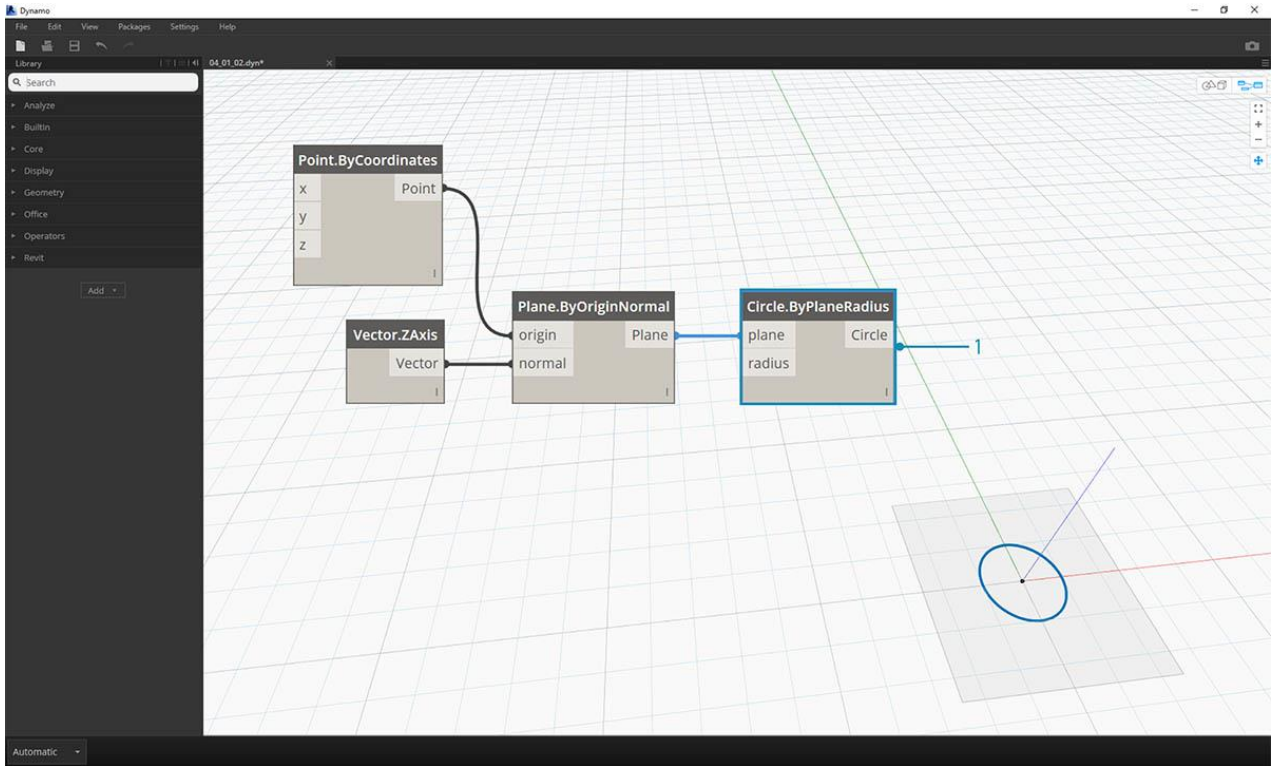


1. **Point.ByCoordinates:** después de agregar el nodo al lienzo, vemos un punto en el origen de la cuadrícula de vista previa de Dynamo. Los valores predeterminados de las entradas x, y y z son 0.0, lo que nos da un punto en esta ubicación.

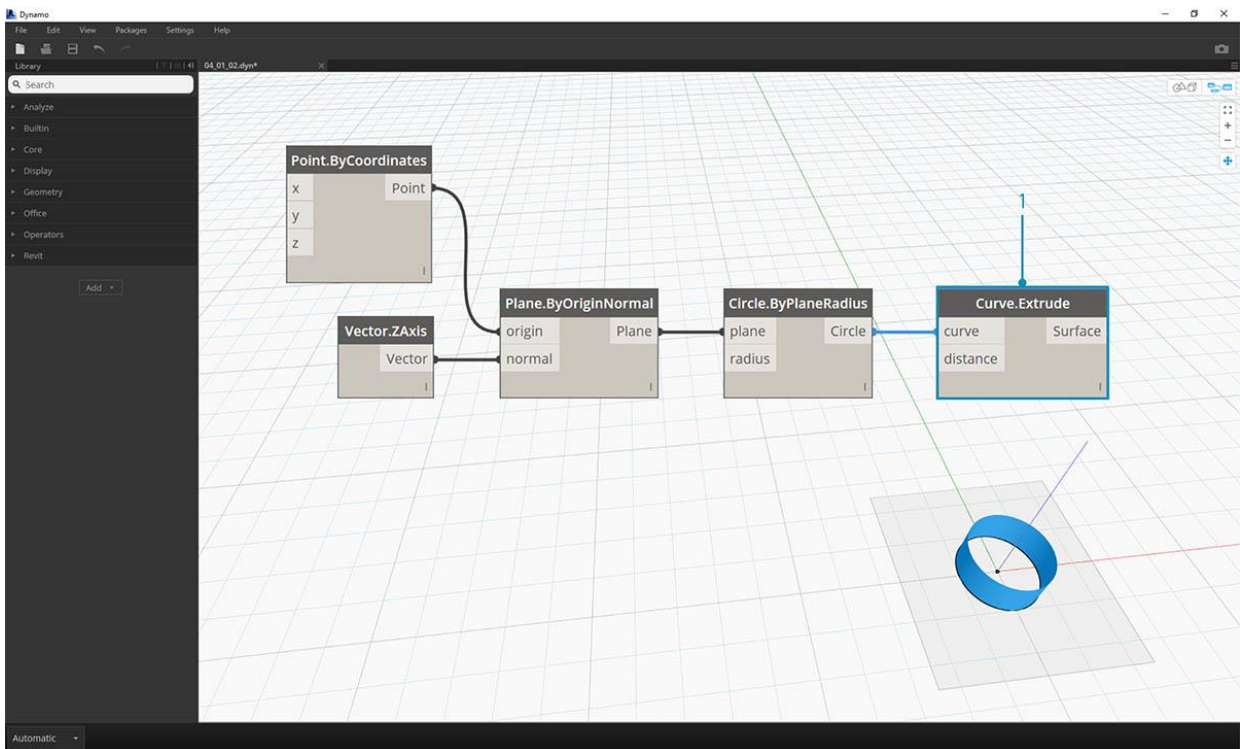


1. **Plane.ByOriginNormal:** el siguiente paso en la jerarquía de geometría es un plano. Hay varias formas de construir un plano, y estamos usando uno que requiere un origen y normal para la entrada. El origen es el nodo de punto creado en el paso anterior.
2. **Vector.ZAxis** - este es un vector unificado en la dirección z. Tenga en cuenta que no hay entradas, solo un vector de [0,0,1] valor. Usamos esto como la entrada *normal* para el nodo *Plane.ByOriginNormal* . Esto nos da un plano rectangular en la vista previa de Dynamo.

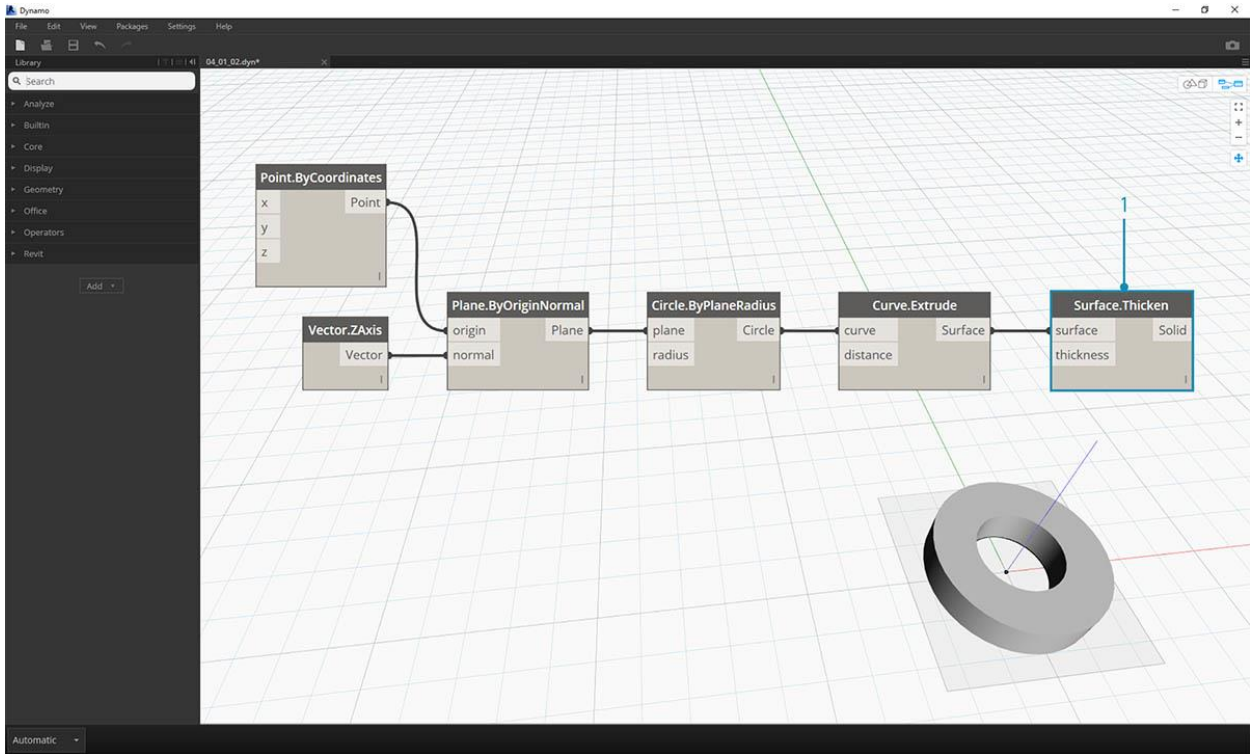
DARCO
DESDE 1988



1. **Circle.ByPlaneRadius:** al subir la jerarquía, ahora creamos una curva desde el plano en nuestro paso anterior. Después de conectarnos al nodo, obtenemos un círculo en el origen. El radio predeterminado en el nodo es el valor de 1.

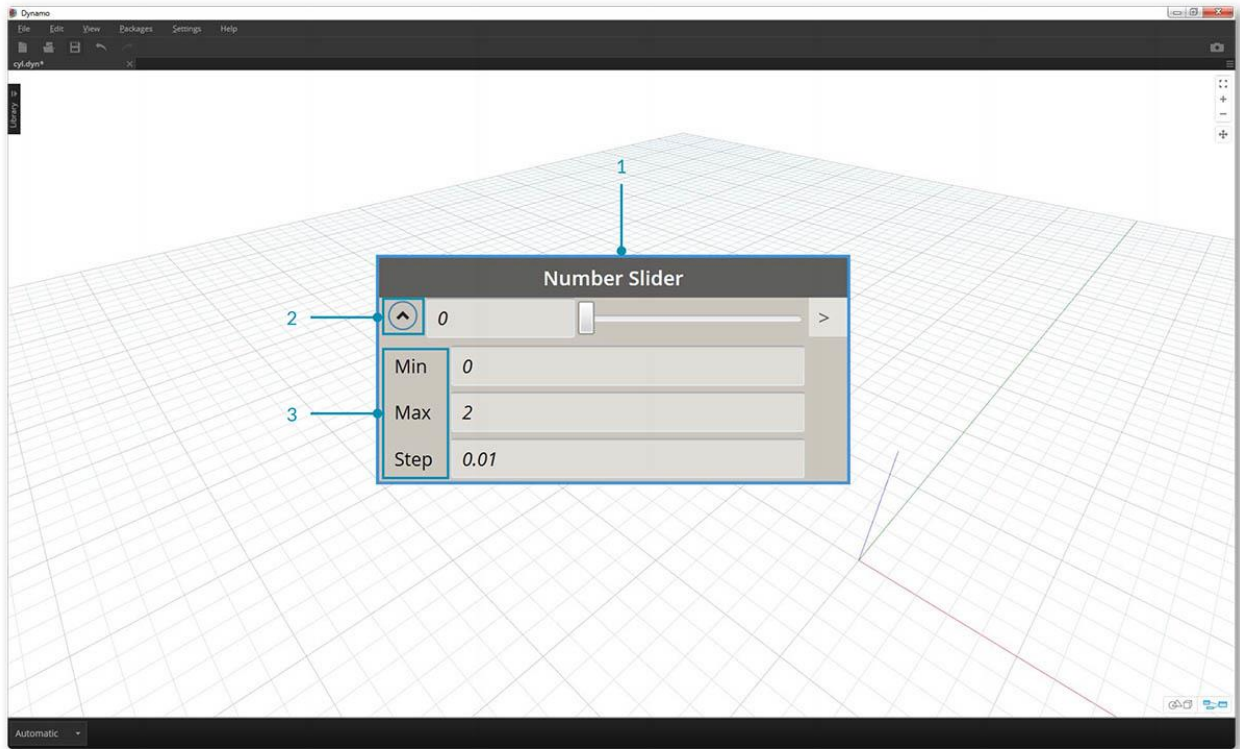


1. **Curve.Extrude** - Ahora hacemos que esta cosa se **destaque** dándole algo de profundidad y entrando en la tercera dimensión. Este nodo crea una superficie a partir de una curva extruyéndola. La distancia predeterminada en el nodo es 1 , y deberíamos ver un cilindro en la ventana gráfica.



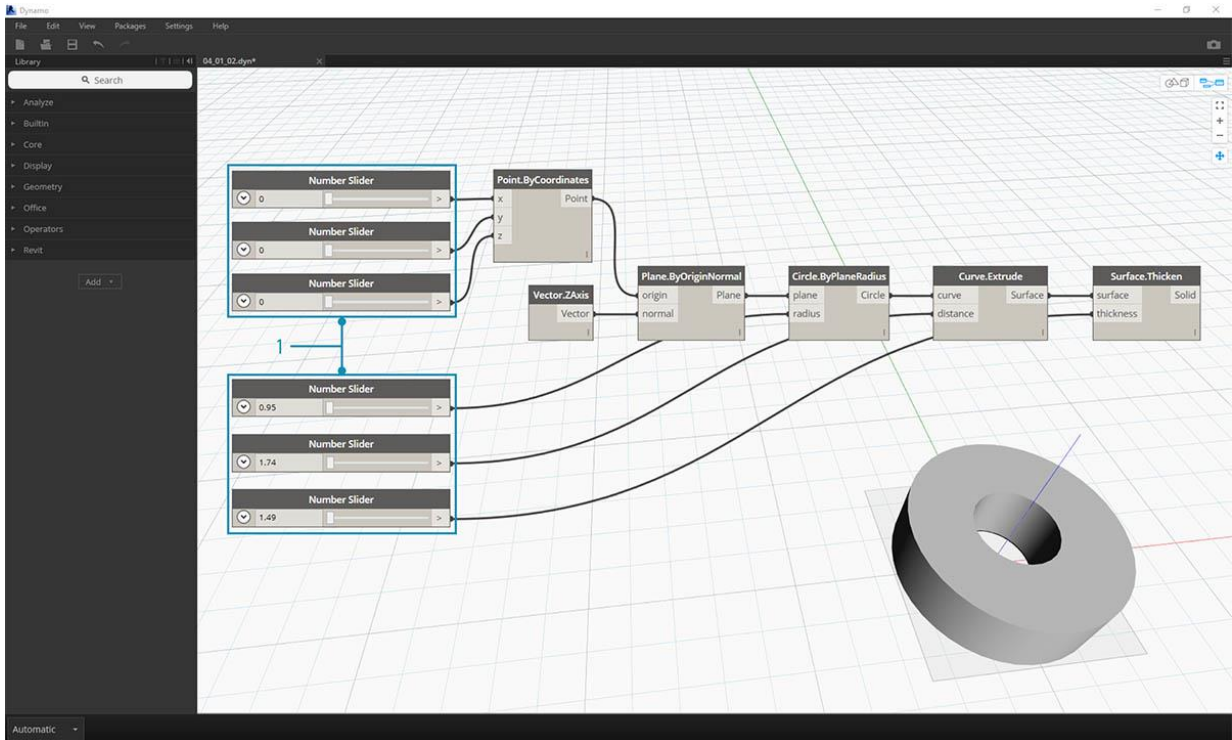
1. **Surface.Thicken** - Este nodo nos da un sólido cerrado al compensar la superficie de una distancia dada y al cerrar la forma. El valor por defecto del espesor es 1, y vemos un cilindro pelado en la ventana gráfica en línea con estos valores.

DARCO
DESDE 1988



1. **Control deslizante numérico:** en lugar de utilizar los valores predeterminados para todas estas entradas, agreguemos algún control paramétrico al modelo.
2. **Editar dominio:** después de agregar el deslizador numérico al lienzo, haga clic en el símbolo de intercalación en la parte superior izquierda para ver las opciones de dominio.
3. **Mín.** / **Máx.** / **Paso:** cambie los valores *mínimos*, *máximo* y de *paso* a 0, 2 y 0.01, respectivamente. Estamos haciendo esto para controlar el tamaño de la geometría general.

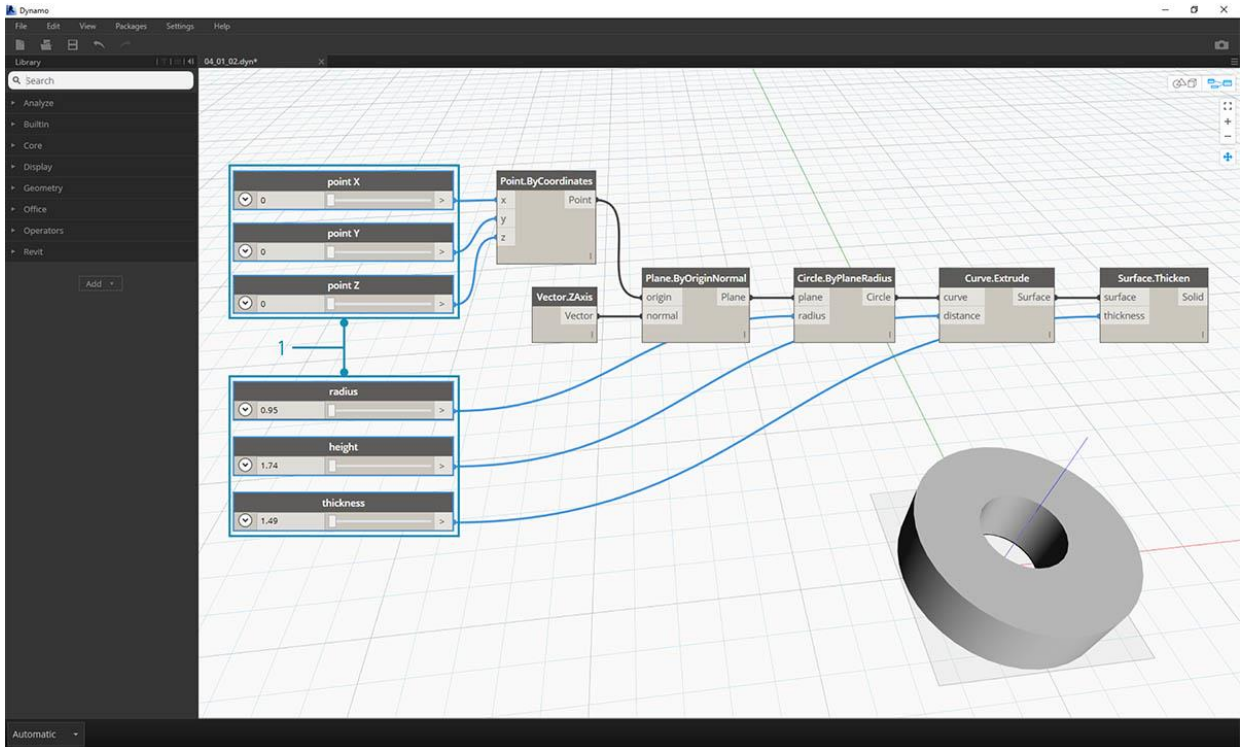
DARCO
DESDE 1988



1. **Número de controles deslizantes:** en todas las entradas predeterminadas, copiemos y peguemos este deslizador numérico (seleccione el control deslizante, pulse Ctrl + C, luego Ctrl + V) varias veces, hasta que todas las entradas con valores predeterminados tengan un control deslizante. Algunos de los valores del control deslizante tendrán que ser mayores que cero para que la definición funcione (es decir, necesita una profundidad de extrusión para que una superficie se espese).

Ahora hemos creado un cilindro paralizado paramétrico con estos controles deslizantes. Intente flexionar algunos de estos parámetros y vea la actualización de la geometría de forma dinámica en la ventana Dynamo.

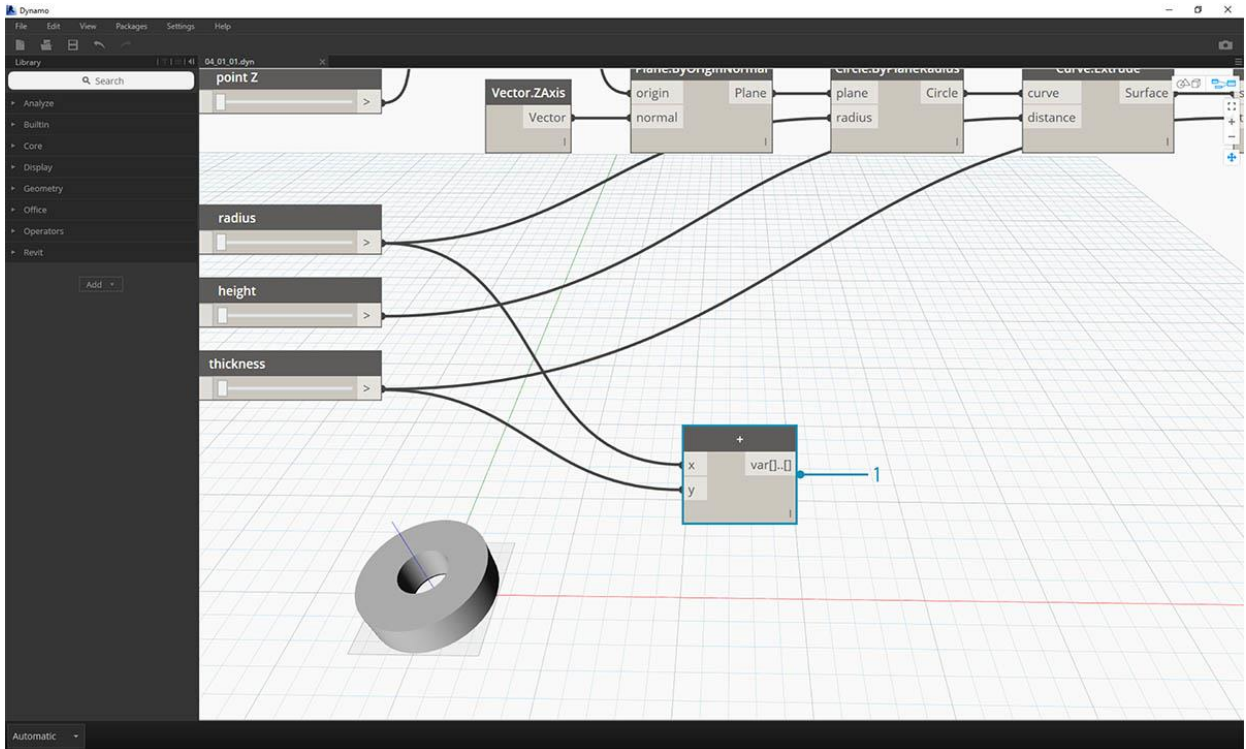
DARCO
DESDE 1988



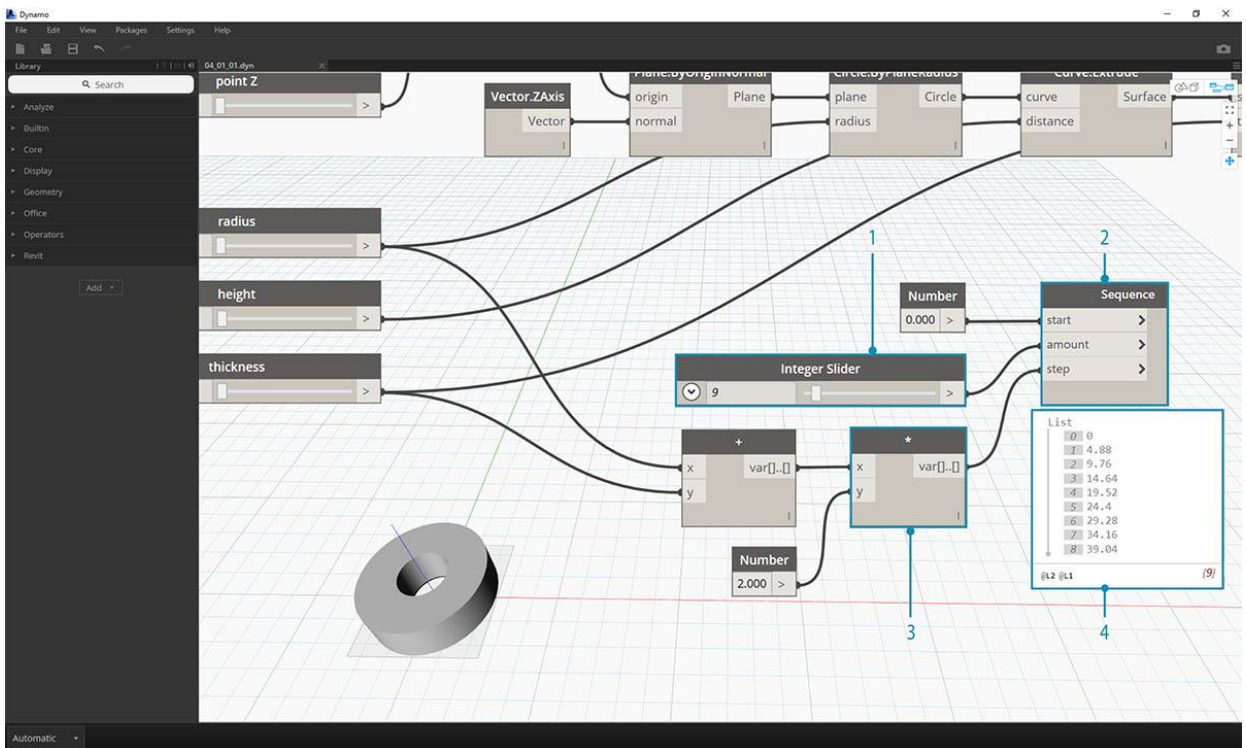
1. **Número de controles deslizantes: yendo** un paso más allá, hemos agregado muchos controles deslizantes al lienzo y necesitamos limpiar la interfaz de la herramienta que acabamos de crear. Haga clic con el botón derecho en un control deslizante, seleccione "Cambiar nombre ..." y cambie cada control deslizante al nombre apropiado para su parámetro. Puede hacer referencia a la imagen de arriba para los nombres.

En este punto, hemos creado un impresionante cilindro de engrosamiento. Este es un objeto en la actualidad, veamos cómo crear una matriz de cilindros que permanezca vinculada dinámicamente. Para hacer esto, vamos a crear una lista de cilindros, en lugar de trabajar con un solo elemento.

DARCO
DESDE 1988

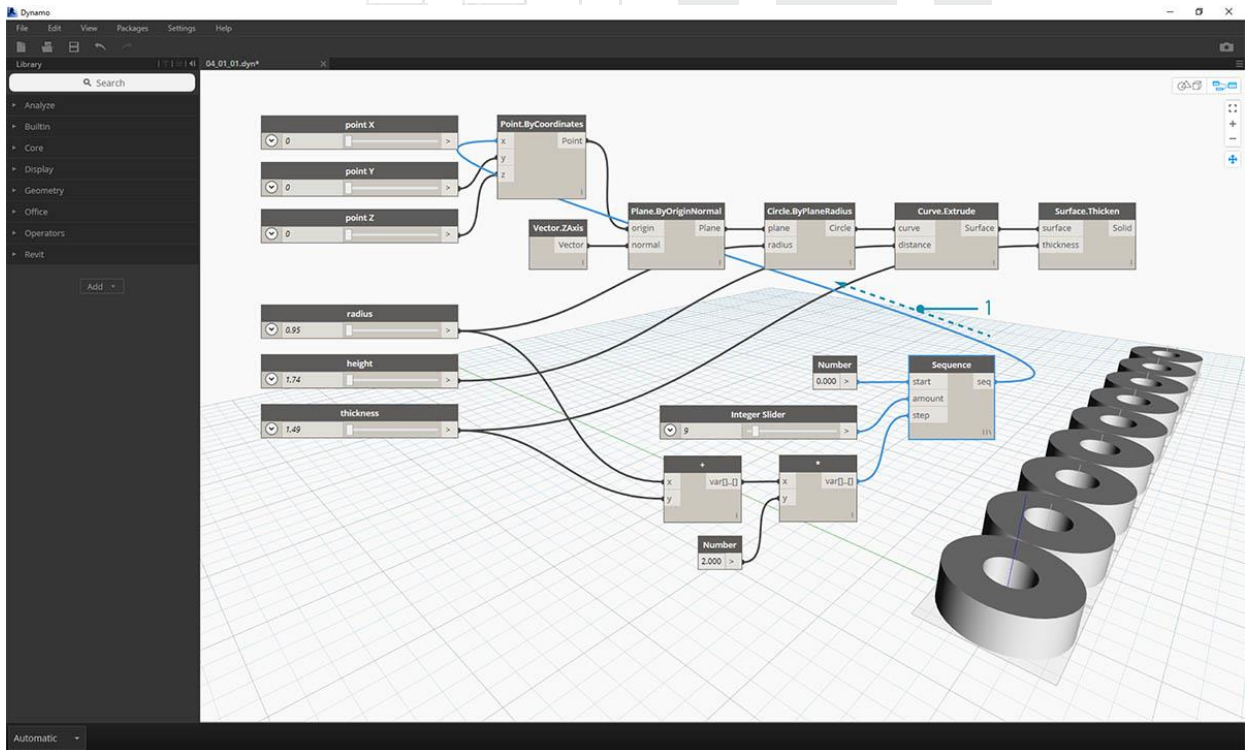


1. **Adición (+)** - Nuestro objetivo es agregar una fila de cilindros al lado del cilindro que hemos creado. Si queremos agregar un cilindro adyacente al actual, debemos considerar tanto el radio del cilindro como el grosor de su caparazón. Obtenemos este número agregando los dos valores de los controles deslizantes.

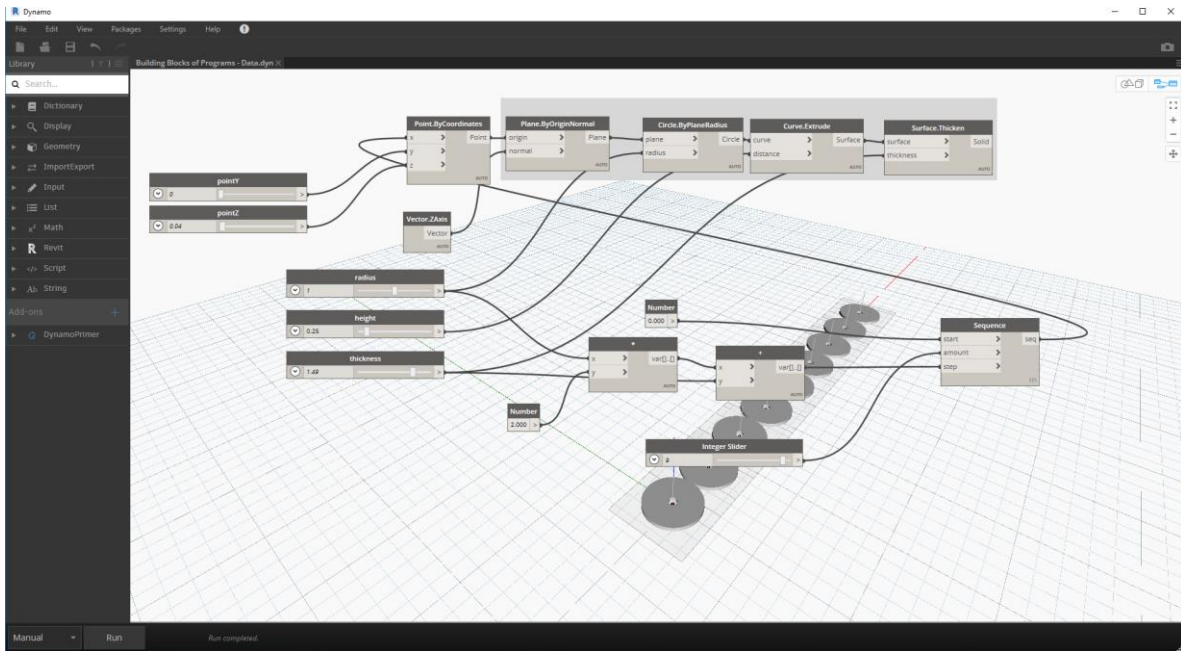


Este paso es más complicado, así que caminemos lentamente: el objetivo final es crear una lista de números que defina las ubicaciones de cada cilindro en una fila.

1. **Multiplicación:** primero, queremos multiplicar el valor del paso anterior por 2. El valor del paso anterior representa un radio, y queremos mover el cilindro con el diámetro completo.
2. **Secuencia numérica:** creamos una matriz de números con este nodo. La primera entrada es el nodo de *multiplicación* desde el paso anterior al valor de *paso*. El valor *inicial* puede establecerse en *0.0* usando un nodo *numérico*.
3. **Control deslizante de enteros:** para el valor de la *cantidad*, conectamos un control deslizante de enteros. Esto definirá cuántos cilindros se crean.
4. **Salida** - Esta lista nos muestra la distancia movida para cada cilindro en la matriz, y es controlada paramétricamente por los controles deslizantes originales.



1. Este paso es bastante simple: conecte la secuencia definida en el paso anterior en la entrada x del punto original . *ByCoordinates* . Esto reemplazará el control deslizante *pointX* que podemos eliminar. Ahora vemos una serie de cilindros en la ventana gráfica (asegúrese de que el control deslizante entero sea mayor que 0).







La cadena de cilindros todavía está vinculada dinámicamente a todos los controles deslizantes. ¡Flexiona cada deslizador para ver la actualización de la definición!

Matemáticas

Si la forma más simple de datos son los números, la forma más fácil de relacionar esos números es a través de las Matemáticas. Desde operadores simples como dividir a funciones trigonométricas, hasta fórmulas más complejas, las matemáticas son una excelente manera de comenzar a explorar patrones y relaciones numéricas.

Operadores aritméticos

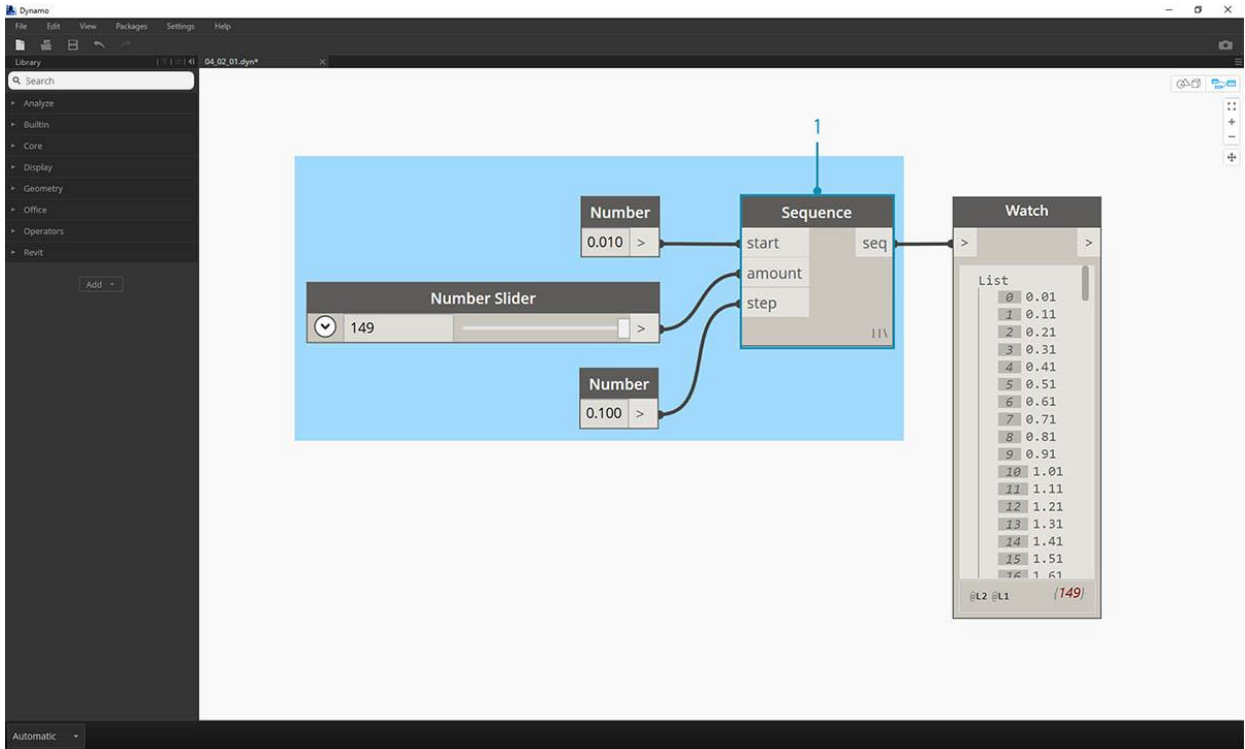
Los operadores son un conjunto de componentes que utilizan funciones algebraicas con dos valores de entrada numéricos, que dan como resultado un valor de salida (suma, resta, multiplicación, división, etc.). Estos se pueden encontrar en Operadores > Acciones.

Icono	Nombre	Sintaxis	Entradas	Salidas
	Añadir	+	var [] ... [], var [] ... []	var [] ... []
	Sustraer	-	var [] ... [], var [] ... []	var [] ... []
	Multiplicar	*	var [] ... [], var [] ... []	var [] ... []
	Dividir	/	var [] ... [], var [] ... []	var [] ... []

Fórmula paramétrica

Descargue el archivo de ejemplo que acompaña este Building Blocks of Programs - Math.dyn . Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

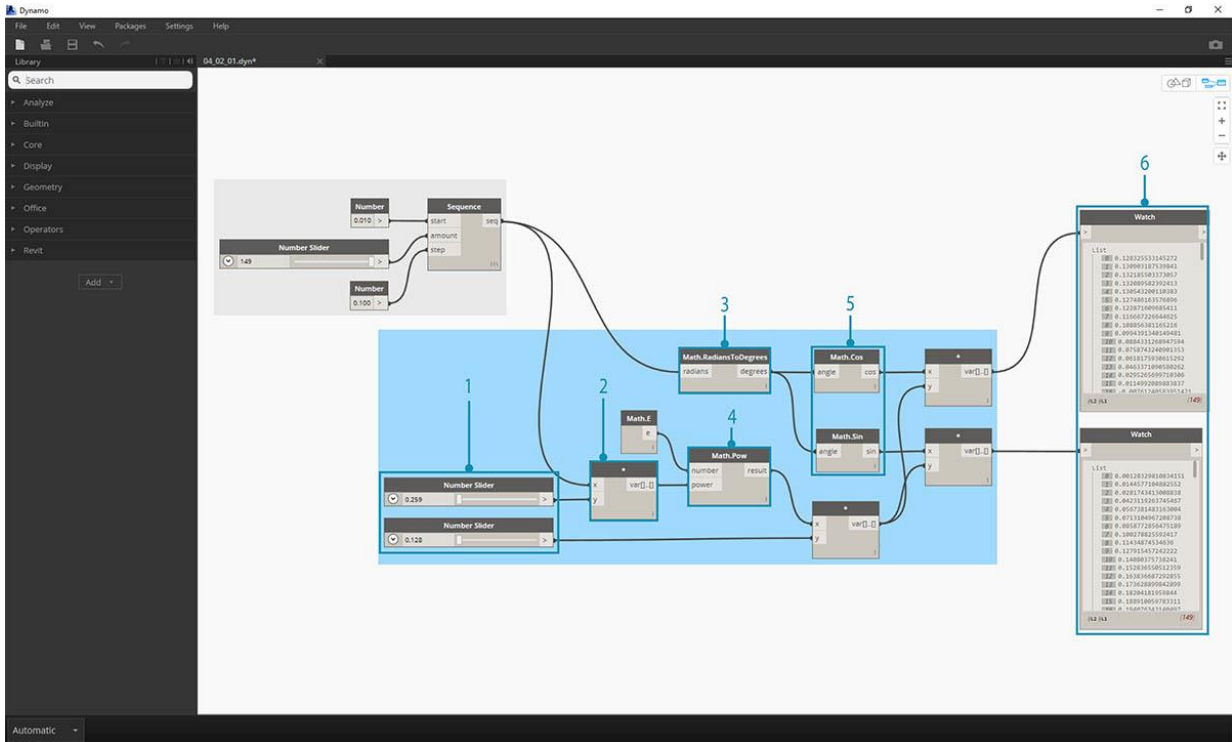
Desde Operadores, el siguiente paso lógico es combinar operadores y variables para formar una relación más compleja a través de **fórmulas**. Hagamos una fórmula que pueda controlarse mediante parámetros de entrada, como controles deslizantes.



1. **Secuencia numérica:** define una secuencia de números basada en tres entradas: *inicio*, *cantidad* y *paso*. Esta secuencia representa la 't' en la ecuación paramétrica, por lo que queremos usar una lista que sea lo suficientemente grande como para definir una espiral.

El paso anterior ha creado una lista de números para definir el dominio paramétrico. La espiral dorada se define como la ecuación: $x = r \cos \theta = a \cos \theta e^{b\theta}$ y $y = r \sin \theta = a \sin \theta e^{b\theta}$. El grupo de nodos a continuación representa esta ecuación en forma de programación visual.

DARCO
DESDE 1988



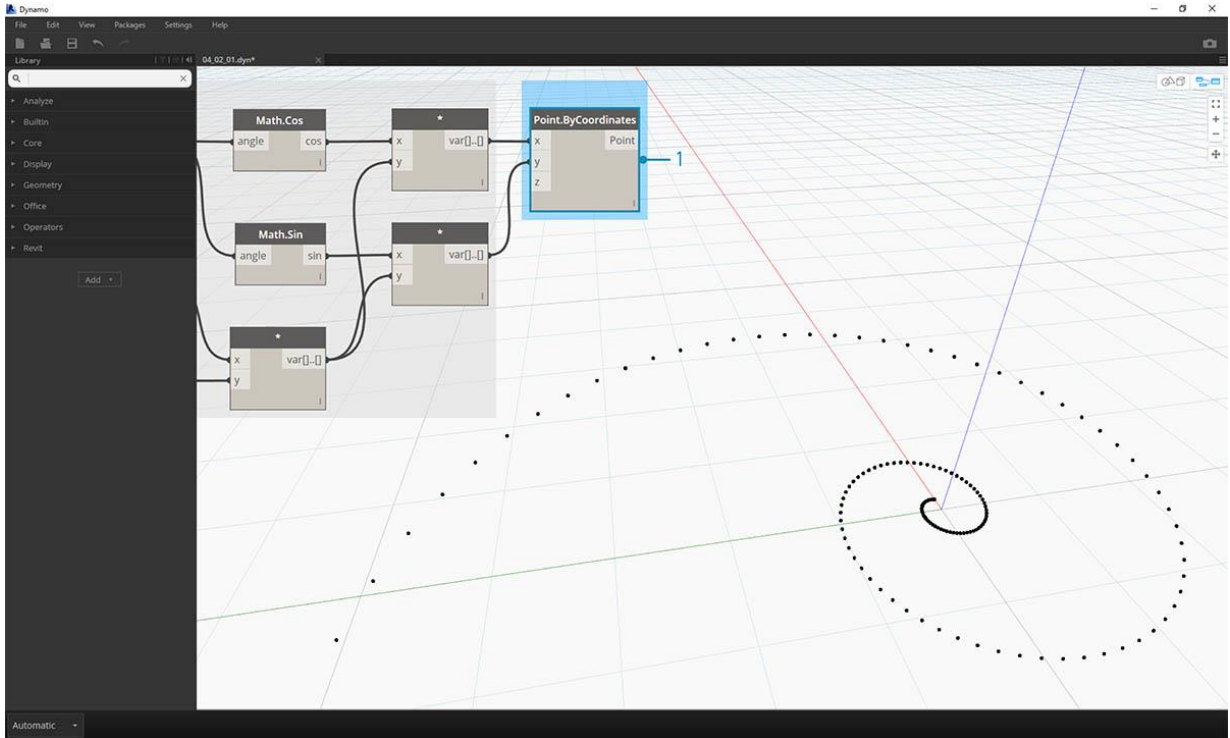
Al recorrer el grupo de Nodos, intente prestar atención al paralelo entre el programa visual y la ecuación escrita.

1. **Control deslizante numérico:** agregue dos controles deslizantes numéricos al lienzo. Estos deslizadores representarán a la *una* y las *b* variables de la ecuación paramétrica. Estos representan una constante que es flexible, o parámetros que podemos ajustar hacia un resultado deseado.
2. *: El nodo de multiplicación está representado por un asterisco. Lo usaremos repetidamente para conectar variables multiplicadoras
3. **Math.RadiansToDegrees:** los valores ' *t* ' deben traducirse a grados para su evaluación en las funciones trigonométricas. Recuerde, Dynamo se predetermina en grados para evaluar estas funciones.
4. **Math.Pow:** como una función de la ' *t* ' y el número ' *e* ' esto crea la secuencia de Fibonacci.
5. **Math.Cos y Math.Sin:** estas dos funciones trigonométricas diferenciarán la coordenada *xy* y la coordenada *y*, respectivamente, de cada punto paramétrico.
6. **Watch:** Ahora vemos que nuestra producción es de dos listas, éstas serán las *x* y *Y* coordenadas de los puntos utilizados para generar la espiral.

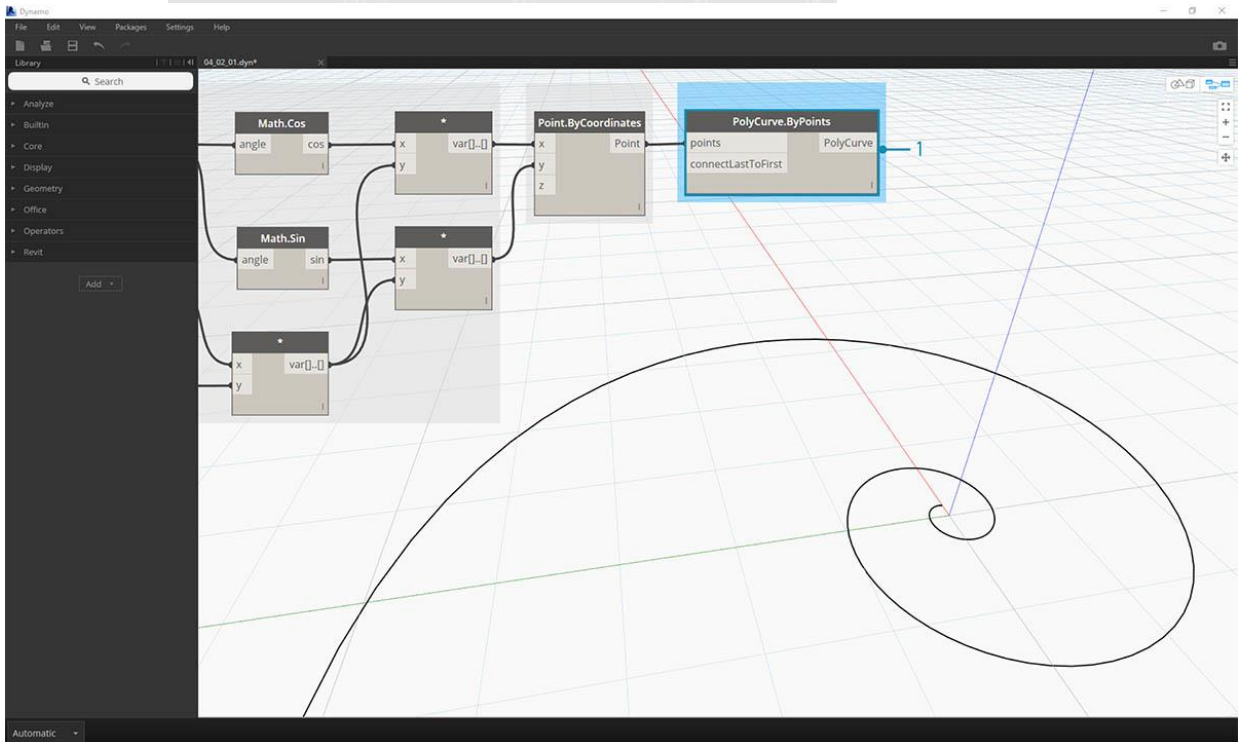
De la fórmula a la geometría

Ahora, la mayor parte de los nodos del paso anterior funcionará bien, pero es mucho trabajo. Para crear un flujo de trabajo más eficiente, eche un vistazo a los **Bloques de Código** (sección 3.3.2.3) para definir una cadena de expresiones de Dynamo en un nodo. En esta próxima serie de pasos, veremos cómo usar la ecuación paramétrica para dibujar la espiral de

Fibonacci.



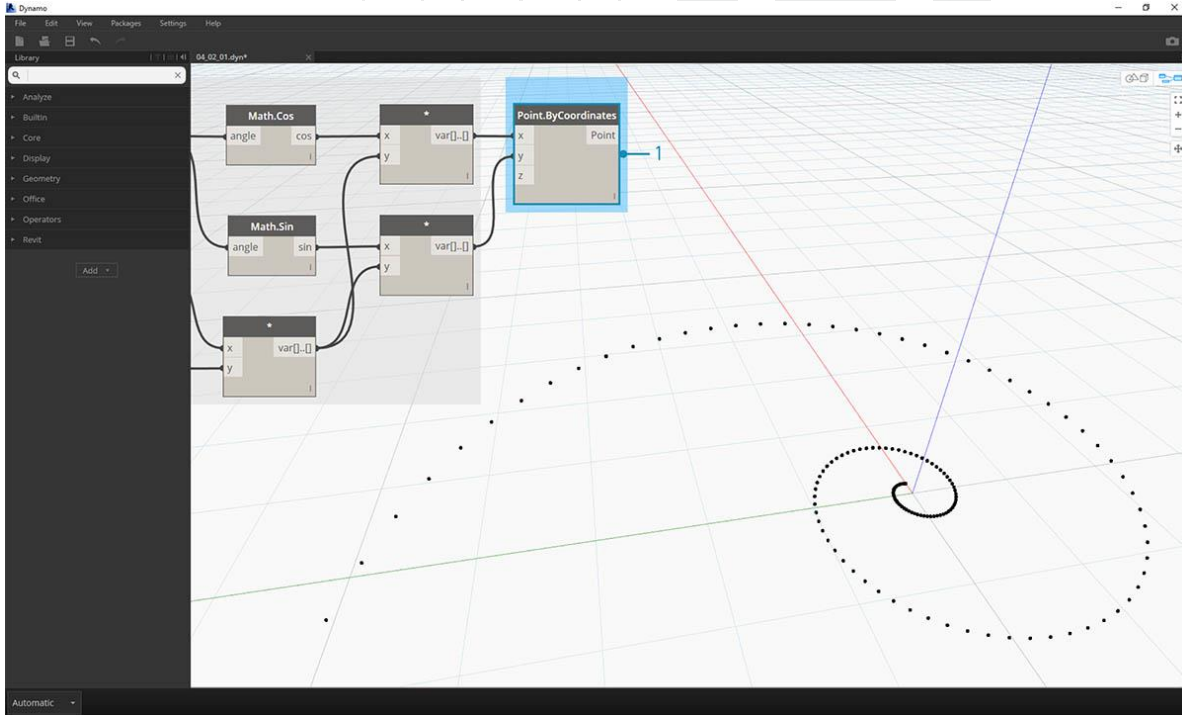
1. **Point.ByCoordinates:** conecta el nodo de multiplicación superior a la entrada ' x ' y el inferior a la entrada ' y '. Ahora vemos una espiral paramétrica de puntos en la pantalla.



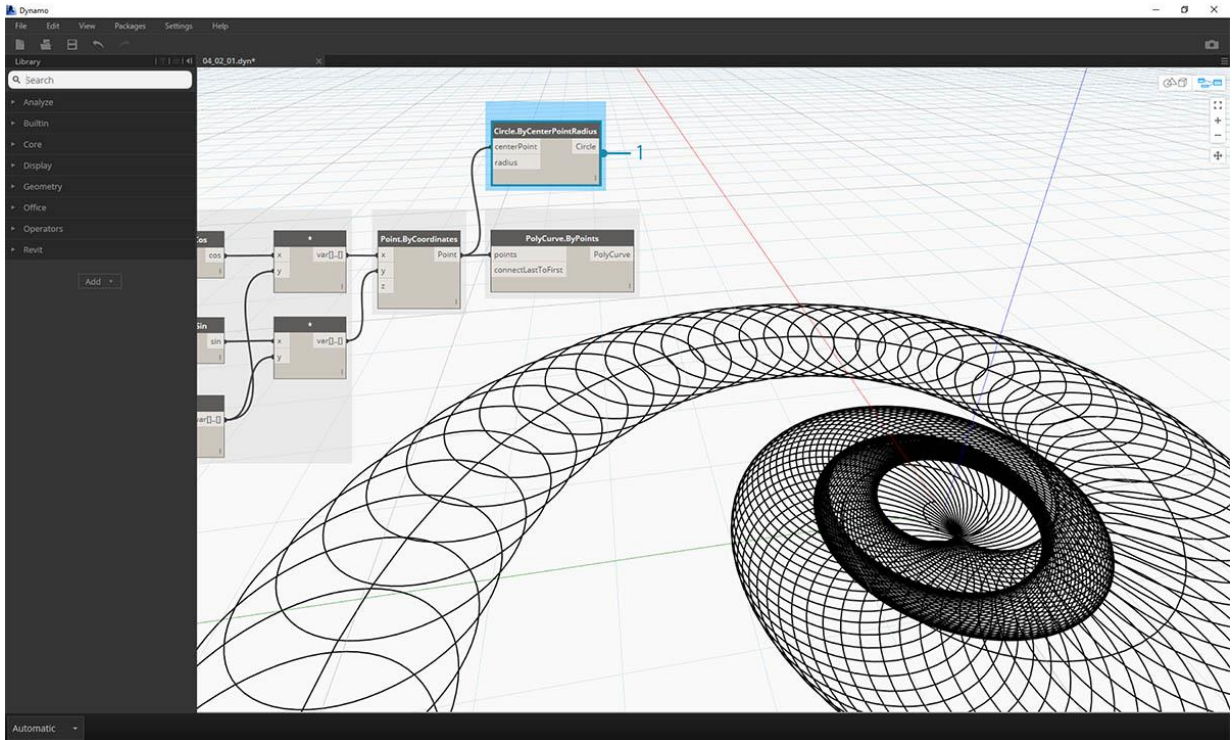
1. **Polycurve.ByPoints:** Connect **Point.ByCoordinates** del paso anterior en *puntos*. Podemos dejar *connectLastToFirst* sin una entrada porque no estamos haciendo una curva cerrada. Esto crea una espiral que pasa por cada punto definido en el paso anterior.

¡Hemos completado la Espiral de Fibonacci! Llevemos esto más allá en dos ejercicios separados desde aquí, que llamaremos el Nautilus y el Girasol. Estas son abstracciones de los sistemas naturales, pero las dos aplicaciones diferentes de la espiral de Fibonacci estarán bien representadas.

De Espiral a Nautilus

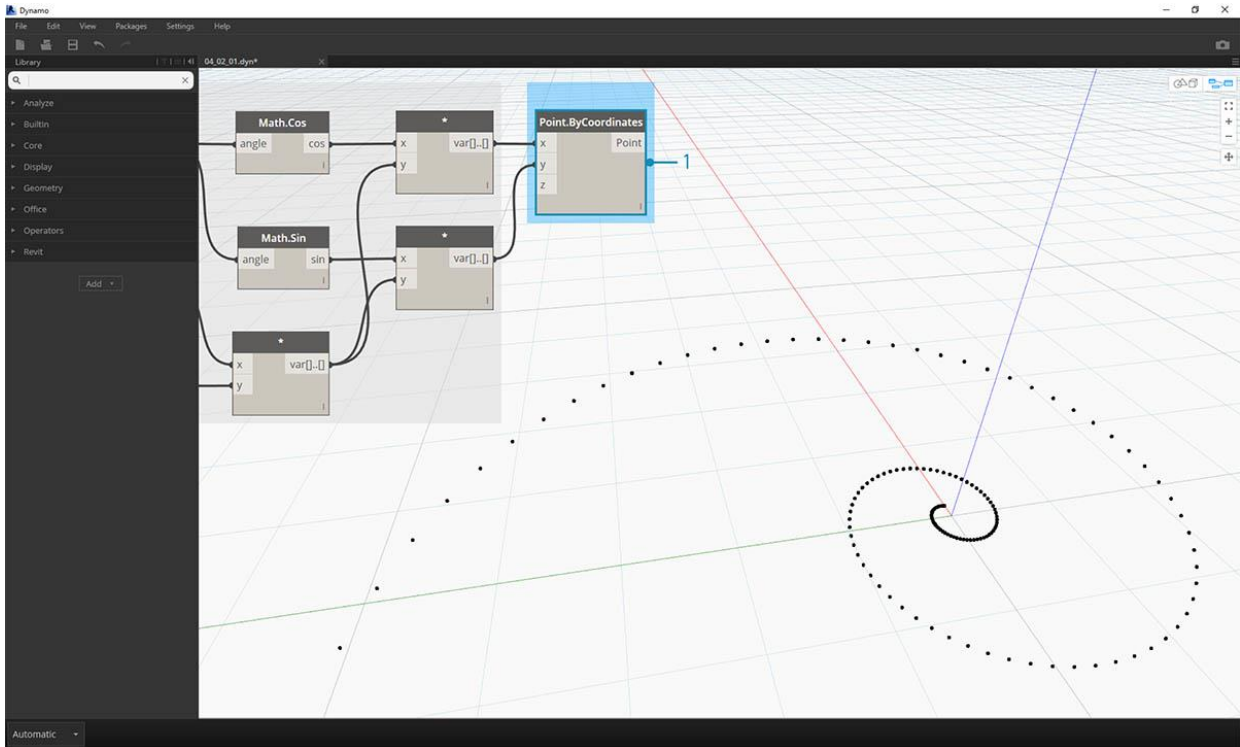


1. Como punto de partida, comencemos con el mismo paso del ejercicio anterior: crear una matriz en espiral de puntos con el nodo **Point.ByCoordinates**.

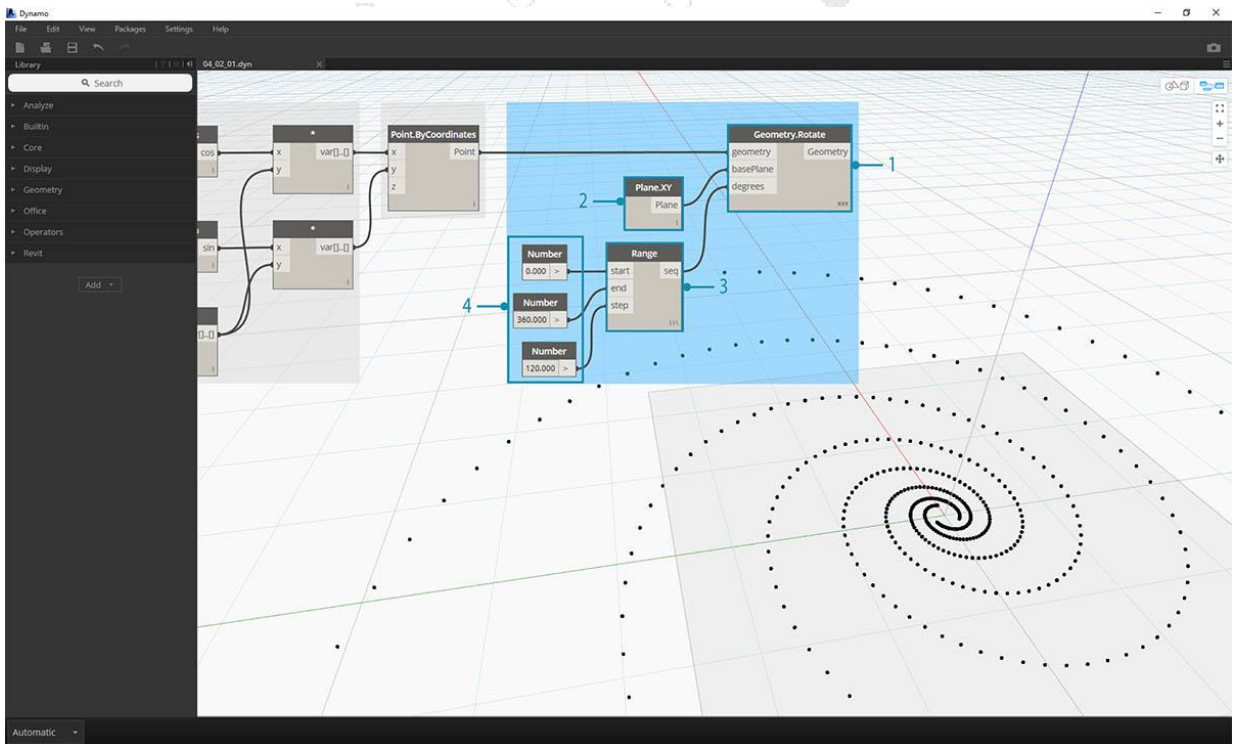


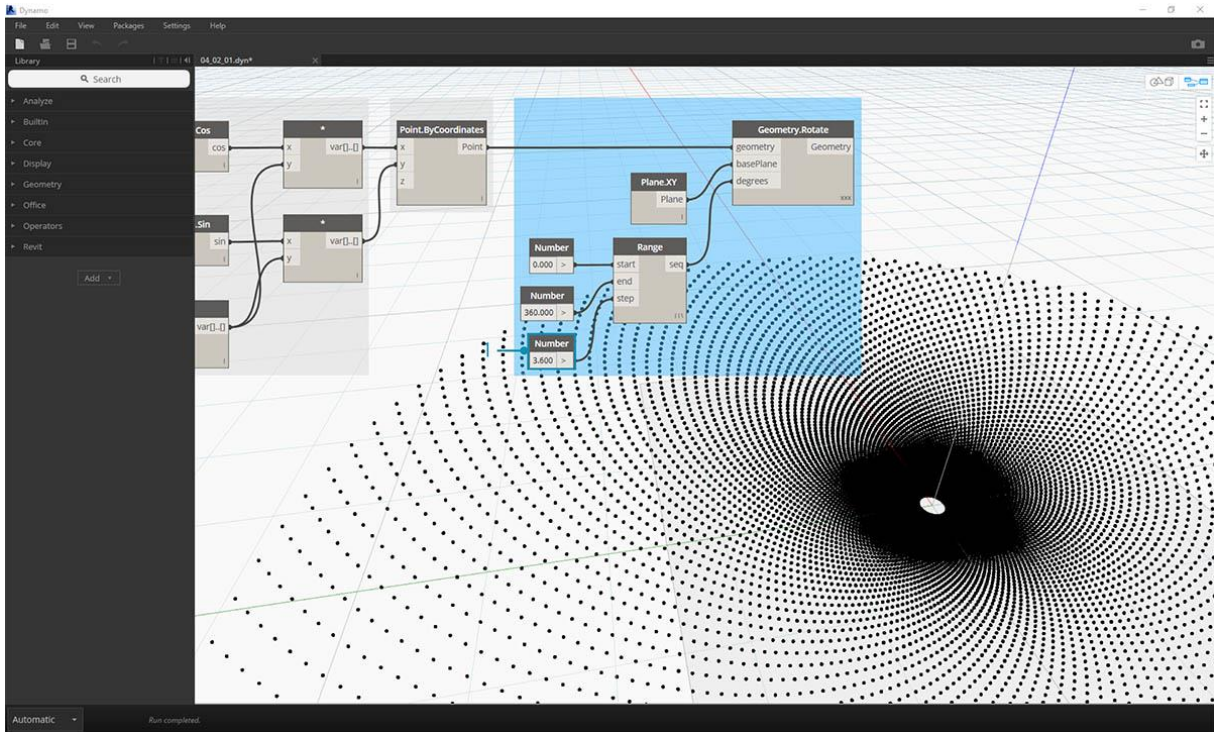
1. **Polycurve.ByPoints:** Nuevamente, este es el Nodo del ejercicio anterior, que usaremos como referencia.
2. **Circle.ByCenterPointRadius:** Usaremos un Nodo circular aquí con las mismas entradas que en el paso anterior. El valor de radio está predeterminado en 1.0 , por lo que vemos un resultado inmediato de círculos. Se vuelve inmediatamente legible cómo los puntos divergen más lejos del origen.

DARCO
DESDE 1988



1. Una vez más, como punto de partida, comencemos con el mismo paso del ejercicio anterior: crear una matriz en espiral de puntos con el nodo **Point.ByCoordinates**.





1. Cambie el tamaño de paso del nodo **Rango de número** de 36.0 a 3.6. Esto ahora nos da una grilla mucho más densa, y la direccionalidad de la espiral no está clara. Señoras y señores, hemos creado un girasol.

Lógica




La **lógica**, o más específicamente, la **Lógica Condicional**, nos permite especificar una acción o un conjunto de acciones basadas en una prueba. Después de evaluar la prueba, tendremos un valor booleano que representa True o False que podemos usar para controlar el flujo del programa.

Booleanos

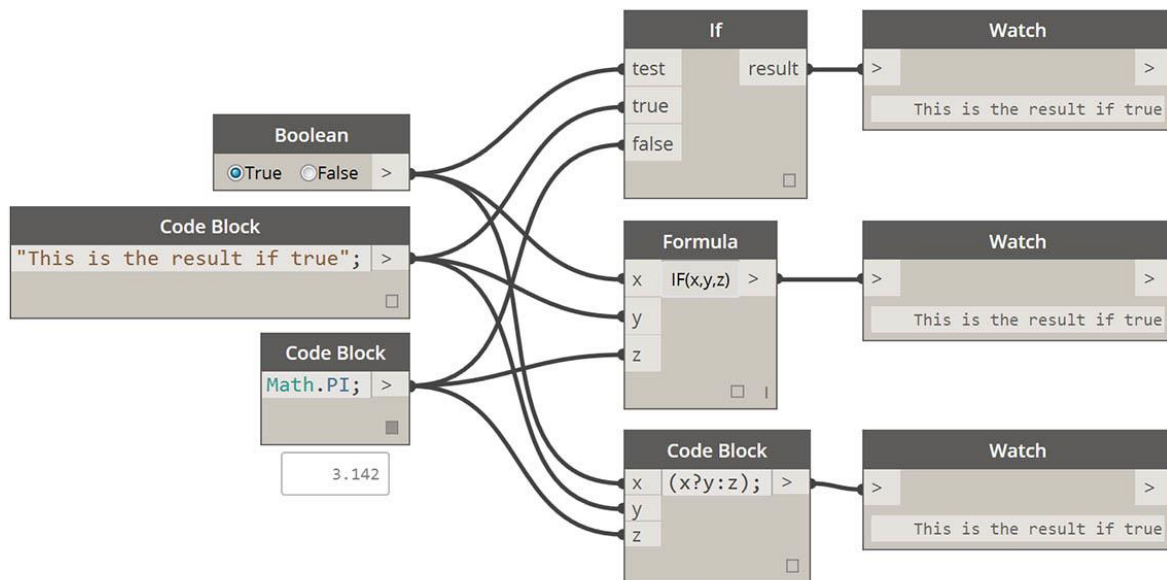
Las variables numéricas pueden almacenar una amplia gama de números diferentes. Las variables booleanas solo pueden almacenar dos valores denominados True o False, Yes o No, 1 o 0. Raramente utilizamos booleanos para realizar cálculos debido a su rango limitado.

Declaraciones condicionales

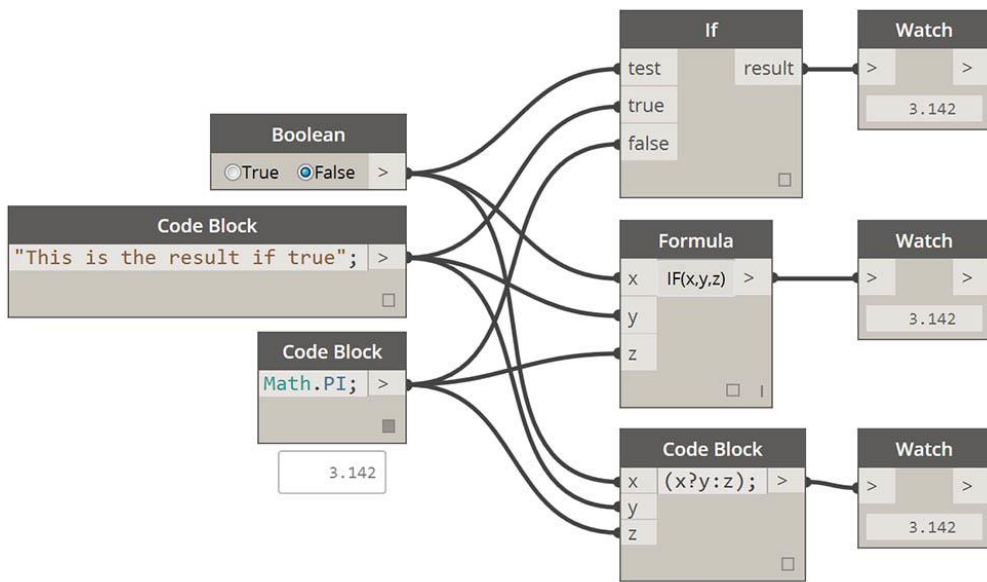
La afirmación "If" es un concepto clave en la programación: "Si esto es cierto, entonces eso sucede, de lo contrario sucede algo más". La acción resultante de la declaración es impulsada por un valor booleano. Hay varias formas de definir un "Si" declaración en Dynamo:

Icono	Nombre	Sintaxis	Entradas	Salidas
	Si	Si	prueba, verdadero, falso	resultado
	Fórmula	IF (x, y, z)	x, y, z	resultado
	Bloque de código de	(x? y: z)	x, y, z	resultado

Repasemos un breve ejemplo de cada uno de estos tres nodos en acción usando la declaración condicional "Si":



En esta imagen, *boolean* se establece en *true*, lo que significa que el resultado es una lectura de cadena: "este es el resultado si es verdadero". Los tres Nodos que crean la instrucción *If* funcionan de manera idéntica aquí.

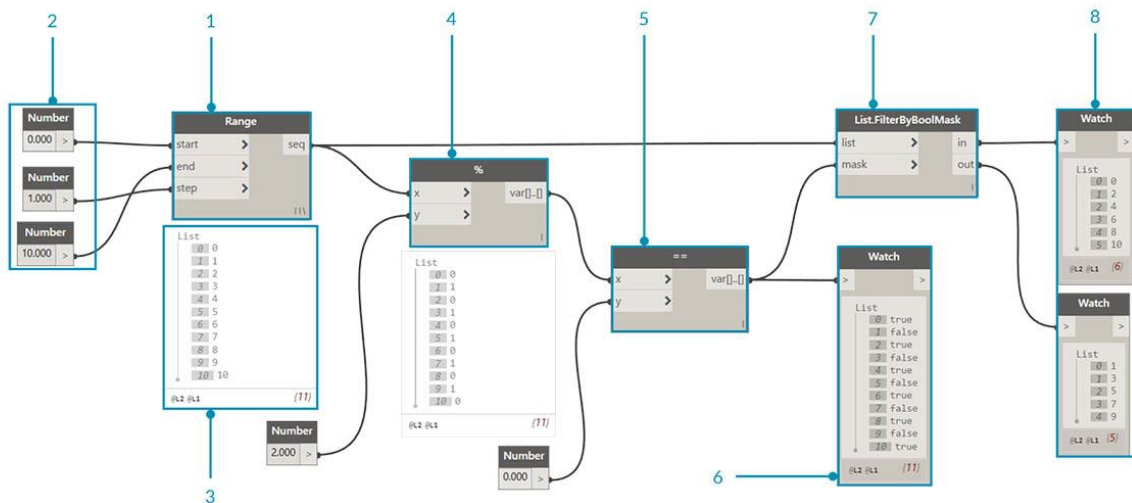


De nuevo, los Nodos funcionan de manera idéntica. Si el valor *booleano* se cambia a *falso*, nuestro resultado es el número *Pi*, tal como se define en la declaración *If* original.

Filtrando una lista

Descargue el archivo de ejemplo que acompaña a este Building Blocks of Programs - Logic.dyn . Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

Usemos la lógica para separar una lista de números en una lista de números pares y una lista de números impares.

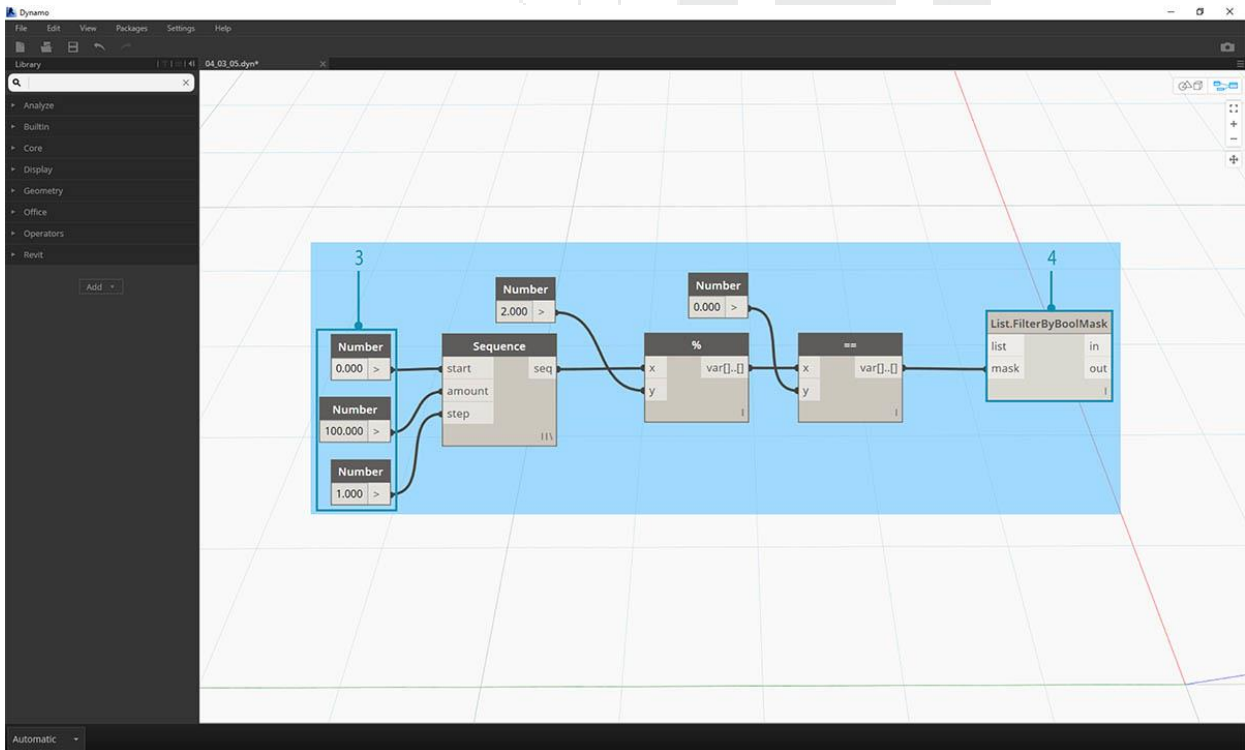


1. **Rango de números:** agrega un rango numérico al lienzo.
2. **Números:** agregue tres nodos numéricos al lienzo. El valor para cada nodo de número debe ser: 0.0 para *inicio*, 10.0 para *final* y 1.0 para *paso*.

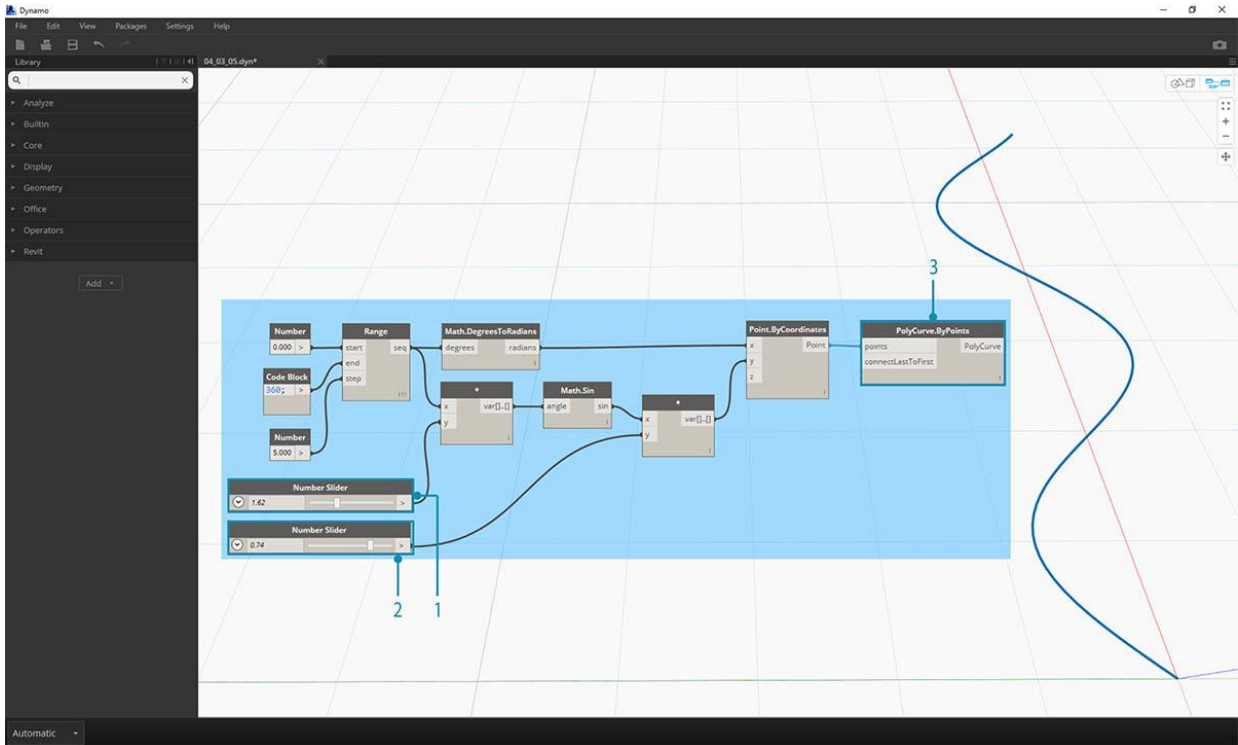
3. **Salida** - nuestra salida es una lista de 11 números que van del 0-10.
4. **Módulo (%)** - *Rango de números* en x y 2.0 en y . Esto calcula el resto para cada número en la lista dividido por 2. El resultado de esta lista nos da una lista de valores alternando entre 0 y 1.
5. **Prueba de igualdad (==)**: agregue una prueba de igualdad al lienzo. Conecte la salida del *módulo* en la entrada x y 0.0 en la entrada y .
6. **Ver**: el resultado de la prueba de igualdad es una lista de valores que alterna entre verdadero y falso. Estos son los valores utilizados para separar los elementos en la lista. 0 (o *verdadero*) representa números pares y 1 , o *falso* representa números impares.
7. **List.FilterByBoolMask**: este nodo filtrará los valores en dos listas diferentes basadas en la entrada booleana. Conecte el *rango de números* original en la entrada de la *lista* y la salida de la *prueba de igualdad* en la entrada de la *máscara*. El *en* salida representa valores verdaderos, mientras que el *cabo* de salida representa valores falsos.
8. **Mira**, como resultado, ahora tenemos una lista de números pares y una lista de números impares. ¡Hemos usado operadores lógicos para separar listas en patrones!

De la Lógica a la Geometría

Partiendo de la lógica establecida en el primer ejercicio, apliquemos esta configuración en una operación de modelado.



1. Saltamos del ejercicio anterior con los mismos Nodos. Las únicas excepciones:
2. Hemos cambiado el formato.
3. Los valores de entrada han cambiado.
4. Hemos desenchufado la entrada de la lista en *List.FilterByBoolMask*. Pondremos estos Nodos a un lado por ahora, pero serán útiles más adelante en el ejercicio.

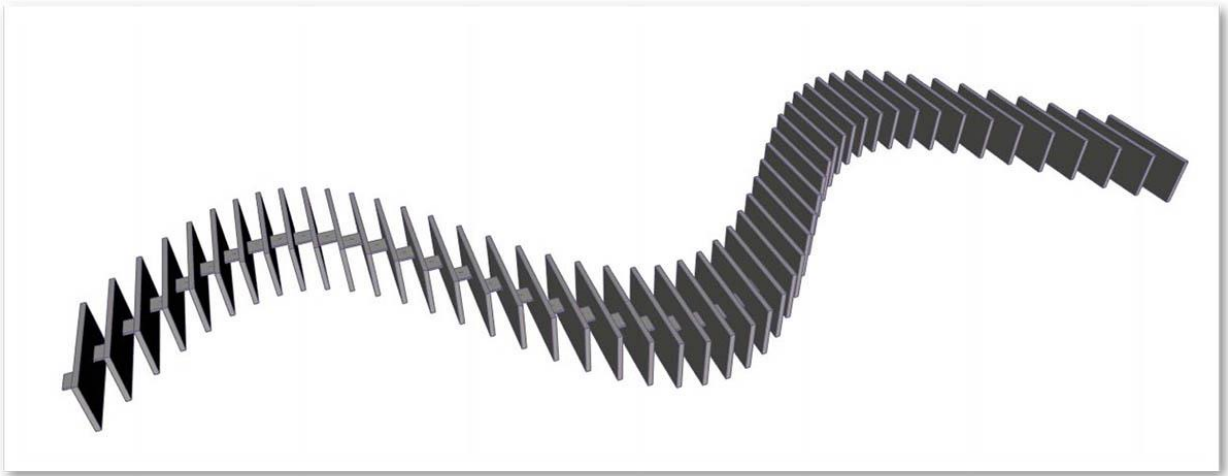


Comencemos conectando los Nodos juntos como se muestra en la imagen de arriba. Este grupo de nodos representa una ecuación paramétrica para definir una curva de línea. Algunas notas:

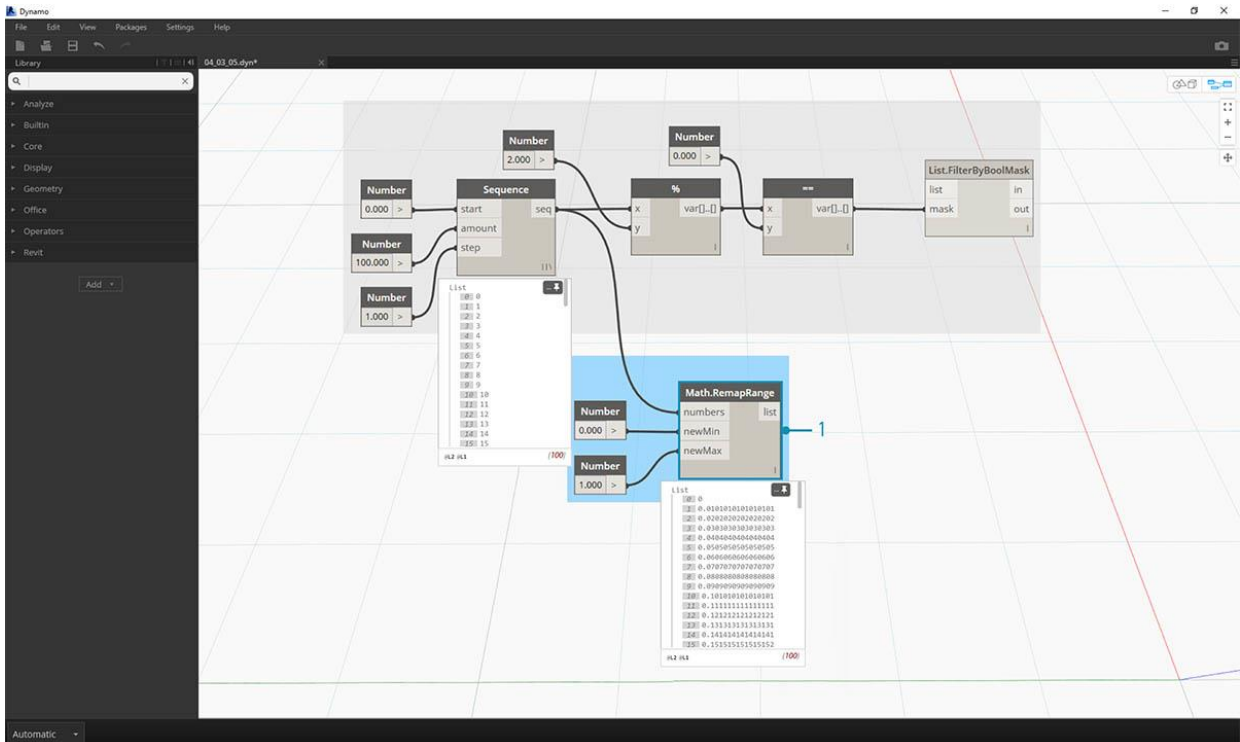
1. El **primer control deslizante** debe tener un mínimo de 1, un máximo de 4 y un paso de 0.01.
2. El **segundo control deslizante** debe tener un mínimo de 0, un máximo de 1 y un paso de 0.01.
3. **PolyCurve.ByPoints:** si se copia el diagrama de nodo anterior, el resultado es una curva sinusoidal en la vista previa de Dynamo.

El método aquí para las entradas: use nodos numéricos para más propiedades estáticas y controles deslizantes numéricos en los más flexibles. Queremos mantener el rango de números original que estamos definiendo al comienzo de este paso. Sin embargo, la curva sinusoidal que creamos aquí debería tener cierta flexibilidad. Podemos mover estos controles deslizantes para ver la curva actualizar su frecuencia y amplitud.

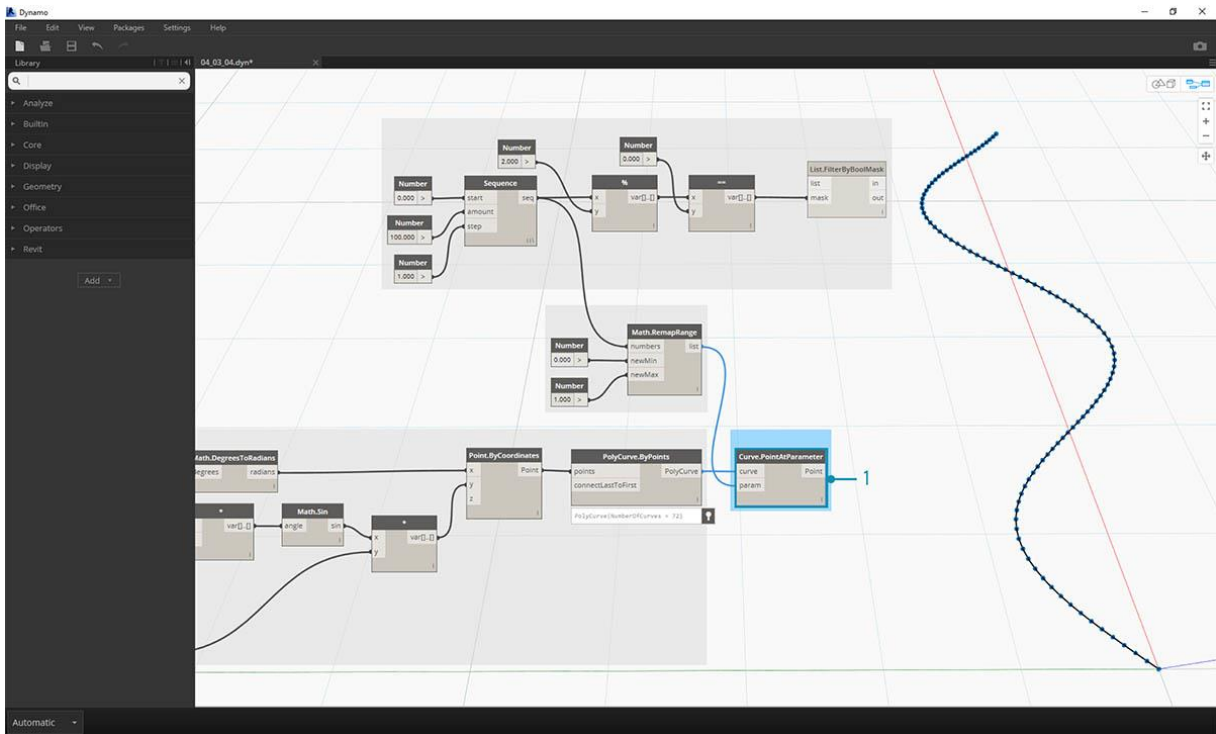
DARCO
DESDE 1988



Vamos a saltar un poco en la definición, así que veamos el resultado final para poder hacer referencia a lo que estamos obteniendo. Los dos primeros pasos se realizan por separado, ahora queremos conectar los dos. Usaremos la curva sinusoidal básica para controlar la ubicación de los componentes de la cremallera, y usaremos la lógica de verdadero / falso para alternar entre cajas pequeñas y cajas más grandes.

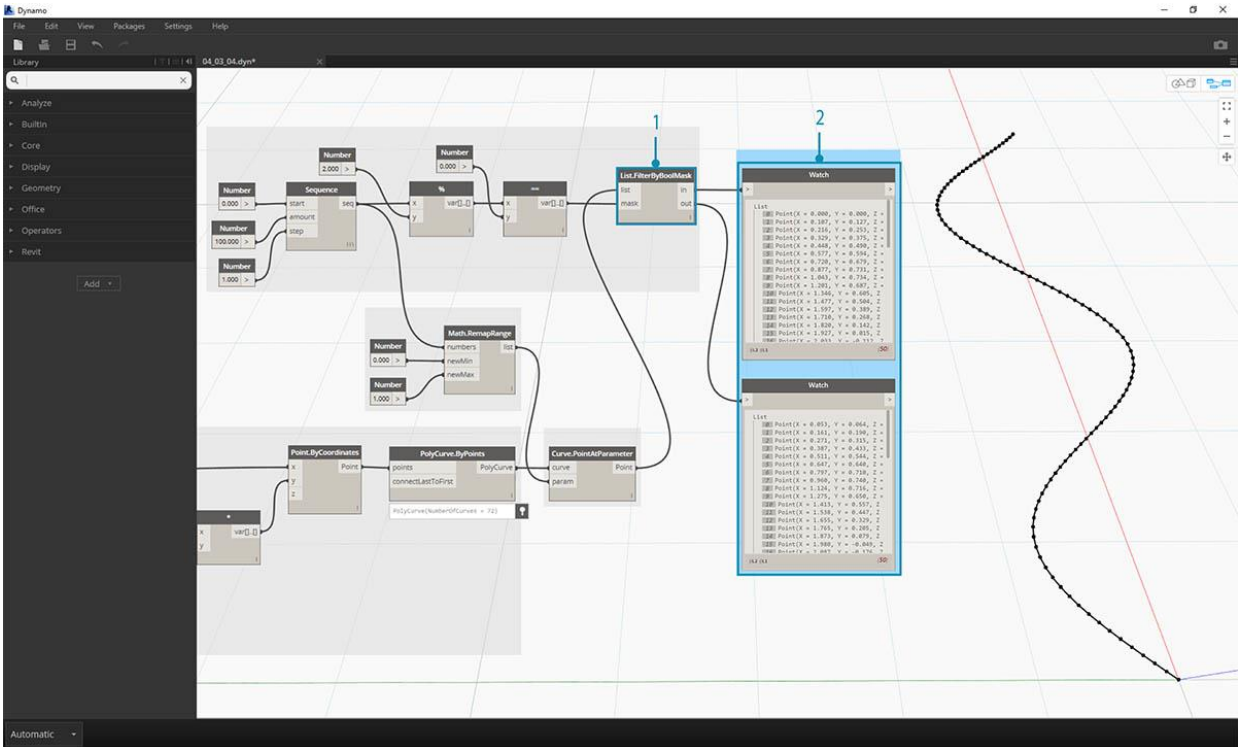


1. **Math.RemapRange** - Usando la secuencia numérica creada en el paso 01, vamos a crear una nueva serie de números reasignando el rango. Los números originales del paso 01 varían de 0 a 100. Estos números van de 0 a 1 por las *nuevas* entradas *Min* y *NewMax*, respectivamente.

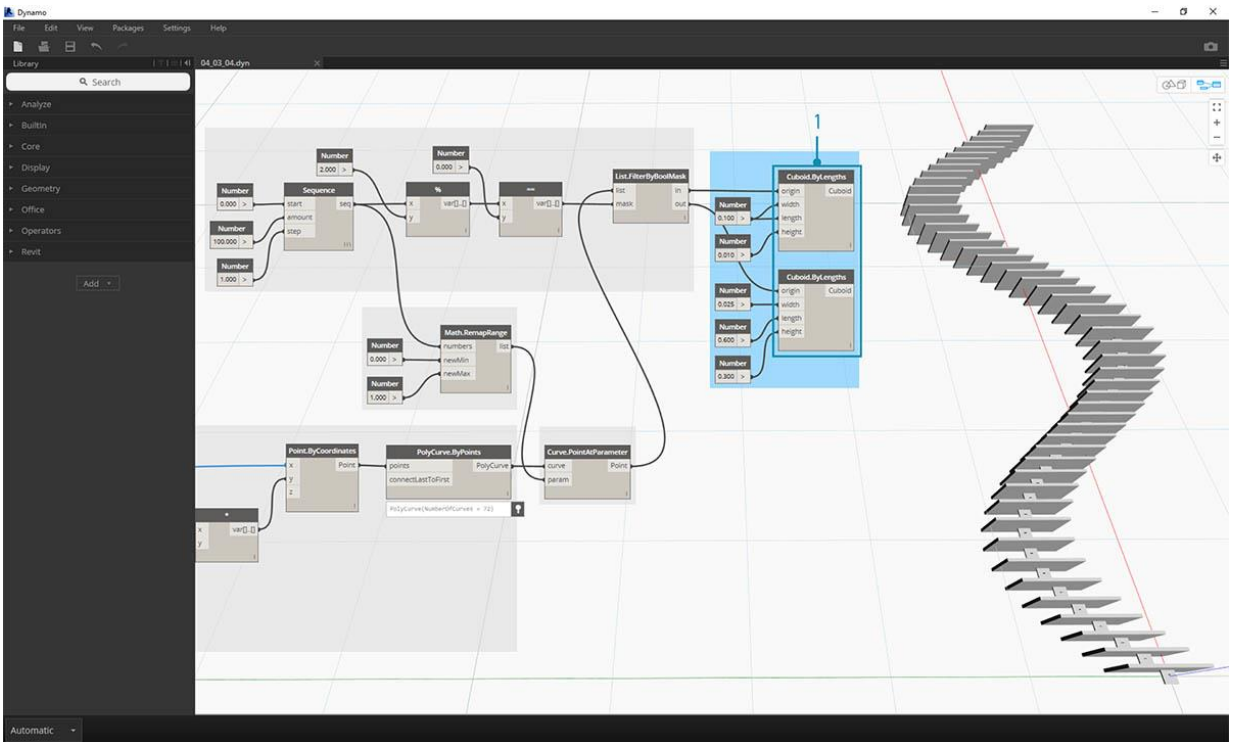


1. **Curve.PointAtParameter** - Conecta *Polycurve.ByPoints* (desde el paso 2) a la *curva* y *Math.RemapRange* en *param*. Este paso crea puntos a lo largo de la curva. Remapeamos los números a 0 a 1 porque la entrada de *param* busca valores en este rango. Un valor de 0 representa el punto de inicio, un valor de 1 representa los puntos finales. Todos los números intermedios se evalúan dentro del rango $[0, 1]$.

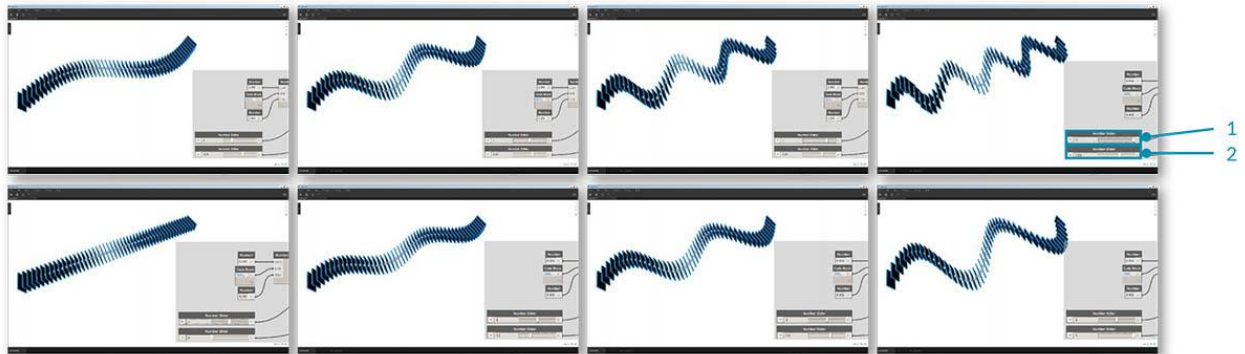
DARCO
DESDE 1988



1. **List.FilterByBoolMask** - Plug *Curve.PointAtParameter* del paso anterior en la entrada de la lista.
2. **Mirar**: un nodo de vigilancia para *dentro* y un nodo de vigilancia para *afuera* muestra que tenemos dos listas que representan incluso índices e índices impares. Estos puntos se ordenan de la misma manera en la curva, lo que demostramos en el siguiente paso.



1. **Cuboid.ByLengths:** recrear las conexiones que se ven en la imagen de arriba para obtener una cremallera a lo largo de la curva sinusoidal. Un cuboide es solo una caja aquí, y estamos definiendo su tamaño en función del punto de la curva en el centro del cuadro. La lógica de la división par / impar ahora debe quedar clara en el modelo.



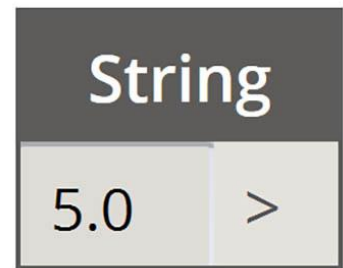
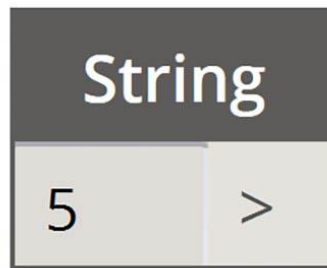
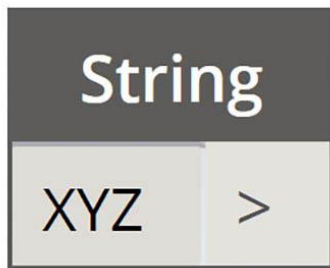
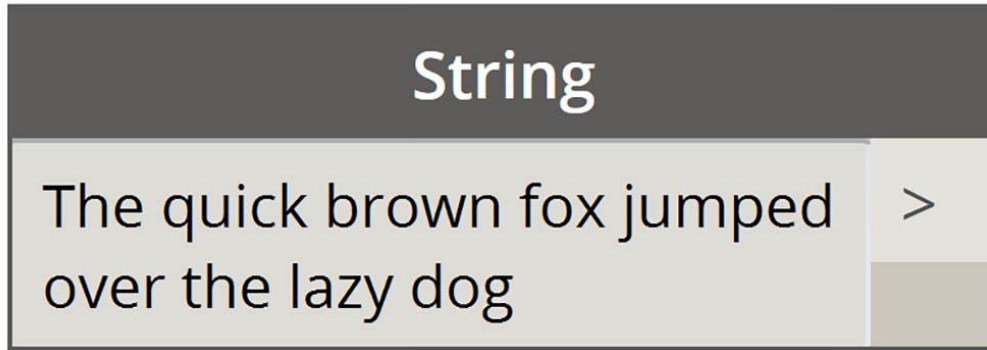
1. **Control deslizante numérico:** retrocediendo al principio de la definición, podemos flexionar el deslizador numérico y ver la actualización de la cremallera. La fila superior de imágenes representa un rango de valores para el control deslizante de número superior. Esta es la frecuencia de la ola.
2. **Control deslizante numérico:** la fila inferior de imágenes representa un rango de valores para el deslizador inferior. Esta es la amplitud de la ola.

Instrumentos de cuerda

Formalmente, una **Cadena** es una secuencia de caracteres que representa una constante literal o algún tipo de variable. Informalmente, una cadena es una jerga de programación para el texto. Hemos trabajado con números, tanto números enteros como decimales, para controlar los parámetros y podemos hacer lo mismo con el texto.

Creando cadenas

Las cadenas se pueden utilizar para una amplia gama de aplicaciones, incluida la definición de parámetros personalizados, la anotación de conjuntos de documentación y el análisis a través de conjuntos de datos basados en texto. El nodo de cadena se encuentra en el núcleo > categoría de entrada.



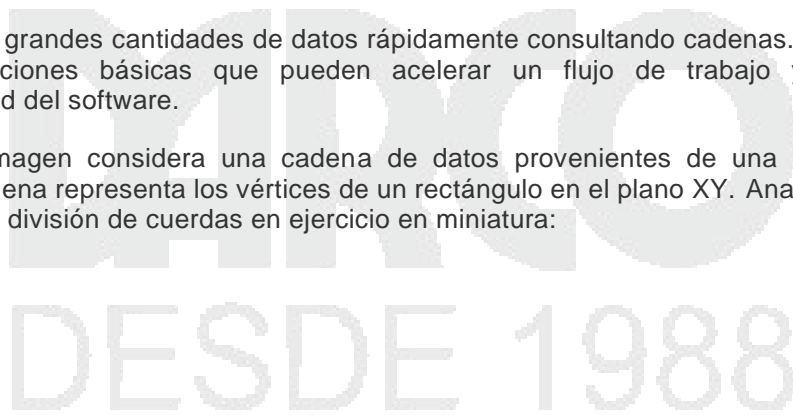
Los Nodos de muestra anteriores son cadenas. Un número puede representarse como una cadena, al igual que una letra o una matriz completa de texto.

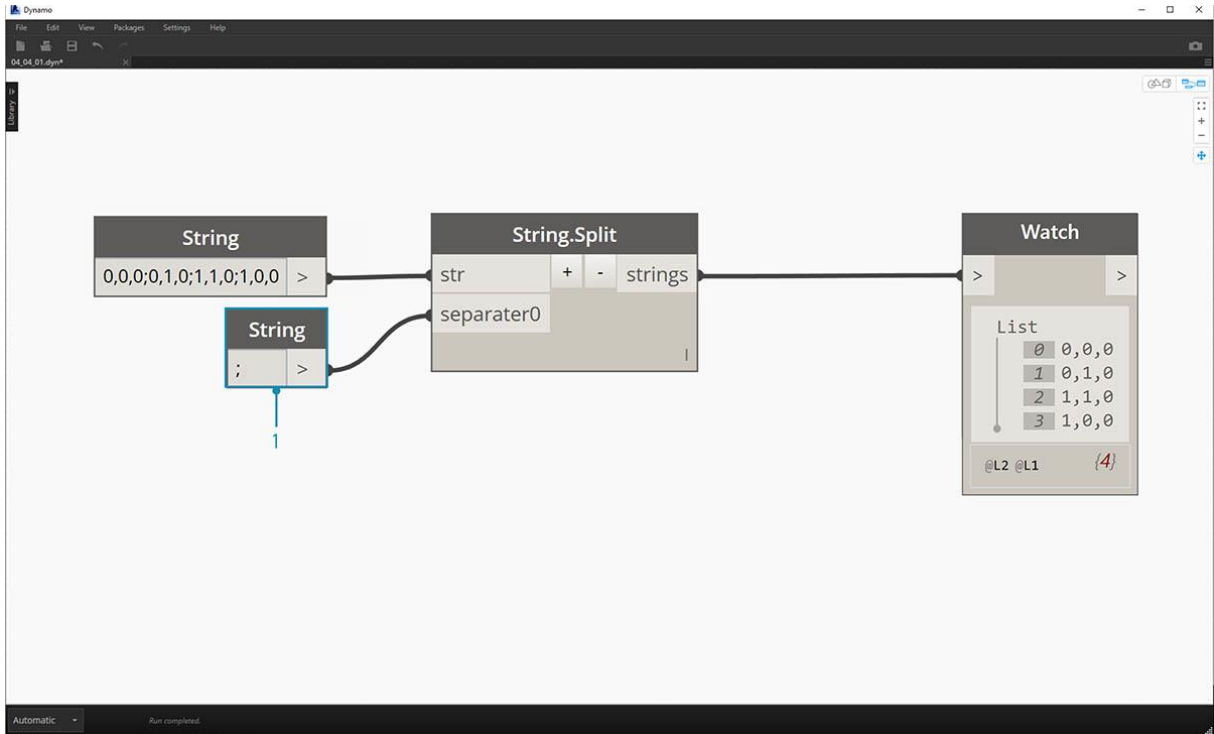
[Consulta de cadenas](#)

Descargue el archivo de ejemplo que acompaña a este ejercicio Building Blocks of Programs - Strings.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

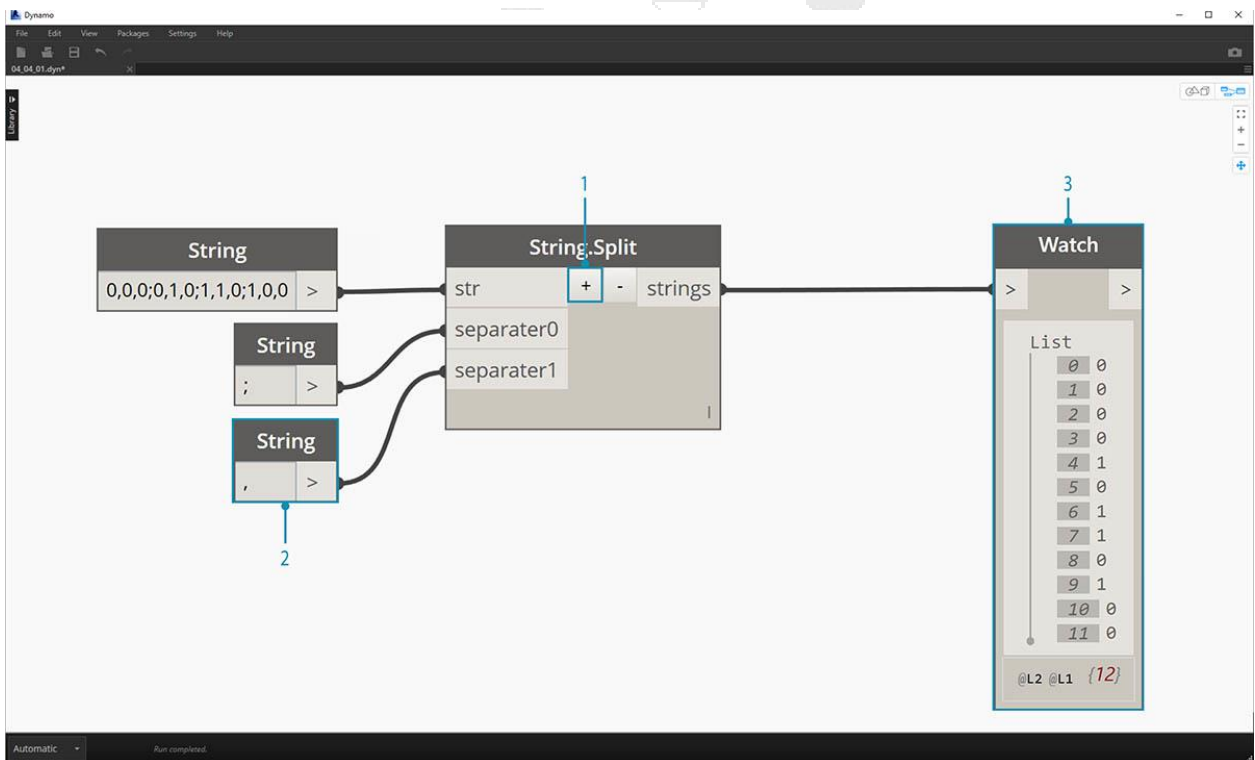
Puede analizar grandes cantidades de datos rápidamente consultando cadenas. Hablaremos de algunas operaciones básicas que pueden acelerar un flujo de trabajo y ayudar a la interoperabilidad del software.

La siguiente imagen considera una cadena de datos provenientes de una hoja de cálculo externa. La cadena representa los vértices de un rectángulo en el plano XY. Analicemos algunas operaciones de división de cuerdas en ejercicio en miniatura:





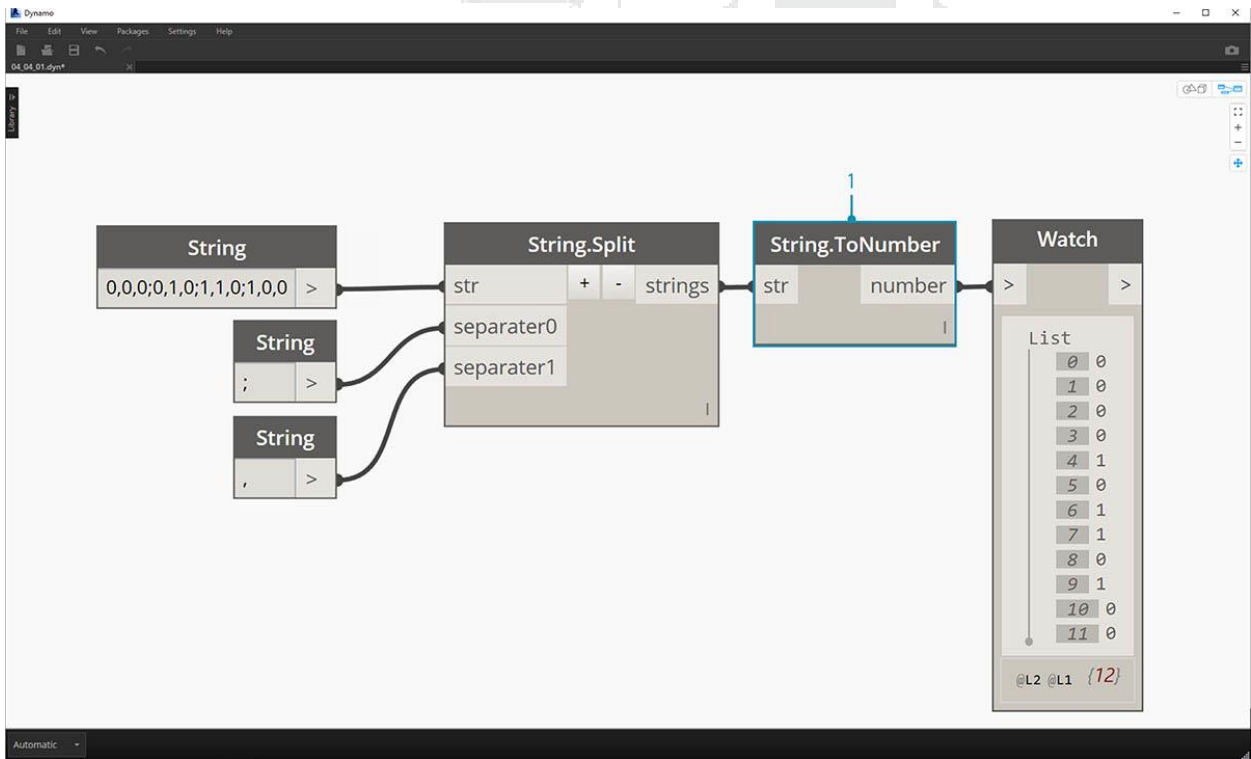
1. Los ";" separador divide cada vértice del rectángulo. Esto crea una lista con 4 elementos para cada vértice.



1. Al presionar el "+" en el medio del Nodo, creamos un nuevo separador.
2. Agregue una cadena ", " al lienzo y conéctela a la nueva entrada del separador.

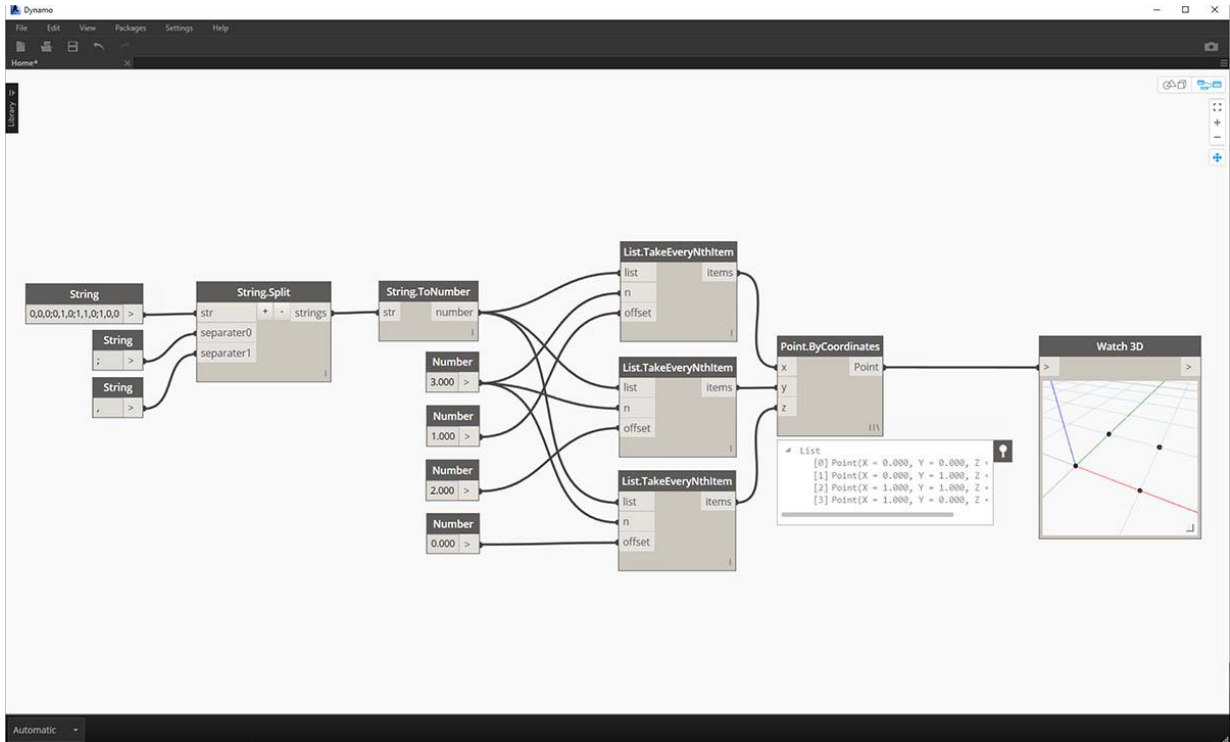
3. Nuestro resultado ahora es una lista de diez artículos. El Nodo primero se divide según el *separador 0*, y luego se basa en el *separador 1*.

Si bien la lista de elementos anteriores puede parecer números, todavía se consideran cadenas individuales en Dynamo. Para crear puntos, su tipo de datos debe convertirse de una cadena a un Número. Esto se hace con el nodo `String.ToNumber`



1. Este nodo es sencillo. Enchufe los resultados `String.Split` en la entrada. La salida no se ve diferente, pero el tipo de datos ahora es un *número* en lugar de una *cadena*.

DARCO
DESDE 1988

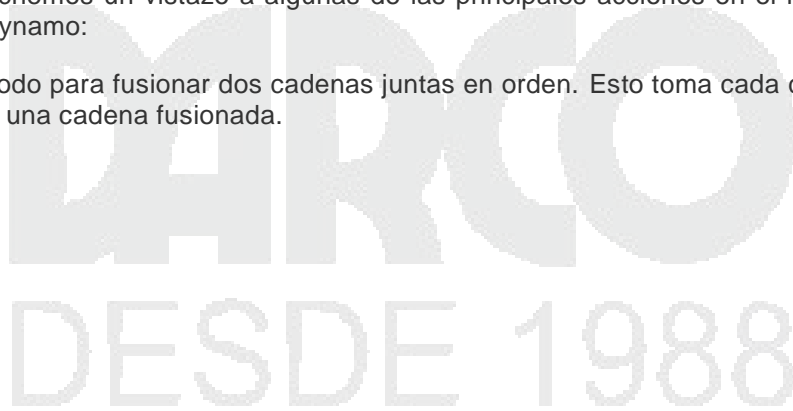


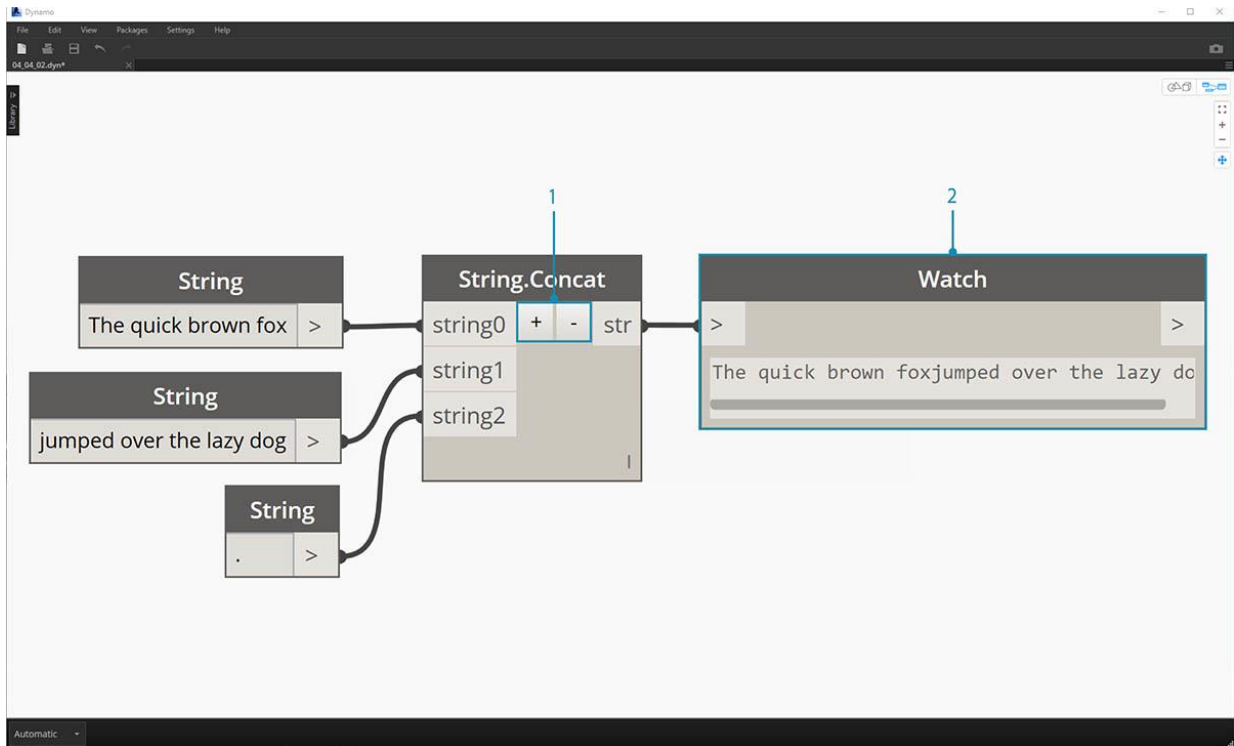
1. Con algunas operaciones adicionales básicas, ahora tenemos un rectángulo dibujado en el origen en función de la entrada de cadena original.

Manipulación de cadenas

Como una cadena es un objeto de texto genérico, alojan una amplia gama de aplicaciones. Echemos un vistazo a algunas de las principales acciones en el núcleo > Cadena Categoría en Dynamo:

Este es un método para fusionar dos cadenas juntas en orden. Esto toma cada cadena literal en una lista y crea una cadena fusionada.





La imagen de arriba representa la concatenación de tres cadenas:

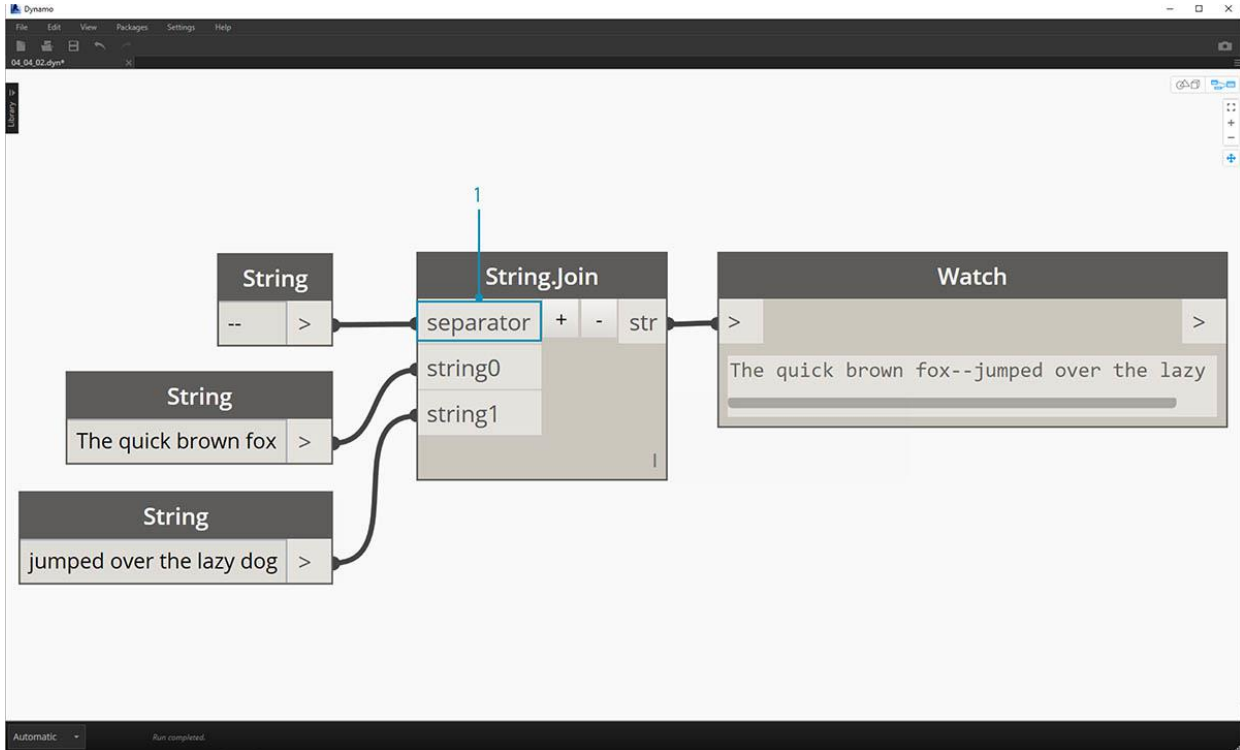
1. Agregue o reste cadenas a la concatenación haciendo clic en los botones +/- en el centro del nodo.
2. El resultado da una cadena concatenada, con espacios y signos de puntuación incluidos.

El método de unión es muy similar a concatenar, excepto que tiene una capa de puntuación agregada.

Si trabajó en Excel, es posible que haya encontrado un archivo CSV. Esto representa valores separados por comas. Se podría usar una coma (o en este caso, dos guiones) como separador con el Nodo de unión para crear una estructura de datos

DARCO
DESDE 1988

similar:

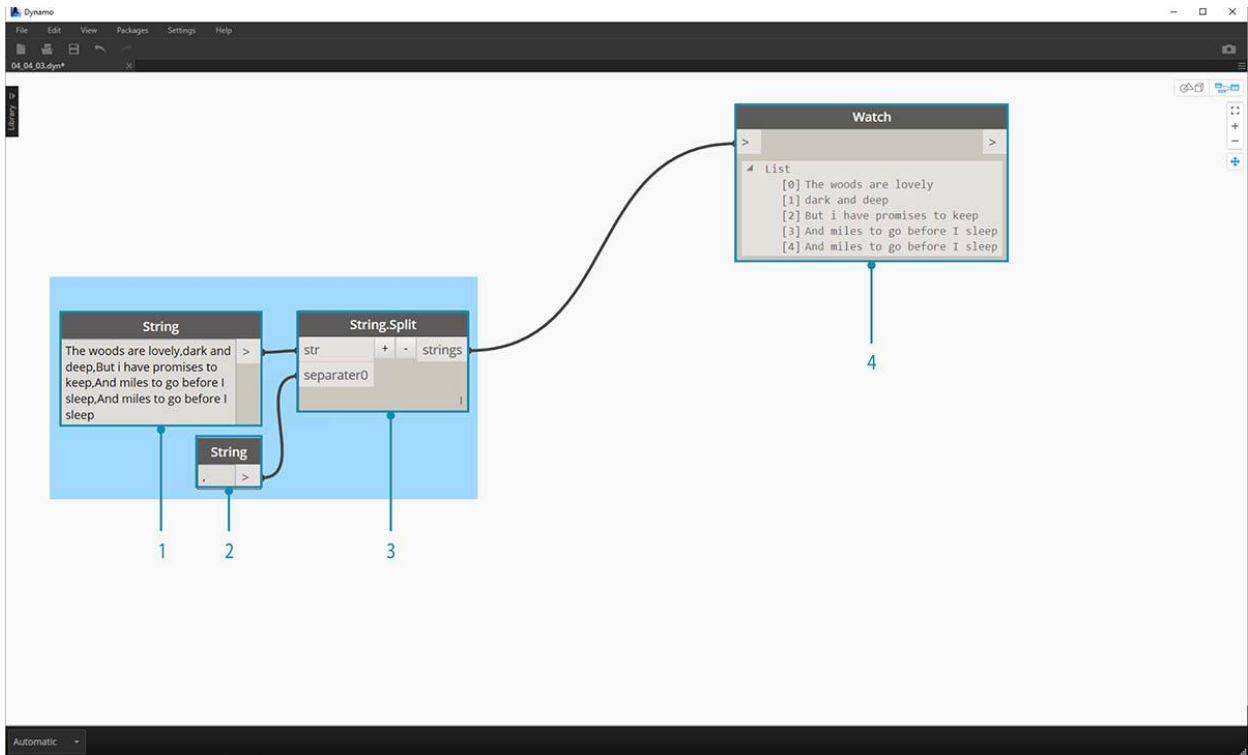


La imagen de arriba representa la unión de dos cadenas:

1. La entrada del separador permite crear una cadena que divide las cadenas unidas.

Trabajando con cadenas

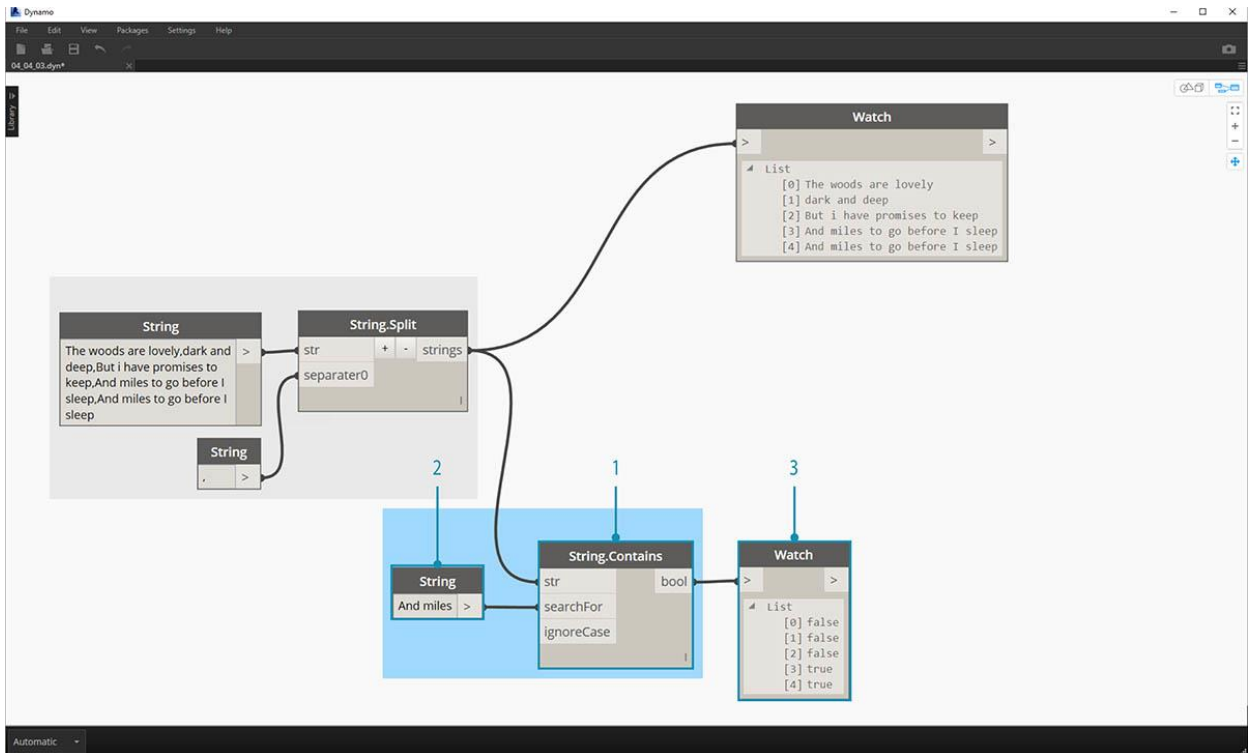
En este ejercicio, vamos a utilizar métodos de consulta y manipulación de cadenas para deconstruir la estrofa final de *Stopping By Woods* de Robert Frost en una tarde nevada. No es la aplicación más práctica, pero nos ayudará a captar acciones de cuerdas conceptuales a medida que las aplicamos a líneas legibles de ritmo y rima.



Comencemos con una división de cuerdas básica de la estrofa. Primero notamos que la escritura está formateada en base a comas. Usaremos este formato para separar cada línea en elementos individuales.

1. La cadena base se pega en un nodo de cadena.
2. Otro nodo de cadena se utiliza para indicar el separador. En este caso, estamos usando una coma.
3. Un String.Split Node se agrega al lienzo y se conecta a las dos cadenas.
4. El resultado muestra que ahora hemos separado las líneas en elementos individuales.

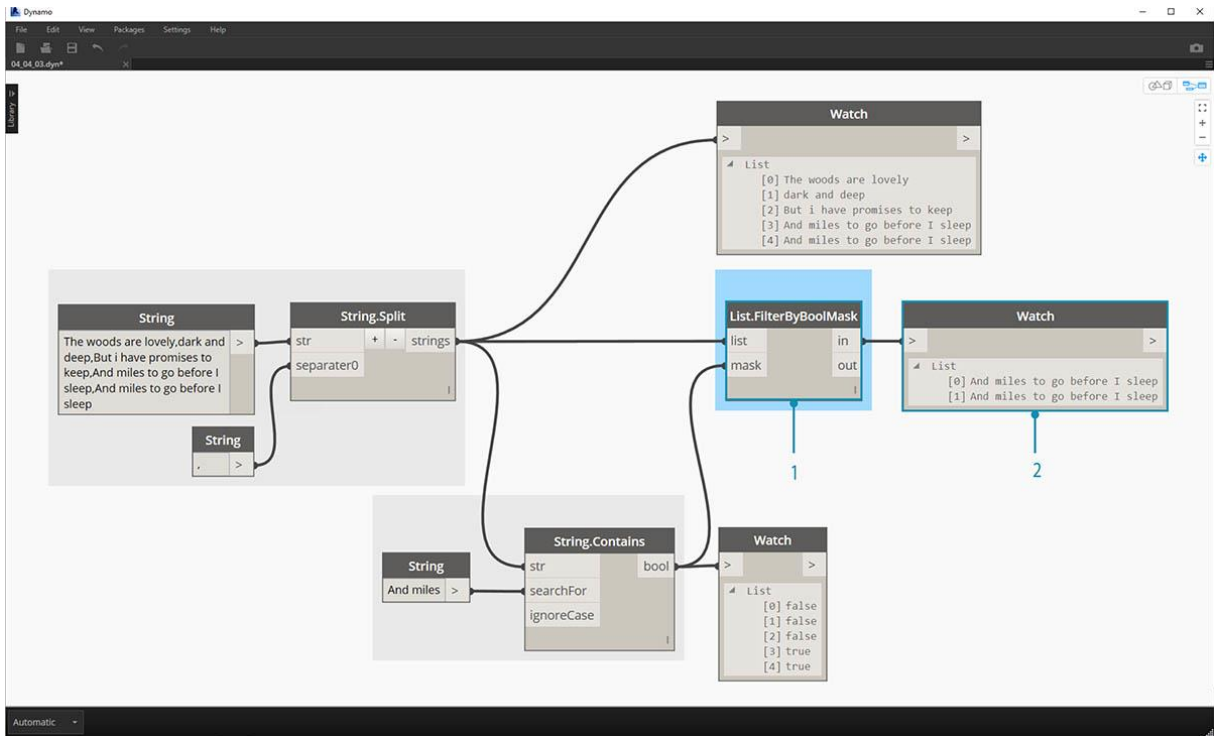
DARCO
DESDE 1988



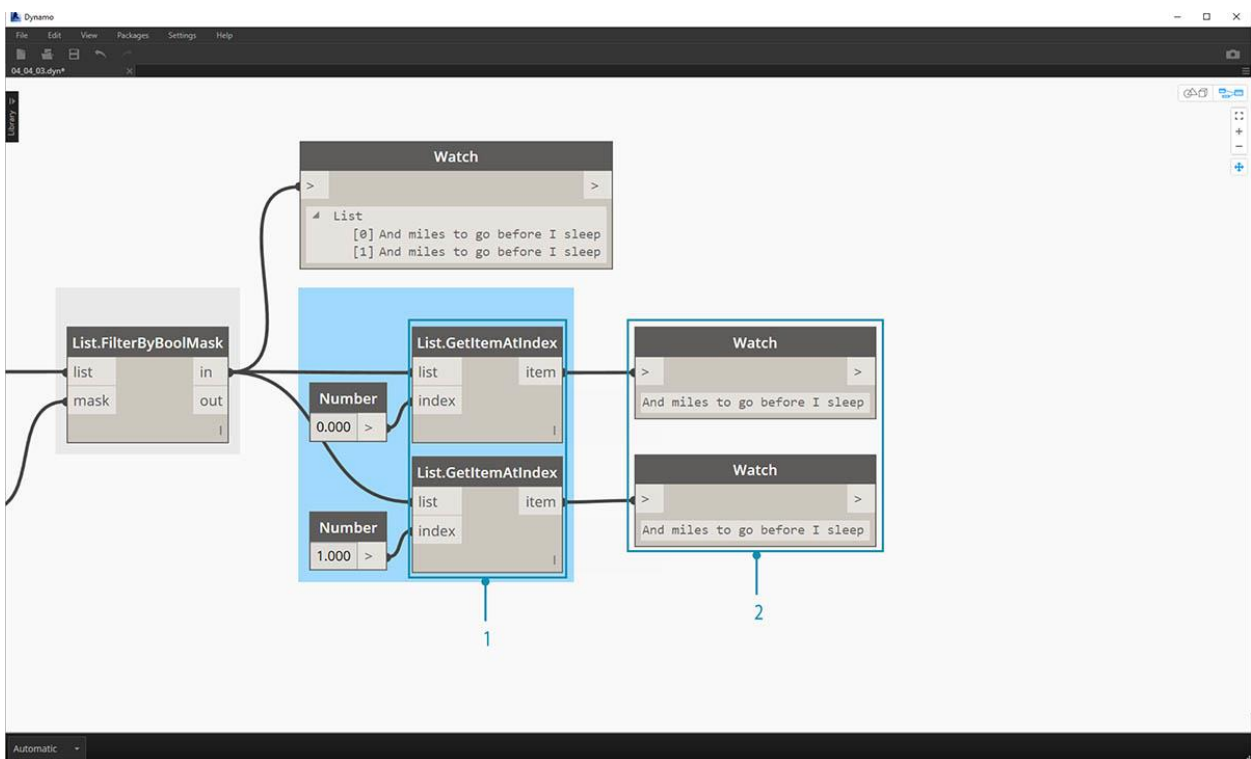
Ahora, vayamos a la parte buena del poema: las dos últimas líneas. La estrofa original era un elemento de datos. Separamos estos datos en elementos individuales en el primer paso. Ahora tenemos que buscar el texto que estamos buscando. Y si bien *podemos* hacer esto seleccionando los dos últimos elementos de la lista, si este fuera un libro completo, no querríamos leer todo y aislar manualmente los elementos.

1. En lugar de buscar manualmente, utilizamos un nodo String.Contains para realizar una búsqueda de un conjunto de caracteres. Esto es similar a hacer el comando "Buscar" en un procesador de textos. En este caso, obtenemos un retorno de "verdadero" o "falso" si esa subcadena se encuentra dentro del elemento.
2. En la entrada "searchFor", definimos una subcadena que estamos buscando dentro de la stanza. Usemos un nodo de cadena con el texto "Y millas".
3. El resultado nos da una lista de falsos y trues. Usaremos esta lógica booleana para filtrar los elementos en el siguiente paso.

DARCO
DESDE 1988

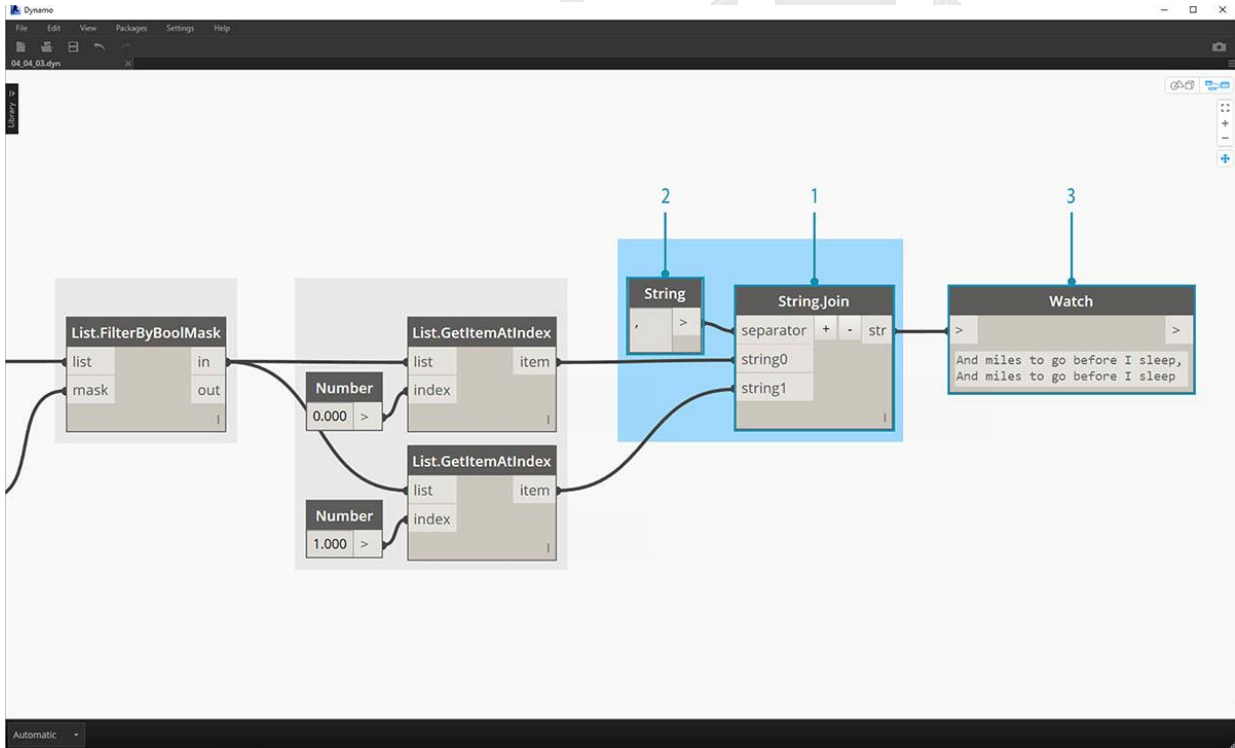


1. List.FilterByBoolMask es el nodo que queremos usar para descartar los falsos y verdaderos. El resultado "in" devuelve las declaraciones con una entrada "máscara" de "verdadero", mientras que la salida "salida" devuelve aquellas que son "falsas".
2. Nuestro resultado del "en" es el esperado, dándonos las dos últimas líneas de la estrofa.



Ahora, queremos llevar a casa la repetición de la estrofa uniendo las dos líneas. Al ver el resultado del paso anterior, notamos que hay dos elementos en la lista:

1. Usando dos nodos List.GetItemAtIndex, podemos aislar los elementos usando los valores de 0 y 1 como la entrada de índice.
2. La salida para cada nodo nos da, en orden, las dos líneas finales.



Para combinar estos dos elementos en uno, usamos el nodo String.Join:

1. Después de agregar el nodo String.Join, notamos que necesitamos un separador.
2. Para crear el separador, agregamos un nodo de cadena al lienzo y escribimos una coma.
3. El resultado final ha fusionado los dos últimos elementos en uno.


Esto puede parecer mucho trabajo para aislar las dos últimas líneas; y es cierto, las operaciones de cadena a menudo requieren un trabajo inicial. Pero son escalables, y se pueden aplicar a grandes conjuntos de datos con relativa facilidad. Si está trabajando paramétricamente con hojas de cálculo e interoperabilidad, asegúrese de tener en cuenta las operaciones de cadena.

Color

El color es un gran tipo de datos para crear efectos visuales atractivos y para representar la diferencia en el resultado de su Programa Visual. Al trabajar con datos abstractos y números variables, a veces es difícil ver qué cambia y hasta qué punto. Esta es una gran aplicación para colores.

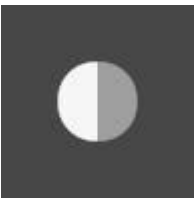



Creando colores

Los colores en Dynamo se crean usando entradas ARGB. Esto corresponde a los canales Alfa, Rojo, Verde y Azul. El alfa representa la *transparencia* del color, mientras que los otros tres se utilizan como colores primarios para generar todo el espectro de color en concierto.

Icono	Nombre	Sintaxis	Entradas	Salidas
	ARGB Color	Color.ByARGB	A, R, G, B	color




Consultar los valores de color

Los colores en la tabla siguiente consultan las propiedades utilizadas para definir el color: Alfa, Rojo, Verde y Azul. Tenga en cuenta que el nodo Color.Components nos da las cuatro salidas diferentes, lo que hace que este nodo sea preferible para consultar las propiedades de un color.

Icono	Nombre	Sintaxis	Entradas	Salidas
	Alfa	Color.Alpha	color	UN
	rojo	Color rojo	color	R
	Verde	Color verde	color	GRAMO
	Azul	Color azul	color	segundo

Icono	Nombre	Sintaxis	Entradas	Salidas
	Componentes	Color.Componentes	color	A, R, G, B

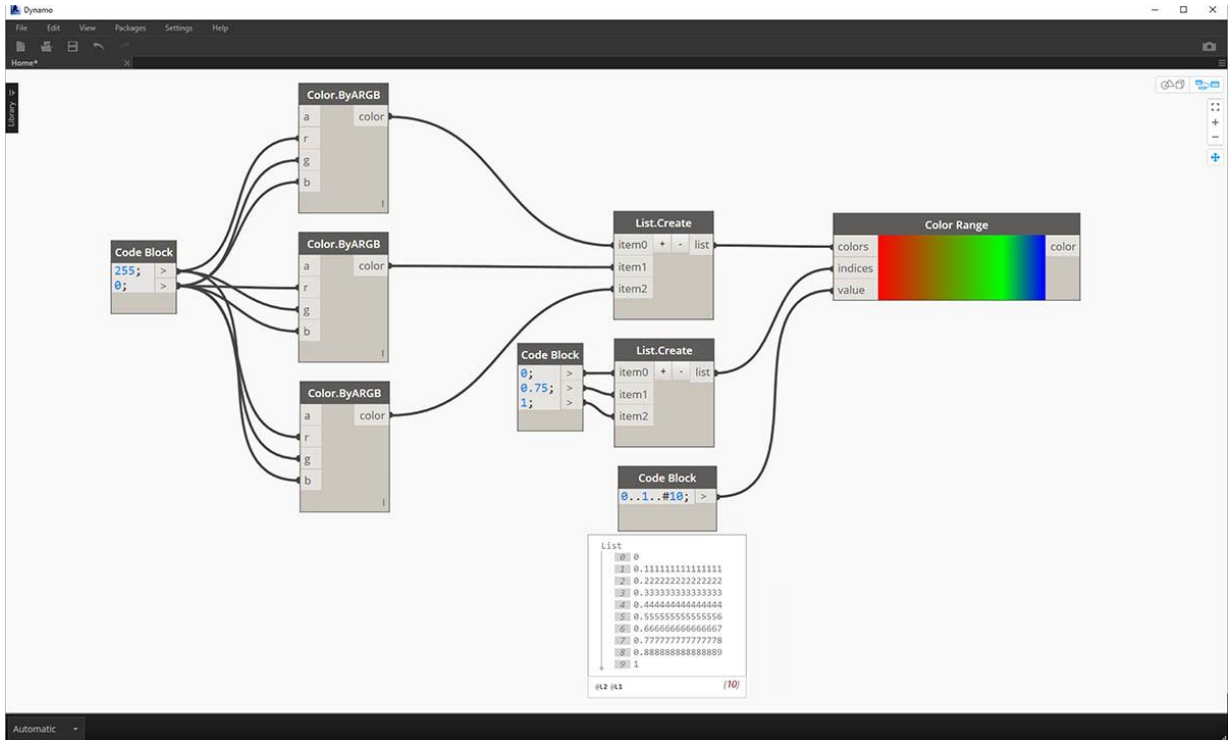
Los colores en la tabla siguiente corresponden al **espacio de color HSB**. Puede decirse que dividir el color en tono, saturación y brillo es más intuitivo para la forma en que interpretamos el color: ¿de qué color debería ser? ¿Qué tan colorido debería ser? ¿Y qué tan claro u oscuro debería ser el color? Este es el desglose del tono, la saturación y el brillo, respectivamente.

Icono	Nombre de consulta	Sintaxis	Entradas	Salidas
	Matiz	Color.Hue	color	Matiz
	Saturación	Saturación de color	color	Saturación
	Brillo	Color.Brightness	color	Brillo

Gama de colores

El rango de color es similar al nodo de **rango** de reasignación de la sección 4.2: reasigna una lista de números en otro dominio. Pero en lugar de mapear a un dominio *numérico*, se asigna a un *gradiente de color* basado en números de entrada que van de 0 a 1.

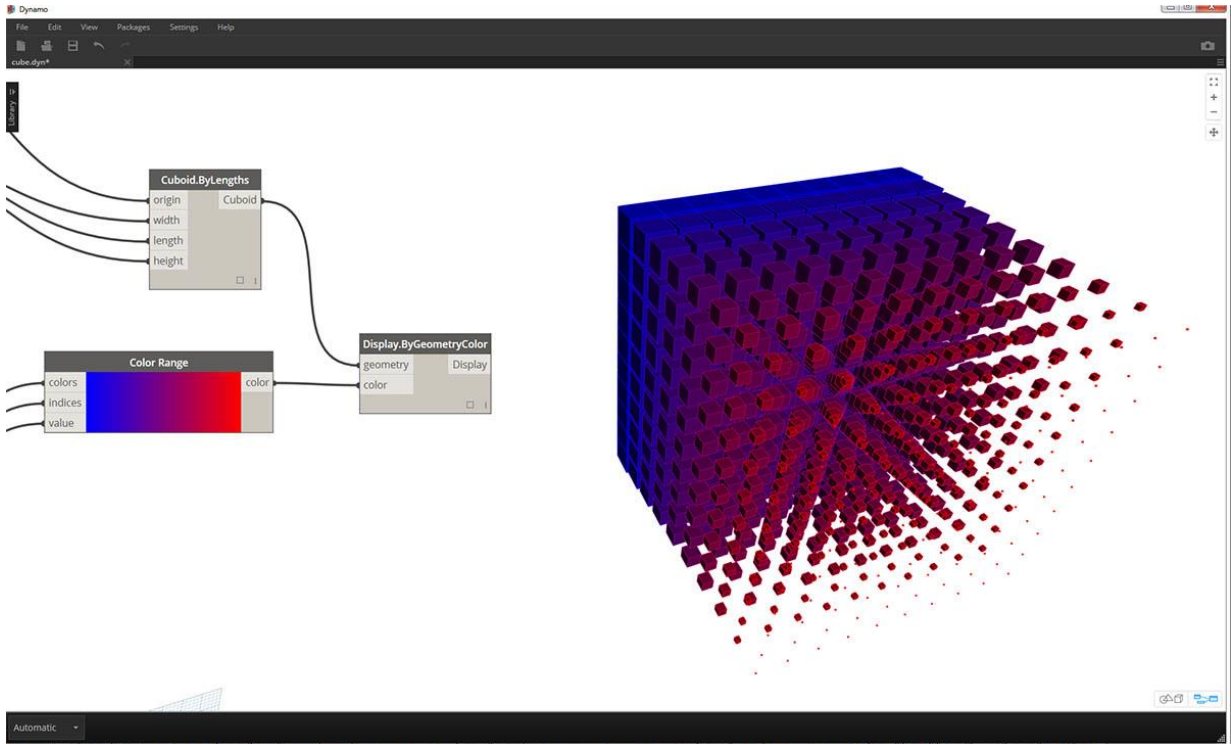
El Nodo actual funciona bien, pero puede ser un poco incómodo hacer que todo funcione la primera vez. La mejor forma de familiarizarse con el gradiente de color es probarlo interactivamente. Hagamos un ejercicio rápido para revisar cómo configurar un degradado con los colores de salida correspondientes a los números.



1. **Defina tres colores:** utilizando un nodo de bloque de código, defina *rojo*, *verde* y *azul* conectando las combinaciones apropiadas de 0 y 255.
2. **Crear lista:** combine los tres colores en una sola lista.
3. **Definir índices:** cree una lista para definir las posiciones de agarre de cada color (que van de 0 a 1). Observe el valor de 0.75 para verde. Esto coloca el color verde 3/4 del camino a través del degradado horizontal en el control deslizante de la gama de colores.
4. **Bloque de código:** valores de entrada (entre 0 y 1) para traducir a colores.

Vista previa de color

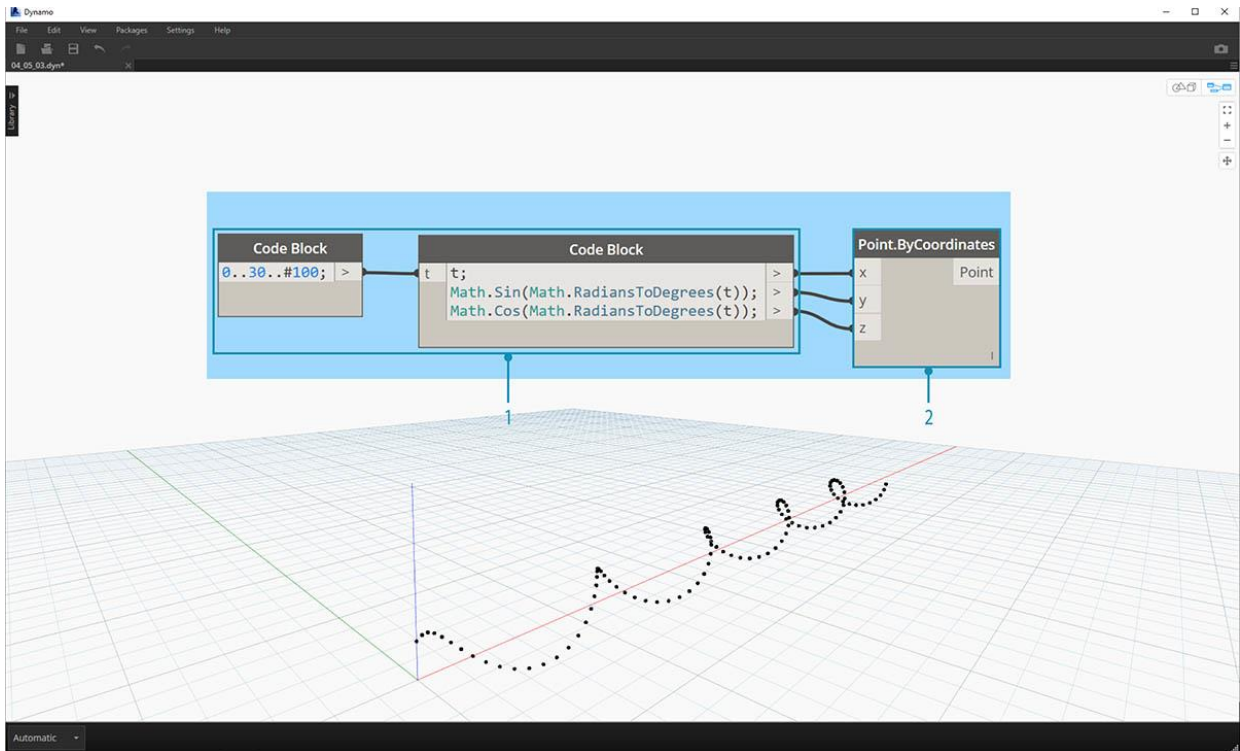
El nodo **Display.ByGeometry** nos da la capacidad de colorear la geometría en la vista de Dynamo. Esto es útil para separar diferentes tipos de geometría, demostrar un concepto paramétrico o definir una leyenda de análisis para la simulación. Las entradas son simples: geometría y color. Para crear un degradado como la imagen anterior, la entrada de color está conectada al Nodo de la **gama de colores**.



Ejercicio de color

Descargue el archivo de ejemplo que acompaña a este ejercicio Building Blocks of Programs - Color.dyn . Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

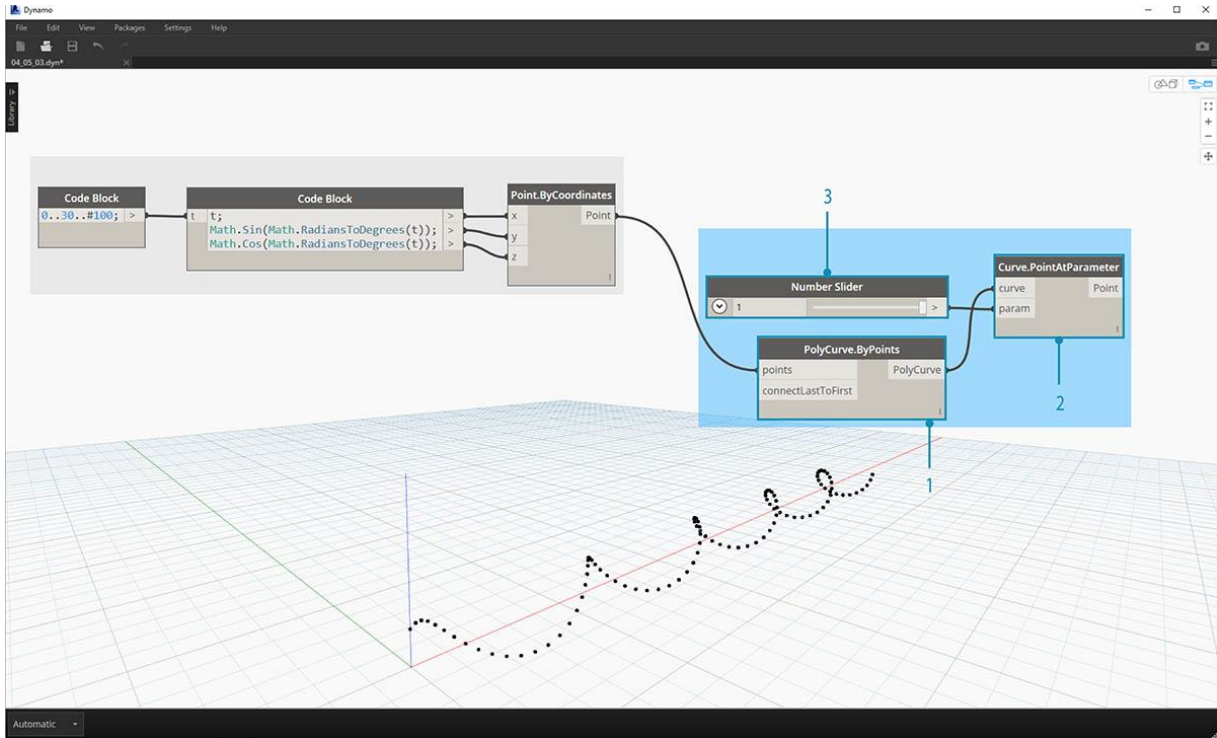
Este ejercicio se enfoca en controlar el color paramétricamente en paralelo con la geometría. La geometría es una hélice básica, que definimos a continuación utilizando el **Bloque de código** (3.2.3). Esta es una manera rápida y fácil de crear una función paramétrica; y dado que nuestro foco está en el color (en lugar de la geometría), usamos el bloque de código para crear eficientemente la hélice sin saturar el lienzo. Utilizaremos el bloque de código con más frecuencia a medida que el cebador se mueva a un material más avanzado.



1. **Bloque de código:** define los dos bloques de código con las fórmulas anteriores. Este es un método paramétrico rápido para crear una espiral.
2. **Point.ByCoordinates:** conecta las tres salidas del bloque de código a las coordenadas para el nodo.

Ahora vemos una serie de puntos que crean una hélice. El siguiente paso es crear una curva a través de los puntos para que podamos visualizar la hélice.

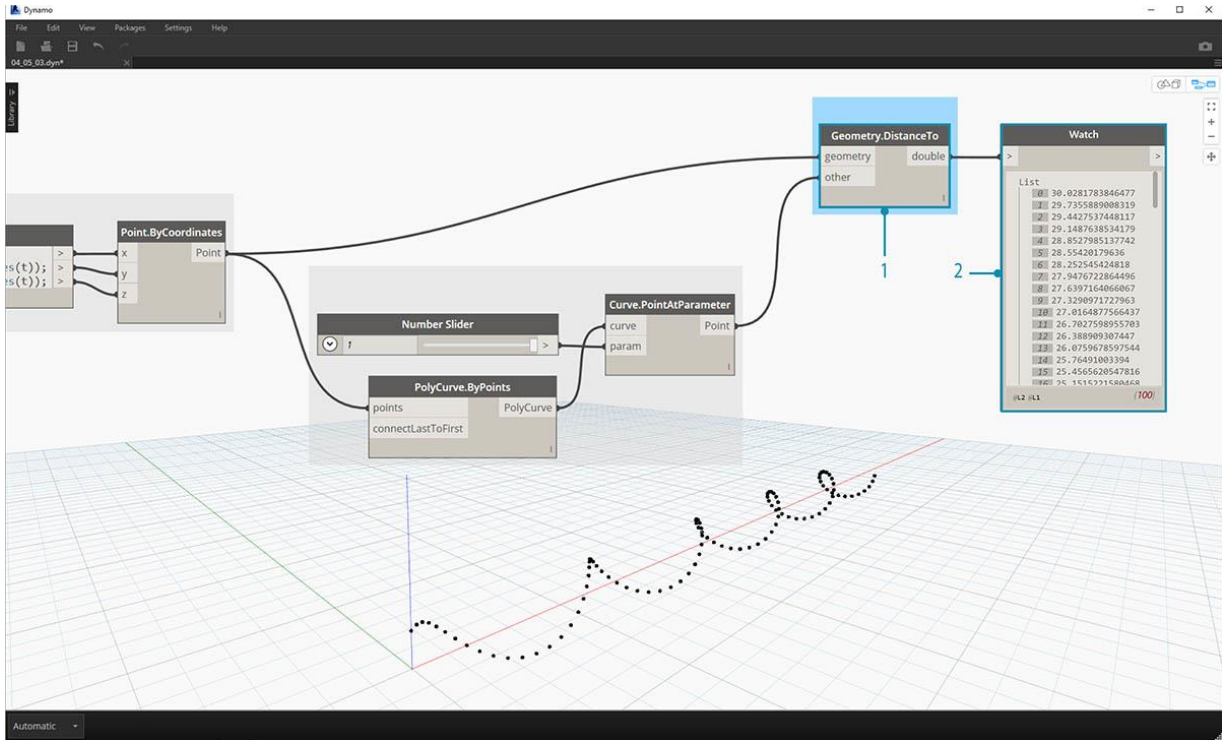
DARCO
DESDE 1988



1. **PolyCurve.ByPoints:** conecta la salida *Point.ByCoordinates* en la entrada de *puntos* para el nodo. Obtenemos una curva helicoidal.
2. **Curve.PointAtParameter:** conecta la salida *PolyCurve.ByPoints* a la entrada de *curva*. El propósito de este paso es crear un punto atractor paramétrico que se deslice a lo largo de la curva. Dado que la curva está evaluando un punto en el parámetro, necesitaremos a la entrada de un *parámetro* valor entre 0 y 1.
3. **Control deslizante numérico:** después de agregar al lienzo, cambie el valor *mínimo* a 0.0, el valor *máximo* a 1.0 y el valor del *paso* a .01. Enchufe la salida del control deslizante en la entrada *para* *Curve.PointAtParameter*. Ahora vemos un punto a lo largo de la hélice, representado por un porcentaje del control deslizante (0 en el punto de inicio, 1 en el punto final).

Con el punto de referencia creado, ahora comparamos la distancia desde el punto de referencia a los puntos originales que definen la hélice. Este valor de distancia impulsará la geometría y el color.

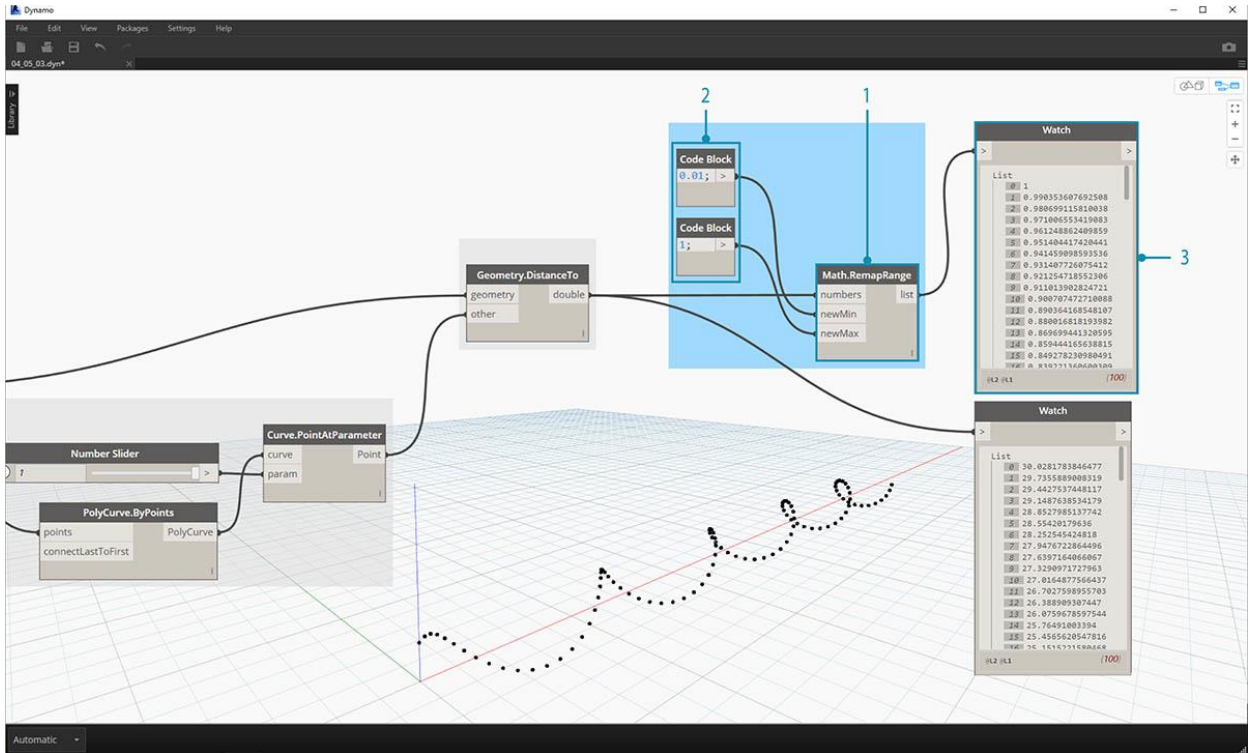
DARCO
DESDE 1988



1. **Geometry.DistanceTo:** conecta la salida *Curve.PointAtParameter* a la entrada. Conecte *Point.ByCoordinates* en la * entrada de geometría.
2. **Mirar:** El resultado resultante muestra una lista de distancias desde cada punto helicoidal hasta el punto de referencia a lo largo de la curva.

Nuestro siguiente paso es conducir los parámetros con la lista de distancias desde los puntos helicoidales hasta el punto de referencia. Usamos estos valores de distancia para definir los radios de una serie de esferas a lo largo de la curva. Para mantener las esferas del tamaño adecuado, debemos *reassignar* los valores de distancia.

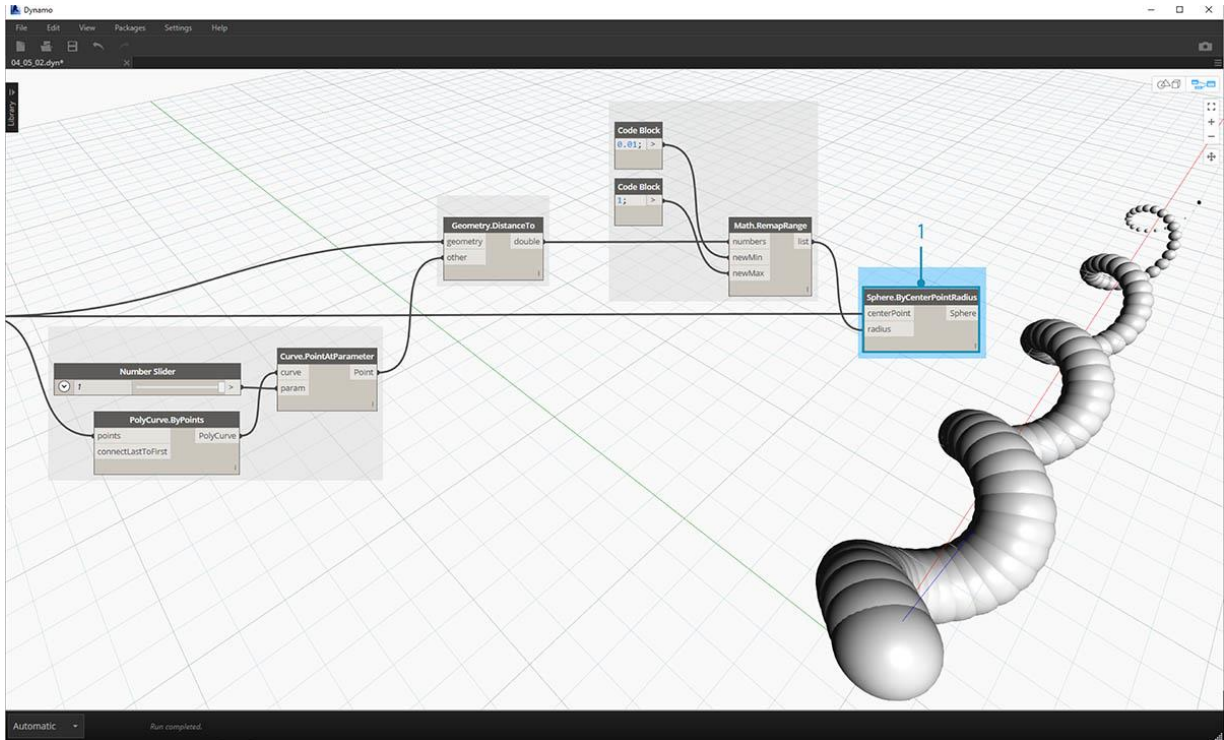
DARCO
DESDE 1988



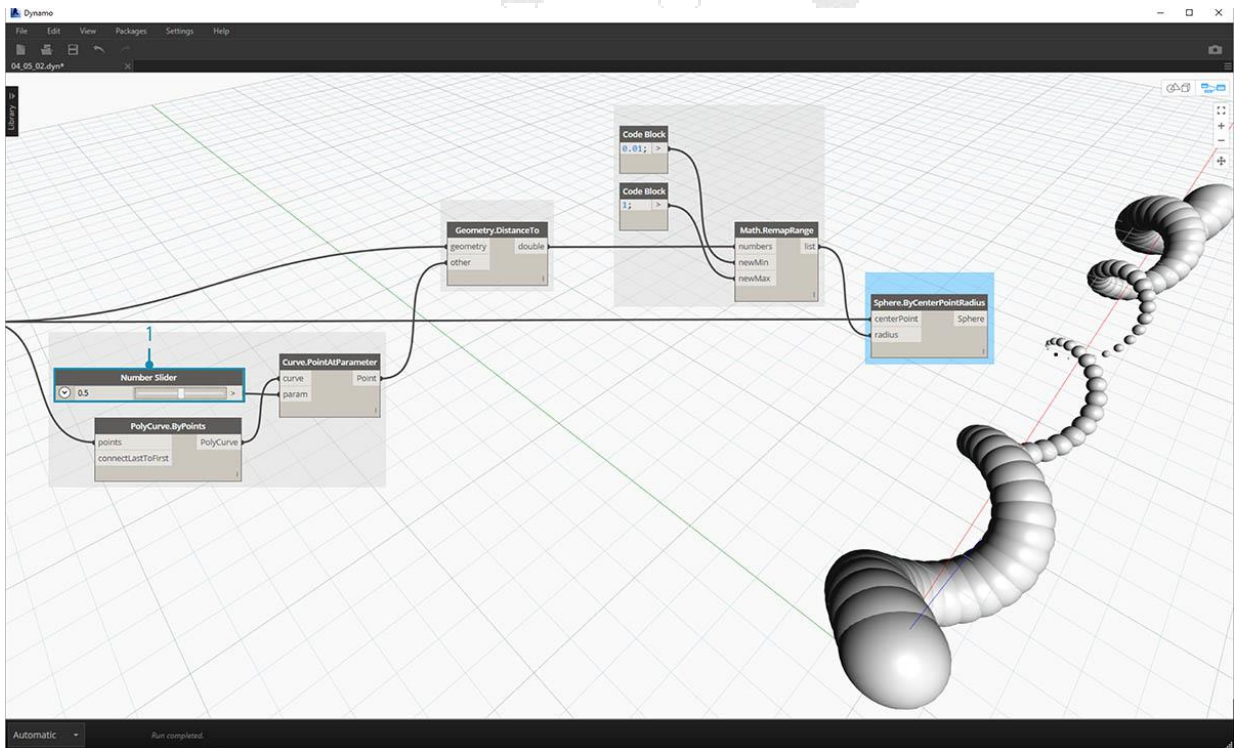
1. **Math.RemapRange:** Connect *Geometry.DistanceTo* salida a la entrada de números.
2. **Bloque de código:** conecte un bloque de código con un valor de 0.01 en la entrada *nueva de Min* y un bloque de código con un valor de 1 en la *nueva* entrada de *Max*.
3. **Mire:** conecte la salida *Math.RemapRange* en un nodo y la salida *Geometry.DistanceTo* en otra. Compare los resultados.

Este paso a reasignado la lista de distancia para que sea un rango más pequeño. Podemos editar los *nuevos* valores de *Min* y *NewMax* como *mejor* podamos. Los valores se reasignarán y tendrán la misma *proporción de distribución* en todo el dominio.

DARCO
DESDE 1988

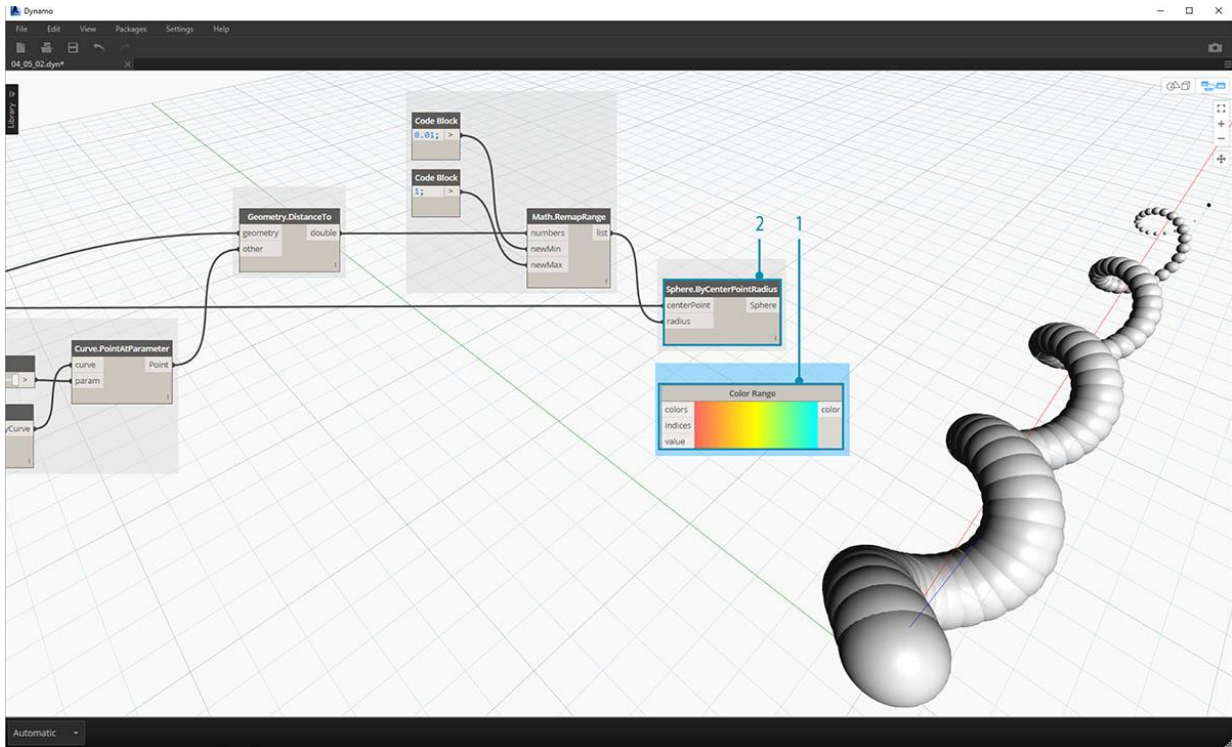


1. **Sphere.ByCenterPointRadius:** conecte la salida *Math.RemapRange* en la entrada de *radio* y la salida *Point.ByCordor* original en la entrada *centerPoint*.



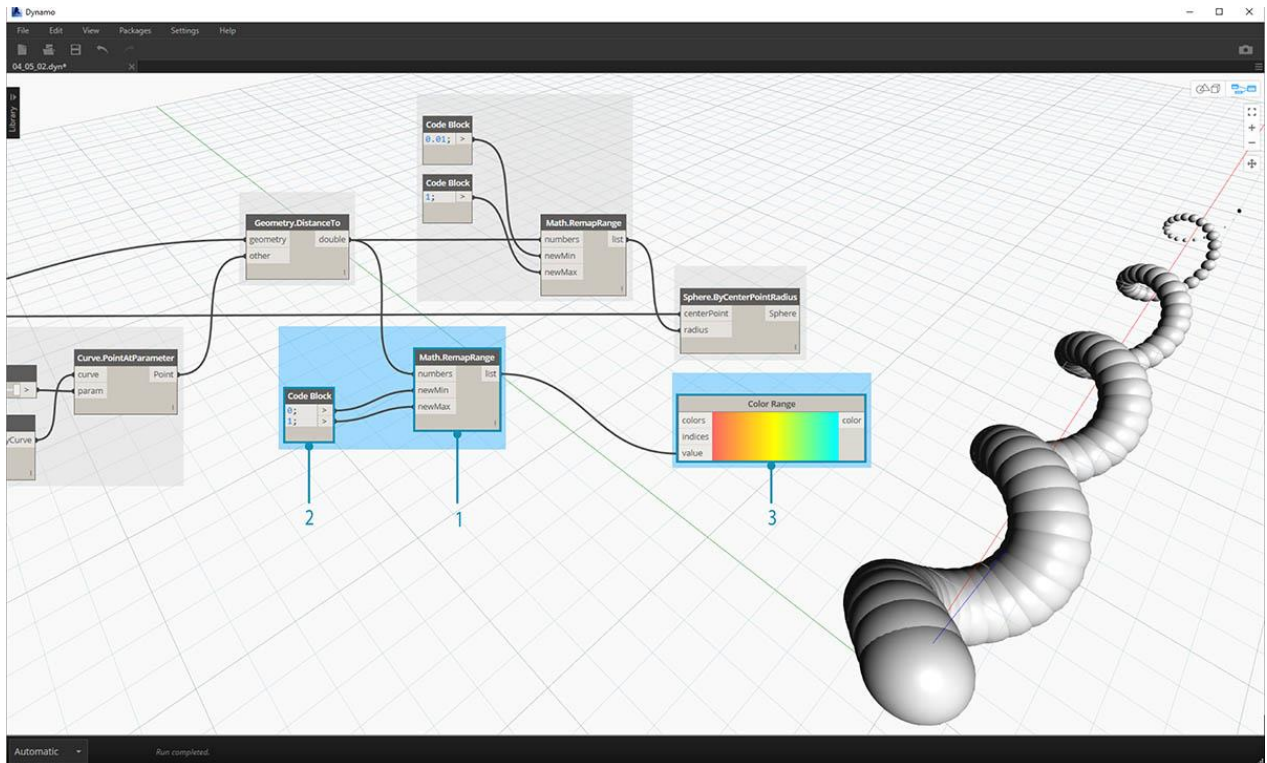
1. **Control deslizante numérico:** cambie el valor del deslizador numérico y observe el tamaño de la actualización de esferas. Ahora tenemos una plantilla paramétrica.

El tamaño de las esferas demuestra la matriz paramétrica definida por un punto de referencia a lo largo de la curva. Usemos el mismo concepto para el radio de la esfera para controlar su color.



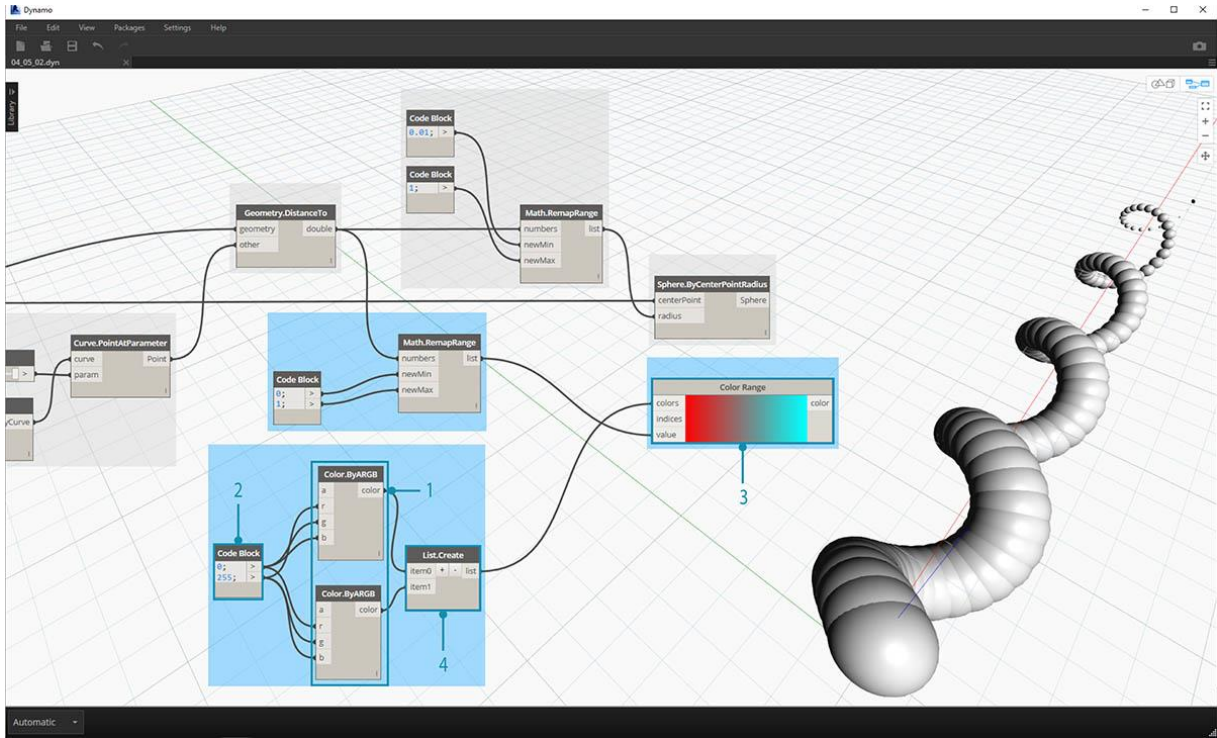
1. **Gama de colores:** agregue arriba el lienzo. Al pasar el puntero sobre la entrada de *valor*, notamos que los números solicitados están entre 0 y 1. Necesitamos reasignar los números desde la salida *Geometry.DistanceTo* para que sean compatibles con este dominio.
2. **Sphere.ByCenterPointRadius:** por el momento, deshabilitemos la vista previa en este nodo (clic *derecho*> *Vista previa*)

DARCO
DESDE 1988



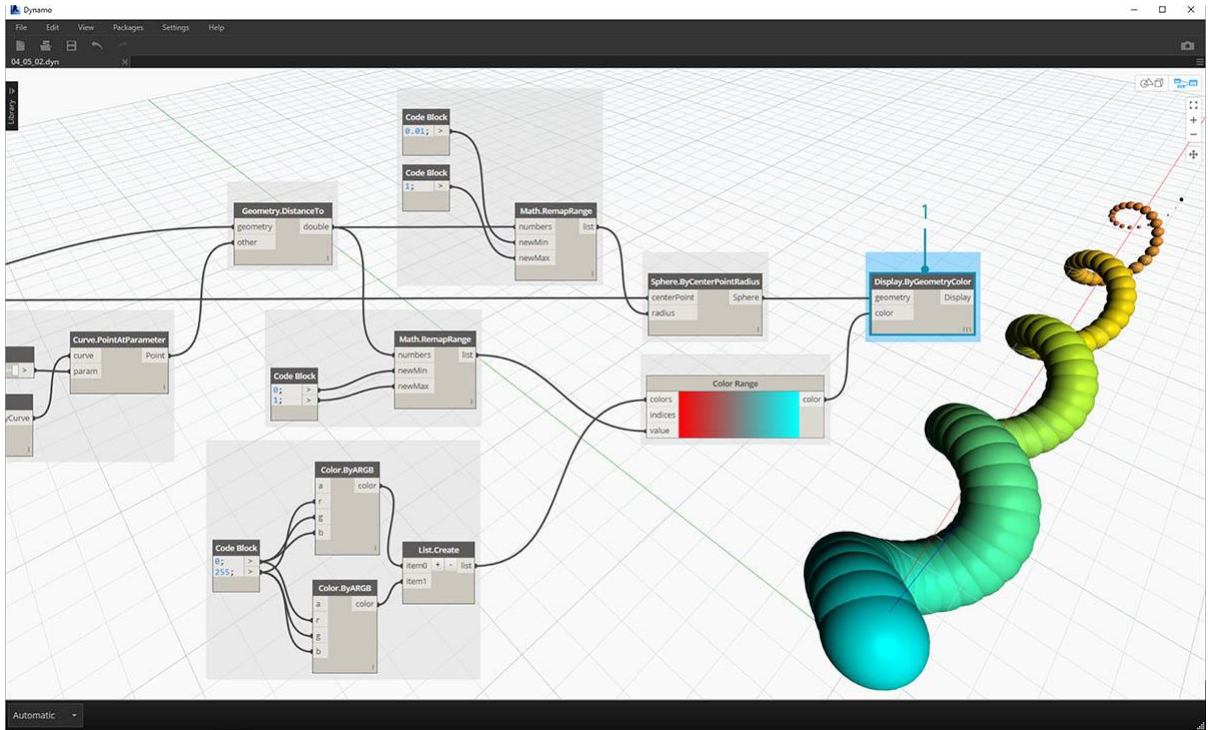
1. **Math.RemapRange:** Este proceso debería ser familiar. Conecte la salida *Geometry.DistanceTo* a la entrada de números.
2. **Bloque de código:** similar a un paso anterior, cree un valor de 0 para la entrada *nueva de Min* y un valor de 1 para la *nueva* entrada de *Max*. Tenga en cuenta que podemos definir dos salidas de un bloque de código en este caso.
3. **Gama de colores:** conecte la salida *Math.RemapRange* en la entrada de *valor*.

DARCO
 DESDE 1988

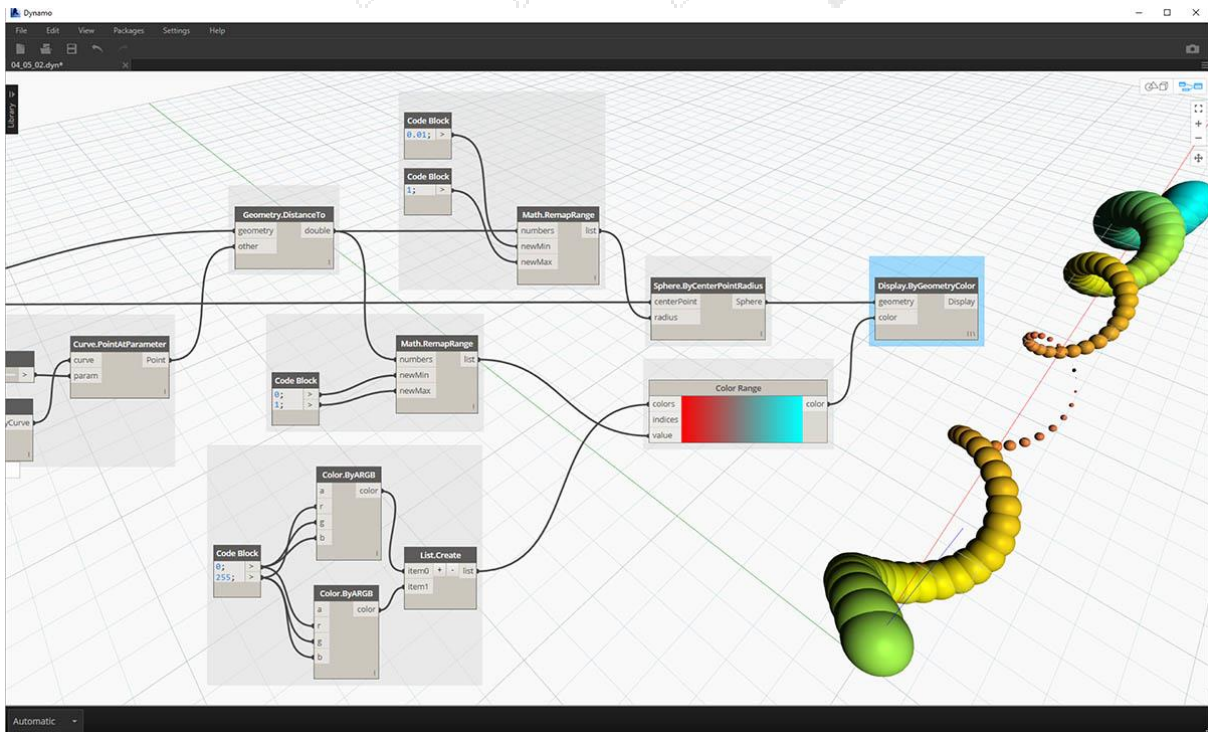


1. **Color.ByARGB:** esto es lo que haremos para crear dos colores. Si bien este proceso puede parecer incómodo, es lo mismo que los colores RGB en otro software, solo estamos usando programación visual para hacerlo.
2. **Bloque de código:** crea dos valores de 0 y 255. Conecte las dos salidas en las dos entradas *Color.ByARGB* de acuerdo con la imagen de arriba (o cree sus dos colores favoritos).
3. **Rango de color:** El *color* de entrada solicita una lista de colores. Necesitamos crear esta lista a partir de los dos colores creados en el paso anterior.
4. **List.Create:** fusiona los dos colores en una sola lista. Conecte la salida a la entrada de *colores* para el *rango de color*.

DARCO
DESDE 1988



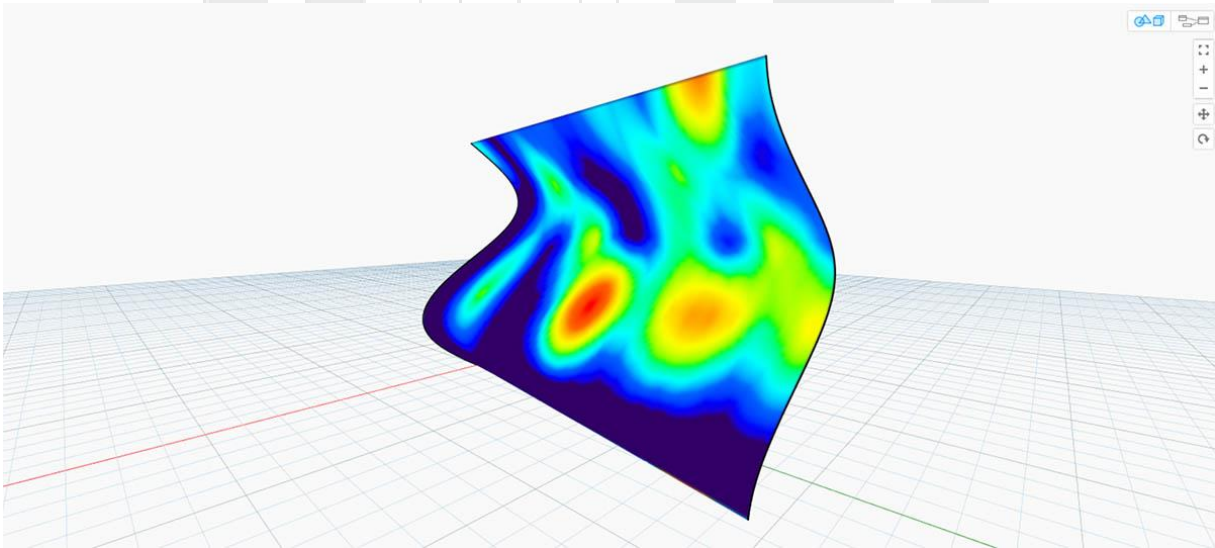
1. **Display.ByGeometryColor:** conecta *Sphere.ByCenterPointRadius* en la entrada de *geometría* y el *rango de color* en la entrada de *color*. Ahora tenemos un gradiente suave en el dominio de la curva.



Si cambiamos el valor del *deslizador numérico* de la definición anterior, los colores y los tamaños se actualizan. Los colores y el tamaño del radio están directamente relacionados en este caso: ¡ahora tenemos un enlace visual entre dos parámetros!

Color en las superficies

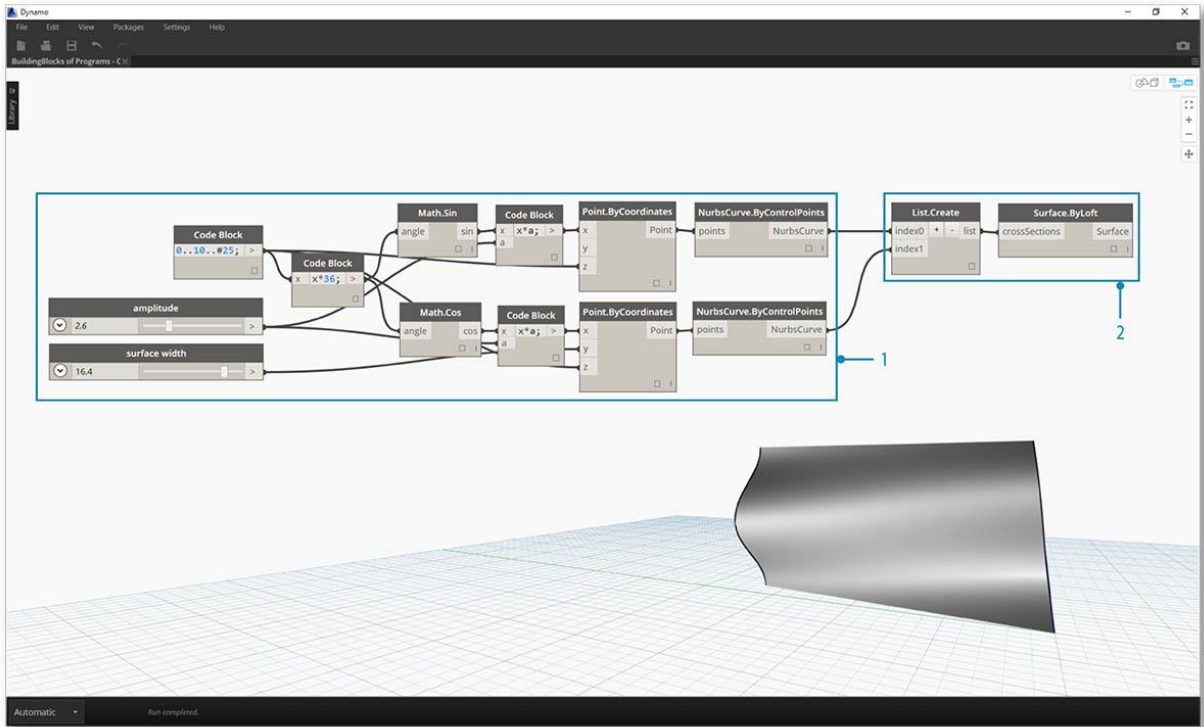
¡El nodo **Display!BySurfaceColors** nos da la capacidad de mapear datos a través de una superficie usando color! Esta funcionalidad presenta algunas posibilidades interesantes para visualizar datos obtenidos a través de análisis discretos como la energía solar y la proximidad. Aplicar color a una superficie en Dynamo es similar a aplicar una textura a un material en otros entornos de CAD. Demostremos cómo usar esta herramienta en el breve ejercicio a continuación.



Ejercicio de color en superficies

Descargue el archivo de ejemplo que acompaña a este ejercicio **Building Blocks of Programs - ColorOnSurface.zip**. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

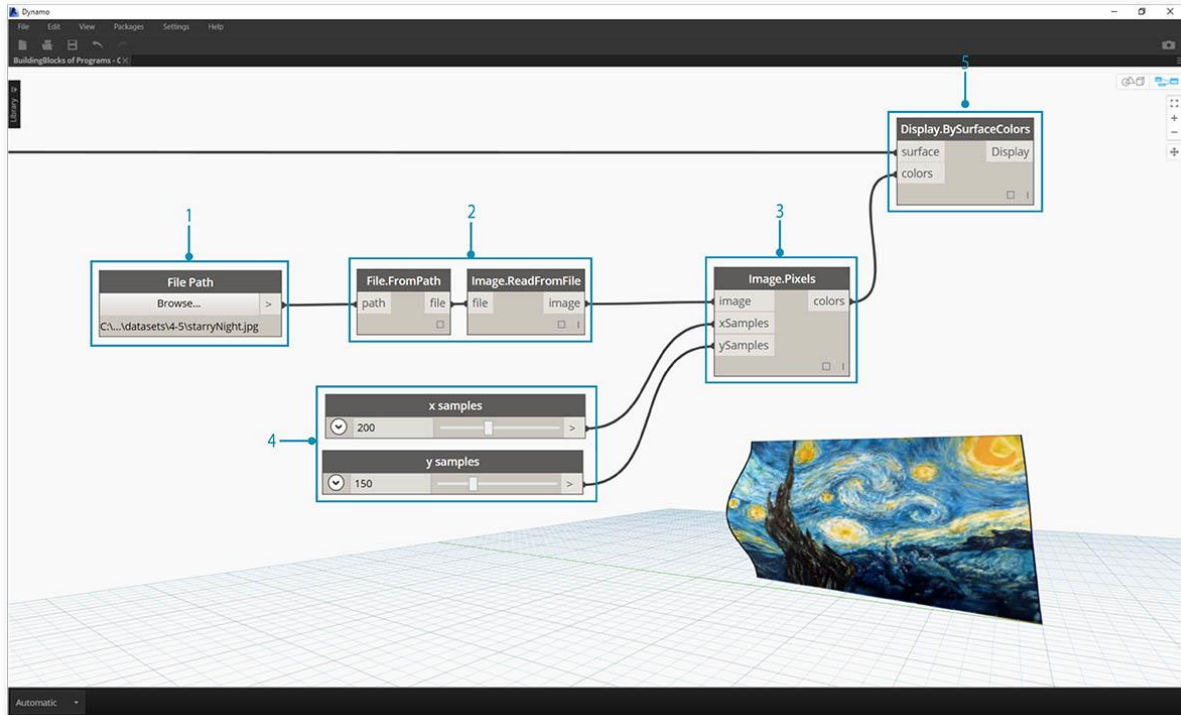
DARCO
DESDE 1988



En primer lugar, debemos crear (o hacer referencia) una superficie para usar como entrada para el nodo **Display.BySurfaceColors** . Para este ejemplo, estamos lofting entre una curva seno y coseno.

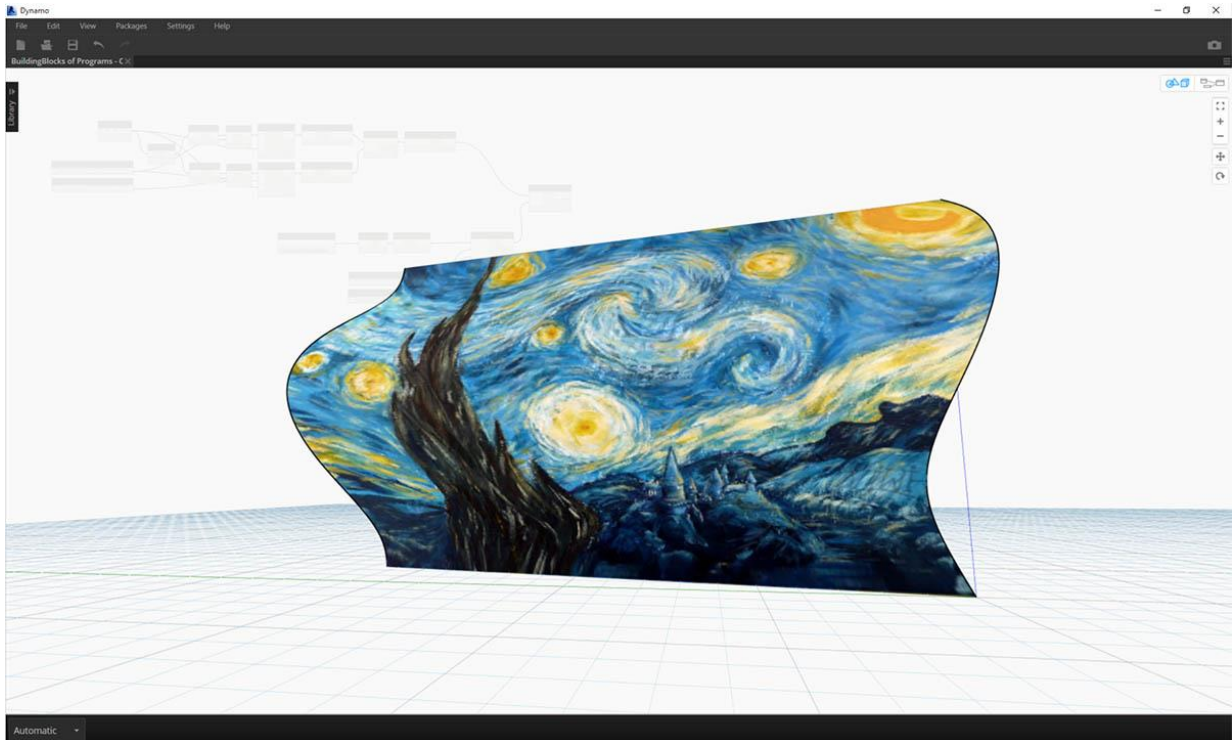
1. Este **grupo** de nodos está creando puntos a lo largo del eje Z y luego los desplaza en función de las funciones seno y coseno. Las dos listas de puntos se usan para generar curvas NURBS.
2. **Surface.ByLoft** : genera una superficie interpolada entre la lista de curvas NURBS.

DARCO
DESDE 1988



1. **Ruta de archivo:** seleccione un archivo de imagen para muestrear para datos de píxel en sentido descendente
2. use **File.FromPath** para convertir la ruta del archivo a un archivo y luego pase a **Image.ReadFromFile** para generar una imagen para el muestreo
3. **Image.Pixels:** ingresa una imagen y proporciona un valor de muestra para usar a lo largo de las dimensiones key de la imagen.
4. **Control deslizante:** proporciona valores de muestra para **Image.Pixels**
5. **Display.BySurfaceColors:** asigna una matriz de valores de color en toda la superficie a lo largo de X e Y, respectivamente

DARCO
DESDE 1988



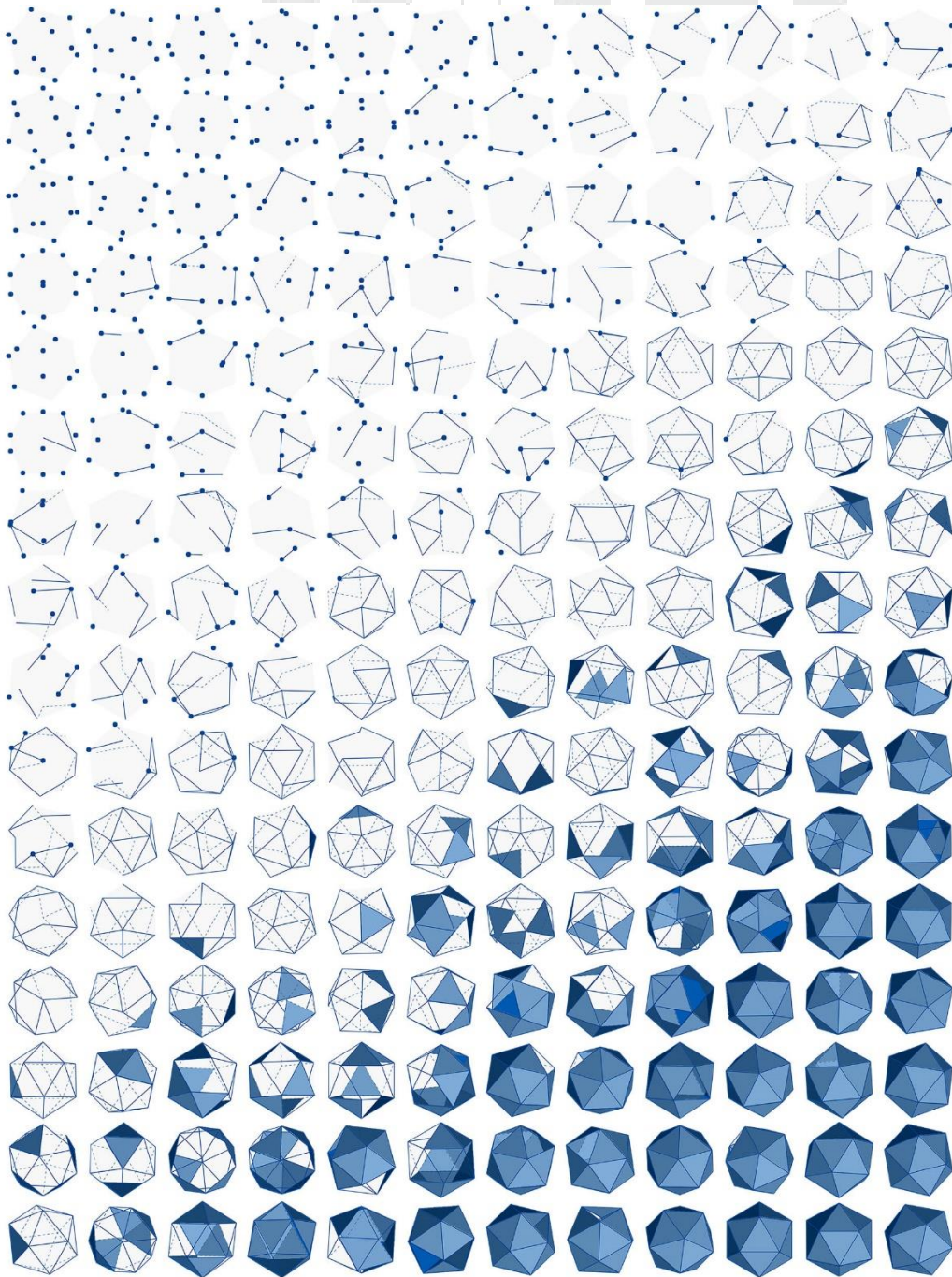
Vista previa de primer plano de la superficie de salida con resolución de 400x300 muestras



DARCO
DESDE 1988

GEOMETRÍA PARA EL DISEÑO COMPUTACIONAL

Como entorno de Programación Visual, Dynamo le permite crear la forma en que se procesan los datos. Los datos son números o texto, pero también lo es Geometry. Tal como lo entiende la Computadora, la Geometría, a veces llamada Geometría Computacional, es la información que podemos usar para crear modelos hermosos, complejos o basados en el rendimiento. Para hacerlo, debemos entender los pormenores de los diversos tipos de geometría que podemos usar.



Resumen de Geometría

La geometría es el lenguaje para el diseño. Cuando un lenguaje o entorno de programación tiene un kernel de geometría en su núcleo, podemos desbloquear las posibilidades para diseñar modelos precisos y robustos, automatizar rutinas de diseño y generar iteraciones de diseño con algoritmos.

Los básicos

La geometría, tradicionalmente definida, es el estudio de la forma, el tamaño, la posición relativa de las figuras y las propiedades del espacio. Este campo tiene una rica historia que se remonta a miles de años atrás. Con el advenimiento y la popularización de la computadora, obtuvimos una poderosa herramienta para definir, explorar y generar geometría. Ahora es muy fácil calcular el resultado de interacciones geométricas complejas, el hecho de que lo hagamos es casi transparente.



Si tiene curiosidad por ver qué tan diversa y compleja puede ser la geometría al usar la potencia de su computadora, haga una búsqueda rápida en la Web del Stanford Bunny, un modelo canónico que se usa para probar algoritmos.

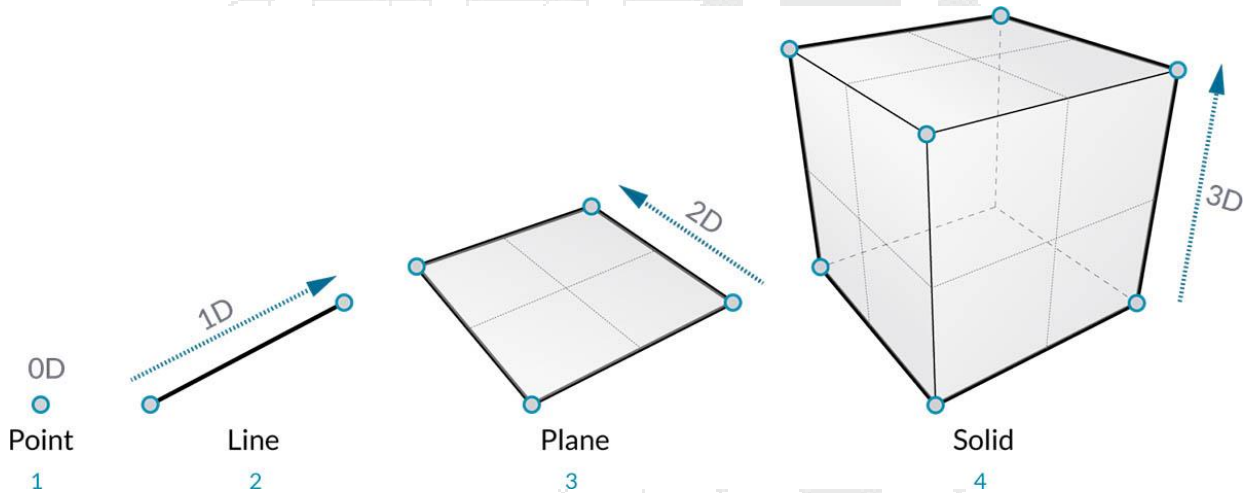
Comprender la geometría en el contexto de los algoritmos, la informática y la complejidad puede sonar desalentador; sin embargo, existen algunos principios clave, relativamente simples, que podemos establecer como fundamentales para comenzar a desarrollar aplicaciones más avanzadas:

1. Geometry is **Data**: para la computadora y Dynamo, un conejito no tan diferente de un número.
2. La geometría se basa en la **abstracción**: fundamentalmente, los elementos geométricos se describen mediante números, relaciones y fórmulas dentro de un sistema de coordenadas espaciales determinado.
3. La geometría tiene una **jerarquía**: los puntos se unen para hacer líneas, las líneas se unen para formar superficies, etc.
4. La geometría describe simultáneamente tanto **la Parte como el Todo**: cuando tenemos una curva, es tanto la forma como todos los puntos posibles a lo largo de ella.

En la práctica, estos principios significan que debemos ser conscientes de con qué estamos trabajando (qué tipo de geometría, cómo se creó, etc.) para poder componer, descomponer y recomponer de forma fluida las diferentes geometrías a medida que desarrollamos más complejos modelos.

Pasando por la jerarquía

Tomemos un momento para ver la relación entre las descripciones abstracta y jerárquica de Geometría. Debido a que estos dos conceptos están relacionados, pero no siempre son obvios al principio, podemos llegar rápidamente a un obstáculo conceptual una vez que comenzamos a desarrollar flujos de trabajo o modelos más profundos. Para empezar, usemos la dimensionalidad como un descriptor fácil de las "cosas" que modelamos. El número de dimensiones requeridas para describir una forma nos da una ventana de cómo la Geometría está organizada jerárquicamente.



1. Un **punto** (definido por coordenadas) no tiene ninguna dimensión, solo son números que describen cada coordenada
2. Una **línea** (definida por dos puntos) ahora tiene *una* dimensión: podemos "caminar" la línea ya sea hacia adelante (dirección positiva) o hacia atrás (dirección negativa)
3. Un **plano** (definido por dos líneas) tiene *dos* dimensiones: caminar más a la izquierda o más a la derecha ahora es posible
4. Un **cuadro** (definido por dos planos) tiene *tres* dimensiones: podemos definir una posición relativa hacia arriba o hacia abajo

La dimensionalidad es una forma conveniente de comenzar a categorizar la Geometría, pero no es necesariamente la mejor. Después de todo, no modelamos solo con Puntos, Líneas, Planos y Cajas, ¿y si quiero algo curvilíneo? Además, hay una categoría entera de tipos geométricos que son completamente abstractos, es decir. definen propiedades como la orientación, el volumen o las relaciones entre las partes. Realmente no podemos agarrar un Vector así que ¿cómo lo definimos en relación con lo que vemos en el espacio? Una categorización más detallada de la jerarquía geométrica debe acomodar la diferencia entre los Tipos abstractos o "Ayudantes", cada uno de los cuales podemos agrupar por lo que ayudan a hacer y los tipos que ayudan a describir la forma de los elementos del modelo.

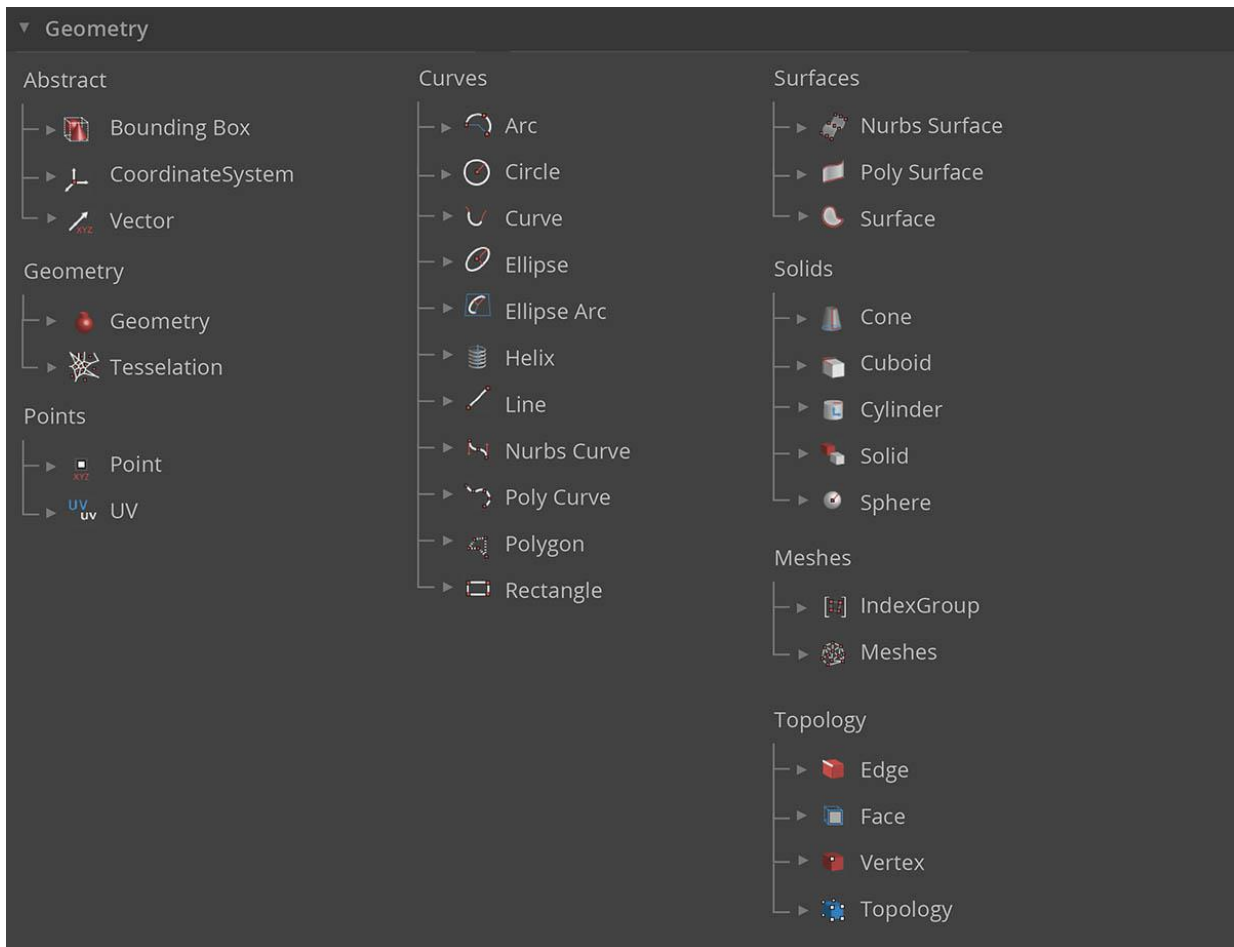
DESDE 1988

Data Type Hierarchy							
Abstract Types			Geometry Types				
Defines Location + Orientation	Defines Position + Volume	Defines Relationships	Model Elements				
Vector	Bounding Box	Topology	Point	Curve	Surface	Solid	Mesh
<ul style="list-style-type: none"> • Vector • Plane • Coordinate System 	<ul style="list-style-type: none"> • Bounding Box 	<ul style="list-style-type: none"> • Vertex • Edge • Face 	<ul style="list-style-type: none"> • XYZ Coordinate • UV Coordinate 	<ul style="list-style-type: none"> • Line • Polygon • Arc • Circle • Ellipse • NURBS Curve • PolyCurve 	<ul style="list-style-type: none"> • NURBS Surface • Polysurface 	<ul style="list-style-type: none"> • Cuboid • Sphere • Cone • Cylinder 	<ul style="list-style-type: none"> • Mesh

Geometría en Dynamo Studio

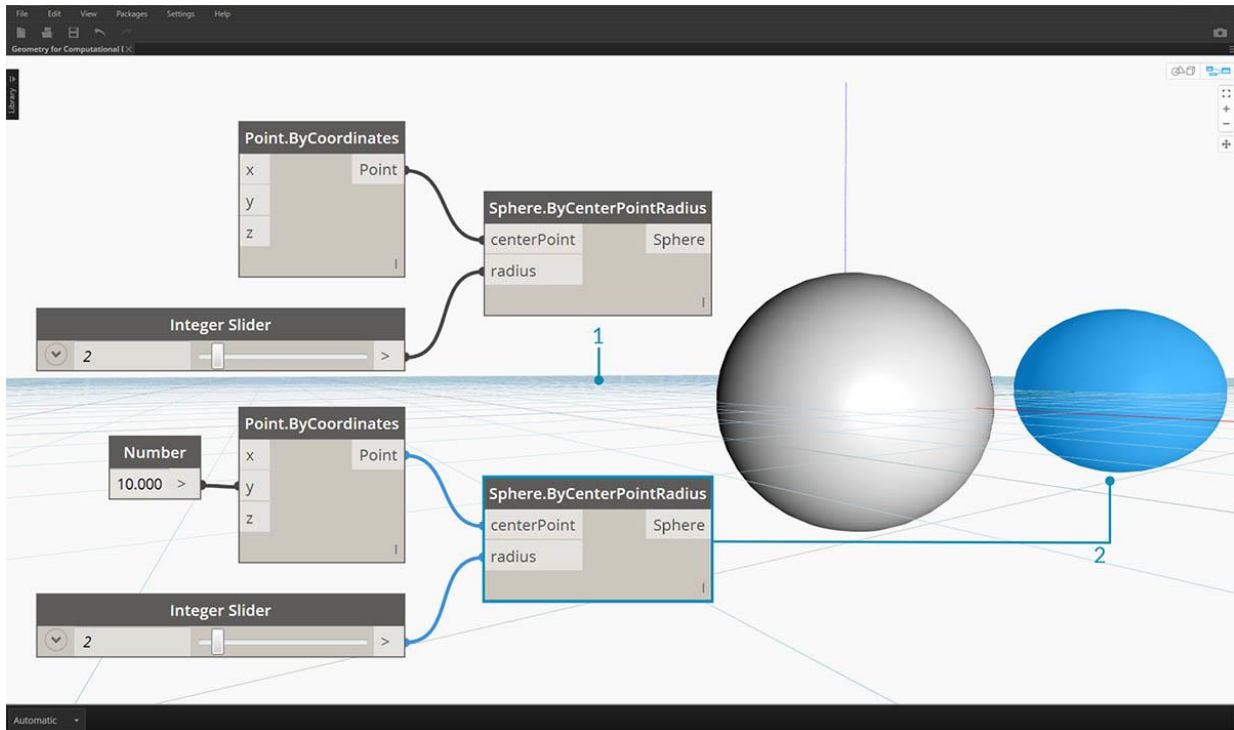
Entonces, ¿qué significa esto para usar Dynamo? Comprender los tipos de Geometría y cómo se relacionan nos permitirá navegar por la colección de **Nodos de Geometría** disponibles en la Biblioteca. Los Nodos de Geometría están organizados alfabéticamente en lugar de jerárquicamente, aquí se muestran de forma similar a su diseño en la interfaz de Dynamo.

DARCO
DESDE 1988



Además, hacer modelos en Dynamo y conectar la vista previa de lo que vemos en la Vista previa en segundo plano con el flujo de datos en nuestro gráfico debería hacerse más intuitivo a lo largo del tiempo.

DARCO
DESDE 1988



1. Tenga en cuenta el sistema de coordenadas supuesto prestado por la cuadrícula y los ejes de color
2. Los nodos seleccionados representarán la geometría correspondiente (si el nodo crea geometría) en el fondo, el color de resaltado

Descargue el archivo de ejemplo que acompaña a esta imagen Geometry for Computational Design - Geometry Overview.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

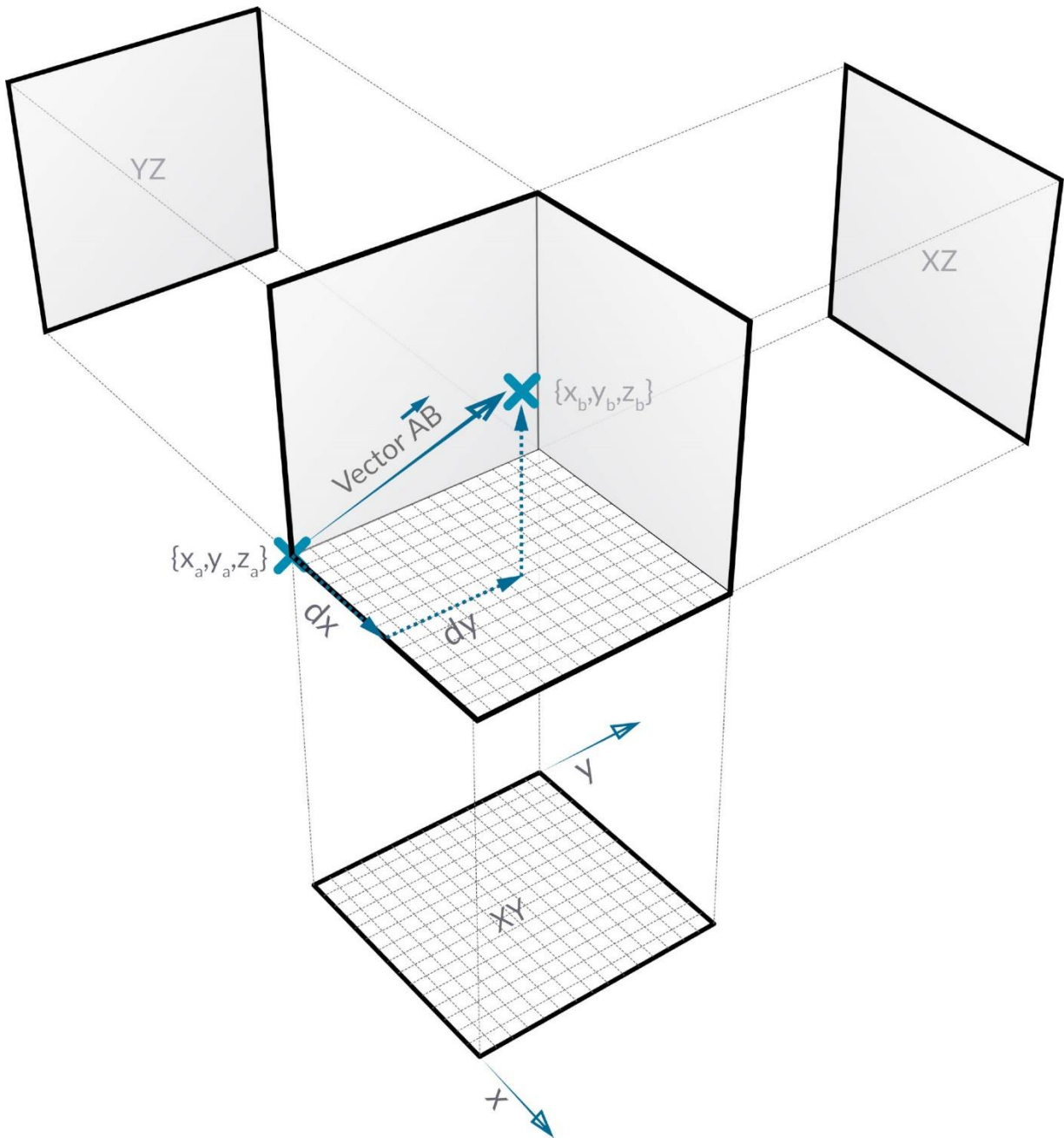
Yendo más allá con la geometría

La creación de modelos en Dynamo no se limita a lo que podemos generar con Nodos. Estas son algunas formas clave para llevar su proceso al siguiente nivel con Geometría:

1. Dynamo le permite importar archivos; intente usar un CSV para nubes de puntos o SAT para traer superficies
2. Al trabajar con Revit, podemos hacer referencia a los elementos de Revit para usar en Dynamo
3. Dynamo Package Manager ofrece funcionalidad adicional para operaciones y tipos de geometría extendida; consulte el paquete Mesh Toolkit

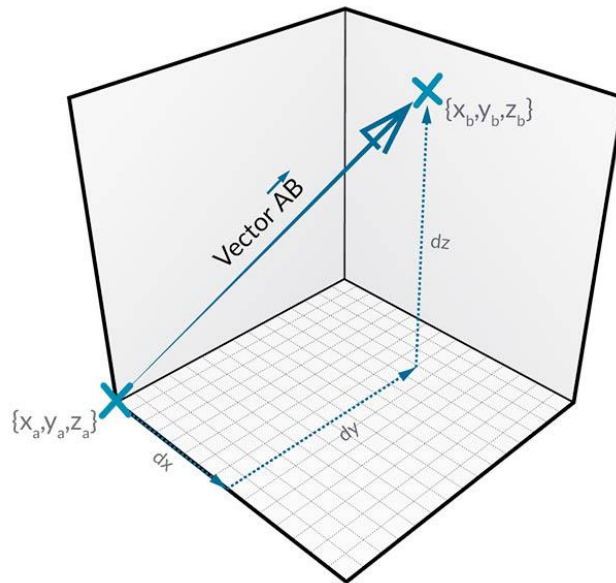
Vectores, planos y sistemas de coordenadas

Vectores, planos y sistemas de coordenadas conforman el grupo primario de tipos de geometría abstracta. Nos ayudan a definir la ubicación, la orientación y el contexto espacial para otra geometría que describe formas. Si digo que estoy en la ciudad de Nueva York en la calle 42 y Broadway (sistema de coordenadas), parado en el nivel de la calle (plano), mirando al norte (Vector), acabo de utilizar estos "ayudantes" para definir dónde estoy. Lo mismo ocurre con un producto de caja de teléfono o un rascacielos: necesitamos este contexto para desarrollar nuestro modelo.



¿Qué es un Vector?

Un vector es una cantidad geométrica que describe la dirección y la magnitud. Los vectores son abstractos; es decir, ellos representan una cantidad, no un elemento geométrico. Los vectores se pueden confundir fácilmente con los puntos porque ambos están compuestos por una lista de valores. Sin embargo, hay una diferencia clave: los puntos describen una posición en un sistema de coordenadas dado mientras que los vectores describen una diferencia relativa en la posición que es lo mismo que decir "dirección".

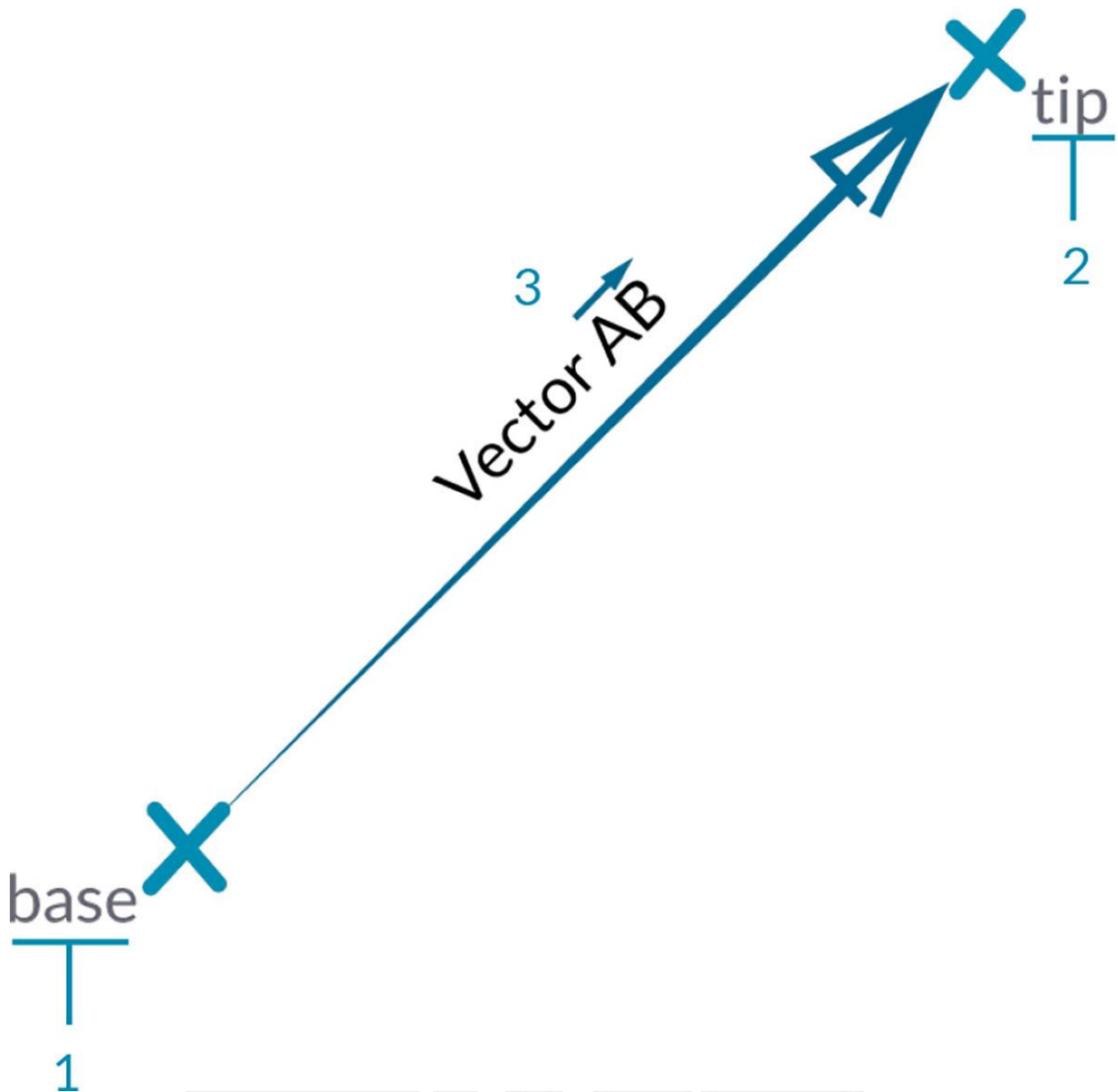


$$\text{Vector } \vec{AB} = \{d_x, d_y, d_z\} = \{x_b - x_a, y_b - y_a, z_b - z_a\}$$

Si la idea de la diferencia relativa es confusa, piense en el Vector AB como "Estoy de pie en el Punto A, mirando hacia el Punto B." La dirección, desde aquí (A) hasta allí (B), es nuestro Vector.

Desglosando los vectores en sus partes utilizando la misma notación AB:

DARCO
DESDE 1988

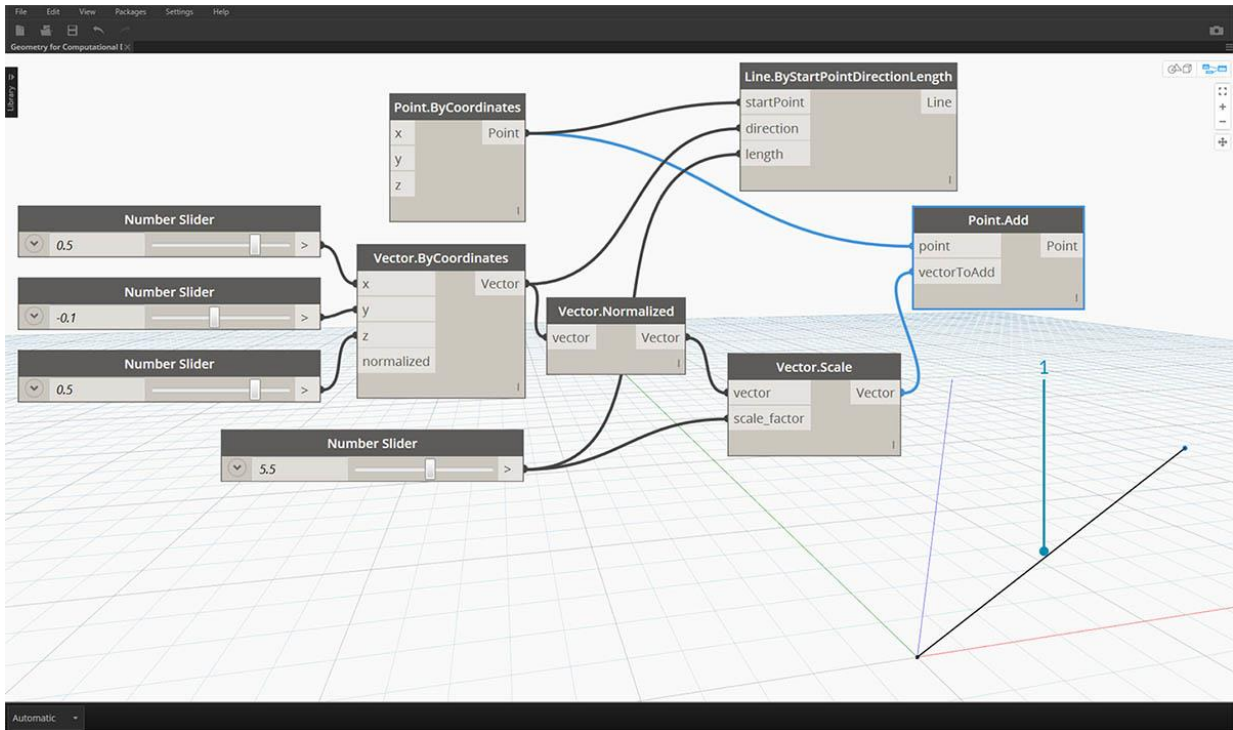


1. El **punto de inicio** del vector se llama **Base**.
2. El **punto final** del vector se llama **punta** o el **sentido**.
3. Vector AB no es lo mismo que Vector BA - que apuntaría en la dirección opuesta.

Si alguna vez necesitas un alivio cómico con respecto a los vectores (y su definición abstracta), mira la comedia clásica Airplane y escucha la línea de mejillas escritas frecuentemente:

Roger, Roger. ¿Cuál es nuestro vector, Victor?

Los vectores son un componente clave para nuestros modelos en Dynamo. Tenga en cuenta que, debido a que están en la categoría Resumen de "Ayudantes", cuando creamos un Vector, no veremos nada en la Vista previa de fondo.

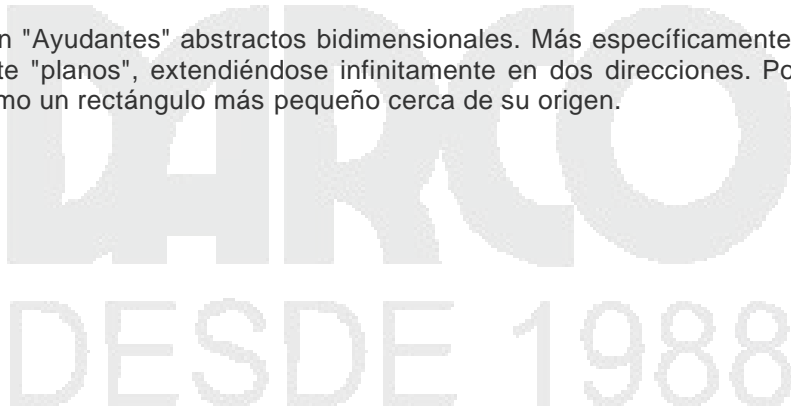


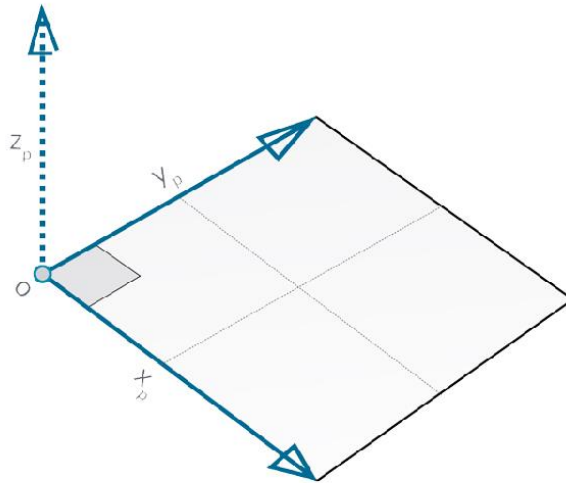
1. Podemos usar una línea como soporte para una vista previa de Vector.

Descargue el archivo de ejemplo que acompaña a esta imagen [Geometry for Computational Design - Vectors.dyn](#). Se puede encontrar una lista completa de archivos de ejemplo en el [Dataset](#).

¿Qué es un plano?

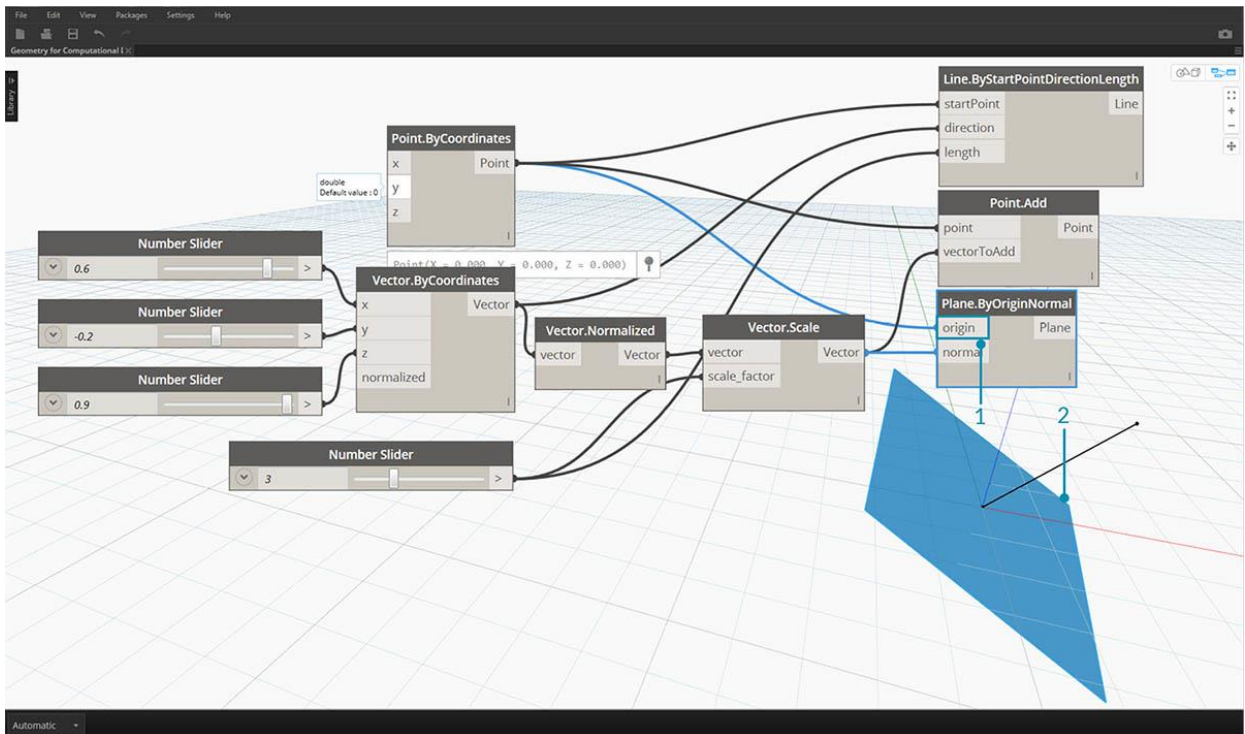
Los aviones son "Ayudantes" abstractos bidimensionales. Más específicamente, los planos son conceptualmente "planos", extendiéndose infinitamente en dos direcciones. Por lo general, se representan como un rectángulo más pequeño cerca de su origen.





Tal vez piense: "¡Espera! ¡Origen! Eso suena como un Sistema de coordenadas ... ¡como el que utilizo para modelar en mi software de CAD!"

¡Y tienes razón! La mayoría del software de modelaje aprovechan los planos de construcción o "niveles" para definir un contexto local de dos dimensiones para proyectar en. XY, XZ, YZ -o- Norte, Sudeste, Plan puede sonar más familiar. Todos estos son Planos, que definen un contexto infinito "plano". Los aviones no tienen profundidad, pero sí nos ayudan a describir la dirección: cada plano tiene un origen, una dirección X, una dirección Y y una dirección Z (arriba).



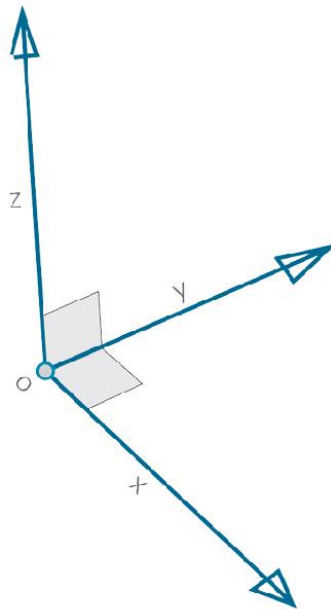
1. Aunque son abstractos, los aviones tienen una posición de origen para que podamos ubicarlos en el espacio.
2. En Dynamo, los aviones se representan en la Vista previa de fondo.

Descargue el archivo de ejemplo que acompaña a esta imagen **Geometry for Computational Design - Planes. Dyn**. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

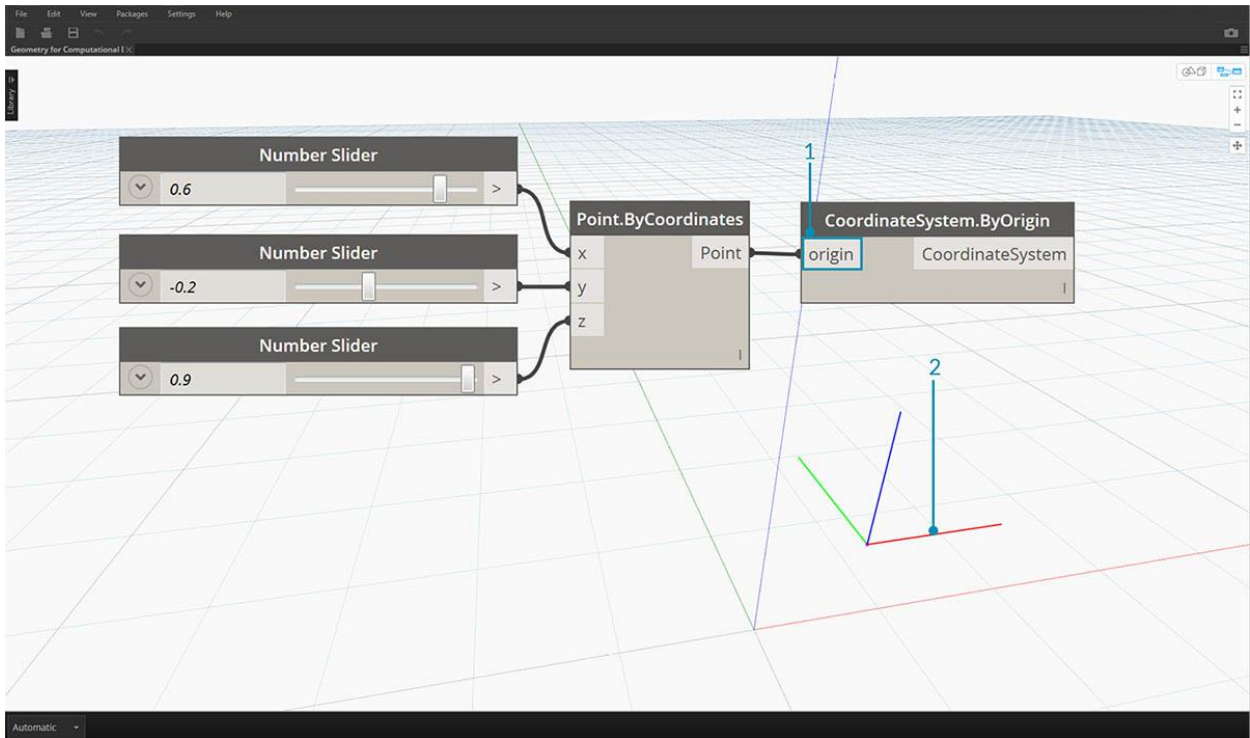
¿Qué es un sistema de coordenadas?

Si nos sentimos cómodos con Planes, estamos a un paso de entender Sistemas de coordenadas. Un plano tiene todas las mismas partes que un sistema de coordenadas, siempre que sea un sistema de coordenadas estándar "euclidiano" o "XYZ".

Sin embargo, existen otros sistemas de coordenadas alternativos, como cilíndricos o esféricos. Como veremos en secciones posteriores, los Sistemas de coordenadas también se pueden aplicar a otros tipos de Geometría para definir una posición en esa geometría.



Agregue sistemas de coordenadas alternativos: cilíndricos, esféricos

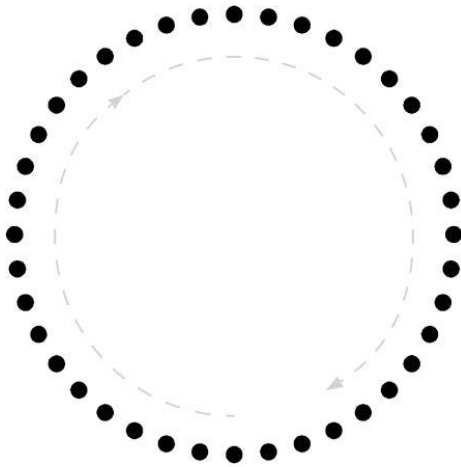


1. Aunque son abstractos, los Sistemas de coordenadas también tienen una posición de origen para que podamos ubicarlos en el espacio.
2. En Dynamo, los sistemas de coordenadas se representan en la vista previa de fondo como un punto (origen) y las líneas que definen los ejes (X es roja, Y es verde y Z es azul siguiendo la convención).

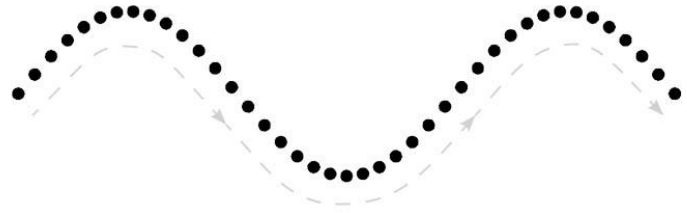
Descargue el archivo de ejemplo que acompaña a esta imagen Geometry for Computational Design - Coordinate System.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

Puntos

Si la geometría es el lenguaje de un modelo, los puntos son el alfabeto. Los puntos son la base sobre la que se crea la otra geometría: necesitamos al menos dos puntos para crear una curva, necesitamos al menos tres puntos para hacer un polígono o una cara de malla, y así sucesivamente. Definir la posición, el orden y la relación entre los puntos (prueba una función sinusoidal) nos permite definir la geometría de orden superior como cosas que reconocemos como círculos o curvas.



1



2

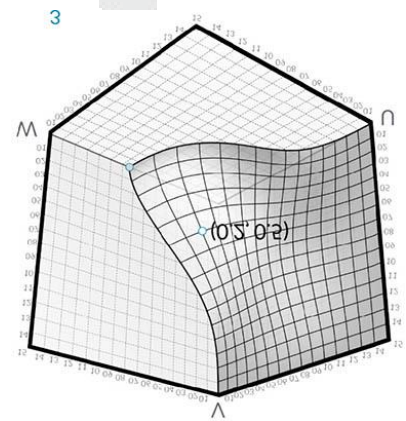
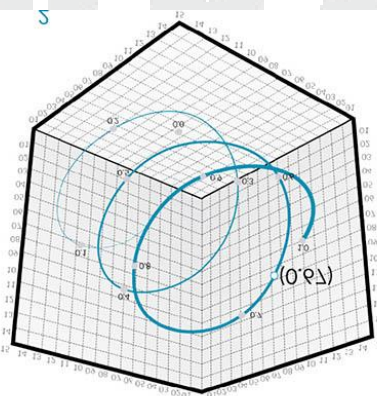
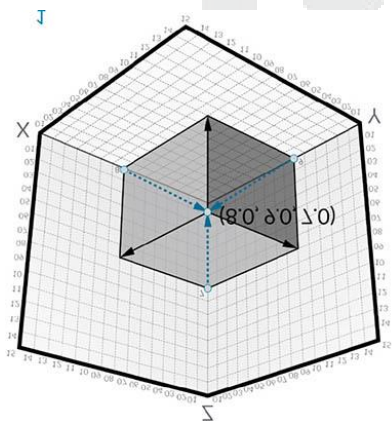
1. Un círculo usando las funciones $x=r*\cos(t)$ y $y=r*\sin(t)$
2. Una curva sinusoidal que usa las funciones $x=r*\cos(t)$ y $y=r*\sin(t)$

¿Qué es un punto?

Un punto se define por nada más que uno o más valores llamados coordenadas. Cuantos valores de coordenadas necesitamos para definir el punto depende del sistema de coordenadas o del contexto en el que reside. El tipo más común de Punto en Dynamo existe en nuestro Sistema de Coordenadas Mundiales tridimensionales y tiene tres coordenadas [X, Y, Z].

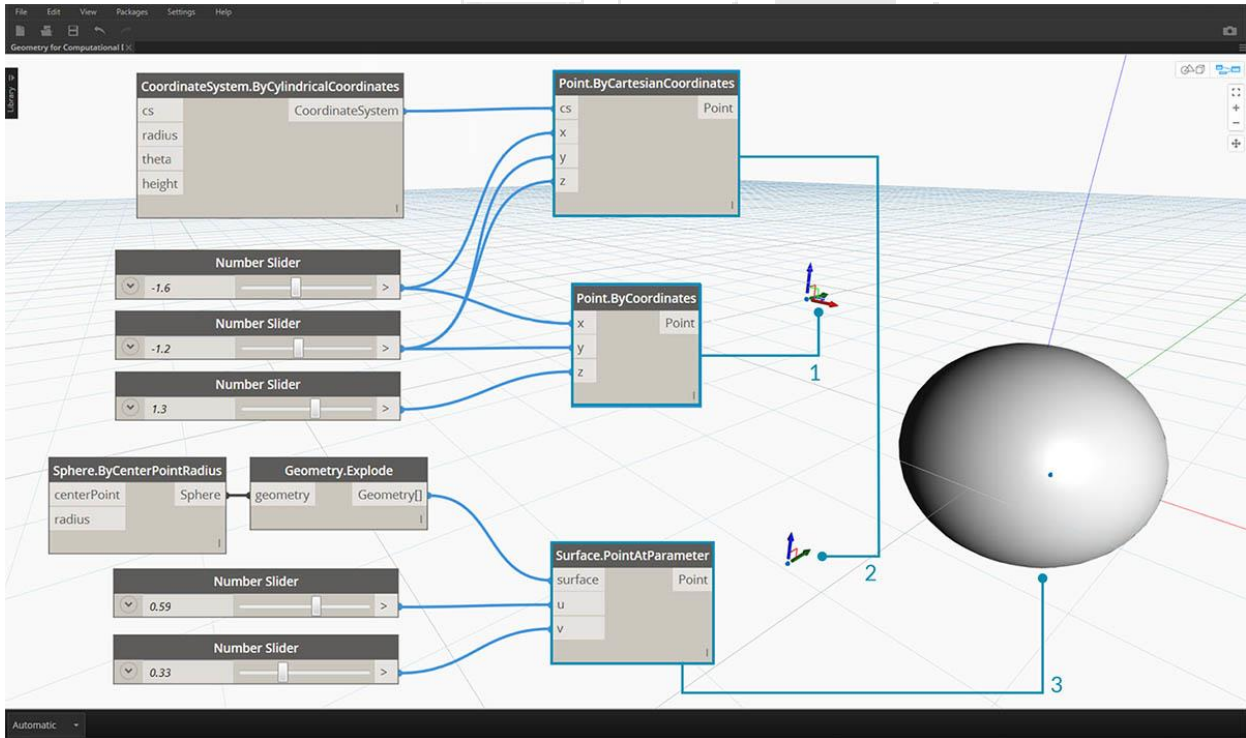
Señalar como coordenadas

Los puntos también pueden existir en un sistema de coordenadas bidimensional. La convención tiene una notación de letras diferente dependiendo del tipo de espacio con el que trabajemos: podríamos estar usando [X, Y] en un plano o [U, V] si estamos en una superficie.



1. Un punto en el sistema de coordenadas euclidianas: [X, Y, Z]
2. Un punto en un sistema de coordenadas de parámetros de curva: [t]
3. Un punto en un sistema de coordenadas de parámetros de superficie: [U, V]

Aunque parezca contra intuitivo, los parámetros tanto para curvas como para superficies son continuos y se extienden más allá del borde de la geometría dada. Dado que las formas que definen el Espacio de parámetros residen en un Sistema de coordenadas mundiales tridimensionales, siempre podemos traducir una Coordenada paramétrica en una Coordenada "Mundial". El punto [0.2, 0.5] en la superficie, por ejemplo, es el mismo que el punto [1.8, 2.0, 4.1] en coordenadas mundiales.

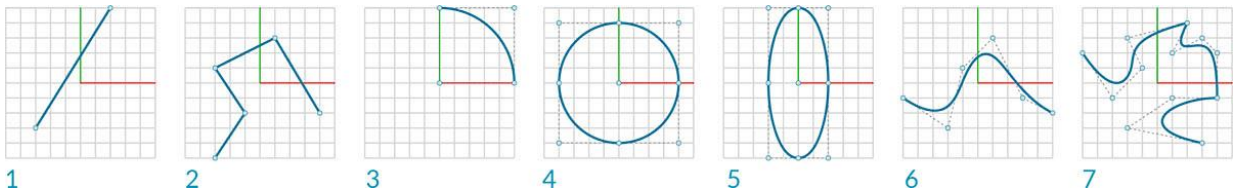


1. Punto en supuestas Coordenadas XYZ Mundiales
2. Punto relativo a un Sistema de Coordenadas dado (Cilíndrico)
3. Point como UV Coordinate en una superficie

Descargue el archivo de ejemplo que acompaña a esta imagen Geometry for Computational Design - Points.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

Curvas

Las curvas son el primer tipo de datos geométricos que hemos cubierto que tienen un conjunto más familiar de propiedades descriptivas de forma: ¿Qué tan curvilíneas o rectas? ¿Qué tan largo o corto? Y recuerde que los puntos siguen siendo nuestros bloques de construcción para definir cualquier cosa, desde una línea hasta una spline y todos los tipos de curvas intermedias.

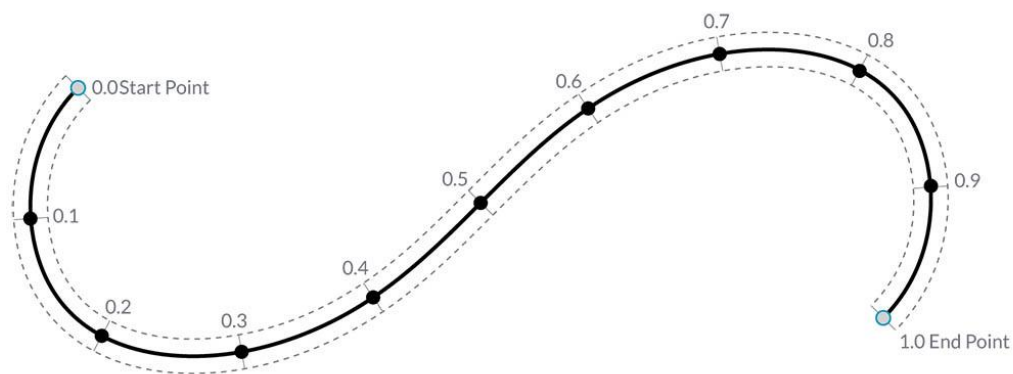


1. Línea
2. Polilínea

3. Arco
4. Círculo
5. Elipse
6. Curva NURBS
7. Polycurve

¿Qué es una curva?

El término **Curve** generalmente es un comodín para todo tipo de formas curvas (incluso rectas). La Curva de capital "C" es la categorización principal para todos los tipos de formas: líneas, círculos, splines, etc. Más técnicamente, una curva describe cada punto posible que se puede encontrar al ingresar "t" en una colección de funciones, que pueden rango de simple ($x = -1.26*t$, $y = t$) a funciones que involucran cálculo. No importa con qué tipo de Curva trabajemos, este **parámetro** llamado "t" es una propiedad que podemos evaluar. Además, independientemente del aspecto de la forma, todas las Curvas también tienen un punto de inicio y un punto final, que casualmente coinciden con los valores t mínimos y máximos utilizados para crear la Curva. Esto también nos ayuda a entender su direccionalidad.



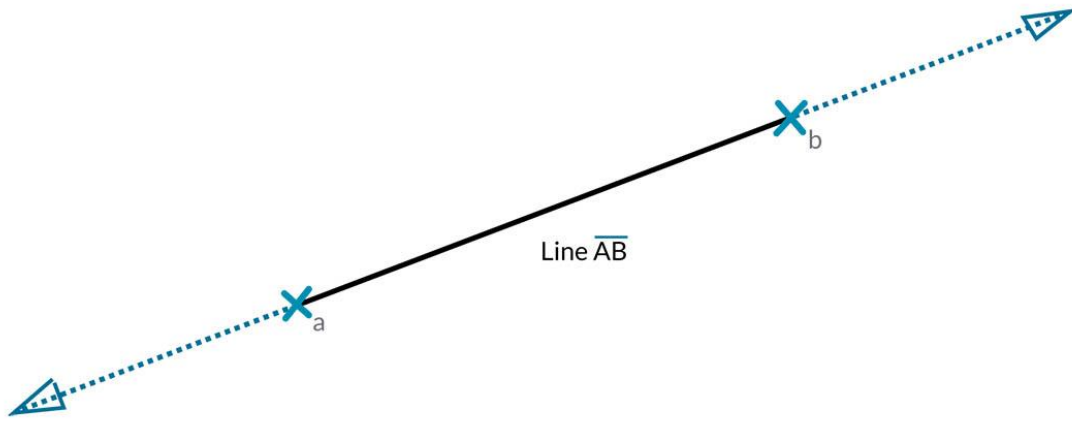
Es importante tener en cuenta que Dynamo supone que el dominio de los valores "t" para una Curva se entiende que es de 0.0 a 1.0.

Todas las curvas también poseen una serie de propiedades o características que se pueden utilizar para describirlas o analizarlas. Cuando la distancia entre los puntos inicial y final es cero, la curva está "cerrada". Además, cada curva tiene un número de puntos de control, si todos estos puntos están ubicados en el mismo plano, la curva es "plana". Algunas propiedades se aplican a la curva como un todo, mientras que otras solo se aplican a puntos específicos a lo largo de la curva. Por ejemplo, la planaridad es una propiedad global, mientras que un vector tangente en un valor t dado es una propiedad local.

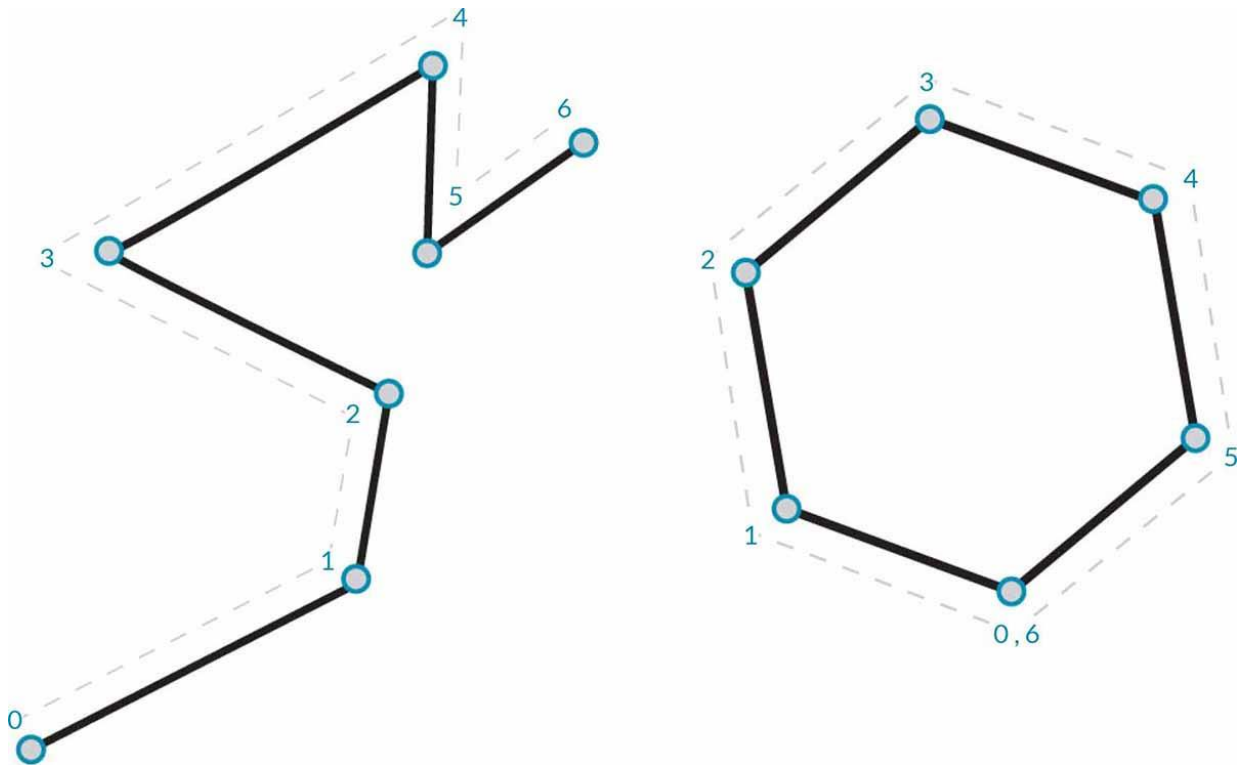
Líneas

Las líneas son la forma más simple de curvas. Puede que no se vean curvilíneos, pero de hecho son curvas, simplemente sin curvatura. Hay algunas formas diferentes de crear Líneas, siendo la

más intuitiva desde el Punto A hasta el Punto B. La forma de la Línea AB se dibujará entre los puntos, pero matemáticamente se extiende infinitamente en ambas direcciones.

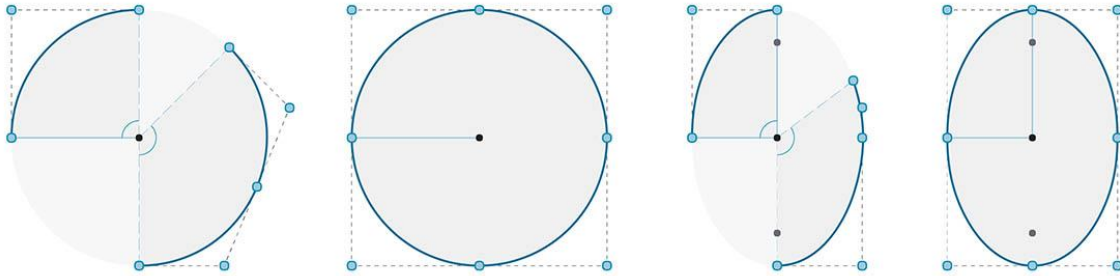


Cuando conectamos dos líneas juntas, tenemos una **poli línea**. Aquí tenemos una representación directa de lo que es un punto de control. La edición de cualquiera de estas ubicaciones de puntos cambiará la forma de la Poli línea. Si Polyline está cerrado, tenemos un Polígono. Si las longitudes de borde del polígono son todas iguales, se describe como regular.



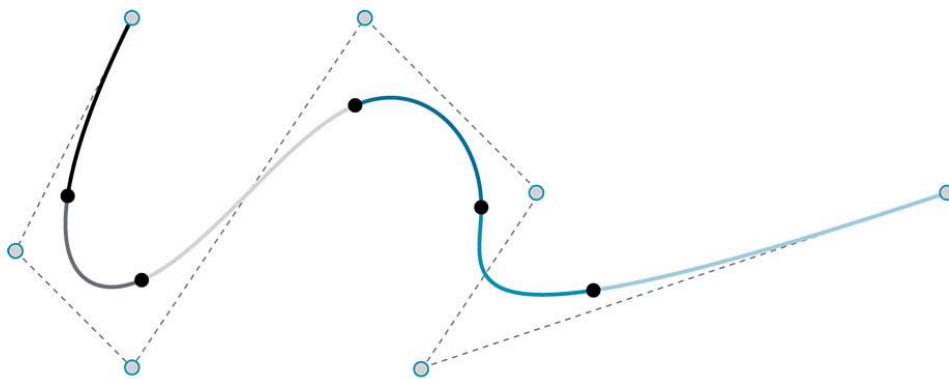
Arcos, círculos, arcos de elipse y elipses

A medida que agregamos más complejidad a las funciones paramétricas que definen una forma, podemos dar un paso más allá de una línea para crear un **arco**, **círculo**, **arco elíptico** o **elipse** describiendo uno o dos radios. Las diferencias entre la versión de Arc y Circle o Ellipse son solo si la forma está cerrada o no.



NURBS + Polycurves

NURBS (non-uniform rational B-spline, Splines de base racional no uniformes) son representaciones matemáticas que pueden modelar con precisión cualquier forma desde una línea bidimensional simple, círculo, arco o rectángulo hasta la Curva orgánica tridimensional de forma libre más compleja. Debido a su flexibilidad (relativamente pocos puntos de control, pero interpolación uniforme basada en la configuración de Grado) y precisión (vinculados por una sólida matemática), los modelos NURBS se pueden utilizar en cualquier proceso, desde ilustración y animación hasta fabricación.



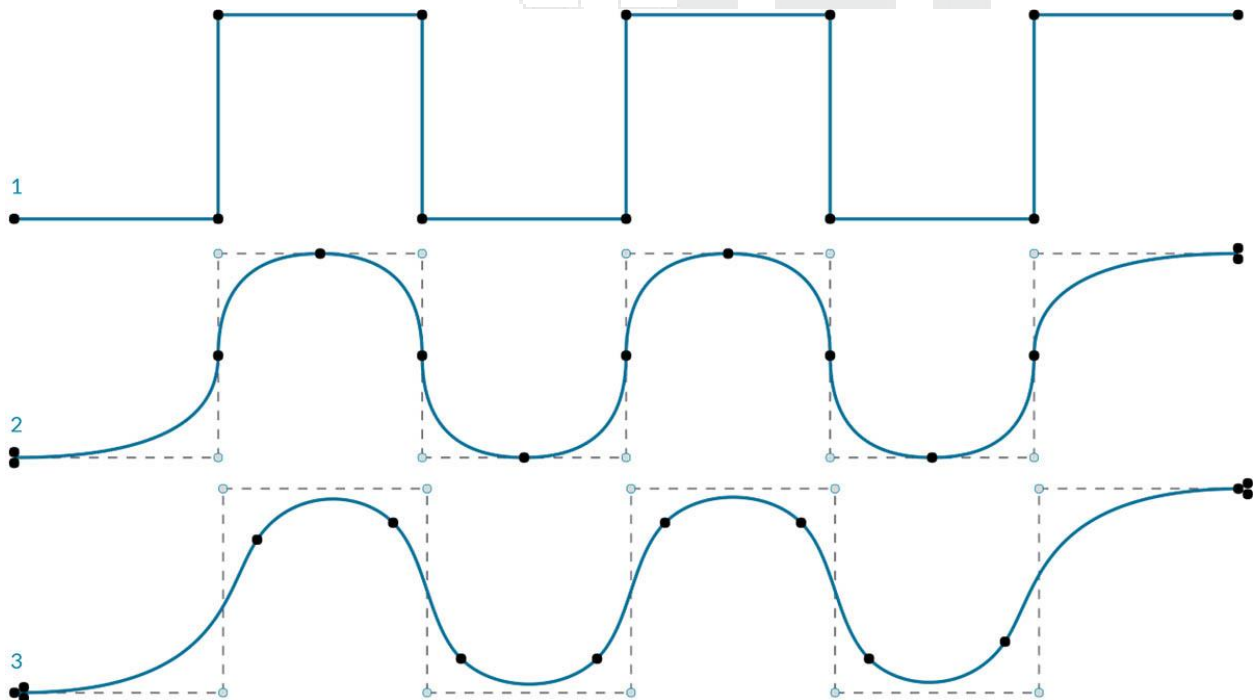
Grado: El Grado de la Curva determina el rango de influencia que los Puntos de Control tienen en una Curva; donde cuanto mayor sea el grado, mayor será el rango. El grado es un número

entero positivo. Este número suele ser 1, 2, 3 o 5, pero puede ser cualquier número entero positivo. Las líneas y poli líneas de NURBS suelen ser del Grado 1 y la mayoría de las Curvas de forma libre tienen el Grado 3 o 5.

Puntos de control: los **puntos de control** son una lista de al menos grados + 1 puntos. Una de las formas más fáciles de cambiar la forma de una curva NURBS es mover sus puntos de control.

Peso: los puntos de control tienen un número asociado llamado peso. Los pesos generalmente son números positivos. Cuando todos los puntos de control de una curva tienen el mismo peso (generalmente 1), la curva se denomina no racional; de lo contrario, la curva se llama racional. La mayoría de las curvas NURBS no son racionales.

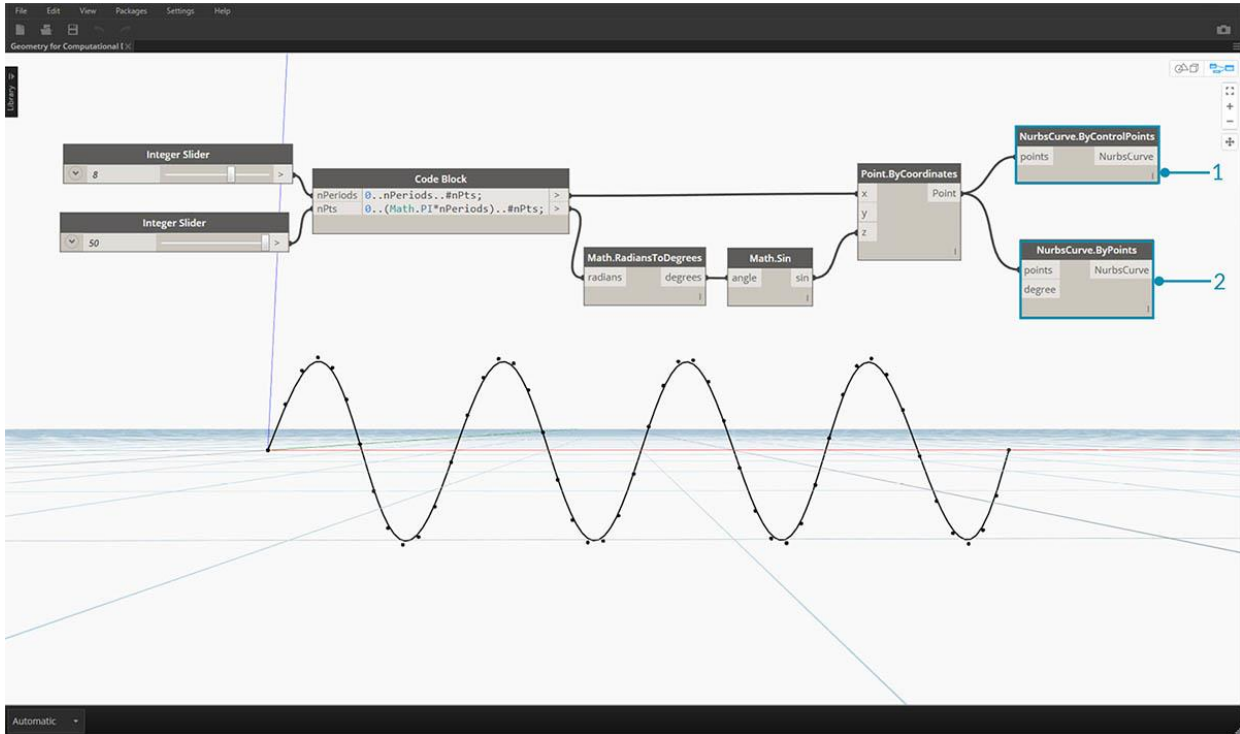
Nudos: los nudos son una lista de números (grados + N-1), donde N es el número de puntos de control. Los nudos se usan junto con los pesos para controlar la influencia de los puntos de control en la curva resultante. Un uso para Knots es crear dobleces en ciertos puntos de la curva.



1. Grado = 1
2. Grado = 2
3. Grado = 3

Tenga en cuenta que cuanto mayor es el valor del grado, más puntos de control se utilizan para interpolar la curva resultante.

Hagamos una curva sinusoidal en Dynamo usando dos métodos diferentes para crear curvas NURBS para comparar los resultados.



1. *NurbsCurve.ByControlPoints* utiliza la Lista de puntos como puntos de control
2. *NurbsCurve.ByPoints* dibuja una curva a través de la lista de puntos

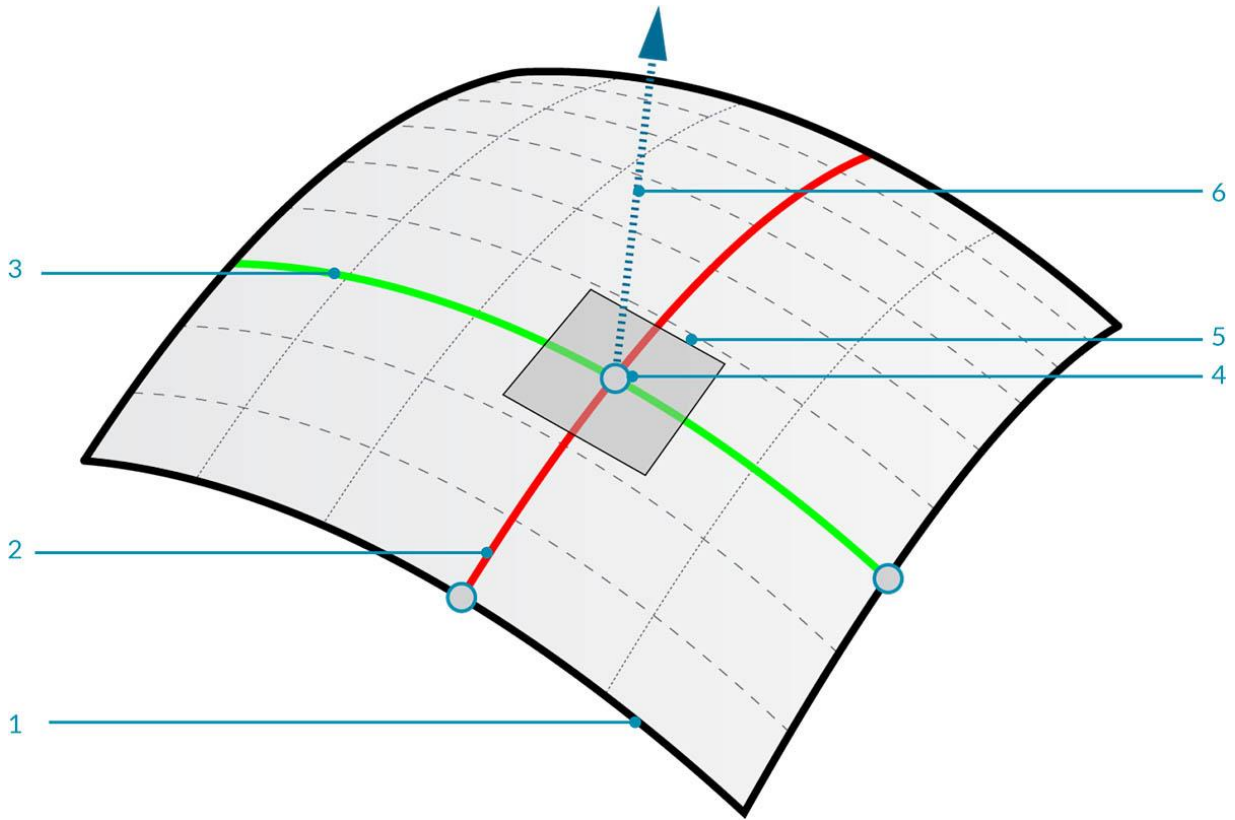
Descargue el archivo de ejemplo que acompaña a esta Geometry for Computational Design - Curves. Dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

Superficies

A medida que pasamos del uso de Curvas al uso de Superficies en un modelo, ahora podemos comenzar a representar los objetos que vemos en nuestro mundo tridimensional. Si bien las curvas no son siempre planas, es decir, son tridimensionales, el espacio que definen está siempre ligado a una dimensión. Las superficies nos dan otra dimensión y una colección de propiedades adicionales que podemos usar dentro de otras operaciones de modelado.

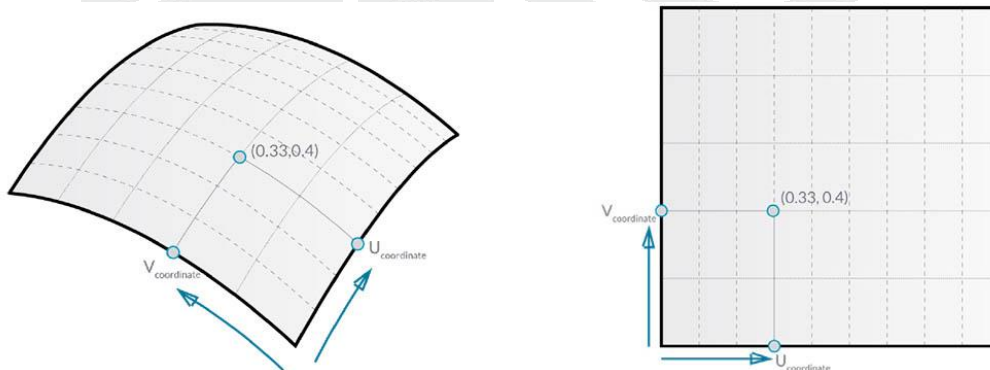
¿Qué es una superficie?

Una superficie es una forma matemática definida por una función y dos parámetros, en lugar de t para curvas, usamos U y V para describir el espacio de parámetros correspondiente. Esto significa que tenemos más datos geométricos para dibujar al trabajar con este tipo de geometría. Por ejemplo, las curvas tienen vectores tangentes y planos normales (que pueden rotar o girar a lo largo de la longitud de la curva), mientras que las superficies tienen vectores normales y planos tangentes que serán consistentes en su orientación.



1. Superficie
2. U Isocurve
3. V Isocurve
4. Coordenada UV
5. Plano perpendicular
6. Vector normal

Dominio de superficie: un dominio de superficie se define como el rango de parámetros (U, V) que se evalúan en un punto tridimensional en esa superficie. El dominio en cada dimensión (U o V) generalmente se describe como dos números (U Min to U Max) y (V Min to V Max).

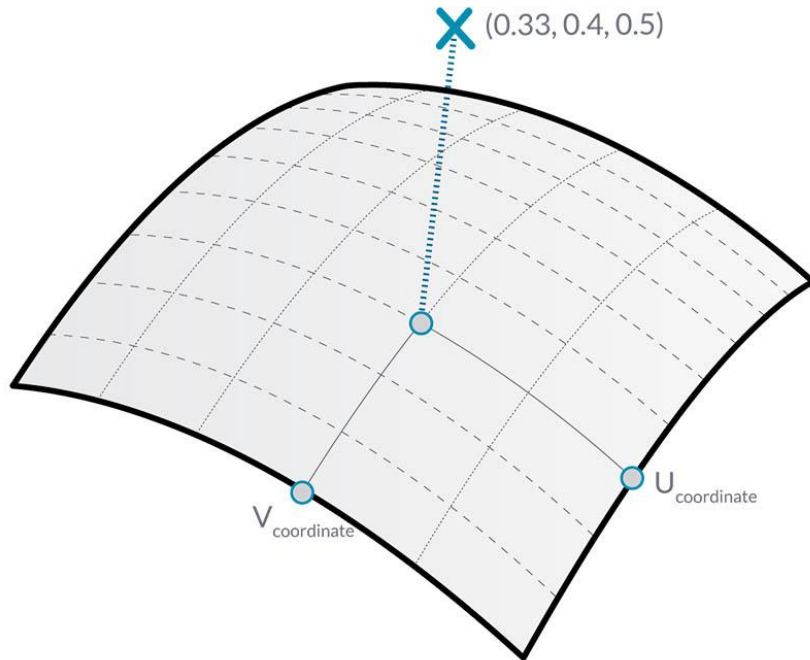


Aunque la forma de la Superficie no se ve "rectangular" y localmente puede tener un conjunto de isocurvas más ajustado o más flexible, el "espacio" definido por su dominio es siempre bidimensional. En Dynamo, siempre se entiende que las superficies tienen un dominio definido

por un mínimo de 0.0 y un máximo de 1.0 en ambas direcciones U y V. Las superficies planas o recortadas pueden tener diferentes dominios.

Isocurve (o Curva isoperimétrica): Curva definida por un valor U o V constante en la superficie y un dominio de valores para la otra dirección U o V correspondiente.

Coordenada UV: el punto en el espacio de parámetros UV definido por U, V y, a veces, W.

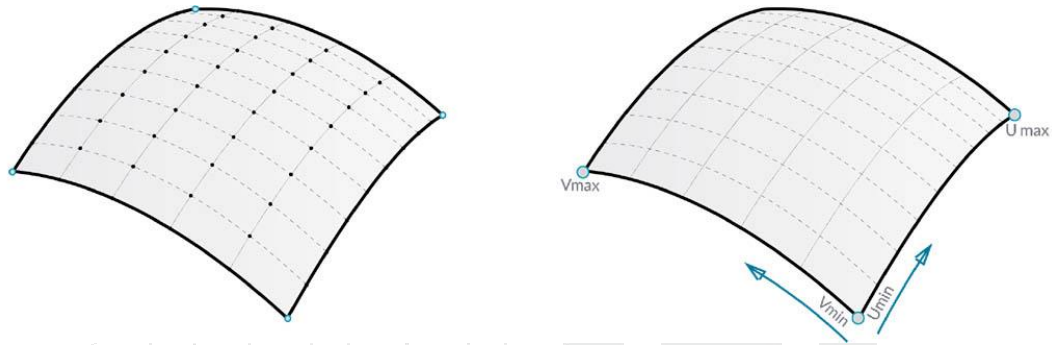


Plano perpendicular: un plano que es perpendicular a las isocurvas U y V en una determinada coordenada UV.

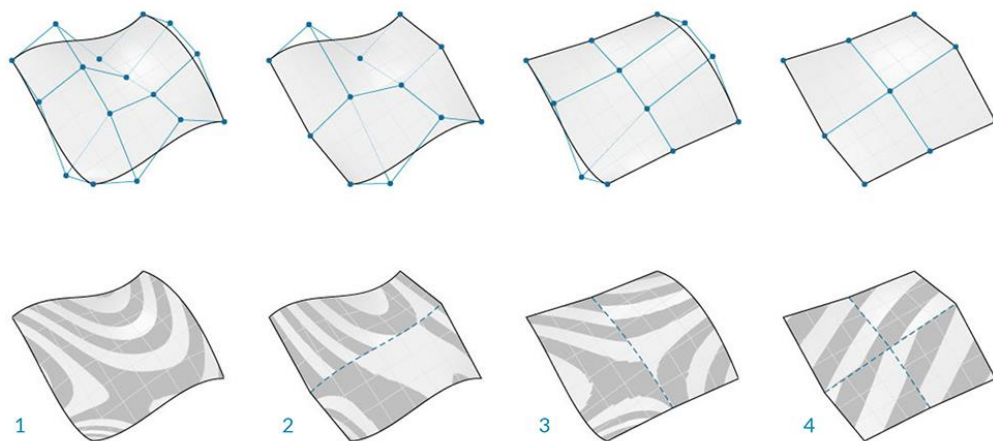
Vector Normal: Un Vector que define la dirección de "arriba" en relación con el Plano Perpendicular.

Superficies NURBS

Las superficies NURBS son muy similares a las curvas NURBS. Puedes pensar en NURBS Surfaces como una grilla de Curvas NURBS que van en dos direcciones. La forma de una superficie NURBS se define por un número de puntos de control y el grado de esa superficie en las direcciones U y V. Los mismos algoritmos se usan para calcular la forma, las normales, las tangentes, las curvaturas y otras propiedades por medio de puntos de control, pesos y grados.



En el caso de las superficies NURBS, hay dos direcciones implícitas en la geometría, porque las superficies NURBS son, independientemente de la forma que vemos, las cuadrículas rectangulares de los puntos de control. Y a pesar de que estas instrucciones son a menudo arbitrarias en relación con el sistema de coordenadas del mundo, las usaremos con frecuencia para analizar nuestros modelos o generar otra geometría basada en la superficie.

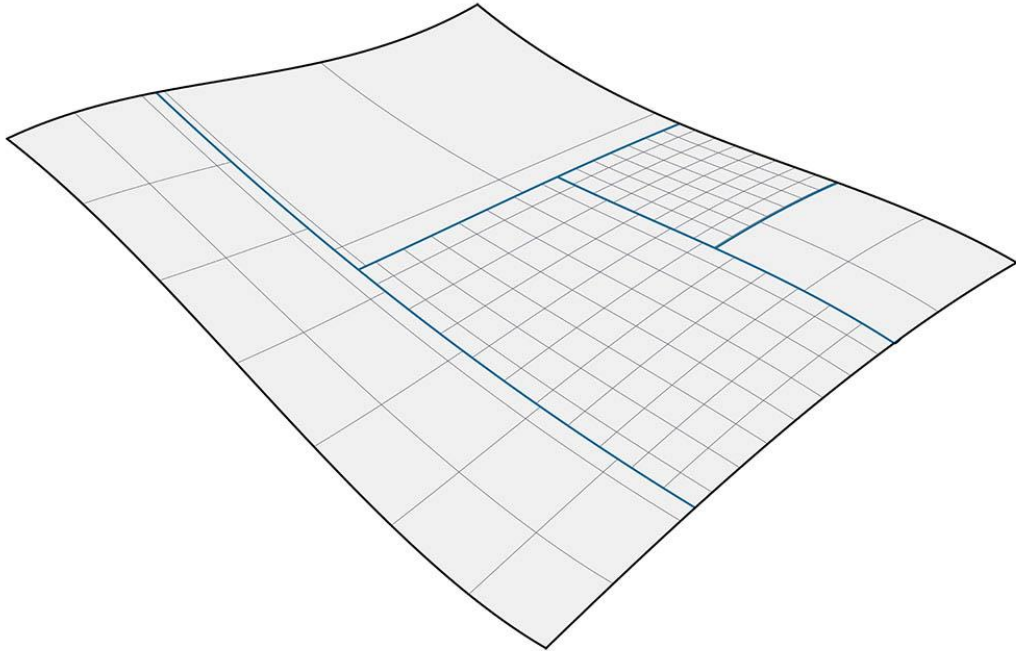


1. Grado (U, V) = (3,3)
2. Grado (U, V) = (3,1)
3. Grado (U, V) = (1,2)
4. Grado (U, V) = (1,1)

Polisuperficies

Las polisuperficies se componen de superficies que se unen a través de un borde. Las polisuperficies ofrecen una definición UV de más de dos dimensiones en el sentido de que ahora podemos movernos a través de las formas conectadas a través de su Topología.

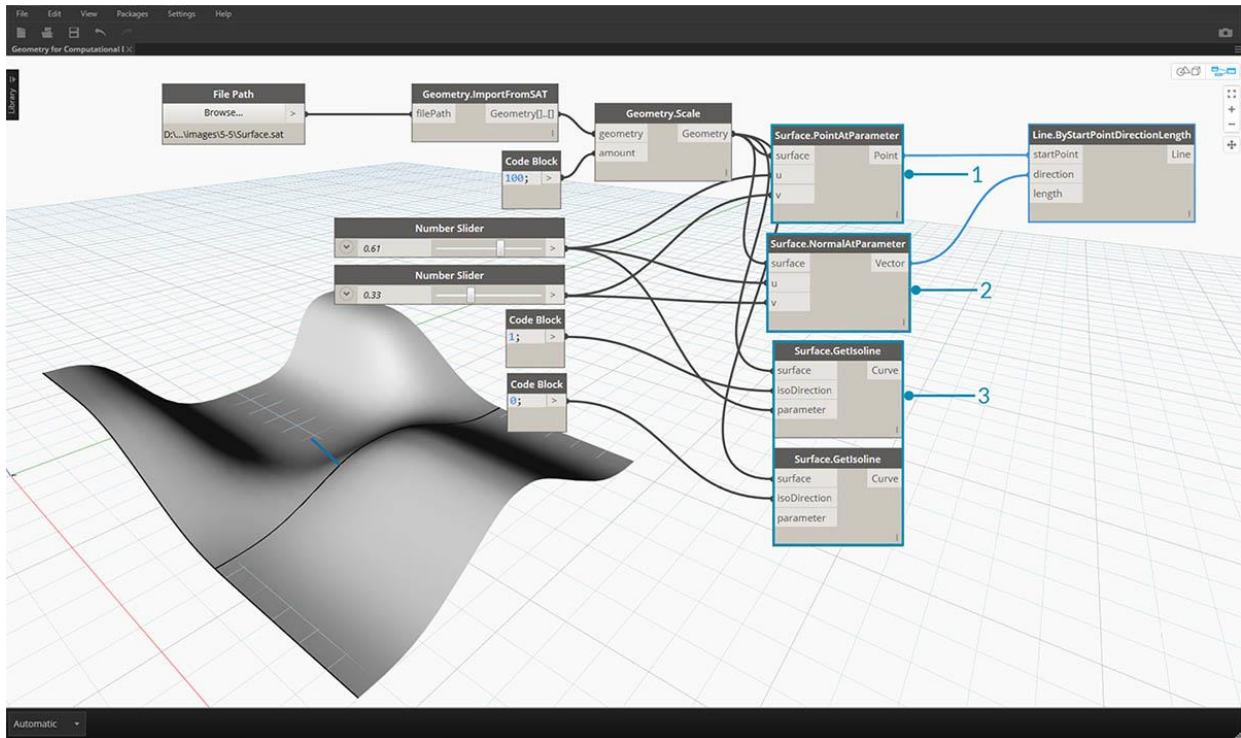
Mientras que "Topología" generalmente describe un concepto sobre cómo se conectan las partes y / o relacionadas. La topología en Dynamo también es un tipo de Geometría. Específicamente, es una categoría principal para Superficies, Polisuperficies y Sólidos.



A veces llamados parches, unir Superficies de esta manera nos permite hacer formas más complejas y definir detalles a lo largo de la costura. Convenientemente, podemos aplicar una operación de fileteado o chaflán a los bordes de una Polysurface.

Vamos a importar y evaluar una superficie en un parámetro en Dynamo para ver qué tipo de información podemos extraer.

DARCO
DESDE 1988



1. *Surface.PointAtParameter* devuelve el punto en una determinada coordenada UV
2. *Surface.NormalAtParameter* devuelve el Vector Normal a una determinada Coordenada UV
3. *Surface.GetIsoline* devuelve la curva isoparamétrica a una coordenada U o V - tenga en cuenta la entrada de isoDirection.

Descargue los archivos de ejemplo que acompañan a esta.

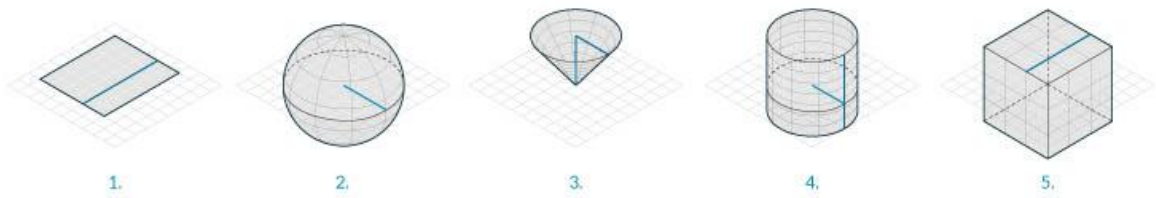
1. Geometría para Diseño Computacional - Surfaces.dyn
2. Surface.sat

Sólidos

Si queremos construir modelos más complejos que no se pueden crear a partir de una sola superficie o si queremos definir un volumen explícito, ahora debemos aventurarnos en el campo de los sólidos (y polisuperficies). Incluso un cubo simple es lo suficientemente complejo como para necesitar seis superficies, una por cara. Los sólidos dan acceso a dos conceptos clave que las superficies no tienen: una descripción topológica más refinada (caras, aristas, vértices) y operaciones booleanas.

¿Qué es un sólido?

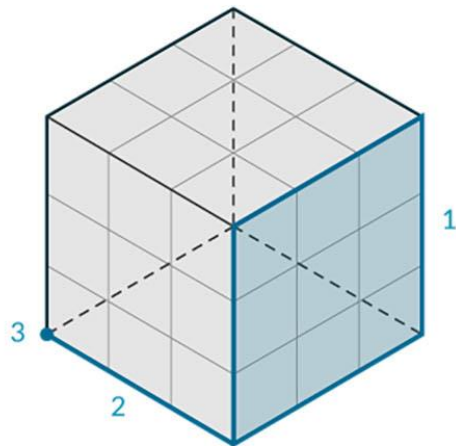
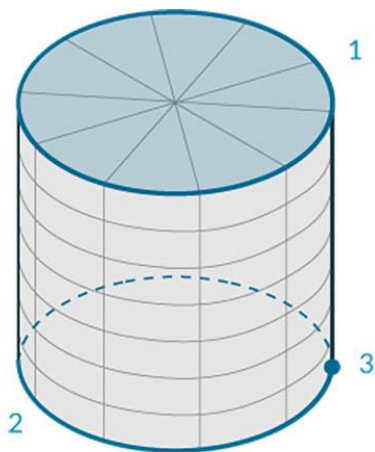
Los sólidos consisten en una o más superficies que contienen volumen por medio de un límite cerrado que define "dentro" o "fuera". Independientemente de cuántas de estas superficies haya, deben formar un volumen "hermético" para que se considere sólido. Se pueden crear sólidos uniendo Superficies o Polisuperficies juntos o usando operaciones como desvío, barrido y revolución. Las primitivas Esfera, Cubo, Cono y Cilindro también son sólidos. Un Cubo con al menos una cara eliminada cuenta como una Polisuperficie, que tiene algunas propiedades similares, pero no es un Sólido.



1. Un plano está hecho de una sola superficie y no es sólido.
2. Una esfera está hecha de una superficie, pero es sólida.
3. Un cono está hecho de dos superficies unidas para formar un sólido.
4. Un cilindro está hecho de tres superficies unidas para formar un sólido.
5. Un cubo está hecho de seis superficies unidas para formar un sólido.

Topología

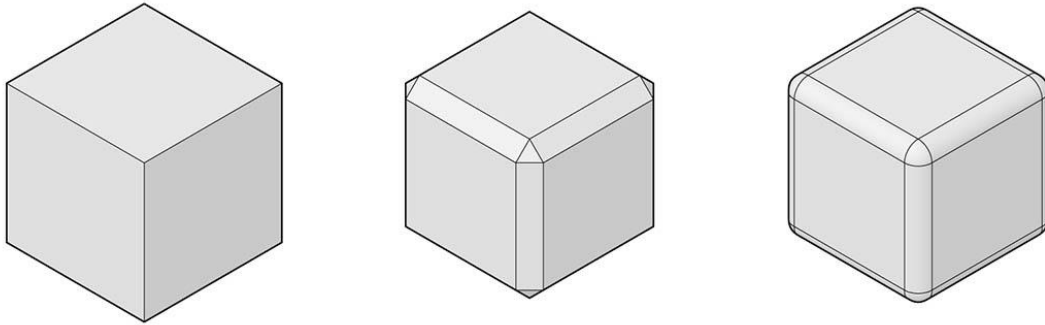
Los sólidos se componen de tres tipos de elementos: vértices, bordes y caras. Las caras son las superficies que componen el Sólido. Los bordes son las curvas que definen la conexión entre las caras adyacentes, y los vértices son los puntos de inicio y final de esas curvas. Estos elementos se pueden consultar utilizando los nodos de Topología.



1. Caras
2. Bordes
3. Vértices

Operaciones

Los sólidos se pueden modificar fileteando o biselando sus bordes para eliminar esquinas y ángulos agudos. La operación de chaflán crea una superficie reglada entre dos caras, mientras que un filete se mezcla entre las caras para mantener la tangencia.



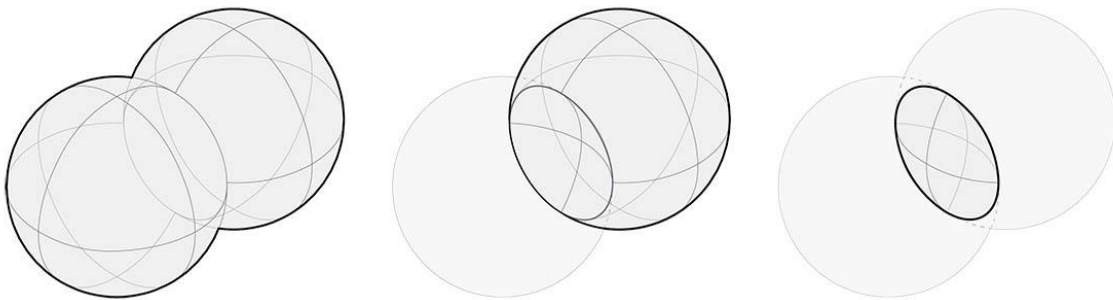
1. Cubo sólido
2. Cubo biselado
3. Cubo fileteado

Operaciones Booleanas

Las operaciones booleanas sólidas son métodos para combinar dos o más sólidos. Una sola operación booleana significa en realidad realizar cuatro operaciones:

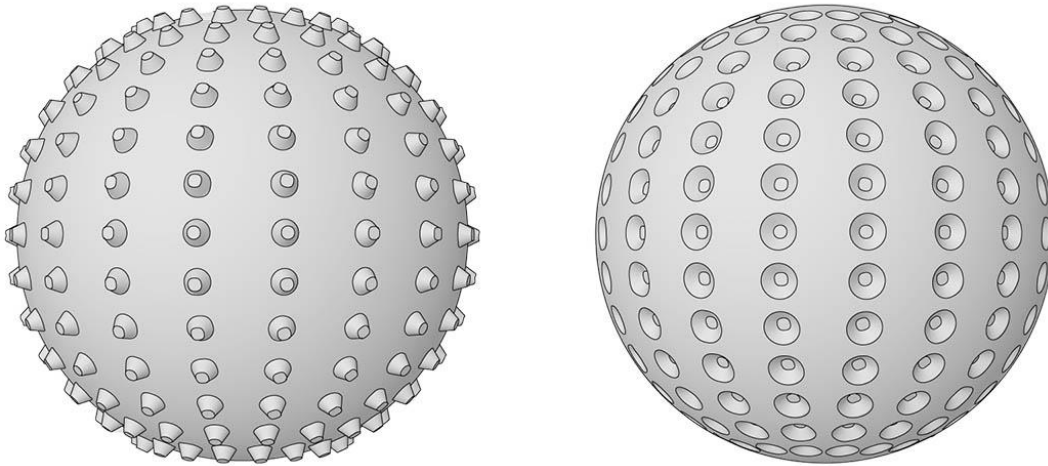
1. **Interseca** dos o más objetos.
2. **Dividirlos** en las intersecciones.
3. **Eliminar** partes no deseadas de la geometría.
4. **Únete a** todo de nuevo.

Esto hace que Solid Booleans sea un poderoso proceso de ahorro de tiempo. Hay tres operaciones booleanas sólidas que distinguen qué partes de la geometría se mantienen.



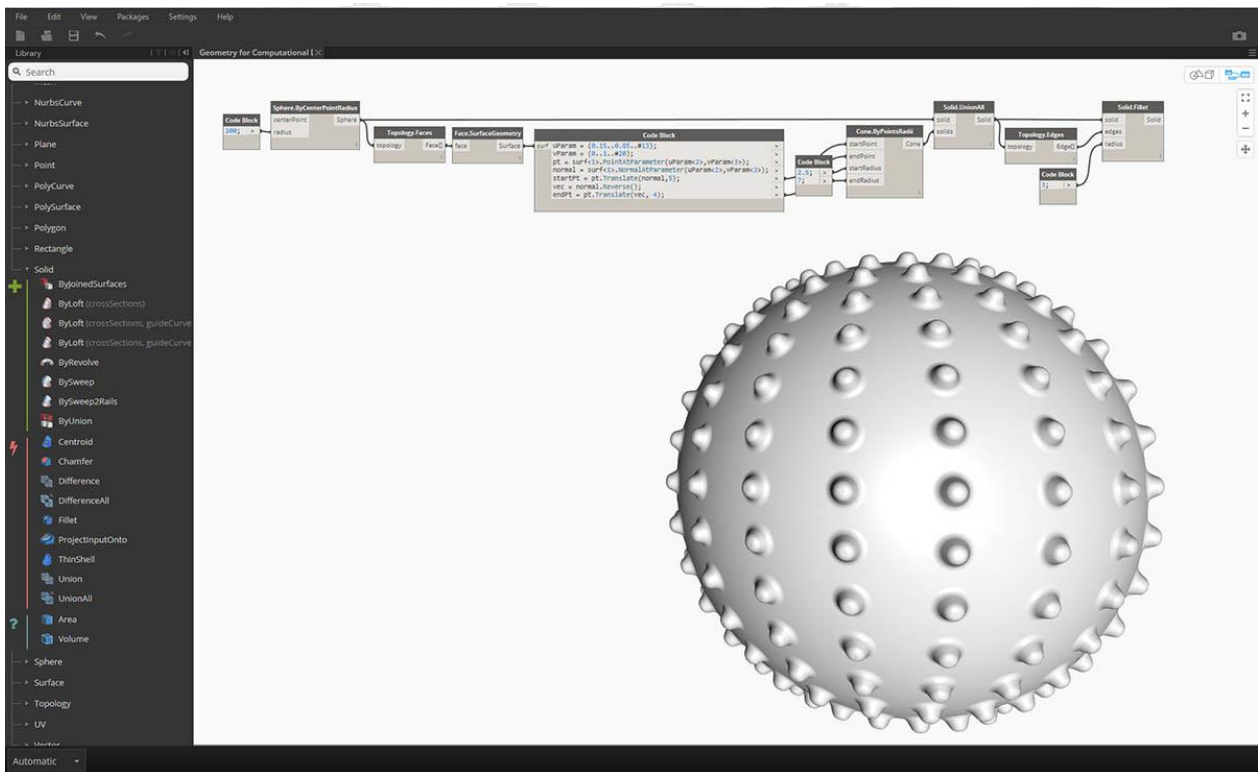
1. **Unión:** elimine las porciones superpuestas de los sólidos y únalos en un solo sólido.
2. **Diferencia:** resta un sólido de otro. El sólido que restar se denomina herramienta. Tenga en cuenta que puede cambiar qué Sólido es la herramienta para mantener el volumen inverso.
3. **Intersección:** mantenga solo el volumen de intersección de los dos sólidos.

Además de estas tres operaciones, Dynamo tiene **Solid.DifferenceAll** y **Solid.Union All** nodes para realizar operaciones de diferencia y unión con múltiples sólidos.



1. **UnionAll**: Operación de unión con esfera y conos hacia afuera
2. **DifferenceAll**: operación de diferencia con esfera y conos orientados hacia adentro

Usemos algunas operaciones booleanas para crear una bola puntiaguda.



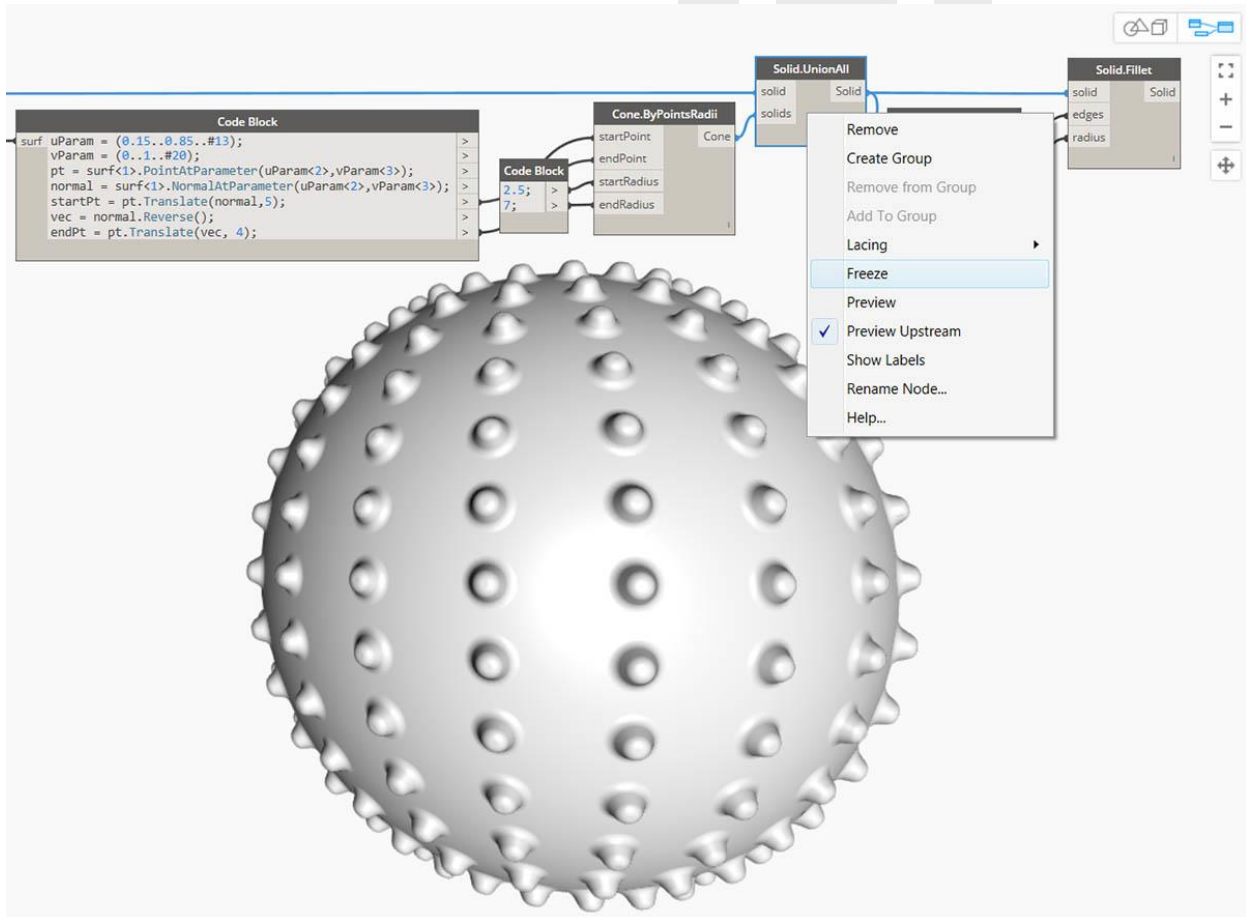
1. **Sphere.ByCenterPointRadius** : crea la base sólida.
2. **Topología.Faces** , **Face.SurfaceGeometry** : consulta las caras del sólido y **conviértelas en geometría de superficie**; en este caso, la esfera solo tiene una cara.

3. **Cone.ByPointsRadii** : construye conos usando puntos en la superficie.
4. **Solid.UnionAll** : Union the Cones and the Sphere.
5. **Topología.Edificios**: consulta los bordes del nuevo sólido
6. **Solid.Fillet**: Fillet los bordes de la bola puntiaguda

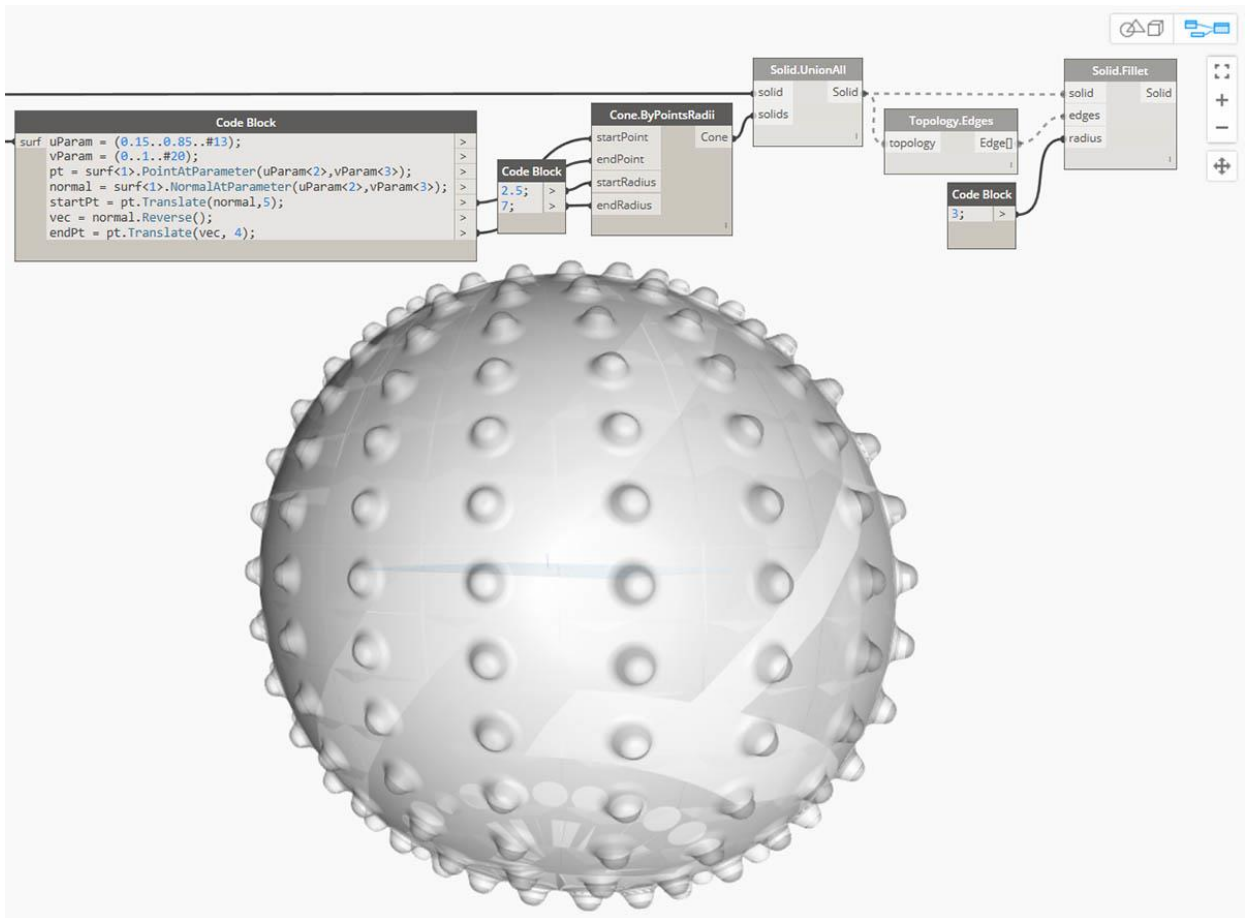
Descargue los archivos de ejemplo que acompañan a esta imagen **Geometry for Computational Design - Solids.dyn**

Congelación

Las operaciones booleanas son complejas y pueden ser lentas de calcular. Use la funcionalidad Freeze para suspender la ejecución de los nodos seleccionados y los nodos afectados aguas abajo.

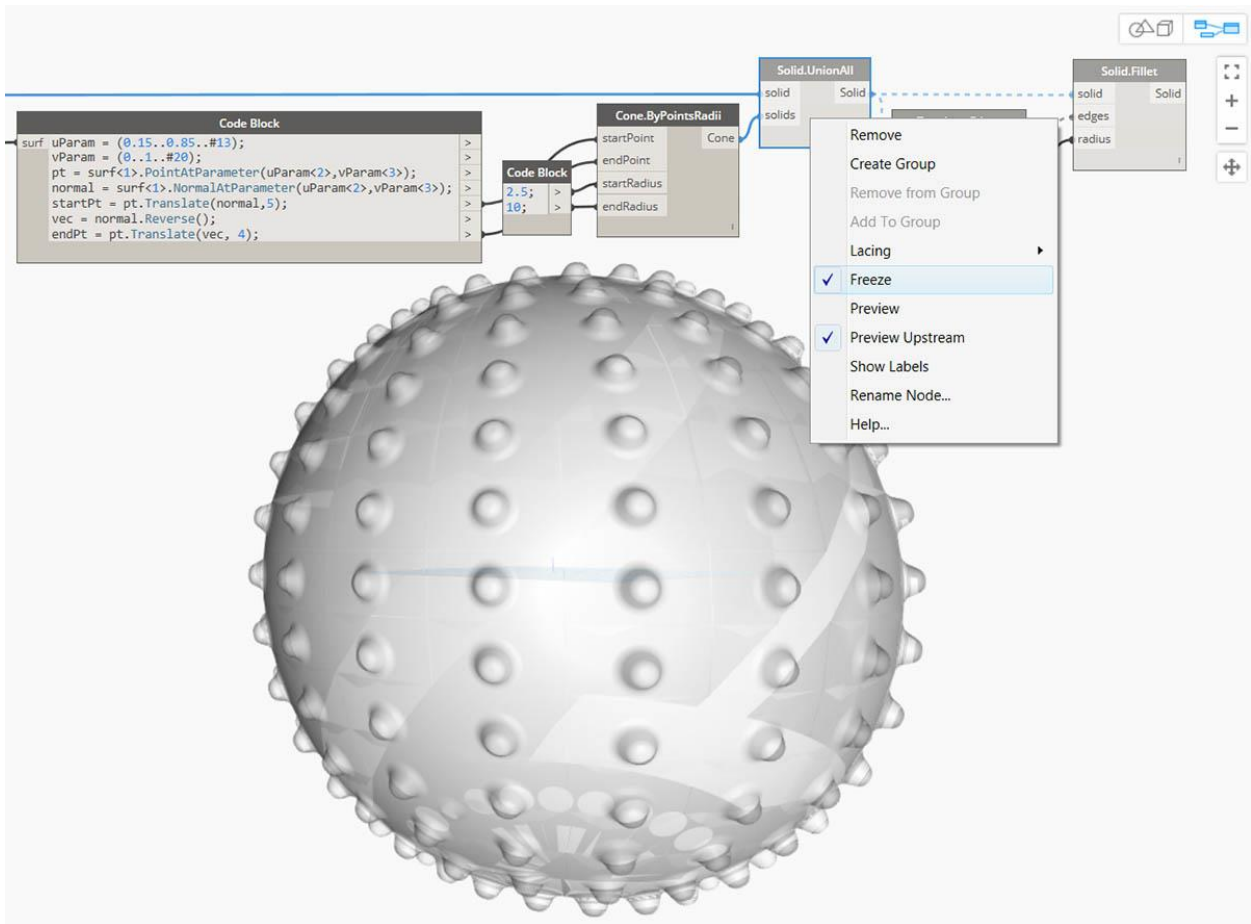


Use el menú contextual del botón derecho para congelar la operación Solid Union



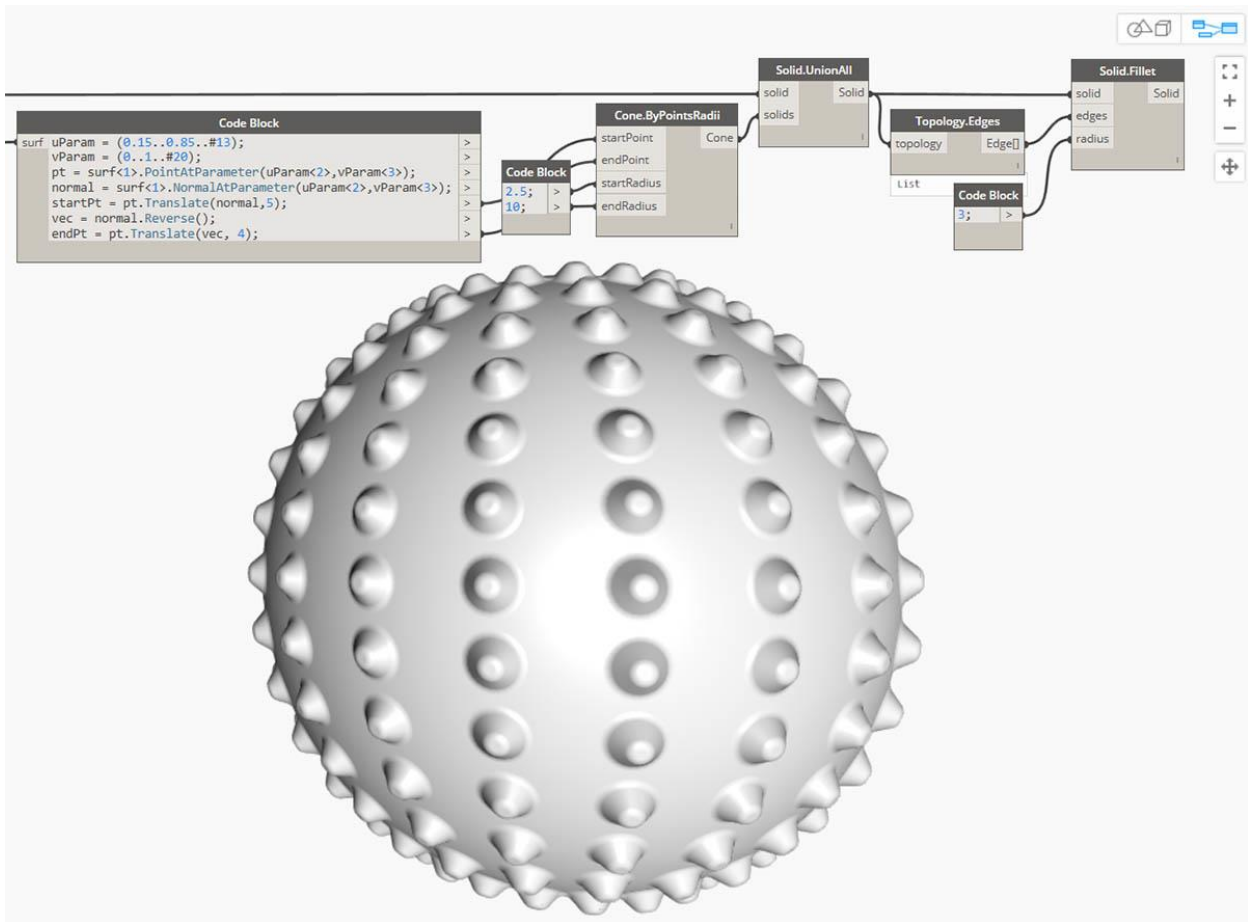
El nodo seleccionado y todos los nodos descendentes obtendrán una vista previa en un modo fantasma gris claro, y los cables afectados se mostrarán como líneas discontinuas. La vista previa de la geometría afectada también aparecerá fantasma. Ahora puede cambiar los valores en sentido ascendente sin calcular la unión booleana.

DARCO
DESDE 1988



Para descongelar los nodos, haga clic derecho y desmarque Congelar.

DARCO
DESDE 1988



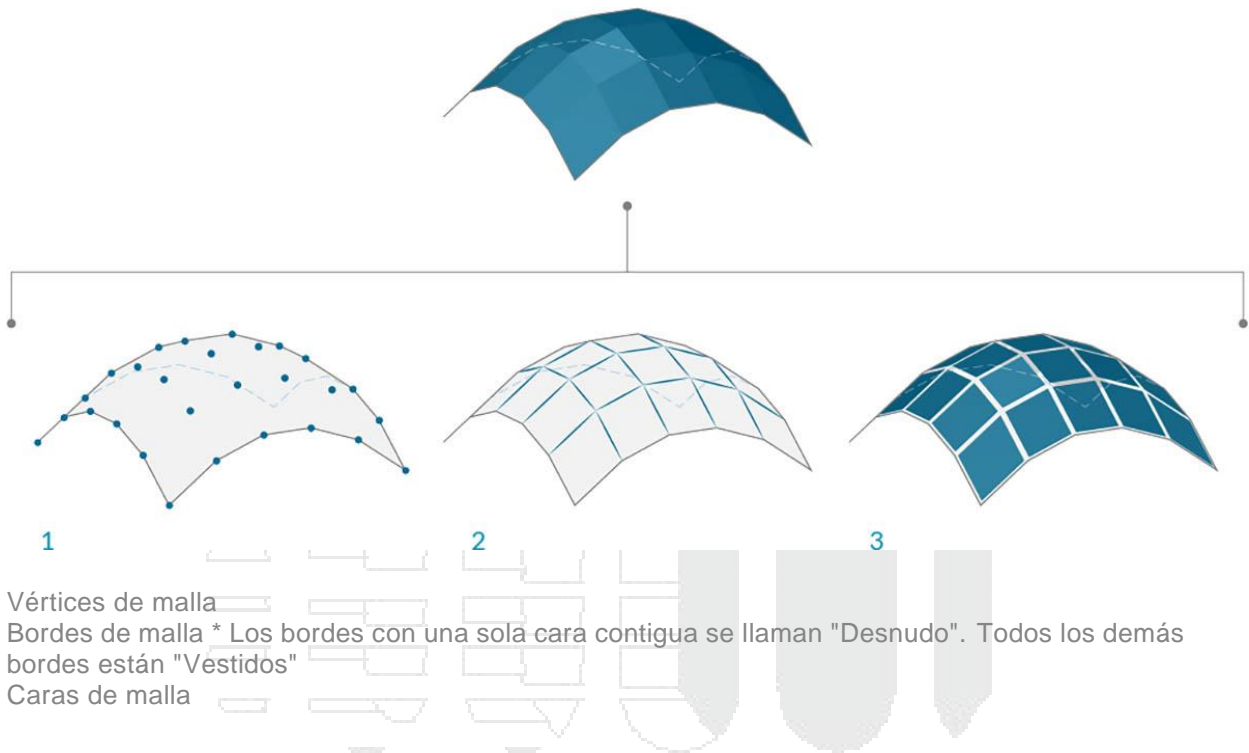
Todos los nodos afectados y las vistas previas de geometría asociadas se actualizarán y volverán al modo de vista previa estándar.

Mallas

En el campo del modelado computacional, las mallas son una de las formas más generalizadas de representar la geometría 3D. La geometría de malla puede ser una alternativa ligera y flexible para trabajar con NURBS, y las Mallas se usan en todo, desde renderizado y visualización hasta fabricación digital e impresión 3D.

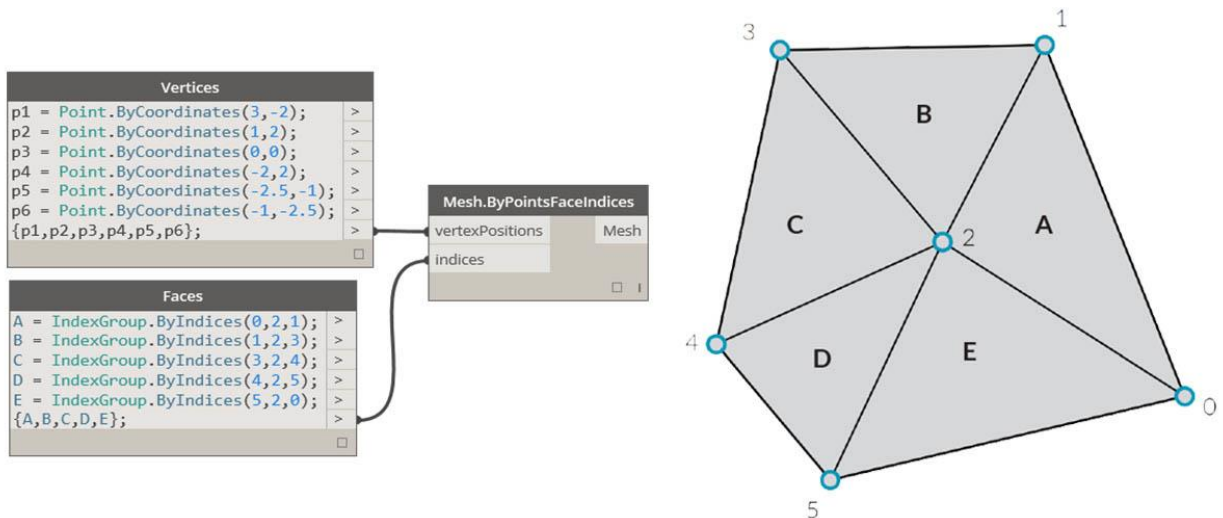
¿Qué es una malla?

A Mesh es una colección de cuadriláteros y triángulos que representa una geometría de superficie o sólida. Al igual que los sólidos, la estructura de un objeto de malla incluye vértices, bordes y caras. Hay propiedades adicionales que también hacen que las Mallas sean únicas, como las normales.



Elementos de malla

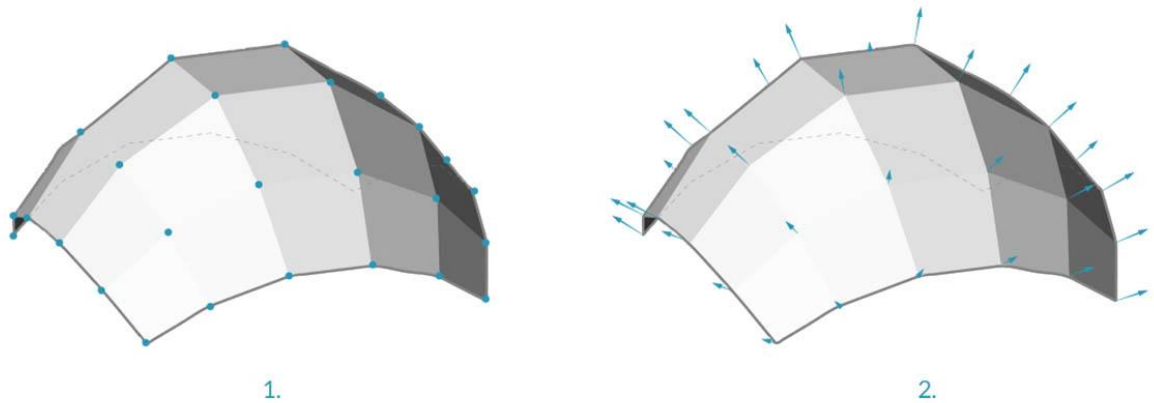
Dynamo define Mallas usando una estructura de datos Face-Vertex. En su nivel más básico, esta estructura es simplemente una colección de puntos que se agrupan en polígonos. Los puntos de una Malla se llaman vértices, mientras que los polígonos de superficie se llaman caras. Para crear una malla, necesitamos una lista de vértices y un sistema para agrupar esos vértices en caras llamadas un grupo de índice.



1. Lista de vértices
2. Lista de grupos de índice para definir caras

Vértices + vértices normales

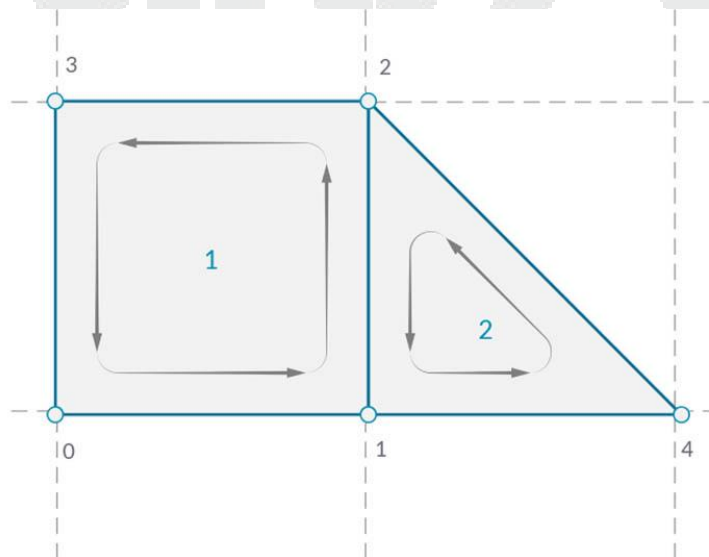
Los vértices de una Malla son simplemente una lista de puntos. El índice de los vértices es muy importante al construir una Malla, u obtener información sobre la estructura de una Malla. Para cada vértice, también hay un vértice correspondiente normal (vector) que describe la dirección promedio de las caras adjuntas y nos ayuda a comprender la orientación de "entrada" y "salida" de la malla.



1. Vértices
2. Vertex Normals

Caras

Una cara es una lista ordenada de tres o cuatro vértices. Por lo tanto, la representación de "superficie" de una cara de malla está implícita de acuerdo con la posición de los vértices que se indexan. Ya tenemos la lista de vértices que componen Mesh, así que, en lugar de proporcionar puntos individuales para definir una cara, simplemente usamos el índice de los vértices. Esto también nos permite usar el mismo vértice en más de una cara.



1. Una cara cuádruple hecha con los índices 0, 1, 2 y 3

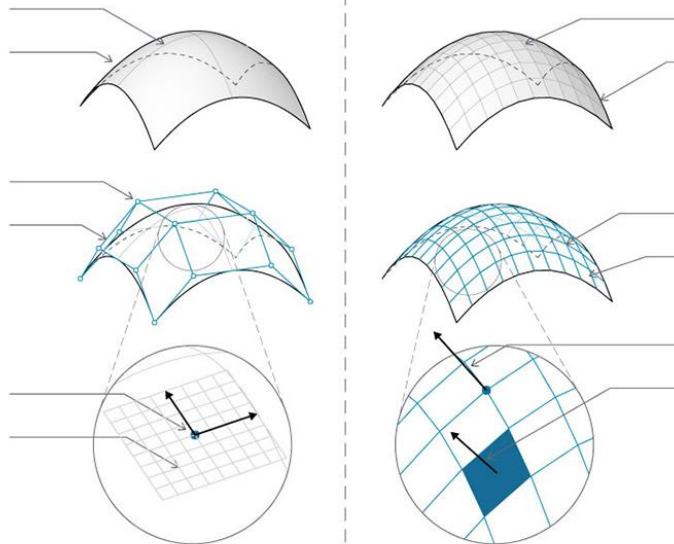
- Una cara de triángulo hecha con los índices 1, 4 y 2 Tenga en cuenta que los grupos de índice se pueden desplazar en su orden: siempre que la secuencia se ordene en sentido contrario a las agujas del reloj, la cara se definirá correctamente

Mallas frente a superficies NURBS

¿En qué se diferencia la geometría Mesh de la geometría NURBS? ¿Cuándo podría querer usar uno en lugar del otro?

Parametrización

En un capítulo anterior, vimos que las superficies NURBS están definidas por una serie de curvas NURBS que van en dos direcciones. Estas instrucciones están etiquetadas U y V permiten que la superficie de un NURB se parametrize según un dominio de superficie bidimensional. Las curvas mismas se almacenan como ecuaciones en la computadora, lo que permite que las superficies resultantes se calculen con un grado arbitrariamente pequeño de precisión. Sin embargo, puede ser difícil combinar varias superficies NURBS juntas. Unir dos superficies NURBS dará como resultado una polisuperficie, donde diferentes secciones de la geometría tendrán diferentes parámetros de UV y definiciones de curva.

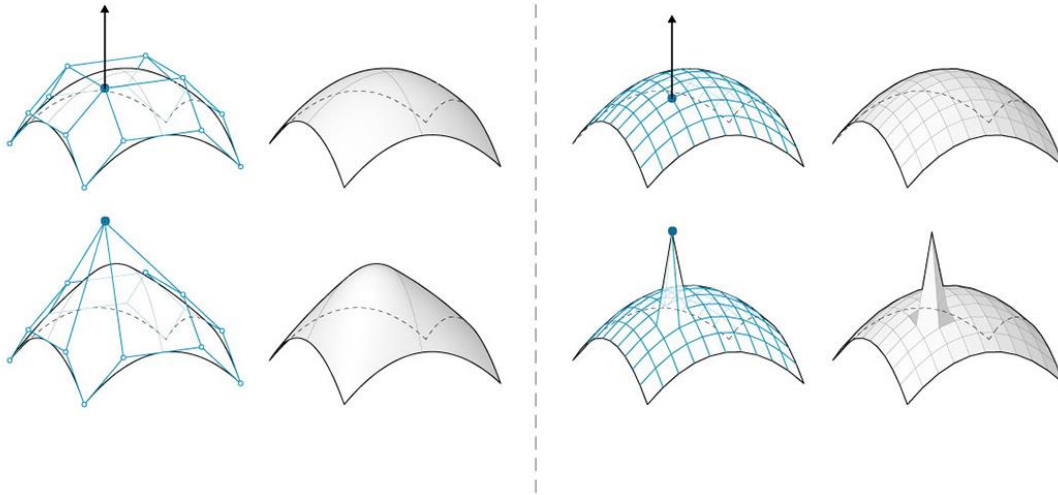


1. Superficie
2. Curva isoparamétrica (isoparm)
3. Punto de control de superficie
4. Polígono de control de superficie
5. Punto isoparametrico
6. Marco de superficie
7. Malla
8. Naked Edge
9. Red de malla
10. Bordes de malla
11. Vértice Normal
12. Mesh Face / Mesh Face Normal

Las mallas, por otro lado, se componen de un número discreto de vértices y caras exactamente definidos. La red de vértices generalmente no se puede definir por UV coordenadas simples, y debido a que las caras son discretas, la cantidad de precisión está integrada en la Malla y solo se puede cambiar refinando la Malla y agregando más caras. La falta de descripciones matemáticas permite que las mallas manejen de manera más flexible la geometría compleja dentro de una sola malla.

Influencia local versus global

Otra diferencia importante es la medida en que un cambio local en la geometría de malla o NURBS afecta a la forma completa. Mover un vértice de una Malla solo afecta las caras que están adyacentes a ese vértice. En las superficies NURBS, la extensión de la influencia es más complicada y depende del grado de la superficie, así como de los pesos y nudos de los puntos de control. En general, sin embargo, mover un único punto de control en una superficie NURBS crea un cambio de geometría más uniforme y extenso.



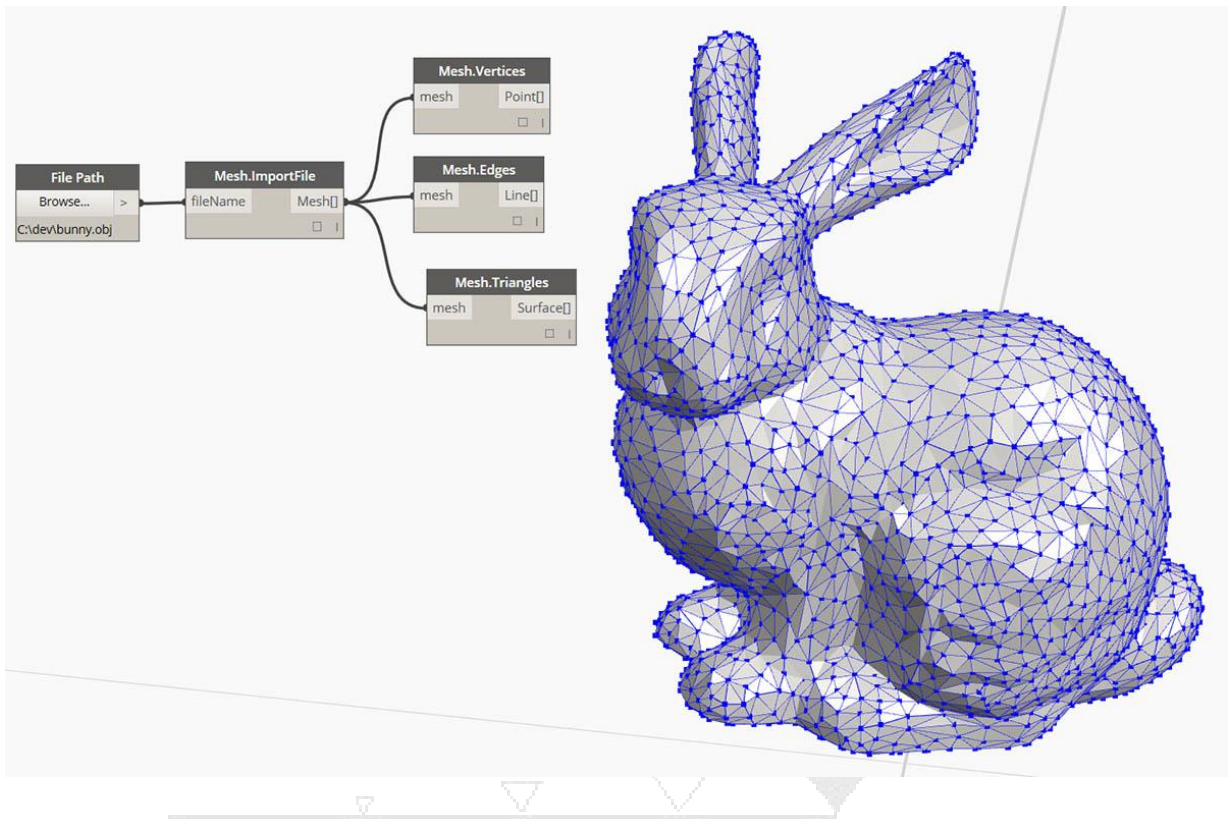
1. Superficie NURBS: mover un punto de control tiene una influencia que se extiende a través de la forma
2. Geometría de malla: mover un vértice solo tiene influencia sobre los elementos adyacentes

Una analogía que puede ser útil es comparar una imagen vectorial (compuesta de líneas y curvas) con una imagen raster (compuesta de píxeles individuales). Si hace un acercamiento a una imagen vectorial, las curvas permanecen nítidas y claras, mientras que al hacer zoom en una imagen ráster resulta que los píxeles individuales se vuelven más grandes. En esta analogía, las superficies NURBS se pueden comparar con una imagen vectorial porque existe una relación matemática uniforme, mientras que una malla se comporta de manera similar a una imagen ráster con una resolución establecida.

Juego de herramientas Mesh

Las capacidades de acoplamiento de Dynamo pueden extenderse instalando el paquete Mesh Toolkit . Dynamo Mesh Toolkit proporciona herramientas para importar mallas desde formatos de archivos externos, crear una malla de objetos de geometría de Dynamo y crear manualmente

mallas por sus vértices e índices. La biblioteca también proporciona herramientas para modificar mallas, reparar mallas o extraer cortes horizontales para su uso en la fabricación.



Importación de geometría

Hay varias formas de importar geometría en Dynamo. Hemos demostrado la importación de mallas utilizando *Mesh Toolkit* en la sección anterior; también podemos importar modelos sólidos de archivos .SAT. Con estos procesos, podemos desarrollar la geometría en otra plataforma, cargarla en Dynamo y aplicar operaciones paramétricas a través de la programación visual.

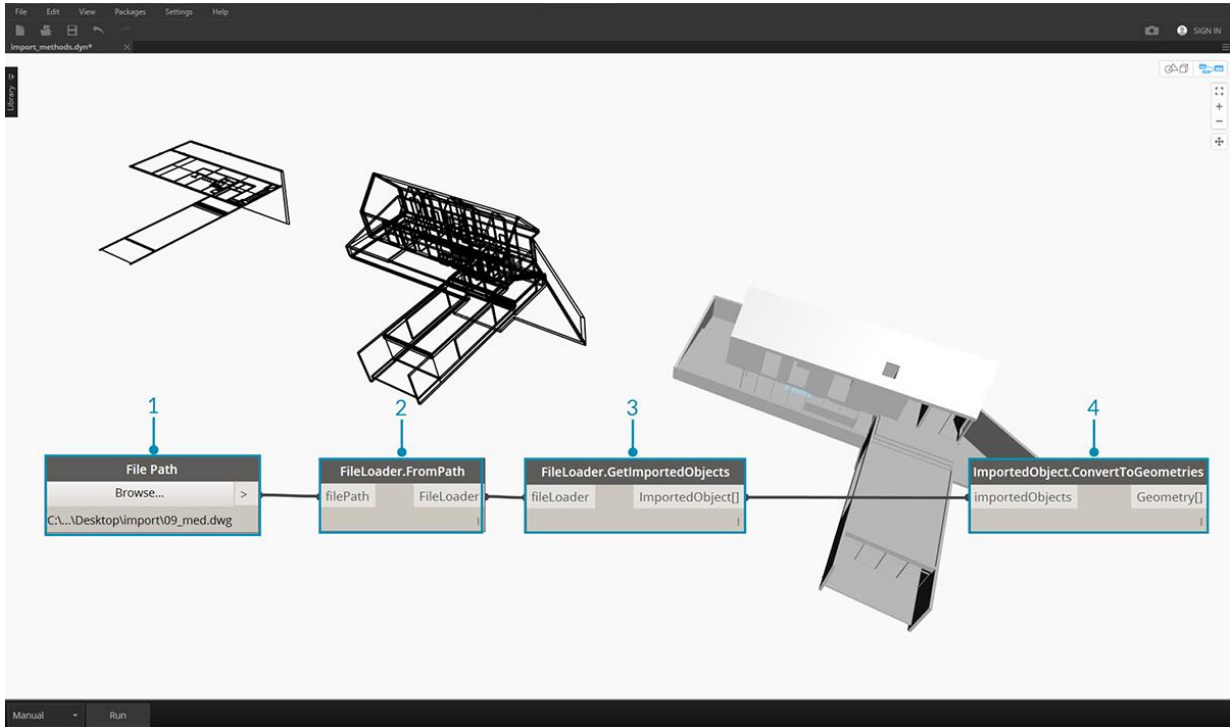
Otro método para importar geometría utiliza un proceso llamado *Traducción ATF*. En este caso, podemos importar no solo la geometría, sino la estructura de un archivo. Por ejemplo, podemos elegir qué capas de un .DWG importar en lugar de importar todo el modelo. Lo demostraremos a continuación con más detalle.

Importación de geometría desde un archivo DWG

Los nodos para importar un DWG en el entorno de Dynamo se encuentran en la pestaña *Traducción* (Nota: estas herramientas solo están disponibles en *Dynamo Studio*). Los siguientes ejemplos muestran una serie de componentes utilizados para buscar un archivo, importar el contenido del archivo y convertirlo en geometría de Dynamo utilizable. Dynamo también nos da la capacidad de filtrar y seleccionar objetos específicos para importar desde un archivo DWG, que demostraremos a continuación. Para obtener más información sobre Importar geometría desde un archivo DWG, lea la publicación de blog de Ben Goh aquí.

Obtener objetos importados

La forma más sencilla de importar DWG a Dynamo Studio es importar todo el archivo al espacio de trabajo:



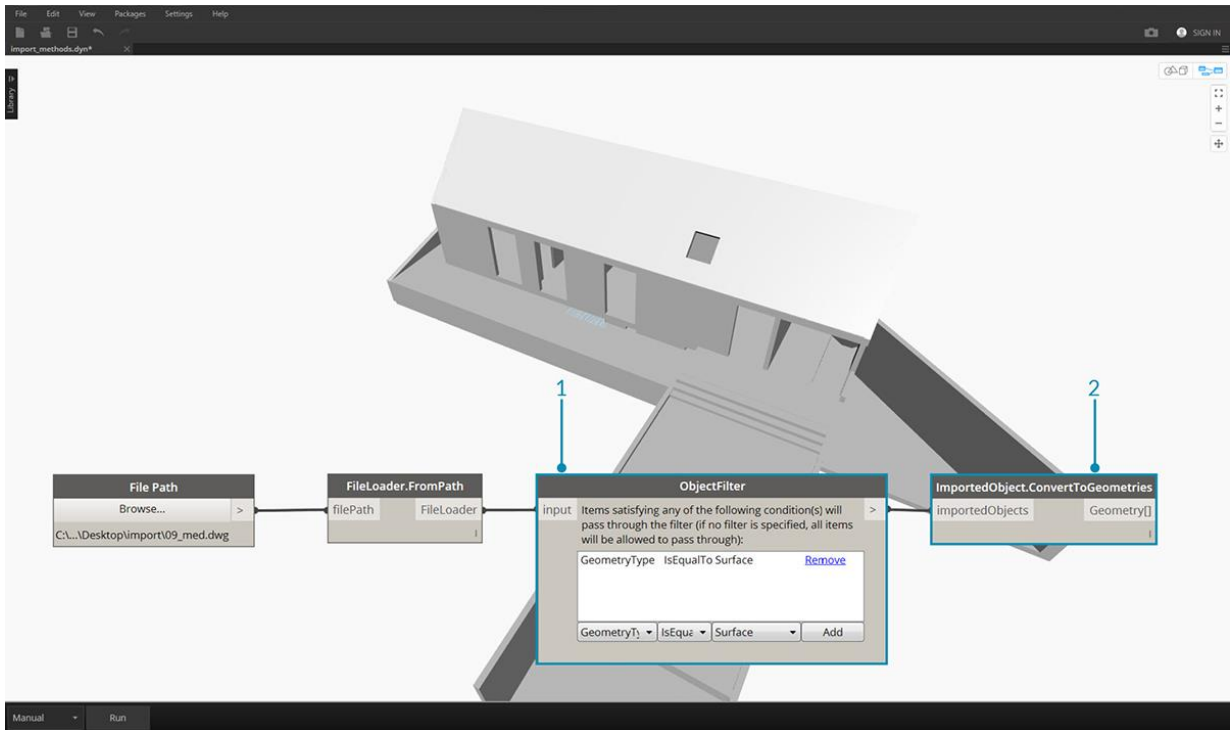
1. Utilice el componente Ruta de archivo para buscar el archivo DWG que se importará en Dynamo.
2. Conéctese a **FileLoader.FromPath** para leer el archivo.
3. Utilice el componente **FileLoader.GetImportedObjects** para analizar la geometría en Dynamo Studio.
4. **ImportedObject.ConvertToGeometries** convertirá los objetos en geometría utilizable en el espacio de trabajo de Dyanamo.

Como se muestra en la imagen de arriba, todos los tipos de geometría en el archivo DWG (superficies, mallas, curvas y líneas) se importan a Dynamo.

Filtro de objeto

Para especificar qué geometrías se importan del archivo DWG, se pueden agregar nodos adicionales de **ObjectFilter** a la definición. El nodo **ObjectFilter** es compatible con un **FileLoader** o una lista de **ImportedObject** y genera una lista de **ImportedObject**.

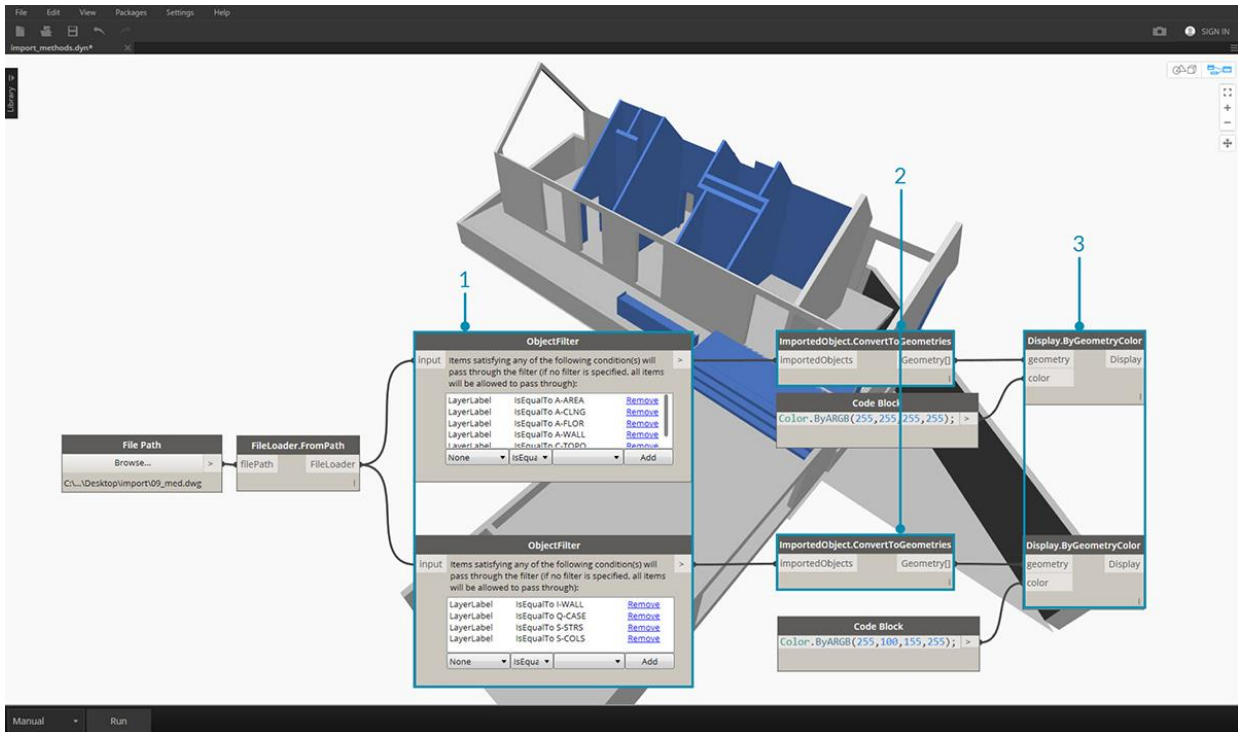
Las siguientes imágenes muestran las instrucciones condicionales dentro de cada nodo **ObjectFilter**. Cualquier objeto **importado** que satisfaga cualquiera de las condiciones enumeradas pasará por el filtro. El filtrado puede basarse en la etiqueta de la capa (es decir, el nombre de la capa), el tipo de geometría, el color difuso, etc., y se puede usar junto con otros filtros para refinar la selección.



1. Reemplace **FileLoader.GetImportedObjects** con **ObjectFilter** para buscar condiciones específicas en el archivo DWG. - en este caso, solo se importará la geometría de la superficie, eliminando toda curva y geometría de línea visibles en la imagen anterior.
2. Conecte el filtro a **ImportedObject.ConvertToGeometries** para importar la geometría filtrada.

Al agregar dos filtros con diferentes declaraciones condicionales, podemos dividir la lista de geometría en múltiples secuencias:

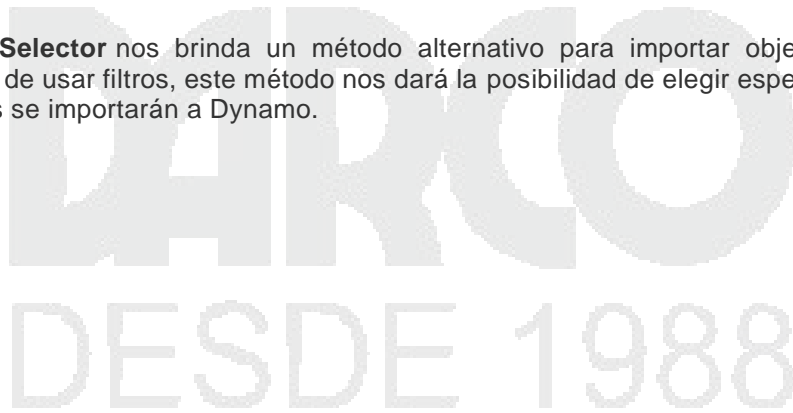
DARCO
DESDE 1988

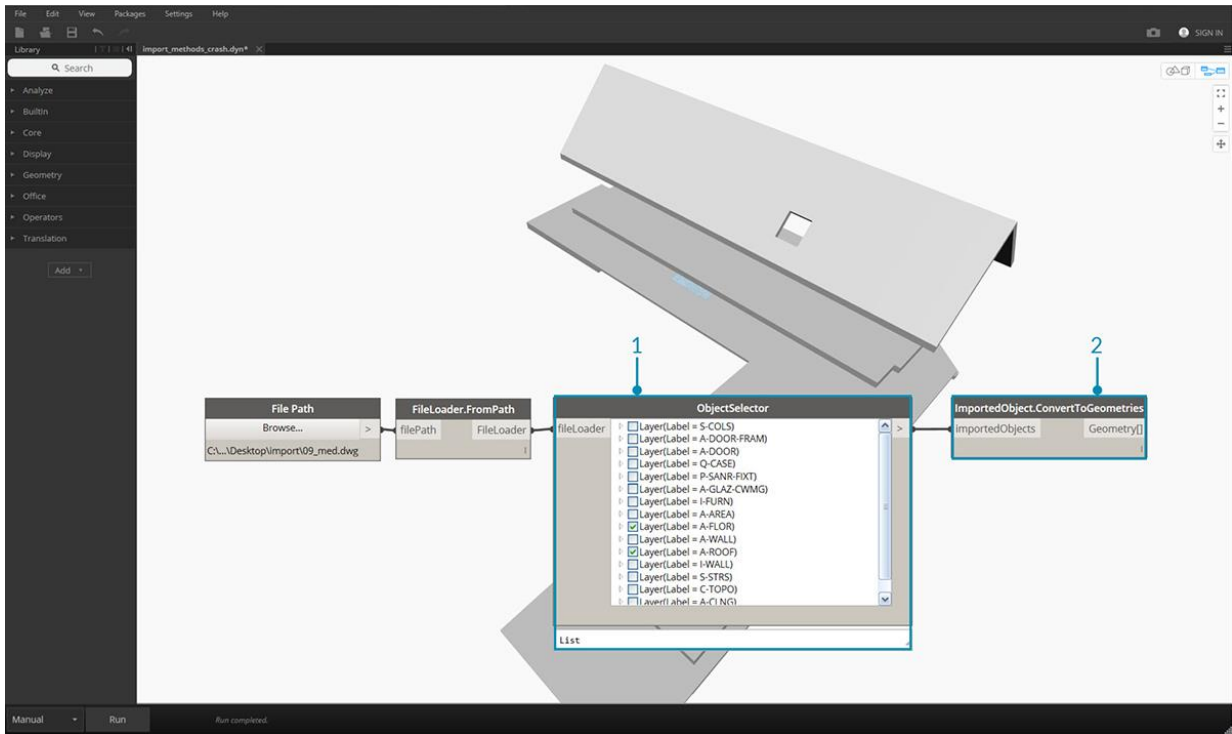


1. Reemplazar **FileLoader.GetImportedObjects** con dos módulos **ObjectFilter** de diferentes instrucciones condicionales. Esto separará la geometría de un archivo en dos flujos diferentes.
2. Conecte el filtro a **ImportedObject.ConvertToGeometries** para importar la geometría filtrada.
3. Conecte **ImportedObject.ConvertToGeometries** a **Display.ByGeometryColor** para visualizar cada flujo en un color diferente.

Selección explícita de objetos

El nodo **ObjectSelector** nos brinda un método alternativo para importar objetos del archivo DWG. En lugar de usar filtros, este método nos dará la posibilidad de elegir específicamente qué objetos y capas se importarán a Dynamo.





1. Reemplazar **FileLoader.GetImportedObjects** con **ObjectSelector** para llamar a capas y objetos específicos dentro de un archivo DWG.
2. Conecte el filtro a **ImportedObject.ConvertToGeometries**.



 DESDE 1988

¿Qué es una lista?

Una lista es una colección de elementos o elementos. Toma un montón de plátanos, por ejemplo. Cada plátano es un artículo dentro de la lista (o grupo). Es más fácil recoger un racimo de plátanos en lugar de cada plátano individualmente, y lo mismo vale para agrupar elementos mediante relaciones paramétricas en una estructura de datos.



Foto de Augustus Binu.

Cuando compramos comestibles, colocamos todos los artículos comprados en una bolsa. Esta bolsa también es una lista. Si estamos haciendo pan de plátano, necesitamos 3 racimos de plátanos (estamos haciendo un *montón* de pan de plátano). La bolsa representa una lista de racimos de plátanos y cada racimo representa una lista de plátanos. La bolsa es una lista de listas (bidimensional) y la banana es una lista (unidimensional).

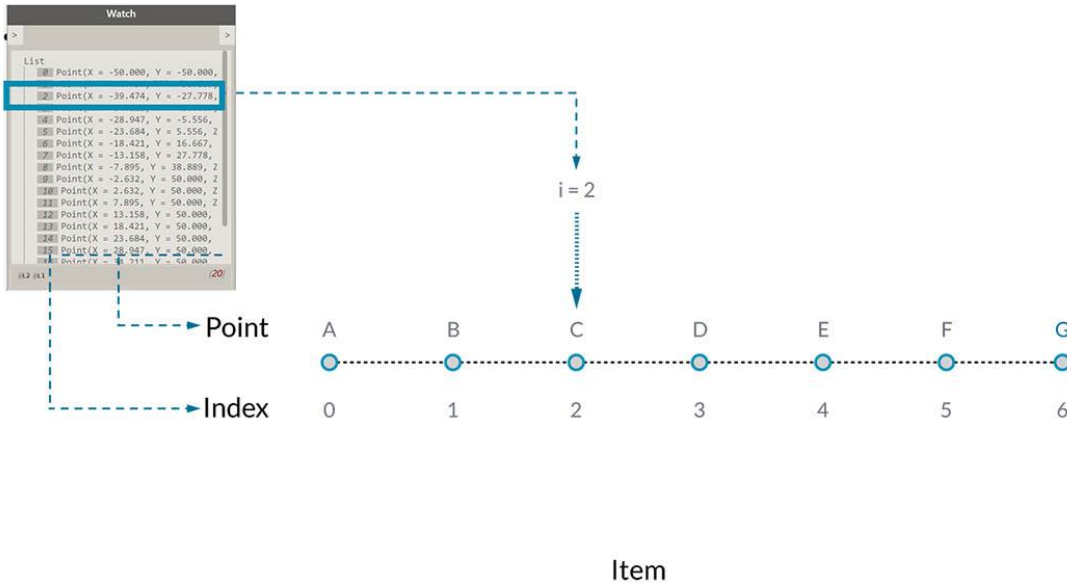
En Dynamo, los datos de la lista se ordenan, y el primer elemento en cada lista tiene un índice "0". A continuación, explicaremos cómo se definen listas en Dynamo y cómo se relacionan múltiples listas entre sí.

Índices basados en cero

Una cosa que puede parecer extraña al principio es que el primer índice de una lista siempre es 0; no 1. Entonces, cuando hablamos sobre el primer ítem de una lista, realmente nos referimos al ítem que corresponde al índice 0.

Por ejemplo, si tuvieras que contar la cantidad de dedos que tenemos en nuestra mano derecha, es probable que hubieras contado de 1 a 5. Sin embargo, si pusieras los dedos en una lista, Dynamo les hubiera dado índices. de 0 a 4. Si bien esto puede parecer un poco extraño para los

principiantes en programación, el índice basado en cero es una práctica estándar en la mayoría de los sistemas de computación.



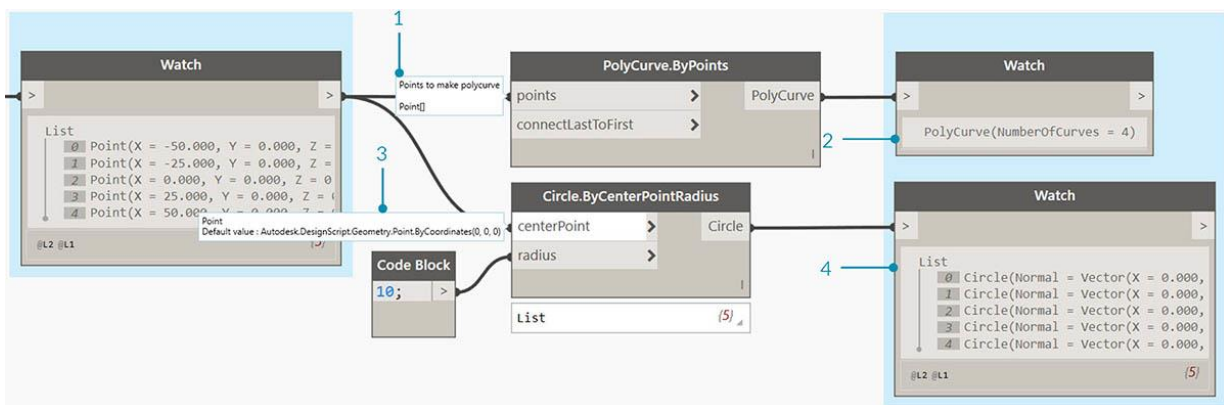
Tenga en cuenta que todavía tenemos 5 elementos en la lista; es solo que la lista está usando un sistema de conteo basado en cero. Y los elementos que se almacenan en la lista no solo tienen que ser números. Pueden ser cualquier tipo de datos compatibles con Dynamo, como puntos, curvas, superficies, familias, etc.

A menudo, la forma más fácil de ver el tipo de datos almacenados en una lista es conectar un nodo de vigilancia a la salida de otro nodo. De forma predeterminada, el nodo de vigilancia muestra automáticamente todos los índices en el lado izquierdo de la lista y muestra los elementos de datos a la derecha.

Estos índices son un elemento crucial cuando se trabaja con listas.

Entradas y salidas

En relación con las listas, las entradas y salidas varían según el nodo de Dynamo que se utilice. A modo de ejemplo, vamos a utilizar una lista de 5 puntos y conectar esta salida a dos diferentes nodos Dynamo: *PolyCurve.ByPoints* y *Circle.ByCenterPointRadius*:



1. La entrada de *puntos* para *PolyCurve.ByPoints* está buscando "*Point []*". Esto representa una lista de puntos.
2. La salida de *PolyCurve.ByPoints* es una única *PolyCurve* creada a partir de una lista de cinco puntos.
3. La entrada *centerPoint* para *Circle.ByCenterPointRadius* pregunta por "*Punto*".
4. La salida para *Circle.ByCenterPointRadius* es una lista de cinco círculos, cuyos centros corresponden a la lista original de puntos.

Los datos de entrada para *PolyCurve.ByPoints* y *Circle.ByCenterPointRadius* son los mismos, sin embargo, el nodo policurva nos da una policurva mientras que el nodo *Circle* nos da 5 círculos con centros en cada punto. Intuitivamente esto tiene sentido: la policurva se dibuja como una curva que conecta los 5 puntos, mientras que los círculos crean un círculo diferente en cada punto. Entonces, ¿qué está pasando con los datos?

Al pasar el *cursor* sobre la entrada de *puntos* para *Polycurve.ByPoints*, vemos que la entrada busca "*Punto []*". Observe los corchetes al final. Esto representa una lista de puntos, y para crear una policurva, la entrada debe ser una lista para cada policurva. Este nodo, por lo tanto, condensará cada lista en una policurva.

Por otro lado, el *centerPoint* entrada para *Circle.ByCenterPointRadius* pide "*Punto*". Este nodo busca un punto, como un elemento, para definir el punto central del círculo. Es por eso por lo que obtenemos cinco círculos de los datos de entrada. Reconocer estas diferencias con las entradas en *Dynamo* ayuda a comprender mejor cómo están funcionando los nodos al administrar los datos.

Cordones

La coincidencia de datos es un problema sin una solución limpia. Ocurre cuando un nodo tiene acceso a entradas de diferentes tamaños. Cambiar el algoritmo de coincidencia de datos puede conducir a resultados muy diferentes.

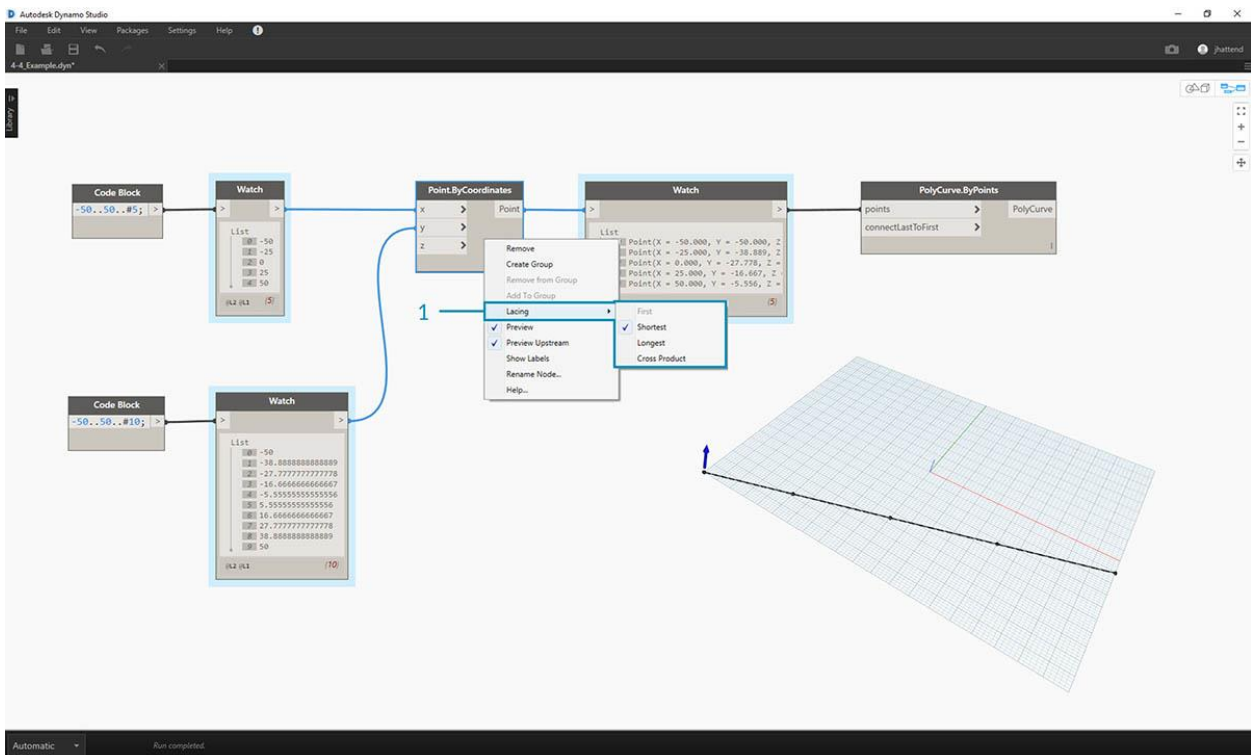
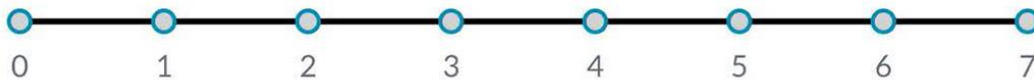
Imagine un nodo que crea segmentos de línea entre puntos (*Line.ByStartPointEndPoint*). Tendrá dos parámetros de entrada que proporcionan coordenadas de puntos:

Como puede ver, existen diferentes formas en que podemos trazar líneas entre estos conjuntos de puntos. Las opciones de ajuste se encuentran haciendo clic con el botón derecho en el centro de un nodo y seleccionando el menú "Cordón".

Archivo base

Descargue el archivo de ejemplo que acompaña a este ejercicio **Lacing.dyn**. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

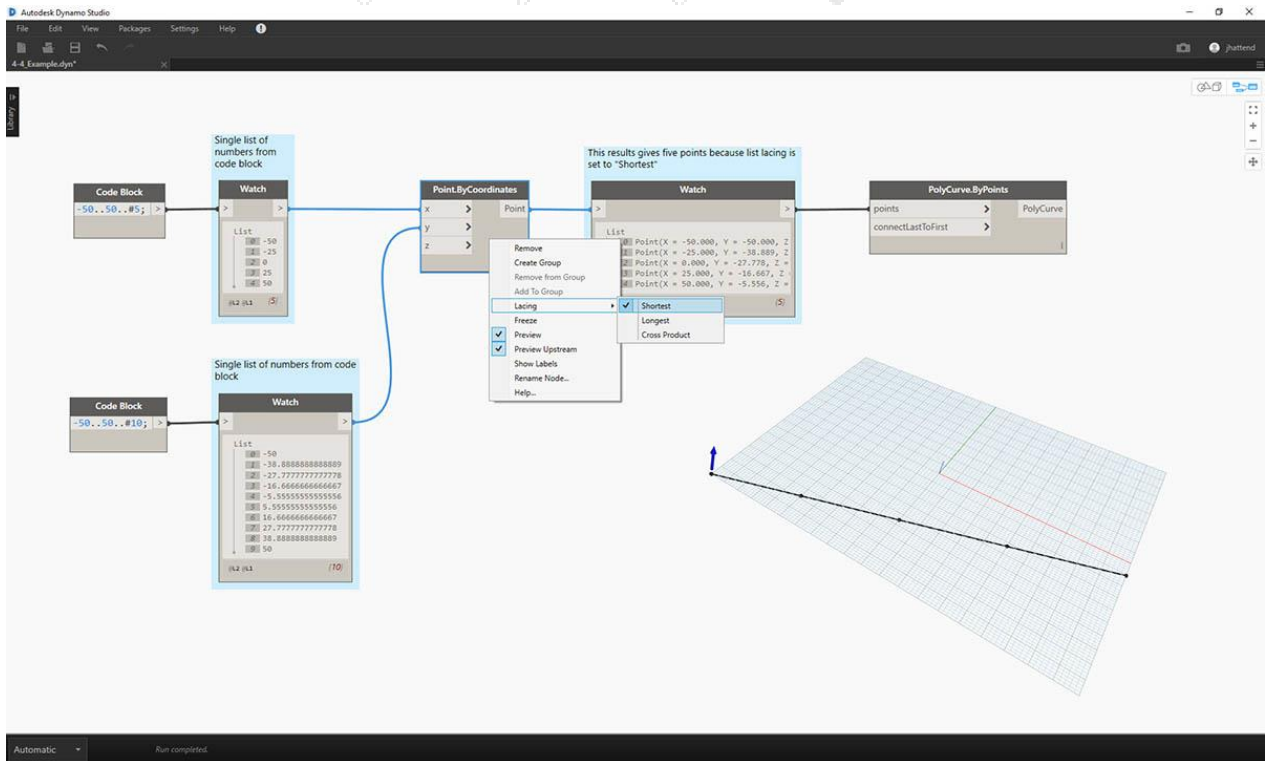
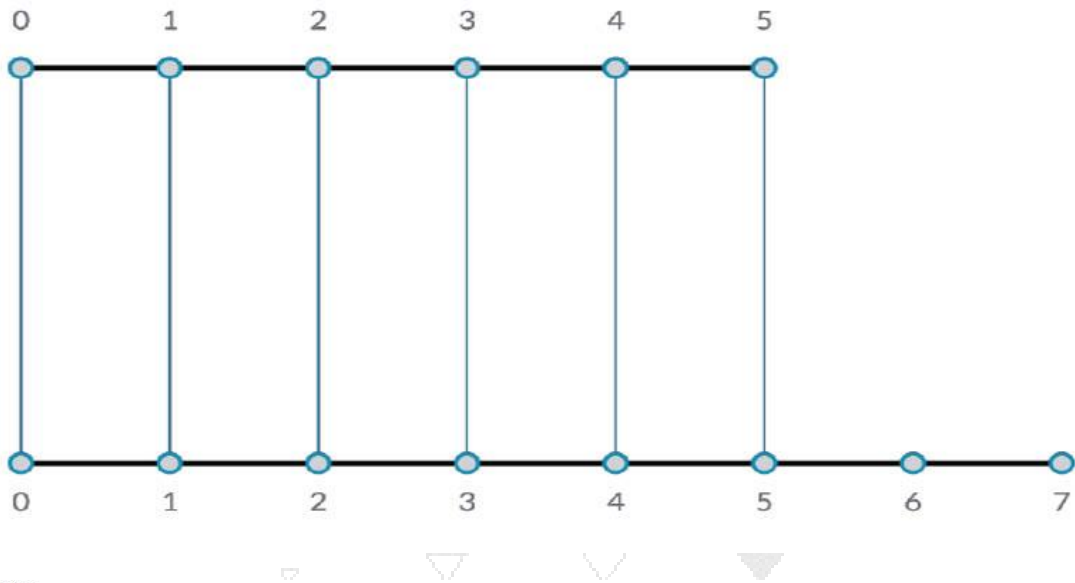
Para demostrar las siguientes operaciones de vinculación, utilizaremos este archivo base para definir la lista más corta, la lista más larga y el producto cruzado.



1. Cambiaremos el cordón en *Point.ByCoordinates*, pero no cambiaremos nada más en el gráfico anterior.

Lista más corta

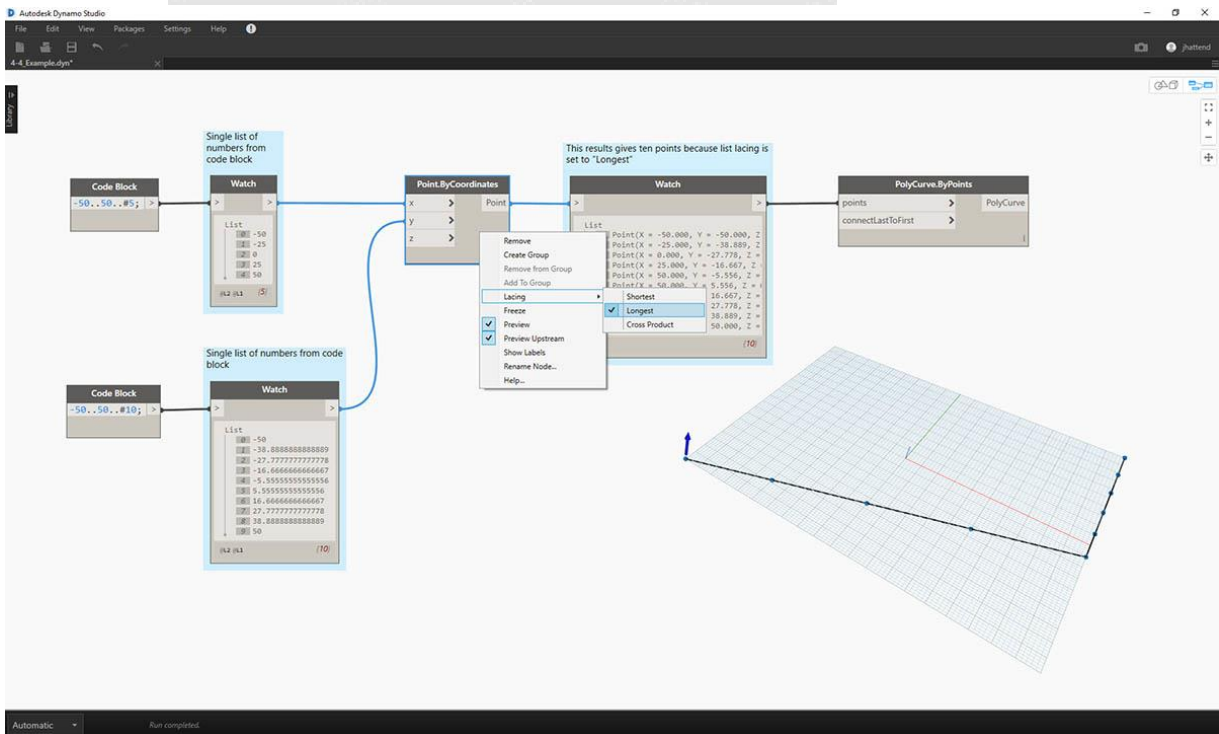
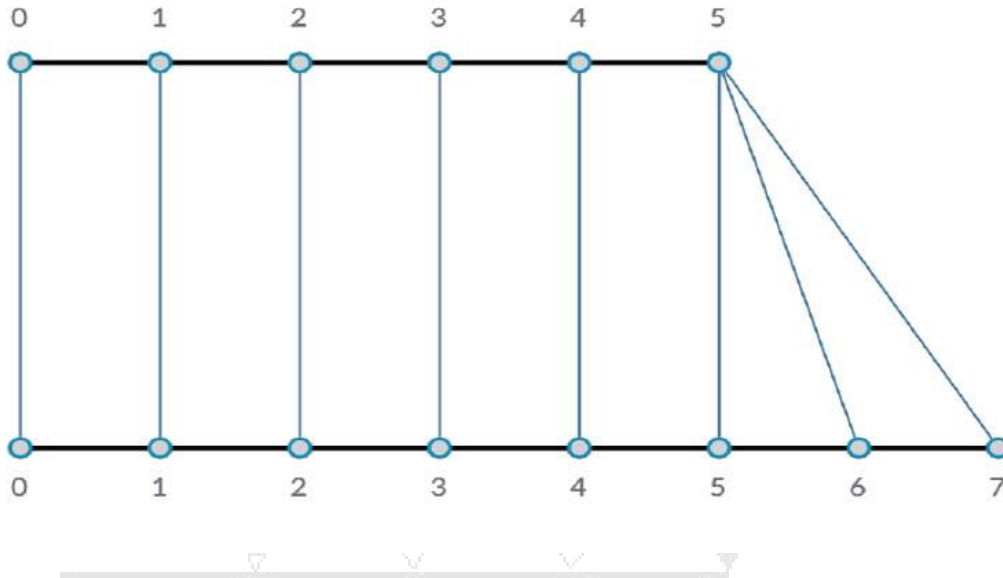
La forma más simple es conectar las entradas uno a uno hasta que una de las secuencias se seque. Esto se llama el algoritmo de "Lista más corta". Este es el comportamiento predeterminado para los nodos Dynamo:



Al cambiar el cordón a la *lista más corta*, obtenemos una línea diagonal básica compuesta por cinco puntos. Cinco puntos es la longitud de la lista menor, por lo que el cordón de la lista más corta se detiene después de que llega al final de una lista.

La lista más larga

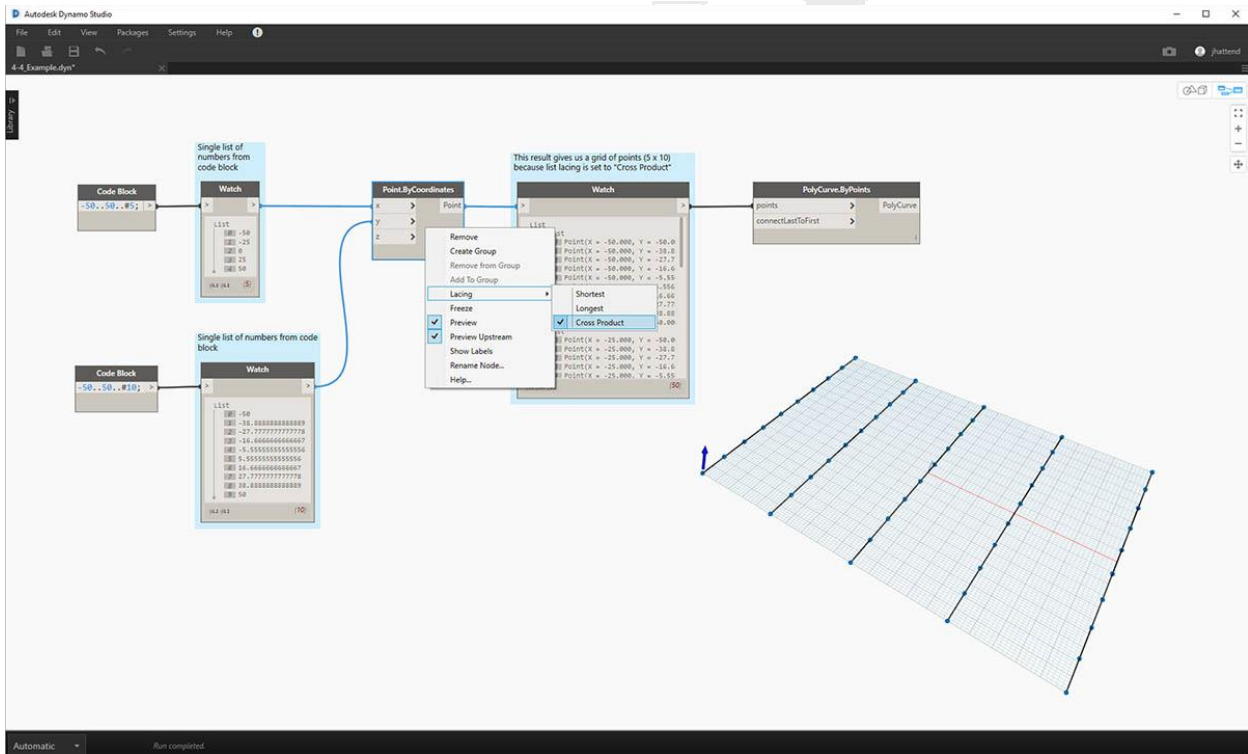
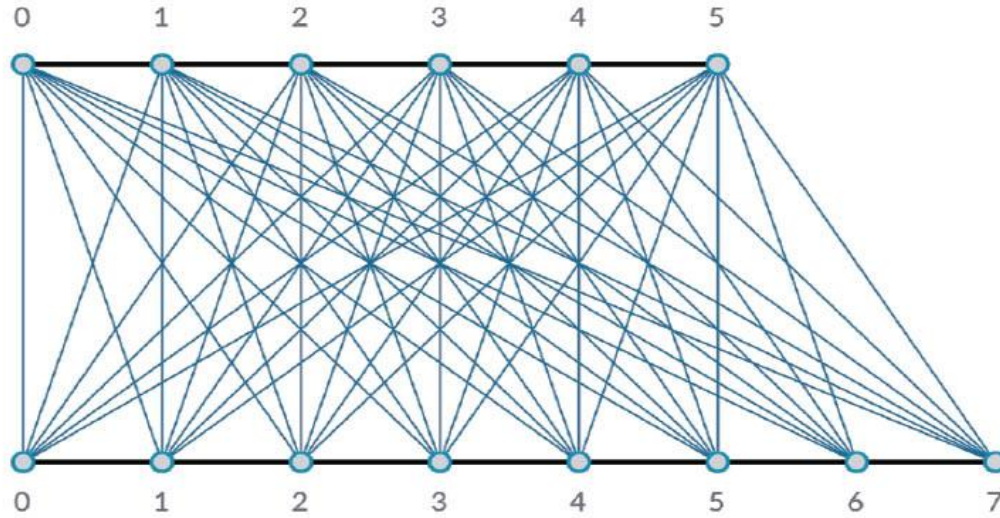
El algoritmo de "Lista más larga" sigue conectando entradas, reutilizando elementos, hasta que todas las transmisiones se agoten:



Al cambiar el cordón a la *lista más larga*, obtenemos una línea diagonal que se extiende verticalmente. Con el mismo método que el diagrama conceptual, se repetirá el último elemento de la lista de 5 elementos para alcanzar la longitud de la lista más larga.

Producto cruzado

Finalmente, el método "Cross Product" hace todas las conexiones posibles:



Al cambiar el cordón a *Cross Product*, obtenemos cada combinación entre cada lista, lo que nos da una grilla de 5x10 puntos. Esta es una estructura de datos equivalente al producto cruzado como se muestra en el diagrama de concepto anterior, excepto que ahora nuestros datos son una

lista de listas. Al conectar una policurva, podemos ver que cada lista se define por su valor X, que nos da una fila de líneas verticales.

Trabajando con listas

Ahora que hemos establecido lo que es una lista, hablemos de las operaciones que podemos realizar en ella. Imagina una lista como un mazo de cartas. Un mazo es la lista y cada naípe representa un ítem.



Foto de Christian Gidlöf

¿Qué **preguntas** podemos hacer de la lista? Esto accede a las propiedades existentes.

- ¿Número de cartas en el mazo? 52.
- ¿Número de trajes? 4.
- ¿Material? Papel.
- ¿Longitud? 3.5 "o 89 mm.
- ¿Anchura? 2.5 "o 64 mm.

¿Qué **acciones** podemos realizar en la lista? Esto cambia la lista en función de una operación determinada.

- Podemos barajar el mazo.
- Podemos ordenar el mazo por valor.
- Podemos ordenar el mazo por palo.
- Podemos dividir el mazo.
- Podemos dividir el mazo repartiendo las manos individuales.
- Podemos seleccionar una carta específica en el mazo.

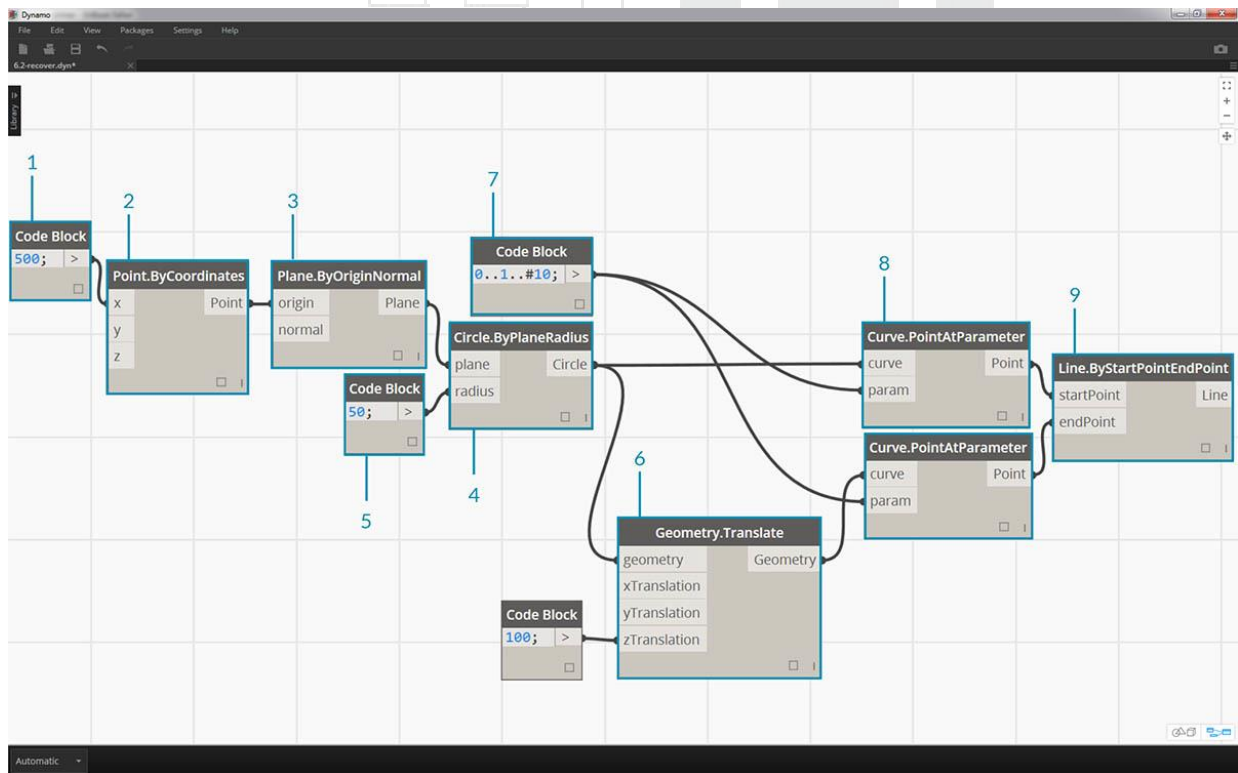
Todas las operaciones enumeradas anteriormente tienen nodos Dynamo análogos para trabajar con listas de datos genéricos. Las lecciones a continuación demostrarán algunas de las operaciones fundamentales que podemos realizar en las listas.

Operaciones de lista

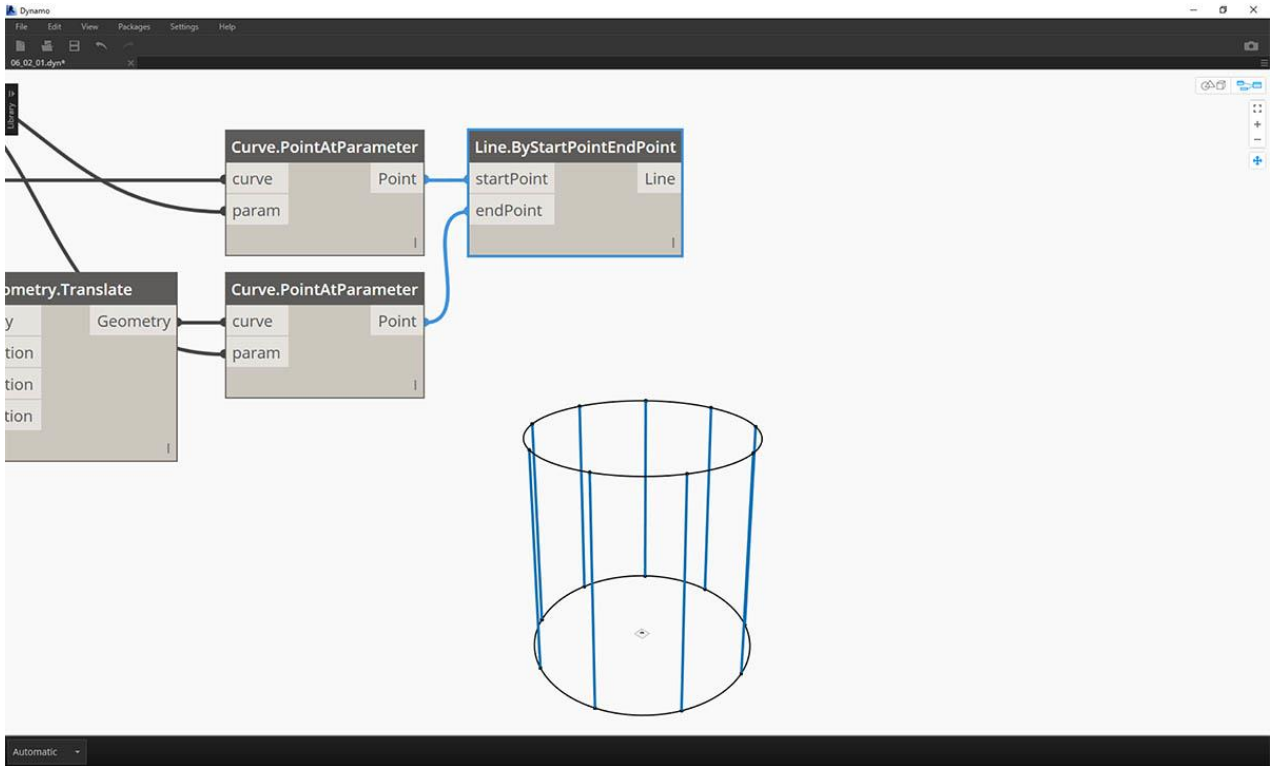
La imagen a continuación es el gráfico base que usaremos para demostrar las operaciones básicas de la lista. Exploraremos cómo administrar los datos dentro de una lista y demostrar los resultados visuales.

Ejercicio - Operaciones de lista

Descargue el archivo de ejemplo que acompaña a este ejercicio [List-Operations.dyn](#). Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

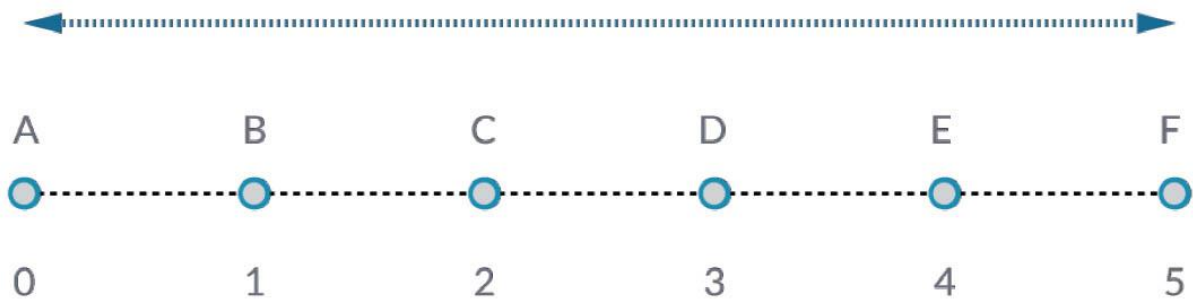


1. Comience con un *bloque de código* con un valor de 500;
2. Enchufe en la entrada *x* de un nodo *Point.ByCoordinates*.
3. Conecte el nodo del paso anterior a la entrada de origen de un nodo *Plane.ByOriginNormal*.
4. Usando un nodo *Circle.ByPlaneRadius*, conecte el nodo del paso anterior a la entrada del plano.
5. Usando el *bloque de código*, designe un valor de 50; para el *radio*. Este es el primer círculo que crearemos.
6. Con un nodo *Geometry.Translate*, mueva el círculo hacia arriba 100 unidades en la dirección Z.
7. Con un nodo de *bloque de código*, defina un rango de diez números entre 0 y 1 con esta línea de código: `0..1..#10;`
8. Conecte el bloque de código del paso anterior a la entrada *param* de dos nodos *Curve.PointAtParameter*. Conecte *Circle.ByPlaneRadius* en la entrada de curva del nodo superior y *Geometry.Translate* en la entrada de curva del nodo debajo de él.
9. Con un *Line.ByStartPointEndPoint*, conecte los dos nodos *Curve.PointAtParameter*.



1. Un nodo *Watch3D* muestra los resultados de *Line.ByStartPointEndPoint*. Estamos dibujando líneas entre dos círculos para representar las operaciones básicas de la lista y usaremos este gráfico base Dynamo para recorrer las acciones de la lista a continuación.

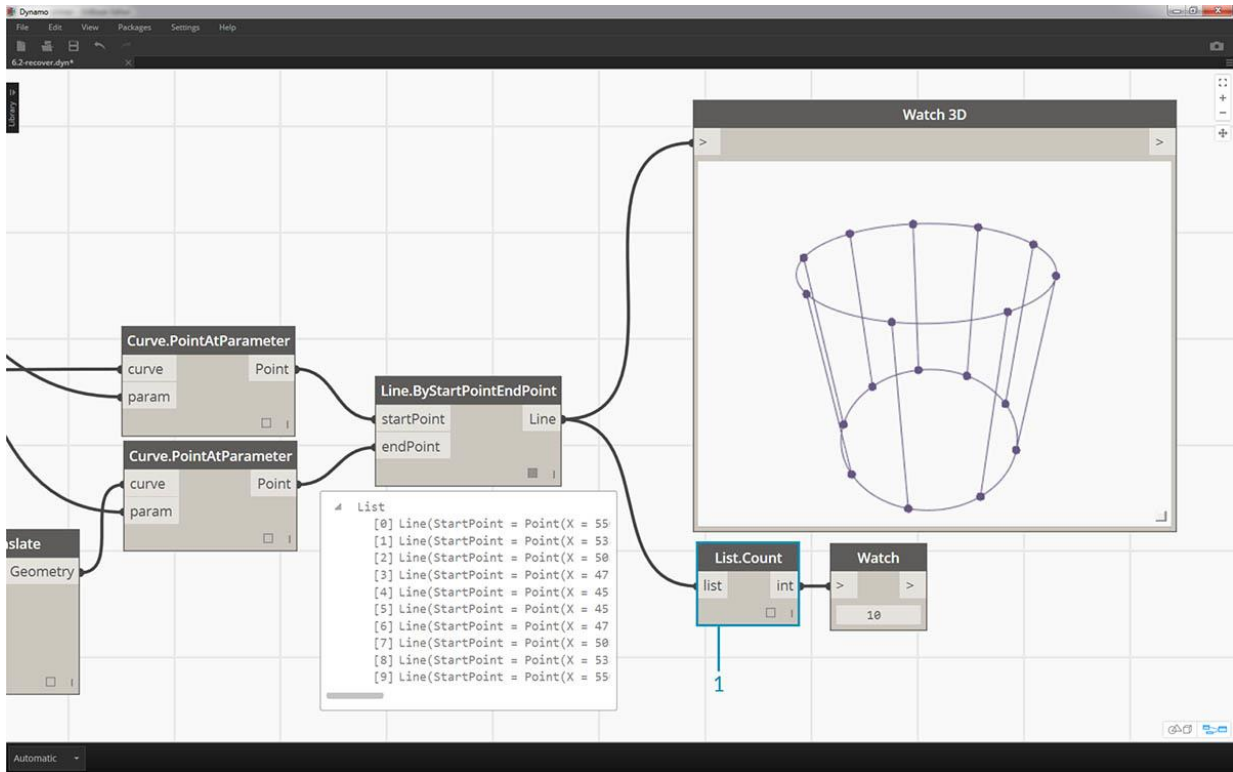
List.Count



El nodo *List.Count* es sencillo: cuenta el número de valores en una lista y devuelve ese número. Este nodo se vuelve más matizado a medida que trabajamos con listas de listas, pero lo demostraremos en las próximas secciones.

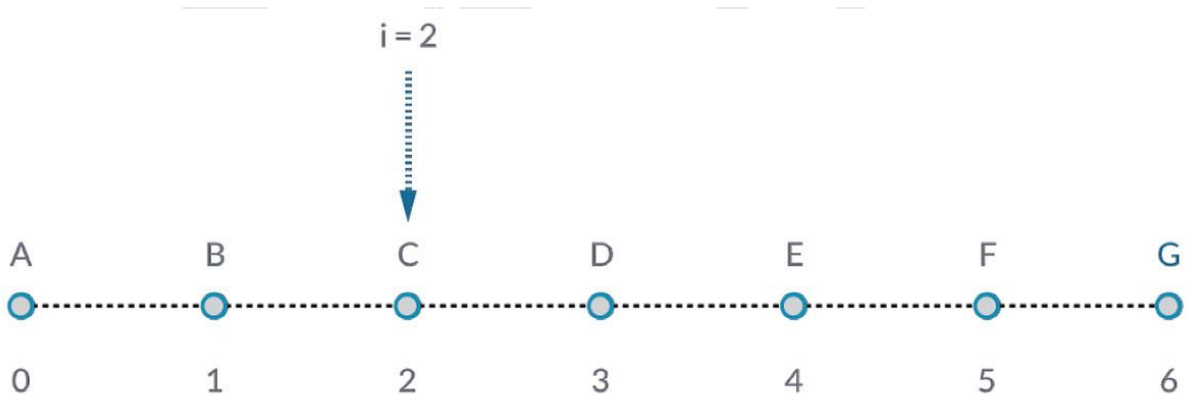
Ejercicio - List.Count

Descargue el archivo de ejemplo que acompaña este ejercicio List-Count.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



1. El nodo *List.Count* devuelve el número de líneas en el nodo *Line.ByStartPointEndPoint*. El valor es 10 en este caso, que coincide con la cantidad de puntos creados a partir del nodo de *bloque de código* original.

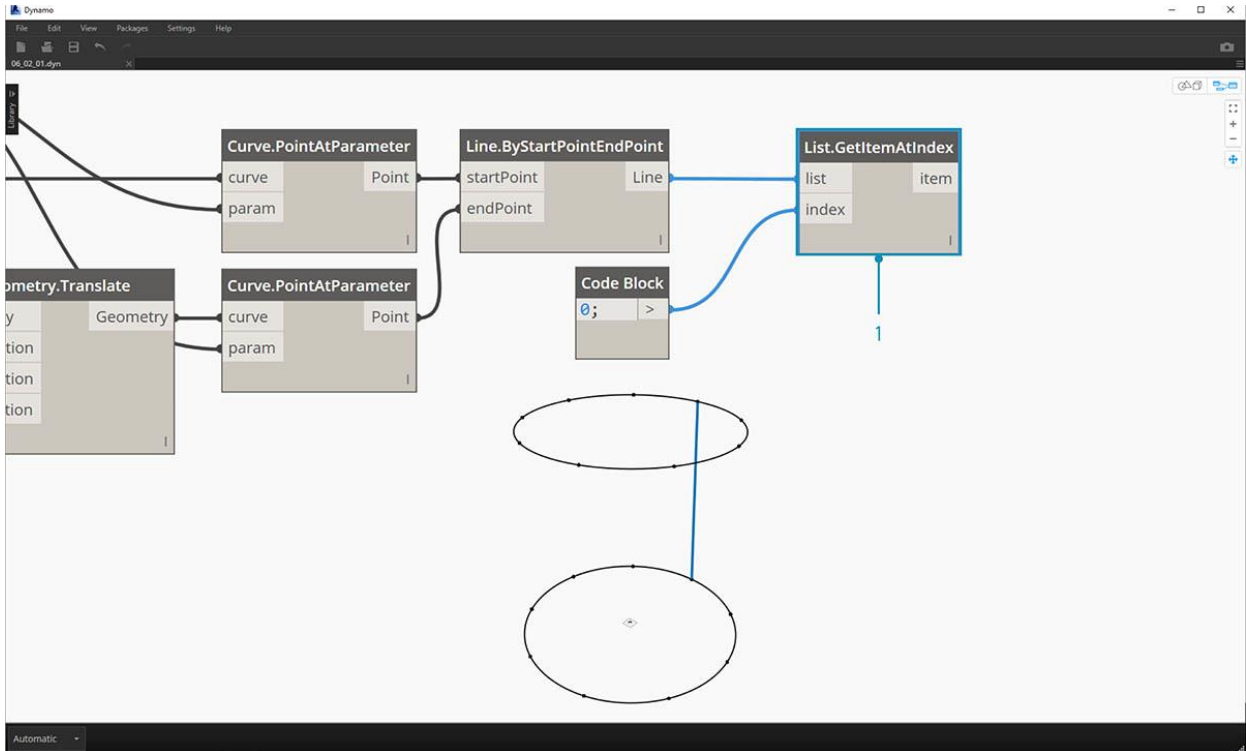
List.GetItemAtIndex



List.GetItemAtIndex es una forma fundamental de consultar un elemento en la lista. En la imagen de arriba, estamos usando un índice de "2" para consultar el punto etiquetado como "C".

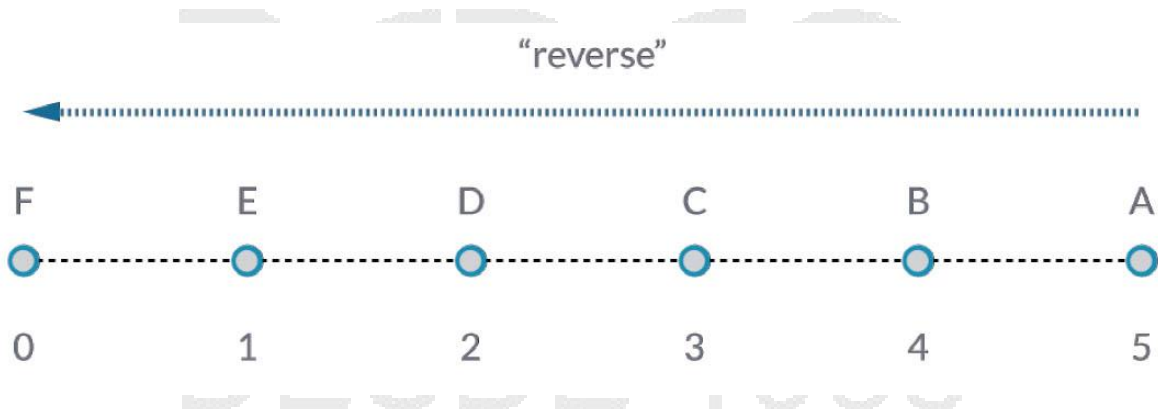
Ejercicio - List.GetItemAtIndex

Descargue el archivo de ejemplo que acompaña a este ejercicio *List-GetItemAtIndex.dyn*. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



1. Usando el nodo *List.GetItemAtIndex*, estamos seleccionando el índice "0", o el primer elemento en la lista de líneas.
2. El nodo *Watch3D* revela que hemos seleccionado una línea. Nota: para obtener la imagen de arriba, asegúrese de desactivar la vista previa de *Line.ByStartPointEndPoint*.

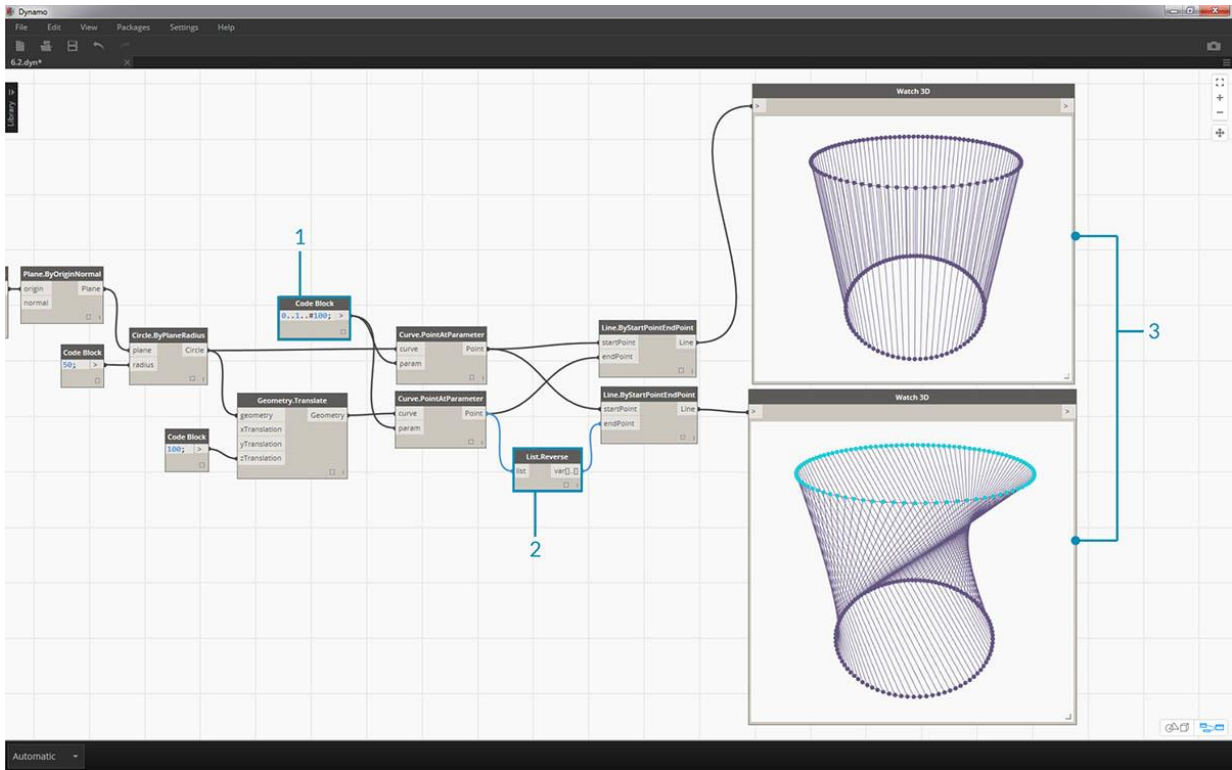
List.Reverse



List.Reverse invierte el orden de todos los elementos en una lista.

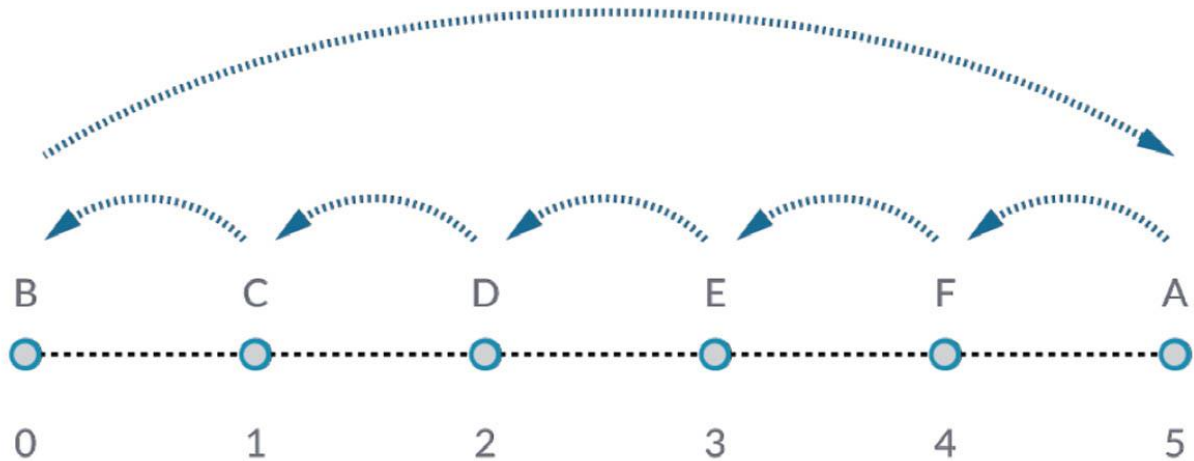
Ejercicio - List.Reverse

Descargue el archivo de ejemplo que acompaña a este ejercicio List-Reverse.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



1. Para visualizar correctamente la lista invertida de líneas, cree más líneas cambiando el bloque de código a `0..1..#100`;
2. Inserte un nodo `List.Reverse` entre `Curve.PointAtParameter` y `Line.ByStartPointEndPoint` para una de la lista de puntos.
3. Los nodos `Watch3D` muestran dos resultados diferentes. El primero muestra el resultado sin una lista invertida. Las líneas se conectan verticalmente a los puntos vecinos. La lista invertida, sin embargo, conectará todos los puntos al orden opuesto en la otra lista.

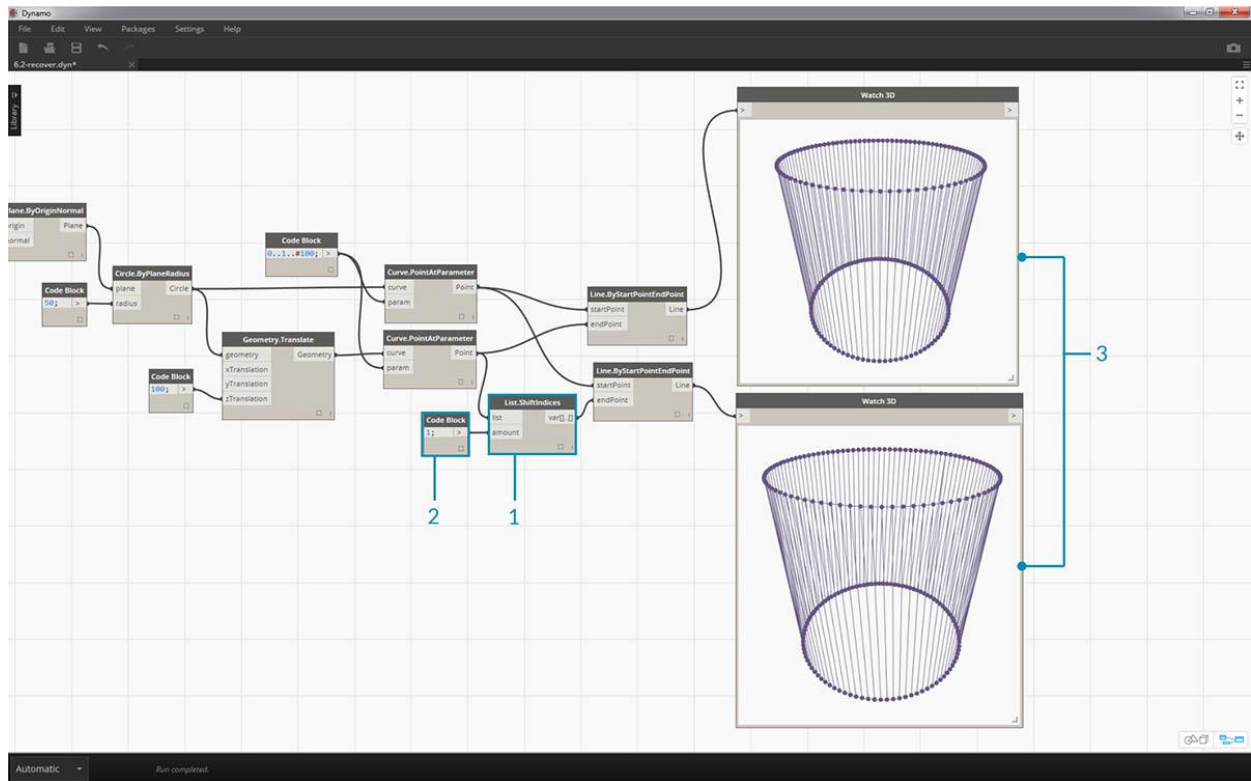
`List.ShiftIndices`



`List.ShiftIndices` es una buena herramienta para crear giros o patrones helicoidales, o cualquier otra manipulación de datos similar. Este nodo desplaza los elementos en una lista un número dado de índices.

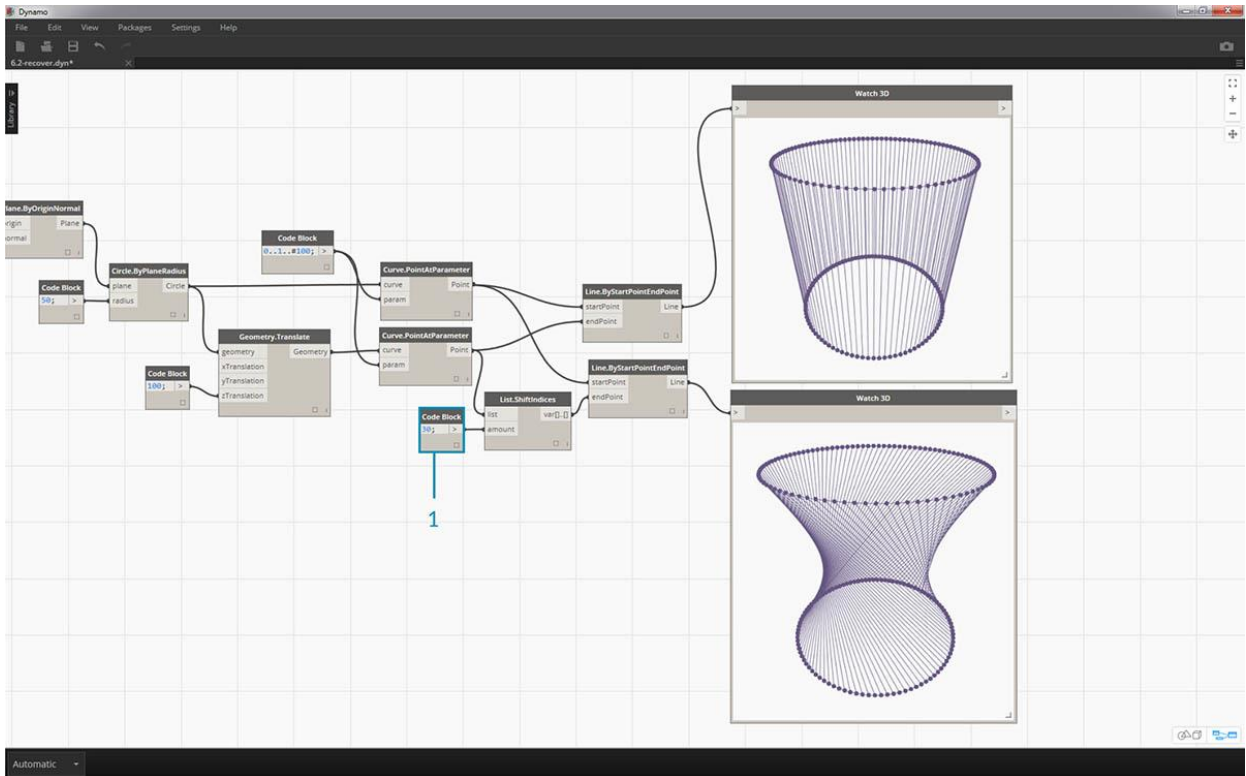
Ejercicio - List.ShiftIndices

Descargue el archivo de ejemplo que acompaña a este ejercicio [List-ShiftIndices.dyn](#). Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



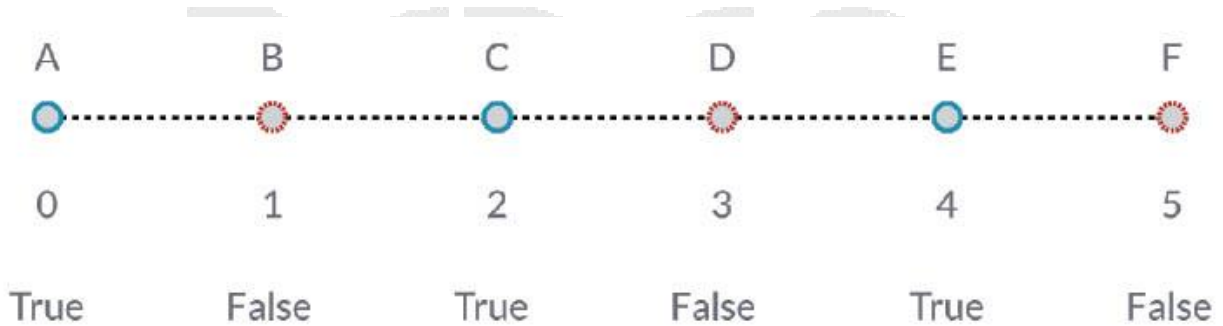
1. En el mismo proceso que la lista inversa, inserte un *List.ShiftIndices* en *Curve.PointAtParameter* y *Line.ByStartPointEndPoint*.
2. Al usar un *bloque de código*, se designa un valor de "1" para cambiar el índice de la lista a uno.
3. Tenga en cuenta que el cambio es sutil, pero todas las líneas del nodo *Watch3D* inferior han cambiado un índice cuando se conectan al otro conjunto de puntos.

DARCO
DESDE 1988



1. Al cambiar a *bloque de código* a un *valor mayor*, "30" por ejemplo, notamos una diferencia significativa en las líneas diagonales. El cambio funciona como el iris de una cámara en este caso, creando un giro en la forma cilíndrica original.

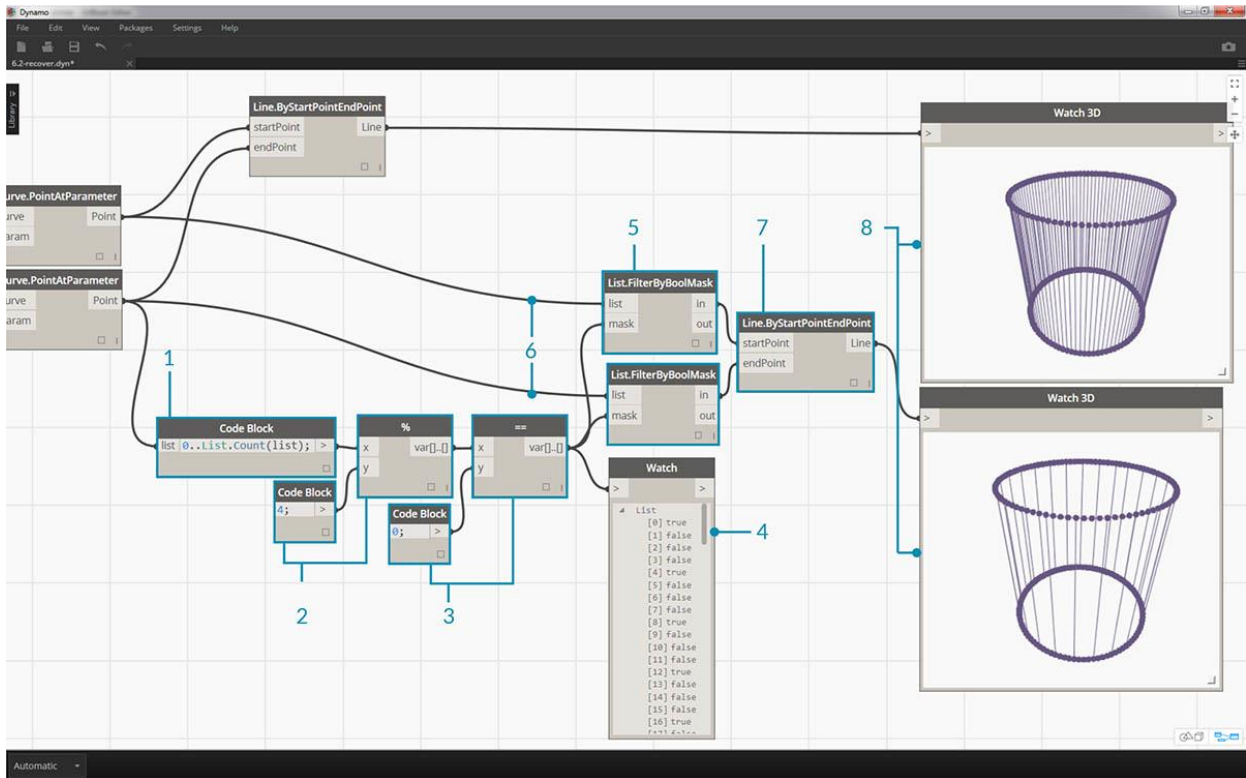
List.FilterByBooleanMask



List.FilterByBooleanMask eliminará ciertos elementos según una lista de booleanos, o valores que digan "verdadero" o "falso".

Ejercicio - List.FilterByBooleanMask

Descargue el archivo de ejemplo que acompaña este ejercicio [List-FilterByBooleanMask.dyn](#). Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



Para crear una lista de valores que lea "verdadero" o "falso", necesitamos un poco más de trabajo

1. El uso de un *bloque de código*, definir una expresión con la sintaxis: `0..List.Count(list);` Conecte el nodo *Curve.PointAtParameter* a la entrada de la *lista*. Analizaremos más esta configuración en el capítulo del bloque de código, pero la línea de código en este caso nos proporciona una lista que representa cada índice del nodo *Curve.PointAtParameter*.
2. Usando un nodo "%" (módulo), conecte la salida del *bloque de código* en la entrada *x*, y un valor de 4 en la entrada *y*. Esto nos dará el resto al dividir la lista de índices por 4. El módulo es un nodo realmente útil para la creación de patrones. Todos los valores se leerán como los posibles residuos de 4: 0, 1, 2, 3.
3. Desde el nodo *módulo*, sabemos que un valor de 0 significa que el índice es divisible por 4 (0,4,8, etc ...). Al usar un nodo "=", podemos probar la divisibilidad probándola contra un valor de "0".
4. El *reloj de nodo* revela esto: tenemos un patrón de verdadero / falso, que dice: *verdadero, falso, falso, falso ...*
5. Con este patrón verdadero / falso, conéctese a la entrada de máscara de dos nodos *List.FilterByBooleanMask*.
6. Conecte el nodo *Curve.PointAtParameter* en cada entrada de lista para *List.FilterByBooleanMask*.
7. La salida de *Filter.ByBooleanMask* lee "in" y "out". "In" representa valores que tenían un valor de máscara de "verdadero", mientras que "fuera" representa valores que tenían un valor de "falso". Al conectar las salidas "in" en las entradas *startPoint* y *endPoint* de un nodo *Line.ByStartPointEndPoint*, hemos creado una lista filtrada de líneas.
8. El nodo *Watch3D* revela que tenemos menos líneas que puntos. ¡Hemos seleccionado solo el 25% de los nodos filtrando solo los valores verdaderos!

Listas de listas

Agreguemos un nivel más a la jerarquía. Si tomamos el mazo de cartas del ejemplo original y creamos un cuadro que contiene múltiples mazos, el cuadro ahora representa una lista de mazos, y cada mazo representa una lista de cartas. Esta es una lista de listas. Para la analogía en esta sección, el cuadro rojo a continuación contiene una lista de rollos de monedas, y cada rollo contiene una lista de monedas de un centavo.



Foto de Dori.

¿Qué **consultas** podemos hacer desde la lista de listas? Esto accede a las propiedades existentes.

- ¿Cantidad de tipos de moneda? 2.
- ¿Valores de tipo de moneda? \$ 0.01 y \$ 0.25.
- ¿Material de cuartos? 75% de cobre y 25% de níquel.
- ¿Material de centavos? 97.5% zinc y 2.5% cobre.

¿Qué **acciones** podemos realizar en la lista de listas? Esto cambia la lista de listas basadas en una operación dada.

- Seleccione una pila específica de cuartos o centavos.
- Seleccione un cuarto específico o centavo.
- Reorganiza las pilas de monedas y centavos.
- Mezcla las pilas.

De nuevo, Dynamo tiene un nodo análogo para cada una de las operaciones anteriores. Dado que estamos trabajando con datos abstractos y no con objetos físicos, necesitamos un conjunto de reglas para controlar cómo movemos hacia arriba y hacia abajo en la jerarquía de datos.

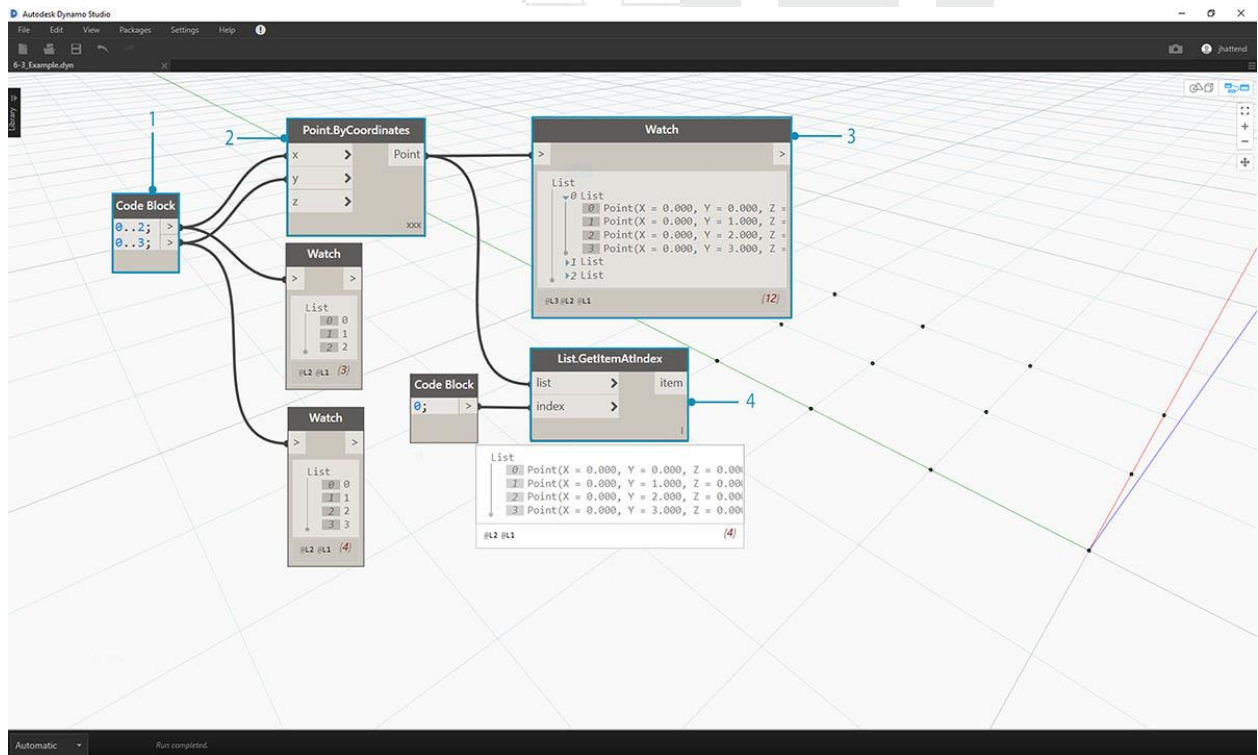
Cuando se trata de listas de listas, los datos son complejos y complejos, pero esto brinda la oportunidad de realizar algunas operaciones paramétricas increíbles. Analicemos los fundamentos y analicemos algunas operaciones más en las lecciones a continuación.

Jerarquía descendente

El concepto fundamental para aprender de esta sección: **Dynamo trata las listas como objetos en sí mismos**. Esta jerarquía de arriba hacia abajo se desarrolla con la programación orientada a objetos en mente. En lugar de seleccionar subelementos con un comando como `List.GetItemAtIndex`, Dynamo seleccionará ese índice de la lista principal en la estructura de datos. Y ese artículo puede ser otra lista. Vamos a descomponerlo con una imagen de ejemplo:

Ejercicio - Jerarquía descendente

Descargue el archivo de ejemplo que acompaña este ejercicio `Top-Down-Hierarchy.dyn`. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



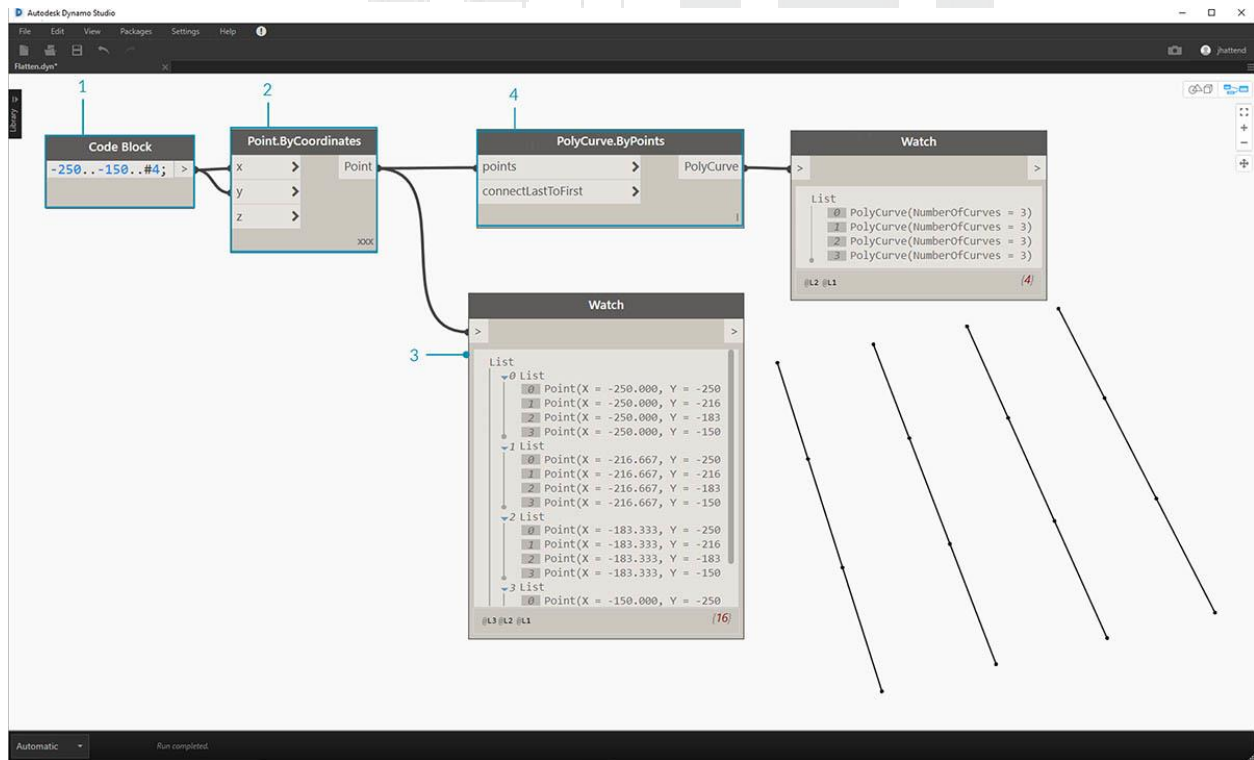
1. Con el *bloque de código*, hemos definido dos rangos: `` `0..2; 0..3; `` `
2. Estos rangos están conectados a un nodo `Point.ByCoordinates` con el conjunto de lazos en "Producto Cruzado". Esto crea una grilla de puntos y también devuelve una lista de listas como salida.
3. Observe que el nodo `Watch` proporciona 3 listas con 4 elementos en cada lista.
4. Al usar `List.GetItemAtIndex`, con un índice de 0, Dynamo selecciona la primera lista y todos sus contenidos. Otros programas pueden seleccionar el primer elemento de cada lista en la estructura de datos, pero Dynamo emplea una jerarquía de arriba hacia abajo cuando se trata de datos.

Acoplar y List.Flatten

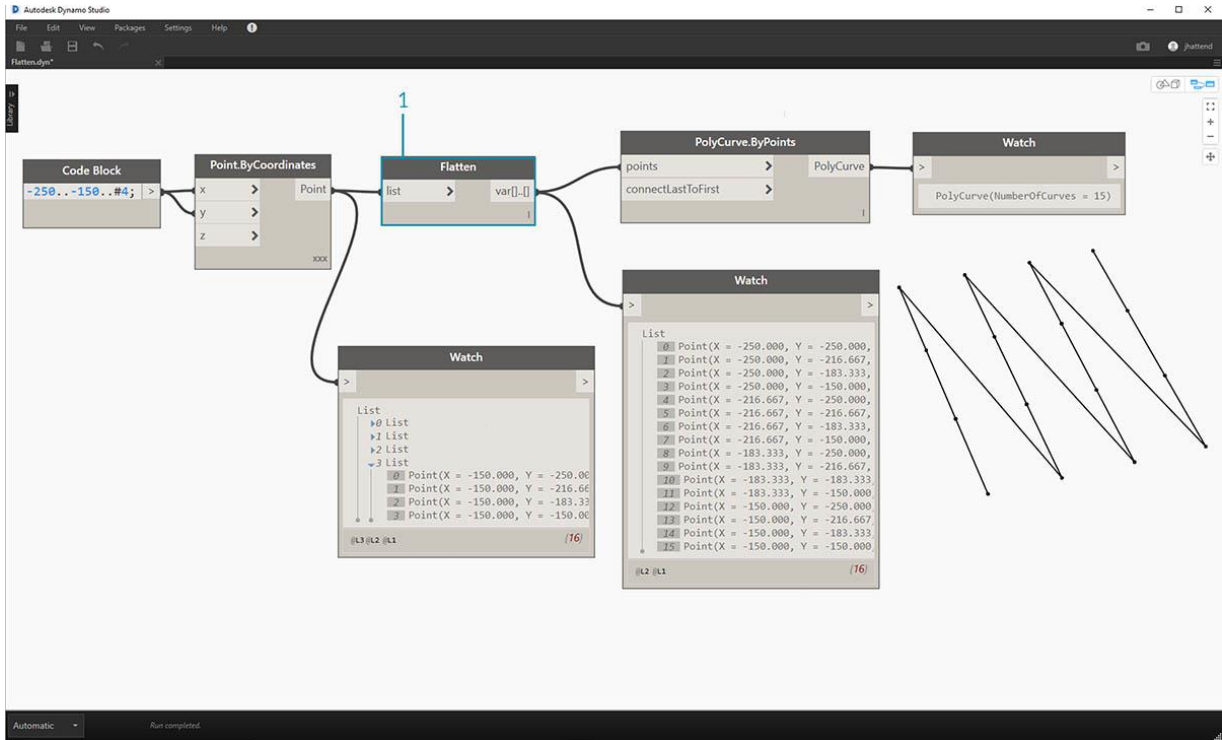
Acoplar elimina todos los niveles de datos de una estructura de datos. Esto es útil cuando las jerarquías de datos no son necesarias para su operación, pero puede ser arriesgado porque elimina información. El siguiente ejemplo muestra el resultado de aplanar una lista de datos.

Ejercicio: aplanar

Descargue el archivo de ejemplo que acompaña a este ejercicio **Flatten.dyn**. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



1. Inserte una línea de código para definir un rango en el *bloque de código* : `` ` -250 .. -150 .. #4; ``
2. Enchufar el *bloque de código* en el X y Y de entrada de un *Point.ByCoordinates* nodo, nos fijamos el cordón de "Cruz del producto" para obtener una rejilla de puntos.
3. El nodo *Watch* muestra que tenemos una lista de listas.
4. Un nodo *PolyCurve.ByPoints* hará referencia a cada lista y creará una policurva respectiva. Observe en la vista previa de Dynamo que tenemos cuatro policurvas que representan cada fila en la grilla.



1. Al insertar un *aplanar* antes del nodo polycurve, creamos una lista única para todos los puntos. El nodo polycurve hace referencia a una lista para crear una curva, y como todos los puntos están en una lista, obtenemos una curva policrible en zig-zag que se ejecuta en toda la lista de puntos.

También hay opciones para aplanar niveles de datos aislados. Con el nodo List.Flatten, puede definir un número determinado de niveles de datos para aplanar desde la parte superior de la jerarquía. Esta es una herramienta realmente útil si está luchando con estructuras de datos complejas que no son necesariamente relevantes para su flujo de trabajo. Y otra opción es usar el nodo aplanar como una función en List.Map. Discutiremos List.Map más a continuación.

Picar

Cuando se utiliza el modelado paramétrico, también hay momentos en los que deseará agregar más estructura de datos a una lista existente. Hay muchos nodos disponibles para esto también, y chop es la versión más básica. Con chop, podemos dividir una lista en sublistas con un número determinado de elementos.

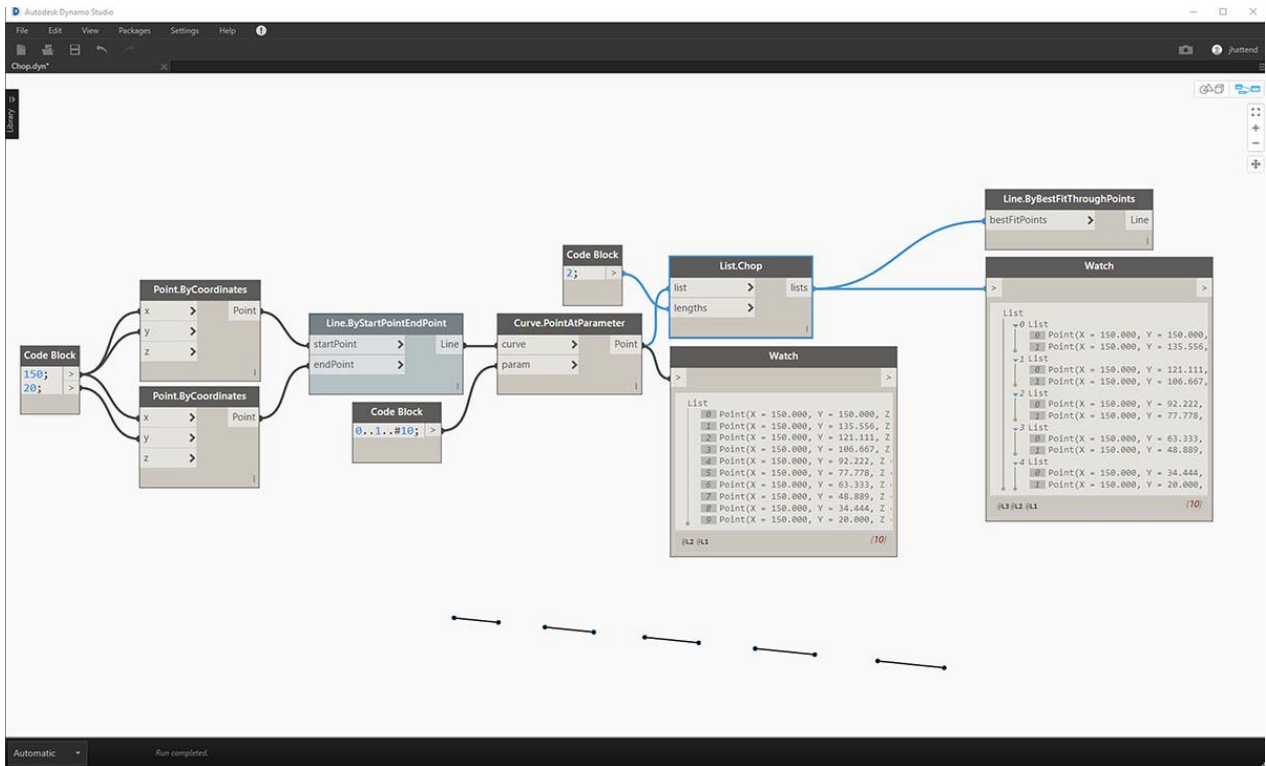
Ejercicio - List.Chop

Descargue el archivo de ejemplo que acompaña a este Chop. Dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset .



Un List.Chop _with un _subLength of 2 crea 4 listas con 2 elementos cada una.

El comando chop divide listas basadas en una longitud de lista dada. De alguna manera, cortar es lo contrario de aplanar: en lugar de eliminar la estructura de datos, agrega nuevos niveles a ella. Esta es una herramienta útil para operaciones geométricas como el siguiente ejemplo.



List.Map y List.Combine

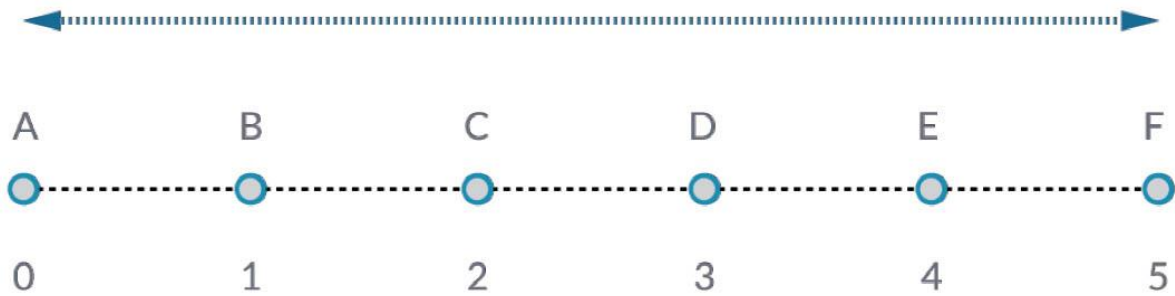
Un List.Map/Combine aplica una función de conjunto a una lista de entrada, pero un paso hacia abajo en la jerarquía. Las combinaciones son las mismas que las de Mapas, excepto que las combinaciones pueden tener múltiples entradas correspondientes a la entrada de una función dada.

Ejercicio - Lista.Mapa

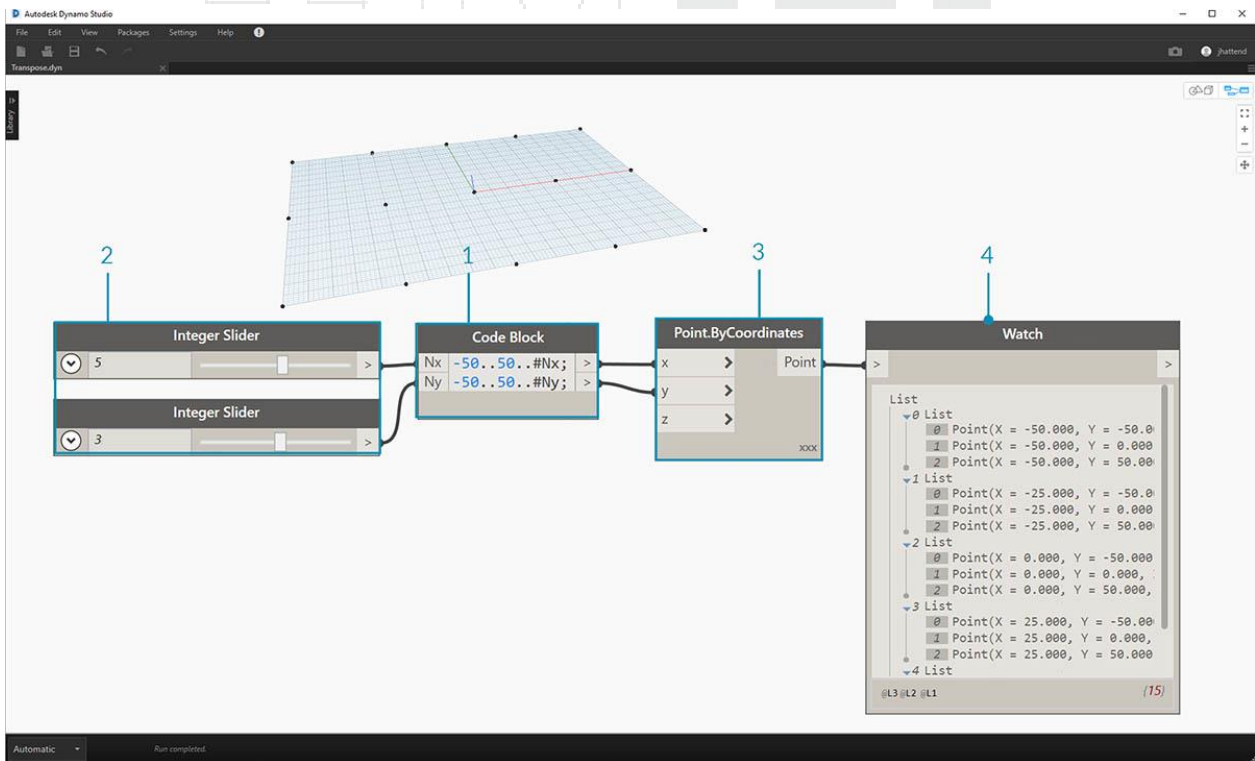
Nota: Este ejercicio se creó con una versión anterior de Dynamo. Gran parte de la funcionalidad de List.Map se ha resuelto con la adición de la función List @ Level. Para obtener más información, vea [List @ Level](#) a continuación.

Descargue el archivo de ejemplo que acompaña a este ejercicio **Map.dyn**. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset .

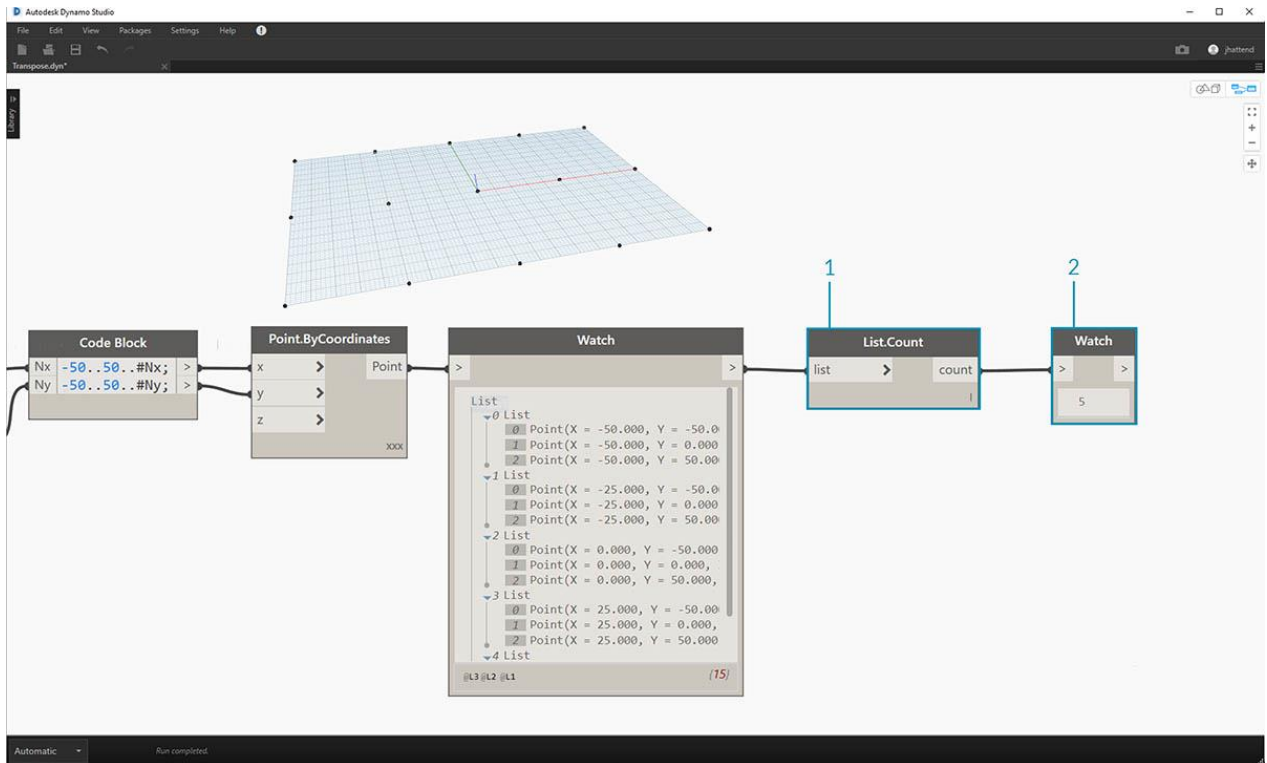
Como introducción rápida, revisemos el nodo List.Count de una sección anterior.



El nodo *List.Count* cuenta todos los elementos en una lista. Lo usaremos para demostrar cómo funciona *List.Map*.



1. Inserta dos líneas de código en el *bloque de código* :
2. `-50..50..#Nx;`
3. `-50..50..#Ny;`
Después de escribir este código, el bloque de código creará dos entradas para *Nx* y *Ny*.
4. Con dos *controles deslizantes enteros* , defina los valores *Nx* y *Ny* conectándolos al *bloque de código* .
5. Conecte cada línea del bloque de código en las respectivas entradas *X* e *Y* de un nodo *Point.ByCoordinates*. Haga clic derecho en el nodo, seleccione "Cordón" y elija "Producto cruzado". Esto crea una grilla de puntos. Debido a que definimos el rango de -50 a 50, estamos abarcando la cuadrícula de Dynamo predeterminada.
6. Un nodo de *vigilancia* revela los puntos creados. Observe la estructura de datos. Hemos creado una lista de listas. Cada lista representa una fila de puntos de la grilla.

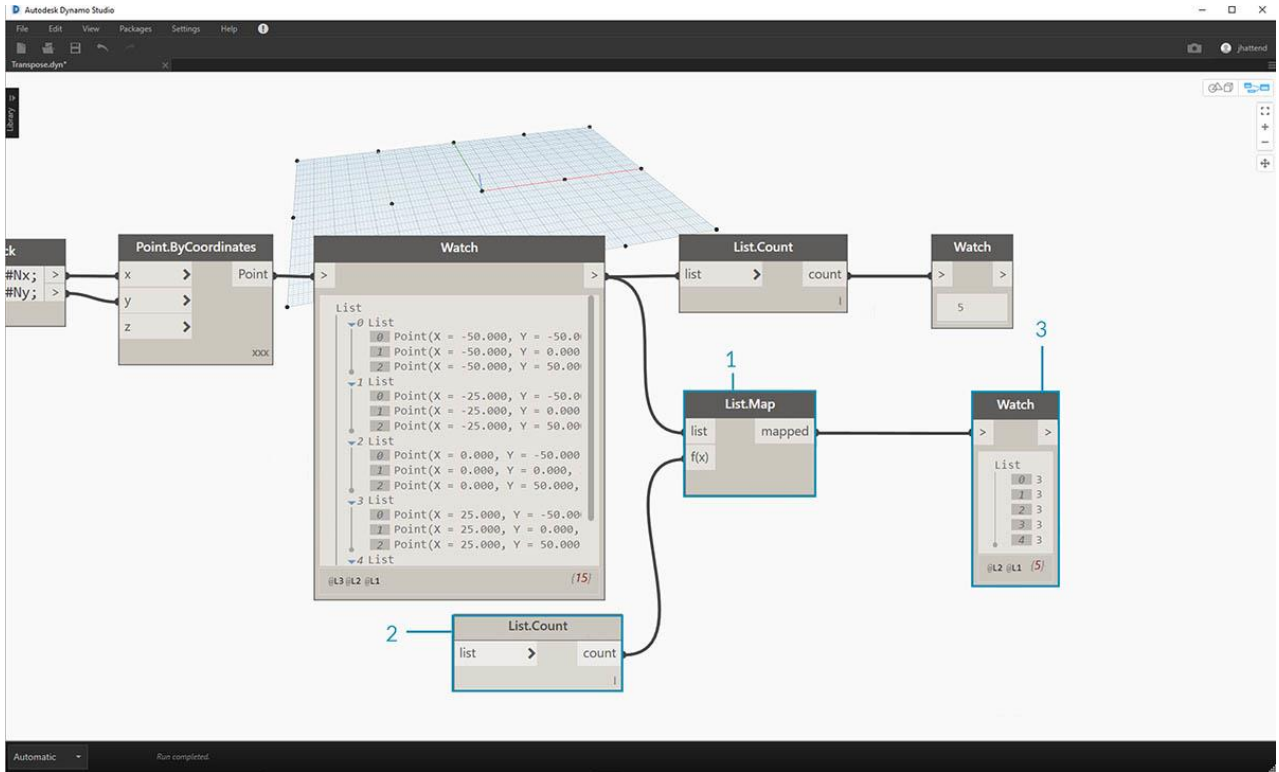


1. Adjunte un nodo *List.Count* a la salida del nodo watch desde el paso anterior.
2. Conecte un nodo *Watch* a la salida *List.Count*.

Observe que el nodo *List.Count* da un valor de 5. Esto es igual a la variable "Nx" como se define en el bloque de código. ¿Por qué es esto?

- Primero, el nodo *Point.ByCoordinates* utiliza la entrada "x" como entrada principal para crear listas. Cuando Nx es 5 y Ny es 3, obtenemos una lista de 5 listas, cada una con 3 elementos.
- Como Dynamo trata las listas como objetos en sí mismas, se aplica un nodo *List.Count* a la lista principal en la jerarquía. El resultado es un valor de 5 o el número de listas en la lista principal.

DARCO
DESDE 1988



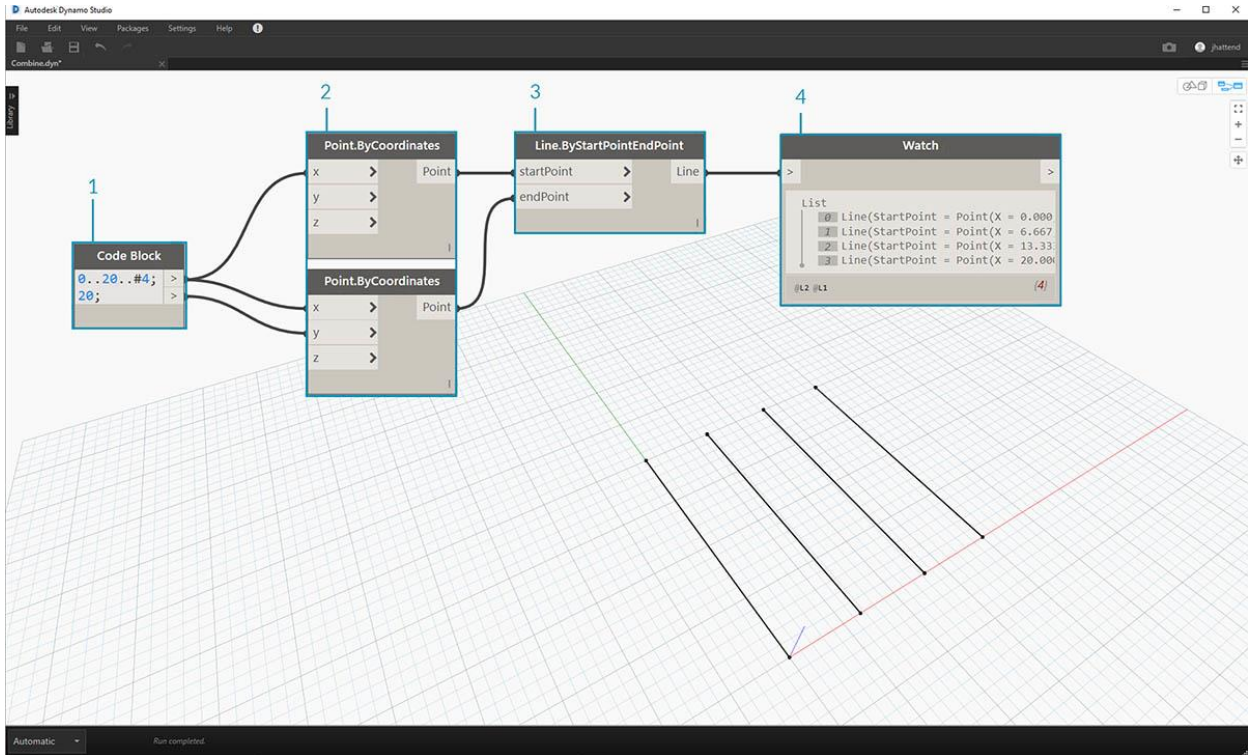
1. Al usar un nodo *List.Map*, damos un paso hacia abajo en la jerarquía y realizamos una "función" en este nivel.
2. Tenga en cuenta que el nodo *List.Count* no tiene entrada. Se está utilizando como una función, por lo que el nodo *List.Count* se aplicará a cada lista individual un paso hacia abajo en la jerarquía. La entrada en blanco de *List.Count* corresponde a la entrada de lista de *List.Map*.
3. Los resultados de *List.Count* ahora proporcionan una lista de 5 elementos, cada uno con un valor de 3. Esto representa la longitud de cada sublista.

Ejercicio - List.Combine

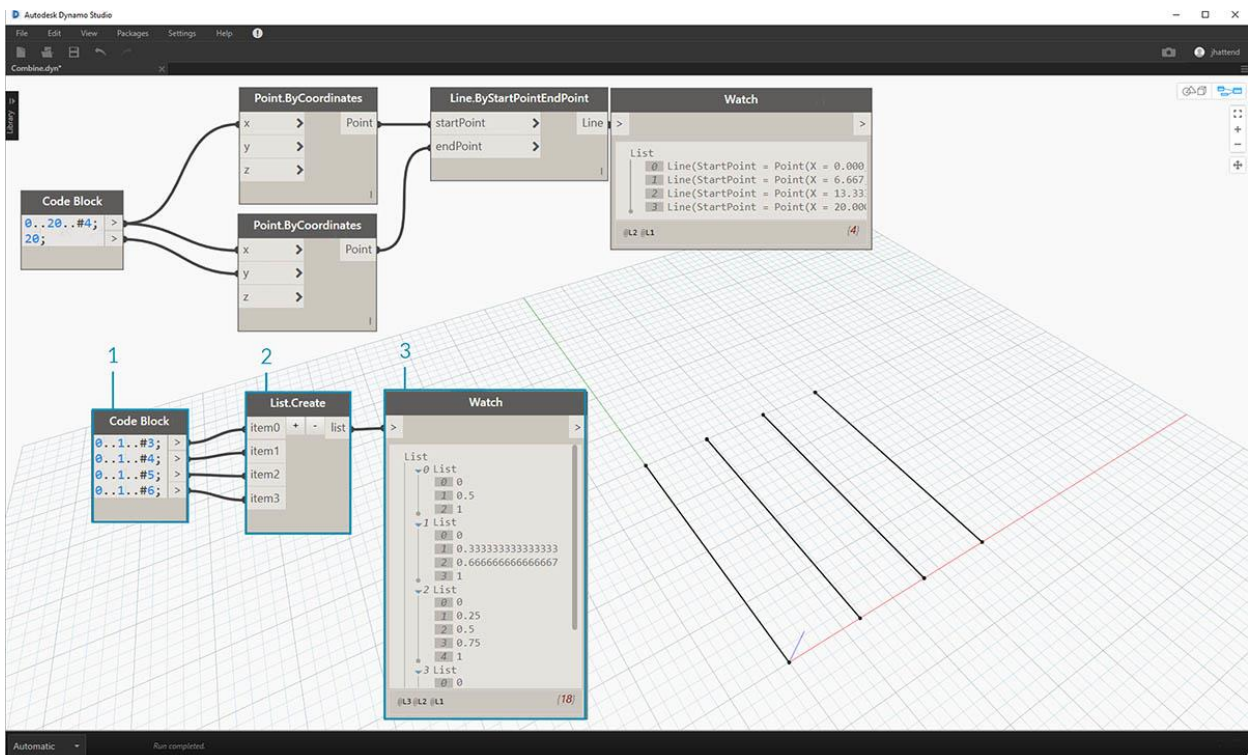
Nota: Este ejercicio se creó con una versión anterior de Dynamo. Gran parte de la funcionalidad de List.Combine se ha resuelto con la adición de la función List @ Level. Para obtener más información, vea [List @ Level](#) a continuación.

Descargue el archivo de ejemplo que acompaña a este ejercicio **Combine**. Dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

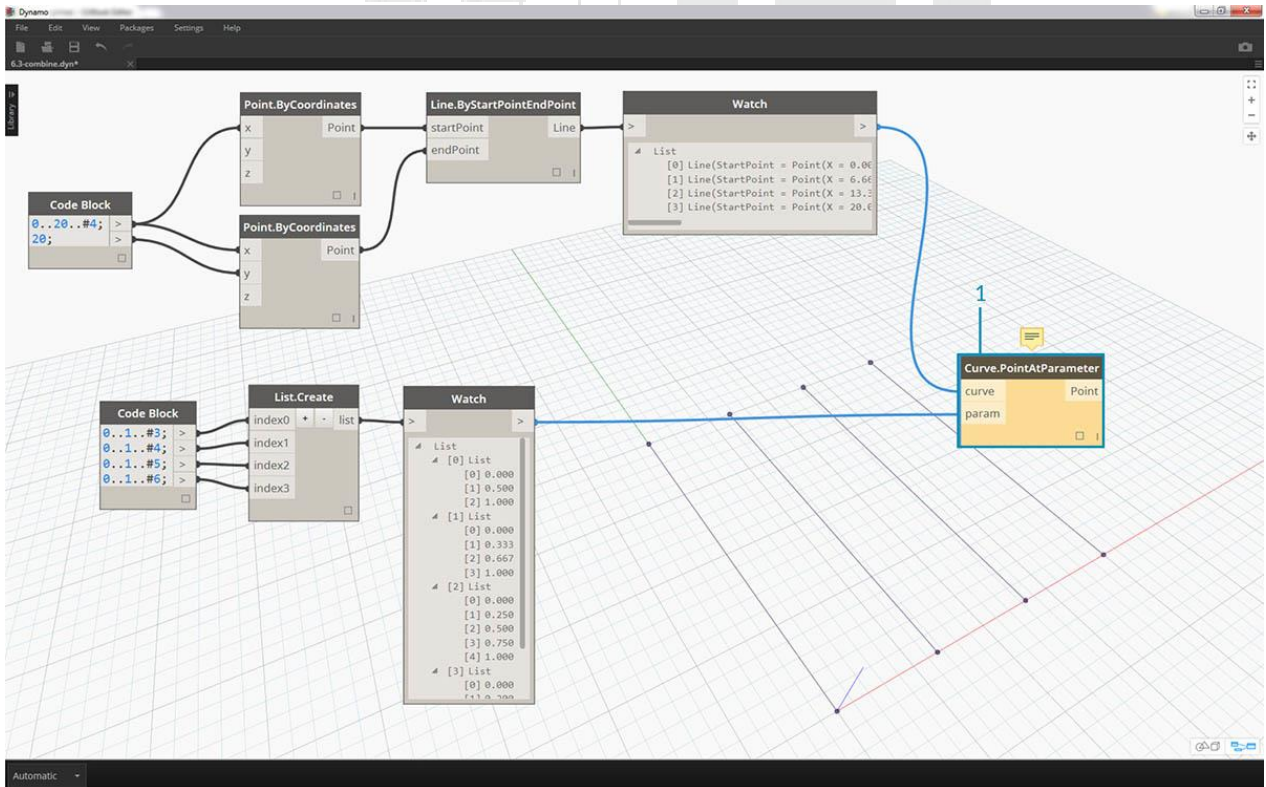
En este ejercicio, usaremos una lógica similar a *List.Map*, pero con múltiples elementos. En este caso, queremos dividir una lista de curvas por un número único de puntos.



1. Usando el *bloque de código*, defina un rango usando la sintaxis: ``` `..20 .. # 4;` and a value of `20;` debajo de esa línea.
2. Conecte el *bloque de código* a dos nodos *Point.ByCoordinates*.
3. Cree un *Line.ByStartPointEndPoint* desde los nodos *Point.ByCoordinates*.
4. El nodo *Watch* muestra cuatro líneas.

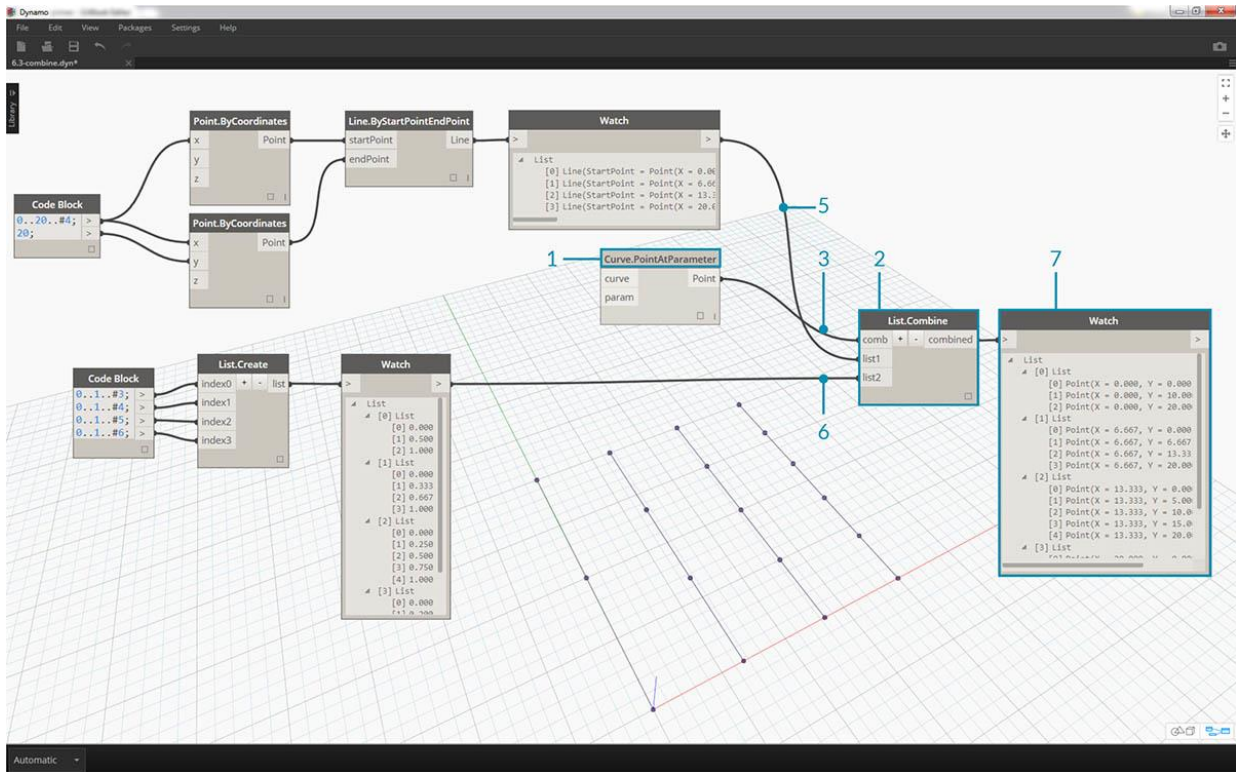


1. Debajo del gráfico para la creación de líneas, queremos usar `_code block _` para crear cuatro rangos distintos para dividir las líneas de manera única. Hacemos esto con las siguientes líneas de código:
2. `0..1..#3;`
3. `0..1..#4;`
4. `0..1..#5;`
5. `0..1..#6;`
6. Con un nodo `List.Create`, fusionamos las cuatro líneas del *bloque de código* en una lista.
7. El nodo `Watch` revela una lista de listas.



1. `Curve.PointAtParameter` no funcionará al conectar las líneas directamente en los valores de los *parámetros*. Necesitamos bajar un nivel en la jerarquía. Para esto, usaremos `List.Combine`.

DARCO
DESDE 1988



Al usar *List.Combine*, podemos dividir con éxito cada línea por los rangos dados. Esto se pone un poco complicado, por lo que lo analizaremos en profundidad.

1. Primero, agregue un nodo *Curve.PointAtParameter* al lienzo. Esta será la "función" o "combinador" que aplicamos al nodo *_List.Combine*. Más sobre esto en un segundo.
2. Agregue un nodo *List.Combine* al lienzo. Presione el "+" o "-" para sumar o restar entradas. En este caso, usaremos las dos entradas predeterminadas en el nodo.
3. Queremos conectar el nodo *Curve.PointAtParameter* en la entrada "comb" de *List.Combine*. Y un nodo más importante: asegúrese de hacer clic derecho en el parámetro "param" *_input de Curve.PointAtParameter* y desmarque "use default value". Los valores predeterminados en las entradas de Dynamo deben eliminarse cuando se ejecuta un nodo como una función. En otras palabras, deberíamos considerar que los valores predeterminados tienen nodos adicionales conectados a ellos. Debido a esto, necesitamos eliminar los valores predeterminados en este caso.
4. Sabemos que tenemos dos entradas, las líneas y los parámetros para crear puntos. Pero ¿cómo los conectamos a la lista? ¿Combina entradas y en qué orden?
5. Las entradas vacías de *Curve.PointAtParameter*, de arriba a abajo, deben completarse en el combinador en el mismo orden. Entonces, las líneas están conectadas a *list1* de *List.Combine*.
6. Siguiendo el ejemplo, los valores de los parámetros se enchufan en la *list2* entrada de *List.Combine*.
7. El nodo *Watch* y la vista previa de Dynamo nos muestran que tenemos 4 líneas, cada una dividida en función de los rangos del *bloque de código*.

List @ Level

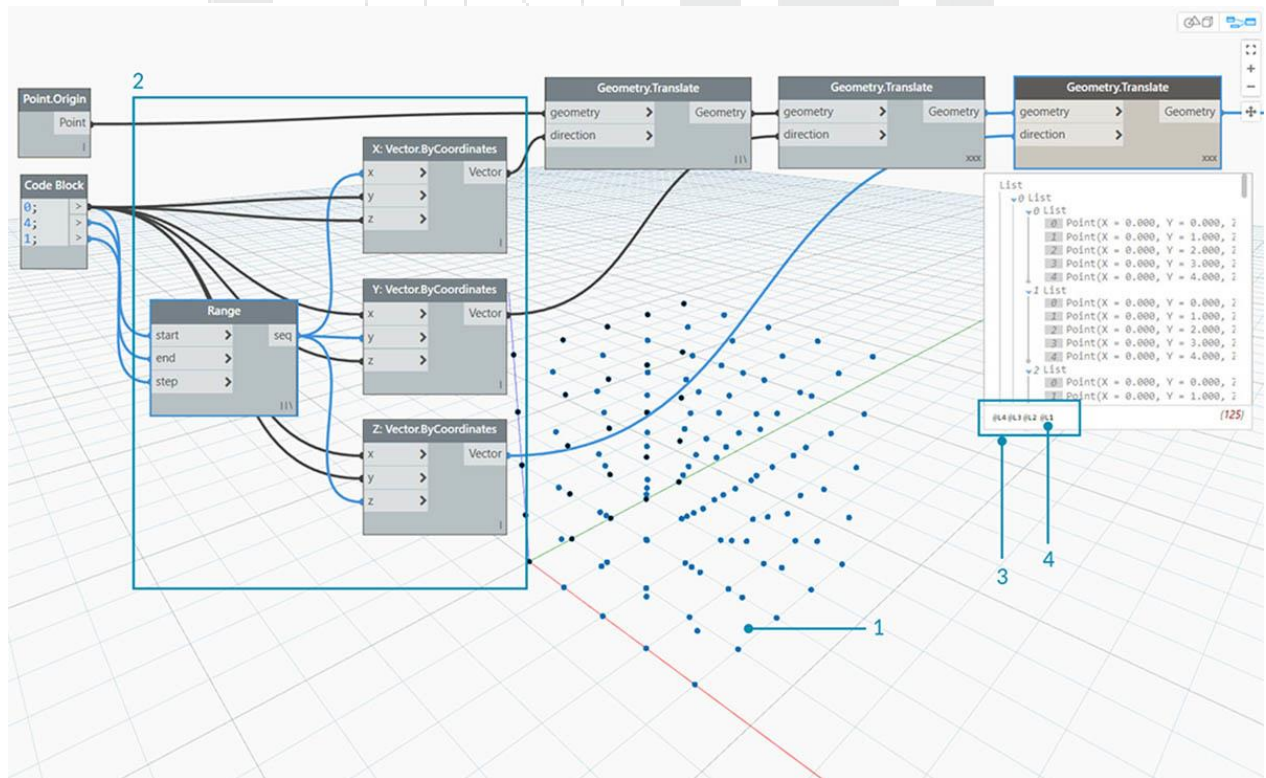
Una alternativa a *List.Map*, la característica *List @ Level* le permite seleccionar directamente el nivel de lista con el que desea trabajar directamente en el puerto de entrada del nodo. Esta característica se puede aplicar a cualquier entrada de un nodo y le permitirá acceder a los niveles

de sus listas de manera más rápida y sencilla que otros métodos. Simplemente dígame al nodo qué nivel de la lista desea usar como entrada y deje que el nodo haga el resto.

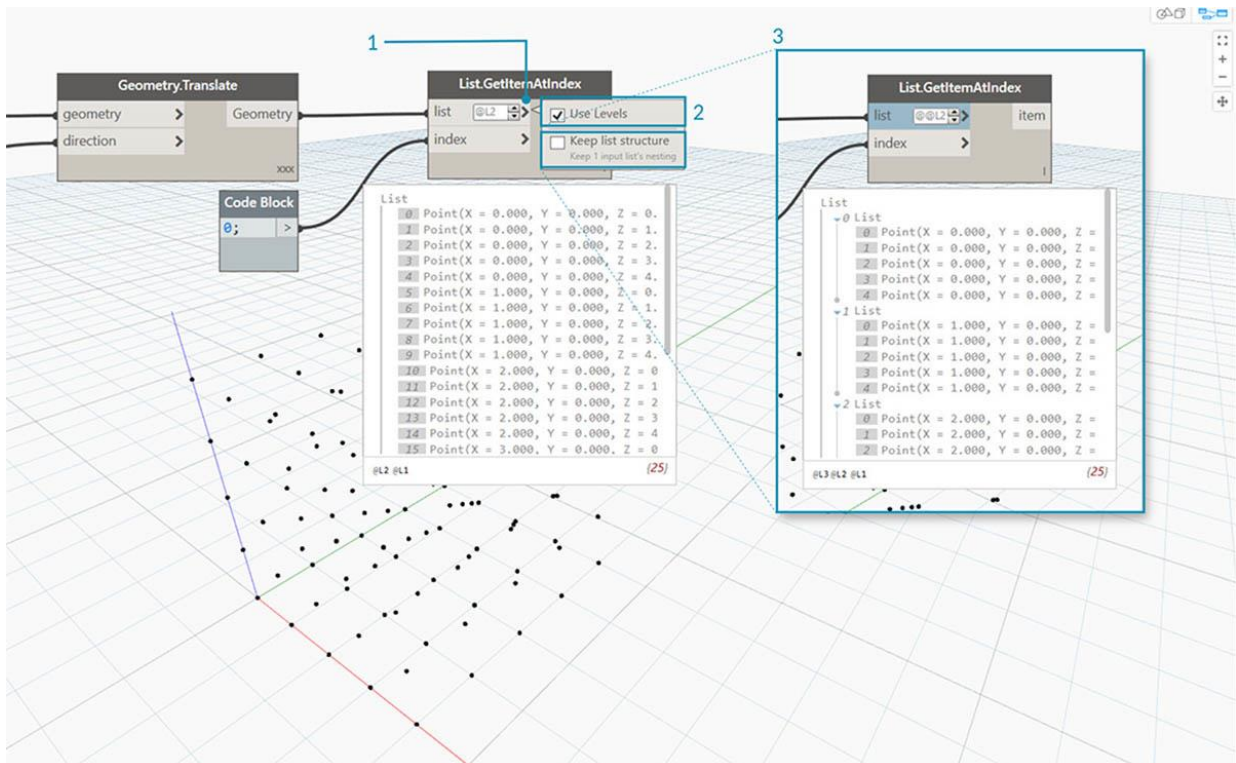
Ejercicio List @ Level

En este ejercicio, usaremos la función List @ Level para aislar un nivel específico de datos.

Descargue el archivo de ejemplo que acompaña a este ejercicio **List @ Level**. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.



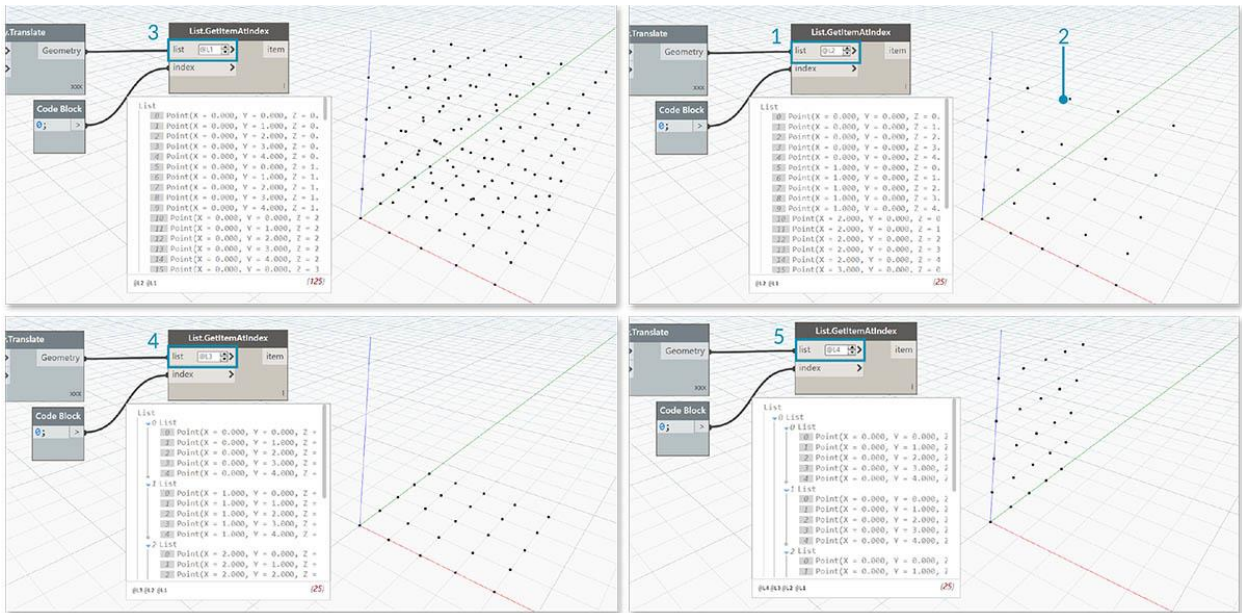
1. Comenzaremos con una simple cuadrícula 3D de puntos.
2. Como la cuadrícula está construida con un rango para X, Y y Z, sabemos que los datos están estructurados con 3 niveles: una lista X, una lista Y y una lista Z.
3. Estos niveles existen en diferentes **niveles**. Los niveles se indican en la parte inferior de la burbuja de vista previa. Las columnas de Niveles de lista corresponden a los datos de lista anteriores para ayudar a identificar en qué nivel trabajar.
4. Los Niveles de lista están organizados en orden inverso para que los datos de nivel más bajo siempre estén en "L1". Esto ayudará a garantizar que sus gráficos funcionen según lo planificado, incluso si algo se cambia en sentido ascendente.



1. Para usar la función List @ Level, haga clic en '>'. Dentro de este menú, verá dos casillas de verificación.
2. **Niveles de uso** : esto habilita la funcionalidad List @ Level. Después de hacer clic en esta opción, podrá hacer clic y seleccionar los niveles de lista de entrada que desea que use el nodo. Con este menú, puede probar rápidamente diferentes opciones de nivel haciendo clic arriba o abajo.
3. **Mantener la estructura de la lista**: si está habilitada, tendrá la opción de mantener la estructura de niveles de esa entrada. A veces, puede haber organizado deliberadamente sus datos en sublistas. Al marcar esta opción, puede mantener intacta su organización de lista y no perder ninguna información.

Con nuestra cuadrícula 3D simple, podemos acceder y visualizar la estructura de la lista al alternar entre los niveles de la lista. Cada combinación de nivel de lista e índice devolverá un conjunto diferente de puntos de nuestro conjunto 3D original.

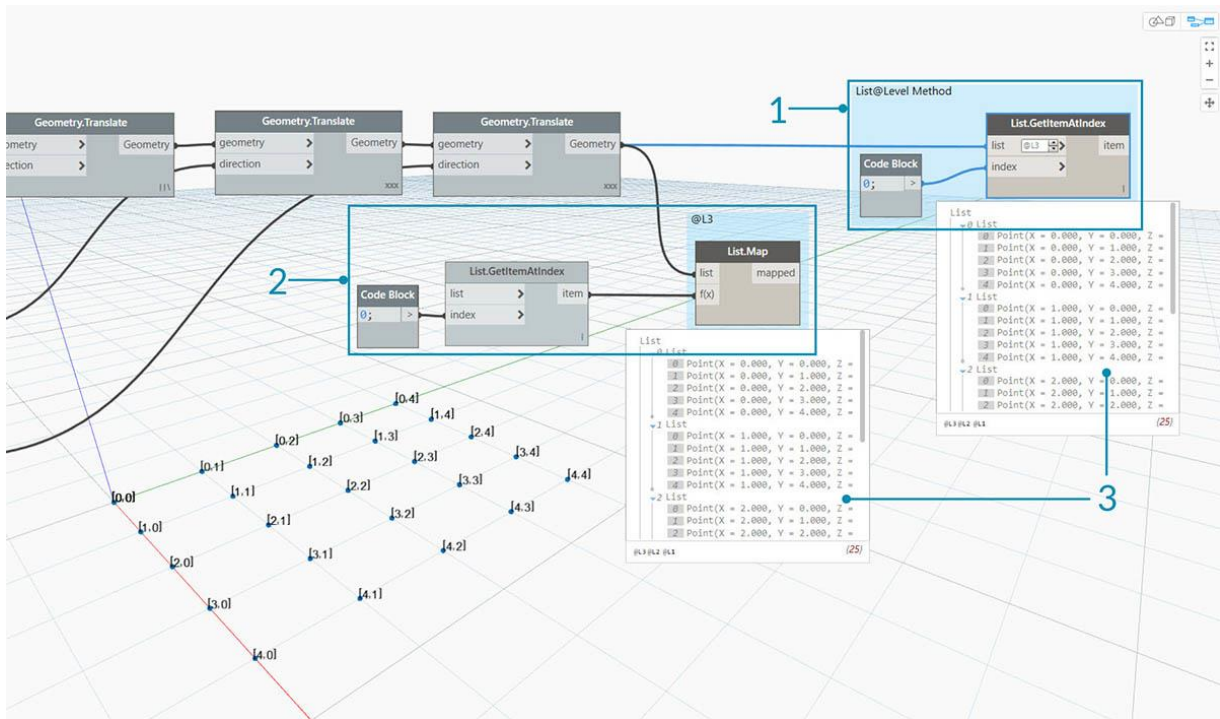
DARCO
DESDE 1988



1. 0 incluye solo el primer conjunto de puntos X, devolviendo solo una cuadrícula YZ. "@ L2" en DesignScript nos permite seleccionar solo la Lista en el Nivel 2.
2. La Lista en el Nivel 2 con el índice 0 incluye solo el primer conjunto de puntos Y, devolviendo solo la cuadrícula XZ.
3. Si cambiamos el filtro de nivel a "L1", podremos ver todo en el primer nivel de lista. La Lista en el Nivel 1 con el índice 0 incluye todos nuestros puntos 3D en una lista plana.
4. Si intentamos lo mismo para "L3" veremos solo los terceros puntos de Nivel de lista. La Lista en el Nivel 3 con el índice 0 incluye solo el primer conjunto de puntos Z, devolviendo solo una cuadrícula XY.
5. Si intentamos lo mismo con "L4", solo veremos los terceros puntos de Nivel de lista. La Lista en el Nivel 4 con el índice

Aunque este ejemplo particular también se puede crear con List.Map, List @ Level simplifica enormemente la interacción, facilitando el acceso a los datos del nodo. Observe a continuación una comparación entre los métodos List.Map y List @ Level:

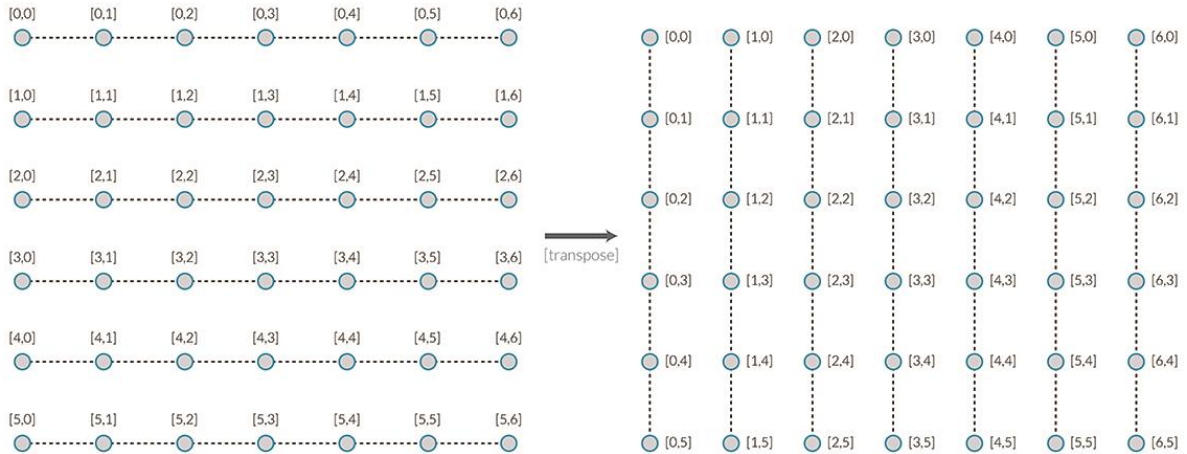
DARCO
DESDE 1988



1. Aunque ambos métodos nos darán acceso a los mismos puntos, el método List @ Level nos permite alternar fácilmente entre capas de datos dentro de un solo nodo.
2. Para acceder a una cuadrícula de puntos con List.Map, necesitaremos un nodo List.GetItemAtIndex junto con List.Map. Para cada nivel de lista que estamos abandonando, necesitaremos usar un nodo List.Map adicional. Dependiendo de la complejidad de sus listas, esto podría requerir que agregue una cantidad significativa de nodos List.Map a su gráfico para acceder al nivel correcto de información.
3. En este ejemplo, un nodo List.GetItemAtIndex con un nodo List.Map reanuda el mismo conjunto de puntos con la misma estructura de lista que el List.GetItemAtIndex con '@ L3' seleccionado.

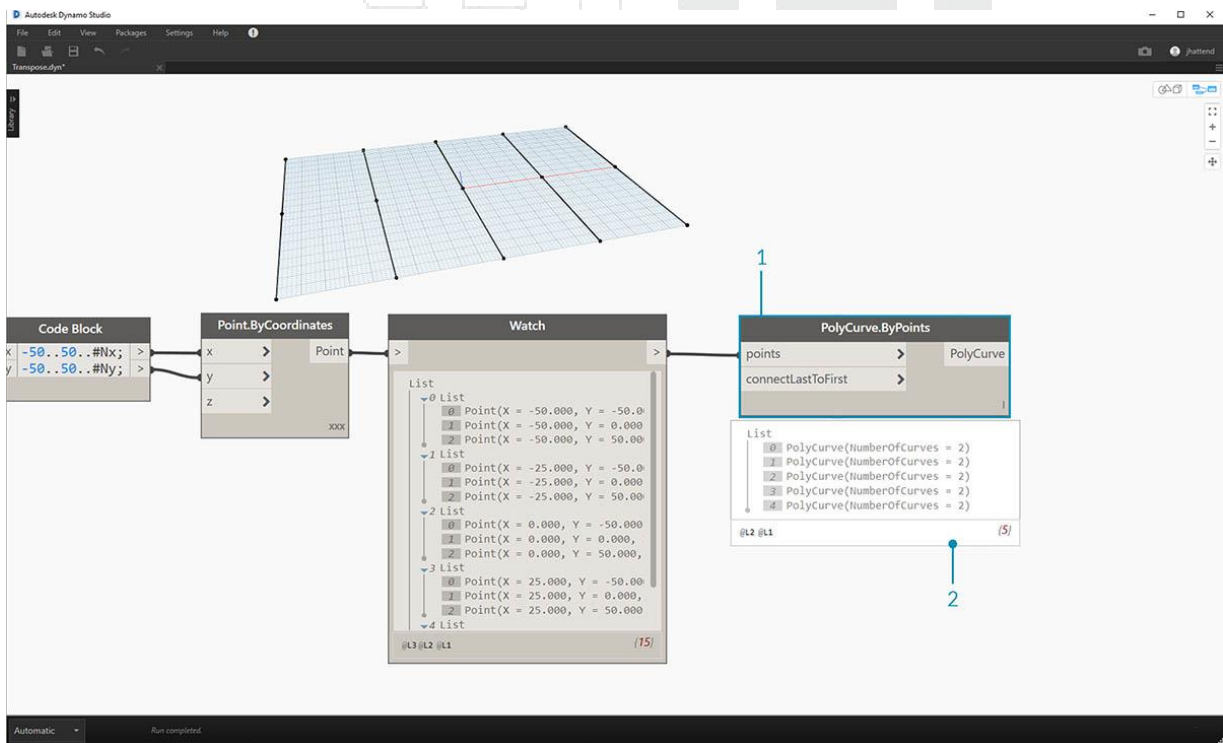
Transponer

Transpose es una función fundamental cuando se trata de listas de listas. Al igual que en los programas de hojas de cálculo, una transposición voltea las columnas y las filas de una estructura de datos. Lo demostraremos con una matriz básica a continuación, y en la siguiente sección, demostraremos cómo se puede usar una transposición para crear relaciones geométricas.



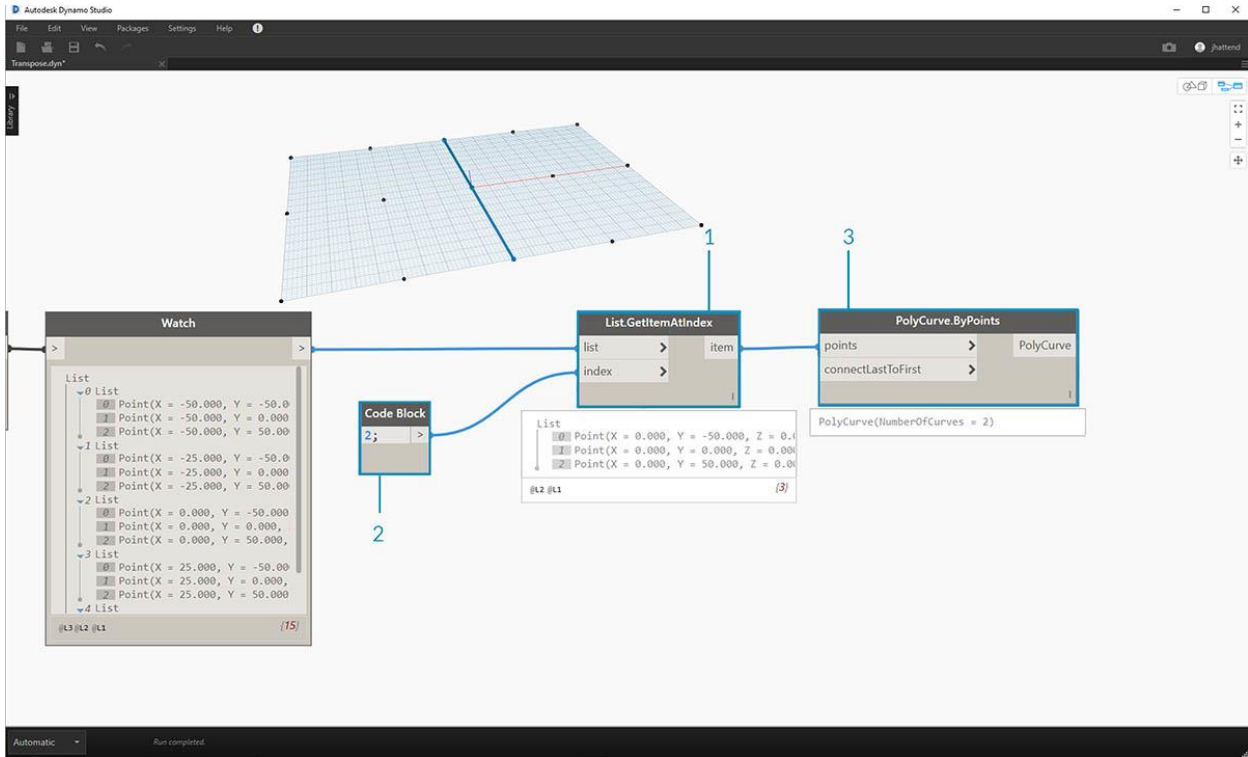
Ejercicio - List.Transpose

Descargue el archivo de ejemplo que acompaña a este ejercicio Transpose.dyn. Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

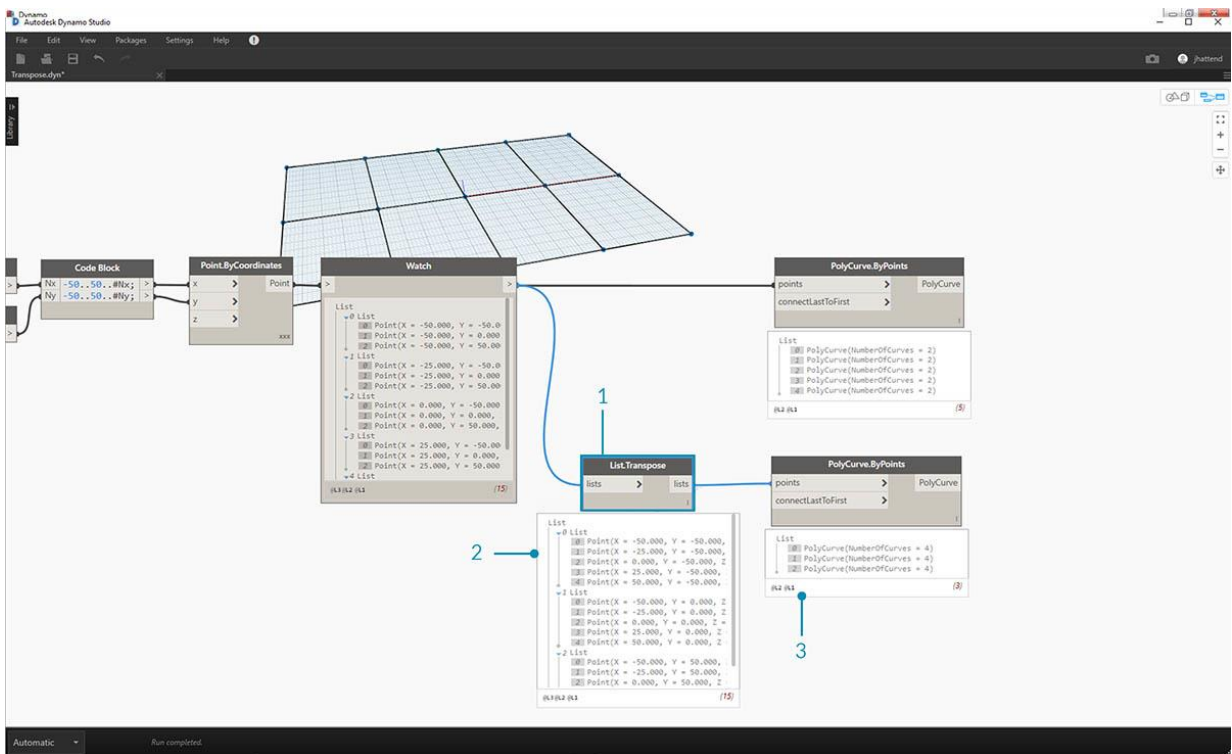


Vamos a eliminar los *List.Count* nodos del ejercicio anterior y pasar a algo de geometría para ver cómo estructuran los datos.

1. Conecte un *PolyCurve.ByPoints* a la salida del nodo watch desde *Point.ByCoordinates*.
2. El resultado muestra 5 polycurvas, y podemos ver las curvas en nuestra vista previa de Dynamo. El nodo Dynamo busca una lista de puntos (o una lista de listas de puntos en este caso) y crea una única policurva a partir de ellos. Esencialmente, cada lista se ha convertido a una curva en la estructura de datos.



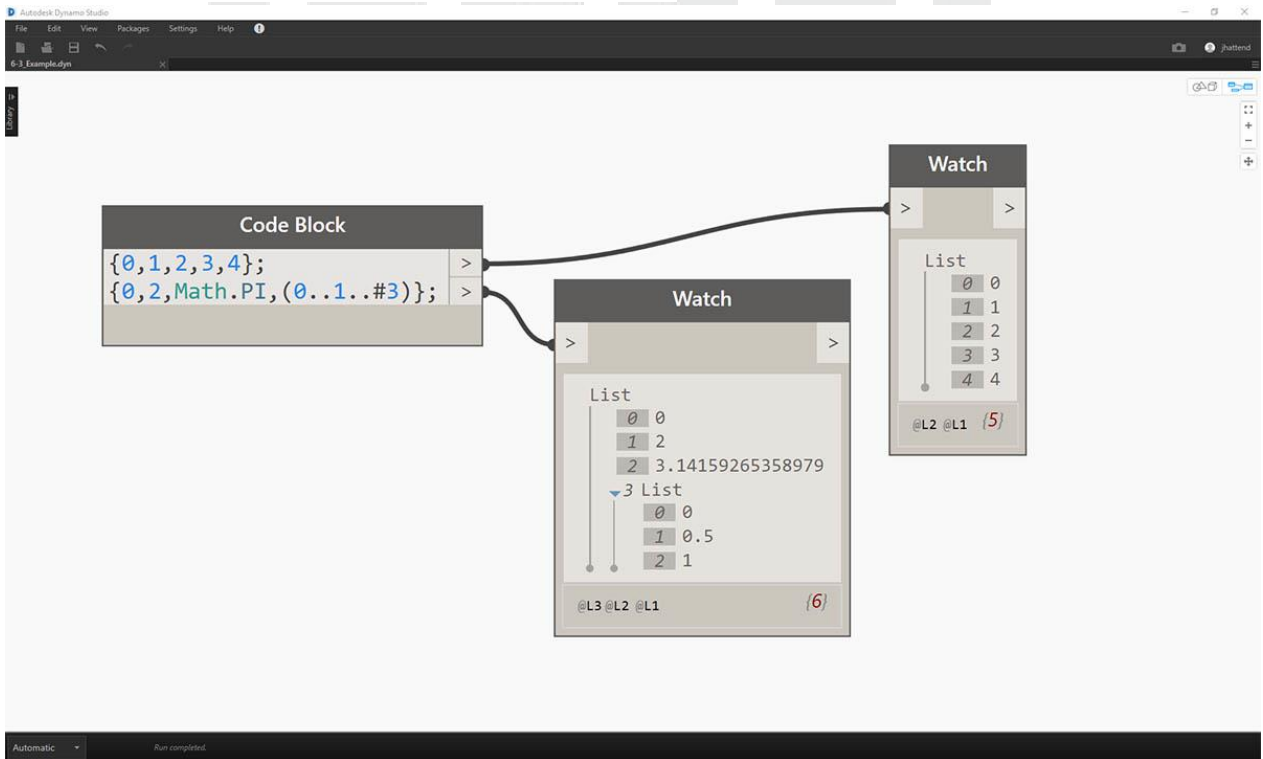
1. Si queremos aislar una fila de curvas, usamos el nodo *List.GetItemAtIndex*.
2. Usando un valor de *bloque de código* de 2, consulta el tercer elemento en la lista principal.
3. El *PolyCurve.ByPoints* nos da una curva, ya que solo una lista está conectada al nodo.



1. Un nodo *List.Transpose* cambiará todos los elementos con todas las listas en una lista de listas. Esto suena complicado, pero es la misma lógica que transponer en Microsoft Excel: cambiar columnas con filas en una estructura de datos.
2. Observe el resultado abstracto: la transposición cambió la estructura de la lista de 5 listas con 3 elementos cada una a 3 listas con 5 elementos cada una.
3. Observe el resultado geométrico: usando *PolyCurve.ByPoints* , obtenemos 3 polycurves en la dirección perpendicular a las curvas originales.

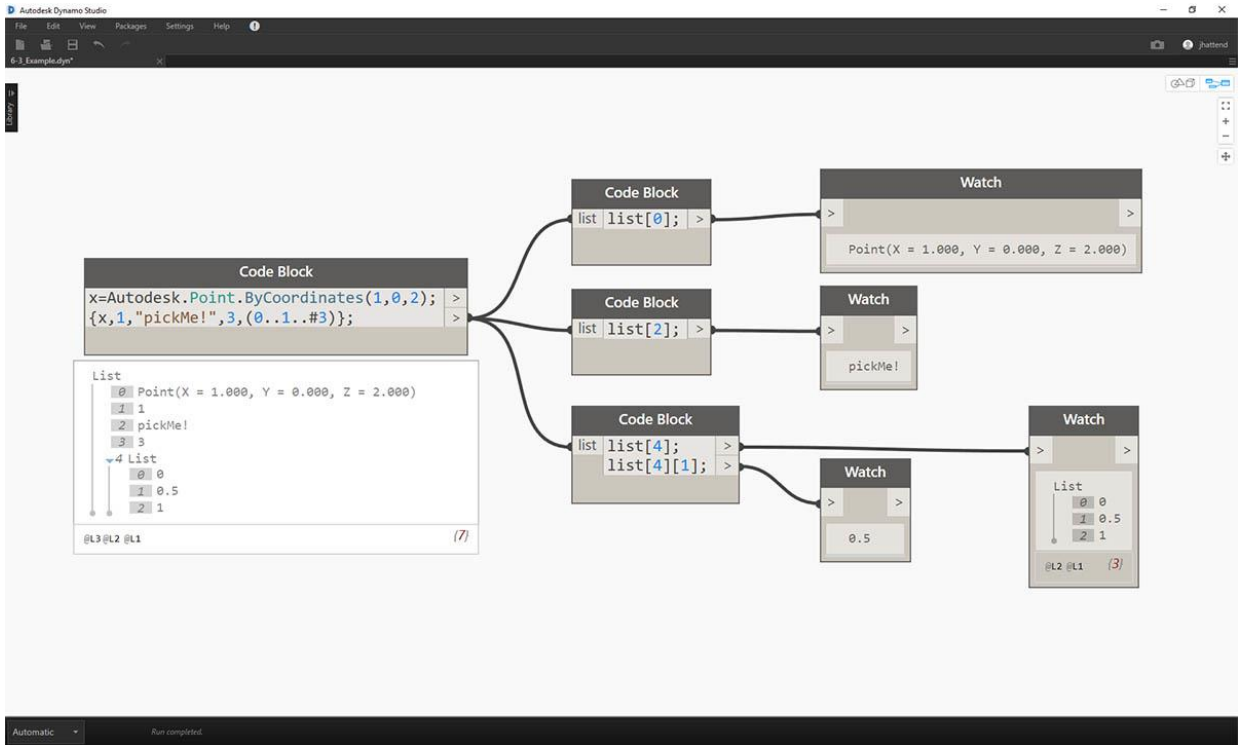
Code Block Creation

La abreviatura de bloque de código usa "{}" para definir una lista. Esta es una forma mucho más rápida y fluida de crear listas que el nodo List.Create. Haga referencia a la siguiente imagen para observar cómo se puede definir una lista con múltiples expresiones con un bloque de código.



Consulta de bloque de código

La taquigrafía de bloque de código utiliza "["] como una manera rápida y fácil de seleccionar elementos específicos que desee de una estructura de datos compleja. Consulte la imagen a continuación para observar cómo se puede consultar una lista con múltiples tipos de datos con un bloque de código.

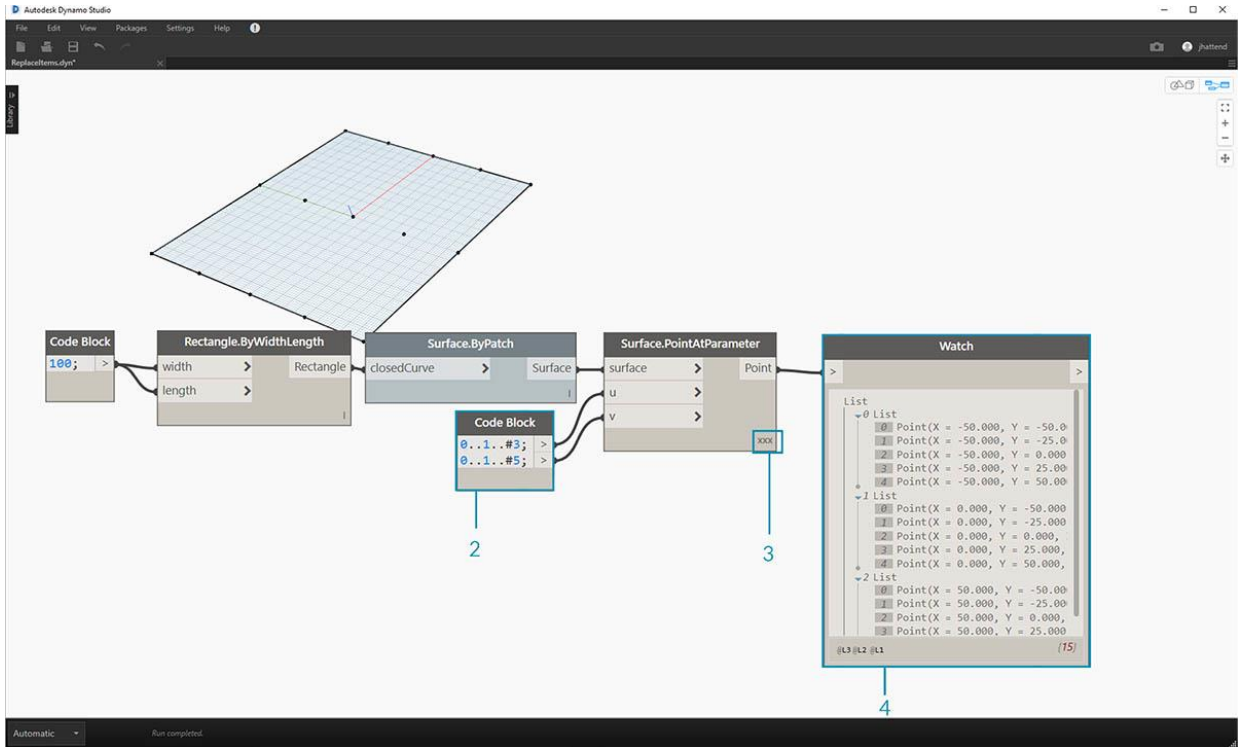


Ejercicio: consultar e insertar datos

Descargue el archivo de ejemplo que acompaña a este ejercicio Replaceltems.dyn . Se puede encontrar una lista completa de archivos de ejemplo en el Dataset.

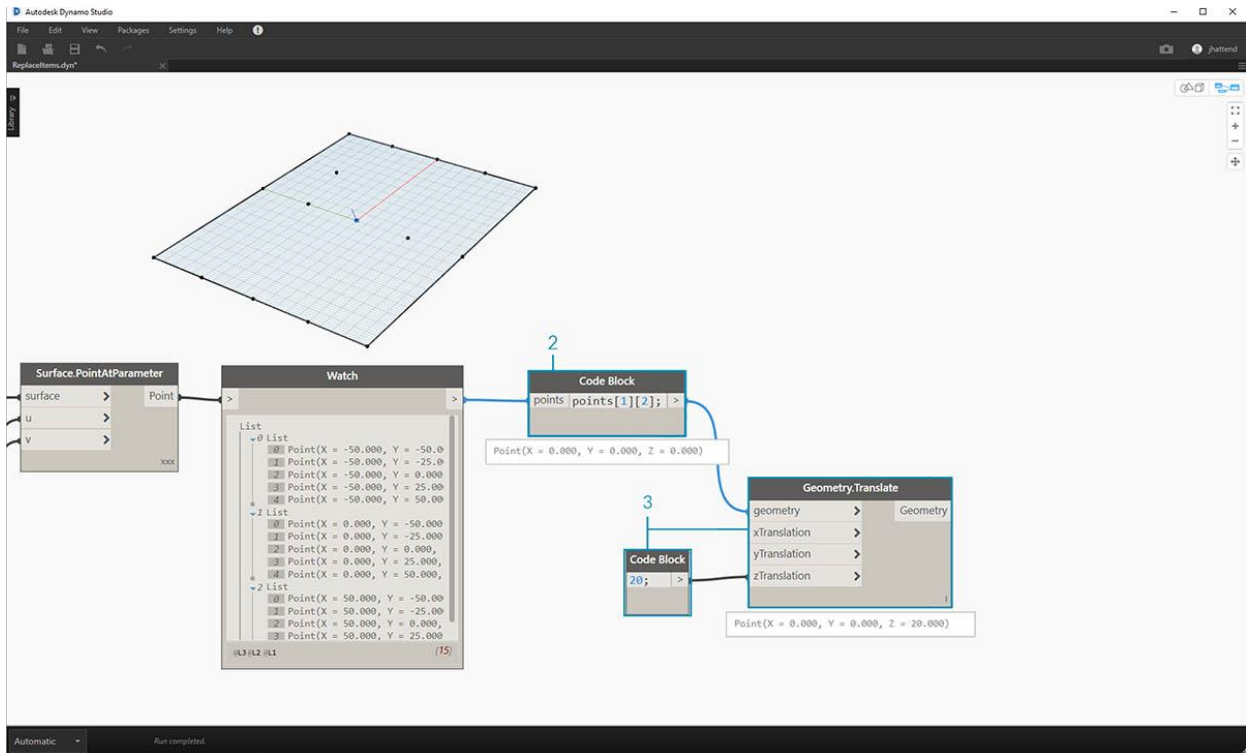
Este ejercicio usa parte de la lógica establecida en el anterior para editar una superficie. Nuestro objetivo aquí es intuitivo, pero la navegación de la estructura de datos será más complicada. Queremos articular una superficie moviendo un punto de control.





1. Comience con la cadena de nodos arriba. Estamos creando una superficie básica que abarca la cuadrícula de Dynamo por defecto.
2. Usando el *bloque de código*, inserte estas dos líneas de código y conéctese a las entradas *u* y *v* de *Surface.PointAtParameter*, respectivamente:
3. -50..50..#3;
4. -50..50..#5;
5. Asegúrese de establecer el *Cordón* de *Surface.PointAtParameter* como "*Producto Cruzado*".
6. El nodo *Watch* muestra que tenemos una lista de 3 listas, cada una con 5 elementos.

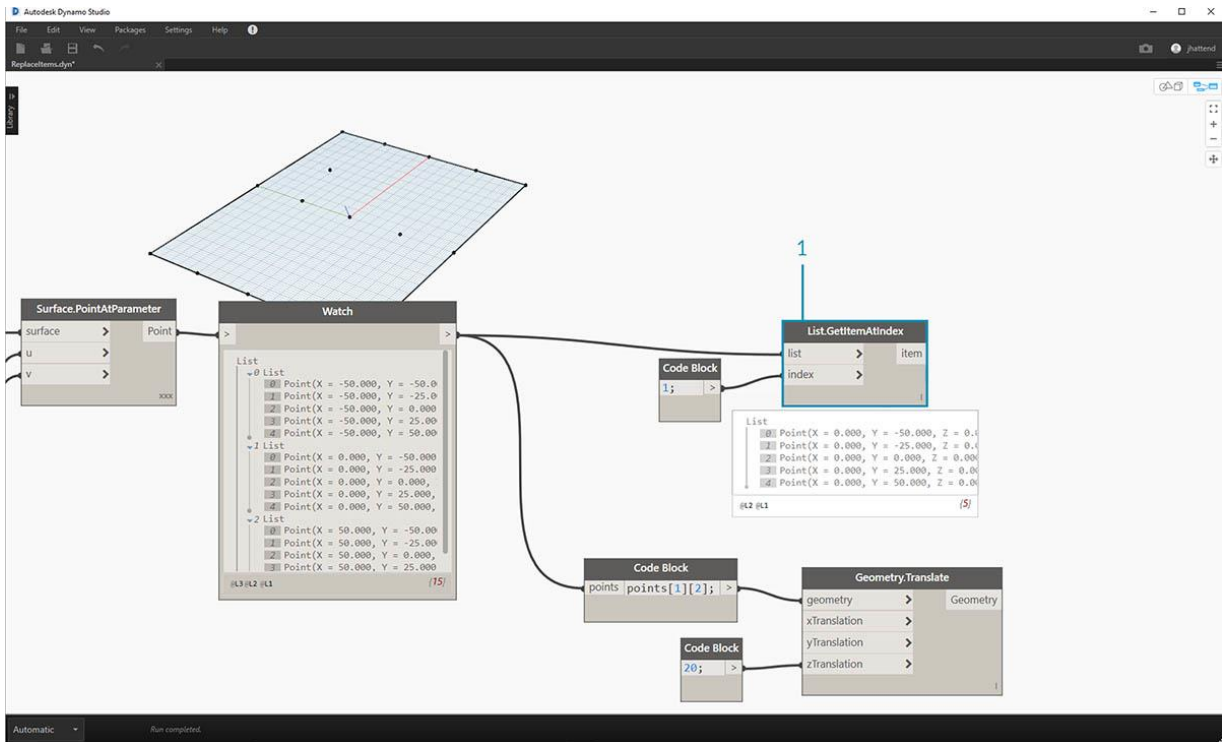
DARCO
DESDE 1988



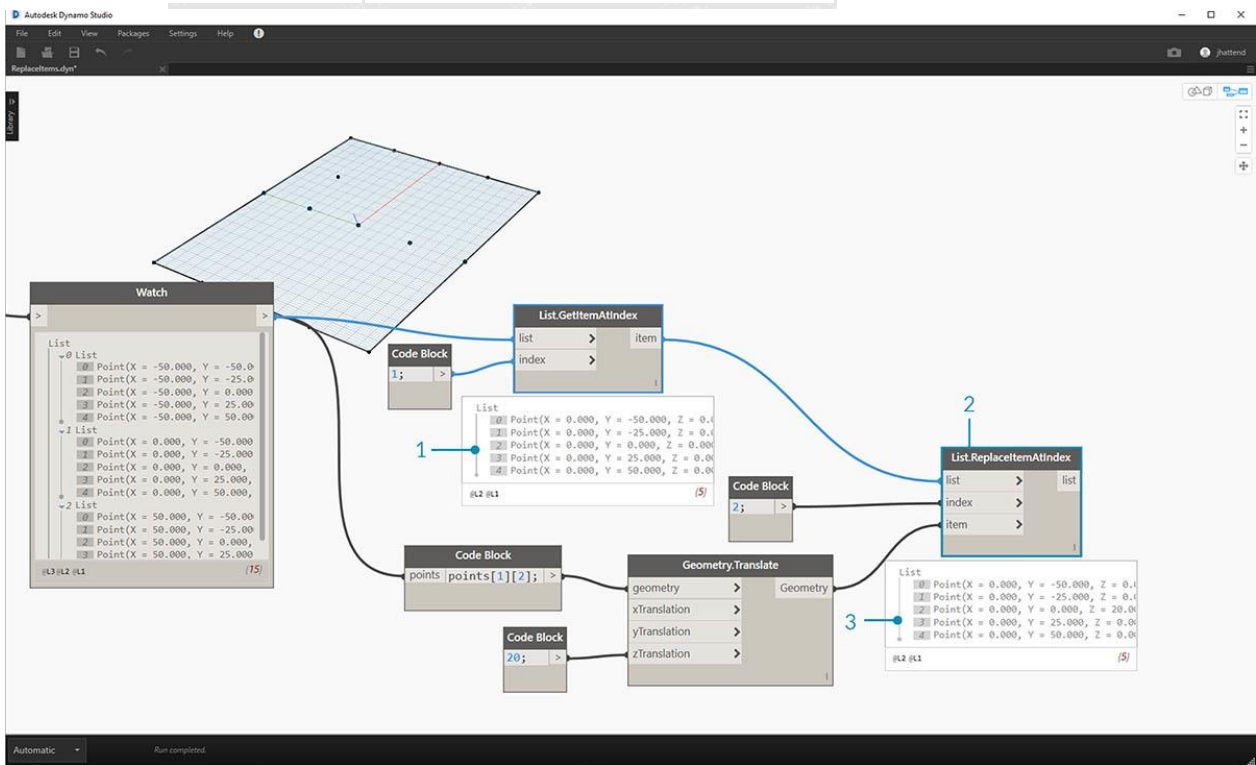
En este paso, queremos consultar el punto central en la cuadrícula que hemos creado. Para hacer esto seleccionaremos el punto medio en la lista del medio. Tiene sentido, ¿verdad?

1. Para confirmar que este es el punto correcto, también podemos hacer clic en los elementos del nodo de observación para confirmar que estamos apuntando a la correcta.
2. Usando el *bloque de código*, escribiremos una línea básica de código para consultar una lista de listas:
`points[1][2];`
3. Usando *Geometry.Translate*, moveremos el punto seleccionado hacia arriba en la dirección Z en 20 unidades.

DARCO
DESDE 1988

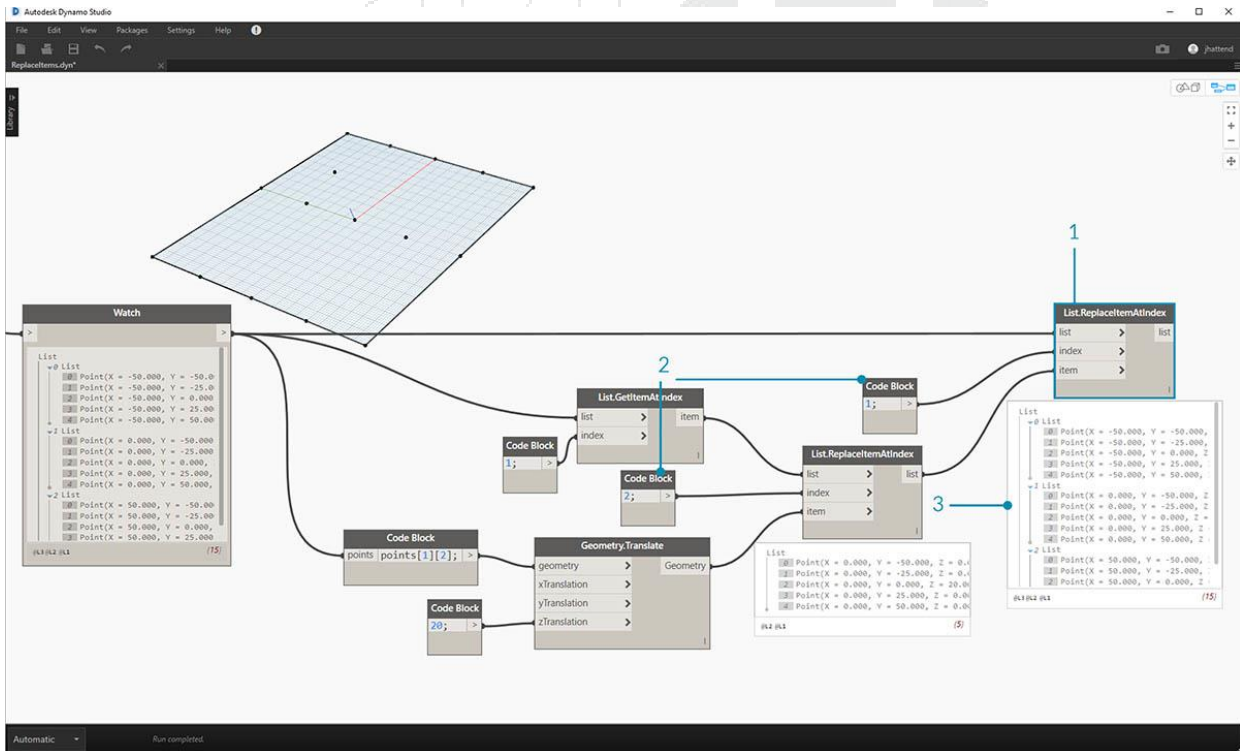


1. Seleccionemos también la fila del medio de puntos con un nodo *List.GetItemAtIndex*. Nota: Similar a un paso anterior, también podemos consultar la lista con el *bloque de código*, usando una línea de `points[1]`;



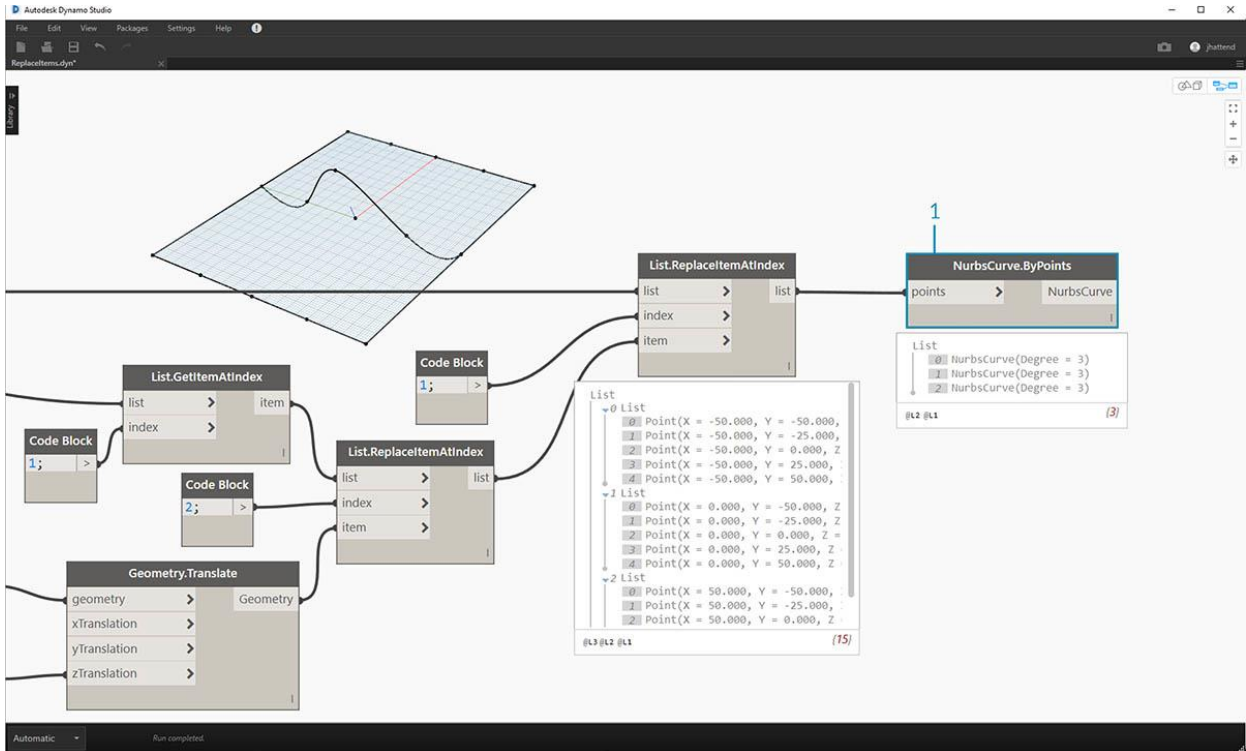
Hasta ahora hemos consultado con éxito el punto central y lo hemos movido hacia arriba. Ahora queremos insertar este punto movido en la estructura de datos original.

1. Primero, queremos reemplazar el elemento de la lista que hemos aislado en un paso anterior.
2. Utilizando *List.ReplaceltemAtIndex*, reemplazaremos el elemento del medio usando el índice de "2", con el elemento de reemplazo conectado al punto movido (*Geometry.Translate*).
3. El resultado muestra que hemos ingresado el punto movido en el elemento medio de la lista.



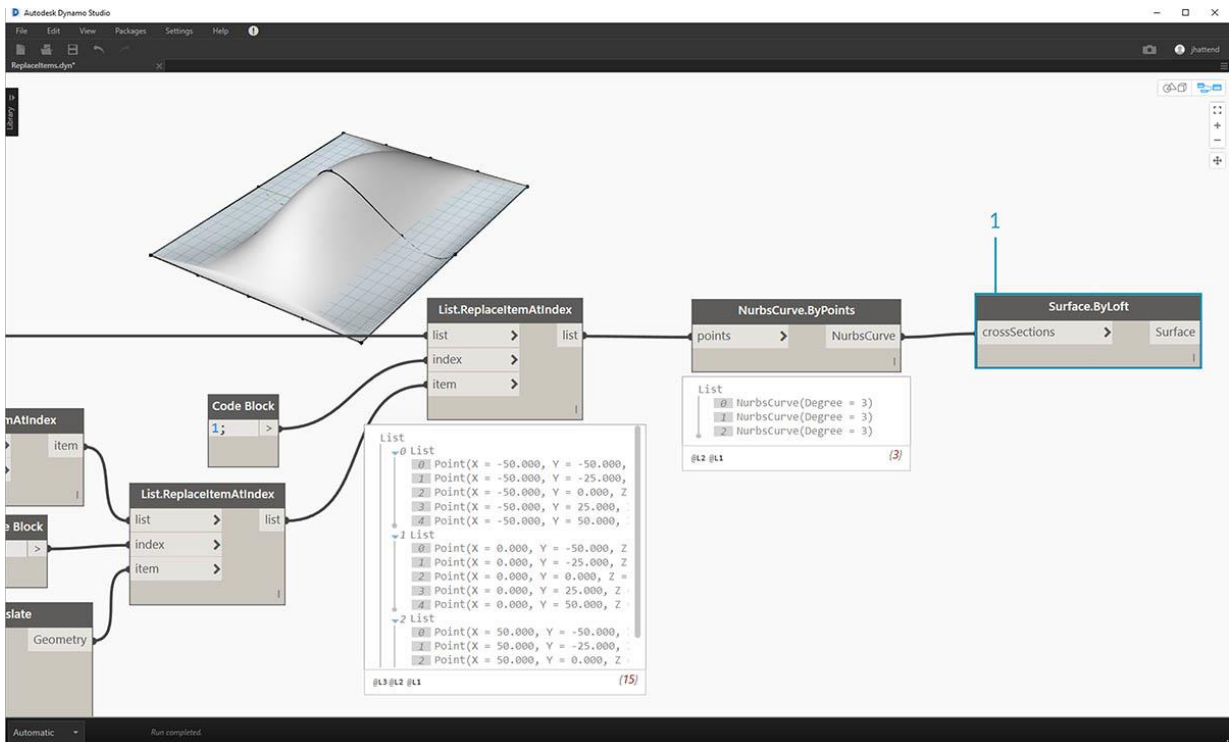
Ahora que hemos modificado la lista, tenemos que volver a insertar esta lista en la estructura de datos original: la lista de listas.

1. Siguiendo la misma lógica, use *List.ReplaceltemAtIndex* para reemplazar la lista del medio con nuestra lista modificada.
2. Observe que los *bloques de código* que definen el índice para estos dos nodos son 1 y 2, que coincide con la consulta original del *bloque de código* (`puntos [1] [2]`).
3. Al seleccionar la lista en el *índice 1*, vemos la estructura de datos resaltada en la vista previa de Dynamo. Combinamos con éxito el punto movido en la estructura de datos original.



Hay muchas maneras de hacer una superficie a partir de este conjunto de puntos. En este caso, vamos a crear una superficie por lofting curvas juntas.

1. Cree un nodo *NurbsCurve.ByPoints* y conecte la nueva estructura de datos para crear tres curvas nurbs.



1. Conecte un *Surface.ByLoft* a la salida de *NurbsCurve.ByPoints*. Ahora tenemos una superficie modificada. Podemos cambiar el valor Z original de Geometría. ¡Traduzca y mire la actualización de geometría!

Listas n-dimensionales

Más abajo en el agujero de conejo, agreguemos aún más niveles a la jerarquía. La estructura de datos puede expandirse mucho más allá de una lista bidimensional de listas. Como las listas son elementos en sí mismos en Dynamo, podemos crear datos con tantas dimensiones como sea posible.

La analogía con la que trabajaremos aquí es Russian Nesting Dolls. Cada lista se puede considerar como un contenedor que contiene varios elementos. Cada lista tiene sus propias propiedades y también se considera como su propio objeto.



Un conjunto de muñecas de anidación rusas (Foto de Zeta) es una analogía para listas n-dimensionales. Cada capa representa una lista, y cada lista contiene elementos dentro de ella. En el caso de Dynamo, cada contenedor puede tener varios contenedores dentro (que representan los elementos de cada lista).

Las listas n-dimensionales son difíciles de explicar visualmente, pero hemos establecido algunos ejercicios en este capítulo que se enfocan en trabajar con listas que se aventuran más allá de dos dimensiones.

Mapeo y Combinaciones

El mapeo es posiblemente la parte más compleja de la administración de datos en Dynamo, y es especialmente relevante cuando se trabaja con jerarquías complejas de listas. Con la serie de ejercicios a continuación, demostraremos cuándo usar mapeo y combinaciones a medida que los datos se vuelven multidimensionales.

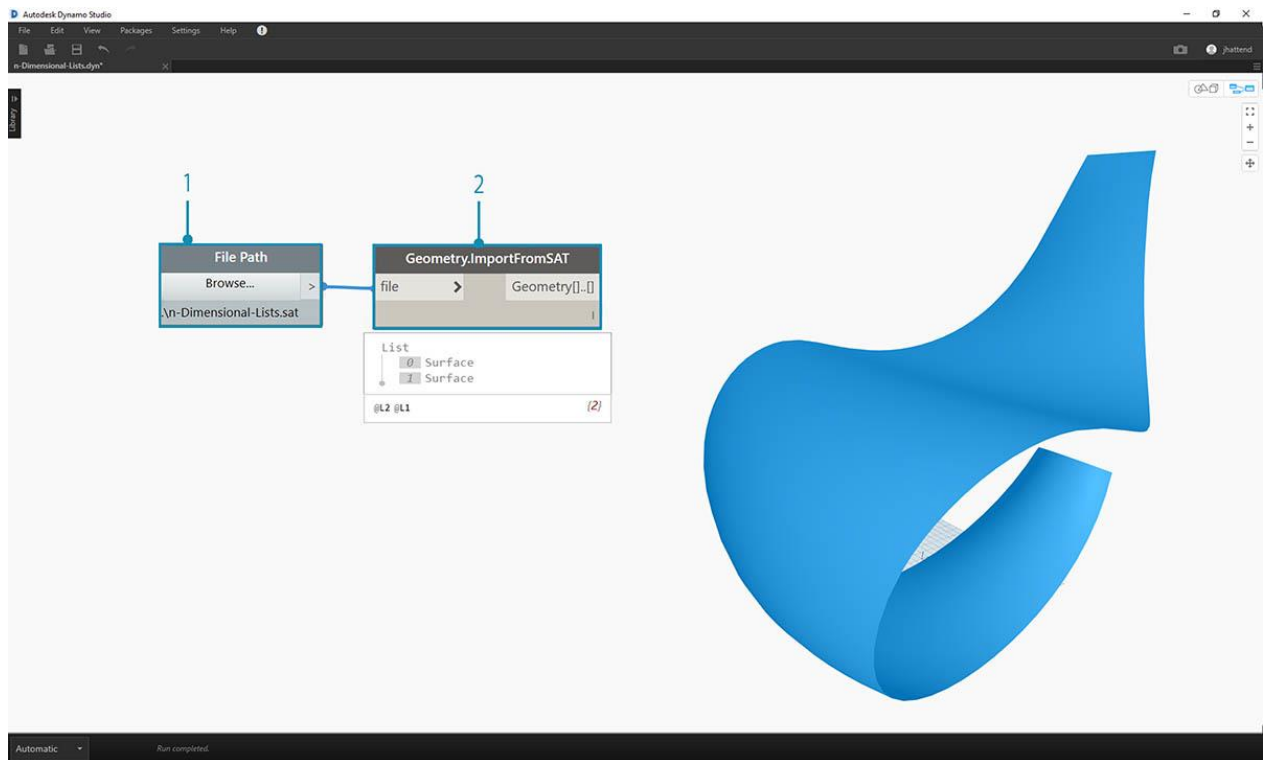
Las introducciones preliminares a List.Map y List.Combine se pueden encontrar en la sección anterior. En el último ejercicio a continuación, usaremos estos nodos en una estructura de datos compleja.

Ejercicio - Listas 2D – Básico

Descargue los archivos de ejemplo que acompañan a este ejercicio

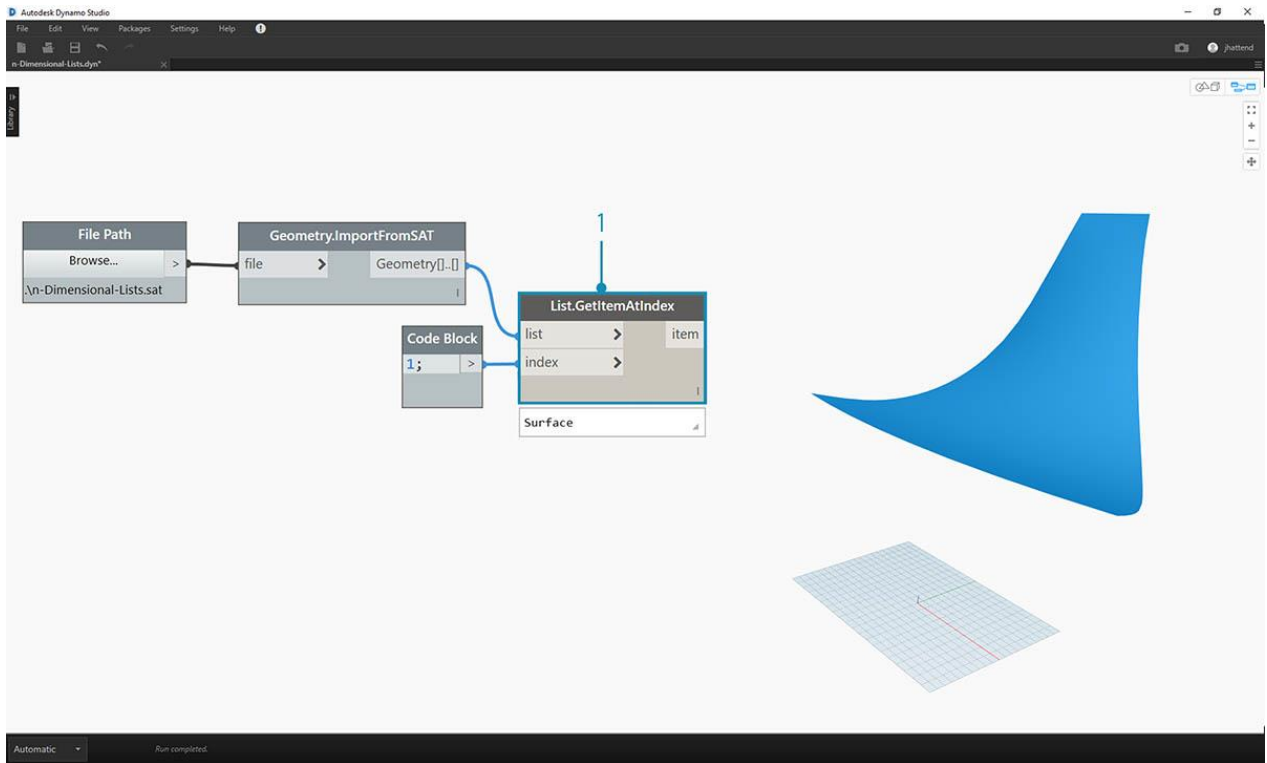
1. n-Dimensional-Lists.dyn
2. n-Dimensional-Lists.sat

Este ejercicio es el primero de una serie de tres que se centra en articular la geometría importada. Cada parte de esta serie de ejercicios aumentará en la complejidad de la estructura de datos.



1. Comencemos con el archivo .sat en la carpeta del archivo de ejercicios. Podemos tomar este archivo usando el nodo *Ruta de archivo*.

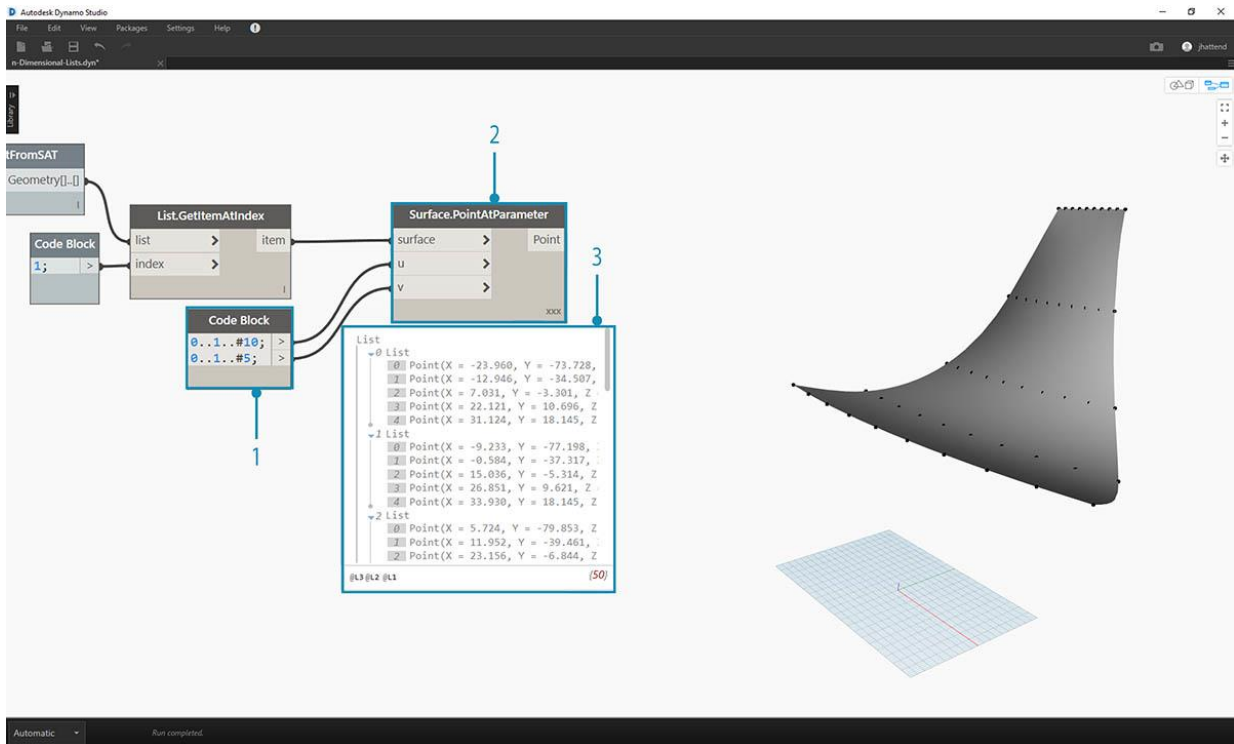
2. Con *Geometry.ImportFromSAT* , la geometría se importa en nuestra vista previa de Dynamo como dos superficies.



Para este ejercicio, queremos mantenerlo simple y trabajar con una de las superficies.

1. Seleccionemos el índice de 1 para agarrar la superficie superior. Hacemos esto con el nodo *List.GetItemAtIndex*.

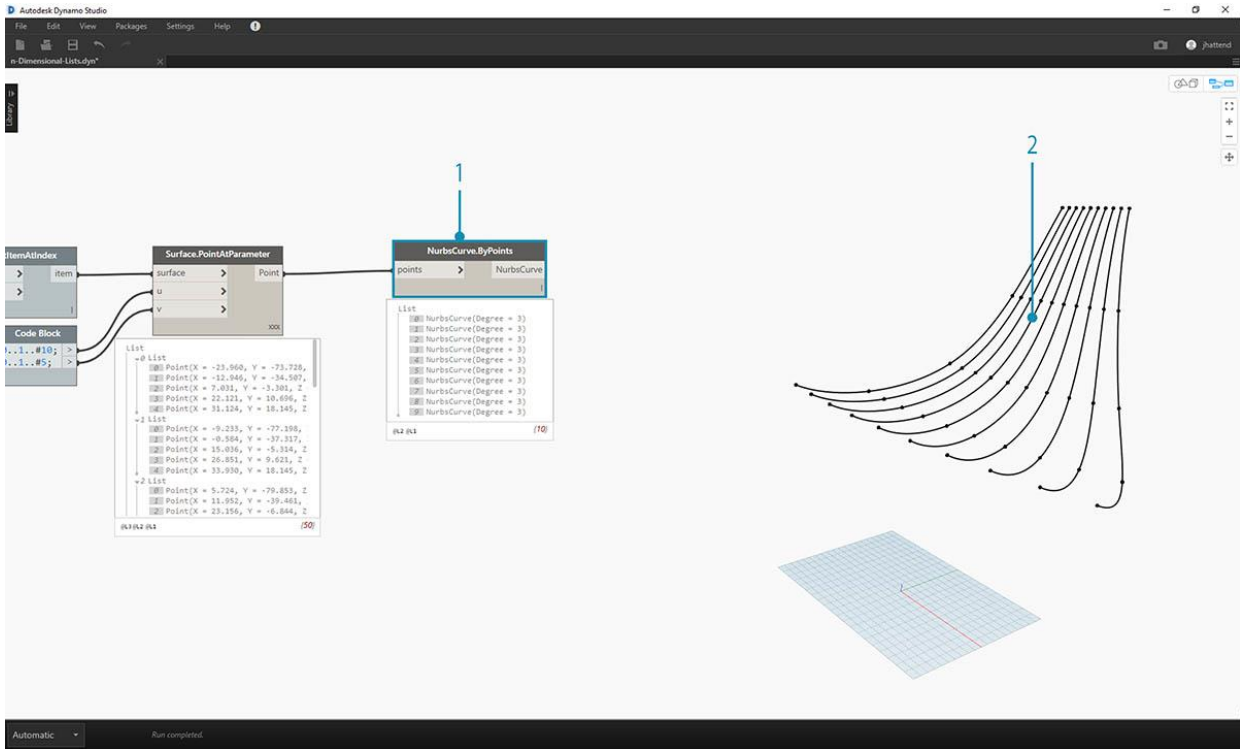
DARCO
DESDE 1988



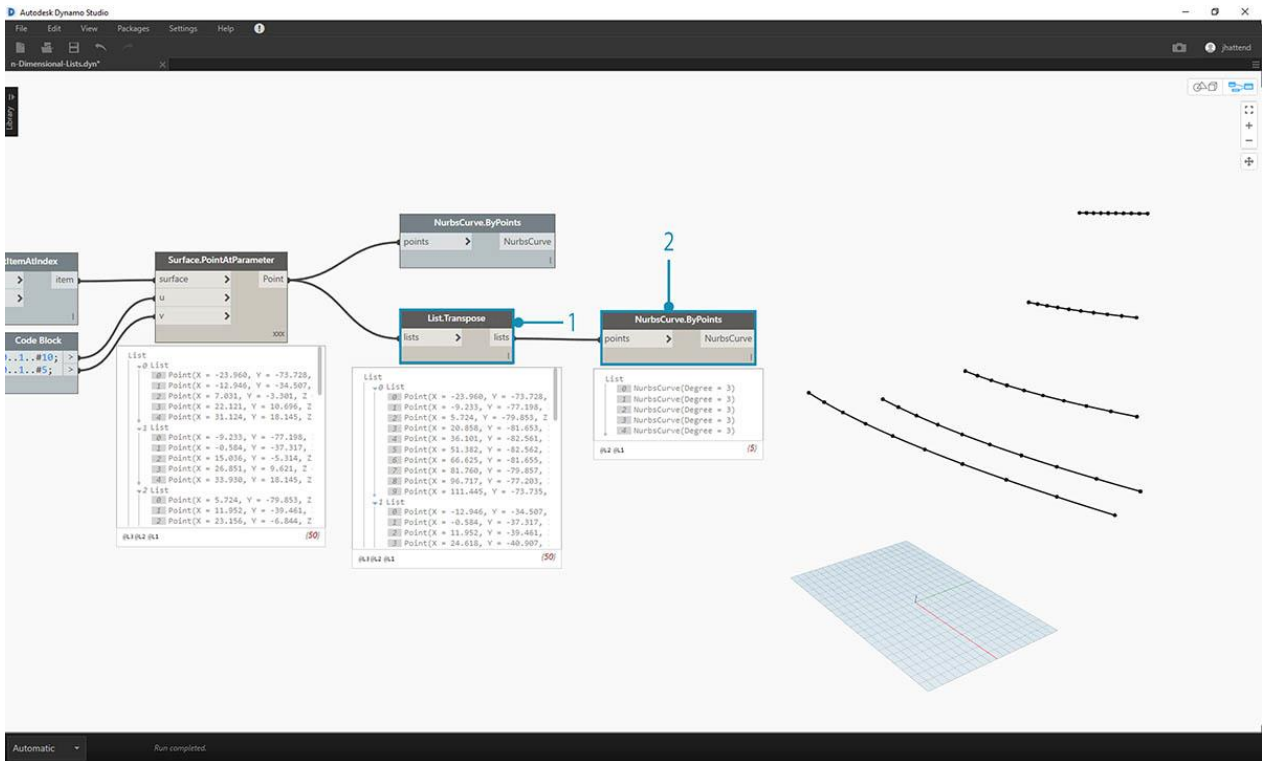
El siguiente paso es dividir la superficie en una grilla de puntos.

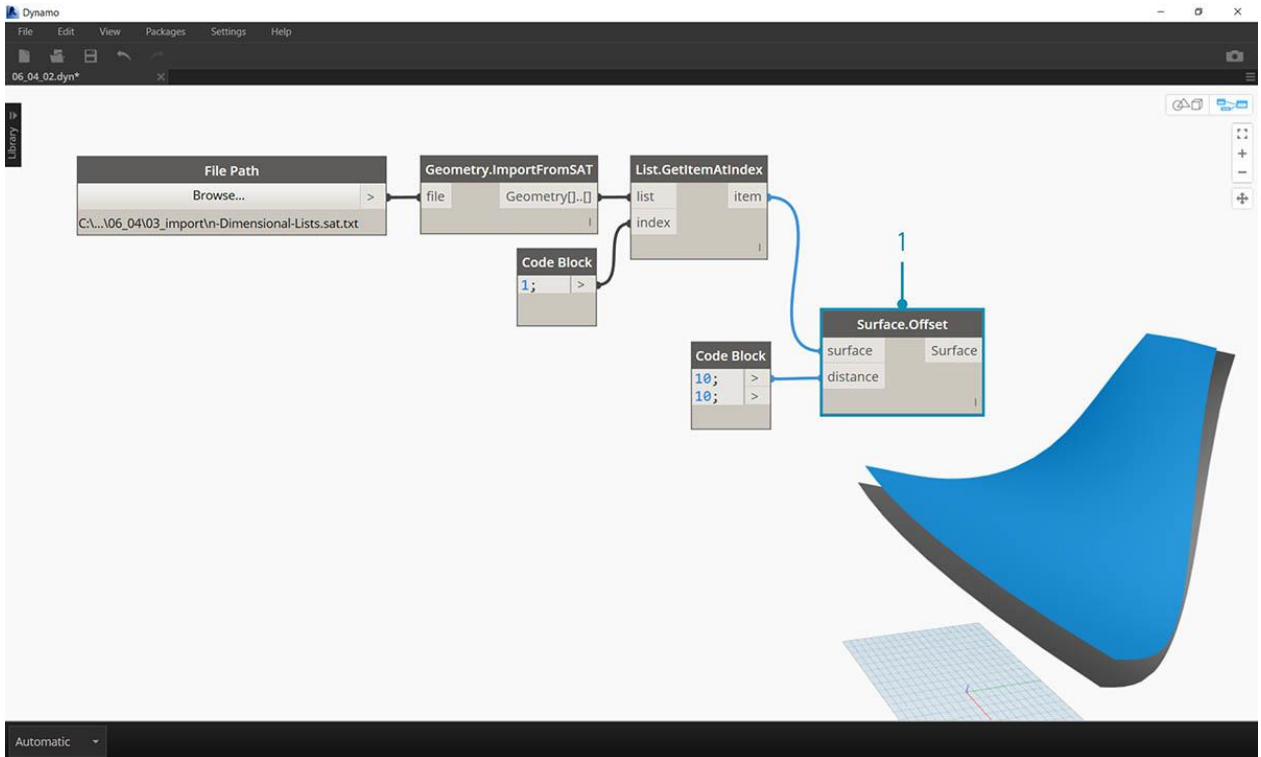
1. Usando el *bloque de código*, inserte estas dos líneas de código:
2. 0..1..#10;
3. 0..1..#5;
4. Con la *Surface.PointAtParameter*, conecte los dos valores de bloque de código de *u* y *v*. Cambie el *cordón* de este nodo a "*Producto Cruzado*".
5. El resultado revela la estructura de datos, que también es visible en la vista previa de Dynamo.

DARCO
DESDE 1988

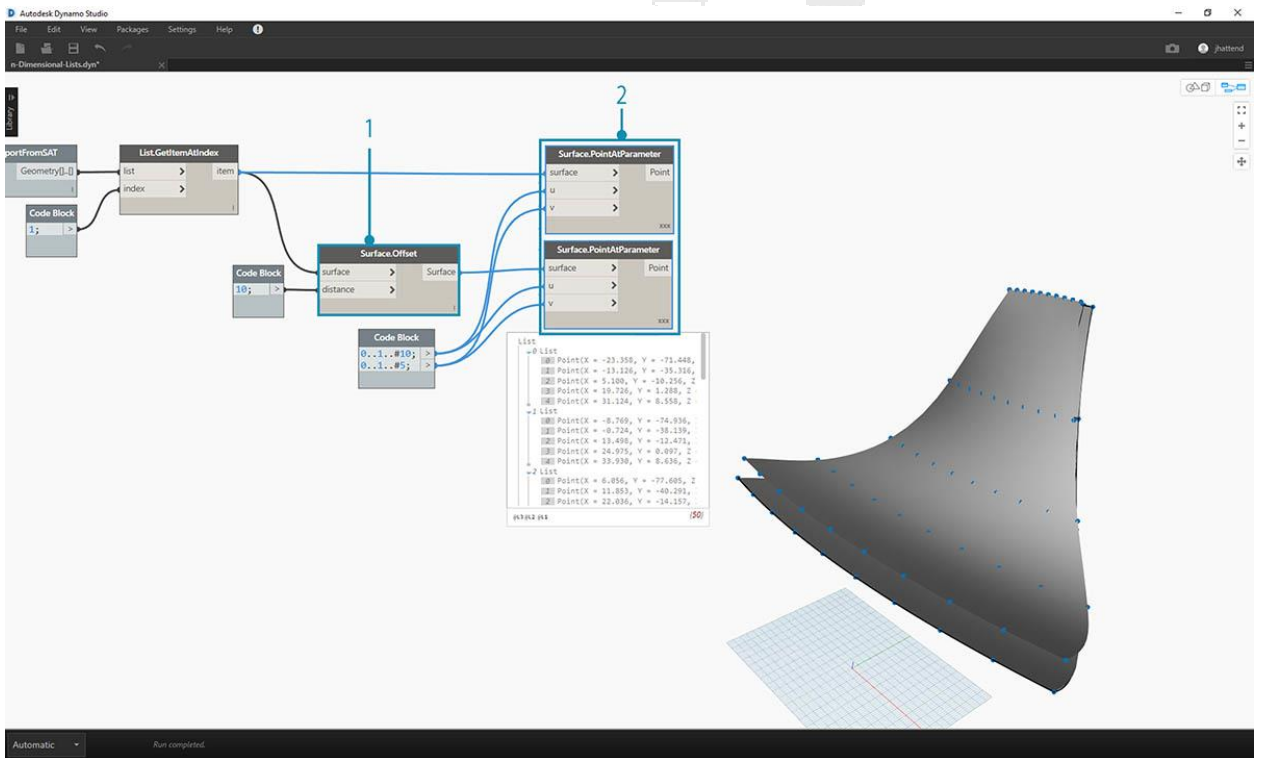


1. Para ver cómo está organizada la estructura de datos, conectemos un *NurbsCurve.ByPoints* a la salida de *Surface.PointAtParameter*.
2. Tenga en cuenta que tenemos diez curvas que se ejecutan verticalmente a lo largo de la superficie.



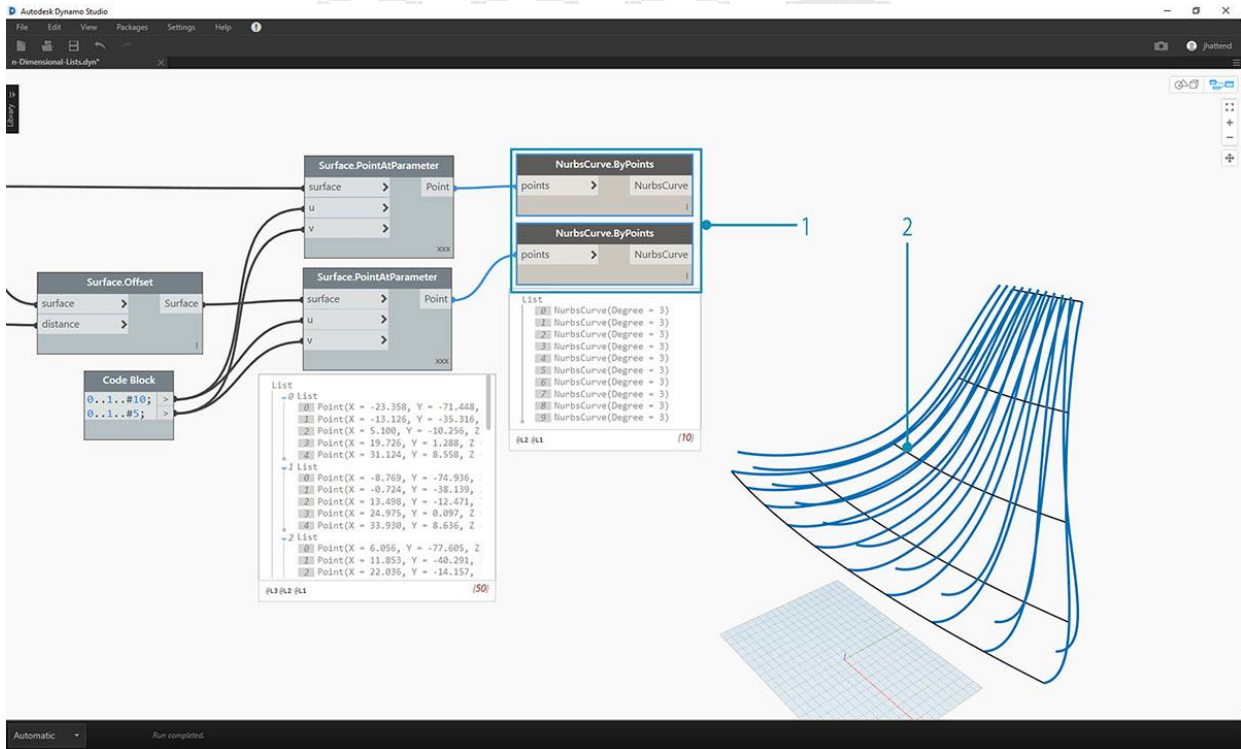


1. Usando *Surface.Offset*, compensa la superficie con un valor de 10.



1. De la misma manera que el ejercicio anterior, defina un *bloque de código* con estas dos líneas de código:

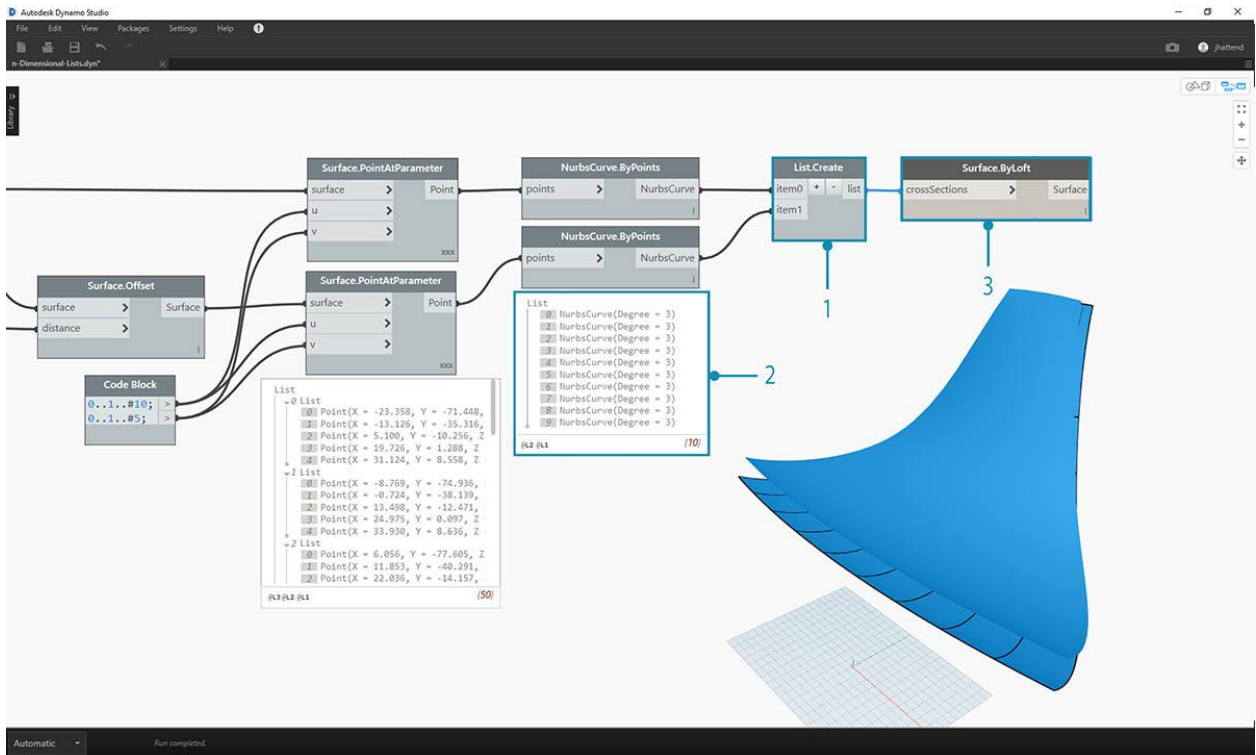
2. 0..1..#10;
3. 0..1..#5;
4. Conecte estas salidas a dos nodos *Surface.PointAtParameter*, cada uno con el conjunto de *lazosen "Cross Product"* . Uno de estos nodos está conectado a la superficie original, mientras que el otro está conectado a la superficie de compensación.



1. Como en el ejercicio anterior, conecte las salidas a dos nodos *NurbsCurve.ByPoints*.
2. Nuestra vista previa de Dynamo muestra dos curvas, correspondientes a dos superficies.

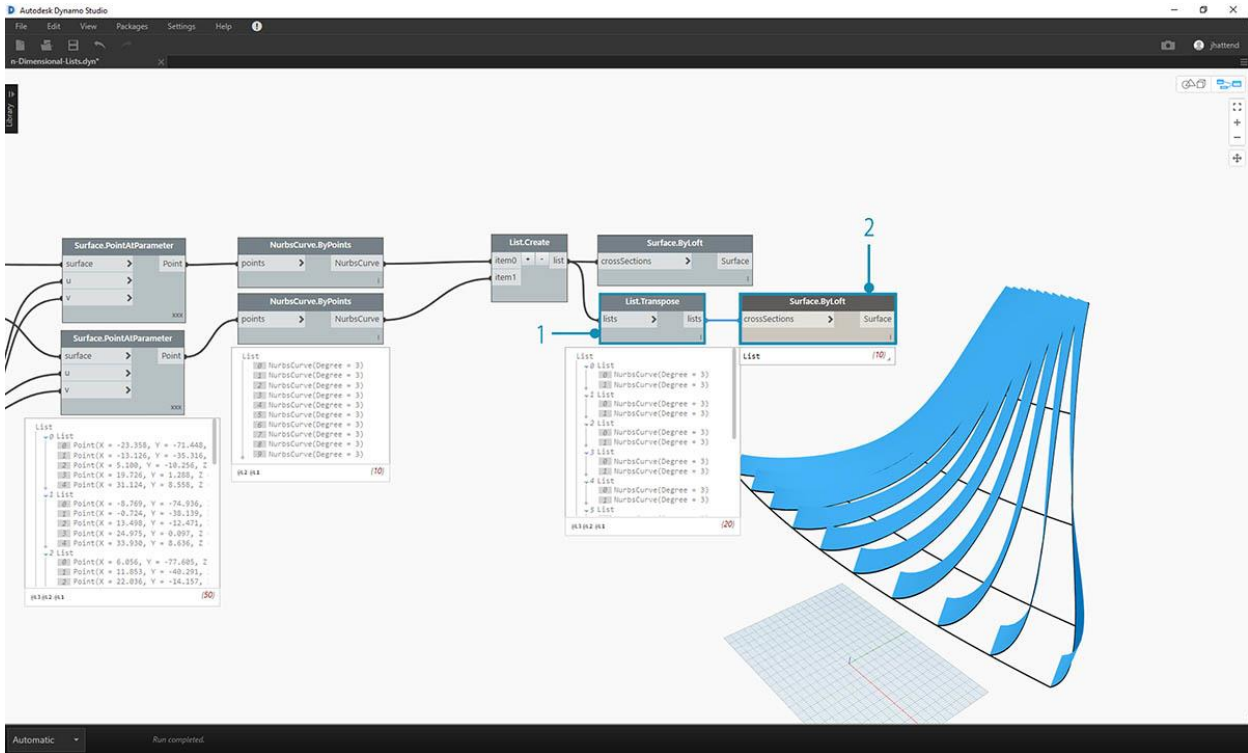
DARCO

DESDE 1988

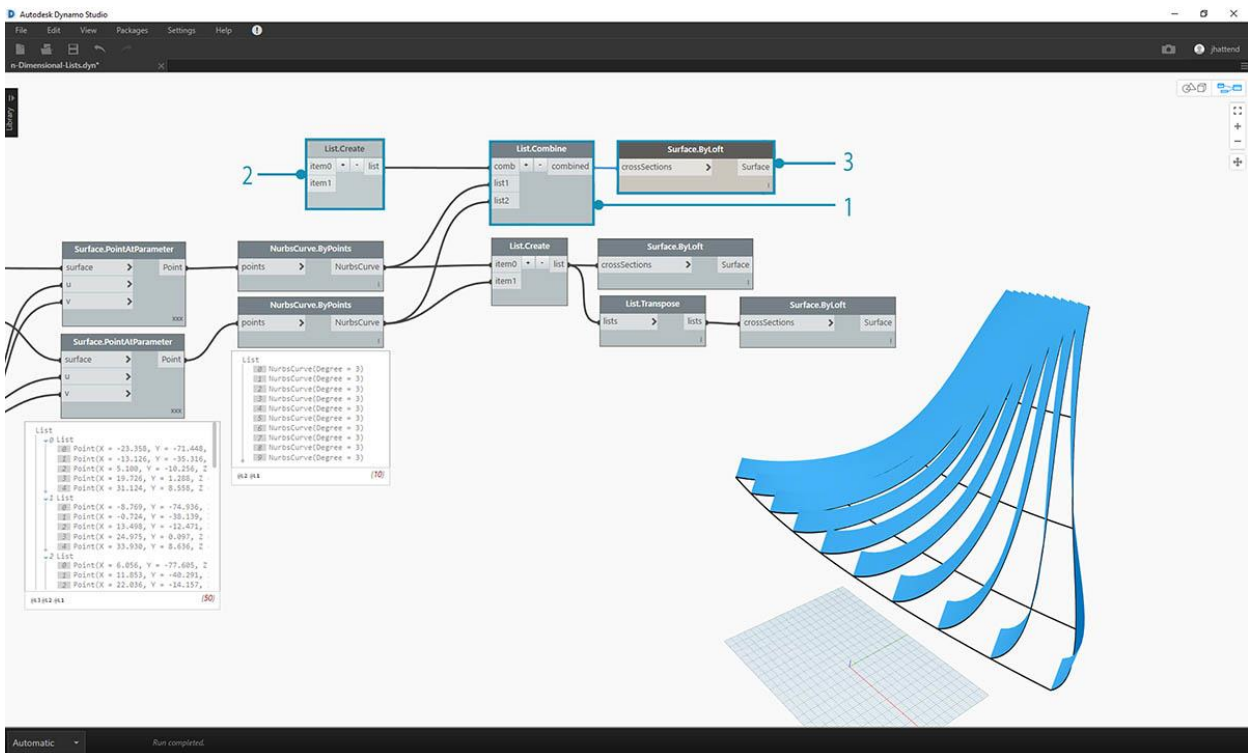


1. Al usar *List.Create*, podemos combinar los dos conjuntos de curvas en una lista de listas.
2. Fíjate en el resultado, tenemos dos listas con diez elementos cada una, que representan cada conjunto de conexiones de las curvas nurbs.
3. Al realizar un *Surface.ByLoft*, podemos dar sentido visual a esta estructura de datos. El nodo deshace todas las curvas en cada sublist.

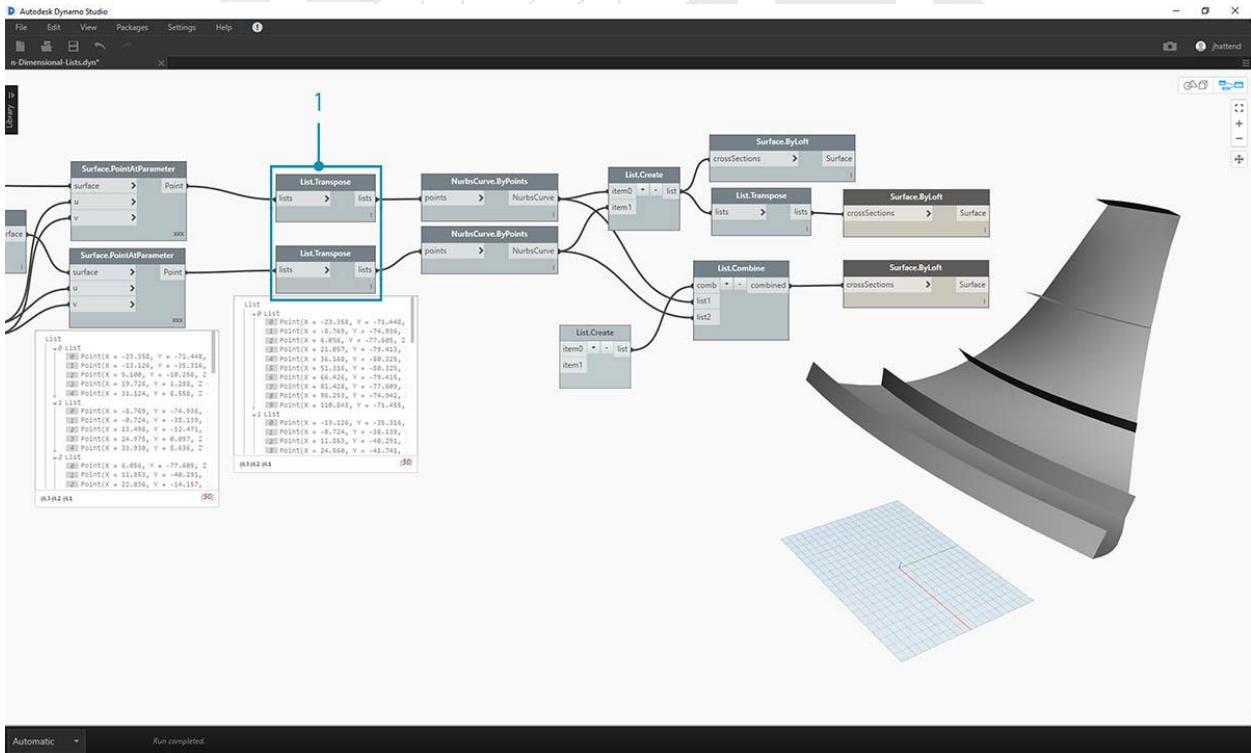
DARCO
DESDE 1988



1. Al usar *List.Transpose*, recuerde, estamos volteando todas las columnas y filas. Este nodo transferirá dos listas de diez curvas en diez listas de dos curvas. Ahora tenemos cada curva nurbs relacionada con la curva contigua en la otra superficie.
2. Usando *Surface.ByLoft*, llegamos a una estructura acanalada.



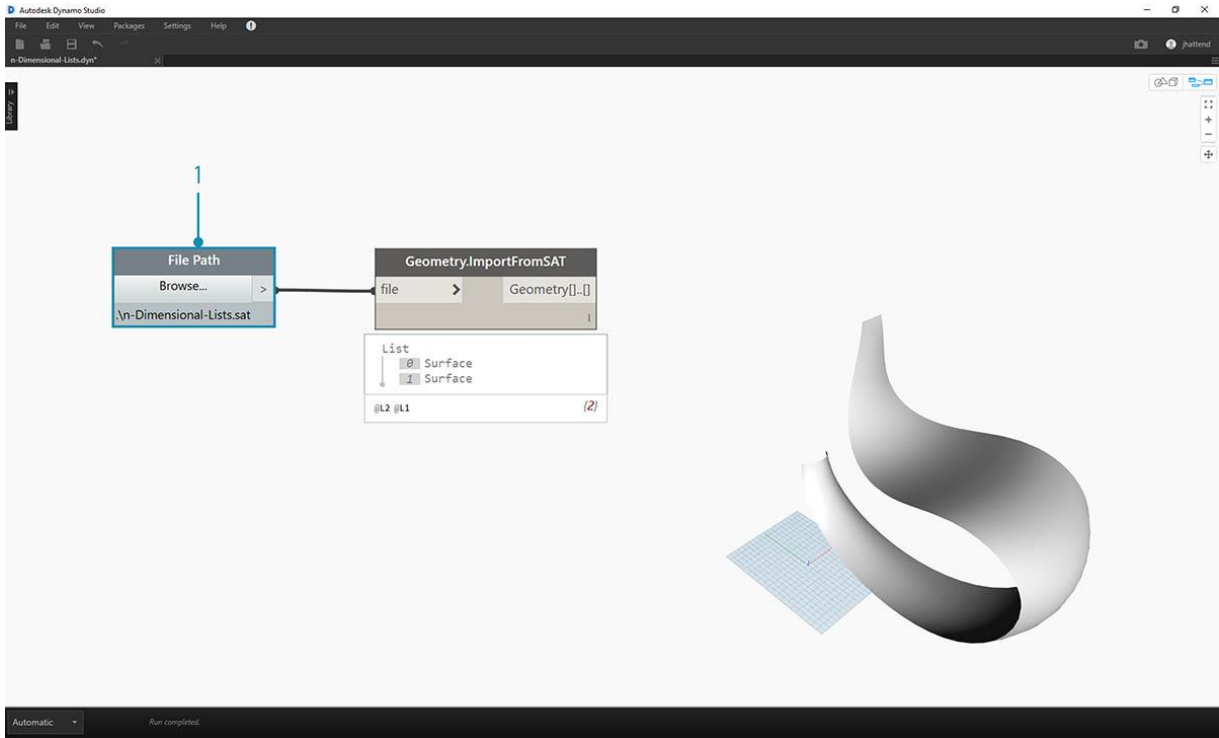
1. Una alternativa a *List.Transpose* usa *List.Combine*. Esto operará un "combinador" en cada sublista.
2. En este caso, estamos usando *List.Create* como el "combinador", que creará una lista de cada elemento en las sublistas.
3. Usando el nodo *Surface.ByLoft*, obtenemos las mismas superficies que en el paso anterior. Transpose es más fácil de usar en este caso, pero cuando la estructura de datos se vuelve aún más compleja, *List.Combine* es más confiable.



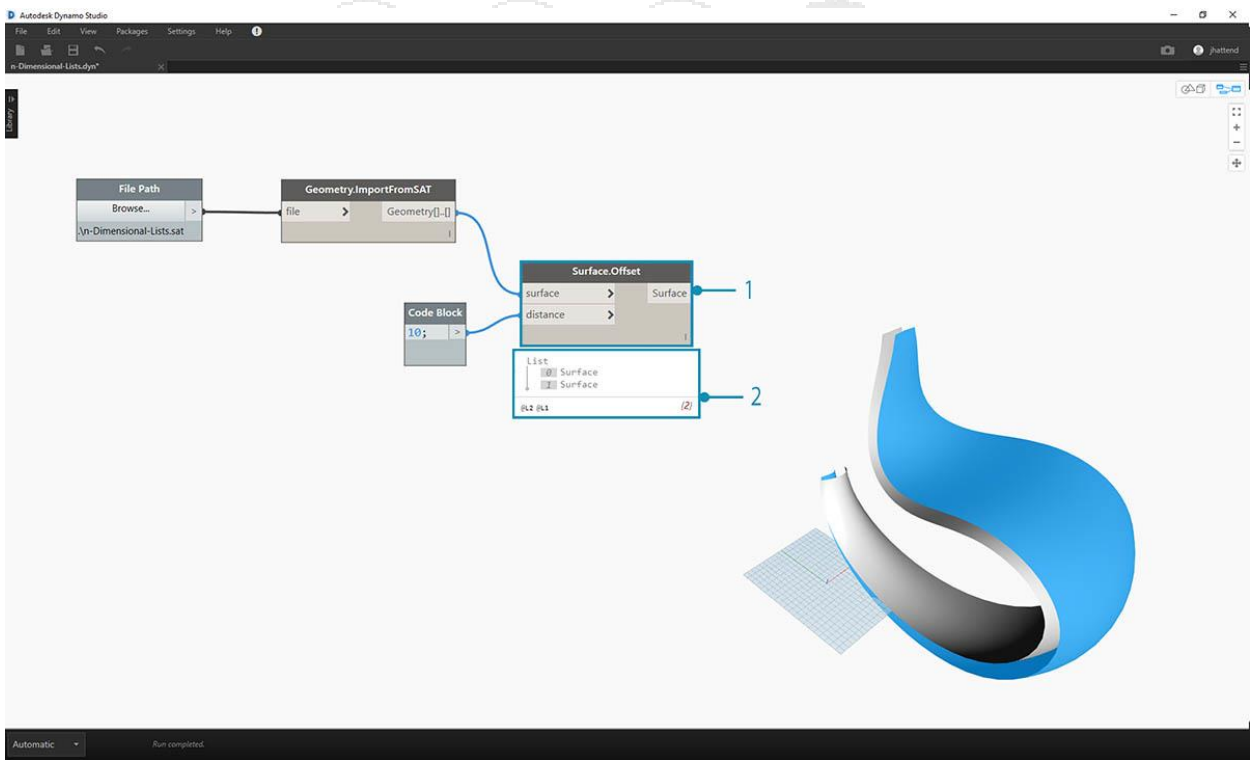
1. Retrocediendo unos pasos, si queremos cambiar la orientación de las curvas en la estructura nervada, queremos usar un *List.Transpose* antes de conectar con *NurbsCurve.ByPoints*. Esto volteará las columnas y las filas, dándonos 5 costillas horizontales.

Ejercicio - Listas 3D

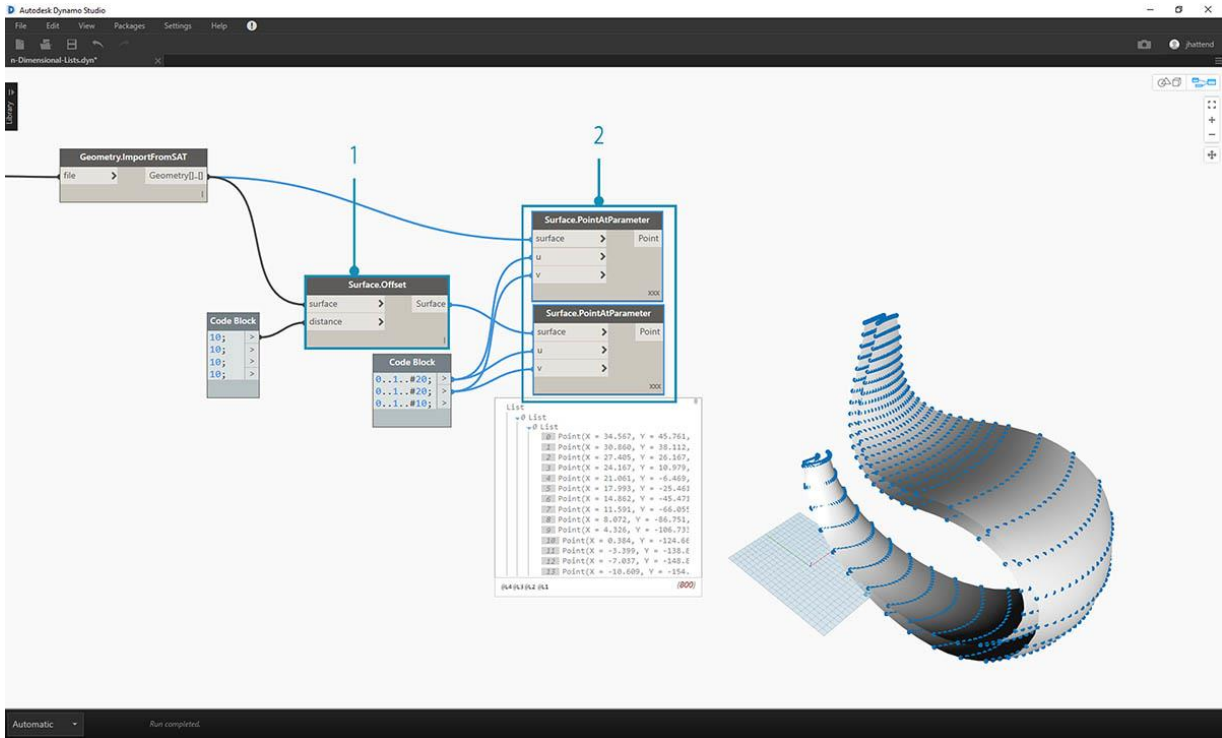
Ahora, vamos a ir incluso un paso más allá. En este ejercicio, trabajaremos con ambas superficies importadas, creando una jerarquía de datos compleja. Aún así, nuestro objetivo es completar la misma operación con la misma lógica subyacente.



1. Comience con el archivo importado del ejercicio anterior.

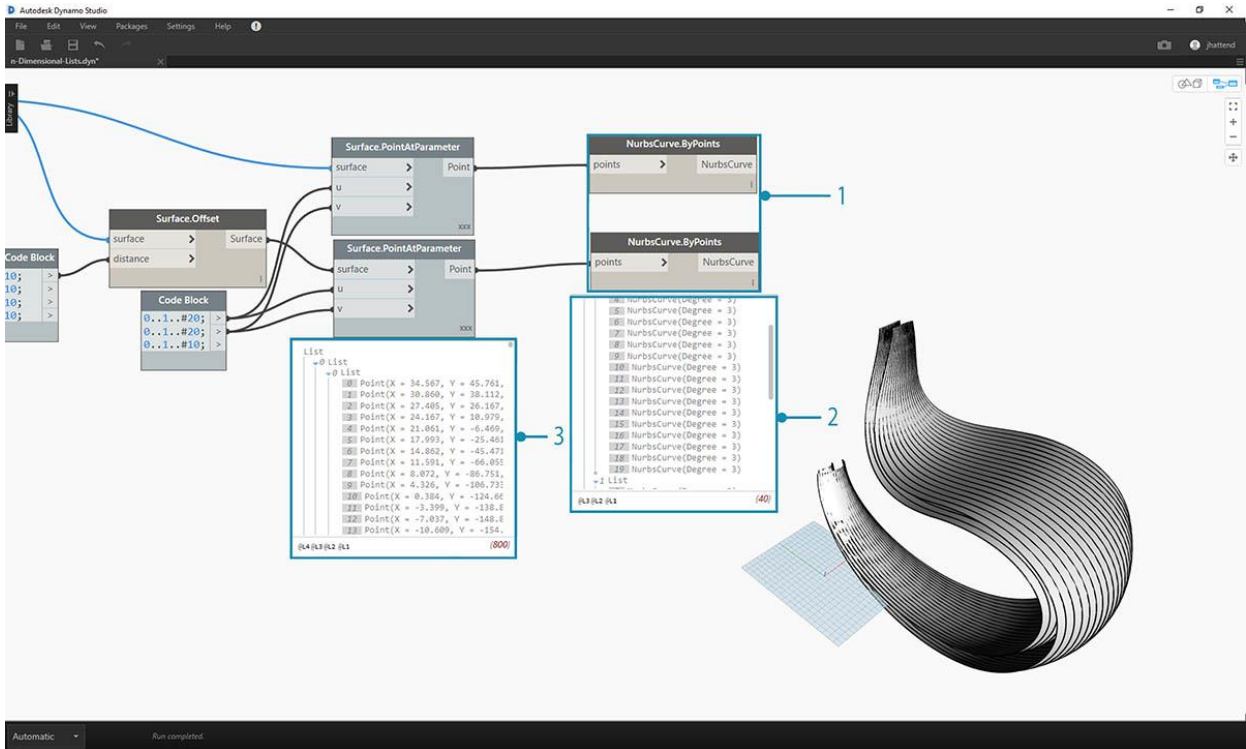


1. Como en el ejercicio anterior, use el nodo *Surface.Offset* para compensar con un valor de 10.
2. Fíjate en la salida que hemos creado dos superficies con el nodo de desplazamiento.



1. De la misma manera que el ejercicio anterior, defina un bloque de código con estas dos líneas de código:
2. 0..1..#20;
3. 0..1..#10;
4. Conecte estas salidas a dos nodos *Surface.PointAtParameter*, cada uno con el conjunto de lazos en "Cross Product". Uno de estos nodos está conectado a las superficies originales, mientras que el otro está conectado a las superficies de compensación.

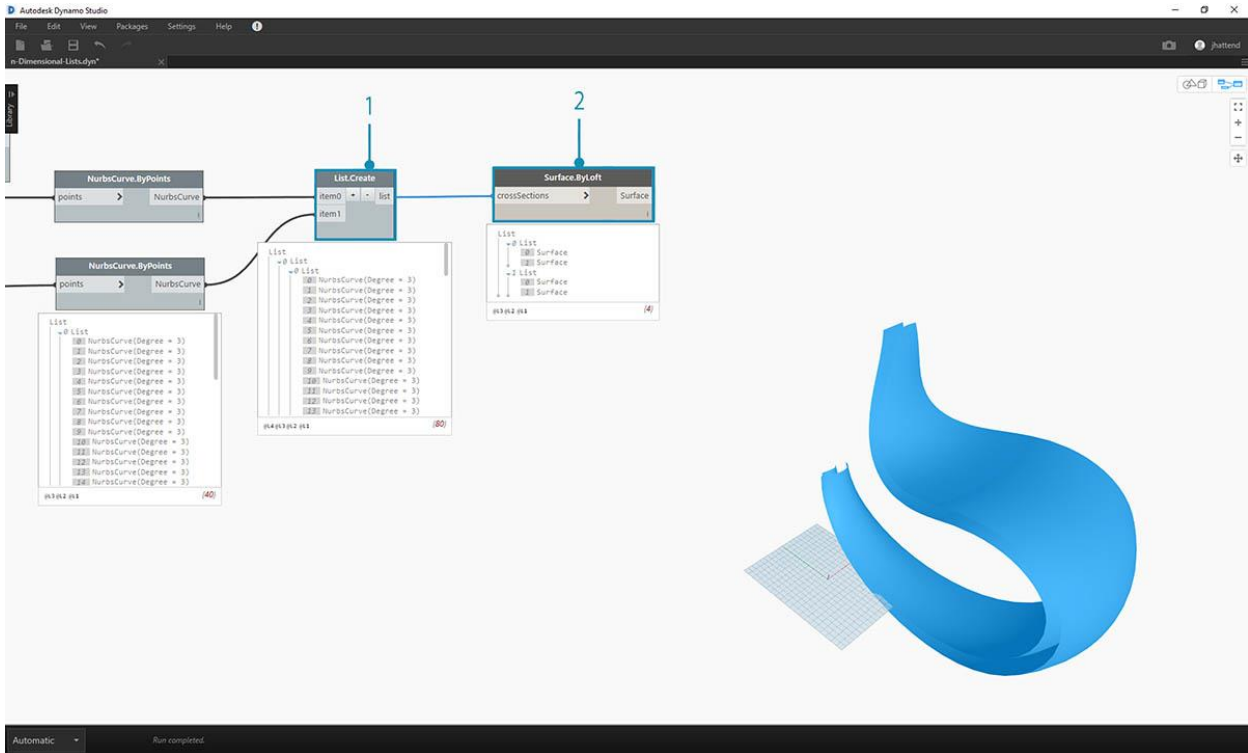
DARCO
DESDE 1988



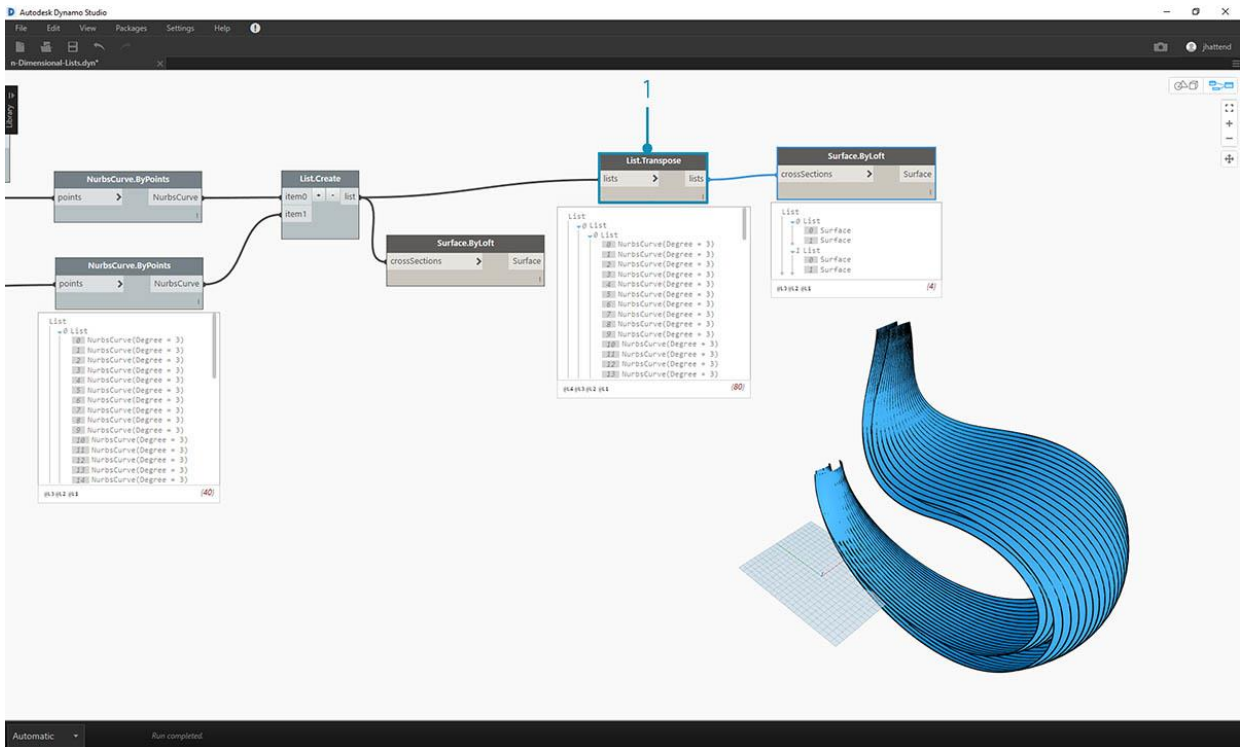
1. Como en el ejercicio anterior, conecte las salidas a dos nodos *NurbsCurve.ByPoints*.
2. Al mirar el resultado de *NurbsCurve.ByPoints*, observe que esta es una lista de dos listas, que es más compleja que el ejercicio anterior. Los datos se categorizan por la superficie subyacente, por lo que hemos agregado otro nivel a los datos estructurados.
3. Tenga en cuenta que las cosas se vuelven más complejas en el nodo *Surface.PointAtParameter*. En este caso, tenemos una lista de listas de listas.

DARCO

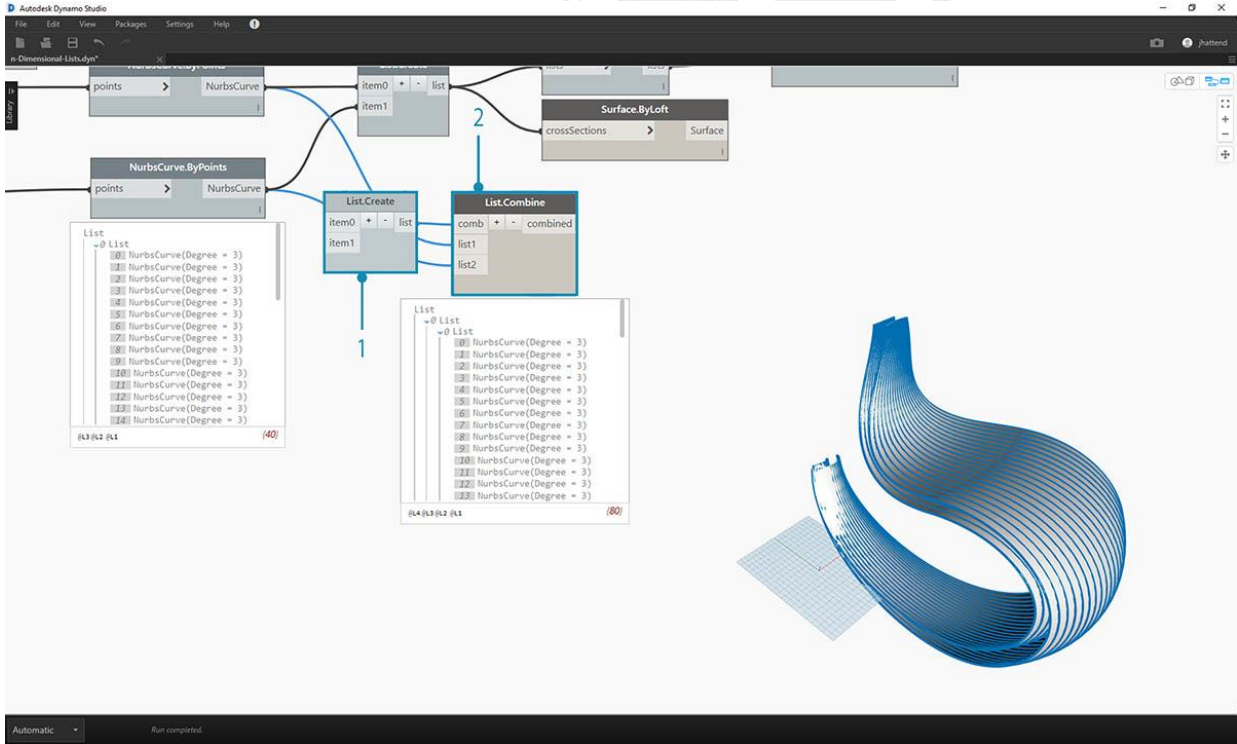
DESDE 1988



1. Usando el nodo *List.Create*, fusionamos las curvas nurbs en una estructura de datos, creando una lista de listas de listas.
2. Al conectar un nodo *Surface.ByLoft*, obtenemos una versión de las superficies originales, ya que cada una permanece en su propia lista como creada a partir de la estructura de datos original.

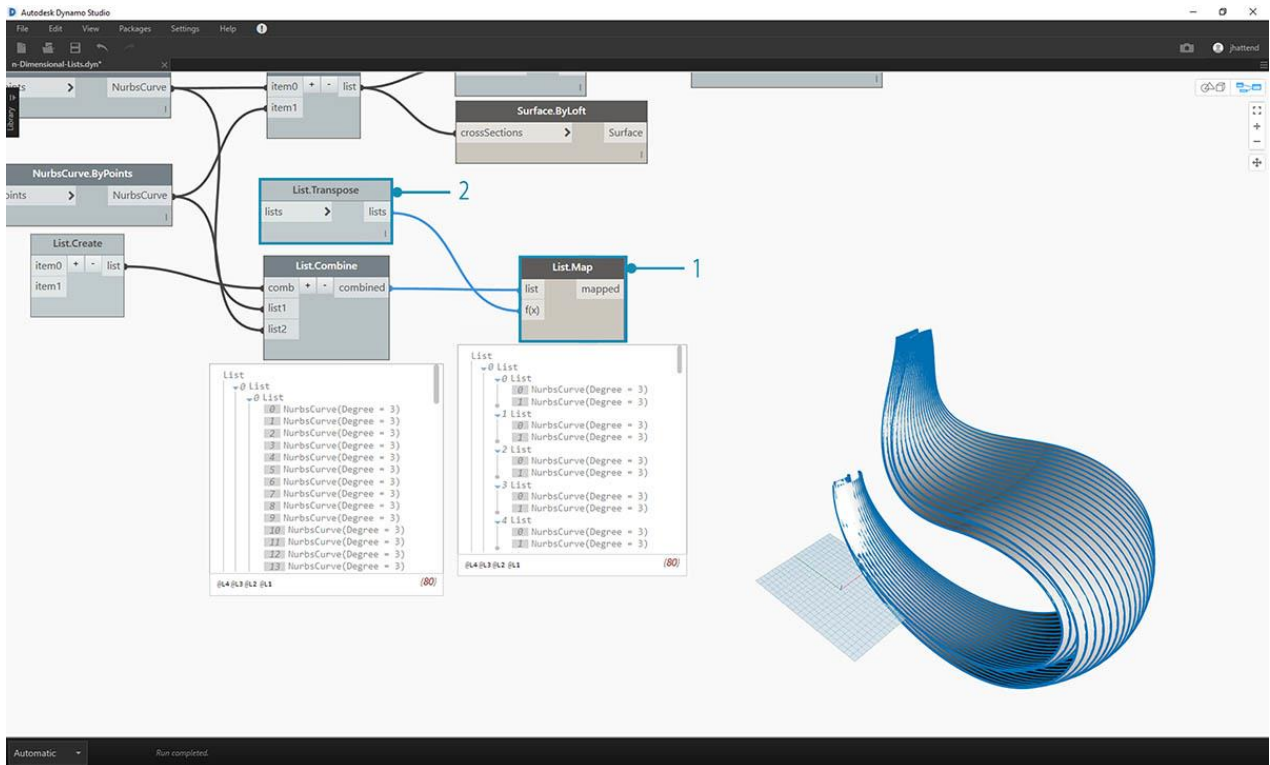


1. En el ejercicio anterior, pudimos usar un *List.Transpose* para crear una estructura acanalada. Esto no funcionará aquí. Una transposición se debe usar en una lista bidimensional, y dado que tenemos una lista tridimensional, una operación de "voltear columnas y filas" no funcionará tan fácilmente. Recuerde, las listas son objetos, por lo que *List.Transpose* *volteará* nuestras listas sin sublistas, pero no invertirá las curvas de nurbs una lista más abajo en la jerarquía.



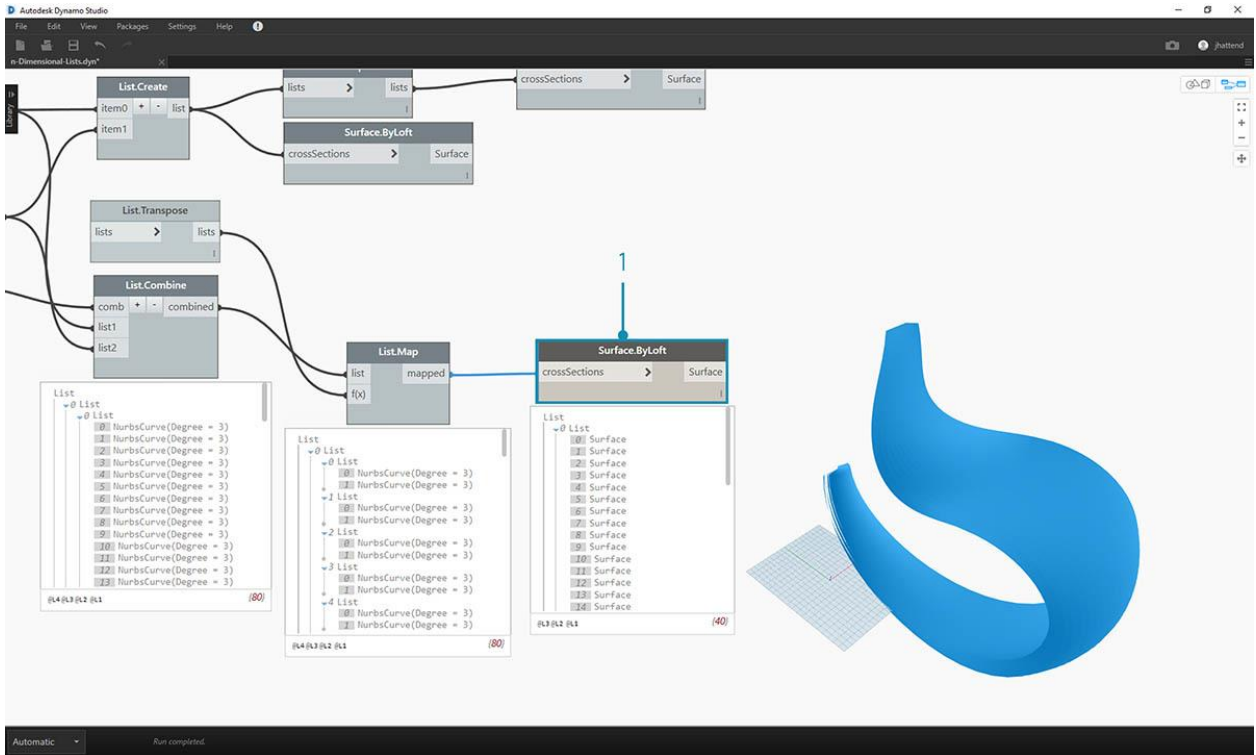
1. *List.Combine* funcionará mejor para nosotros aquí. Queremos usar los nodos *List.Map* y *List.Combine* cuando llegemos a estructuras de datos más complejas.
2. Usando *List.Create* como el "combinador", creamos una estructura de datos que funcionará mejor para nosotros.

DARCO
DESDE 1988

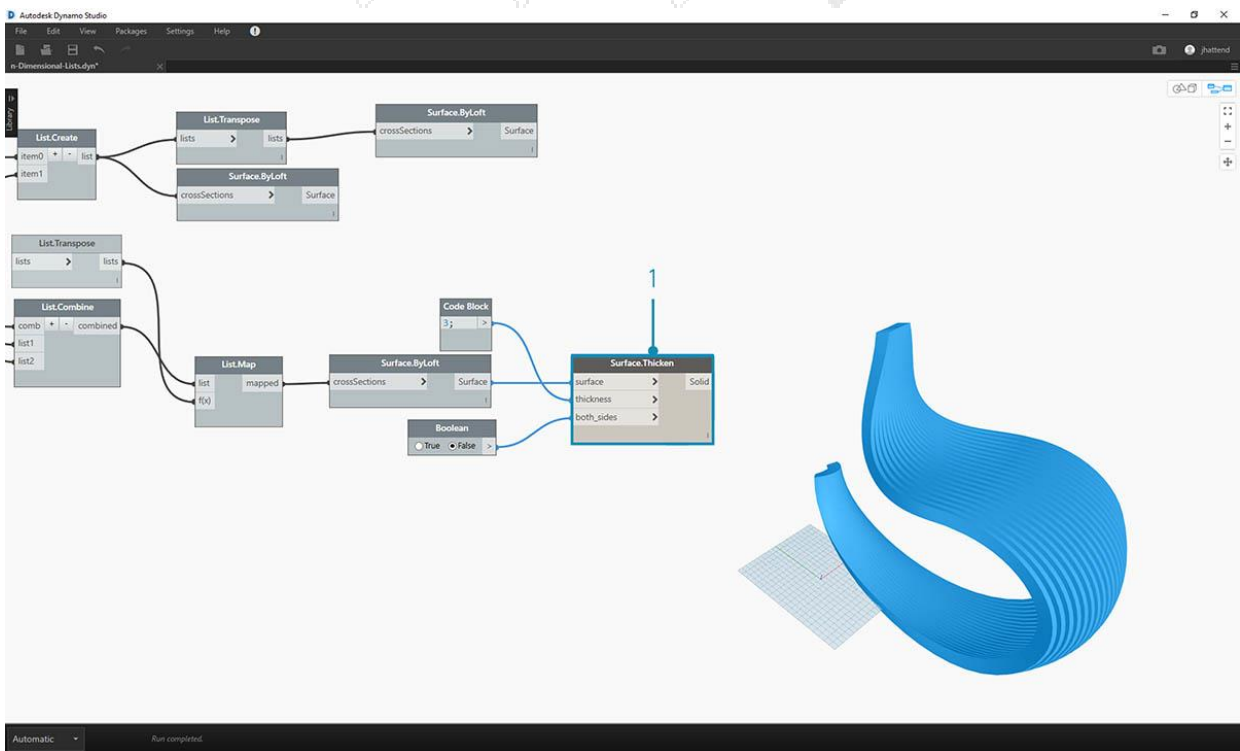


1. La estructura de datos aún necesita ser transpuesta en un paso hacia abajo en la jerarquía. Para hacer esto, utilizaremos *List.Map*. Esto funciona como *List.Combine*, excepto con una lista de entrada, en lugar de dos o más.
2. La función que aplicaremos a *List.Map* es *List.Transpose*, que *volteará* las columnas y filas de las sublistas dentro de nuestra lista principal.

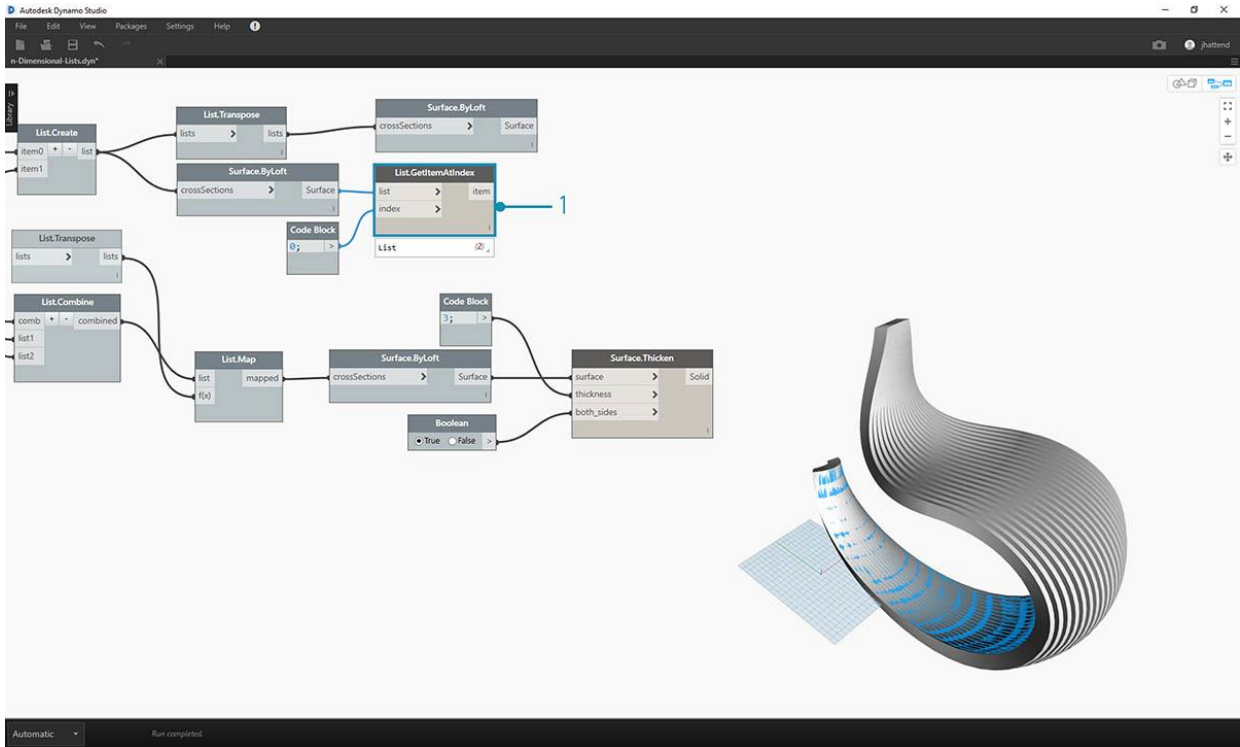
DARCO
 DESDE 1988



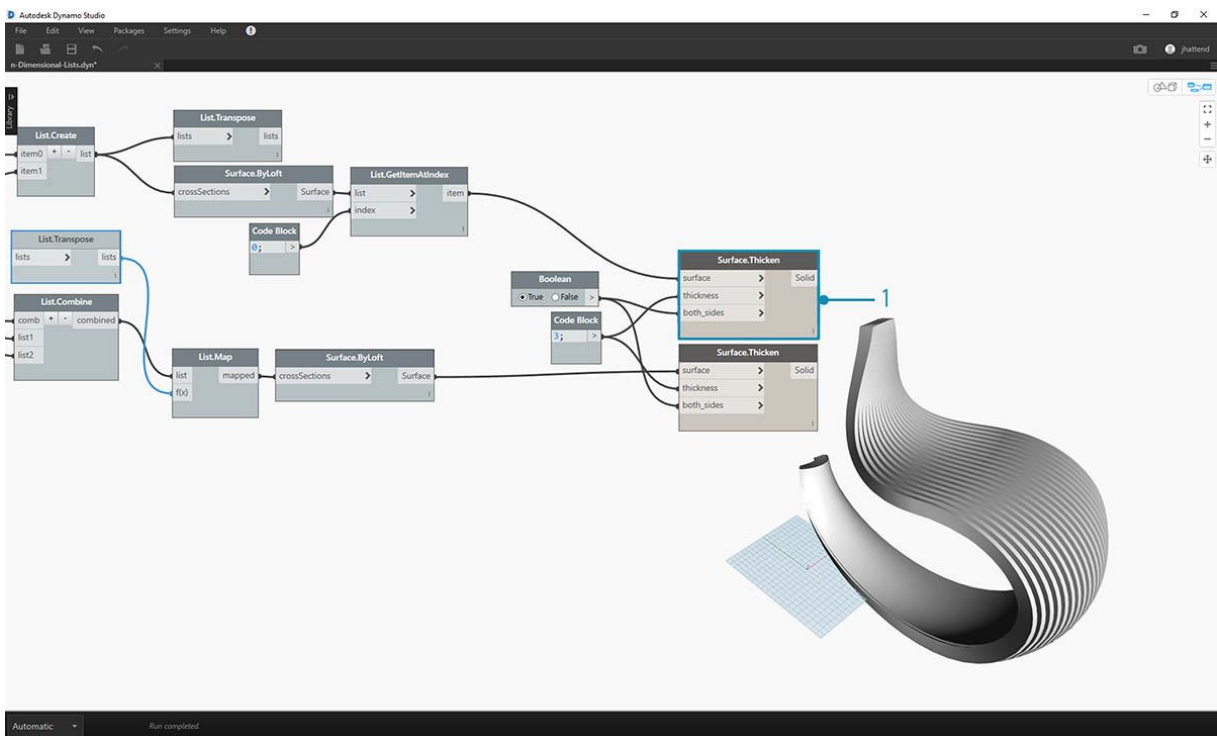
1. Finalmente, podemos allanar las curvas de nurbs junto con una jerarquía de datos adecuada, dándonos una estructura estriada.



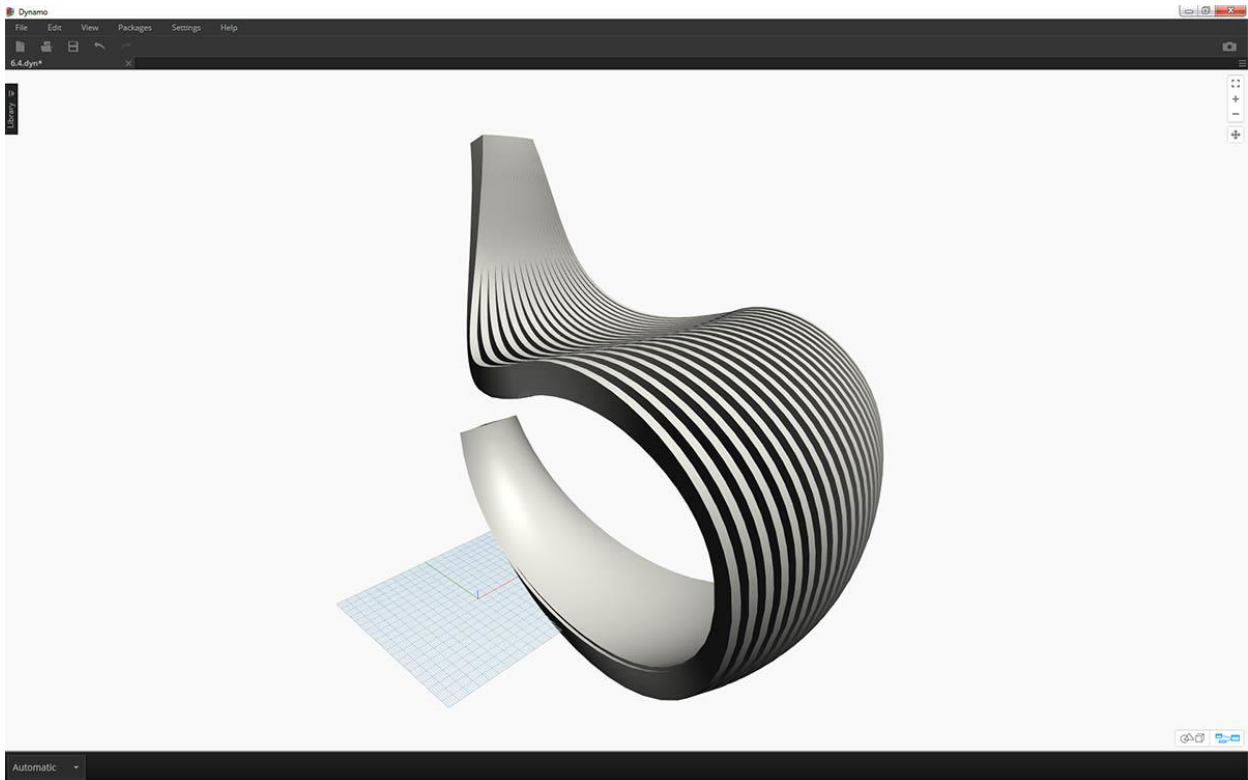
1. Agreguemos algo de profundidad a la geometría con un nodo *Surface.Thicken*.



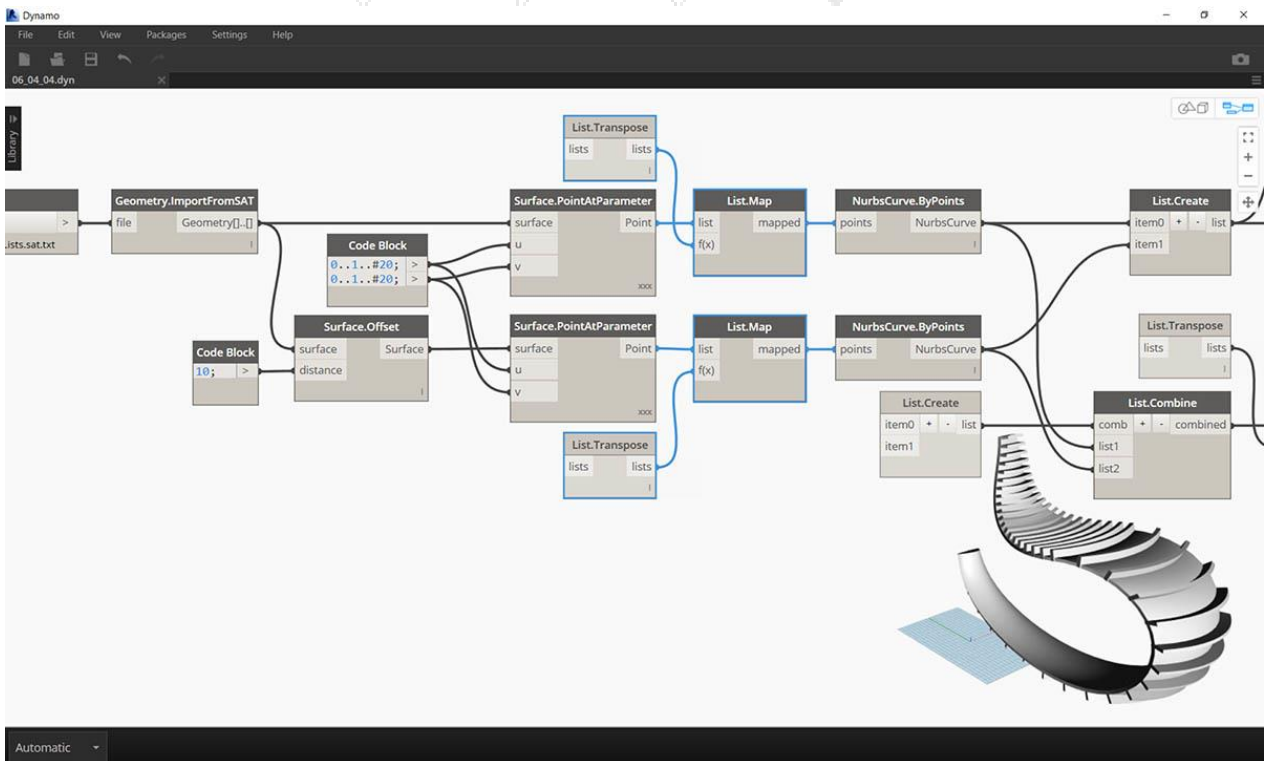
1. Sería bueno agregar un respaldo de superficie a esta estructura, entonces usaremos *List.GetItemAtIndex* para seleccionar la superficie posterior de las superficies desplegadas de los pasos anteriores.



1. Y espesando estas superficies seleccionadas, nuestra articulación es completa.

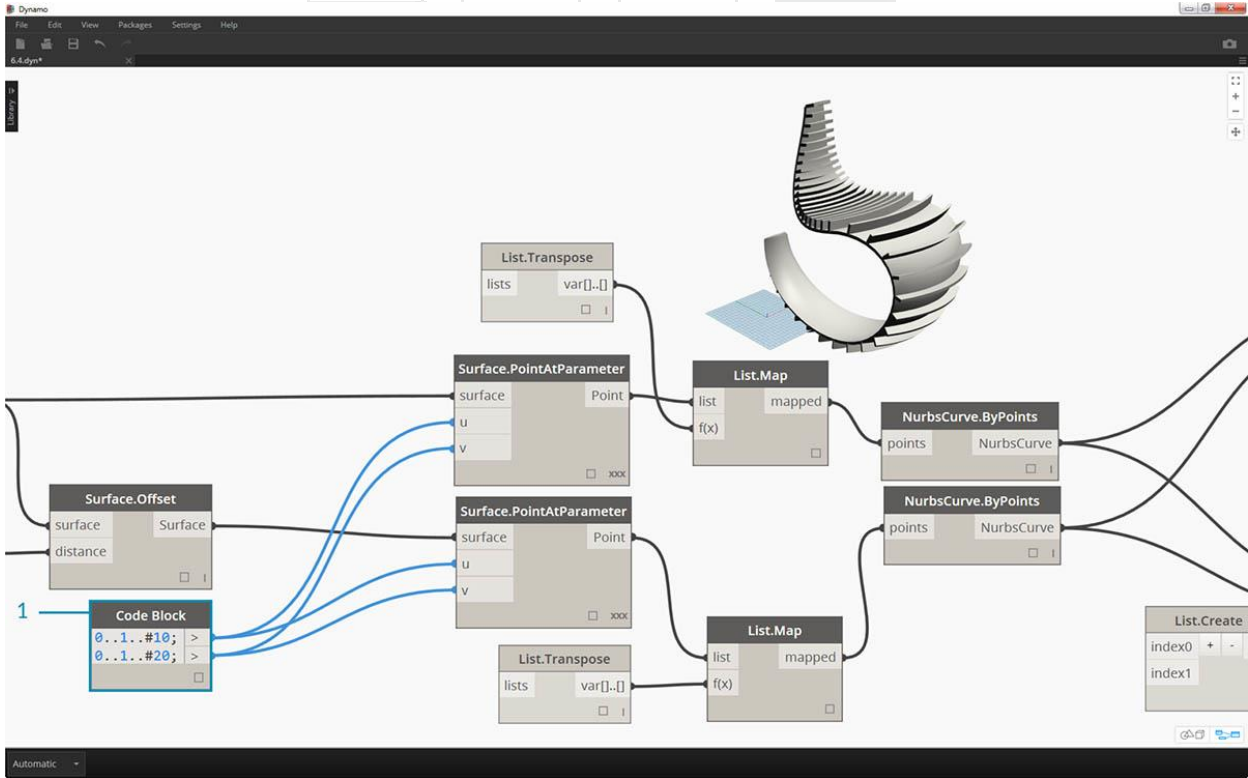


No es la mecedora más cómoda de la historia, pero tiene una gran cantidad de datos en funcionamiento.



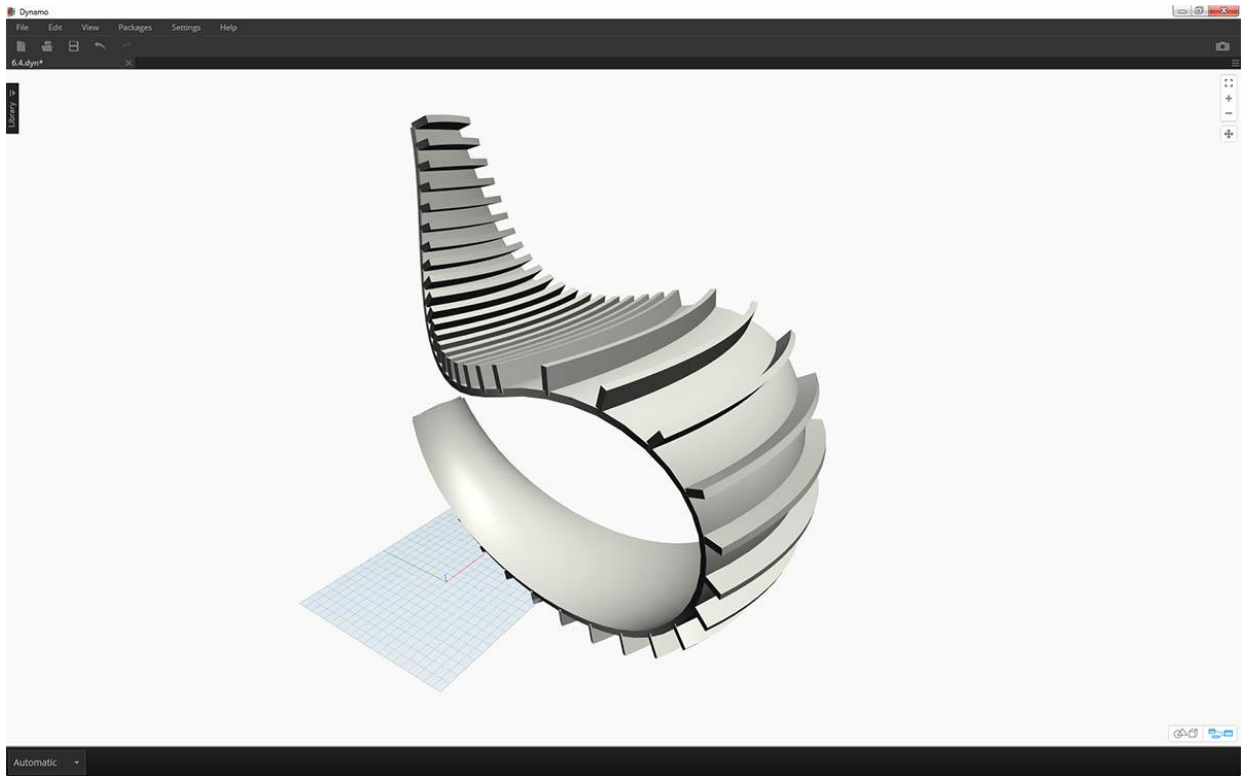
El último paso, invirtamos la dirección de los miembros estriados. Como usamos la transposición en el ejercicio anterior, haremos algo similar aquí.

1. Como tenemos un nivel más en la jerarquía, necesitamos usar *List.Map* con una función *List.Transpose* para cambiar la dirección de las curvas de nuestros nurbs.



1. Es posible que deseemos aumentar el número de huellas, por lo que podemos cambiar el bloque de código para
2. 0..1..#20;
3. 0..1..#10;

DARCO
DESDE 1988

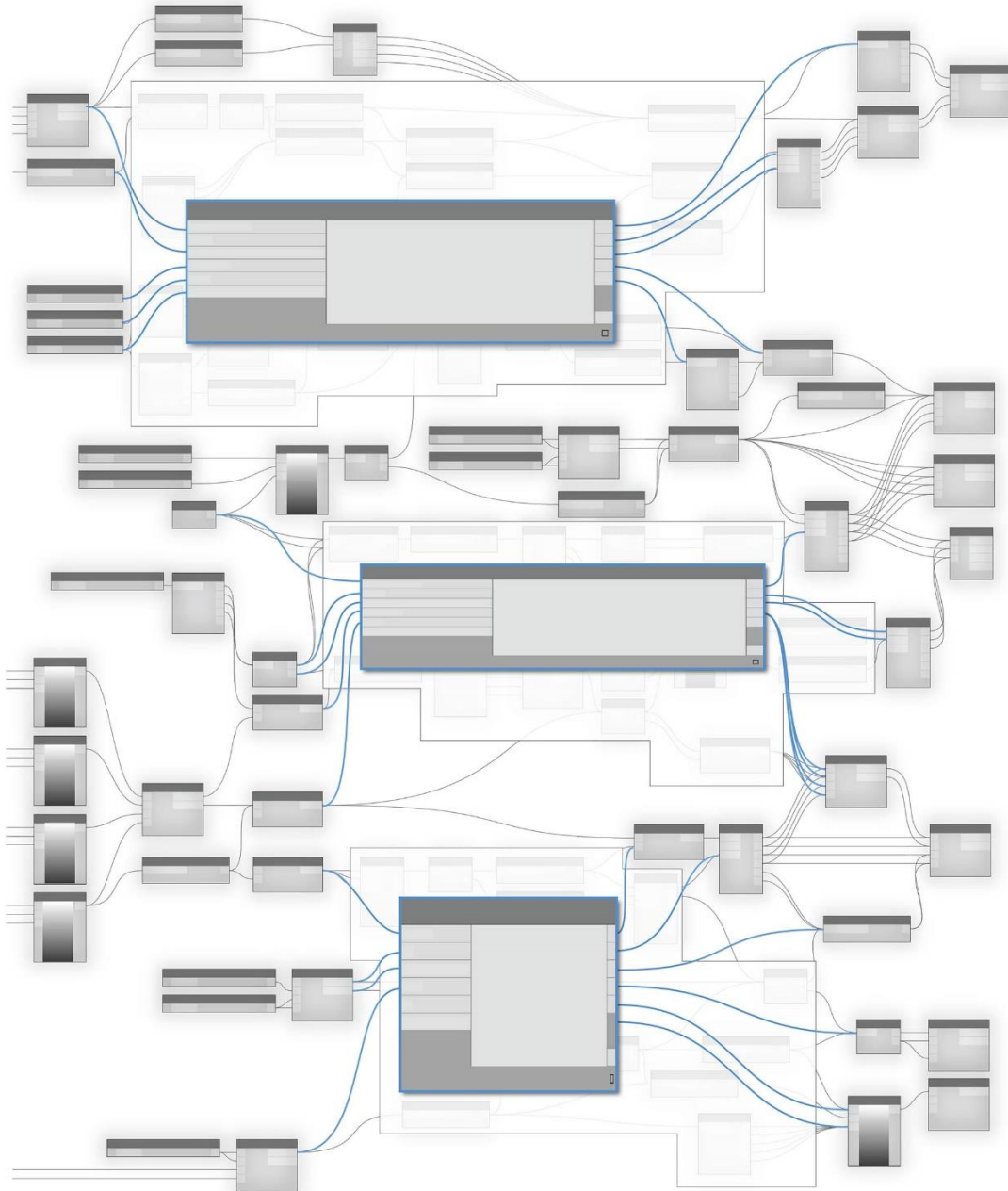


La primera versión de la mecedora era elegante, por lo que nuestro segundo modelo ofrece una versión recumbency de uso deportivo fuera de carretera.

DARCO
DESDE 1988

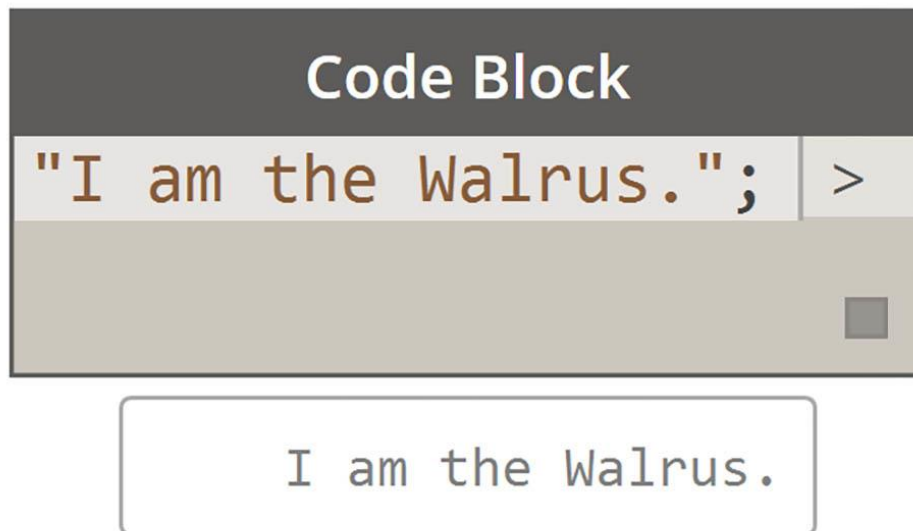
CODE BLOCKS Y DESIGNSCRIPT

El bloque de código es una característica única en Dynamo que vincula dinámicamente un entorno de programación visual con uno basado en texto. El bloque de código tiene acceso a todos los nodos de Dynamo y puede definir un gráfico completo en un nodo. Lea este capítulo detenidamente, ya que el bloque de código es un componente fundamental de Dynamo.



¿Qué es un bloque de código?

Los bloques de código son una ventana profunda en DesignScript, el lenguaje de programación en el corazón de Dynamo. Construido desde cero para soportar flujos de trabajo de diseño exploratorio, DesignScript es un lenguaje legible y conciso que ofrece retroalimentación inmediata a pequeños fragmentos de código y también escala a interacciones grandes y complejas. DesignScript también forma la columna vertebral del motor que impulsa la mayoría de los aspectos de Dynamo "debajo del capó". Debido a que casi toda la funcionalidad que se encuentra en los nodos e interacciones de Dynamo tiene una relación de uno a uno con el lenguaje de scripting, existen oportunidades únicas para moverse entre las interacciones basadas en nodos y los scripts de forma fluida.

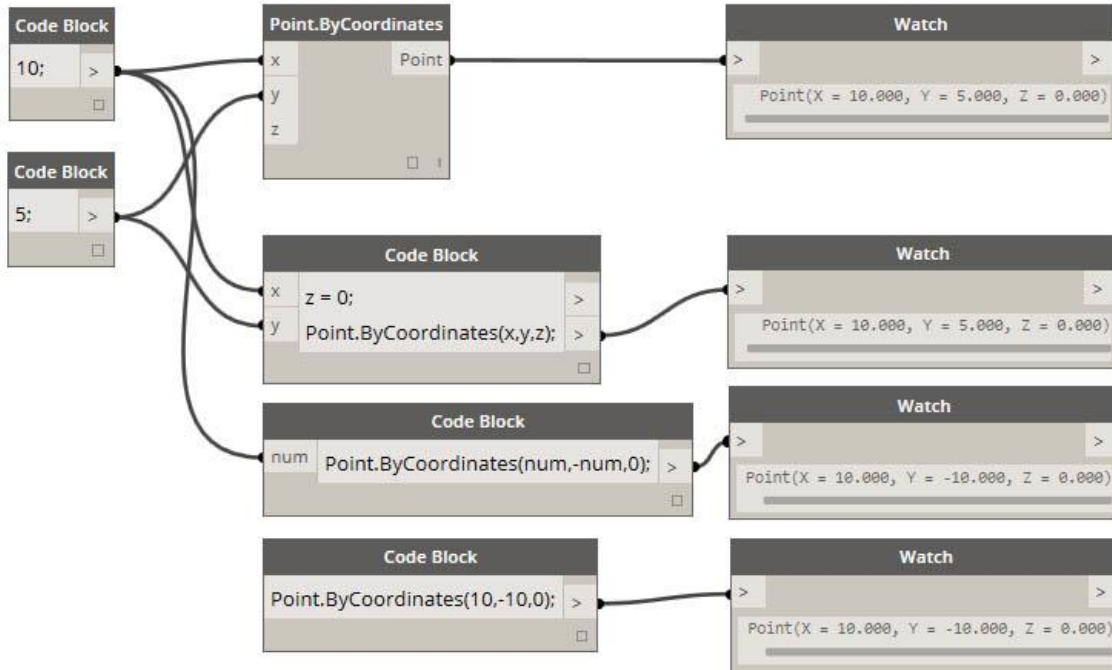


Para los principiantes, los nodos pueden convertirse automáticamente en sintaxis de texto para ayudarlos a aprender DesignScript o simplemente para reducir el tamaño de secciones de gráficos más grandes. Esto se hace usando un proceso llamado "Nodo para codificar", que se describe con más detalle en la sección **Sintaxis de DesignScript**. Los usuarios más experimentados pueden usar Code Blocks para crear mashups personalizados de la funcionalidad existente y las relaciones creadas por los usuarios usando muchos paradigmas de codificación estándar. Entre el usuario principiante y avanzado, hay una gran cantidad de accesos directos y fragmentos de código que acelerarán sus diseños. Si bien el término "bloque de código" puede ser un poco intimidante para los no programadores, es fácil de usar y robusto. Un principiante puede usar el bloque de código de manera eficiente con una codificación mínima, y un usuario avanzado puede definir definiciones guionizadas para que sean recordadas en otra parte de una definición de Dynamo.

Bloque de código: una breve descripción

En resumen, los bloques de código son una interfaz de secuencias de comandos de texto dentro de un entorno de scripting visual. Se pueden usar como números, cadenas, fórmulas y otros tipos de datos. El bloque de código está diseñado para Dynamo, por lo que uno puede definir variables arbitrarias en el bloque de código, y esas variables se agregan automáticamente a las entradas del nodo:

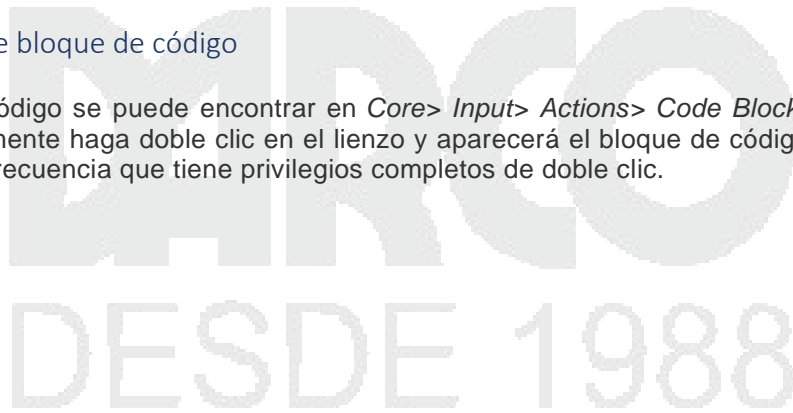
Con los bloques de código, un usuario tiene la flexibilidad de decidir cómo especificar las entradas. Aquí hay varias maneras diferentes de hacer un punto básico con coordenadas (10, 5, 0):

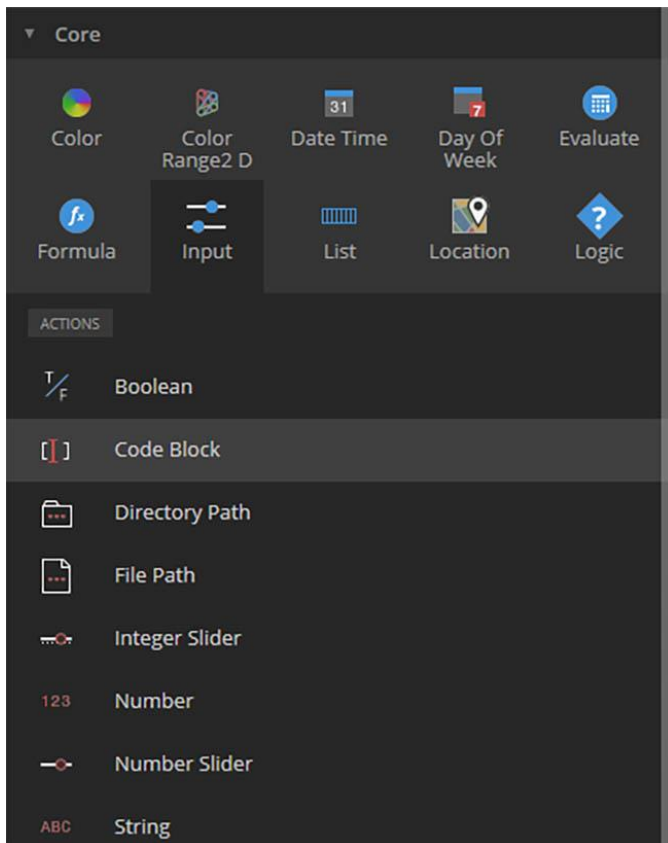


A medida que aprenda más de las funciones disponibles en la biblioteca, incluso podría encontrar que escribir "Point.ByCoordinates" es más rápido que buscar en la biblioteca y encontrar el nodo apropiado. Cuando escribe "Punto", por ejemplo, Dynamo mostrará una lista de posibles funciones para aplicar a un Punto. Esto hace que los scripts sean más intuitivos y ayudará a aprender cómo aplicar funciones en Dynamo.

Crear nodos de bloque de código

El bloque de código se puede encontrar en `Core > Input > Actions > Code Block`. Pero aún más rápido, simplemente haga doble clic en el lienzo y aparecerá el bloque de código. Este nodo se usa con tanta frecuencia que tiene privilegios completos de doble clic.

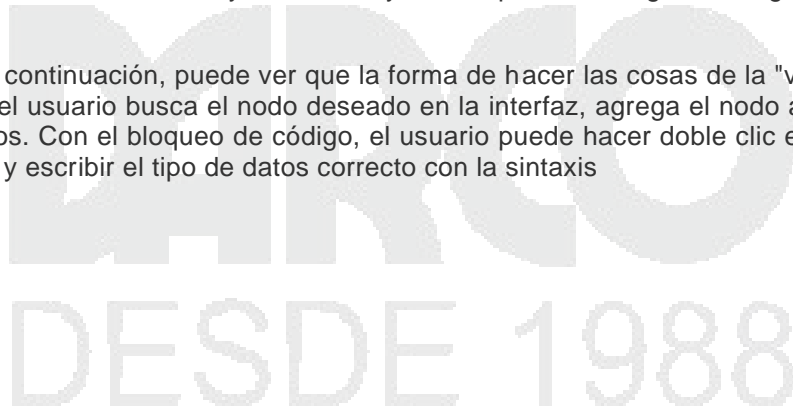




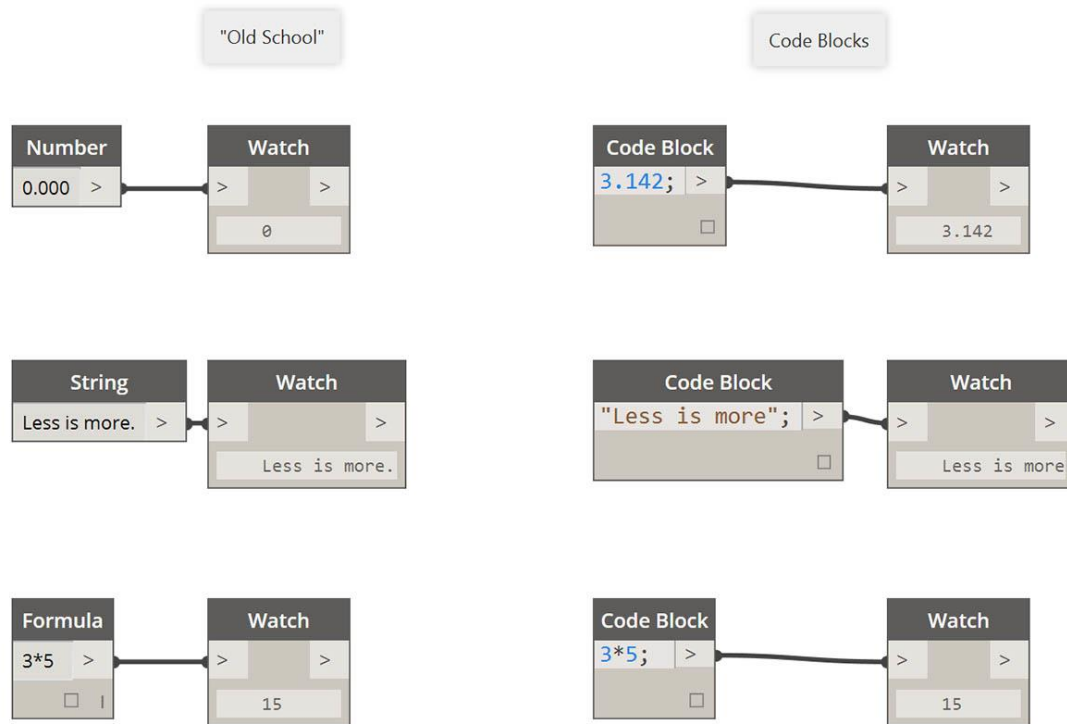
Números, cadenas y fórmulas

Los bloques de código también son flexibles para los tipos de datos. El usuario puede definir rápidamente números, cadenas y fórmulas, y el bloque de códigos entregará el resultado deseado.

En la imagen a continuación, puede ver que la forma de hacer las cosas de la "vieja escuela" es un poco larga: el usuario busca el nodo deseado en la interfaz, agrega el nodo al lienzo y luego ingresa los datos. Con el bloqueo de código, el usuario puede hacer doble clic en el lienzo para extraer el nodo y escribir el tipo de datos correcto con la sintaxis



básica.



Los nodos de *número*, *cadena* y *fórmula* son tres ejemplos de nodos de Dynamo que, sin duda, son obsoletos en comparación con el *bloque de código*.

Sintaxis de DesignScript

Puede haber notado un tema común en los nombres de los nodos en Dynamo: cada nodo usa un "." sintaxis sin espacios. Esto se debe a que el texto en la parte superior de cada nodo representa la sintaxis real para la creación de scripts, y el "." (o *notación de puntos*) separa un elemento de los posibles métodos que podemos llamar. Esto crea una traducción fácil de scripting visual a scripting basado en texto.



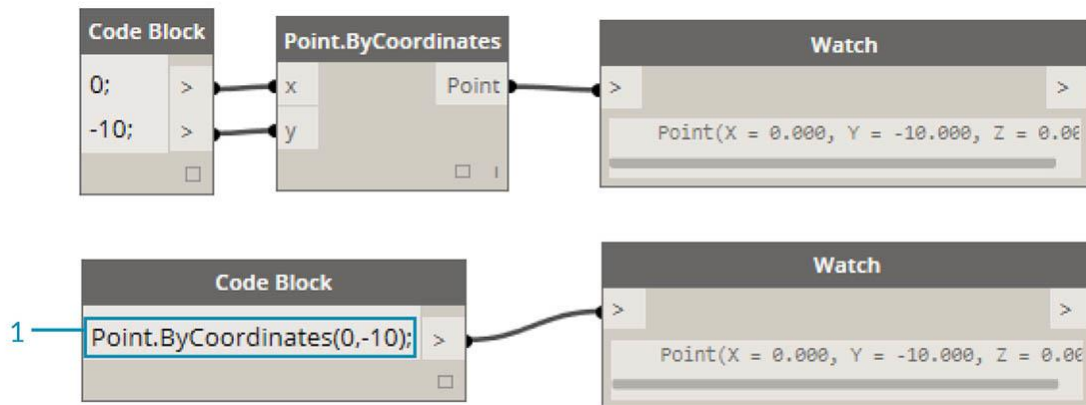
Como una analogía general para la notación de puntos, ¿cómo podemos lidiar con una manzana paramétrica en Dynamo? A continuación, presentamos algunos métodos que ejecutaremos en la manzana antes de decidir comerlos. (Nota: estos no son métodos reales de Dynamo):

Humanamente Readible	Notación de puntos	Salida
¿De qué color es la manzana?	Apple.color	rojo
¿Está madura la manzana?	Apple.isRipe	cierto
¿Cuánto pesa la manzana?	Apple.weight	6 onzas.
¿De dónde vino la manzana?	Apple.parent	árbol
¿Qué crea la manzana?	Apple.children	semillas
¿Esta manzana se cultiva localmente?	Apple.distanceFromOrchard	60 mi.

No sé ustedes, pero a juzgar por los resultados de la tabla anterior, parece una manzana sabrosa. Creo que voy a *Apple.eat ()*.

Notación de puntos en el bloque de código

Con la analogía de la manzana en mente, veamos *Point.ByCoordinates* y mostremos cómo podemos crear un punto usando el bloque de código:



La sintaxis de *bloque de código* *Point.ByCoordinates(0,10);* da el mismo resultado que un nodo *Point.ByCoordinates* en Dynamo, excepto que podemos crear un punto usando un nodo. Esto es más eficiente que la conexión de un nodo separado a "X" e "Y".

1. Al utilizar *Point.ByCoordinates* en el bloque de código, estamos especificando las entradas en el mismo orden que el nodo out-of-the-box (X, Y).

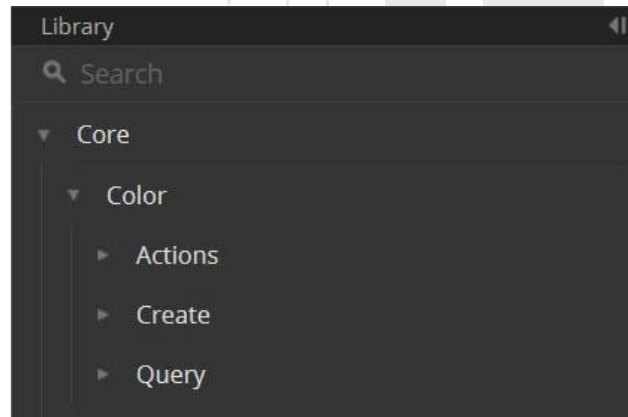
Nodos de llamada

Puede llamar a cualquier nodo regular en la biblioteca a través de un Bloque de código, siempre que el nodo no sea un nodo especial "UI": aquellos con una función de interfaz de usuario especial. Por ejemplo, puede llamar a *Circle.ByCenterPointRadius*, pero no tendría mucho sentido llamar a un nodo *Watch 3D*.

Los nodos regulares (la mayoría de su biblioteca) generalmente vienen en tres tipos:

- **Crear** - Crear (o construir) algo
- **Acción** : realizar una acción en algo
- **Consulta** : obtener una propiedad de algo que ya existe

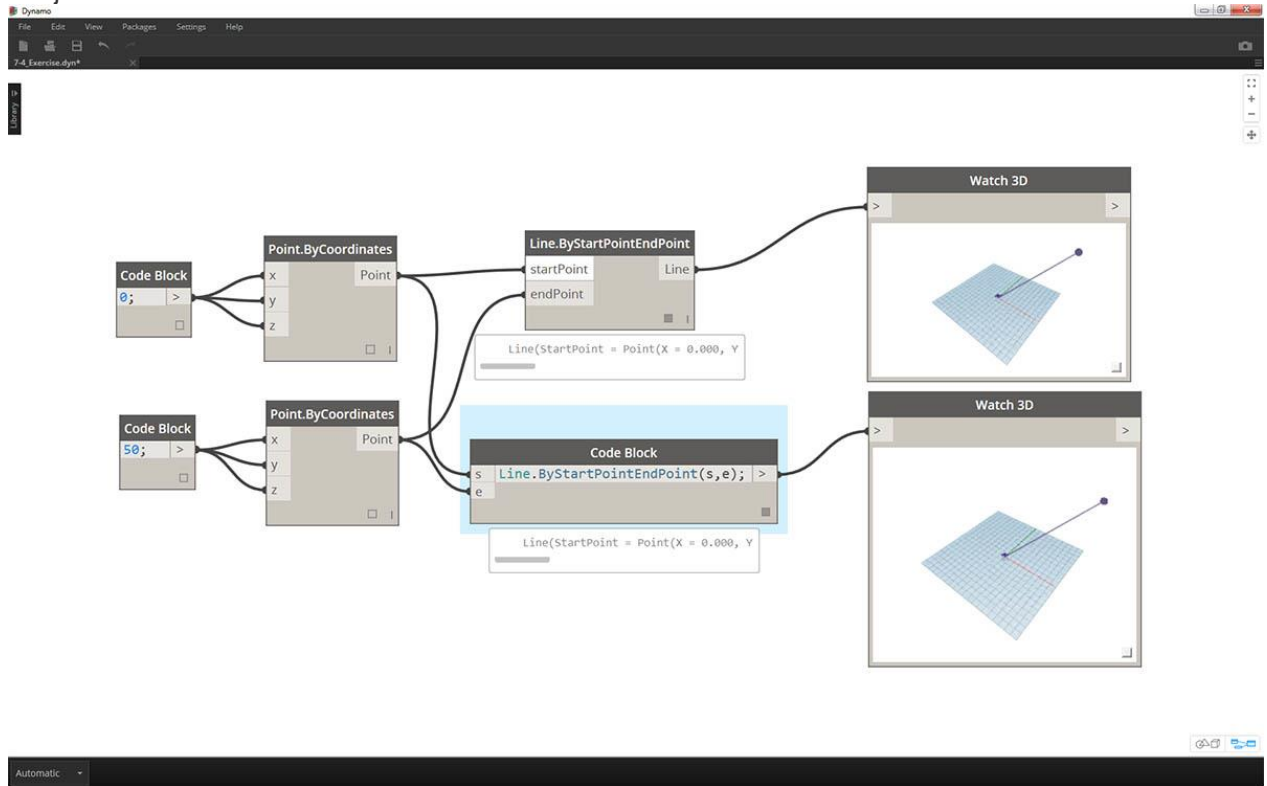
Descubrirá que la biblioteca está organizada teniendo en cuenta estas categorías. Los métodos o nodos de estos tres tipos se tratan de forma diferente cuando se invocan dentro de un Bloque de Código.



Crear

La categoría "Crear" construirá geometría desde cero. Ingresamos valores en el bloque de código de izquierda a derecha. Estas entradas están en el mismo orden que las entradas en el nodo de arriba a

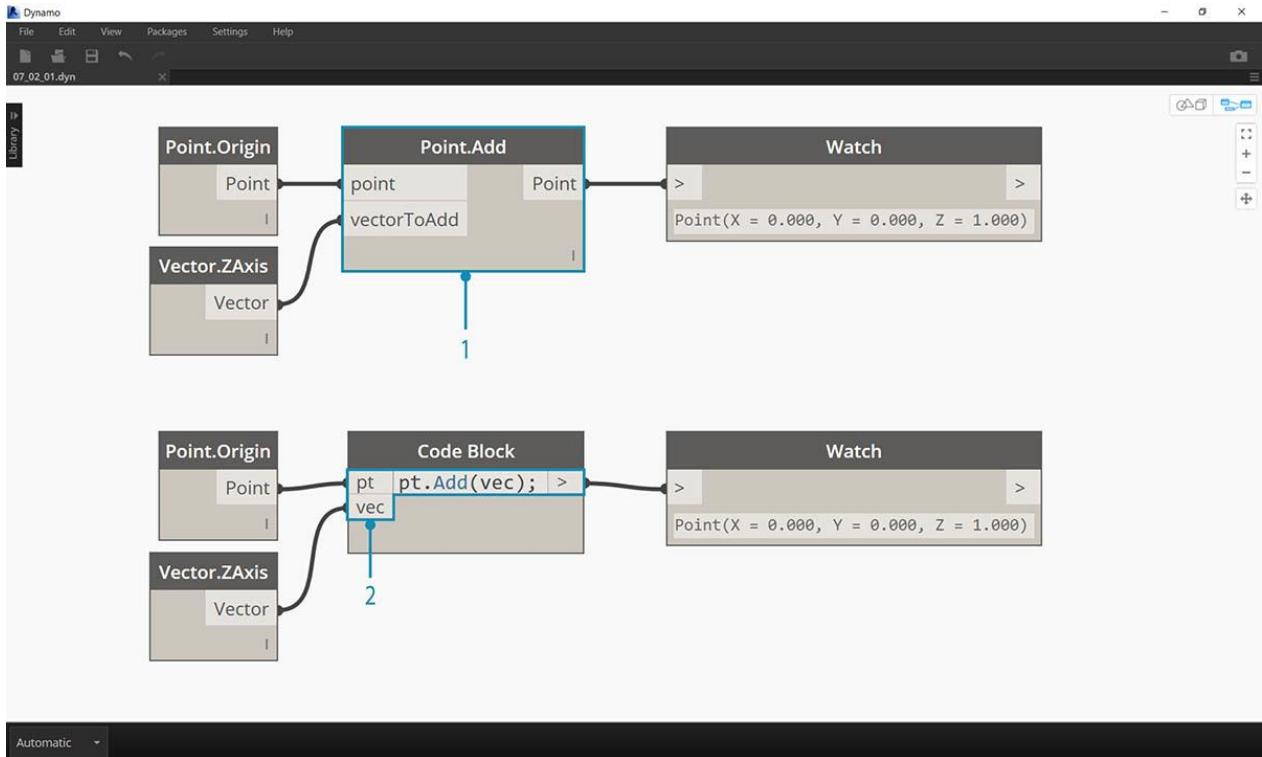
abajo:



Al comparar el nodo *Line.ByStartPointEndPoint* y la sintaxis correspondiente en el bloque de código, obtenemos los mismos resultados.

Acción

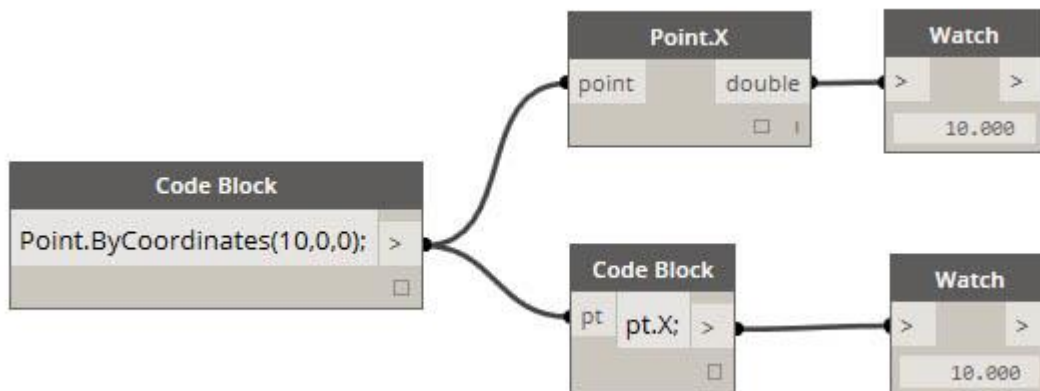
Una acción es algo que le haces a un objeto de ese tipo. Dynamo utiliza *la notación de puntos*, común en muchos lenguajes de codificación, para aplicar una acción a una cosa. Una vez que tenga el objeto, escriba un punto y luego el nombre de la acción. La entrada del método de acción se coloca entre paréntesis al igual que los métodos de tipo de creación, solo que no es necesario especificar la primera entrada que ve en el nodo correspondiente. En cambio, especificamos el elemento sobre el que estamos realizando la acción:



1. El nodo *Point.Add* es un nodo de tipo acción, por lo que la sintaxis funciona de forma un poco diferente.
2. Las entradas son (1) el punto y (2) el vector para agregarlo. En un *bloque de código*, hemos nombrado el punto (la cosa) "pt". Para agregar un vector llamado "vec" a "pt", escribiríamos `pt.Add(vec)`, o: cosa, punto, acción. La acción Agregar solo tiene una entrada, o todas las entradas del nodo *Punto.Añadir* menos la primera. La primera entrada para el nodo *Point.Add* es el punto en sí.

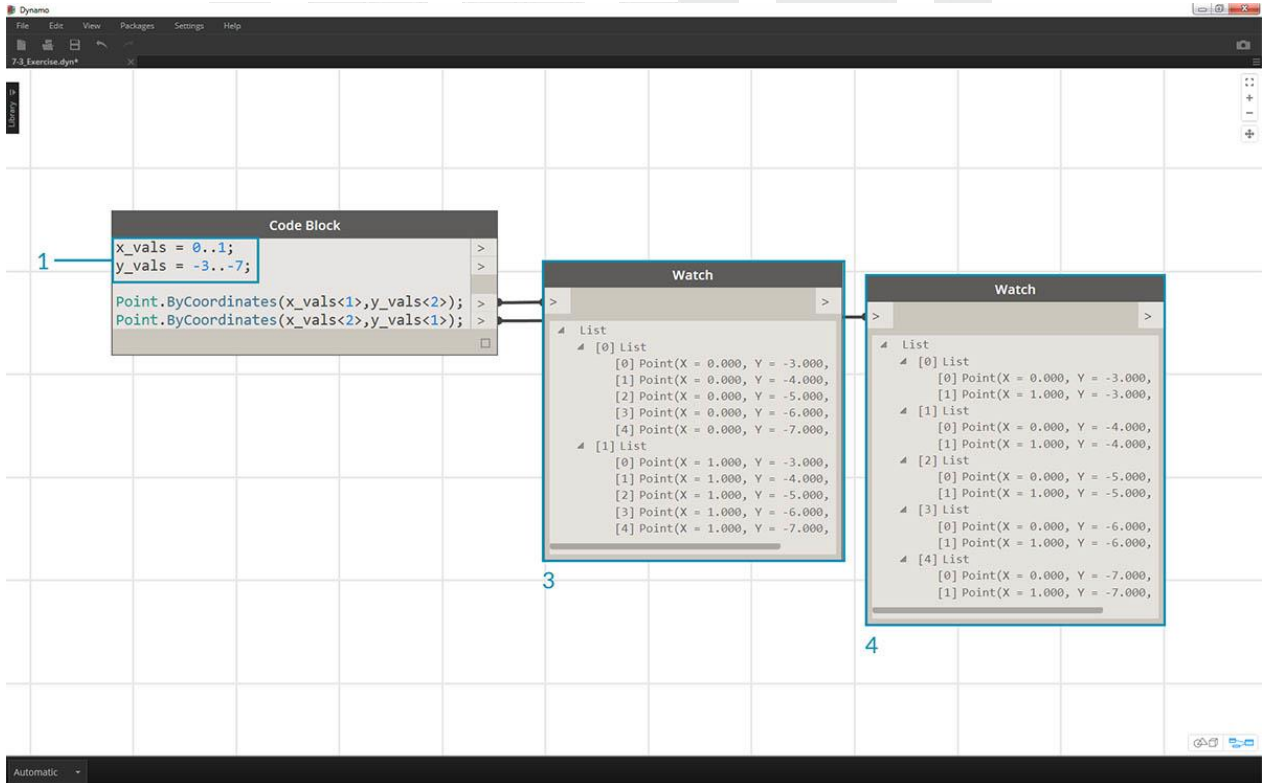
Consulta

El método de tipo consulta obtienen una propiedad de un objeto. Como el objeto en sí es la entrada, no tiene que especificar ninguna entrada. No se requieren paréntesis



¿Qué tal el cordón?

El enlazado con nodos es algo diferente de la vinculación con el bloque de código. Con los nodos, el usuario hace clic derecho en el nodo y selecciona la opción de ajuste para realizar. Con el bloqueo de código, el usuario tiene mucho más control sobre cómo se estructuran los datos. El método de taquigrafía de bloque de código utiliza *guías de replicación* para establecer cómo se deben emparejar varias listas unidimensionales. Los números en paréntesis angulares "<>" definen la jerarquía de la lista anidada resultante: <1>, <2>, <3>, etc.



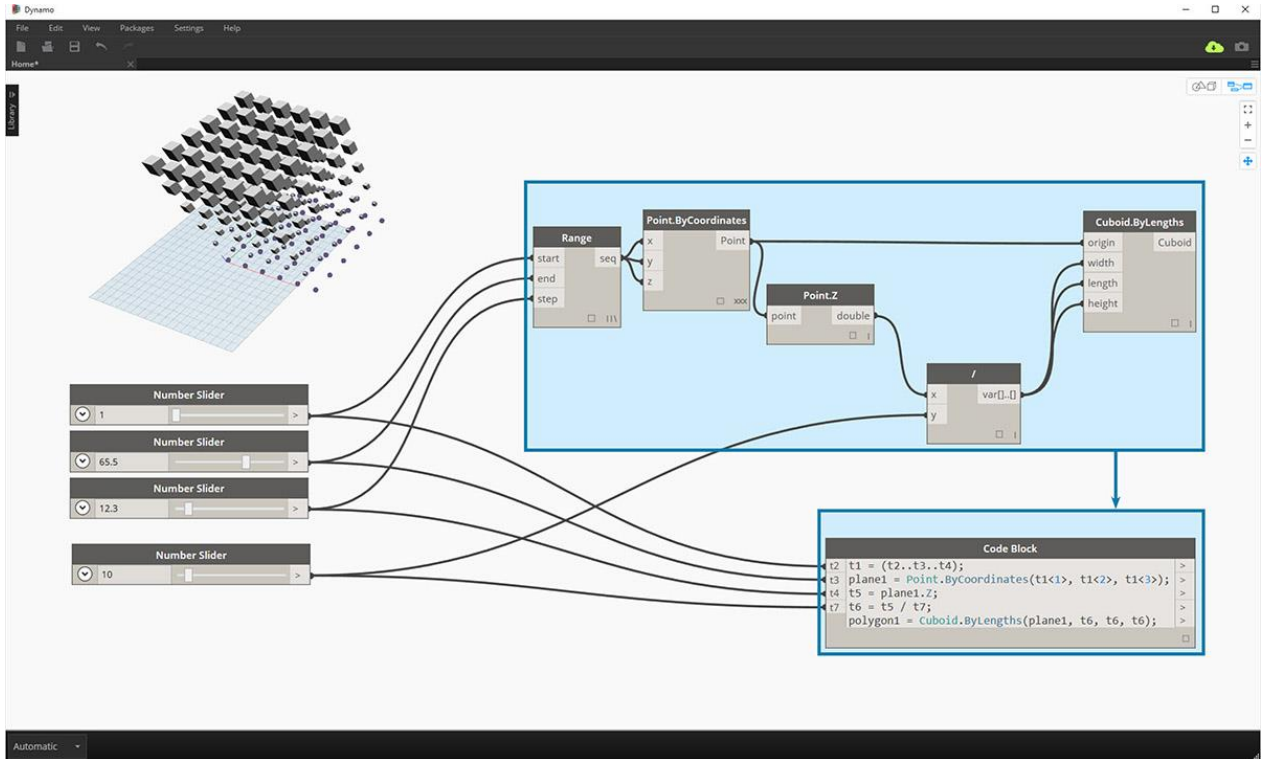
1. En este ejemplo, usamos una taquigrafía para definir dos rangos (más en taquigrafía en la siguiente sección de este capítulo). En resumen, 0..1; es equivalente {0,1} y -3..-7 es equivalente a {-3,-4,-5,-6,-7}. El resultado nos da listas de 2 valores x y 5 valores y. Si no usamos guías de replicación con estas listas no coincidentes, obtenemos una lista de dos puntos, que es la longitud de la lista más corta. Con las guías de replicación, podemos encontrar todas las posibles combinaciones de coordenadas 2 y 5 (o, un **producto cruzado**).
2. Usando la sintaxis `Point.ByCoordinates(x_vals<1>,y_vals<2>);` obtenemos **dos** listas con **cinco** elementos en cada lista.
3. Usando la sintaxis `Point.ByCoordinates(x_vals<2>,y_vals<1>);` obtenemos **cinco** listas con **dos** elementos en cada lista.

Con esta notación, también podemos especificar qué lista será dominante: 2 listas de 5 cosas o 5 listas de 2 cosas. En el ejemplo, cambiar el orden de las guías de replicación hace que el resultado sea una lista de filas de puntos o una lista de columnas de puntos en una grilla.

Nodo para codificar

Si bien los métodos de bloqueo de código anteriores pueden tardar un poco en acostumbrarse, hay una función en Dynamo llamada "Node to Code" que facilitará el proceso. Para usar esta función, seleccione una matriz de nodos en su gráfico Dynamo, haga clic derecho en el lienzo y

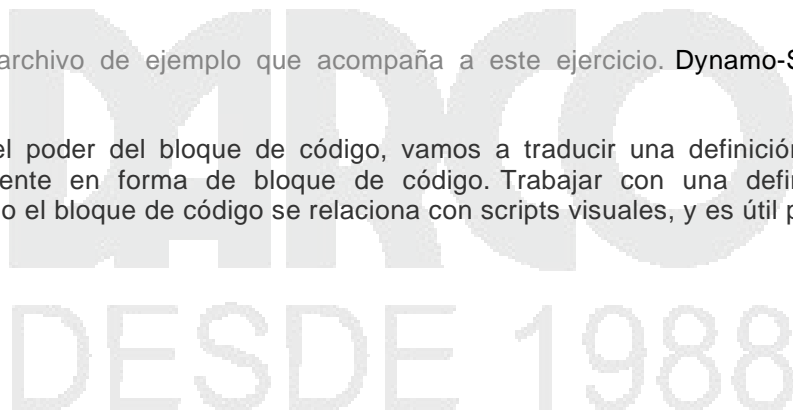
seleccione "Nodo para codificar". ¡Dynamo condensa estos nodos en un bloque de código, con todas las entradas y salidas! No solo es una gran herramienta para aprender a bloquear códigos, sino que también le permite trabajar con un gráfico Dynamo más eficiente y paramétrico. Concluiremos el siguiente ejercicio utilizando "Node to Code", así que no se lo pierda.



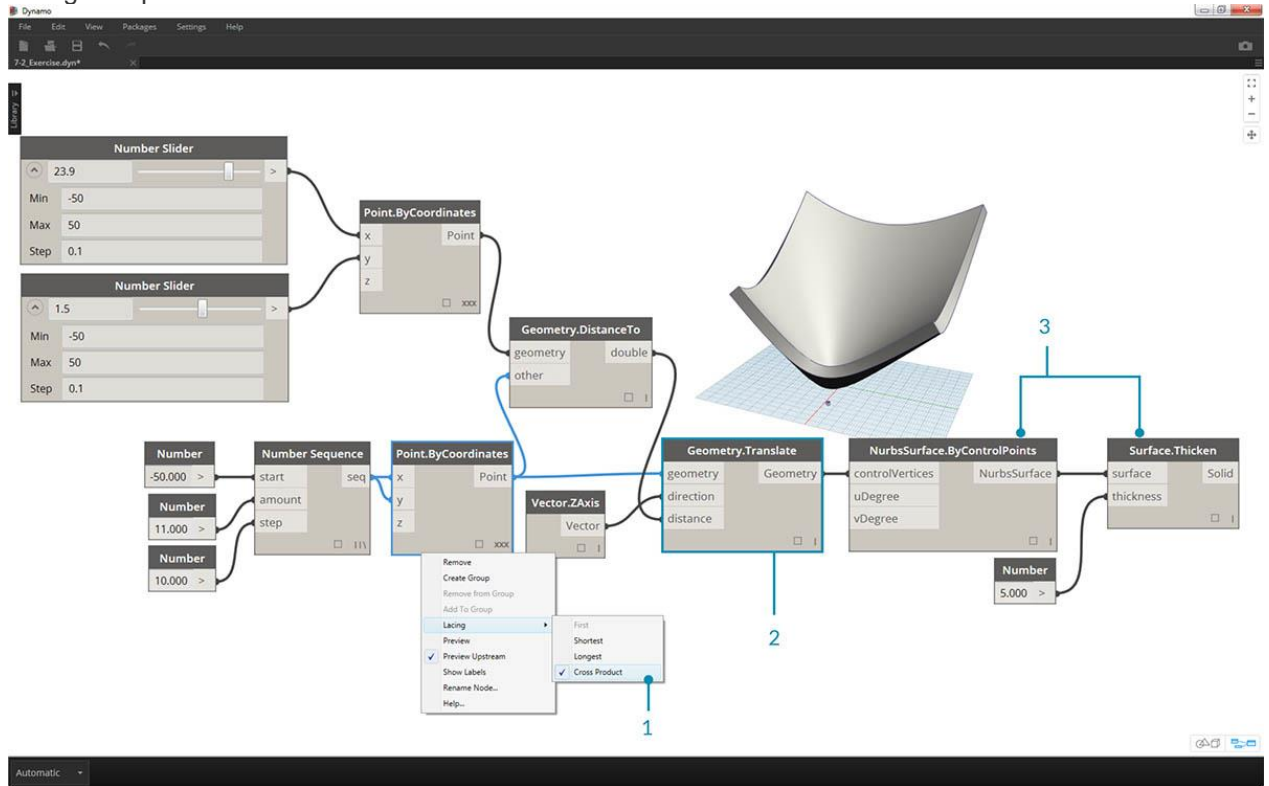
Ejercicio

Descargue el archivo de ejemplo que acompaña a este ejercicio. [Dynamo-Syntax_Atractor-Surface.dyn](#)

Para mostrar el poder del bloque de código, vamos a traducir una definición de campo de atracción existente en forma de bloque de código. Trabajar con una definición existente demuestra cómo el bloque de código se relaciona con scripts visuales, y es útil para aprender la sintaxis de



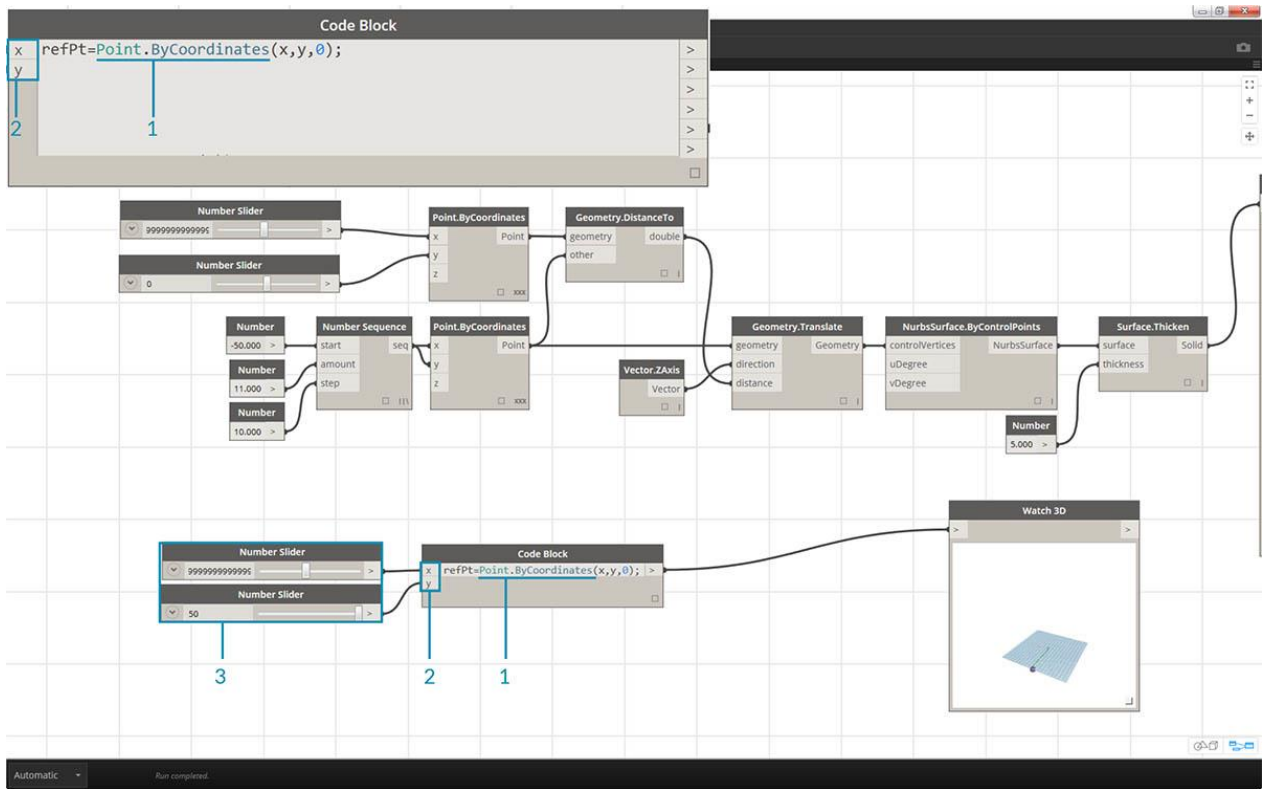
DesignScript.



Comience por recrear la definición en la imagen de arriba (o abriendo el archivo de muestra).

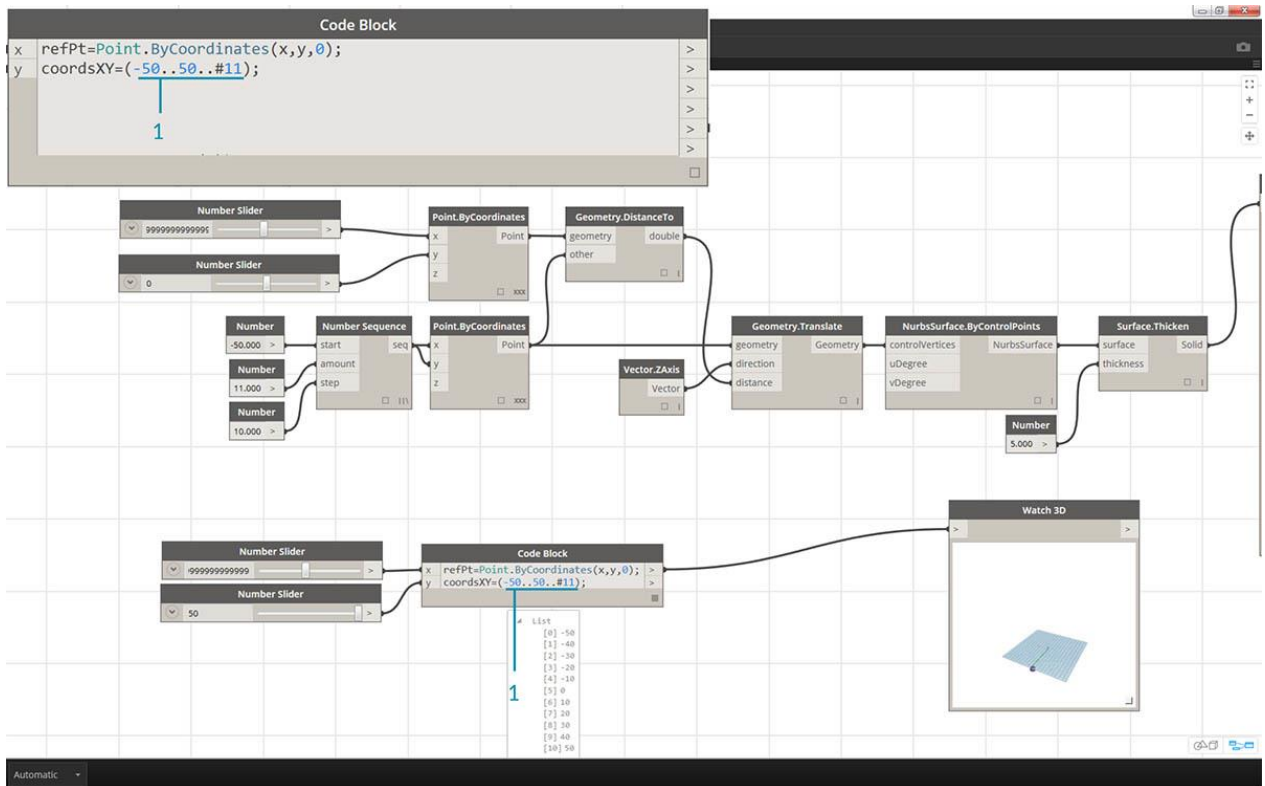
1. Observe que el cordón en *Point.ByCoordinates* se ha establecido en *Cross Product*.
2. Cada punto de una cuadrícula se mueve hacia arriba en la dirección Z en función de su distancia al punto de referencia.
3. Una superficie se recrea y engrosa, creando un abultamiento en la geometría relativa a la distancia al punto de referencia.

DARCO
DESDE 1988



1. A partir del comienzo, vamos a definir el punto de referencia en primer lugar: `Point.ByCoordinates(x,y,0);`. Usamos la misma sintaxis de `Point.ByCoordinates` que se especifica en la parte superior del nodo del punto de referencia.
2. Las variables `x` e `y` se insertan en el bloque de código para que podamos actualizar dinámicamente éstos con los deslizadores.
3. Agregue algunos *controles deslizantes* a las entradas del *bloque de código* que van desde -50 a 50 . De esta forma, podemos abarcar toda la cuadrícula Dynamo predeterminada.

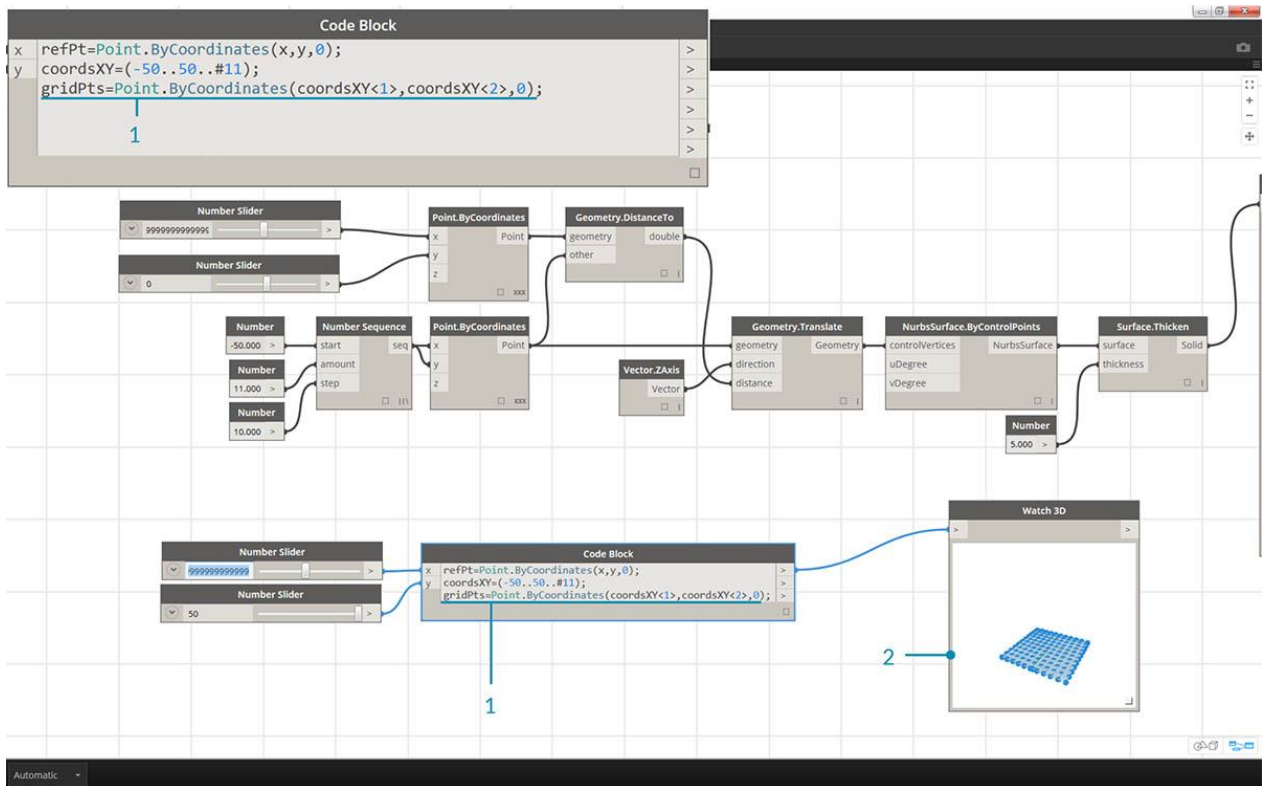
DARCO
DESDE 1988



1. En la segunda línea del *bloque de código*, se define una abreviatura para reemplazar el nodo secuencia de números: `coordsXY = (-50..50..#11);`. Discutiremos esto más en la siguiente sección. Por ahora, observe que esta abreviatura es equivalente al nodo de *Secuencia numérica* en el guión visual.

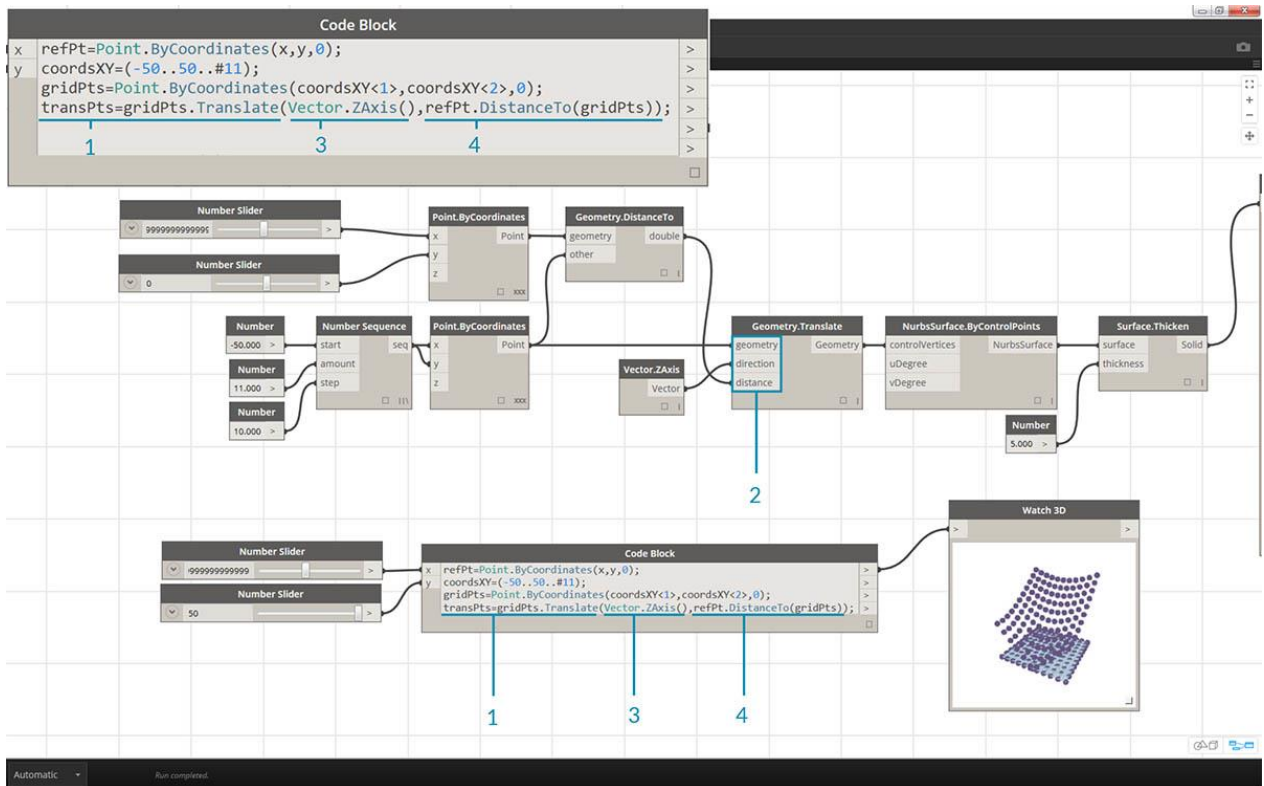
DARCO

DESDE 1988



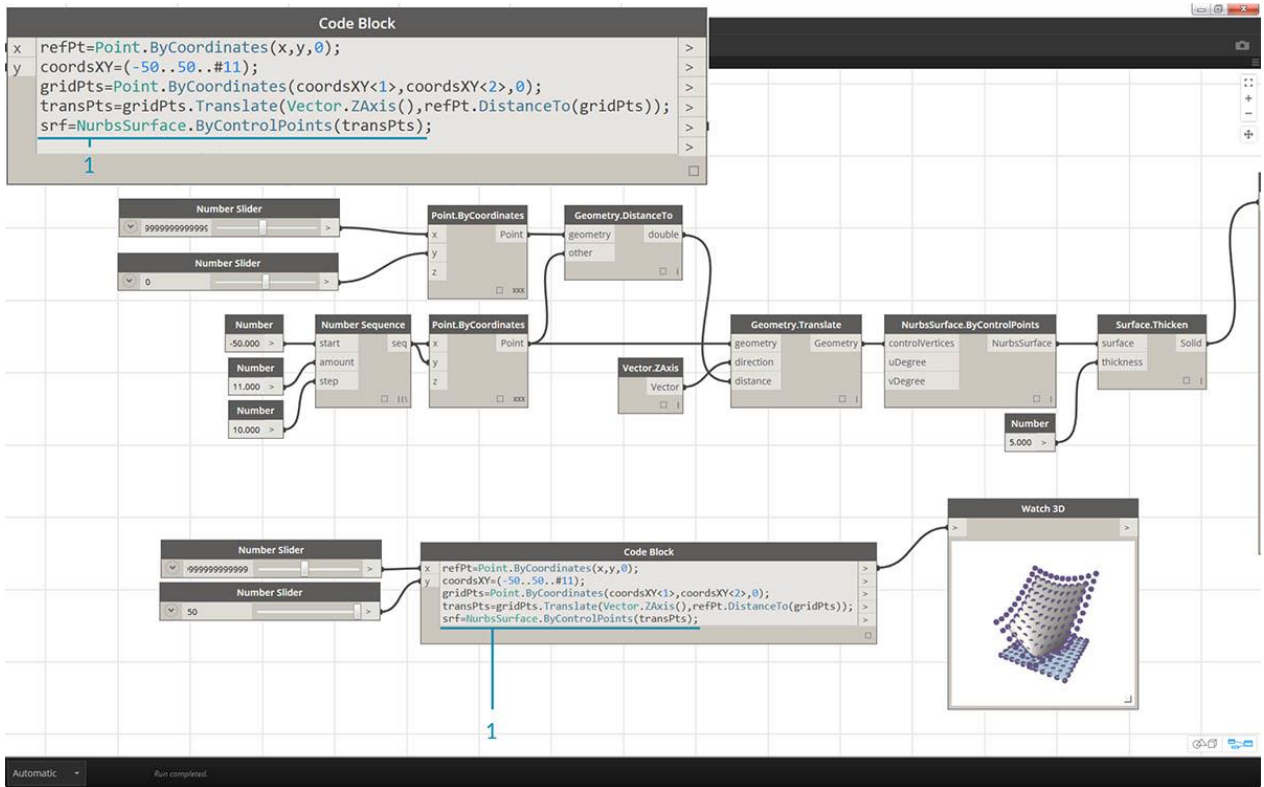
1. Ahora, queremos crear una grilla de puntos a partir de la secuencia *coordsXY*. Para hacer esto, queremos usar la sintaxis *Point.ByCoordinates*, pero también necesitamos iniciar un *producto cruzado* de la lista de la misma manera que lo hicimos en el script visual. Para ello, escriba la línea: `gridPts = Point.ByCoordinates(coordsXY<1>,coordsXY<2>,0);`. Los corchetes en ángulo denotan la referencia cruzada del producto.
2. Observe en el nodo *Watch3D* que tenemos una grilla de puntos en la grilla Dynamo.

DARCO
DESDE 1988

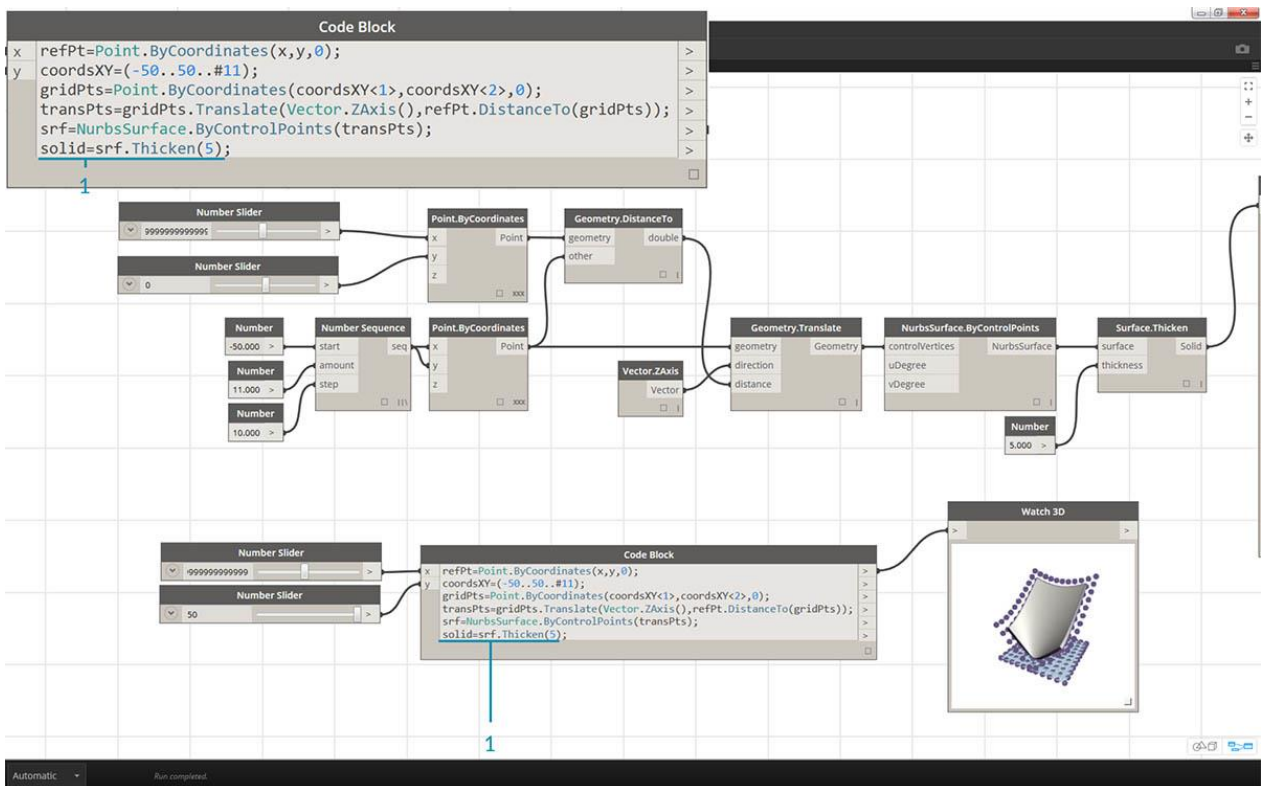


1. Ahora, para la parte difícil: queremos mover la grilla de puntos hacia arriba en función de su distancia al punto de referencia. Primero, llamemos a este nuevo conjunto de puntos *transPts*. Y dado que una traducción es una acción sobre un elemento existente, en lugar de usar *Geometry.Translate...*, lo usamos *gridPts.Translate*.
2. Leyendo desde el nodo real en el lienzo, vemos que hay tres entradas. La geometría para traducir ya está declarada porque estamos realizando la acción en ese elemento (con *gridPts.Translate*). Las dos entradas restantes se insertarán en el paréntesis de la función: *dirección* y *distancia*.
3. La dirección es bastante simple, usamos a *Vector.ZAxis()* para moverse verticalmente.
4. La distancia entre el punto de referencia y cada punto de la rejilla aún debe calcularse, por lo que hacemos esto como una acción para el punto de referencia de la misma manera: *refPt.DistanceTo(gridPts)*
5. La última línea de código nos da los puntos traducidos: *transPts = gridPts.Translate(Vector.ZAxis(),refPt.DistanceTo(gridPts));*

DESDE 1988



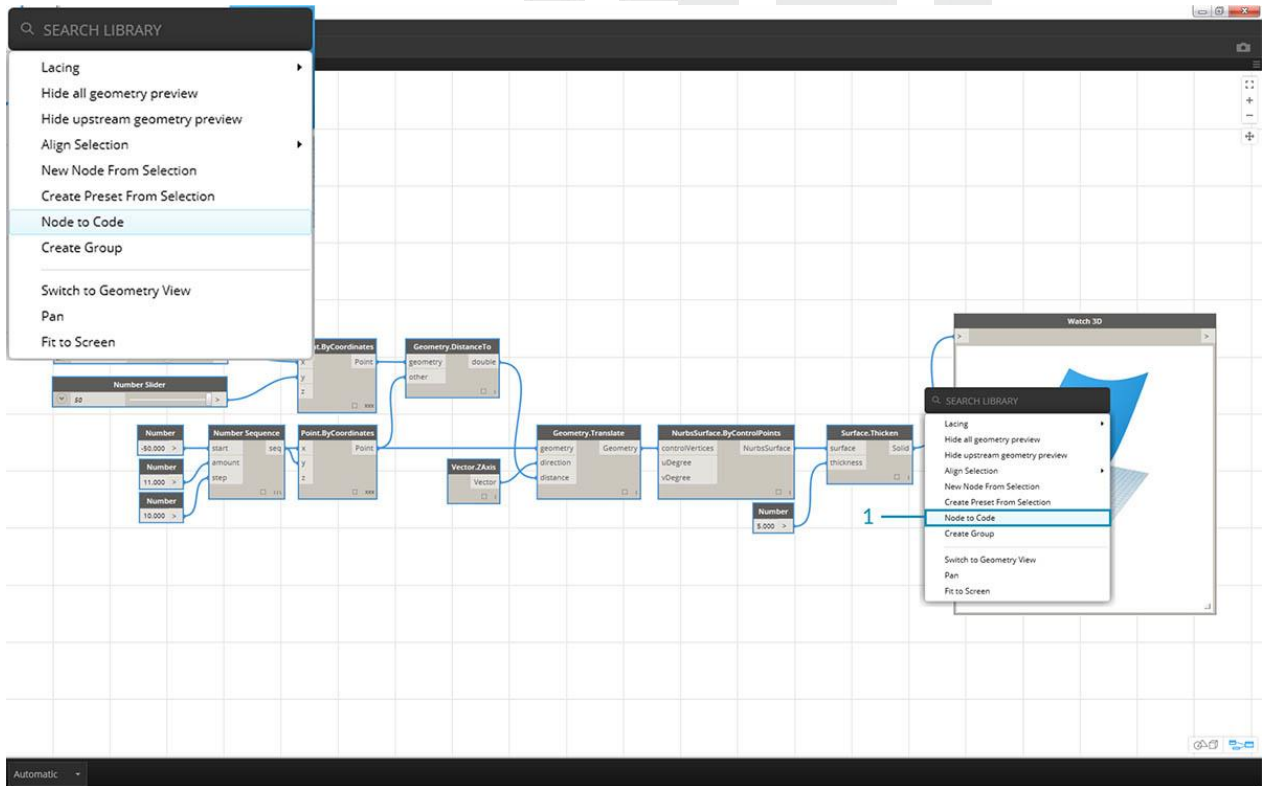
1. Ahora tenemos una grilla de puntos con la estructura de datos apropiada para crear una superficie Nurbs. Construimos la superficie usando `srf = NurbsSurface.ByControlPoints(transPts);`



- Y finalmente, para agregar algo de profundidad a la superficie, construimos un sólido usando `solid = srf.Thicken(5)`; En este caso, engrosamos la superficie en 5 unidades en el código, pero siempre podemos declarar esto como una variable (llamándolo *grosor*, por ejemplo) y luego controlar eso valor con un control deslizante.

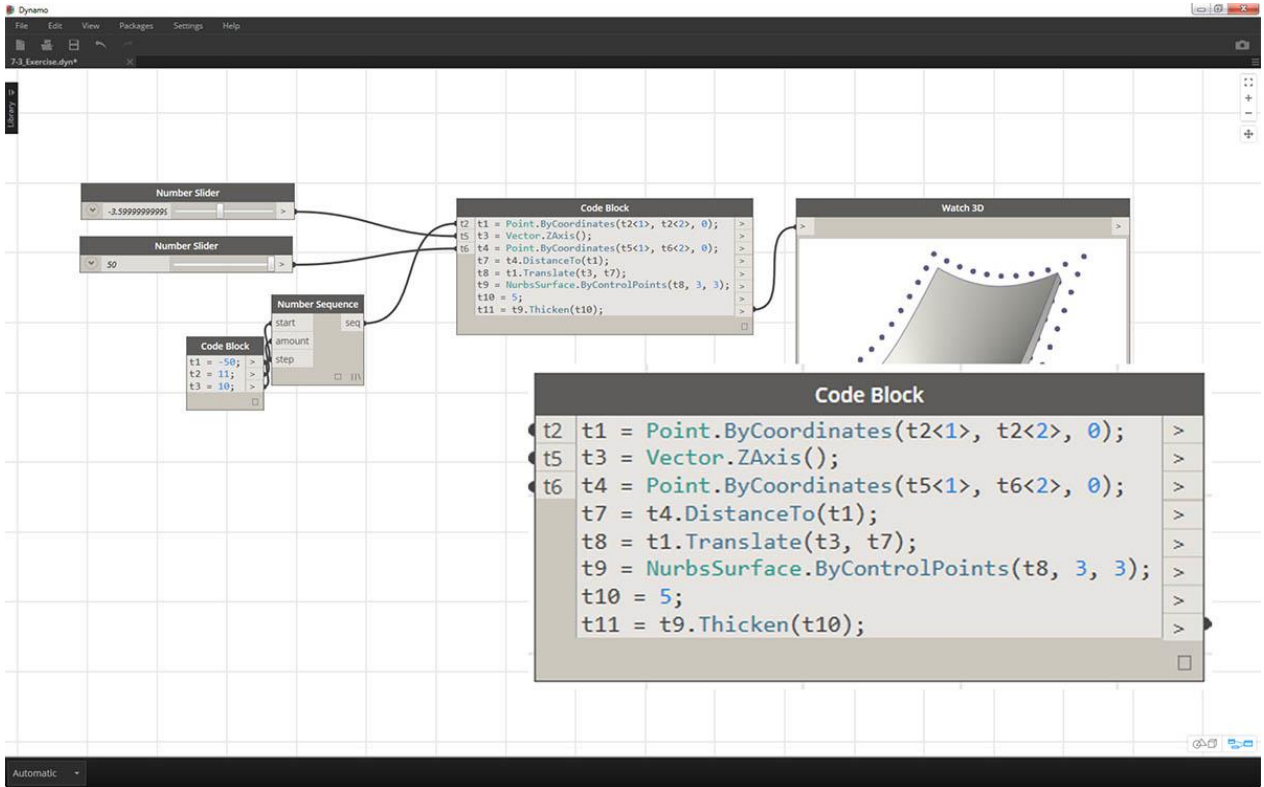
Simplifica el gráfico con "Nodo al código"

La función "Nodo a código" automatiza todo el ejercicio que acabamos de completar con solo hacer clic en un botón. Esto no solo es poderoso para crear definiciones personalizadas y bloques de código reutilizables, sino que también es una herramienta realmente útil para aprender a programar en Dynamo:



- Comience con la secuencia de comandos visual existente desde el paso 1 del ejercicio. Seleccione todos los nodos, haga clic derecho en el lienzo y seleccione "Nodo para codificar". Simple como eso.

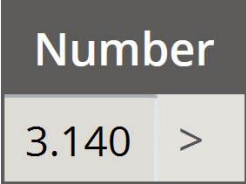
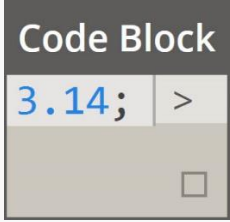
DESDE 1988

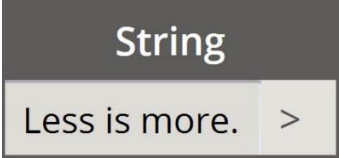

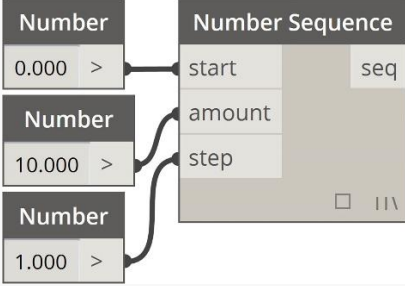
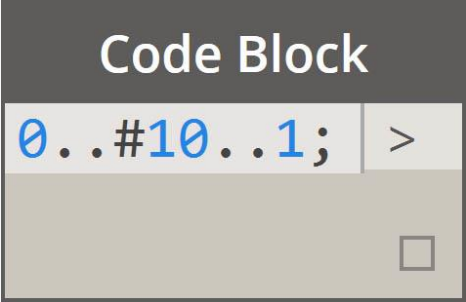
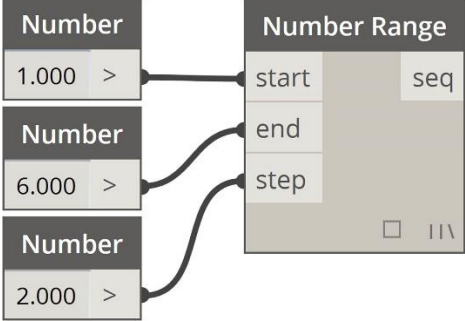
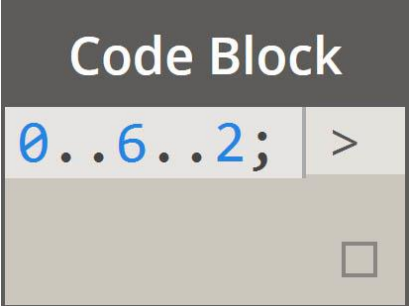
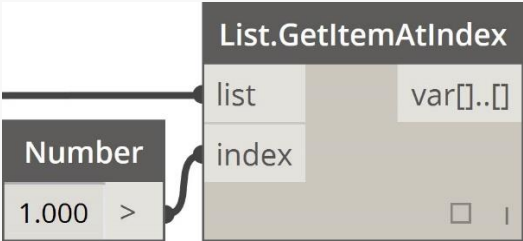
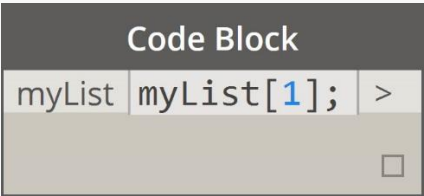
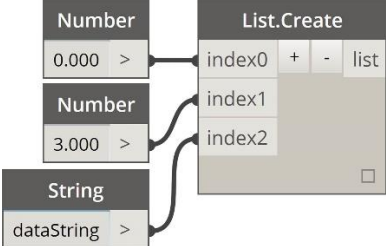
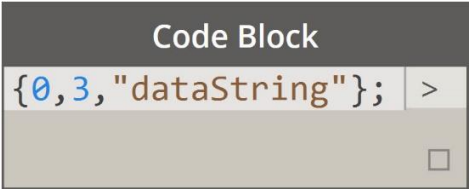


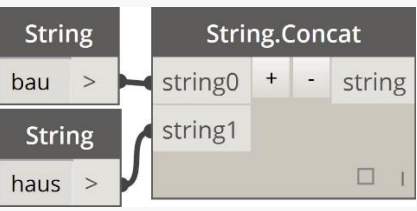
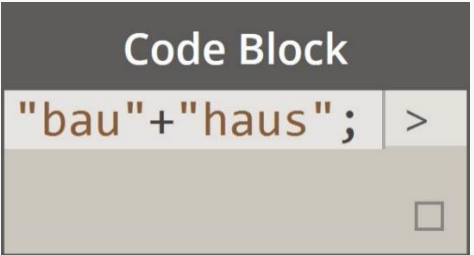
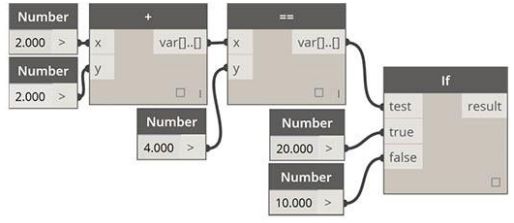
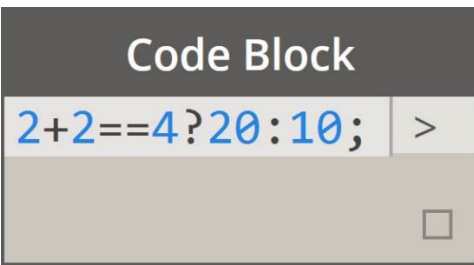
Dynamo ha automatizado una versión basada en texto del gráfico visual, cordones y todo. ¡Pruebe esto en sus scripts visuales y libere la potencia del bloque de códigos!

Taquigrafía

Hay algunos métodos básicos de taquigrafía en el bloque de código que, en pocas palabras, hacen que la administración de datos sea *mucho* más fácil. Desglosaremos los conceptos básicos a continuación y analizaremos cómo se puede usar esta abreviatura para crear y consultar datos.

Tipo de datos	Dynamo estándar	Código Bloque Equivalent
Números		

<p>Instrumentos de cuerda</p>		
<p>Secuencias</p>		
<p>Rangos</p>		
<p>Obtener artículo en el índice</p>		
<p>Crear lista</p>		

Cadenas de concatenados		
Declaraciones condicionales		

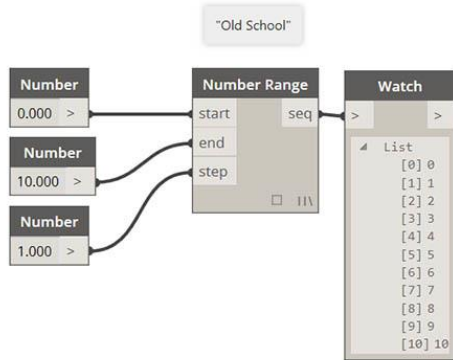
Sintaxis adicional

Nodo (s)	Equivalente de bloque de código	Nota
Cualquier operador (+, &&, >=, Not, etc.)	+, &&, >=, !, etc.	Tenga en cuenta que "No" se convierte en "!" Pero el nodo se llama "No" para distinguirlo de "Factorial"
Boolean True	cierto;	Nota minúscula
Booleano falso	falso;	Nota minúscula

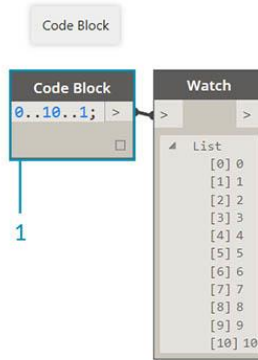
Rangos

El método para definir rangos y secuencias se puede reducir a taquigrafía básica. Utilice la imagen siguiente como guía para la sintaxis ".." para definir una lista de datos numéricos con bloque de código. Después de familiarizarse con esta notación, crear datos numéricos es un

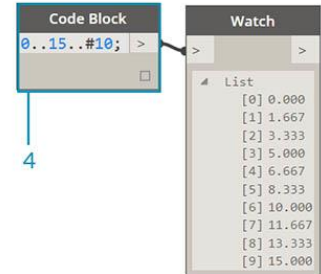
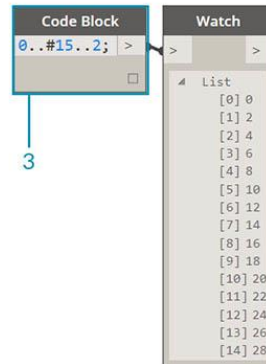
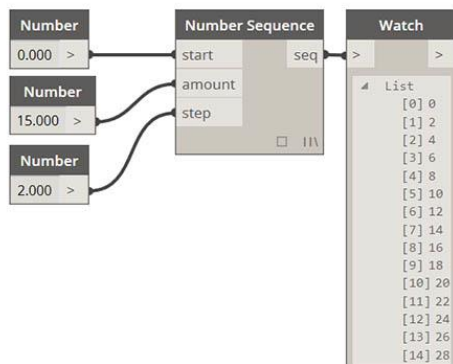
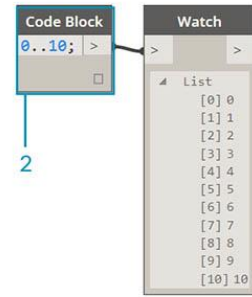
proceso



realmente



eficiente:

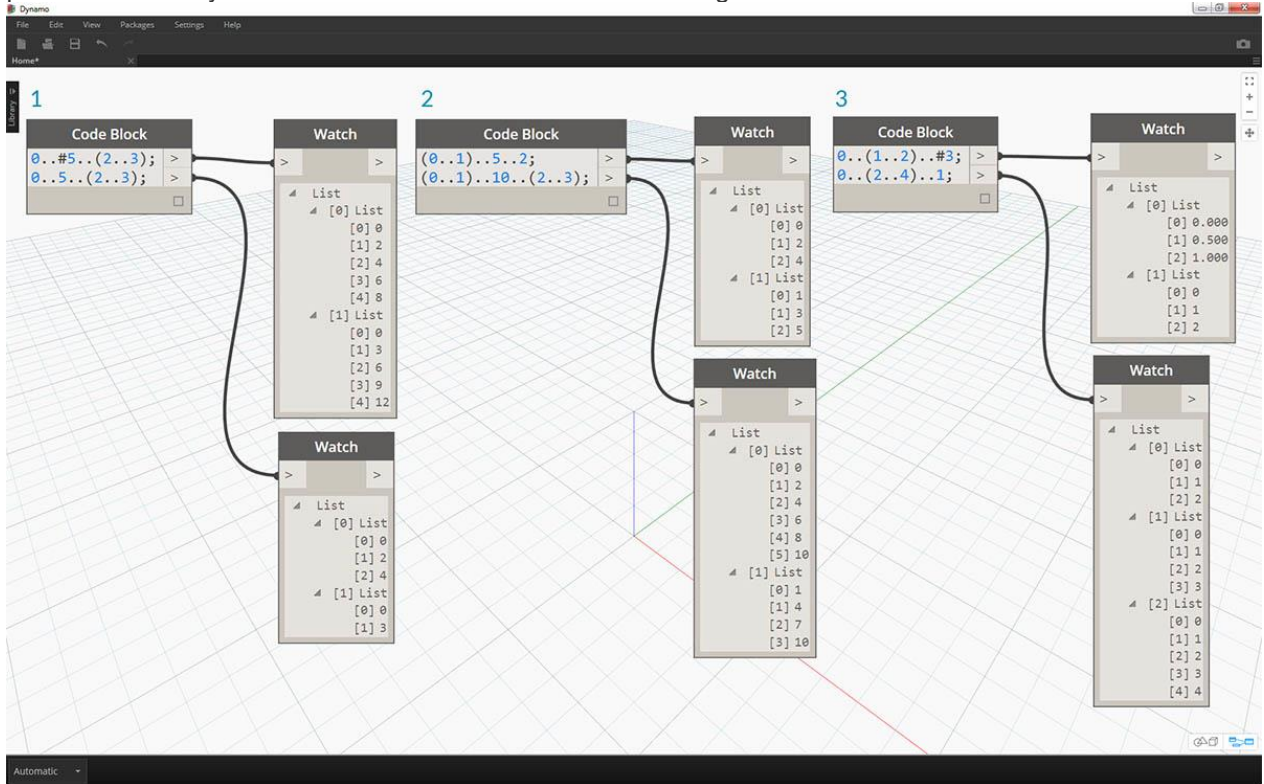


1. En este ejemplo, un rango de números se reemplaza por una sintaxis de bloque de código básico que define el beginning..end..step-size;. Representados numéricamente, obtenemos:0..10..1;
2. Tenga en cuenta que la sintaxis 0..10..1;es equivalente a 0..10;. Un tamaño de paso de 1 es el valor predeterminado para la notación abreviada. Entonces 0..10;dará una secuencia de 0 a 10 con un tamaño de paso de 1.
3. La *secuencia de números* ejemplo es similar, excepto que utilizamos un "#" para indicar que queremos 15 valores en la lista, en lugar de una lista que sube a 15. En este caso, estamos definiendo: beginning..#ofSteps..step-size;. La sintaxis real para la secuencia es0..#15..2
4. Usando el "#" del paso anterior, ahora lo colocamos en la porción de "paso de tamaño" de la sintaxis. Ahora, tenemos un *rango de números que abarca desde el "principio" hasta el "final"* y la notación "tamaño de paso" distribuye uniformemente una cantidad de valores entre los dos:beginning..end..#ofSteps

Rangos avanzados

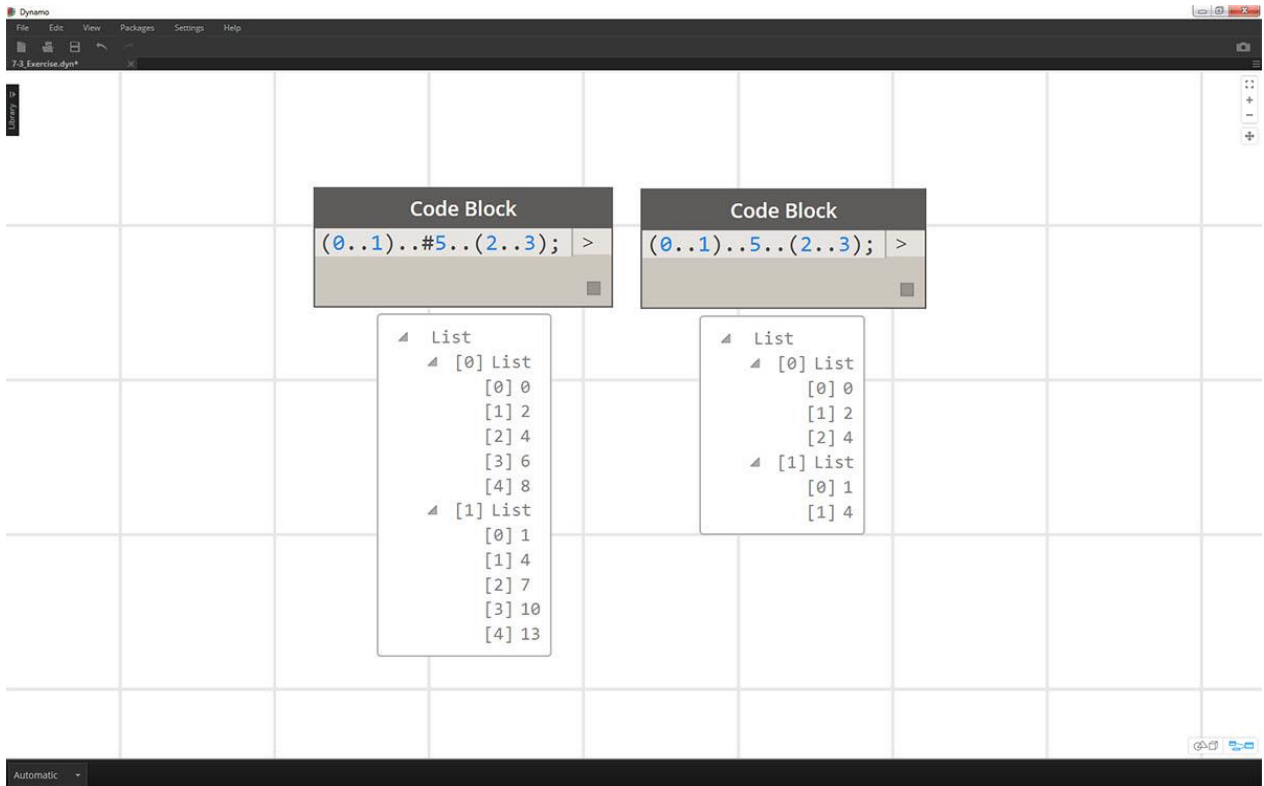
La creación de rangos avanzados nos permite trabajar con la lista de listas de una manera simple. En los ejemplos a continuación, estamos aislando una variable de la notación de rango

primario y creando otro rango de esa lista.



1. Creando rangos anidados, compara la notación con un "#" vs. la notación sin. Se aplica la misma lógica que en los rangos básicos, excepto que se vuelve un poco más complejo.
2. Podemos definir un sub-rango en cualquier lugar dentro del rango primario, y notamos que también podemos tener dos sub-rangos.
3. Al controlar el valor "final" en un rango, creamos más rangos de diferentes longitudes.

DARCO
DESDE 1988

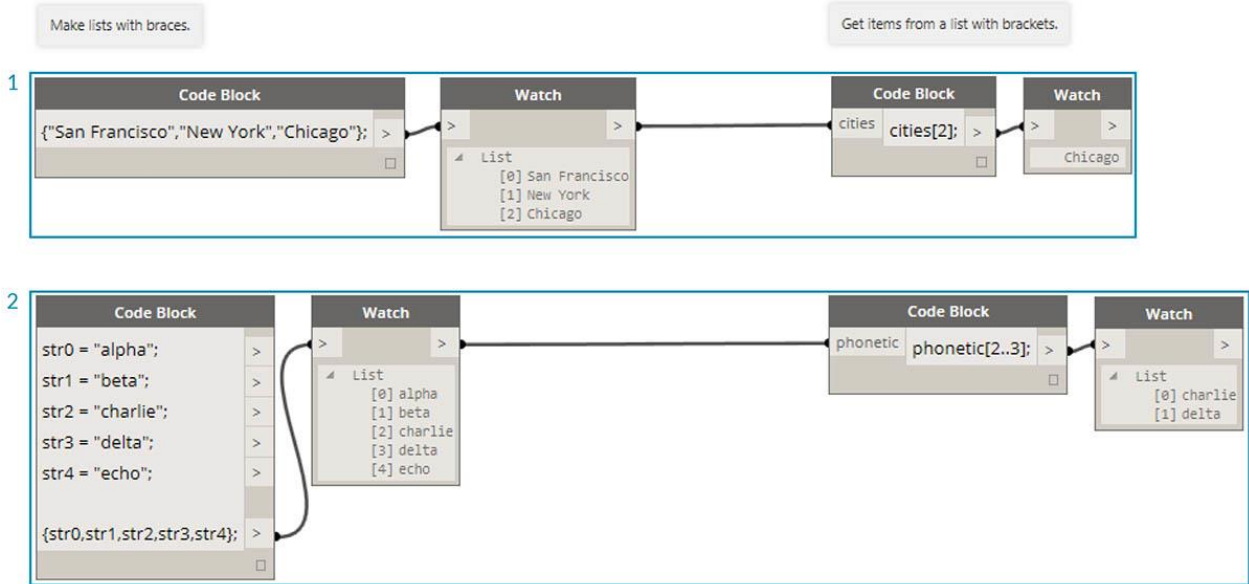


Como ejercicio lógico, compare los dos atajos anteriores e intente analizar a través de cómo los *subintervalos* y la notación "#" manejan la salida resultante.

Haga listas y obtenga artículos de una lista

Además de hacer listas con taquigrafía, también podemos crear listas sobre la marcha. Esta lista puede contener una amplia gama de tipos de elementos y también se puede consultar (recuerde, las listas son objetos en sí mismos). Para resumir, con el bloque de código haces listas con llaves (también conocidas como "llaves") y consultas elementos de una lista con corchetes (también conocidos como "corchetes"):

DARCO
DESDE 1988



1. Crea listas rápidamente con cadenas y consúltalas usando el índice de elementos.
2. Crea listas con variables y consulta utilizando la notación abreviada de rango.

Y gestionar con listas anidadas es un proceso similar. Tenga en cuenta el orden de la lista y recuerde usar múltiples conjuntos de corchetes:

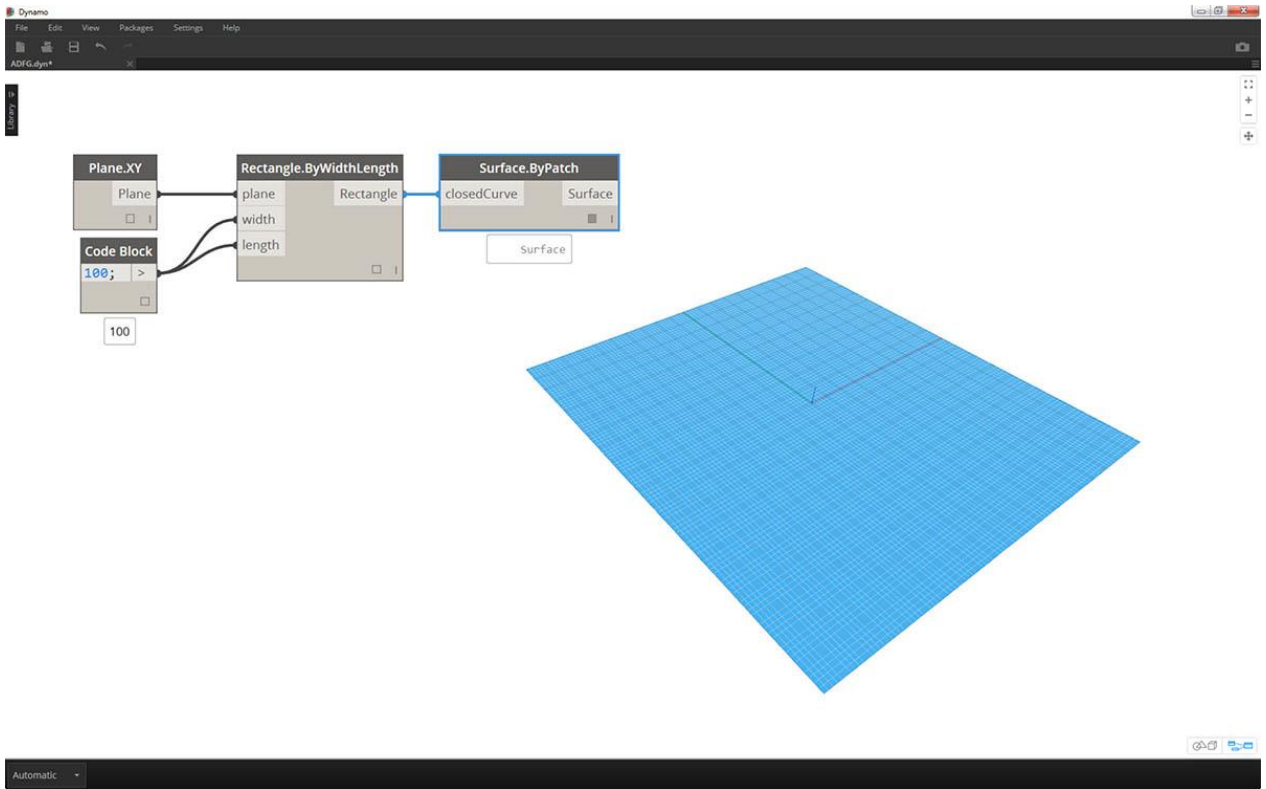


1. Definir una lista de listas
2. Consulta una lista con notación de corchetes individuales.
3. Consulta un elemento con notación de doble corchete.

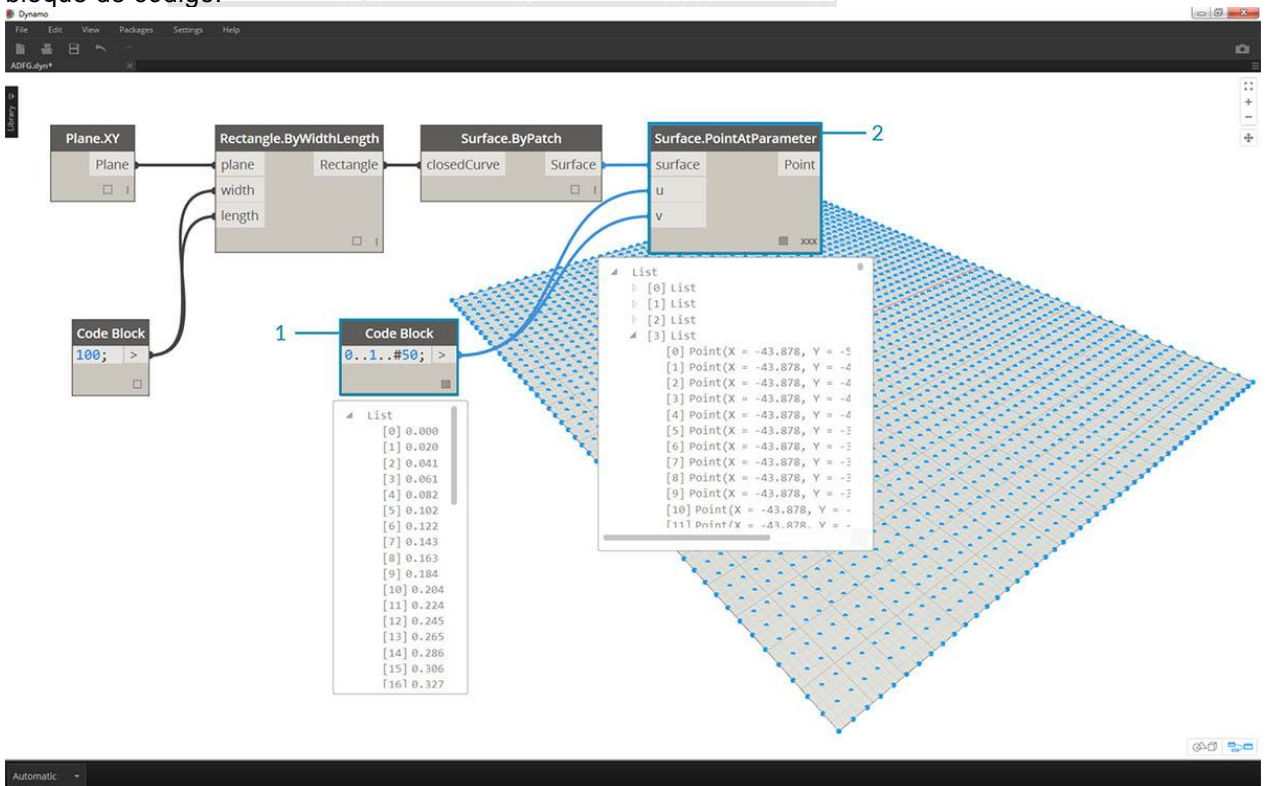
Ejercicio

Descargue el archivo de ejemplo que acompaña a este ejercicio [Obsolete-Nodes_Sine-Surface.dyn](#)

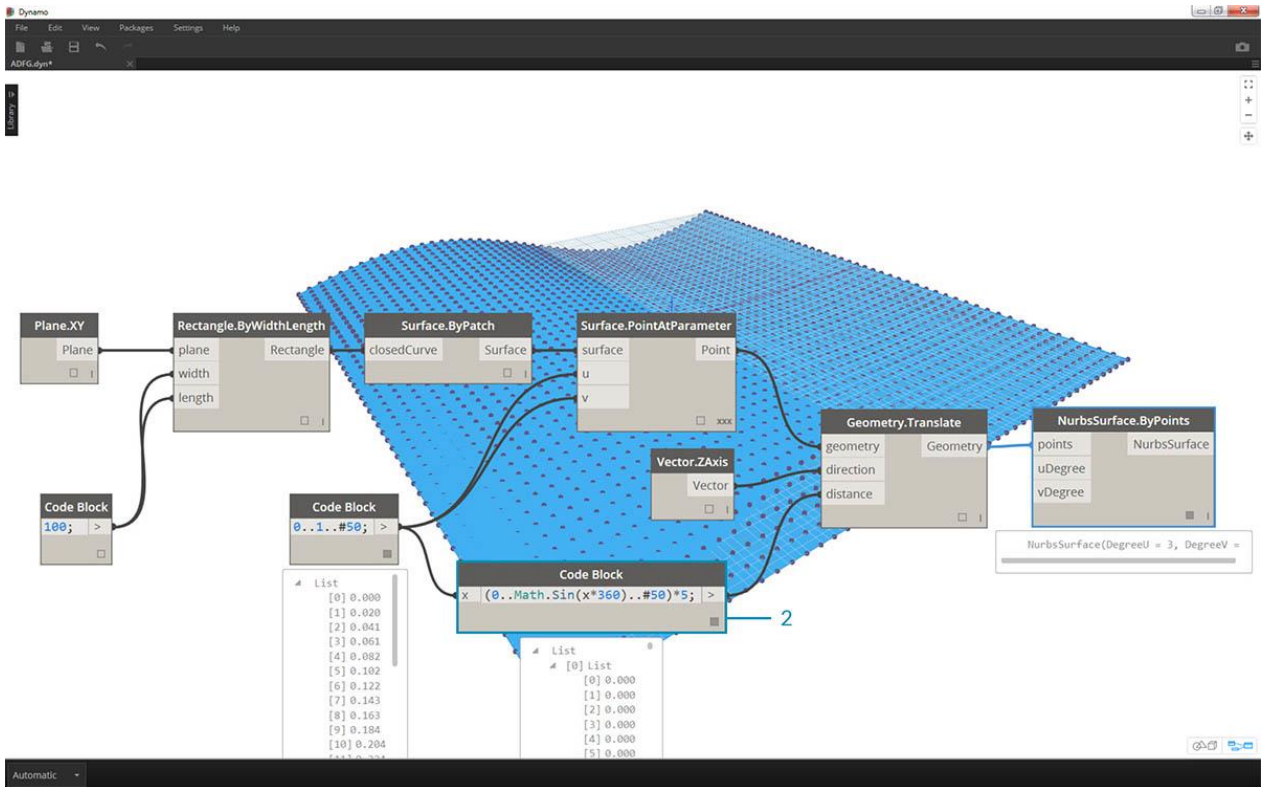
En este ejercicio, flexibilizaremos nuestras nuevas habilidades taquigráficas para crear una superficie de cáscara de huevo funky-cool definida por rangos y fórmulas. Durante este ejercicio, observe cómo usamos el bloque de código y los nodos de Dynamo existentes en tándem: usamos el bloque de código para la elevación de datos pesados mientras que los nodos de Dynamo se presentan visualmente para la legibilidad de la definición.



Comienza creando una superficie conectando los nodos de arriba. En lugar de usar un nodo numérico para definir el ancho y la longitud, haga doble clic en el lienzo y escriba 100; en un bloque de código.

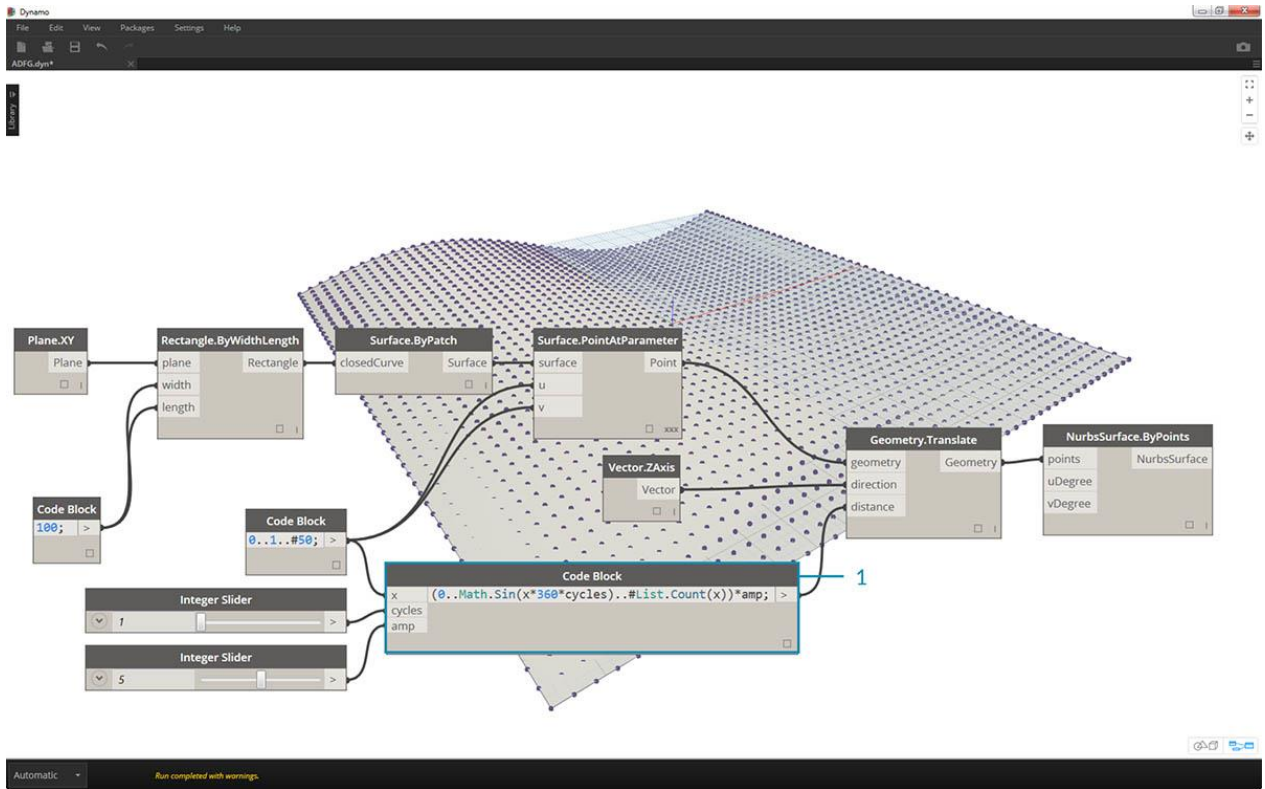


1. Defina un rango entre 0 y 1 con 50 divisiones escribiendo `0..1..#50` en un bloque de código.
2. Conectar el rango en `Surface.PointAtParameter`, que toma u y v valores entre 0 y 1 a través de la superficie. Recuerde cambiar el `Lacing to Cross Product` haciendo clic derecho en el nodo `Surface.PointAtParameter`.



En este paso, empleamos nuestra primera función para mover la cuadrícula de puntos hacia arriba en la Z. Esta cuadrícula generará una superficie generada basada en la función subyacente.

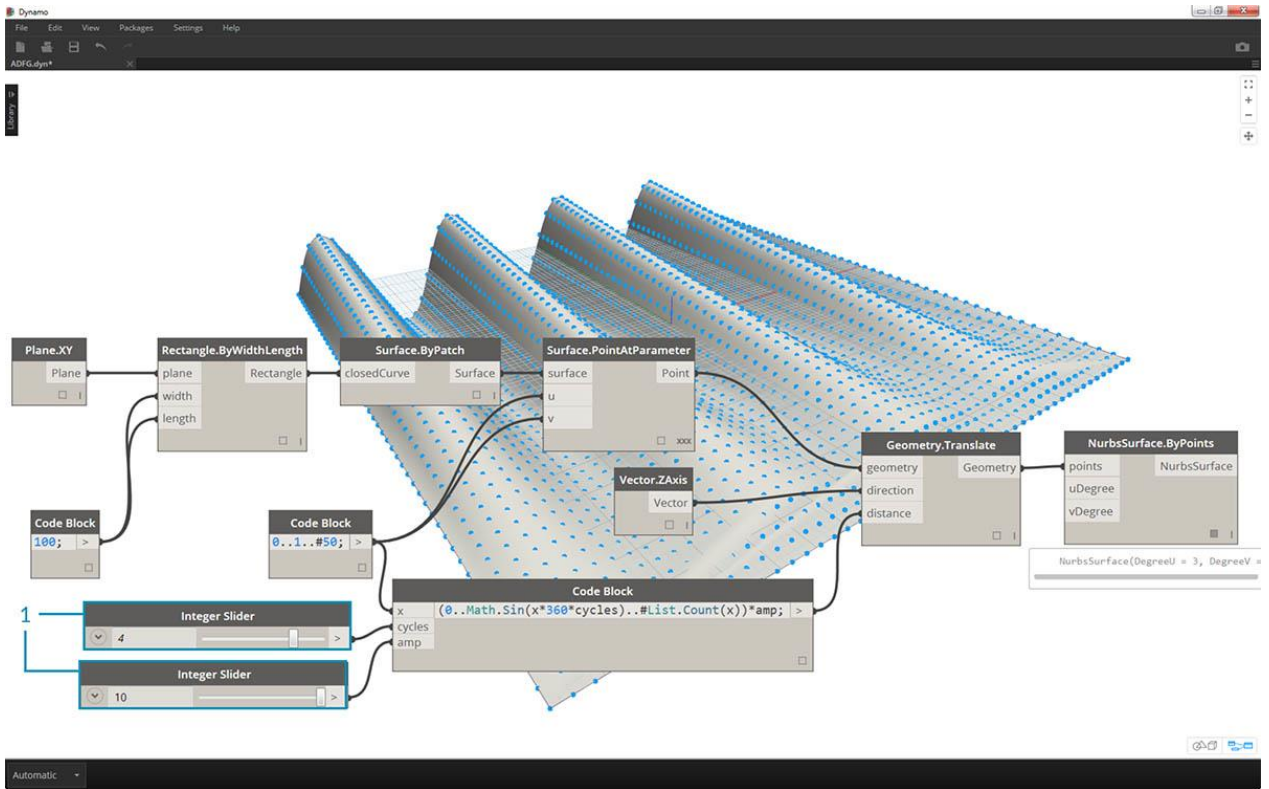
1. Agregue los nodos visuales al lienzo como se muestra en la imagen de arriba.
2. En lugar de utilizar un nodo fórmula, se utiliza un bloque de código con la línea: `(0..Math.Sin(x*360)..#50)*5;`. Para romper esto rápidamente, estamos definiendo un rango con una fórmula dentro de él. Esta fórmula es la función seno. La función sinusoidal recibe entradas de grados en Dynamo, por lo que para obtener una onda sinusoidal completa, multiplicamos nuestros valores x (esta es la entrada de rango de 0 a 1) por 360. A continuación, queremos el mismo número de divisiones que los puntos de la cuadrícula de control para cada fila, por lo que definimos cincuenta subdivisiones con `# 50`. Finalmente, el multiplicador de 5 simplemente aumenta la amplitud de la traducción para que podamos ver el efecto en la vista previa de Dynamo.



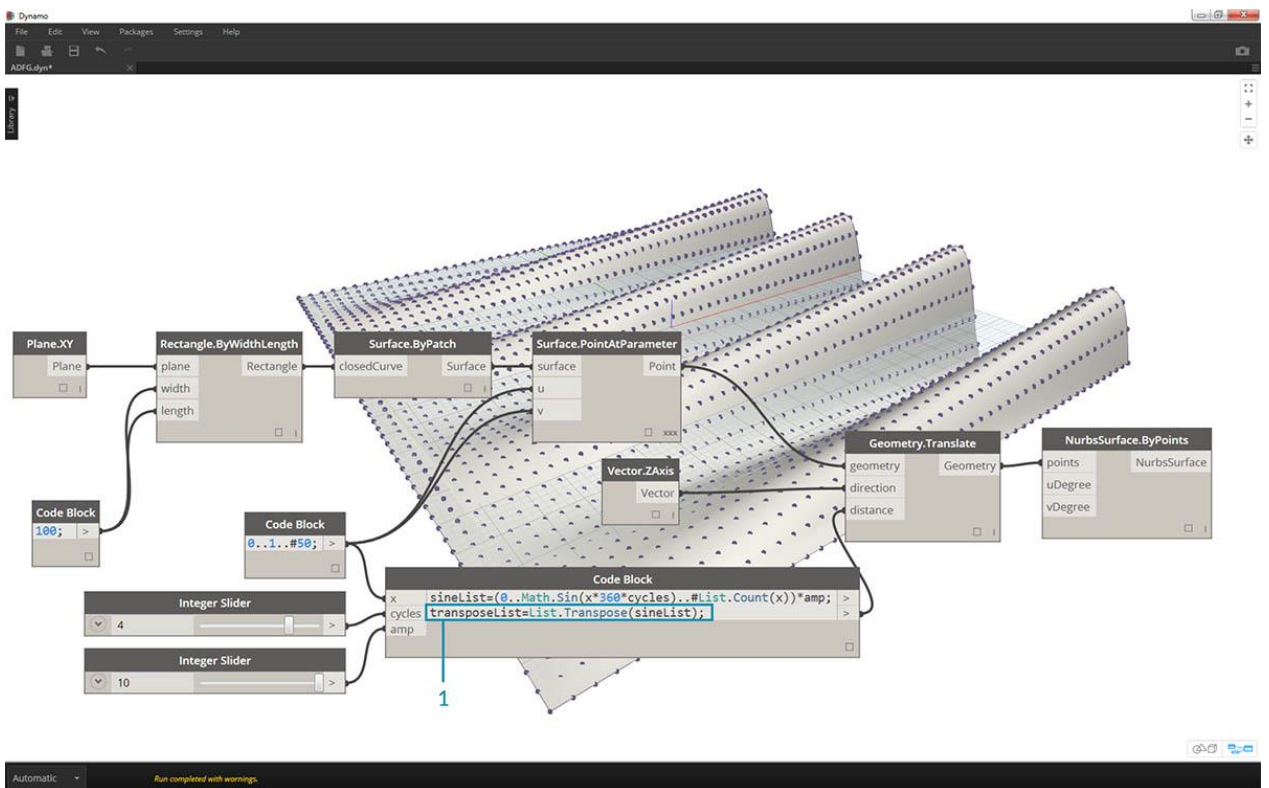
1. Si bien el bloque de código anterior funcionaba bien, no era completamente paramétrico. Queremos impulsar dinámicamente sus parámetros, por lo que reemplazaremos la línea del paso anterior $(0..Math.Sin(x*360*cycles)..#List.Count(x))*amp;$. Esto nos da la capacidad de definir estos valores en función de las entradas.

DARCO

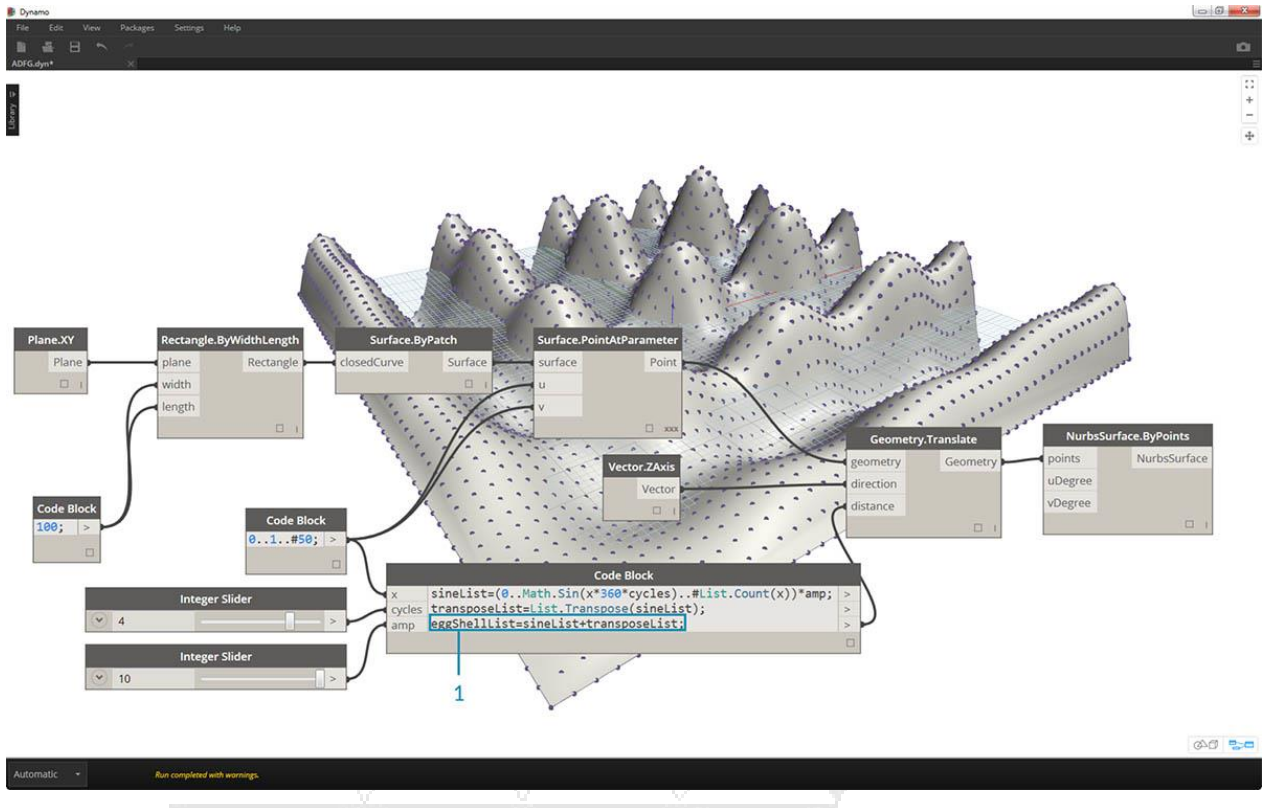
DESDE 1988



1. Al cambiar los controles deslizantes (que van de 0 a 10), obtenemos algunos resultados interesantes.

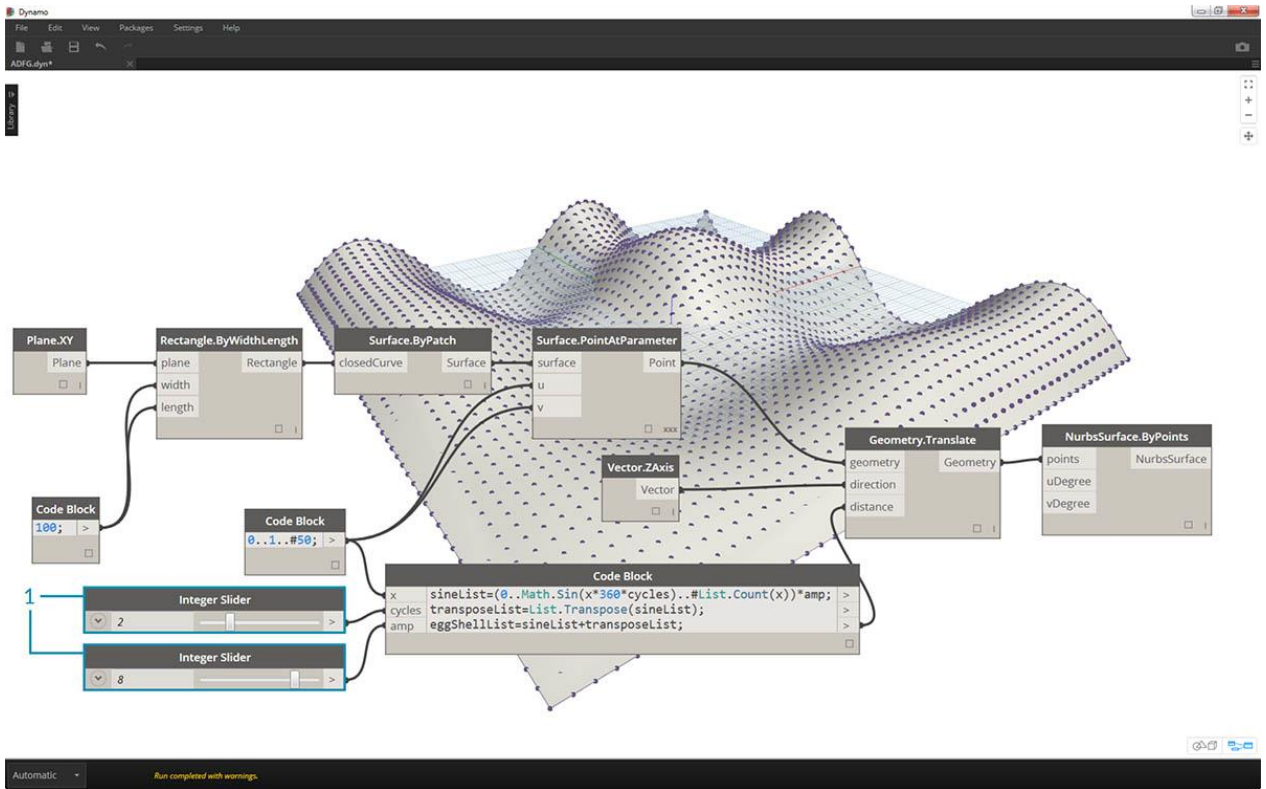


- Al hacer una transposición en el rango numérico, invertimos la dirección de la onda de la cortina: `transposeList = List.Transpose(sineList);`

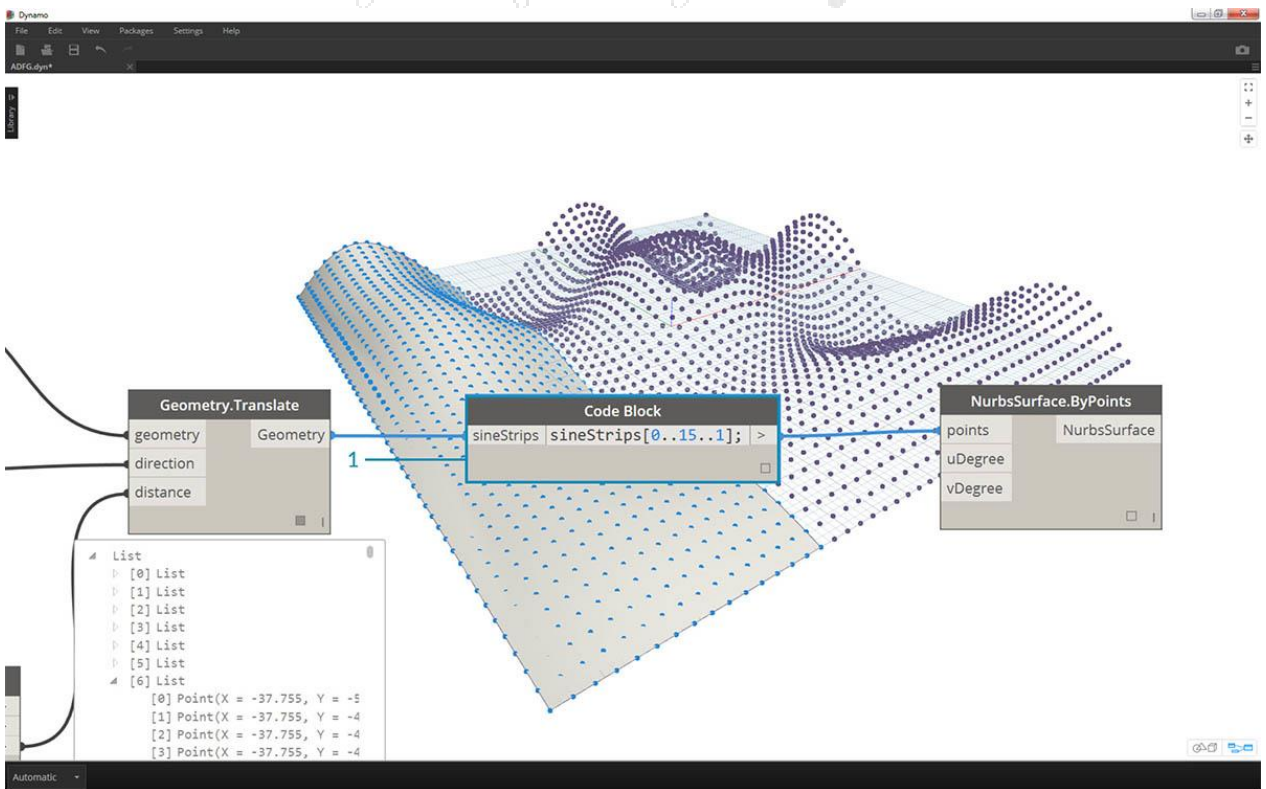


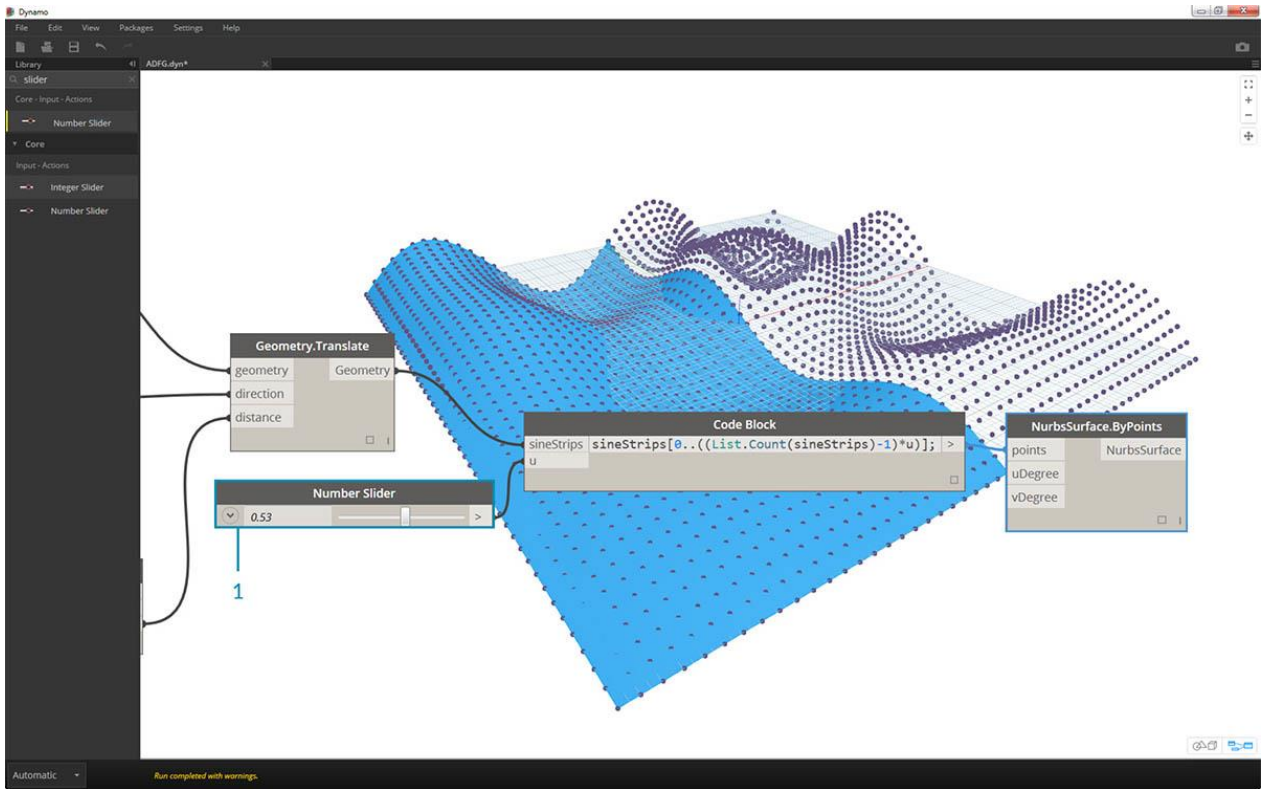
- Obtenemos una superficie de cáscara de huevo distorsionada cuando agregamos sineList y la lista de transposición: `eggShellList = sineList+transposeList;`

DARCO
DESDE 1988

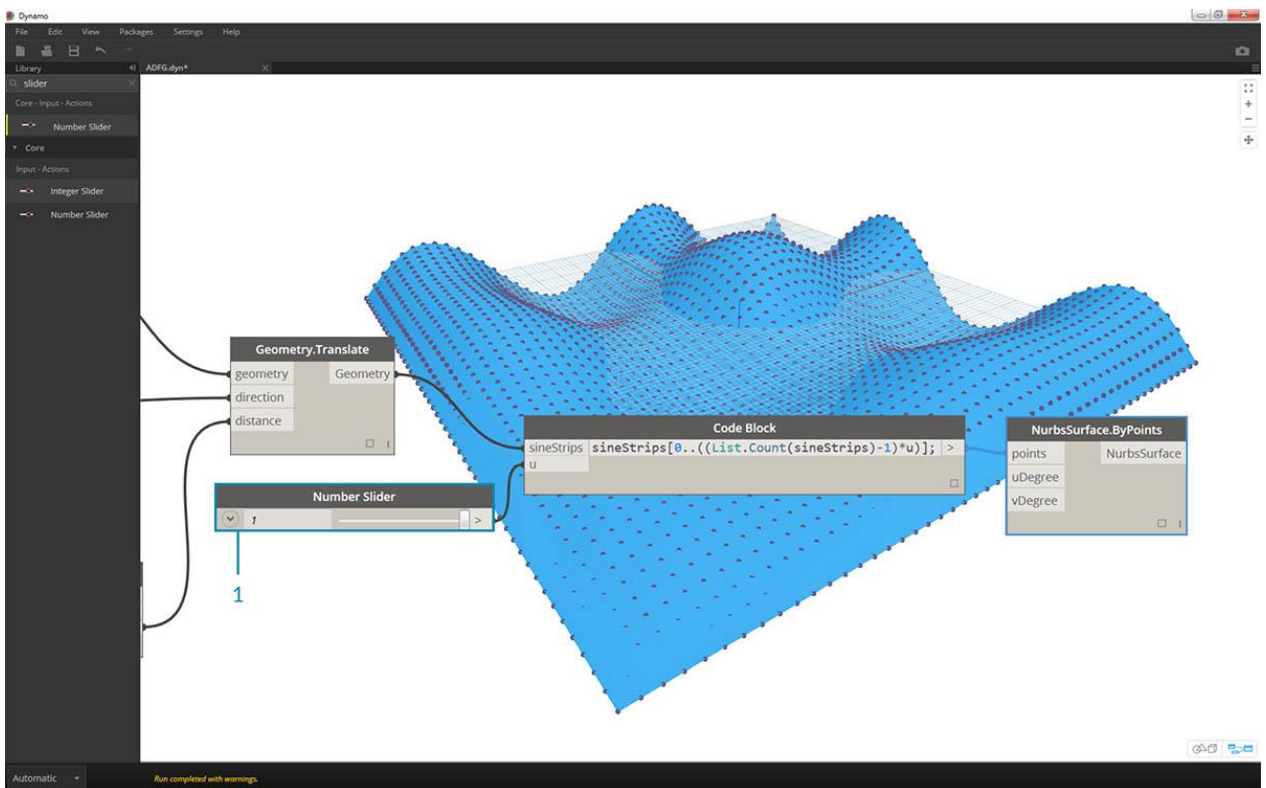


1. Cambiando los controles deslizantes de nuevo, vamos a calmar las aguas de este algoritmo.

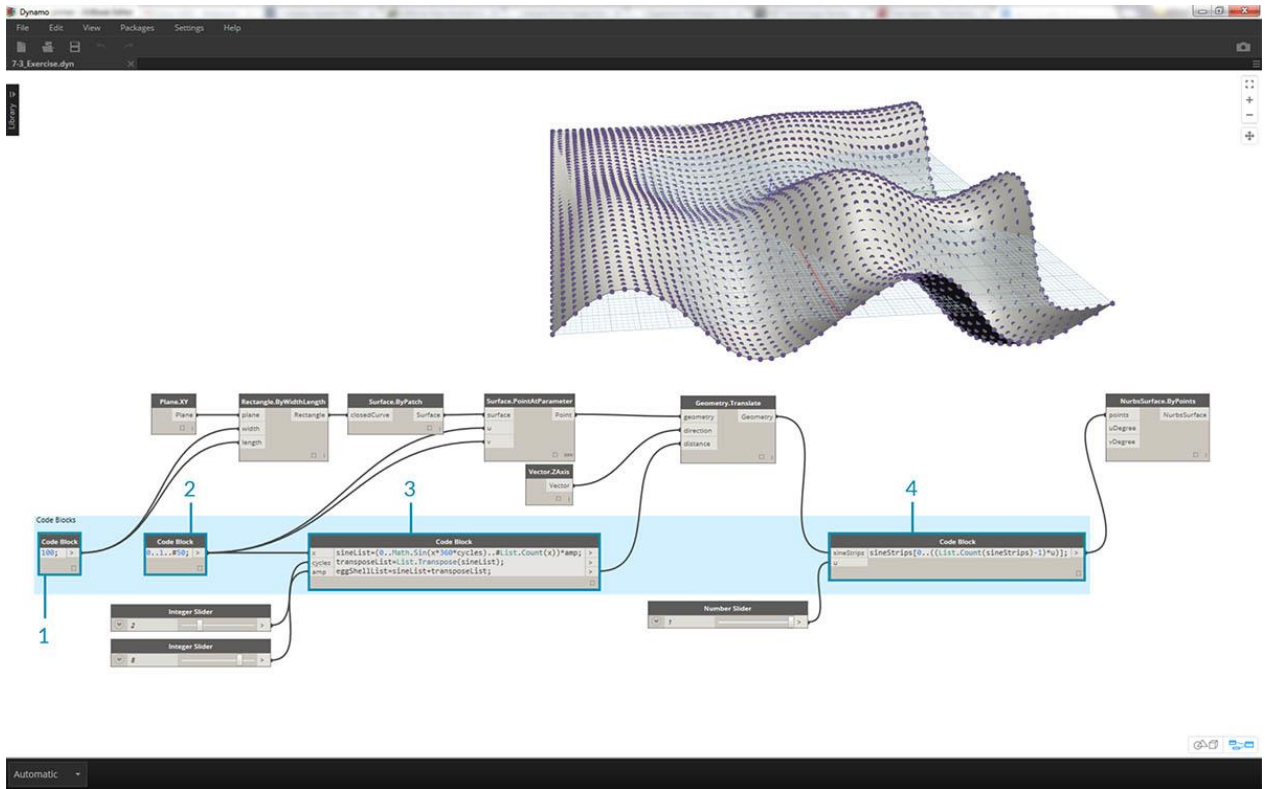




1. Un valor de .53 en el control deslizante crea una superficie justo después del punto medio de la cuadrícula.



1. Y como se esperaba, un control deslizante de 1 crea una superficie a partir de la grilla completa de puntos.



Al observar el gráfico visual resultante, podemos resaltar los bloques de código y ver cada una de sus funciones.

1. El primer bloque de código reemplaza al nodo *Número*.
2. El segundo bloque de código reemplaza al nodo *Rango de número*.
3. El tercer bloque de código reemplaza al nodo *Fórmula* (así como a *List.Transpose*, *List.Count* y *Number Range*).
4. El cuarto bloque de código consulta una lista de listas, reemplazando el nodo *List.GetItemAtIndex*.

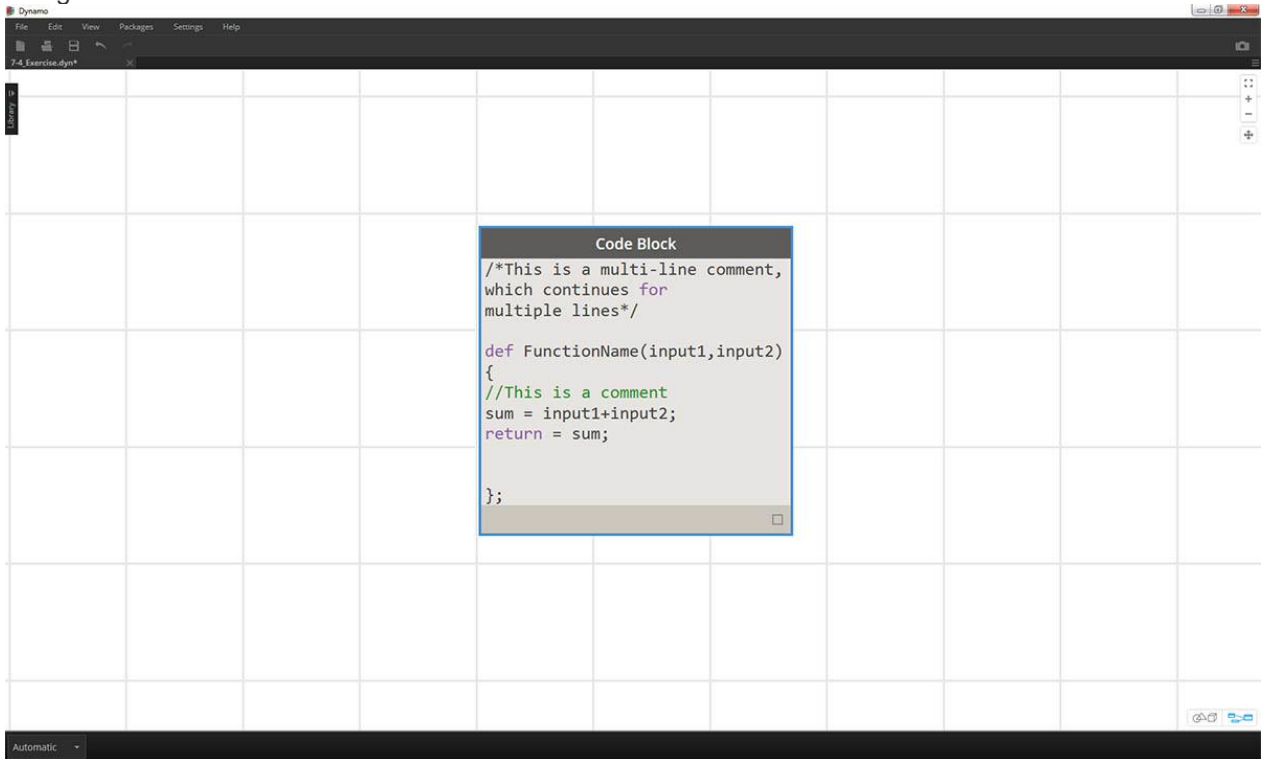
Funciones de bloque de código

Las funciones se pueden crear en un bloque de código y recordar en otra parte en una definición de Dynamo. Esto crea otra capa de control en un archivo paramétrico y se puede ver como una versión basada en texto de un nodo personalizado. En este caso, el bloque de código "principal" es fácilmente accesible y puede ubicarse en cualquier parte del gráfico. ¡No se necesitan cables!

Padre

La primera línea tiene la palabra clave "def", luego el nombre de la función, luego los nombres de las entradas entre paréntesis. Las llaves definen el cuerpo de la función. Devuelve un valor con "return =". Los bloques de código que definen una función no tienen de entrada o salida porque se llaman desde otros bloques de

código.

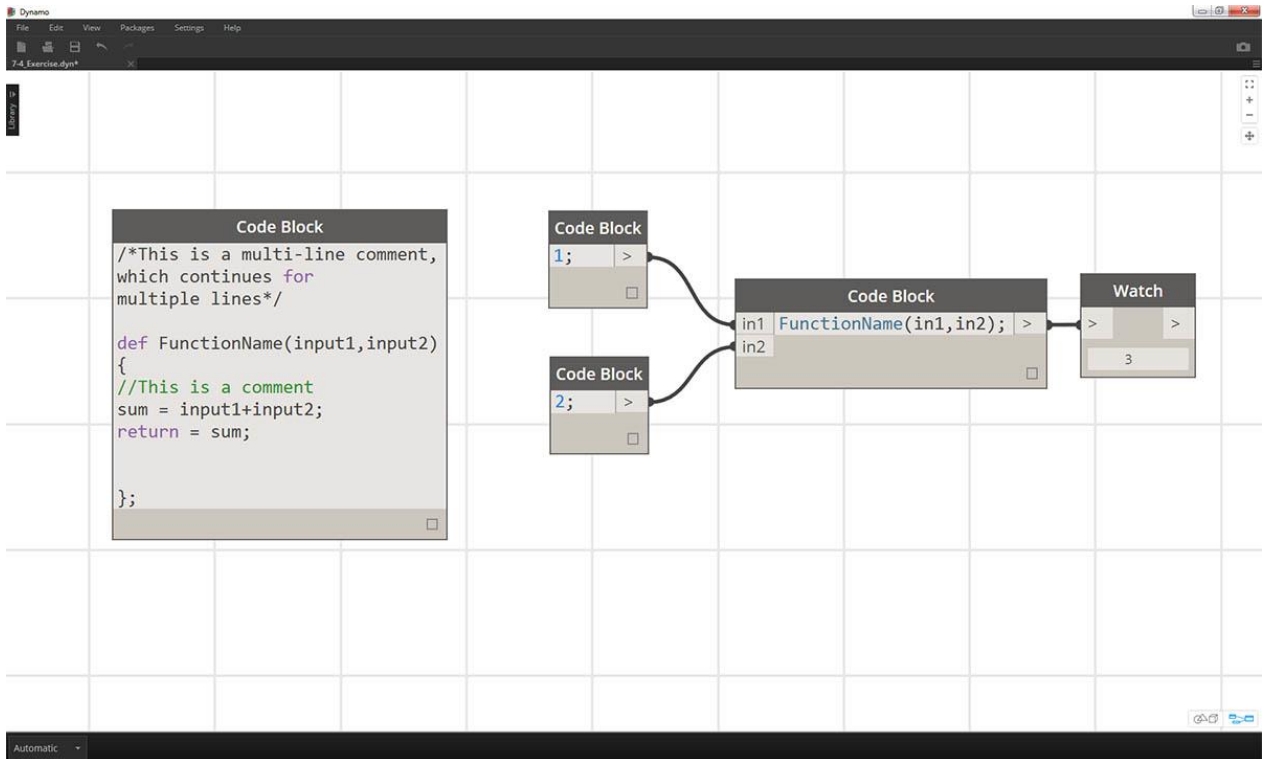


```
/*This is a multi-line comment,
which continues for
multiple lines*/
```

```
def FunctionName(input1,input2)
{
//This is a comment
sum = input1+input2;
return = sum;
};
```

Niños

Llame a la función con otro Bloque de Código en el mismo archivo dando el nombre y la misma cantidad de argumentos. Funciona igual que los nodos listos para usar en su biblioteca.



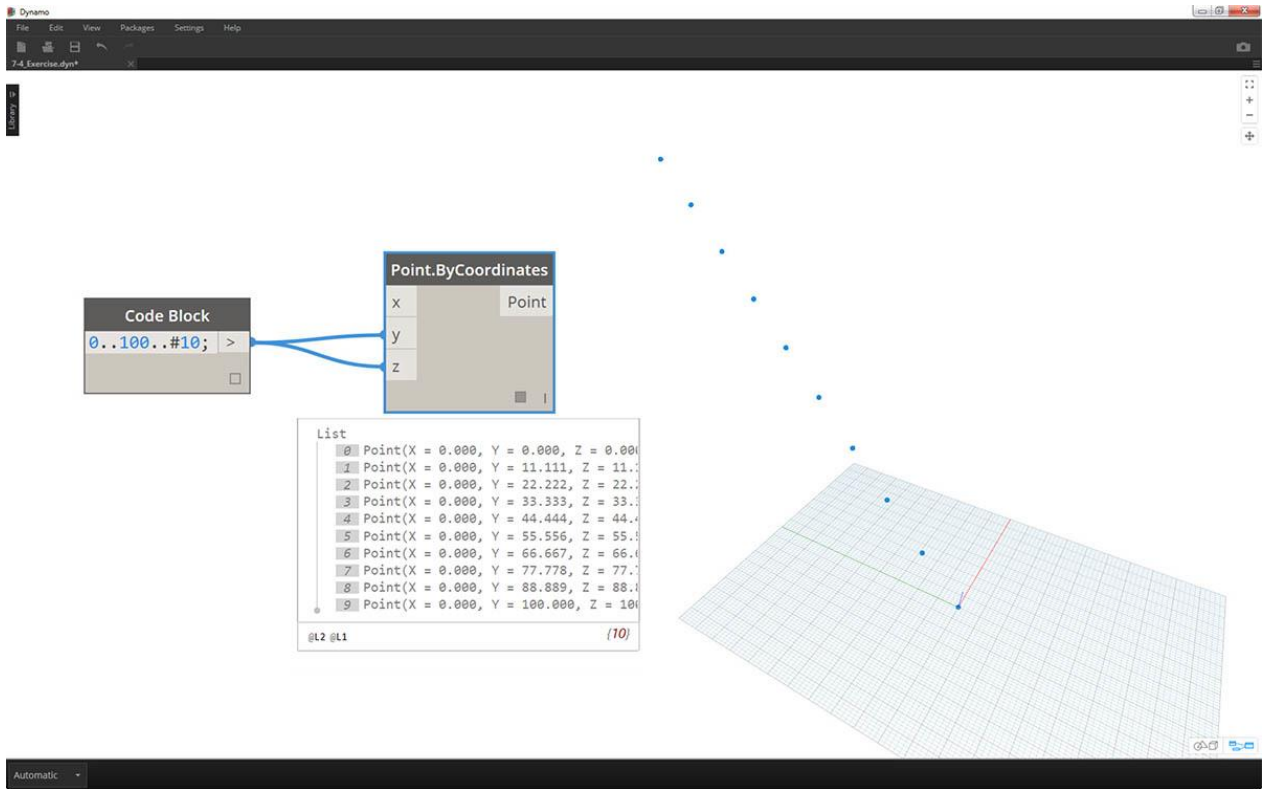
FunctionName(in1,in2);

Ejercicio

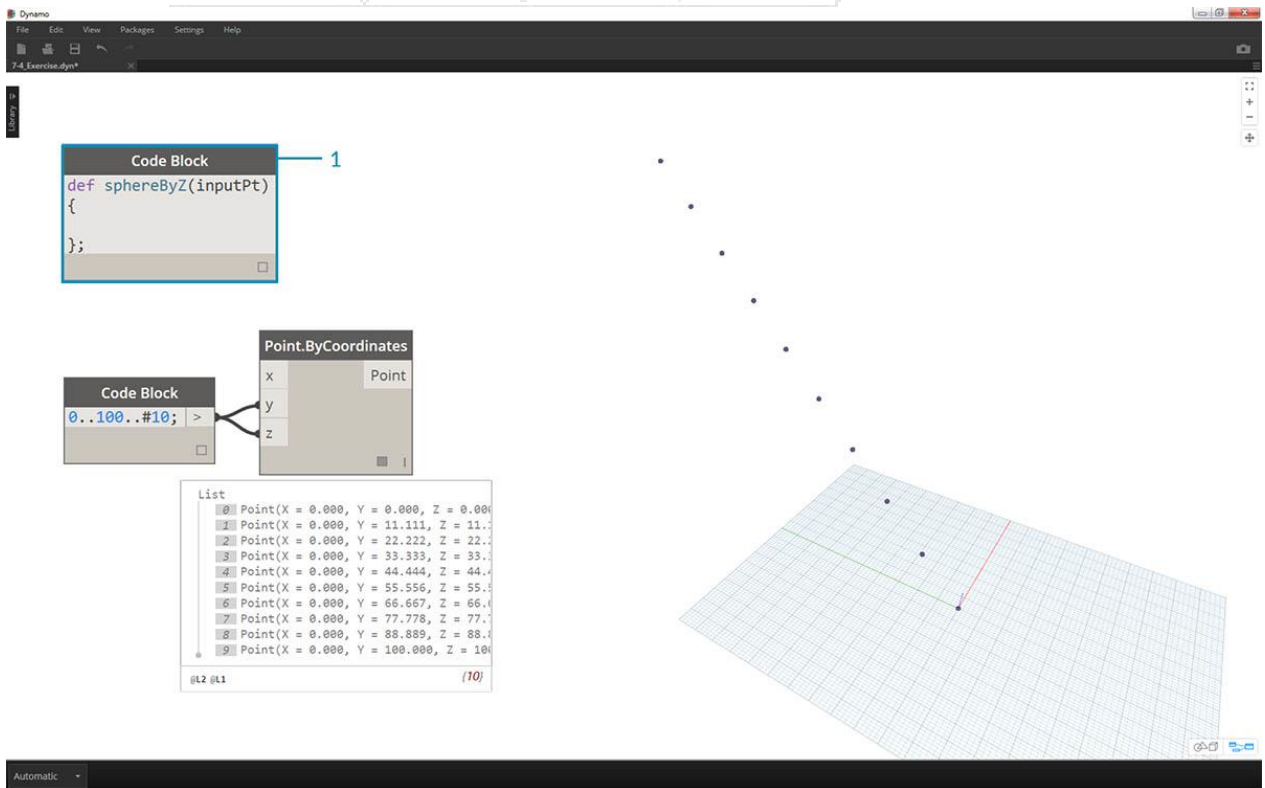
Descargue el archivo de ejemplo que acompaña a este ejercicio. [Functions_SphereByZ.dyn](#)

En este ejercicio, crearemos una definición genérica que creará esferas a partir de una lista de puntos de entrada. El radio de estas esferas está dirigido por la propiedad Z de cada punto.

DARCO
DESDE 1988

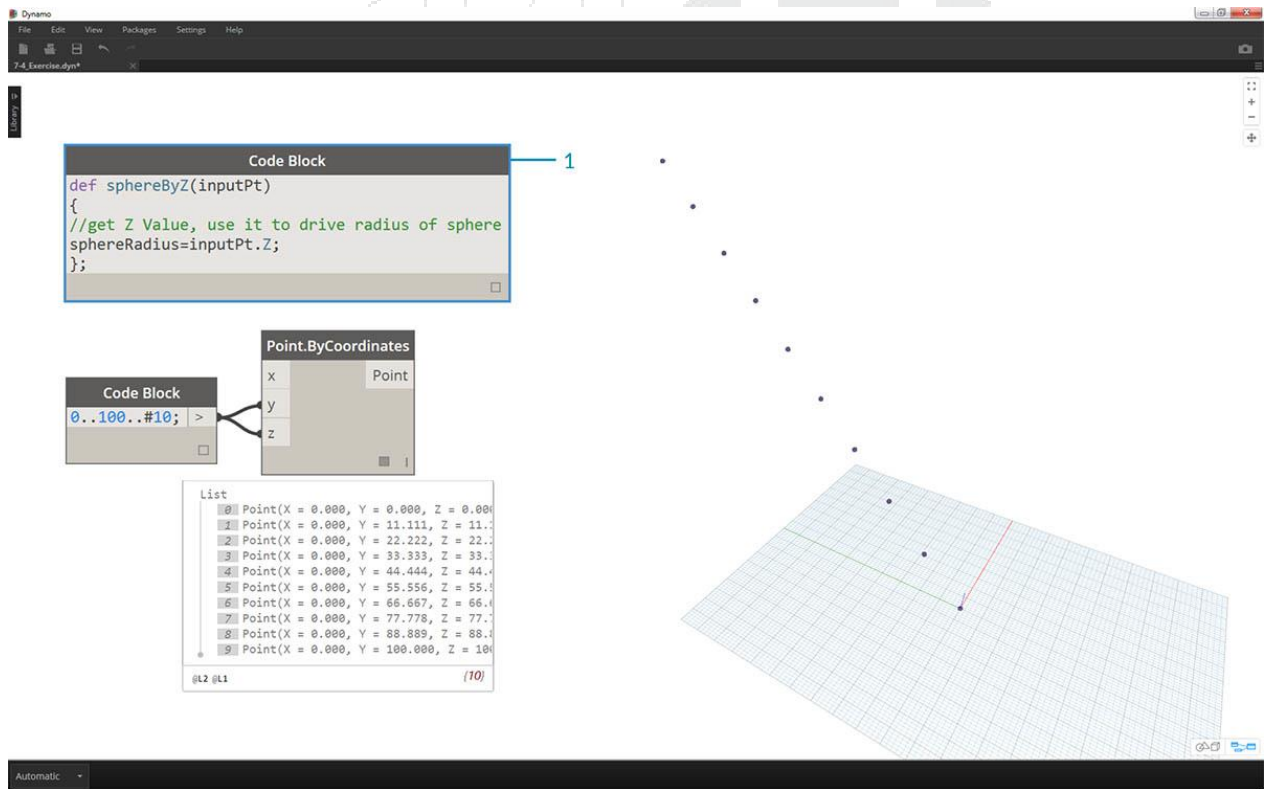


Comencemos con un rango numérico de diez valores que van de 0 a 100. Conéctelos a un nodo *Point.ByCoordinates* para crear una línea diagonal.

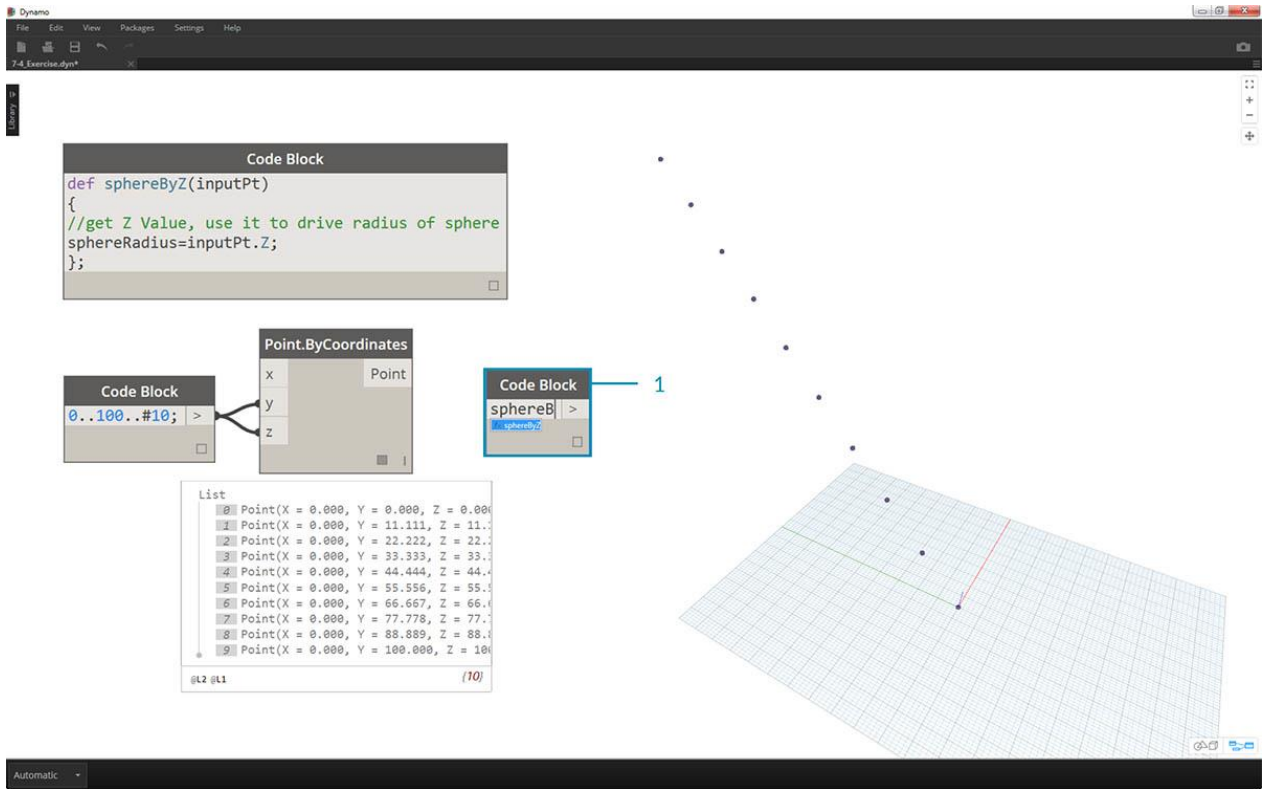


1. Crea un *bloque de código* e introduce nuestra definición usando la línea de código:
2. `def sphereByZ(inputPt){`
3. `};`

El *inputPt* es el nombre que hemos dado para representar los puntos que impulsarán la función. A partir de ahora, la función no está haciendo nada, pero construiremos esta función en los pasos por venir.



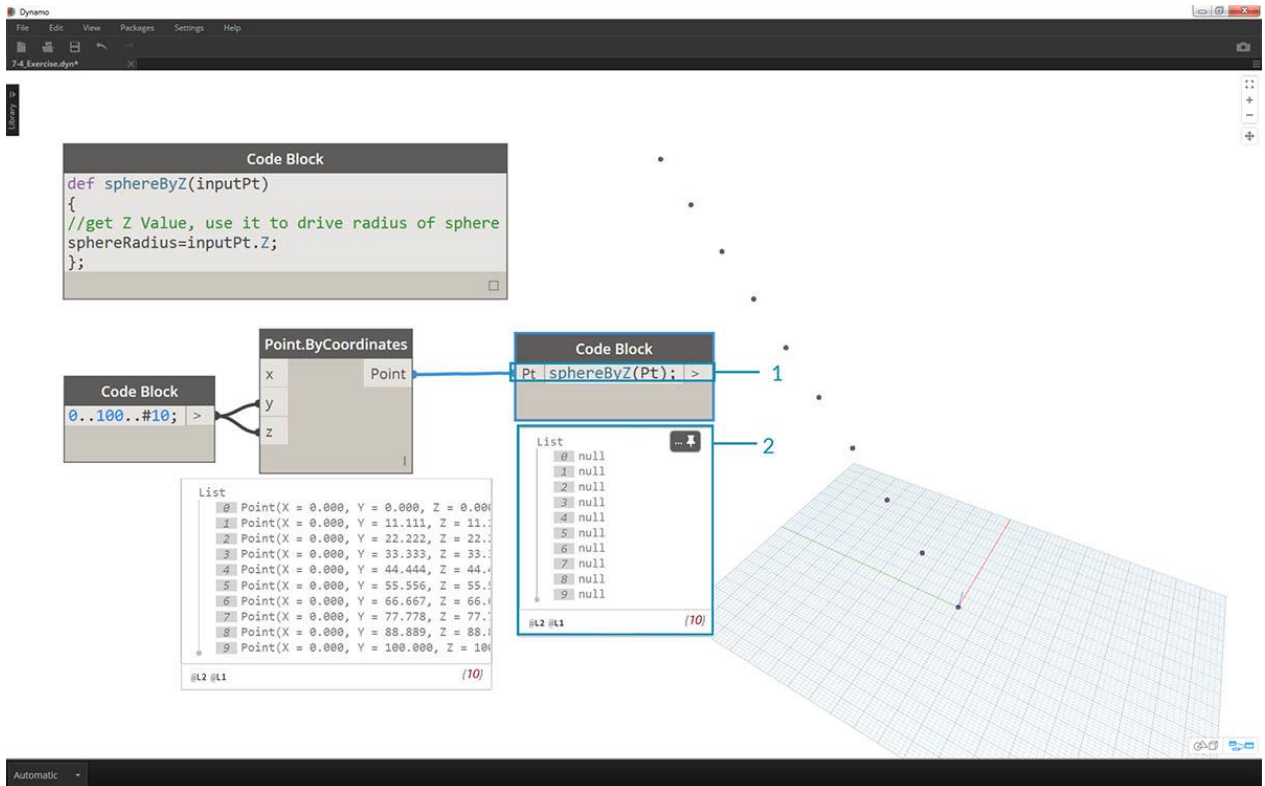
1. Añadiendo a la función de *bloque de código*, colocamos un comentario y una variable *sphereRadius* que consulta la posición Z de cada punto. Recuerde, *inputPt.Z* no necesita parentesis como método. Esta es una *consulta* de las propiedades de un elemento existente, por lo que no se necesitan entradas:
2. `def sphereByZ(inputPt,radiusRatio)`
3. `{`
4. `//get Z Value, use it to drive radius of sphere`
5. `sphereRadius=inputPt.Z;`
6. `};`



1. Ahora, recordemos la función que hemos creado en otro *bloque de código*. Si hacemos doble clic en el lienzo para crear un nuevo *bloque de código* y escribimos *sphereB*, notamos que Dynamo sugiere la función *sphereByZ* que hemos definido. ¡Su función ha sido agregada a la biblioteca intellisense! Muy genial.

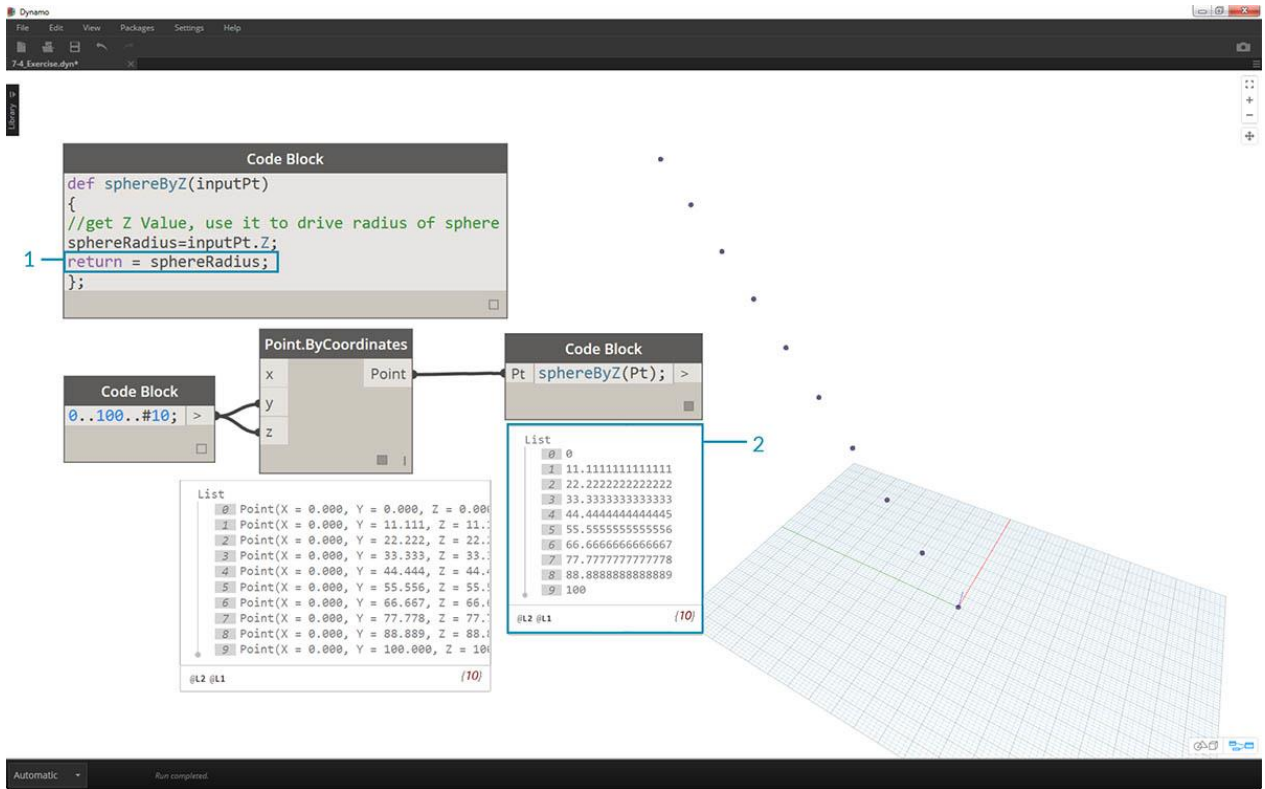


 DESDE 1988



1. Ahora llamamos a la función y creamos una variable llamada *Pt* para conectar los puntos creados en los pasos anteriores: `sphereByZ(Pt)`
2. Observamos desde el resultado que tenemos todos los valores nulos. ¿Por qué es esto? Cuando definimos la función, estamos calculando la variable *sphereRadius*. pero no definimos qué *devuelva* la función como *salida*. Podemos arreglar esto en el siguiente paso.

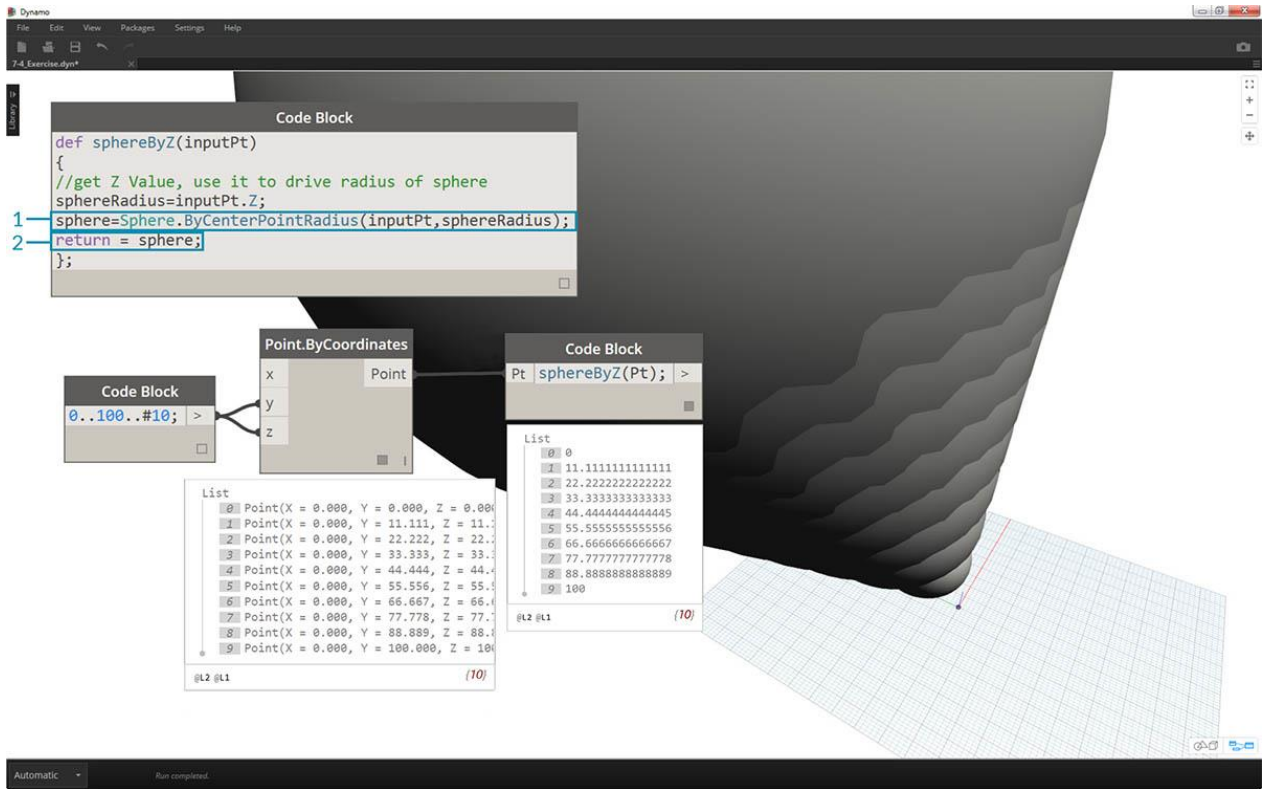
DARCO
DESDE 1988



1. Un paso importante, necesitamos definir la salida de la función agregando la línea `return = sphereRadius;` a la función `sphereByZ`.
2. Ahora vemos que la salida del `bloque de código` nos da las coordenadas Z de cada punto.

DARCO

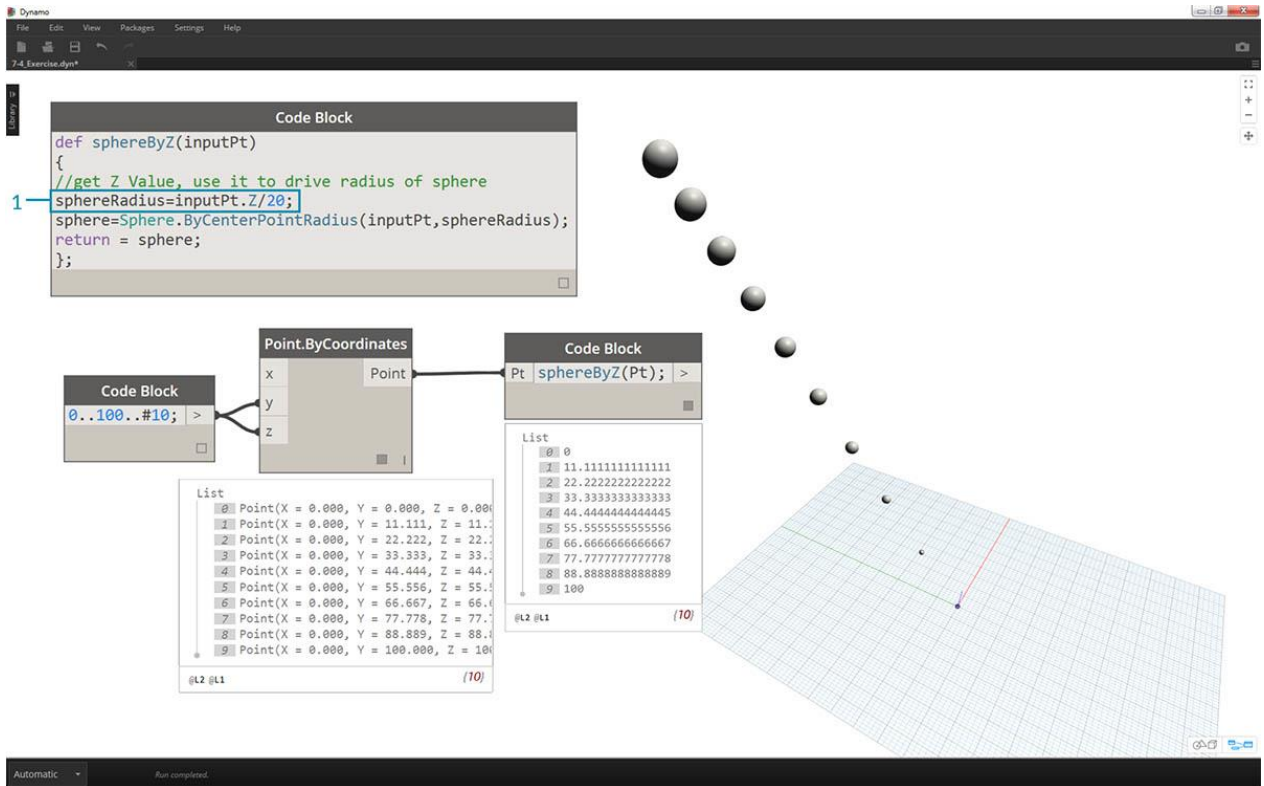
DESDE 1988



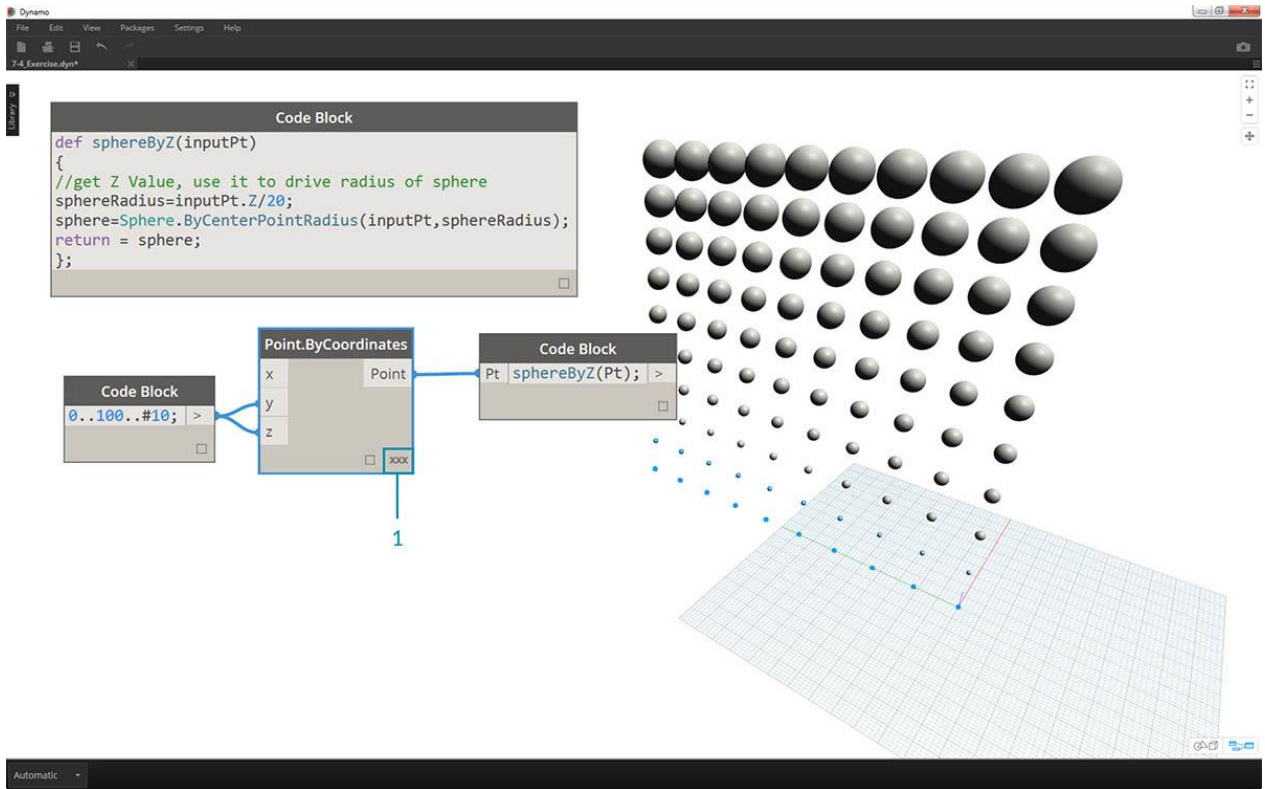
Vamos a crear esferas reales ahora editando el *Padre* función.

1. Primero definimos una esfera con la línea de código: `sphere=Sphere.ByCenterPointRadius(inputPt,sphereRadius);`
2. A continuación, se cambia el valor de retorno para ser el *ámbito* en lugar de la *sphereRadius* : `return = sphere;`. ¡Esto nos da algunas esferas gigantes en nuestra vista previa de Dynamo!

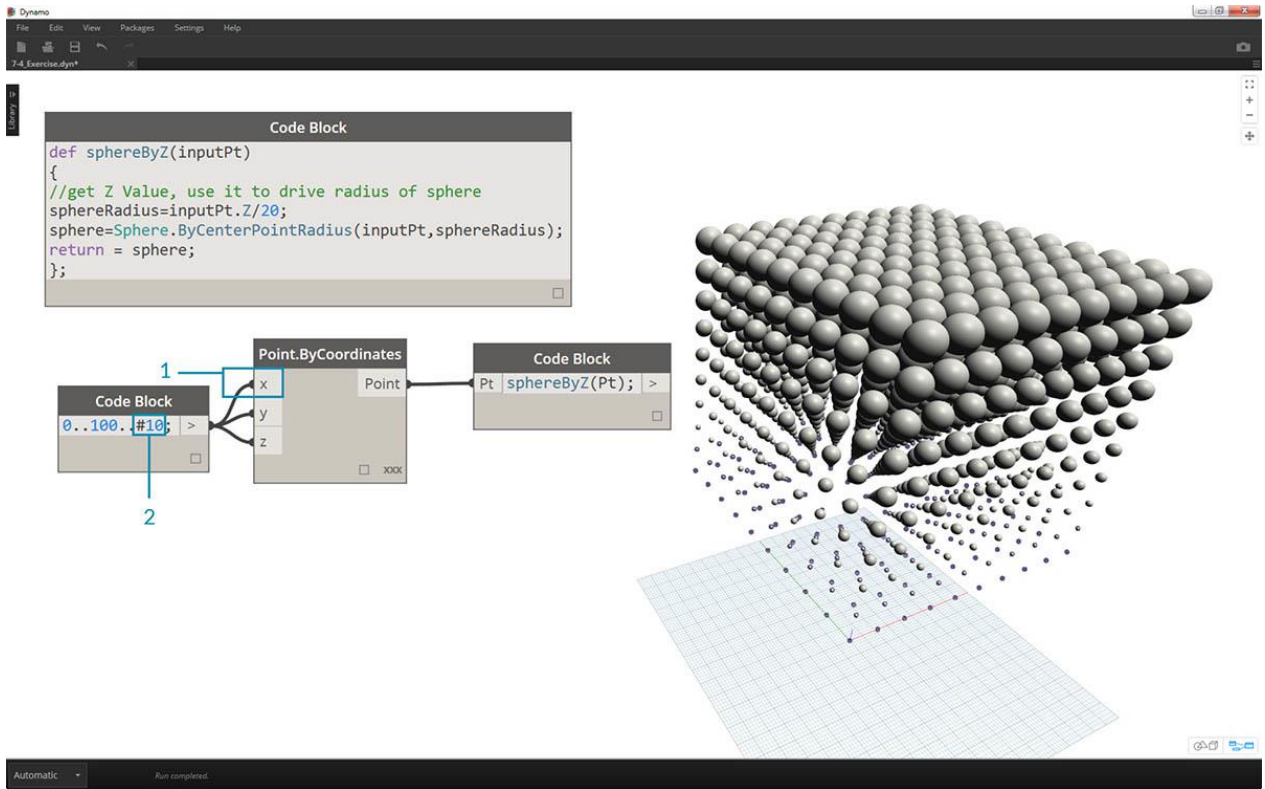
DARCO
DESDE 1988



1. Para moderar el tamaño de estas esferas, vamos a actualizar el *sphereRadius* valor mediante la adición de un divisor: $sphereRadius = inputPt.Z/20$. Ahora podemos ver las esferas separadas y comenzar a dar sentido a la relación entre el radio y el valor Z.

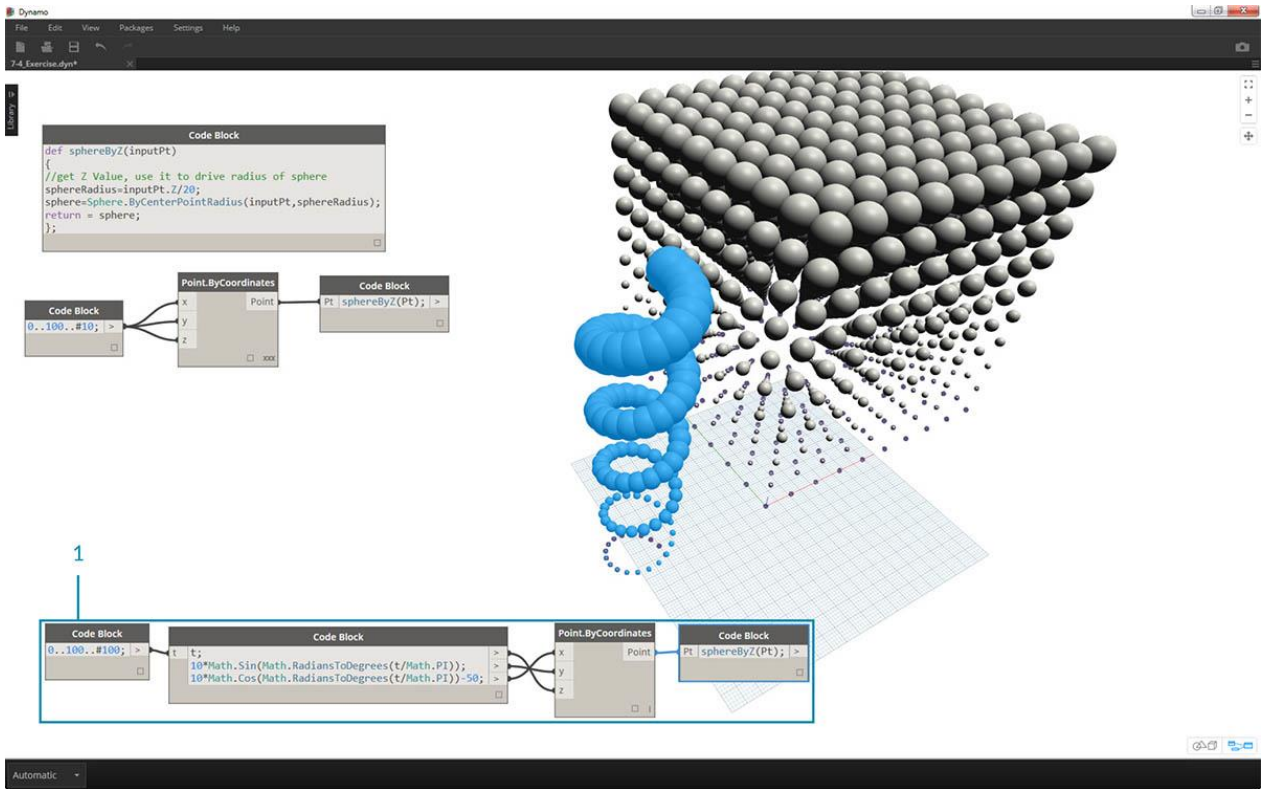


1. En el nodo *Point.ByCoordinates*, al cambiar el cordón de la *Lista más corta* al *Producto cruzado*, creamos una grilla de puntos. La función *sphereByZ* todavía está en pleno efecto, por lo que todos los puntos crean esferas con radios basados en valores Z.

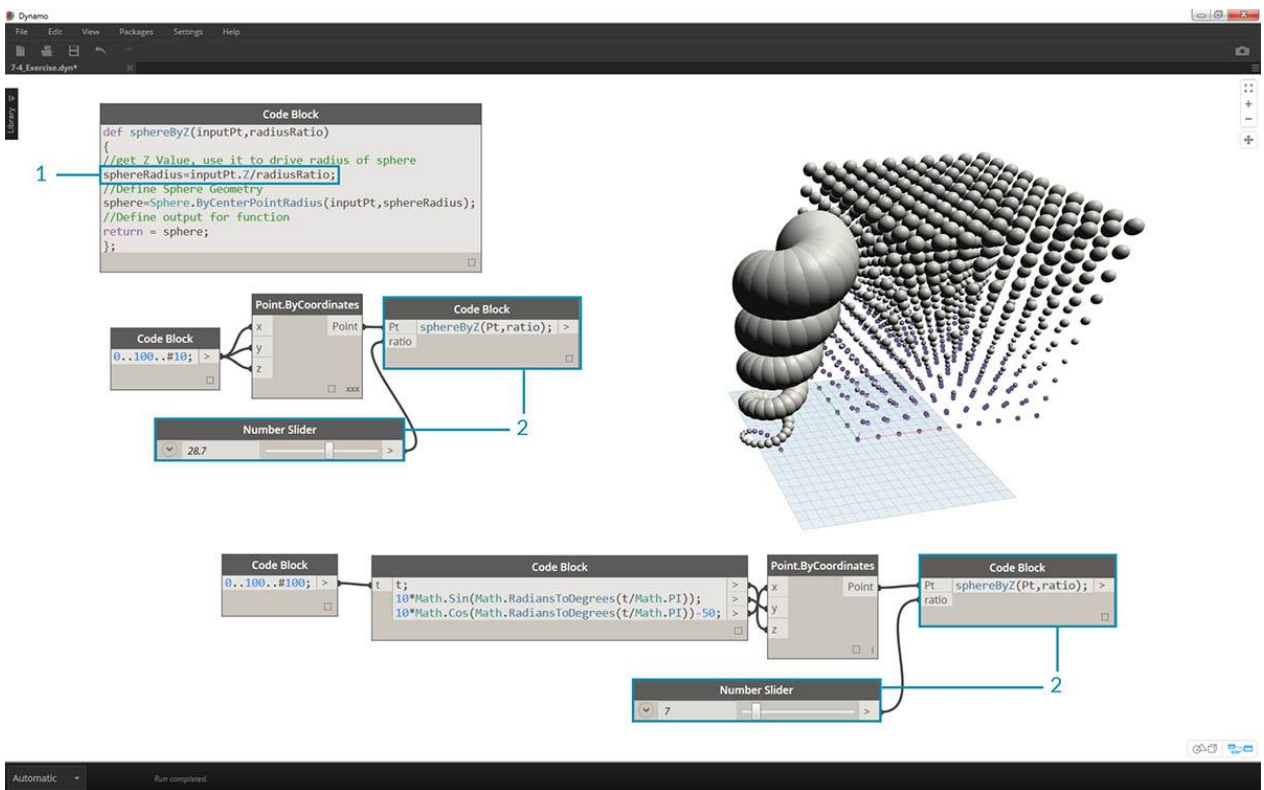


1. Y solo para probar las aguas, conectamos la lista original de números en la entrada X para *Point.ByCoordinates*. Ahora tenemos un cubo de esferas.
2. Nota: si esto lleva mucho tiempo calcular en su computadora, intente cambiar el # 10 a algo como # 5.

DARCO
DESDE 1988



1. Recuerde, la función *sphereByZ* que hemos creado es una función genérica, por lo que podemos recordar la hélice de una lección anterior y aplicarle la función.



Un último paso: manejemos la relación de radio con un parámetro definido por el usuario. Para hacer esto, necesitamos crear una nueva entrada para la función y también reemplazar el divisor 20 con un parámetro.

1. Actualice la definición de *sphereByZ* a:

```
def sphereByZ(inputPt,radiusRatio)
{
//get Z Value, use it to drive radius of sphere
sphereRadius=inputPt.Z/radiusRatio;
//Define Sphere Geometry
sphere=Sphere.ByCenterPointRadius(inputPt,sphereRadius);
//Define output for function
return = sphere;
};
```

1. Actualice los bloques de códigos secundarios agregando una variable de *relación* a la entrada: *sphereByZ(Pt,ratio)*; conecte un control deslizante a la entrada del bloque de código recién creado y varíe el tamaño de los radios en función de la relación del radio.

DARCO
DESDE 1988