

# Table of Contents

1. [Info](#) 0
2. [Einführung](#) 1
  1. [Was ist visuelle Programmierung?](#) 1.1
  2. [Was ist Dynamo?](#) 1.2
  3. [Dynamo in Aktion](#) 1.3
3. [Dynamo!](#) 2
  1. [Dynamo installieren und starten](#) 2.1
  2. [Die Benutzeroberfläche von Dynamo](#) 2.2
  3. [Der Arbeitsbereich](#) 2.3
  4. [ERSTE SCHRITTE](#) 2.4
4. [Die Anatomie visueller Programme](#) 3
  1. [Blöcke](#) 3.1
  2. [Drähte](#) 3.2
  3. [Dynamo-Bibliothek](#) 3.3
  4. [Programme verwalten](#) 3.4
5. [Grundbausteine von Programmen](#) 4
  1. [Daten](#) 4.1
  2. [Math](#) 4.2
  3. [Logik](#) 4.3
  4. [Zeichenfolgen](#) 4.4
  5. [Farbe](#) 4.5
6. [Geometrie für Computational Design](#) 5
  1. [Geometrie – Überblick](#) 5.1
  2. [Vektoren, Ebenen und Koordinatensysteme](#) 5.2
  3. [Punkte](#) 5.3
  4. [Kurven](#) 5.4
  5. [Oberflächen](#) 5.5
  6. [Volumenkörper](#) 5.6
  7. [Netze](#) 5.7
  8. [Importieren von Geometrie](#) 5.8
7. [Entwerfen mit Listen](#) 6
  1. [Was ist eine Liste?](#) 6.1
  2. [Arbeiten mit Listen](#) 6.2
  3. [Listen von Listen](#) 6.3
  4. [n-dimensionale Listen](#) 6.4
8. [Codeblöcke und DesignScript](#) 7
  1. [Was ist ein Codeblock?](#) 7.1
  2. [DesignScript-Syntax](#) 7.2
  3. [Kurzschreibweisen](#) 7.3
  4. [Codeblock-Funktionen](#) 7.4
9. [Dynamo for Revit](#) 8
  1. [Verbindung zu Revit](#) 8.1
  2. [Auswählen](#) 8.2
  3. [Bearbeiten](#) 8.3
  4. [Erstellen](#) 8.4
  5. [Anpassen](#) 8.5
  6. [Dokumentation](#) 8.6
10. [Wörterbücher in Dynamo](#) 9
  1. [Wörterbücher](#) 9.1
  2. [Wörterbuch-Blöcke](#) 9.2
  3. [Wörterbücher in Codeblöcken](#) 9.3
  4. [Wörterbücher – Revit-Anwendungsfälle](#) 9.4
11. [Benutzerdefinierte Blöcke](#) 10
  1. [Benutzerdefinierte Blöcke](#) 10.1
  2. [Benutzerdefinierte Blöcke erstellen](#) 10.2
  3. [Hinzufügen zu Ihrer Bibliothek](#) 10.3
  4. [Python](#) 10.4
  5. [Python und Revit](#) 10.5
  6. [Python-Vorlagen](#) 10.6
12. [Pakete](#) 11
  1. [Pakete](#) 11.1
  2. [Fallstudie zu Paketen: Mesh Toolkit](#) 11.2

- 3. [Entwickeln von Paketen](#) 11.3
- 4. [Veröffentlichen von Paketen](#) 11.4
- 5. [Was ist Zero-Touch?](#) 11.5
- 13. [Dynamo im Internet](#) 12
  - 1. [Ins Internet senden \(mit Dynamo Studio\)](#) 12.1
  - 2. [Customizer-Ansicht](#) 12.2
- 14. [Optimale Verfahren](#) 13
  - 1. [Vorgehensweisen für Diagramme](#) 13.1
  - 2. [Vorgehensweisen zur Skripterstellung](#) 13.2
  - 3. [Referenz für die Skripterstellung](#) 13.3
- 15. [Anhang](#) 14
  - 1. [Ressourcen](#) 14.1
  - 2. [Index: Blöcke](#) 14.2
  - 3. [Dynamo-Pakete](#) 14.3
  - 4. [Dynamo-Beispieldateien](#) 14.4

# Info

## Dynamo Primer

### Für Dynamo 2.0

Dynamo Primer v1.3 [hier](#) herunterladen



# Dynamo

Dynamo ist eine Open-Source-Plattform zur visuellen Programmierung, die für Konstrukteure konzipiert ist.

### Willkommen

Sie haben gerade Dynamo Primer geöffnet, die umfassende Einführung in die visuelle Programmierung in Autodesk Dynamo Studio. Bei diesem Primer handelt es sich um ein laufendes Projekt zum Austauschen der Grundlagen der Programmierung. Zu den enthaltenen Themen gehören die Verwendung der der Dynamo-Plattform für rechnerische Geometrie (Computational Geometry), Best Practices für regelbasierte Konstruktionen, interdisziplinäre Programmieranwendungen und andere.

Der Leistungsumfang von Dynamo spiegelt sich in einer breiten Palette von konstruktionsbezogenen Aktivitäten wider. Dynamo bietet eine umfassende Reihe an leicht zugänglichen Möglichkeiten für den ersten Einstieg, die ständig erweitert wird:

- **Erste Schritte** mit der visuellen Programmierung
- Arbeitsabläufe in verschiedenen Softwareanwendungen **verbinden**
- Eine aktive Community an Benutzern, Beitragenden und Entwicklern **einbeziehen**
- Eine Open-Source-Plattform für die fortlaufende Optimierung **entwickeln**

Vor dem Hintergrund dieser Aktivitäten und den aufregenden Möglichkeiten, die uns die Arbeit mit Dynamo bietet, benötigen wir ein Dokument desselben Kalibers: Dynamo Primer.

Dieser Leitfaden enthält Kapitel, die mit Mode Lab entwickelt wurden. In diesen Kapiteln liegt der Schwerpunkt auf den Grundlagen, die Sie zum Entwickeln Ihrer eigenen visuellen Programme mit Dynamo benötigen, und auf wichtigen Einblicken in die weiteren Schritte mit Dynamo. Der Primer bietet die folgenden Lerninhalte:

- **Kontext** – Was genau ist die "Visuelle Programmierung", und mit welchen Konzepten muss ich vertraut sein, um tiefer in Dynamo einzutauchen?
- **Erste Schritte** – Wie erhalte ich Dynamo und erstelle damit mein erstes Programm?
- **Bestandteile eines Programms** – Was sind die funktionalen Teile von Dynamo, und wie kann ich sie verwenden?
- **Bausteine** – Was sind "Daten", und was sind einige grundlegende Typen, mit deren Verwendung ich in meinen Programmen beginnen kann?
- **Geometrie für Konstruktionen** – Wie arbeite ich mit geometrischen Elementen in Dynamo?
- **Listen, Listen, Listen** – Wie verwalte und koordiniere ich meine Datenstrukturen?
- **Code in Blöcken** – Wie kann ich Dynamo um meinen eigenen Code erweitern?
- **Berechnungs-BIM** – Wie kann ich Dynamo mit einem Revit-Modell verwenden?
- **Benutzerdefinierte Blöcke** – Wie kann ich meine eigenen Blöcke erstellen?
- **Pakete** – Wie kann ich meine Werkzeuge mit der Community austauschen?

Es wird eine aufregende Zeit für Sie werden – Sie werden mehr über Dynamo erfahren, mit Dynamo arbeiten und für Dynamo entwickeln. Legen Sie einfach los!



## Open Source

Dynamo Primer ist ein Open-Source-Projekt. Wir haben uns dazu verpflichtet, hochwertige Inhalte bereitzustellen, und begrüßen daher jegliche Art von Feedback von Ihnen. Wenn Sie ein Problem melden möchten, posten Sie es auf unserer GitHub-Problemseite: <https://github.com/DynamoDS/DynamoPrimer/issues>

Wenn Sie sich mit einem neuen Abschnitt, Änderungen oder etwas anderem an diesem Projekt beteiligen möchten, navigieren Sie zum Github-Repository, um loszulegen: <https://github.com/DynamoDS/DynamoPrimer>.

---

## Das Projekt Dynamo Primer

Dynamo Primer ist ein Open-Source-Projekt, das von Matt Jezyk und dem Dynamo-Entwicklungsteam bei Autodesk initiiert wurde.

**Mode Lab** wurde damit beauftragt, die erste Version dieses Primers zu verfassen. Wir bedanken uns bei allen für ihre Arbeit beim Zusammenstellen dieses wichtigen Nachschlagewerks.



**John Pierson von Parallax Team** wurde damit beauftragt, den Primer mit den Änderungen an Dynamo 2.0 zu aktualisieren.



## Danksagungen

Ein besonderer Dank geht an Ian Keough für die Initiierung und Begleitung des Dynamo-Projekts.

Vielen Dank auch an Matt Jezyk, Ian Keough, Zach Kron, Racel Williams und Colin McCrone für die tolle Zusammenarbeit und die Möglichkeit zur Teilnahme an einer breiten Palette von Dynamo-Projekten.

## Software und Ressourcen

**Dynamo** Die aktuelle stabile Version von Dynamo ist Version 2.0.

<http://dynamobim.com/download/> oder <http://dynamobuilds.com>

**DynamoBIM** Die beste Quelle für zusätzliche Informationen, Lerninhalte und Foren ist die DynamoBIM-Website.

<http://dynamobim.org>

**Dynamo GitHub** Dynamo ist ein Open-Source-Entwicklungsprojekt auf Github. Um zu dem Projekt beizutragen, gehen Sie zu DynamoDS.

<https://github.com/DynamoDS/Dynamo>

**Kontakt** Informieren Sie uns über Probleme mit diesem Dokument.

Dynamo@autodesk.com

## Lizenz

Copyright 2018 Autodesk

Lizenziert unter der Apache-Lizenz, Version 2.0 (die "Lizenz"); Sie dürfen diese Datei nur unter Einhaltung der Lizenzbestimmungen verwenden. Sie erhalten die Lizenz unter:

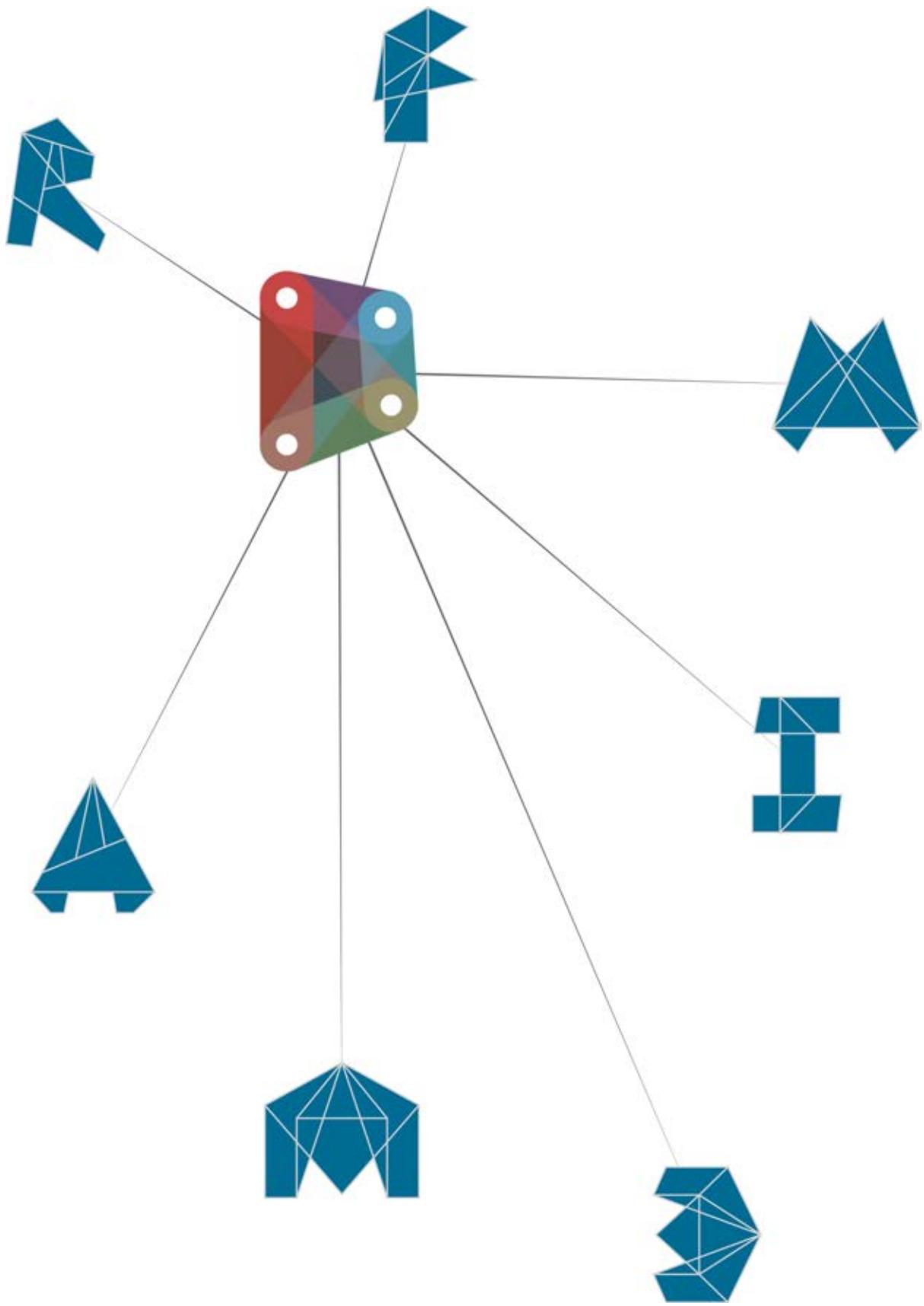
<http://www.apache.org/licenses/LICENSE-2.0>

Außer wenn durch geltendes Recht gefordert oder mit schriftlicher Genehmigung, wird die Software unter der Lizenz "WIE BESEHEN" bereitgestellt, OHNE ausdrückliche oder stillschweigende GARANTIEN ODER BEDINGUNGEN JEGLICHER ART. Informationen zu spezifischen sprachenrelevanten Rechten und Einschränkungen finden Sie in der Lizenz.

# **Einführung**

## **Einführung**

Dynamo wurde ursprünglich als Zusatzmodul für Building Information Modeling in Revit entwickelt, ist seitdem gewachsen und kann auf vielfältige Weise genutzt werden. In erster Linie ist Dynamo eine Plattform, die es Designern ermöglicht, mit visueller Programmierung zu experimentieren, Probleme zu lösen und eigene Werkzeuge zu entwickeln. Zu Beginn dieser Einführung finden Sie hier eine Beschreibung des Kontexts für Dynamo: Was ist dieses Programm und wie können Sie es verwenden?



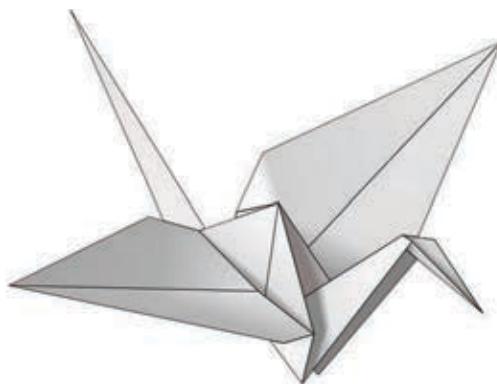
# Was ist visuelle Programmierung?

## Was ist visuelle Programmierung?

Beim Entwurfsprozess müssen häufig visuelle, systemrelevante oder geometrische Beziehungen zwischen den Teilen eines Entwurfs eingerichtet werden. In der Mehrzahl der Fälle werden bei der Entwicklung dieser Beziehungen Arbeitsabläufe verwendet, die mithilfe von Regeln vom Konzept zum Endergebnis führen. Dabei setzen Sie, vielleicht ohne es zu wissen, Algorithmen ein: Sie definieren in Einzelschritten nacheinander ablaufende Aktionen, die einer grundlegenden Logik aus Eingabe, Verarbeitung und Ausgabe folgen. Bei der Programmierung können Sie weiterhin auf diese Weise arbeiten, wobei die Algorithmen allerdings formalisiert werden müssen.

## Algorithmen konkret

Algorithmen sind hocheffizient und bieten vielfältige Möglichkeiten; der Begriff **Algorithmus** kann jedoch missverstanden werden. Algorithmen generieren eventuell unerwartete, verrückte oder coole Ergebnisse, mit Zauberei haben sie jedoch nichts zu tun. Sie sind im Gegenteil an sich recht einfach. Dies wird hier an einem konkreten Beispiel erläutert: einem Origami-Kranich. Dabei beginnen Sie mit einem quadratischen Blatt Papier (Eingabe), führen eine Folge von Faltvorgängen aus (Verarbeitungsaktionen) und erhalten einen Kranich (Ausgabe).

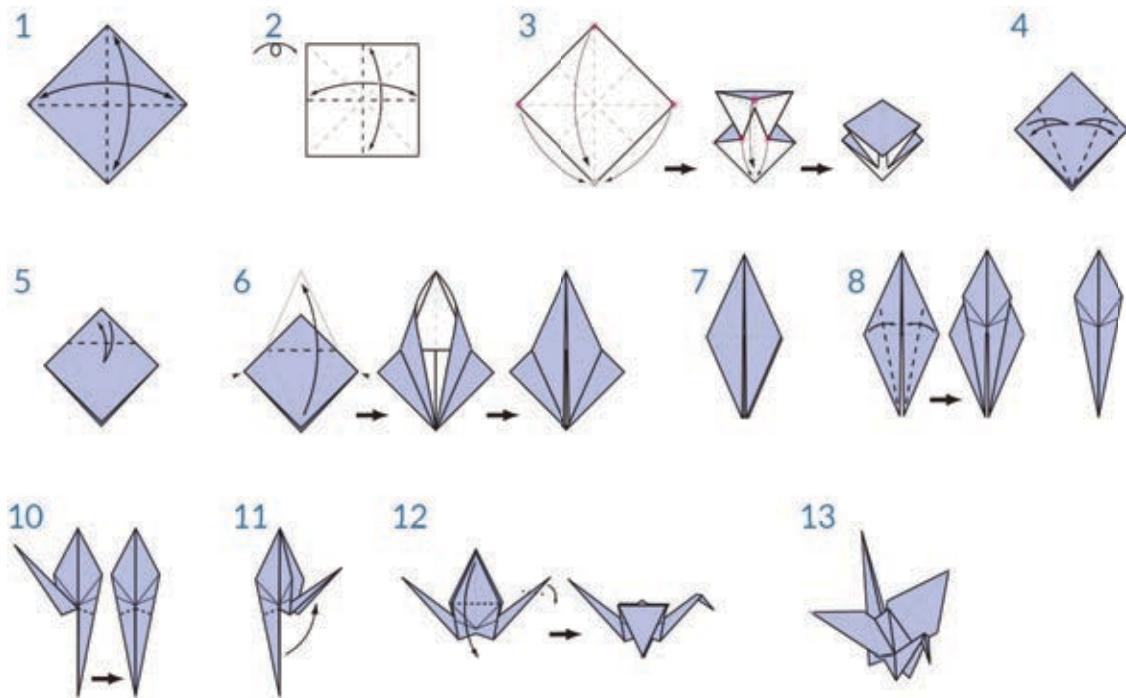


Worin besteht hier der Algorithmus? Er ist die abstrakte Abfolge von Schritten, die Sie auf unterschiedliche Weise darstellen können: in Textform oder grafisch.

### Textanweisungen:

1. Beginnen Sie mit einem quadratischen Blatt Papier, wobei die farbige Seite oben liegt. Falten Sie es auf die Hälfte und entfalten Sie es wieder. Falten Sie es anschließend in der anderen Richtung auf die Hälfte.
2. Wenden Sie das Papier auf die weiße Seite. Falten Sie das Papier auf die Hälfte, falzen Sie es scharf und entfalten Sie es wieder. Falten Sie es auf dieselbe Weise in die andere Richtung.
3. Führen Sie mithilfe der vorhandenen Falten die drei oberen Ecken des Modells nach unten auf die untere Ecke. Drücken Sie das Modell flach.
4. Falten Sie die oben liegenden dreieckigen Klappen zur Mitte und entfalten Sie sie wieder.
5. Falten Sie den oberen Teil des Modells nach unten, falzen Sie ihn scharf und entfalten Sie ihn wieder.
6. Öffnen Sie die zuoberst liegende Klappe des Modells, führen Sie sie nach oben und drücken Sie zugleich die Seiten des Modells nach innen. Drücken Sie das Modell flach und falzen Sie es scharf.
7. Drehen Sie das Modell um und wiederholen Sie die Schritte 4 bis 6 auf der Rückseite.
8. Falten Sie die oberen Klappen zur Mitte.
9. Wiederholen Sie dies auf der anderen Seite.
10. Falten Sie beide "Beine" des Modells nach oben, falzen Sie sie scharf und entfalten Sie sie.
11. Falten Sie mit inneren Gegenfalten die "Beine" entlang den neuen Falzlinien.
12. Formen Sie mit einer inneren Gegenfalte auf einer Seite den Kopf und falten Sie die Flügel nach unten.
13. Damit haben Sie einen Kranich gefaltet.

### Grafische Anleitung:



### Programmierung – Definition

Mit beiden Anleitungen erhalten Sie einen Kranich. Wenn Sie die Anweisungen ausgeführt haben, haben Sie damit einen Algorithmus angewendet. Der einzige Unterschied besteht in der Art und Weise, in der die formale Darstellung der Anweisungsfolge gelesen wird. Damit gelangen Sie zur **Programmierung**. Programmierung, häufig als Kurzform für *Computerprogrammierung* verwendet, ist die Formalisierung einer Reihe von Aktionen, sodass ein ausführbares Programm entsteht. Wenn Sie die oben stehenden Anweisungen zum Herstellen eines Kranichs in ein Format umwandeln, das ein Computer lesen und ausführen kann, programmieren Sie.

Der entscheidende Schritt und zugleich das erste Hindernis beim Programmieren besteht darin, dass eine effiziente Kommunikation mit dem Computer nur unter Zuhilfenahme von Abstraktion möglich ist. Hierfür kommt eine Vielzahl von Programmiersprachen zum Einsatz, etwa Javascript, Python oder C. Wenn Sie eine reproduzierbare Folge von Anweisungen wie diejenige für den Origami-Kranich verfassen können, muss diese lediglich für den Computer übersetzt werden. Sie ermöglichen dadurch letztlich die Herstellung eines Kranichs oder sogar vieler Kraniche, die sich geringfügig unterscheiden, durch den Computer. Darin liegt die große Stärke der Programmierung: Der Computer führt jede beliebige Aufgabe oder Folge von Aufgaben, die Sie ihm zuweisen, wiederholt ohne Verzögerungen und ohne menschliche Irrtümer aus.

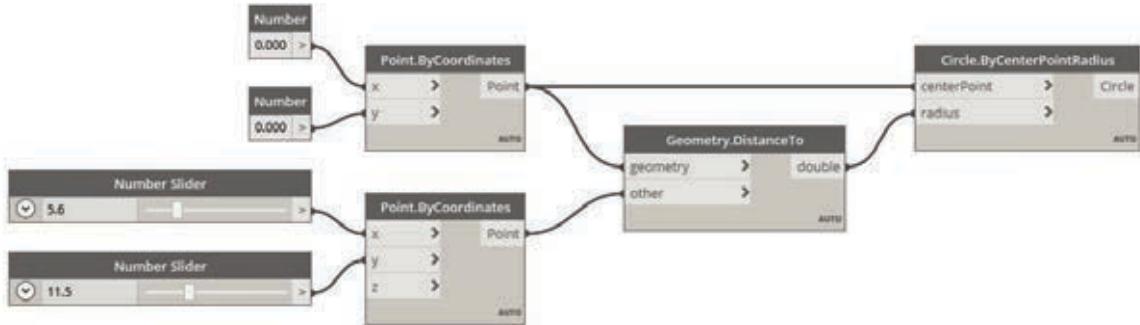
### Visuelle Programmierung – Definition

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Visual Programming - Circle Through Point.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

Angenommen, Sie werden aufgefordert, Anweisungen zum Falten eines Origami-Kranichs zu verfassen: Wie würden Sie vorgehen? Würden Sie Abbildungen, Text oder eine Kombination aus beiden verwenden?

Wenn Sie in Ihrer Antwort Abbildungen nennen, ist die **visuelle Programmierung** definitiv für Sie geeignet. Die visuelle Programmierung folgt im Wesentlichen demselben Ablauf wie die Textprogrammierung. Beiden liegen dieselben Prinzipien der Formalisierung zugrunde. Die Anweisungen und Beziehungen des Programms werden jedoch über eine grafische ("visuelle") Benutzeroberfläche definiert. Sie geben keinen durch eine Syntax geregelten Text ein, sondern verbinden vordefinierte Blöcke miteinander. Vergleichen Sie hier die Programmierung desselben Algorithmus "Zeichne einen Kreis durch einen Punkt" in Form von Blöcken und als Code.

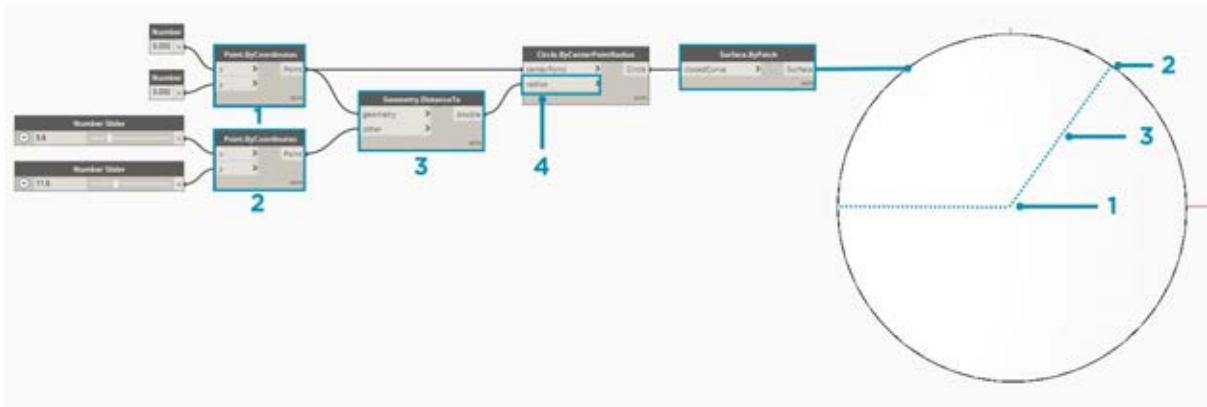
### Visuelles Programm:



### Textprogramm:

```
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);
```

Die Ergebnisse des Algorithmus:



Das Visuelle an dieser Art der Programmierung erleichtert den Einstieg, und Designer fühlen sich häufig davon angesprochen. Dynamo folgt dem Muster der visuellen Programmierung. Sie können jedoch, wie später gezeigt wird, nach wie vor auch die Textprogrammierung in dieser Anwendung verwenden.

# **Was ist Dynamo?**

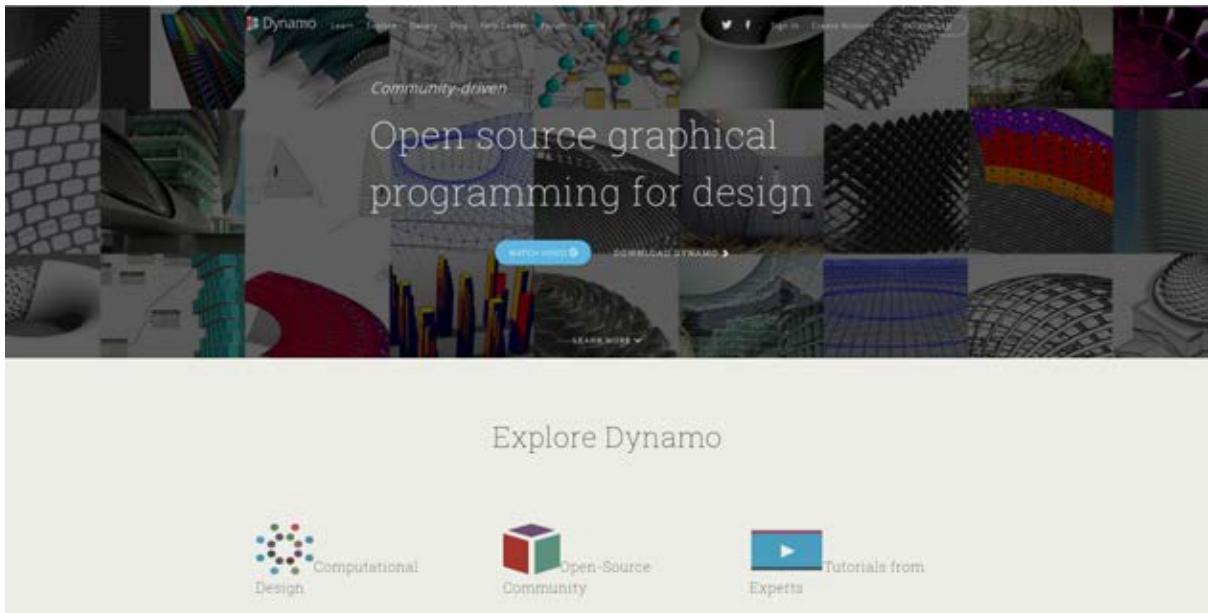
## **Was ist Dynamo?**

Was Dynamo ist, ist davon abhängig, wie Sie es verwenden. Die Arbeit mit Dynamo kann darin bestehen, die Anwendung zusammen mit anderer Autodesk-Software oder allen einzusetzen, einen Prozess aus der visuellen Programmierung zu nutzen oder sich an der großen Community der Benutzer und Beitragenden zu beteiligen.

## **Die Anwendung**

Die Anwendung Dynamo ist eine Software, die Sie herunterladen und entweder im Standalone-Modus (Sandbox) oder als Zusatzmodul für andere Software wie Revit oder Maya ausführen können. Sie wird wie folgt beschrieben:

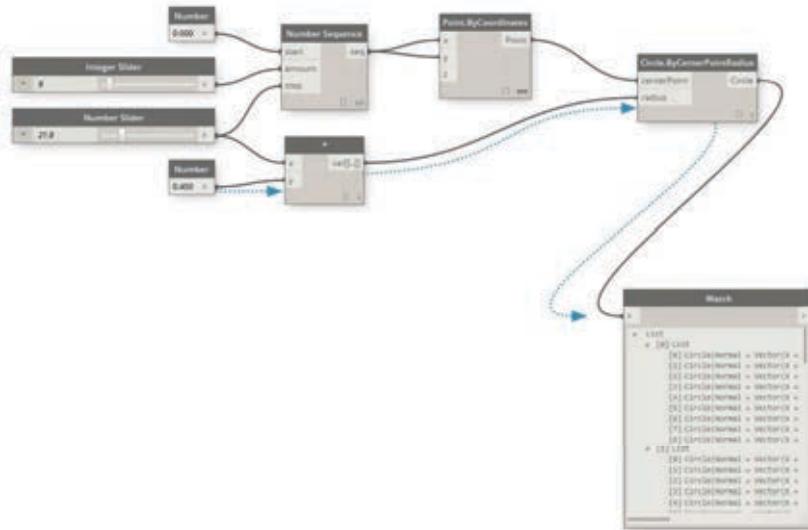
Ein Tool für die visuelle Programmierung, das für Nichtprogrammierer und Programmierer gleichermaßen einsetzbar sein soll. Der Benutzer hat die Möglichkeit, visuell Skripts für das Programmverhalten zu erstellen, benutzerdefinierte Logik zu definieren sowie Skripts in verschiedenen textbasierten Programmiersprachen zu erstellen.



1. Demonstration von Dynamo mit Revit
2. Installationsprogramm herunterladen

## Prozess

Nachdem Sie die Anwendung installiert haben, haben Sie in Dynamo die Möglichkeit, in einem visuellen Programmierprozess zu arbeiten, wobei Sie Elemente miteinander verbinden und dadurch die Beziehungen und Abfolgen von Aktionen definieren, aus denen sich benutzerdefinierte Algorithmen zusammensetzen. Ihre Algorithmen können Sie für ein breites Spektrum an Verwendungszwecken von der Verarbeitung von Daten bis zum Generieren von Geometrie einsetzen – in Echtzeit und ohne eine einzige Zeile Code zu schreiben.



Fügen Sie Elemente hinzu, verbinden Sie sie und schon sind Sie dabei, visuelle Programme zu erstellen.

## Community

Ohne seine starke Community engagierter Benutzer und aktiver Beitragender hätte sich Dynamo nie so weit entwickelt. Engagieren auch Sie sich in der Community, indem Sie das Blog lesen, Ihre Arbeiten der Galerie hinzufügen oder an Forumsdiskussionen zu Dynamo teilnehmen.

The screenshot shows the homepage of the Dynamo Forum. At the top, there's a navigation bar with links for Learn, Support, Gallery, Shop, Help Center, Forum, and a search bar. Below the navigation, there's a header with "all categories" and a "Listed" button. The main content area is divided into sections: "Topic" (with a post about Python in Rhino), "Category" (listing posts by category like HQ, Testing, etc.), and "Recent" (listing recent posts). Each post includes a thumbnail, title, category, user profile, replies, views, and activity.

## Die Plattform

Dynamo ist als visuelles Programmierwerkzeug für Designer konzipiert. Dies ermöglicht es, Tools zu entwickeln, die externe Bibliotheken oder beliebige Autodesk-Produkte nutzen, in denen eine API zur Verfügung steht. In Dynamo Studio können Sie Programme in einer Sandbox-ähnlichen Anwendung entwickeln. Das Dynamo-Ökosystem wächst jedoch nach wie vor weiter.

Da der Quellcode für dieses Projekt als Open Source zur Verfügung steht, können Sie seine Funktionen ganz nach Ihren Vorstellungen erweitern. Besuchen Sie das Projekt auf GitHub und sehen Sie sich die laufenden Beiträge der Benutzer an, die Dynamo anpassen.

S Features Business Explore Marketplace Pricing This repository Search Sign in Sign up

DynamoDS / Dynamo

Code Issues 630 Pull requests 9 Projects 2 Wiki Insights

Open Source Graphical Programming for Design <http://dynamobim.org>

28,546 commits 70 branches 0 releases 69 contributors

Branch: master New pull request Find file Close or download

File	Description	Last Commit
.github	Fixing a typo about DynamoRevit repo	9 months ago
doc	Upgrade Samples and fix smoke tests (#8715)	2 days ago
extern	LibG Binaries Update (#8695)	8 days ago
src	Merge branch 'master' of https://github.com/DynamoDS/Dynamo into cust...	2 days ago
test	Deserialize Node View IsSetAsInput property on DynamoMode File Open (#...	2 days ago
tools	Show dictionary docs	29 days ago
.gitattributes	One time renormalization of line endings.	5 years ago
.gitignore	Update .gitignore	8 months ago
.gitmodules	checking ssh key	3 years ago
.travis.yml	Update travis	3 years ago
CONTRIBUTING.md	Fixing typo about DynamoRevit	9 months ago
LICENSE.txt	Clarify dependency licenses and their provenance	2 years ago
README.md	Updated pull request link to new wiki page	9 months ago
appveyor.yml	resolved merge conflicts	3 years ago
dynamo-nuget.config	Set nuget dependency version to highest.	2 years ago
dynamo_sublime	Add a sublime text project.	4 years ago
README.md		

BUILD PASSING build passing



# Dynamo

Dynamo is a visual programming tool that aims to be accessible to both non-programmers and programmers alike. It gives users the ability to visually script behavior, define custom pieces of logic, and script using various textual programming languages.

## Get Dynamo

Looking to learn or download Dynamo? Check out [dynamobim.org!](http://dynamobim.org)

## Develop

### Create a Node Library for Dynamo

If you're interested in developing a Node library for Dynamo, the easiest place to start is by browsing the [DynamoSamples](#). These samples use the [Dynamo NuGet packages](#) which can be installed using the NuGet package manager in Visual Studio.

[Documentation of the Dynamo API](#) with a searchable index of public API calls for core functionality. This will be expanded to include regular nodes and Revit functionality.

Durchsuchen, Verzweigen und Erweitern von Dynamo für Ihre Anforderungen

# **Dynamo in Aktion**

## **Dynamo in Aktion**

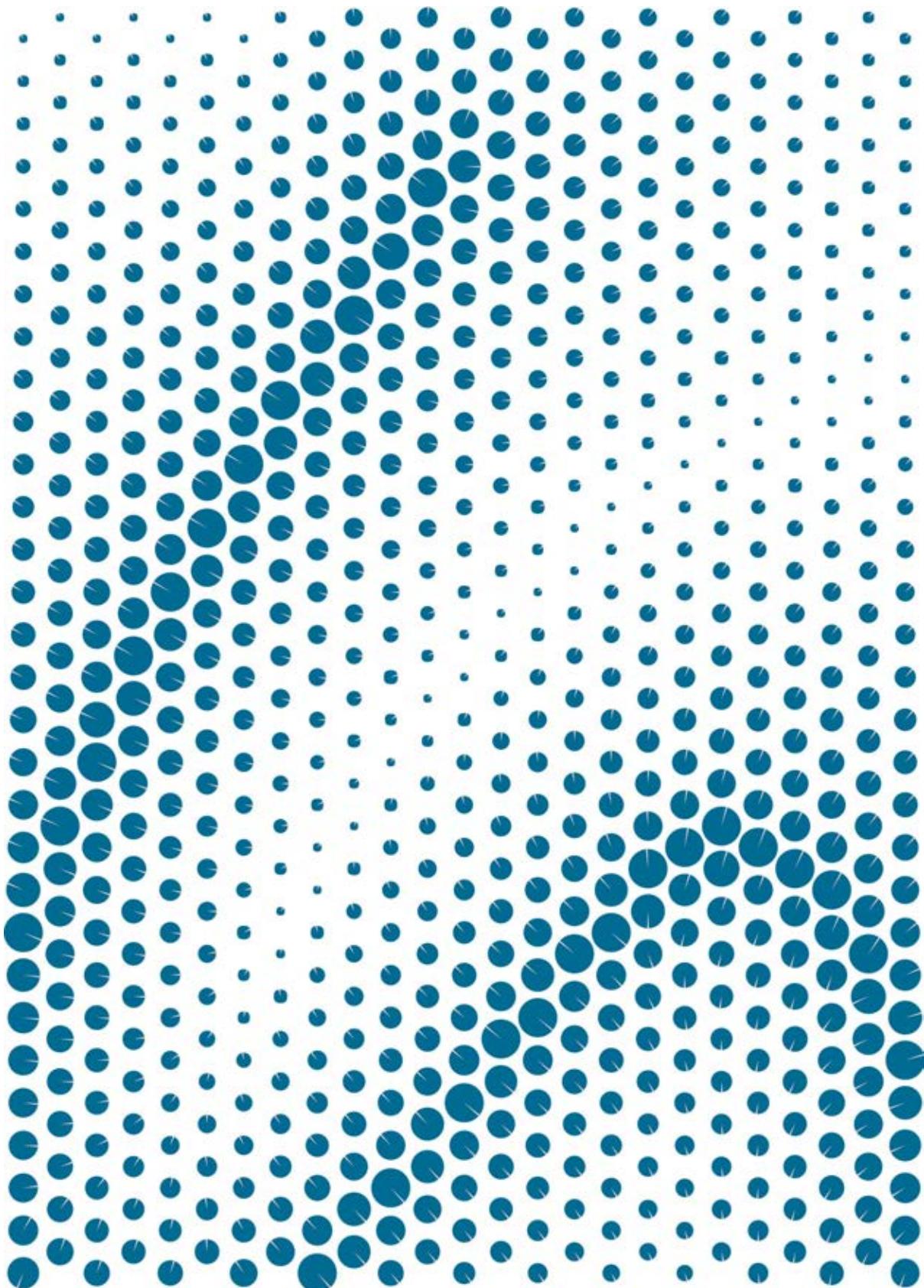
Ob Sie die visuelle Programmierung für Arbeitsabläufe in Projekten nutzen oder eigene Werkzeuge entwickeln: Dynamo ist unverzichtbarer Bestandteil in einer Vielzahl möglicher Anwendungen.

[Folgen Sie dem Dynamo in Action-Board auf Pinterest.](#)

# **Dynamo!**

## **Dynamo!**

Im Grunde genommen handelt es sich bei Dynamo um eine Plattform für die visuelle Programmierung – und damit um ein flexibles und erweiterbares Konstruktionswerkzeug. Da Dynamo sowohl als eigenständige Anwendung als auch als Add-on zu anderer Designsoftware einsetzbar ist, können Sie damit eine breite Palette an kreativen Arbeitsabläufen entwickeln. Installieren Sie Dynamo, und beginnen Sie zunächst damit, sich mit den wichtigsten Funktionen der Benutzeroberfläche vertraut zu machen.



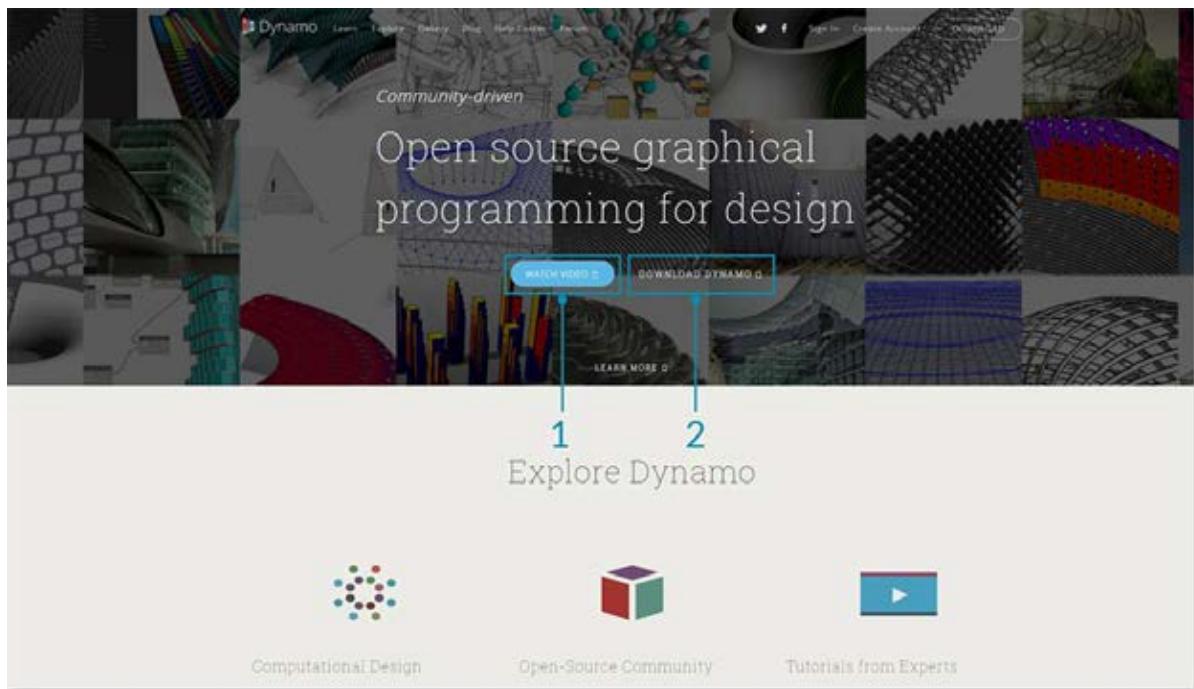
# Dynamo installieren und starten

## Dynamo installieren und starten

Dynamo ist ein aktives Open-Source-Entwicklungsprojekt mit herunterladbaren Installationsprogrammen für offizielle und Vorab-Releases, d. h. "Daily Build". Laden Sie für den Einstieg die offizielle Version herunter, oder tragen Sie aktiv zur Entwicklung von Dynamo bei, indem Sie sich an den Builds des Tages oder dem Github-Projekt beteiligen.

### Herunterladen

Um das offizielle Release von Dynamo herunterzuladen, besuchen Sie die [Dynamo-Website](#). Starten Sie den Download unmittelbar über die Startseite oder durch Navigieren zur entsprechenden Download-Seite.



1. Video zur computergestützten Konstruktion mit Dynamo in der Architektur ansehen
2. Oder zur Download-Seite navigieren

Hier können Sie die aktuellen Entwicklungsversionen herunterladen oder zum [Dynamo Github](#)-Projekt wechseln.



**Dynamo**

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

- ✓ Rapid design iteration and broad interoperability
  - ✓ Lightweight scripting interface
- ✓ Current builds for Autodesk® Revit™ 2016, 2017 and 2018

**1**

Lesson 1.32

**DYNAMO STUDIO**

Autodesk® Dynamo Studio is a visual programming platform that functions fully independently of any other application. Embrace 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

**2**

Version 1.0.0 (Please make sure to update your initial install using the Autodesk Desktop App)

**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**   
Dynamorev2.0.0.20180404170007.exe  
Dynamostudio2.0.0.20180404170007.exe  
Dynamorev2.0.0.20180404171145.exe

**3**

## Node Packages

Packages are user-created extensions for Dynamo that are shared with the community with the Dynamo Package Manager.

1001256  
Package Downloads

1258  
Packages

461  
Authors

**3**

**Get Involved with Open Source**

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

**4**

VISIT THE SOURCE CODE | VISIT THE ISSUE TRACKER

1. Installationsprogramm für das offizielle Release herunterladen
2. Installationsprogramme für Daily Builds herunterladen
3. Benutzerdefinierte Anwendungen aus der Entwickler-Community auschecken
4. Sich an der Entwicklung von Dynamo auf Github beteiligen

## Installieren

Navigieren Sie zum Verzeichnis des heruntergeladenen Installationsprogramms, und führen Sie die ausführbare Datei aus. Während des Installationsprozesses können Sie die Komponenten anpassen, die installiert werden.

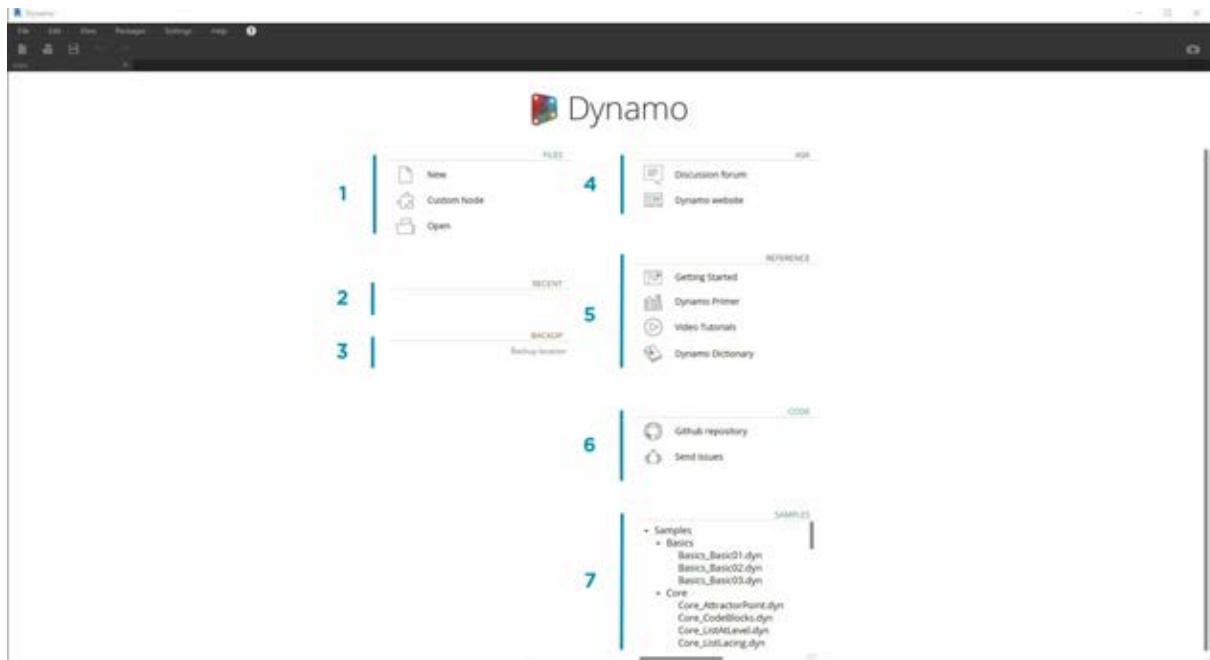


1. Zu installierende Komponenten auswählen

Hier müssen Sie entscheiden, ob Sie die Komponenten einschließen möchten, die Dynamo mit anderen installierten Anwendungen wie Revit verbinden. Weitere Informationen zur Dynamo-Plattform finden Sie in **Kapitel 1.2**.

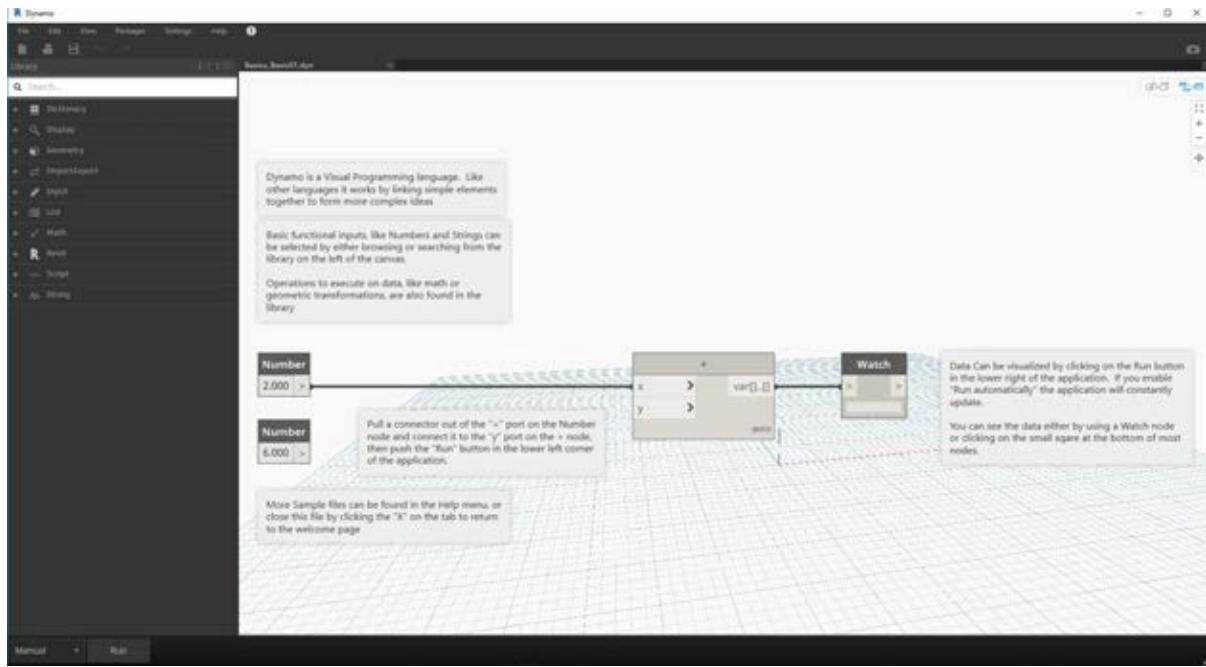
## Starten

Um Dynamo zu starten, navigieren Sie zu \Programme\Dynamic\Dynamic Revit\x.y und wählen Sie dann DynamoSandbox.exe. Dadurch wird die eigenständige Version geöffnet und die *Startseite* von Dynamo angezeigt. Auf dieser Seite werden die Standardmenüs und der Werkzeugkasten sowie eine Reihe von Verknüpfungen angezeigt, die den Zugriff auf Dateifunktionen und zusätzliche Ressourcen ermöglichen.



1. Dateien: Neue Datei erstellen oder vorhandene Datei öffnen
2. Letzte: Liste der zuletzt verwendeten Dateien
3. Backup - Zugriff auf Ihre Sicherungsdateien
4. Fragen: Für direkten Zugriff auf das Benutzerforum oder die Dynamo-Website
5. Referenz: Vertiefung mithilfe zusätzlicher Schulungsressourcen
6. Code: Zur Beteiligung am Open-Source-Entwicklungsprojekt
7. Beispiele: Beispiele austesten, die im Lieferumfang der Installation enthalten sind

Öffnen Sie die erste Beispieldatei, um Ihren ersten Arbeitsbereich zu erstellen und zu bestätigen, dass Dynamo ordnungsgemäß ausgeführt wird. Klicken Sie auf Beispiele > Grundlagen > **Basics\_Basic01.dyn**.



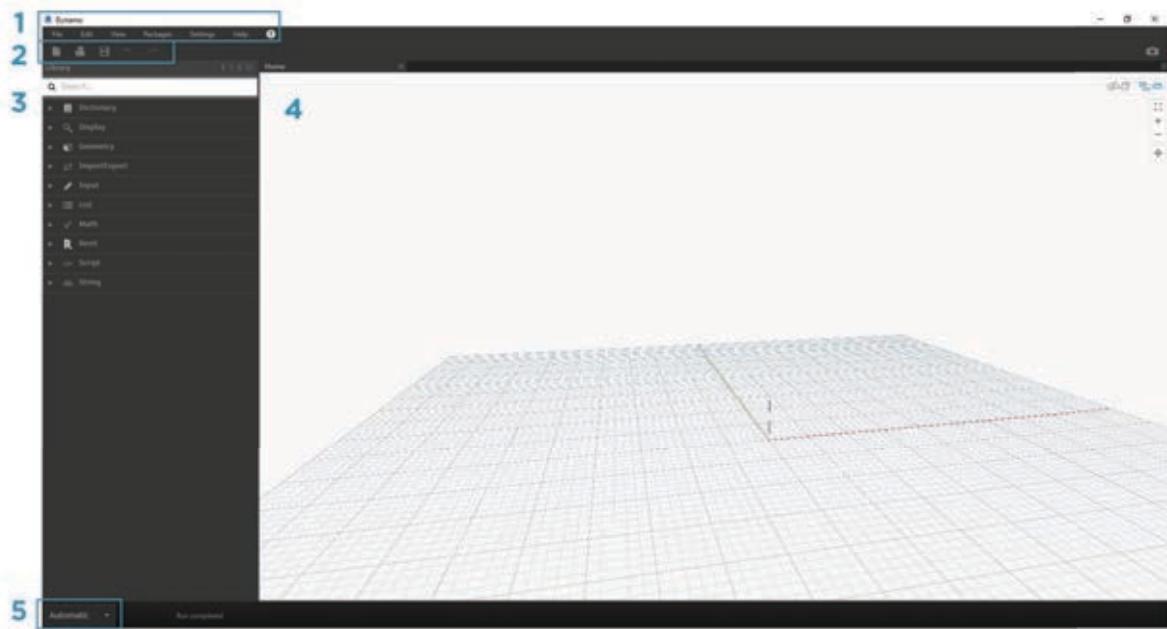
1. Stellen Sie sicher, dass in der Ausführungsleiste "Automatisch" angezeigt wird, oder klicken Sie auf Ausführen.
2. Folgen Sie den Anweisungen und verbinden Sie den Block **Number** mit dem Block **+**.
3. Bestätigen Sie, dass dieser Watch-Block ein Ergebnis anzeigt.

Wenn diese Datei erfolgreich geladen wird, sollten Sie in der Lage sein, Ihr erstes visuelles Programm mit Dynamo auszuführen.

# **Die Benutzeroberfläche von Dynamo**

## **Die Benutzeroberfläche von Dynamo**

Die Benutzeroberfläche (UI) von Dynamo weist fünf Hauptbereiche auf, von denen der größte der Arbeitsbereich ist, in dem Sie Ihre visuellen Programme erstellen.

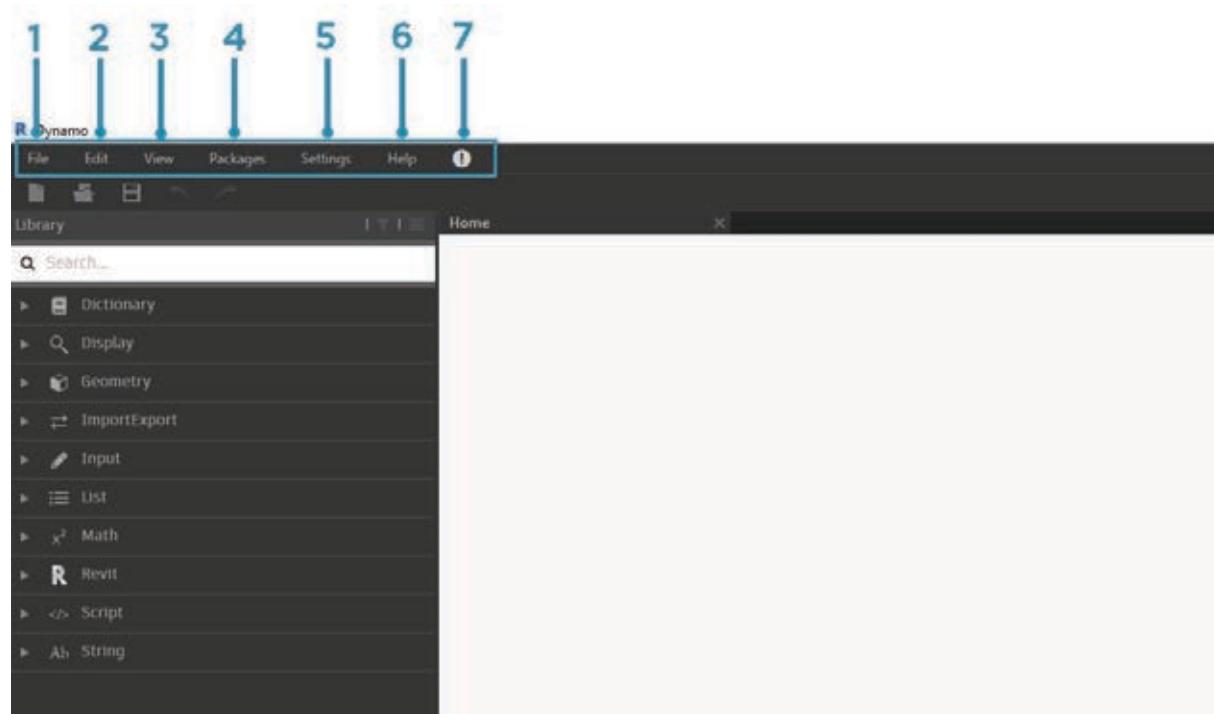


1. Menüs
2. Werkzeugkasten
3. Bibliothek
4. Arbeitsbereich
5. Ausführungsleiste

Machen Sie sich genauer mit der Benutzeroberfläche vertraut, und untersuchen Sie die Funktionalität der einzelnen Bereiche.

## Menüs

In den Dropdown-Menüs können Sie einige der grundlegenden Funktionen der Dynamo-Anwendung aufrufen. Wie in fast jeder anderen Windows-Software finden Sie die Aktionen zum Verwalten von Dateien sowie Operationen zum Auswählen und Bearbeiten von Inhalten in den ersten beiden Menüs. Die übrigen Menüs sind spezifisch für Dynamo.

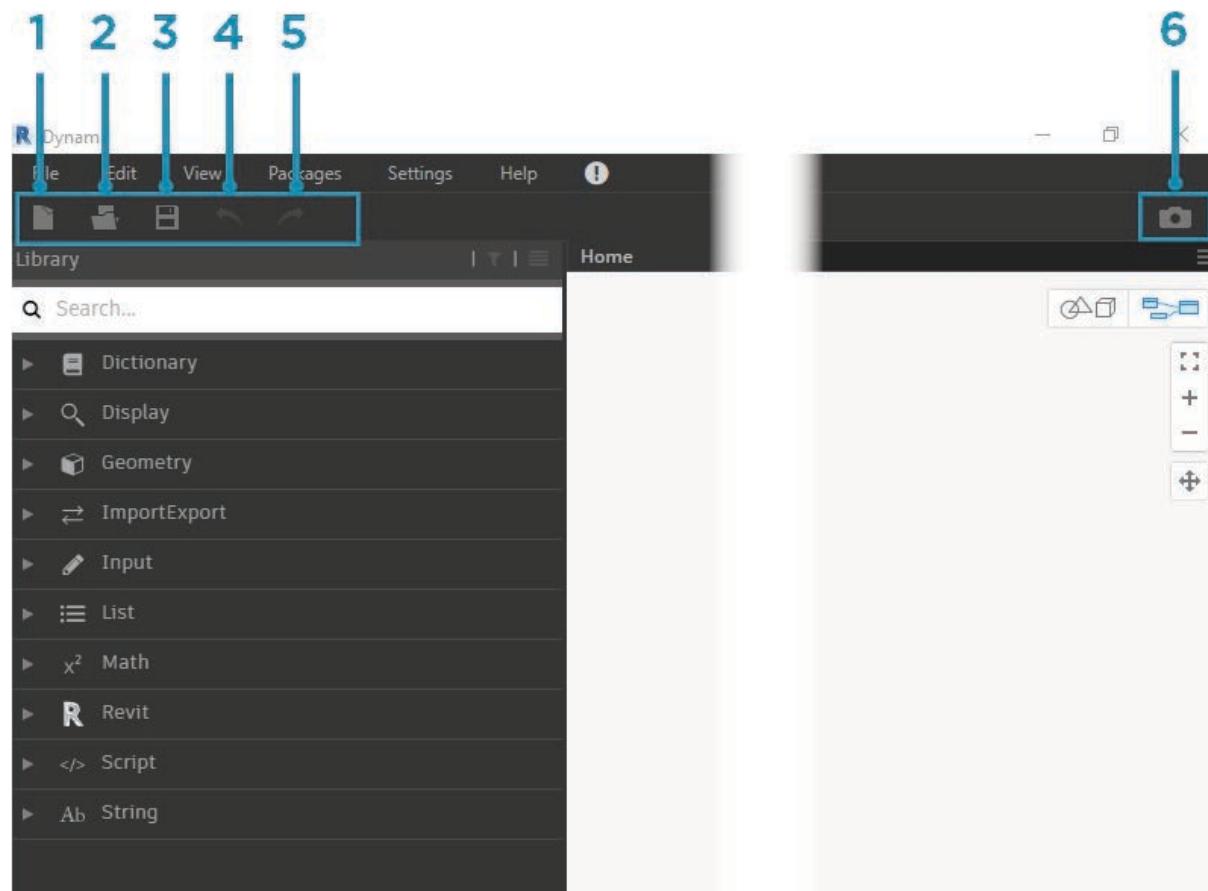


1. Datei
2. Bearbeiten

3. Ansicht
4. Pakete
5. Einstellungen
6. Hilfe
7. Benachrichtigungen

## Werkzeugkasten

Der Werkzeugkasten von Dynamo enthält eine Reihe von Schaltflächen für den Schnellzugriff zum Arbeiten mit Dateien sowie die Befehle Rückgängig [Ctrl + Z] und Wiederholen [Ctrl + Y]. Ganz rechts befindet sich eine weitere Schaltfläche, über die Sie einen Snapshot des Arbeitsbereichs exportieren können. Dies ist für die Dokumentation und die gemeinsame Bearbeitung mit anderen äußerst nützlich.



1. Neu: Neue .dyn-Datei erstellen
2. Öffnen: Vorhandene .dyn-Datei (Arbeitsbereich) oder .dyf-Datei (benutzerdefinierter Block) öffnen
3. Speichern/Speichern unter: Aktive .dyn- oder .dyf-Datei speichern
4. Rückgängig: Die letzte Aktion rückgängig machen
5. Wiederholen: Die nächste Aktion wiederherstellen
6. Arbeitsbereich als Bild exportieren: Den angezeigten Arbeitsbereich als PNG-Datei exportieren

## Bibliothek

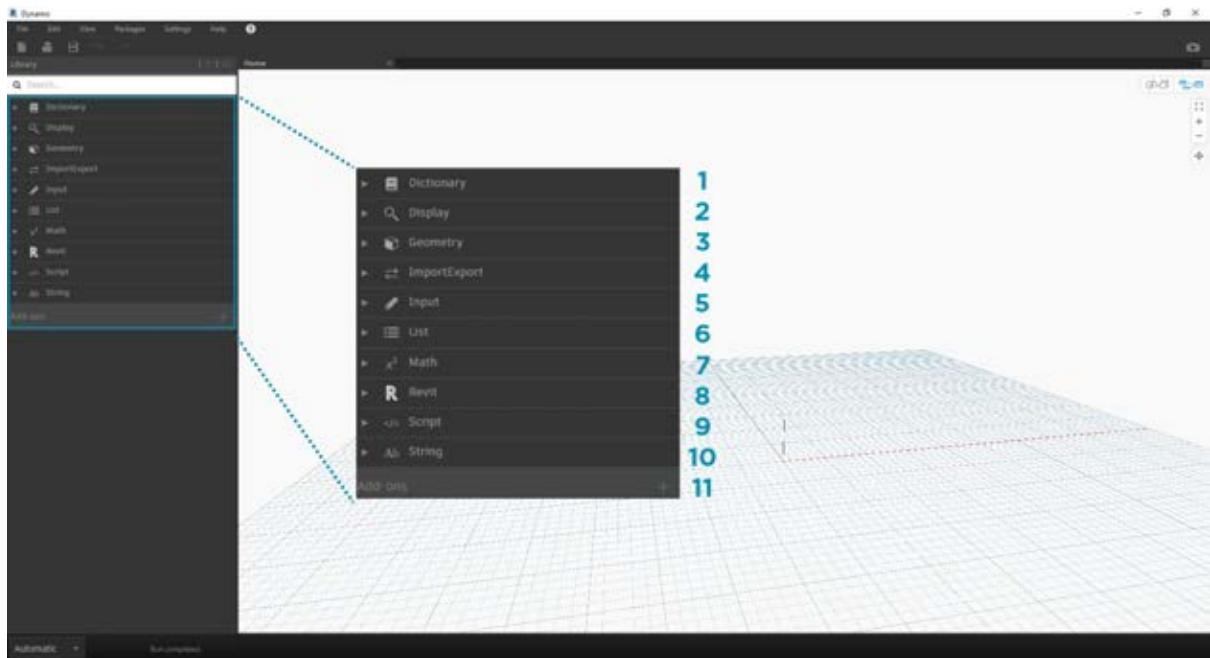
Die Bibliothek enthält alle geladenen Blöcke, einschließlich der vorgabemäßigen Blöcke, die zum Lieferumfang gehören, sowie der zusätzlich geladenen benutzerdefinierten Blöcke und Pakete. Die Blöcke in der Bibliothek sind in Abhängigkeit davon, ob die Blöcke Daten **erstellen**, eine **Aktion** ausführen oder Daten **abfragen**, hierarchisch in Bibliotheken, Kategorien und ggf. Unterkategorien geordnet.

### Durchsuchen

Standardmäßig enthält die **Bibliothek** acht Kategorien von Blöcken. Am besten untersuchen Sie zunächst die Kategorien **Core** und **Geometry**, da sie die größte Anzahl an Blöcken enthalten. Das Durchsuchen dieser Kategorien stellt die schnellste Möglichkeit dar, um die Hierarchie dessen zu verstehen, was Sie zu Ihrem Arbeitsbereich hinzufügen können, und um neue

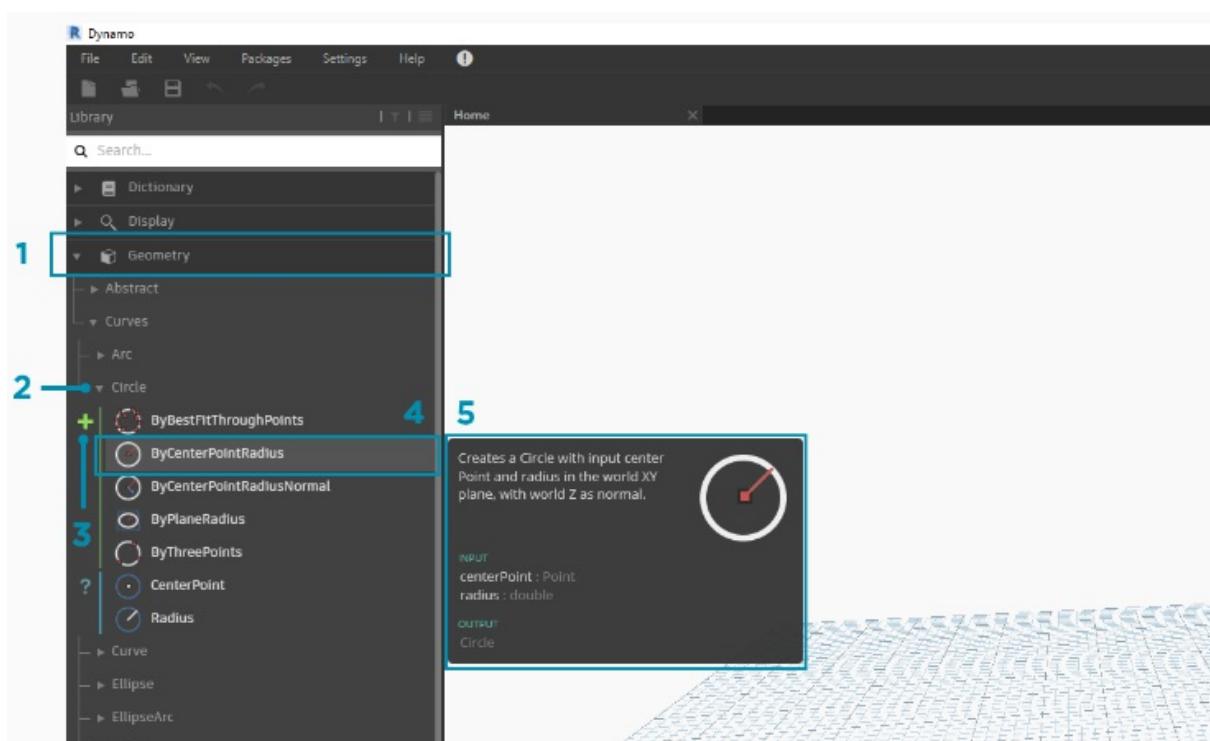
Blöcke zu entdecken, die Sie niemals zuvor verwendet haben.

Konzentrieren Sie sich zunächst auf die Standardsammlung an Blöcken. Beachten Sie, dass Sie diese Bibliothek später um benutzerdefinierte Blöcke, zusätzliche Bibliotheken und den Package Manager erweitern werden.



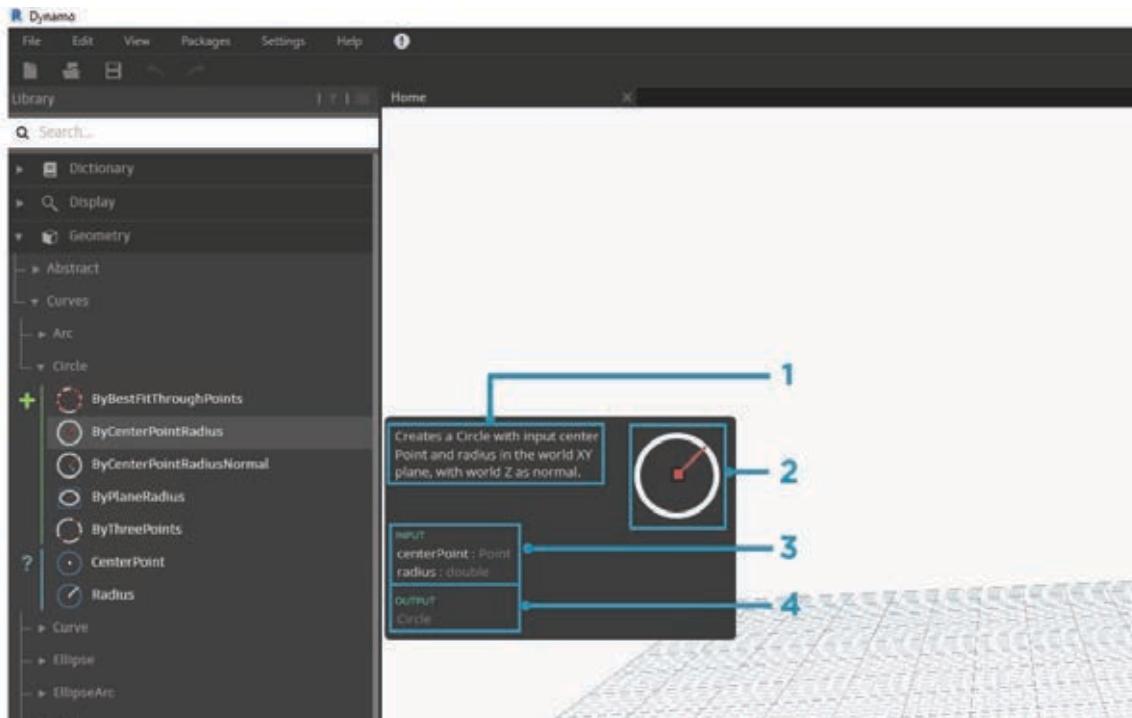
1. Wörterbuch
2. Anzeige
3. Geometrie
4. Import/Export
5. Eingabeblocks (Input)
6. Liste
7. +Findet
8. Revit
9. Skript
10. Zeichenfolge
11. Add-ons

Durchsuchen Sie die Bibliothek, indem Sie durch die Menüs klicken. Klicken Sie auf Geometry > Curves > Circle. Beachten Sie den neuen Abschnitt des Menüs, der eingeblendet wird, insbesondere die Bezeichnungen **Erstellen** und **Abfrage**.



1. Bibliothek
2. Kategorie
3. Unterkategorie: Erstellen/Aktionen/Abfrage
4. Punkt
5. Blockbeschreibung und -eigenschaften: Wird angezeigt, wenn Sie den Cursor auf das Blocksymbol bewegen.

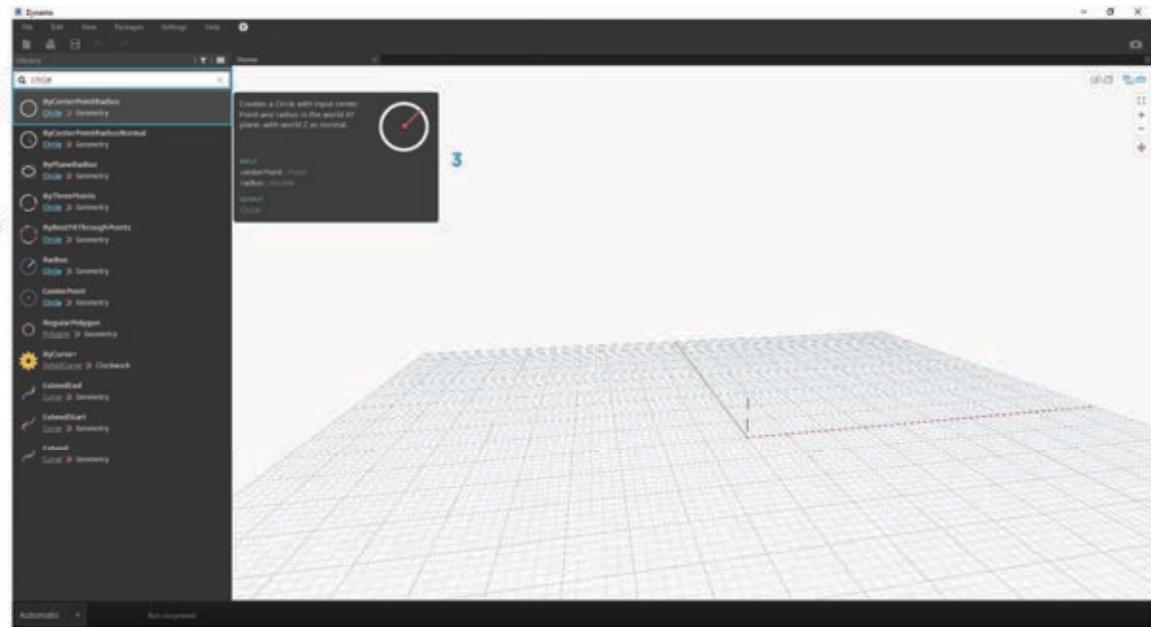
Bewegen Sie den Cursor im Menü Circle auf **ByCenterPointRadius**. Das daraufhin angezeigte Fenster enthält über den Namen und das Symbol hinaus noch weitere detaillierte Informationen zu dem Block. Dadurch können Sie schnell nachvollziehen, welche Aktion der Block ausführt, welche Eingaben erforderlich sind und was von dem Block ausgegeben wird.



1. Beschreibung: Kurze Beschreibung des Blocks
2. Symbol: Größere Version des Symbols im Menü Bibliothek
3. Eingabe(n): Name, Datentyp und Datenstruktur
4. Ausgabe(n): Datentyp und Struktur

#### Suchen

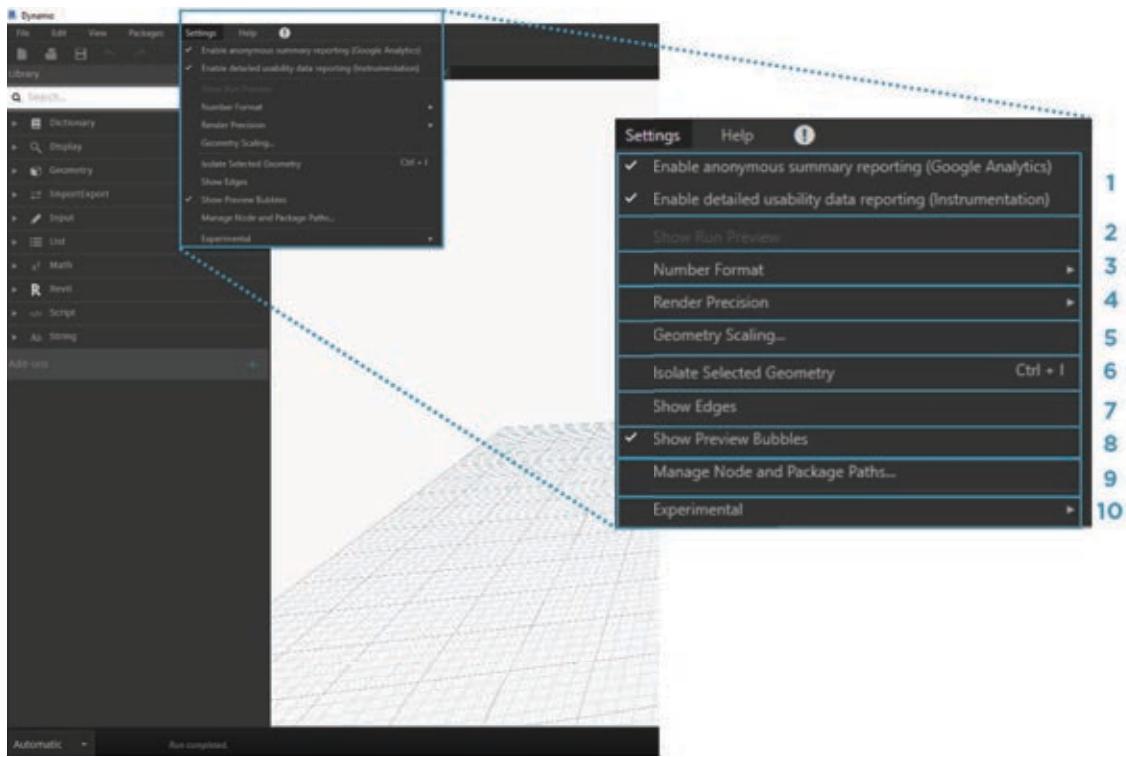
Wenn Sie relativ genau wissen, welchen Block Sie zu Ihrem Arbeitsbereich hinzufügen möchten, können Sie das Feld **Suchen** verwenden. Solange Sie keine Einstellungen bearbeiten oder Werte im Arbeitsbereich angeben, befindet sich der Cursor in diesem Feld. Sobald Sie etwas in das Feld eingeben, werden in der Dynamo-Bibliothek die beste Übereinstimmung (mit Breadcrumbs dafür, wo der Suchbegriff in den Blockkategorien gefunden werden kann) und eine Liste alternativer Übereinstimmungen der Suche angezeigt. Wenn Sie die Eingabetaste drücken oder im eingeschränkten Browser auf das Element klicken, wird der hervorgehobene Block in der Mitte des Arbeitsbereichs hinzugefügt.



1. Suchfeld
2. Am besten übereinstimmendes Ergebnis/Ausgewählt
3. Alternative Übereinstimmungen

## Einstellungen

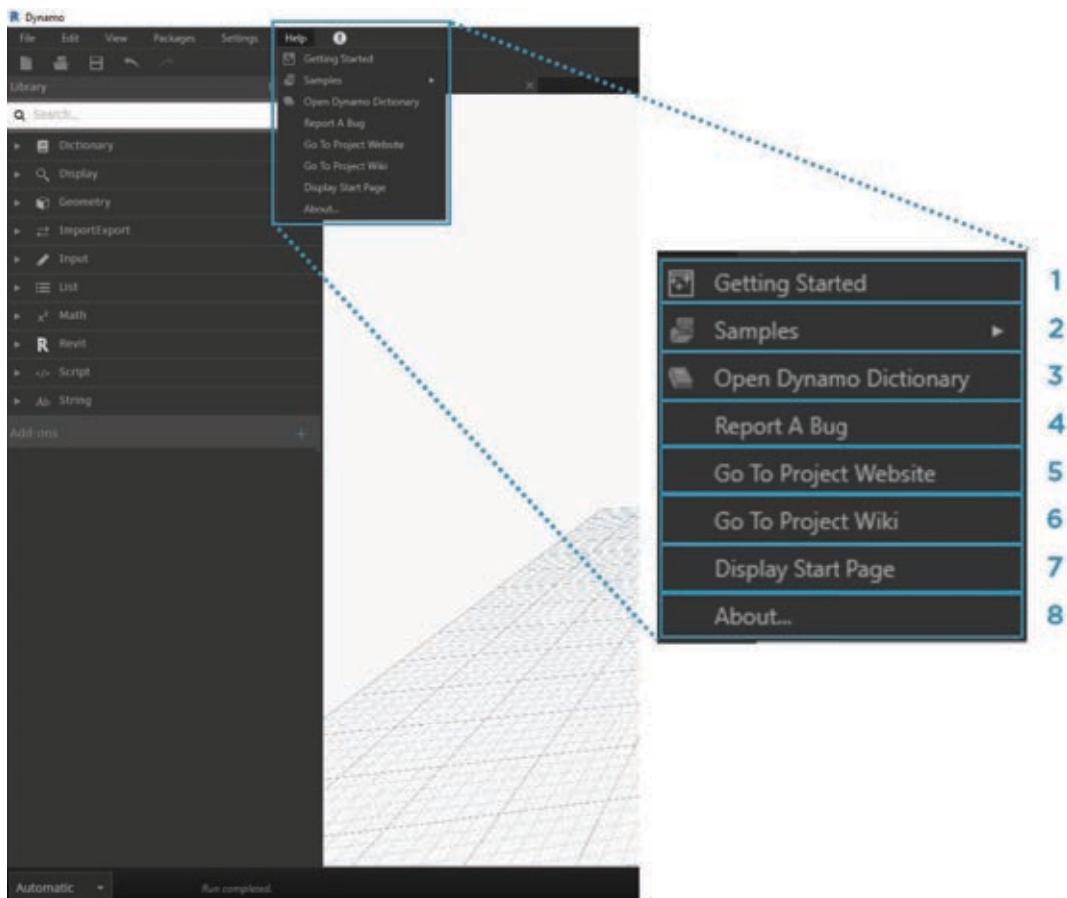
Im Menü **Einstellungen** sind sowohl geometrische als auch Benutzereinstellungen verfügbar. Hier können Sie auch die Freigabe Ihrer Benutzerdaten zur Verbesserung von Dynamo aktivieren bzw. deaktivieren sowie die Anzahl an Dezimalstellen und die Renderqualität der Geometrie definieren.



1. Berichte aktivieren: Optionen zur Weitergabe von Benutzerdaten für die Verbesserung von Dynamo.
2. Vorschau von Ausführung anzeigen: Zeigt den Ausführungszustand des Diagramms als Vorschau an. Blöcke, die zur Ausführung geplant sind, werden im Diagramm hervorgehoben.
3. Optionen für das Zahlenformat: Ändern der Dokumenteneinstellungen für Dezimalstellen.
4. Rendergenauigkeit: Einstellen einer höheren oder niedrigen Renderqualität.
5. Skalierung für Geometrie: Bereich der Geometrie, an der Sie gerade arbeiten.
6. Ausgewählte Geometrie isolieren: Hintergrundgeometrie basierend auf den Blöcken auswählen.
7. Geometriekanten anzeigen/ausblenden: Ein- oder Ausblenden von 3D-Geometriekanten.
8. Vorschaufenster anzeigen/ausblenden: Ein- oder Ausblenden der Datenvorschaufenster.
9. Pfade für Blöcke und Pakete verwalten: Verwalten der Dateipfade, damit Blöcke und Pakete in der Bibliothek angezeigt werden.
10. Experimentelle Funktionen aktivieren: Verwenden neuer Beta-Funktionen in Dynamo.

## Hilfe

Wenn Sie nicht weiterkommen, verwenden Sie das Menü **Hilfe**. Hier finden Sie die Beispieldateien, die zum Lieferumfang Ihrer Installation gehören, und können in Ihrem Internet-Browser auf eine der Referenz-Sites von Dynamo zugreifen. Falls erforderlich, können Sie über die Option **Info** überprüfen, welche Version von Dynamo installiert ist und ob sie aktuell ist.



- 1 Getting Started
- 2 Samples
- 3 Open Dynamo Dictionary
- 4 Report A Bug
- 5 Go To Project Website
- 6 Go To Project Wiki
- 7 Display Start Page
- 8 About...

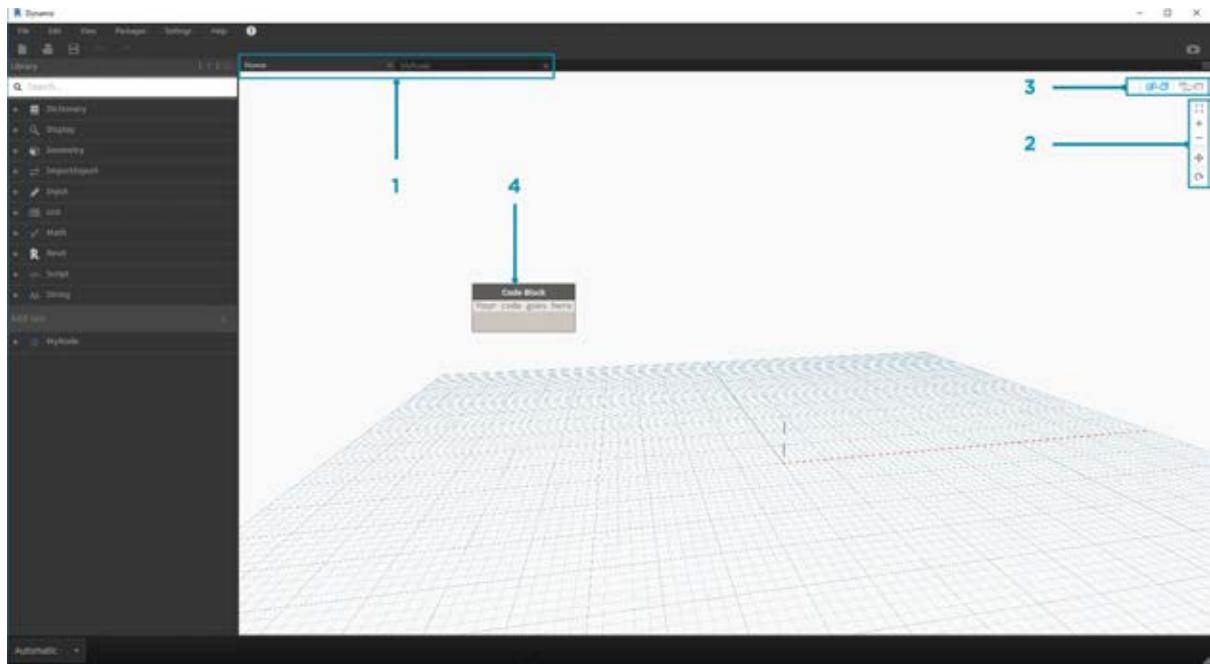
1. Erste Schritte: eine kurze Einführung in die Verwendung von Dynamo.
2. Beispiele: Beispieldateien als Referenz.
3. Dynamo-Wörterbuch öffnen: Ressource mit Dokumentation für alle Blöcke.
4. Fehler melden: Melden Sie ein Problem auf GitHub.
5. Wechseln zu Projekt-Website: Zeigen Sie das Dynamo-Projekt auf GitHub an.
6. Zu Projekt-Wiki wechseln: Im Wiki erhalten Sie Entwicklungsinformationen mithilfe der Dynamo-API, unterstützenden Bibliotheken und Tools.
7. Startseite anzeigen: Kehren Sie von einem Dokument aus zur Dynamo-Startseite zurück.
8. Info: Angaben zur Version von Dynamo.

# **Der Arbeitsbereich**

## **Der Arbeitsbereich**

Der **Dynamo-Arbeitsbereich** ist der Bereich, in dem Sie Ihre visuellen Programme entwickeln und eine Vorschau der resultierenden Geometrie anzeigen. Unabhängig davon, ob Sie im Start-Arbeitsbereich oder einem benutzerdefinierten Block arbeiten, können Sie mit der Maus oder über die Schaltflächen rechts oben navigieren. Über die Schaltfläche rechts unten können Sie den Vorschau-Modus umschalten, in dem Sie navigieren.

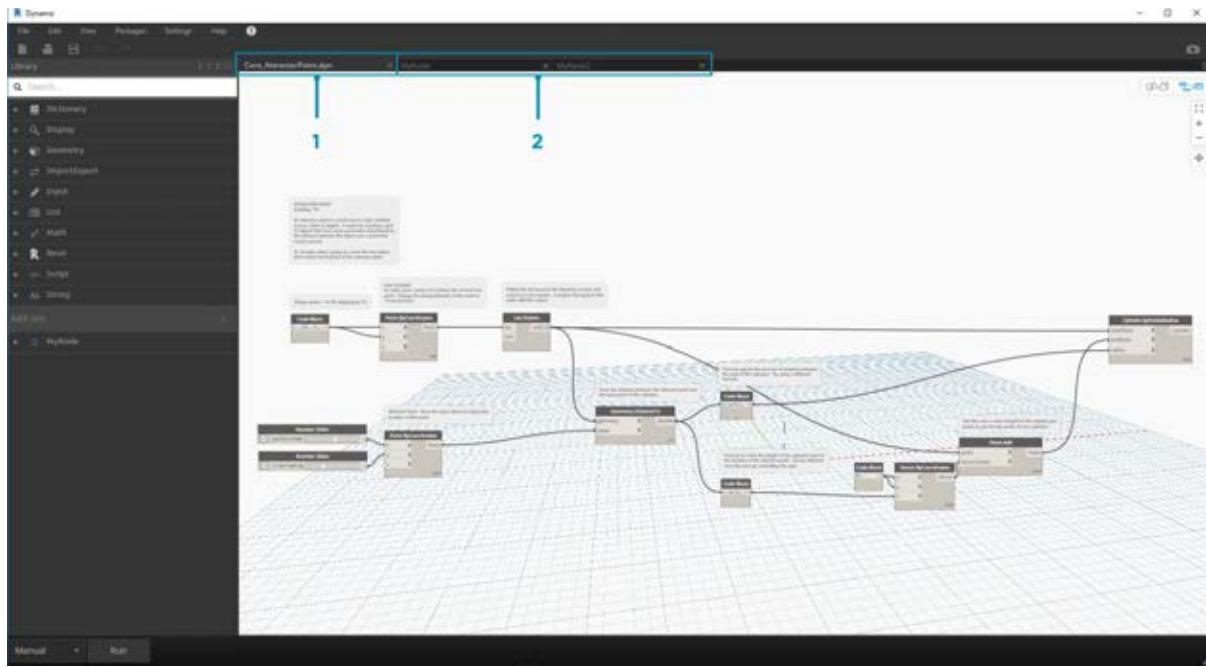
Hinweis: Die Blöcke und Geometrie weisen eine Zeichnungsreihenfolge auf, sodass Objekte möglicherweise übereinander gerendert werden. Dies kann unübersichtlich sein, wenn mehrere Blöcke nacheinander hinzugefügt werden, da sie möglicherweise im Arbeitsbereich in derselben Position gerendert werden.



1. Registerkarten
2. Schaltflächen Zoom/Schwenken
3. Vorschaumodus
4. In den Arbeitsbereich doppelklicken

## Registerkarten

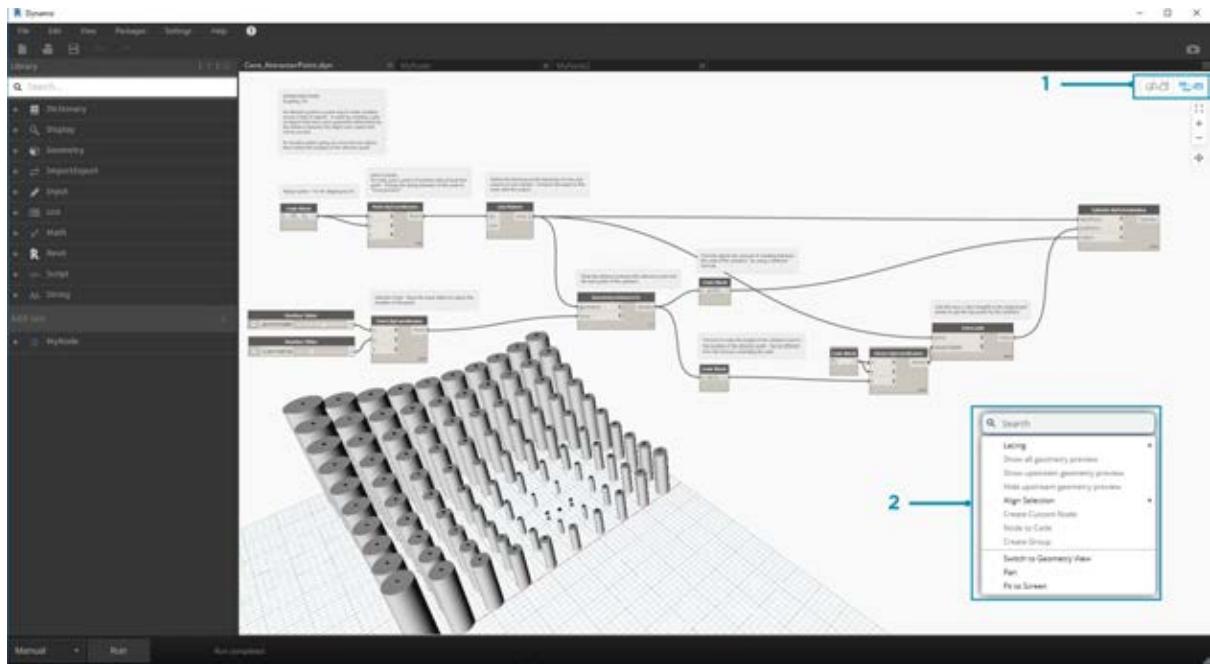
Auf der aktiven Registerkarte des Arbeitsbereichs können Sie durch Ihr Programm navigieren und es bearbeiten. Beim Öffnen einer neuen Datei öffnen Sie standardmäßig einen neuen **Start**-Arbeitsbereich. Sie können auch einen neuen Arbeitsbereich für **benutzerdefinierte Blöcke** über das Menü Datei oder über die Kontextmenüoption *Neuer Block aus Auswahl* öffnen, wenn Blöcke ausgewählt sind (weitere Informationen zu dieser Funktion weiter unten).



Hinweis: Es kann jeweils nur ein Start-Arbeitsbereich geöffnet sein. Sie können daneben jedoch gleichzeitig mehrere Arbeitsbereiche für benutzerdefinierten Block auf zusätzlichen Registerkarten öffnen.

### Navigation in Diagrammen und in der 3D-Vorschau im Vergleich

In Dynamo werden sowohl das Diagramm als auch die 3D-Ergebnisse des Diagramms (wenn Sie Geometrie erstellen) im Arbeitsbereich gerendert. Standardmäßig ist das Diagramm die aktive Vorschau. Sie können das Diagramm durchlaufen, indem Sie es mithilfe der Navigationstasten oder der mittleren Maustaste schwenken oder zoomen. Zum Umschalten zwischen den aktiven Voransichten sind drei Möglichkeiten verfügbar:

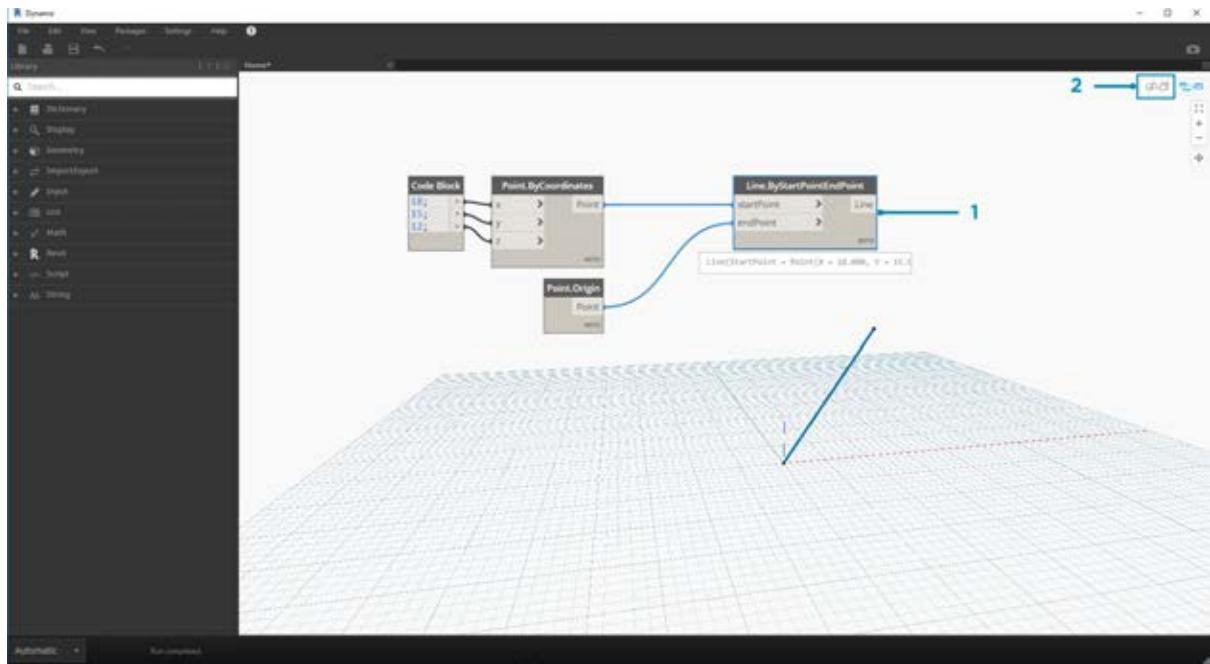


1. Schaltflächen zum Umschalten der Vorschau im Arbeitsbereich
2. Mit der rechten Maustaste in den Arbeitsbereich klicken und *Zu ...ansicht wechseln* auswählen
3. Tasturbefehl (Strg + B)

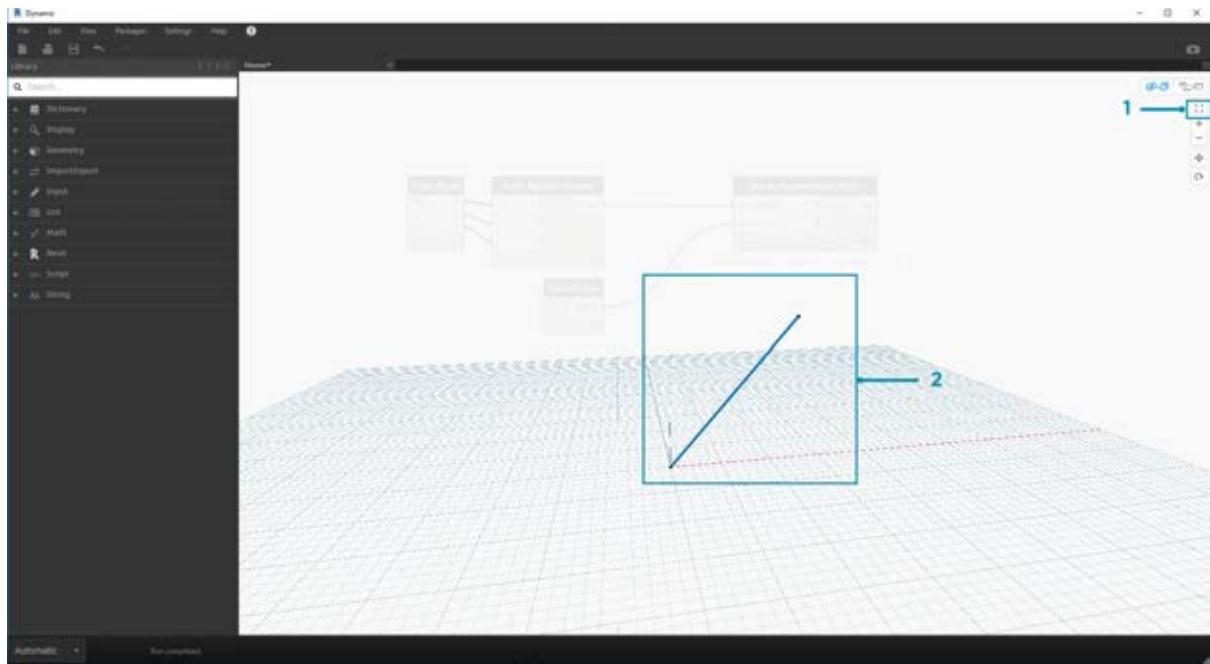
Der Modus Navigation in 3D-Vorschau bietet uns auch die Möglichkeit zur **Direktbearbeitung** von Punkten, wie im Abschnitt [Erste Schritte](#) veranschaulicht wird.

### **Zoomen und neu zentrieren**

Im Modus Navigation in 3D-Vorschau können Sie mühelos schwenken, zoomen und drehen. Sie können jedoch gezielt auf das von einem bestimmten Geometrieblock erstellte Objekt einzoomen, indem Sie den betreffenden Block auswählen und dann das Symbol Zoom alles verwenden.



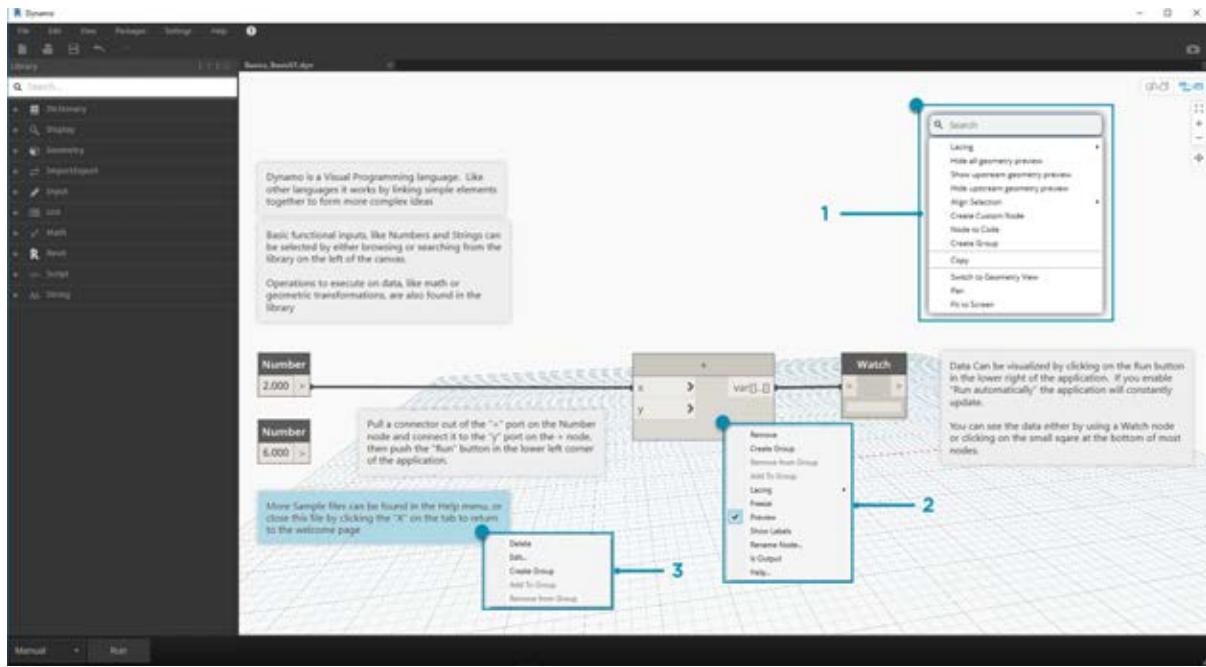
1. Wählen Sie den Block für die Geometrie aus, die in der Ansicht zentriert werden soll.
2. Wechseln Sie zur Navigation in der 3D-Vorschau.



1. Klicken Sie auf das Symbol Zoom alles in der rechten oberen Ecke.
2. Die ausgewählte Geometrie wird in der Ansicht zentriert.

## Mausnavigation

Ihre Maustasten verhalten sich je nach dem aktiven Vorschaumodus unterschiedlich. Im Allgemeinen können Sie durch Klicken mit der linken Maustaste Eingaben auswählen und angegeben, durch Klicken mit der rechten Maustaste auf Optionen zugreifen und durch Klicken mit der mittleren Maustaste durch den Arbeitsbereich navigieren. Die Optionen, die durch Klicken mit der rechten Maustaste angezeigt werden, basieren auf dem Kontext, in dem Sie geklickt haben.



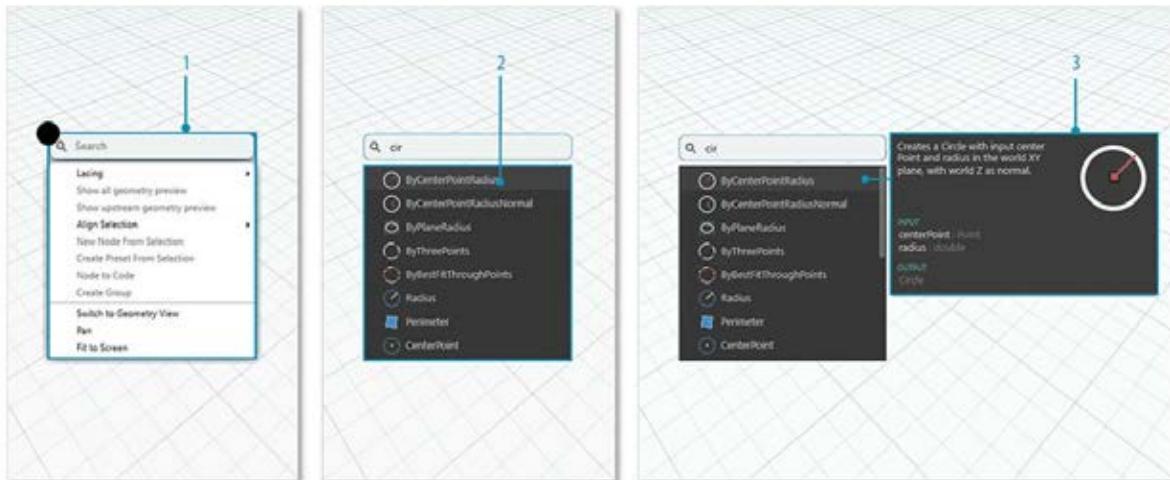
1. Mit der rechten Maustaste in den Arbeitsbereich klicken
2. Mit der rechten Maustaste auf einen Block klicken
3. Mit der rechten Maustaste auf eine Anmerkung klicken

Der folgenden Tabelle können Sie die Mausinteraktionen je nach Vorschau entnehmen:

Mausaktion	Diagrammvorschau	3D-Vorschau
Mit der linken Maustaste klicken	Auswählen	-
Mit der rechten Maustaste klicken	Kontextmenü	Zoomoptionen
Mit der mittleren Maustaste klicken	Schwenken	Schwenken
Bildlauf	Vergrößern/Verkleinern	Vergrößern/Verkleinern
Doppelklicken	Codeblock erstellen	-

## Suche im Ansichtsbereich

Durch Einsatz der Funktion "Suche im Ansichtsbereich" können Sie die Geschwindigkeit Ihres Dynamo-Workflows erheblich steigern, indem Sie Zugriff auf Blockbeschreibungen und QuickInfos erhalten, ohne dass Sie die Position in Ihrem Diagramm verlassen müssen. Durch einfaches Klicken mit der rechten Maustaste können Sie von jeder beliebigen Position im Ansichtsbereich, an der Sie gerade arbeiten, auf alle nützlichen "Suchfunktionen der Bibliothek" zugreifen.



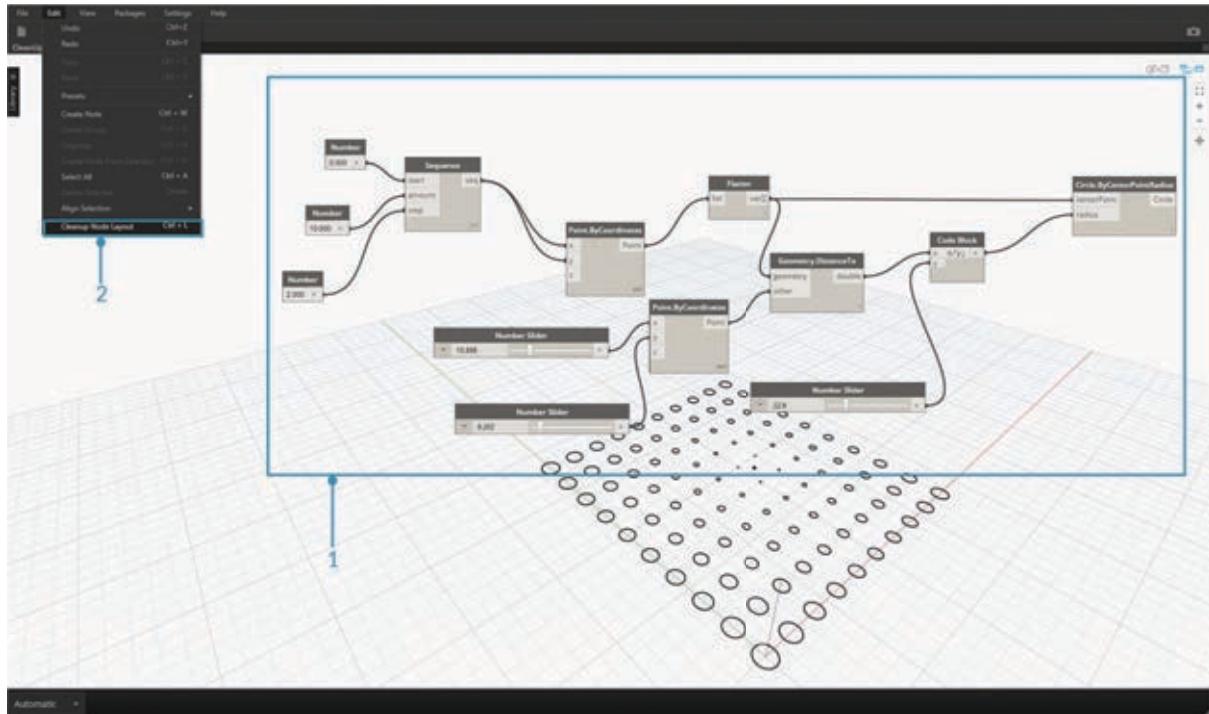
1. Klicken Sie mit der rechten Maustaste auf eine beliebige Stelle im Ansichtsbereich, um die Suchfunktion

- aufzurufen. Wenn die Suchleiste leer ist, werden im Dropdown-Menü Voransichten angezeigt.
2. Während Sie einen Suchbegriff in die Suchleiste eingeben, wird das Dropdown-Menü ständig aktualisiert, um die relevantesten Suchergebnisse anzuzeigen.
  3. Bewegen Sie den Cursor auf die Suchergebnisse, um die zugehörigen Beschreibungen und QuickInfos anzuzeigen.

## Bereinigen von Blocklayouts

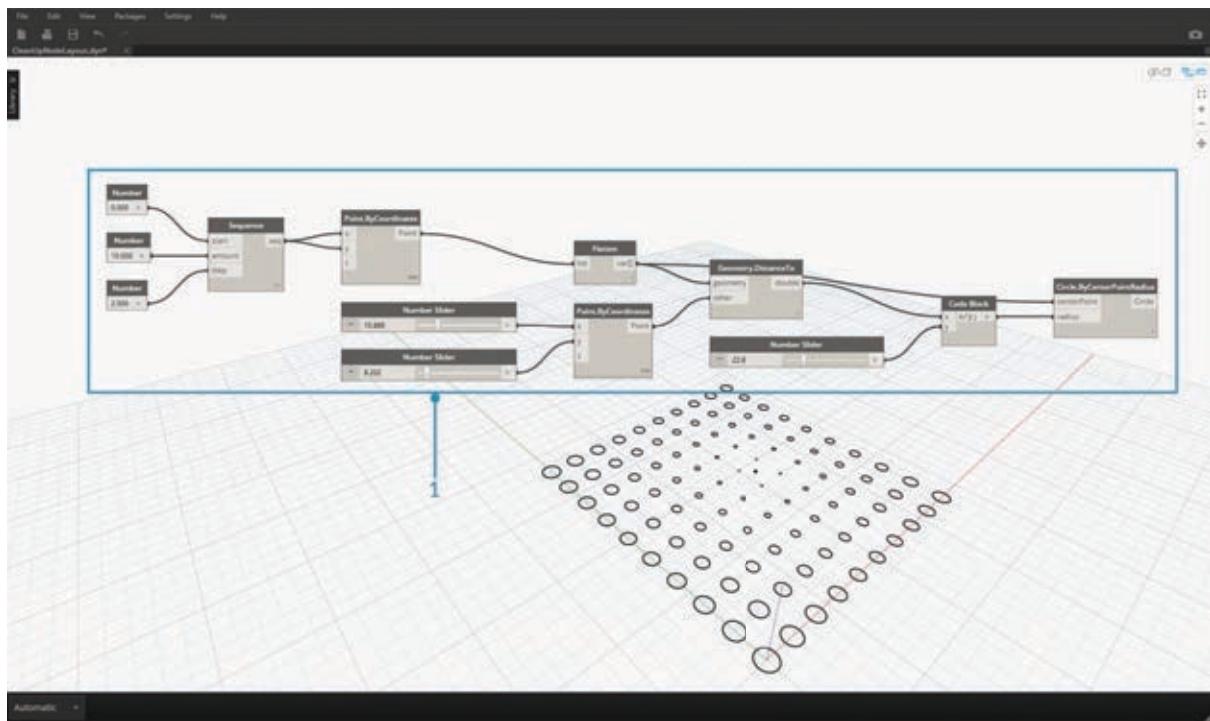
Das Organisieren Ihres Dynamo-Ansichtsbereichs wird zunehmend wichtig, je komplexer Ihre Dateien werden. Neben dem Werkzeug **Auswahl ausrichten** zum Arbeiten mit einer kleinen Anzahl an ausgewählten Blöcken ist in Dynamo auch das Werkzeug **Blocklayout bereinigen** verfügbar, das Sie generell bei der Dateibereinigung unterstützt.

### Vor der Blockbereinigung



1. Aktivieren Sie die Blöcke, die automatisch organisiert werden sollen, oder lassen Sie alle Blöcke deaktiviert, um alle Blöcke in der Datei zu bereinigen.
2. Die Funktion Blocklayout bereinigen befindet sich unter der Registerkarte Bearbeiten.

### Nach der Blockbereinigung



1. Die Blöcke werden automatisch neu verteilt und ausgerichtet, indem gestapelte oder überlappende Blöcke bereinigt und mit benachbarten Blöcken ausgerichtet werden.

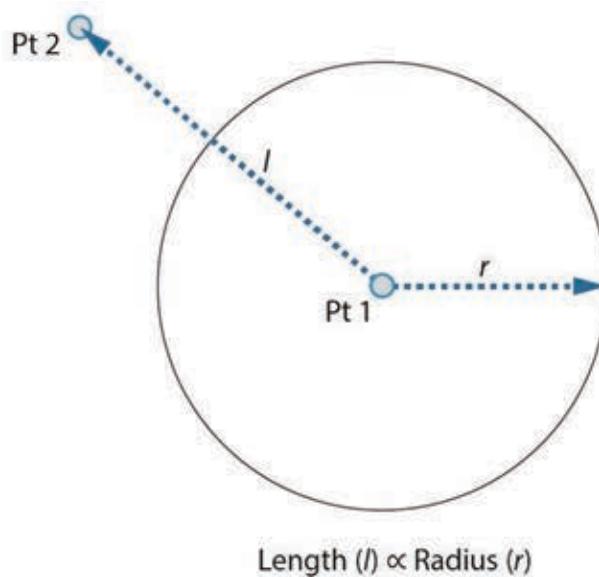
# ERSTE SCHRITTE

## ERSTE SCHRITTE

Nachdem Sie sich mit dem Layout der Benutzeroberfläche und dem Navigieren im Arbeitsbereich vertraut gemacht haben, wird in diesem Abschnitt ein typischer Arbeitsablauf für die Entwicklung eines Diagramms in Dynamo erläutert. Sie erstellen zunächst einen Kreis mit einer dynamischen Größe und dann eine Reihe von Kreisen mit unterschiedlichen Radien.

### Ziele und Beziehungen definieren

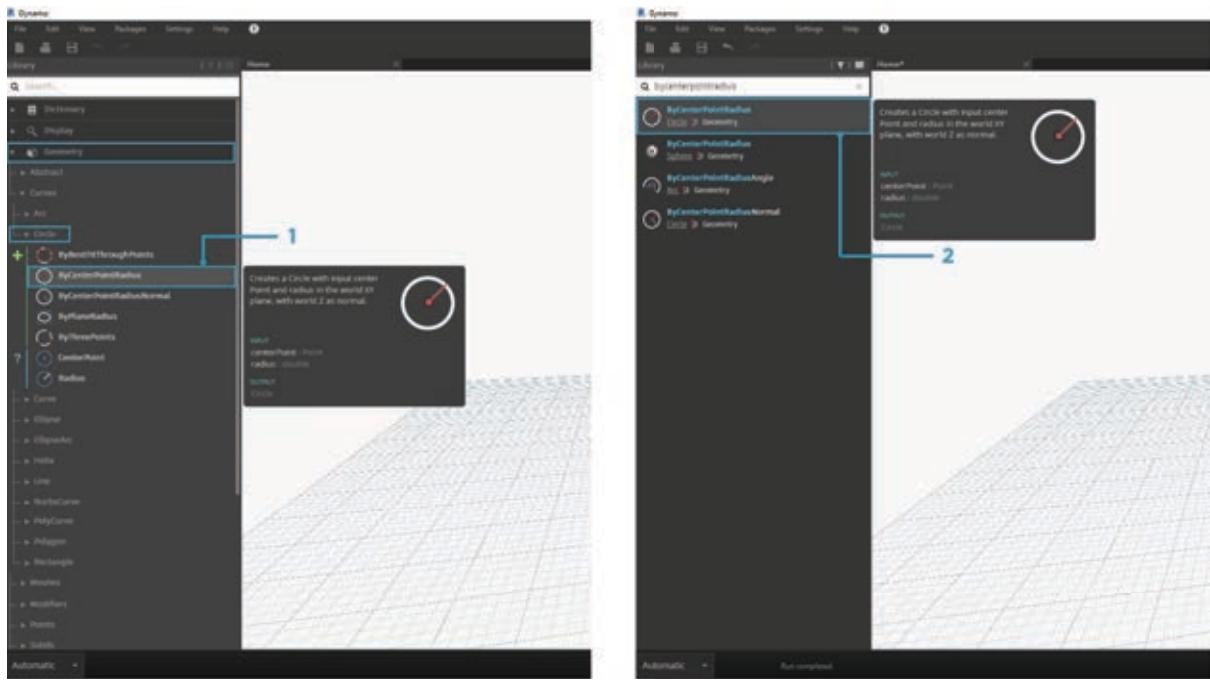
Bevor Sie etwas zum Dynamo-Arbeitsbereich hinzufügen, ist es von grundlegender Bedeutung, ein klares Verständnis dafür zu entwickeln, was erreicht werden soll und welche Beziehungen dafür wichtig sind. Denken Sie daran, dass bei jedem Verbinden von zwei Blöcken eine explizite Verknüpfung zwischen ihnen erstellt wird. Sie können den Datenfluss zu einem späteren Zeitpunkt ändern, aber sobald eine Verbindung hergestellt wurde, ist die Beziehung festgelegt. In dieser Übung erstellen Sie einen Kreis (*Ziel*), dessen Radius durch die Entfernung zu einem nahegelegenen Punkt (*Beziehung*) definiert wird.



Ein Punkt, der eine entfernungsabhängige Beziehung definiert, wird häufig als "Attraktor" bezeichnet. In diesem Beispiel wird die Entfernung zum Attraktorpunkt verwendet, um anzugeben, wie groß der Kreis sein soll.

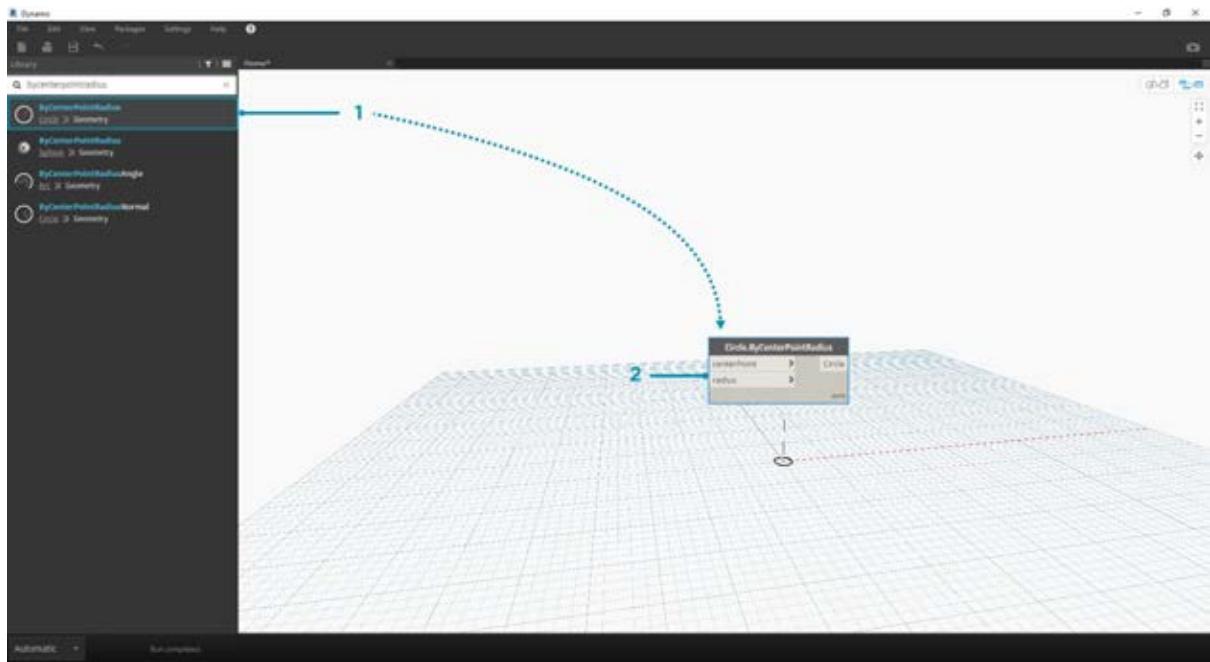
### Blöcke zum Arbeitsbereich hinzufügen

Nachdem Sie die Ziele und Beziehungen skizziert haben, können Sie mit dem Erstellen des Diagramms beginnen. Sie benötigen die Blöcke, die die Reihenfolge der Aktionen darstellen, die von Dynamo ausgeführt werden. Da Sie einen Kreis erstellen möchten, suchen Sie nach einem Block, der diese Aktion ausführt. Über das Suchfeld oder durch Durchsuchen der Bibliothek werden Sie feststellen, dass es zum Erstellen eines Kreises mehrere Möglichkeiten gibt.



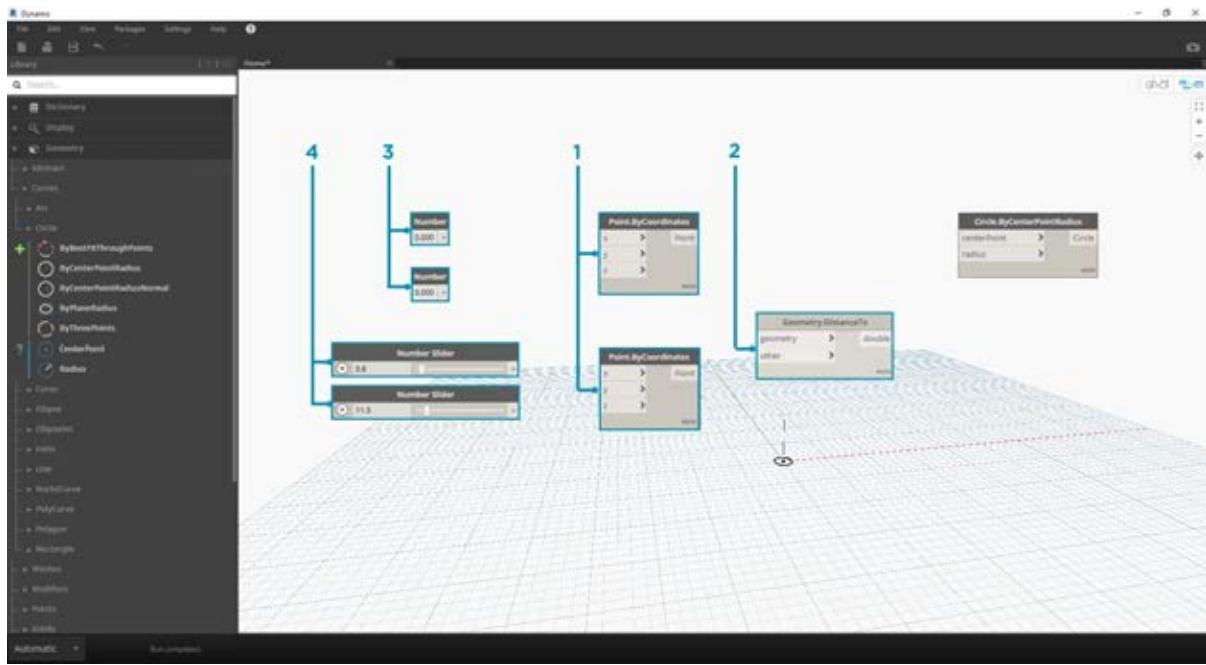
1. Navigieren Sie zu Geometry > Circle > **Circle.ByPointRadius**.
2. Suchen Sie nach > "ByCenterPointRadius..."

Fügen Sie den Block **Circle.ByPointRadius** zum Arbeitsbereich hinzu, indem Sie in der Bibliothek darauf klicken. Dadurch wird der Block in der Mitte des Arbeitsbereichs hinzugefügt.



1. Der Block Circle.ByPointandRadius in der Bibliothek
2. Durch Klicken auf den Block in der Bibliothek wird er zum Arbeitsbereich hinzugefügt.

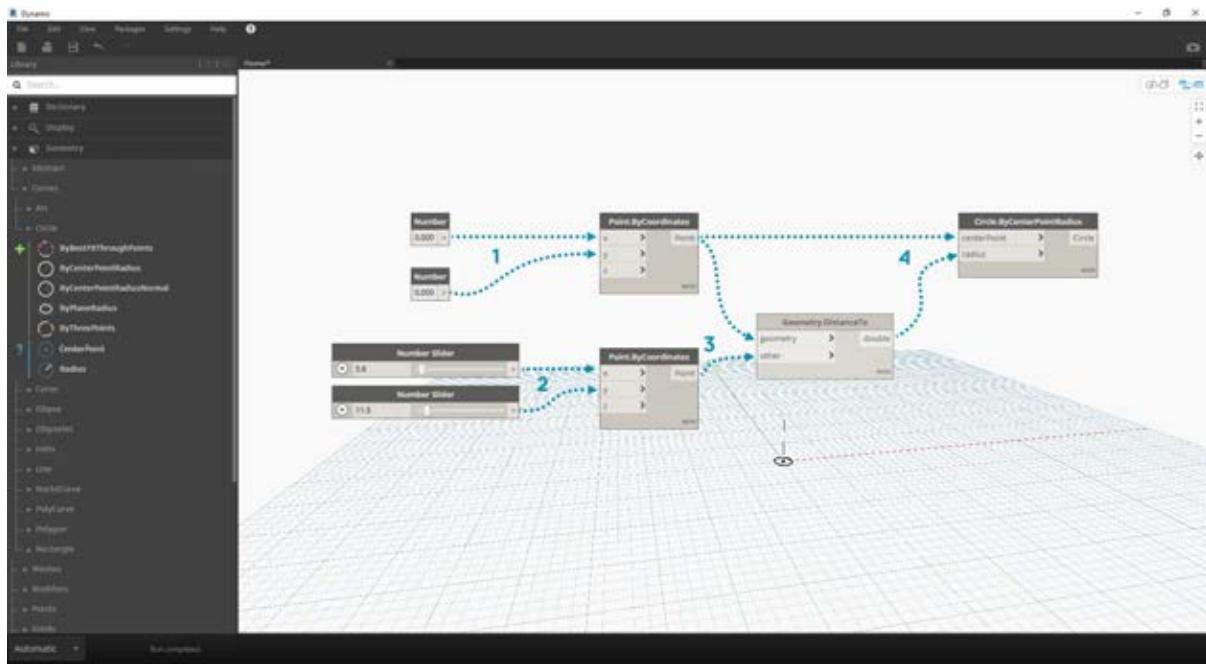
Darüber hinaus benötigen Sie die Blöcke **Point.ByCoordinates**, **Number Input** und **Number Slider**.



1. Geometry > Points > Point > **Point.ByCoordinates**
2. Geometry > Geometry > **DistanceTo**
3. Input > Basic > **Number**
4. Input > Basic > **Number Slider**

### Blöcke mit Drähten verbinden

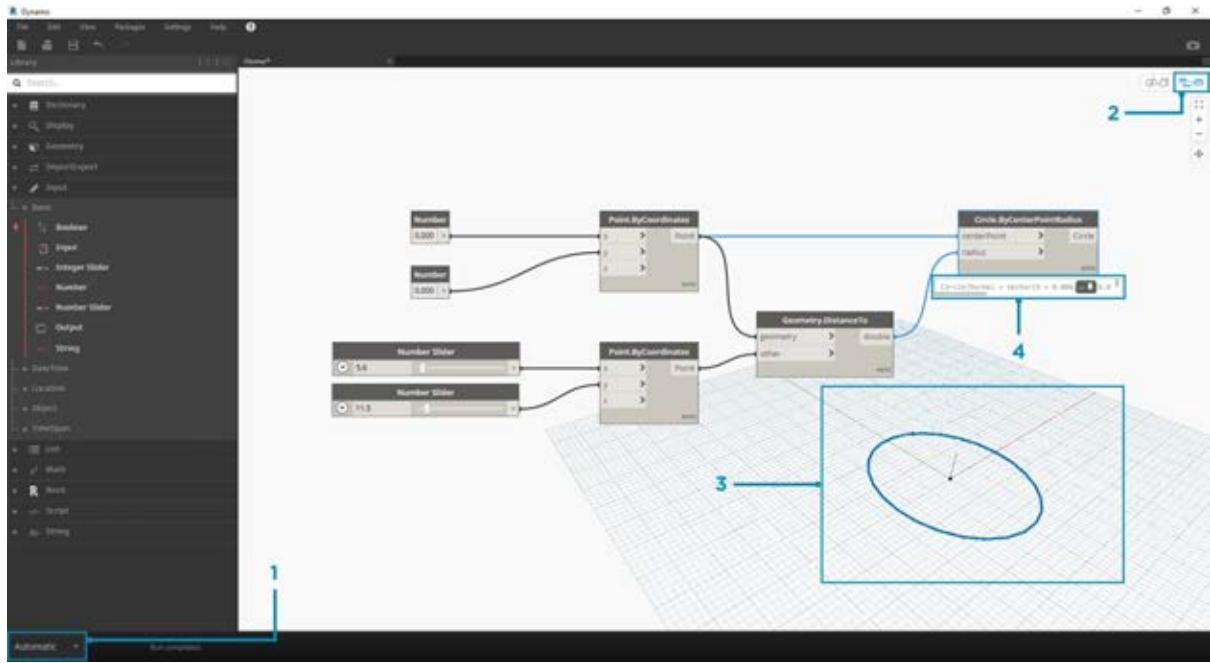
Nachdem Sie mehrere Blöcke zum Arbeitsbereich hinzugefügt haben, müssen Sie die Anschlüsse der Blöcke mit Drähten verbinden. Diese Verbindungen definieren den Datenfluss.



1. Number zu Point.ByCoordinates
2. Number Sliders zu Point.ByCoordinates
3. Point.ByCoordinates (2) zu DistanceTo
4. Point.ByCoordinates und DistanceTo zu Circle.ByCenterPointRadius

## Programm ausführen

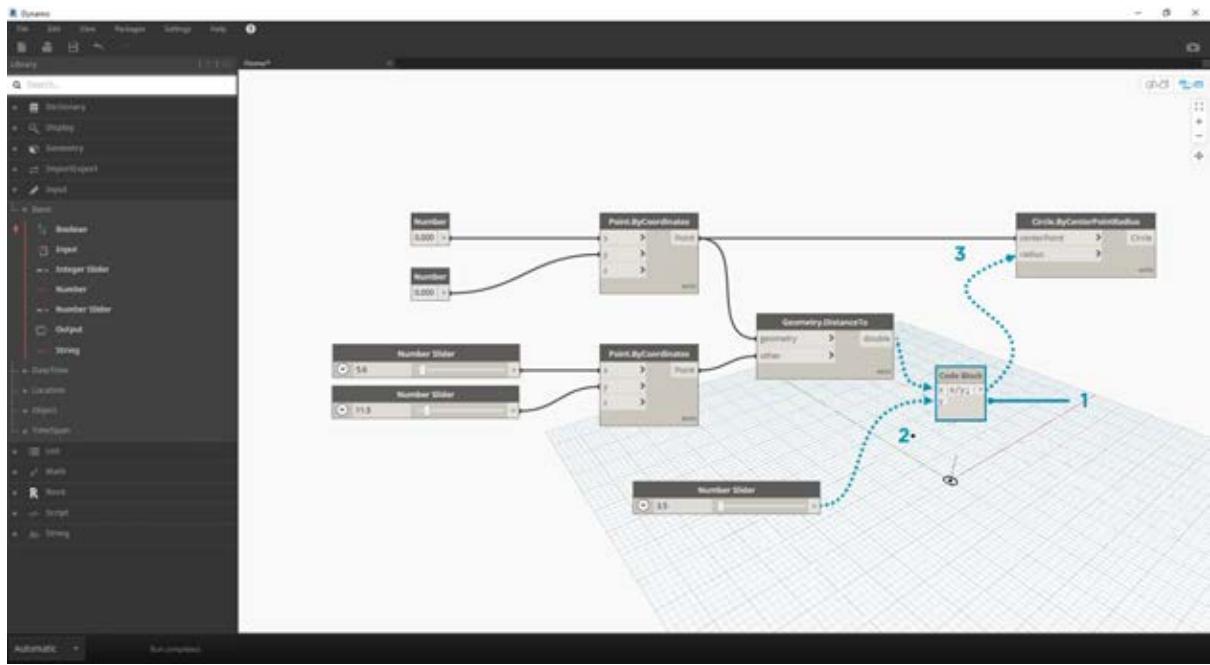
Nachdem Sie den Programmablauf definiert haben, müssen Sie Dynamo noch mitteilen, wie es ausgeführt werden soll. Wenn das Programm ausgeführt wird (entweder automatisch oder durch Klicken auf Ausführen im manuellen Modus), werden Daten über die Drähte übertragen und die Ergebnisse in der 3D-Vorschau angezeigt.



1. (Auf Ausführen klicken): Wenn sich die Ausführungsleiste im manuellen Modus befindet, muss auf Ausführen geklickt werden, um das Diagramm auszuführen.
2. Blockvorschau: Durch Verschieben des Mauszeigers auf das Feld in der rechten unteren Ecke eines Blocks wird ein Popup-Fenster mit den Ergebnissen angezeigt.
3. 3D-Vorschau: Wenn einer der verfügbaren Blöcke Geometrie erzeugt, wird eine 3D-Vorschau angezeigt.
4. Die Ausgabe von Geometrie im Erstellungsblock.

## Details hinzufügen

Wenn das Programm ordnungsgemäß funktioniert, wird in der 3D-Vorschau ein Kreis angezeigt, der durch den Attraktorpunkt verläuft. Anschließend können Sie weitere Details oder Steuerelemente hinzufügen. Passen Sie die Eingabe für den Kreis-Block an, um den Einfluss auf den Radius zu kalibrieren. Fügen Sie einen weiteren **Number Slider**-Block zum Arbeitsbereich hinzu, und doppelklicken Sie auf einen leeren Bereich im Arbeitsbereich, um einen **Code Block**-Block hinzuzufügen. Bearbeiten Sie das Feld im Code Block-Block, indem Sie X/Y angeben.

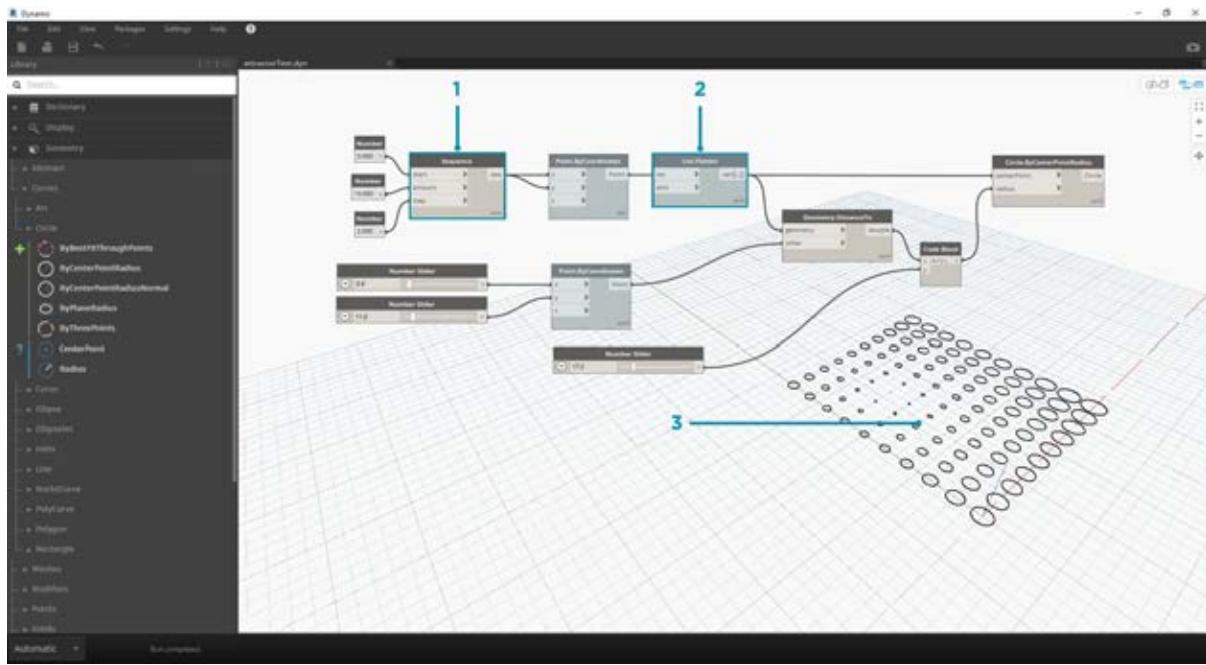


### 1. Code Block

1. **Code Block**
2. **DistanceTo und Number Slider zu Code Block**
3. **Code Block zu Circle.ByCenterPointRadius**

### Komplexität hinzufügen

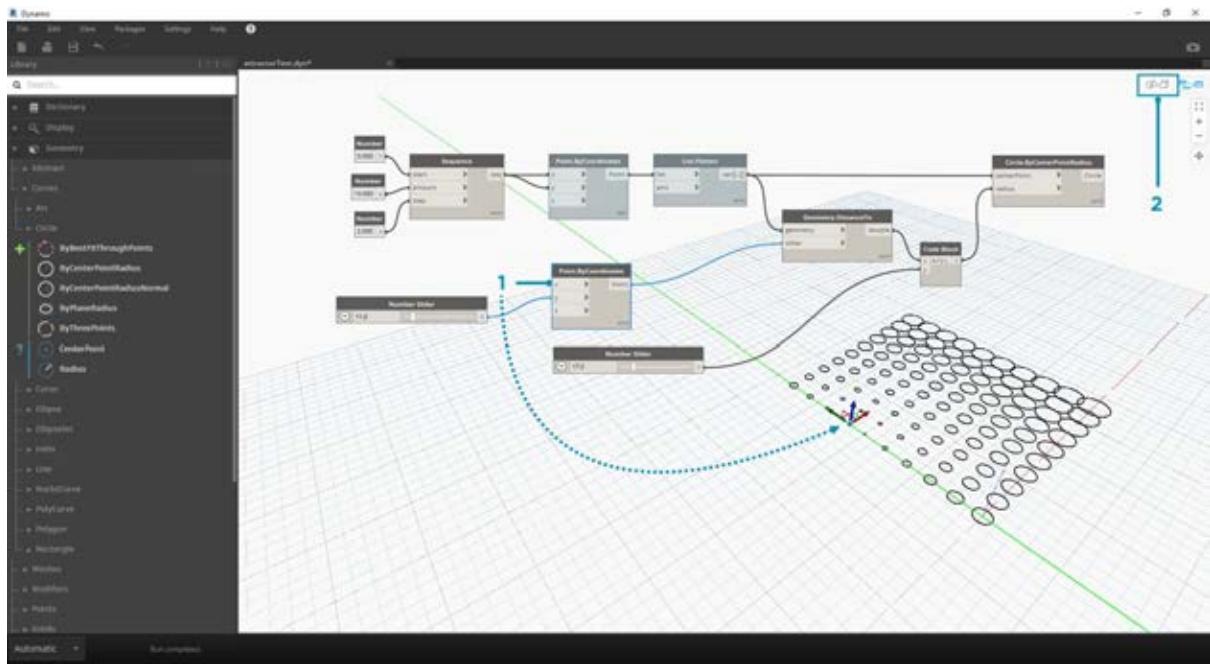
Einfach zu beginnen und dann die Komplexität zu erhöhen stellt eine effektive Möglichkeit dar, Programme schrittweise zu entwickeln. Wenn das Programm also für einen Kreis funktioniert, können Sie es auch für mehrere Kreise ausführen. Wenn Sie anstelle eines Mittelpunkts ein Raster an Punkten verwenden und diese Änderung in der resultierenden Datenstruktur umsetzen, werden von Ihrem Programm viele Kreise erzeugt, jeder mit einem eindeutigen Radiuswert, der durch die kalibrierte Entfernung zum Attraktorpunkt definiert wird.



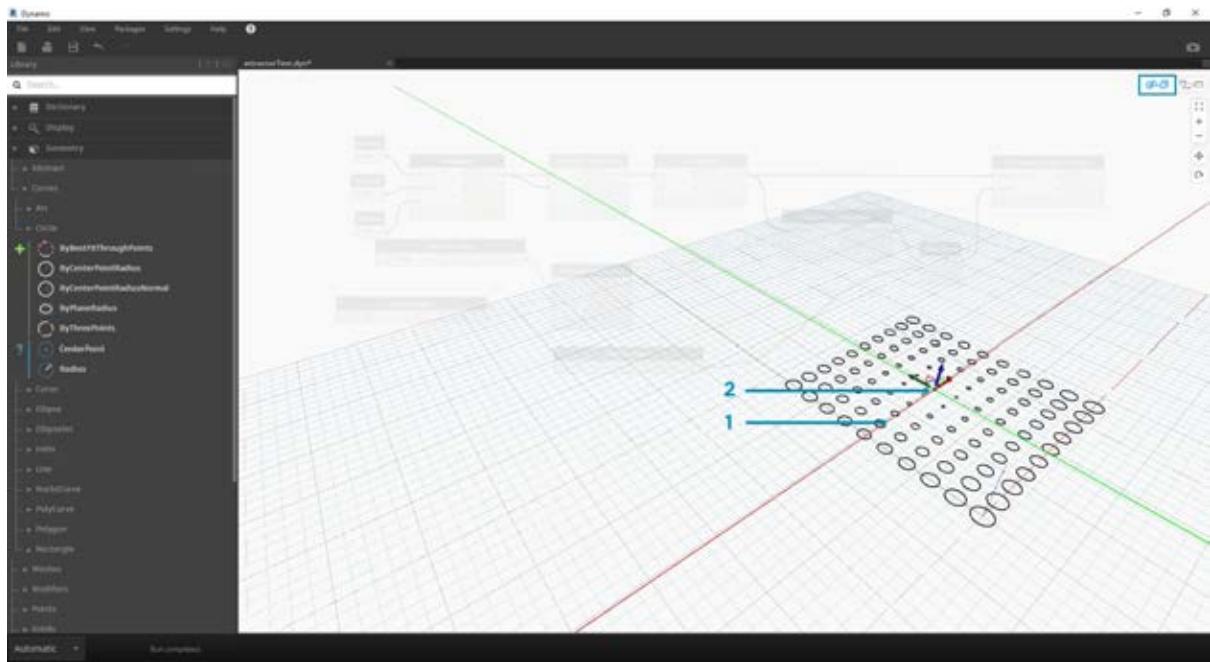
1. Fügen Sie einen **Number Sequence**-Block hinzu und ersetzen Sie die Eingaben von **Point.ByCoordinates**: Klicken Sie mit der rechten Maustaste auf Point.ByCoordinates und wählen Sie Vergitterung > Kreuzprodukt.
2. Fügen Sie einen **Flatten**-Block nach Point.ByCoordinates hinzu. Zum vollständigen Abflachen einer Liste belassen Sie Vorgabeeinstellung von **amt** auf **-1**.
3. Die 3D-Vorschau wird mit einem Raster von Kreisen aktualisiert.

### Mit Direktbearbeitung anpassen

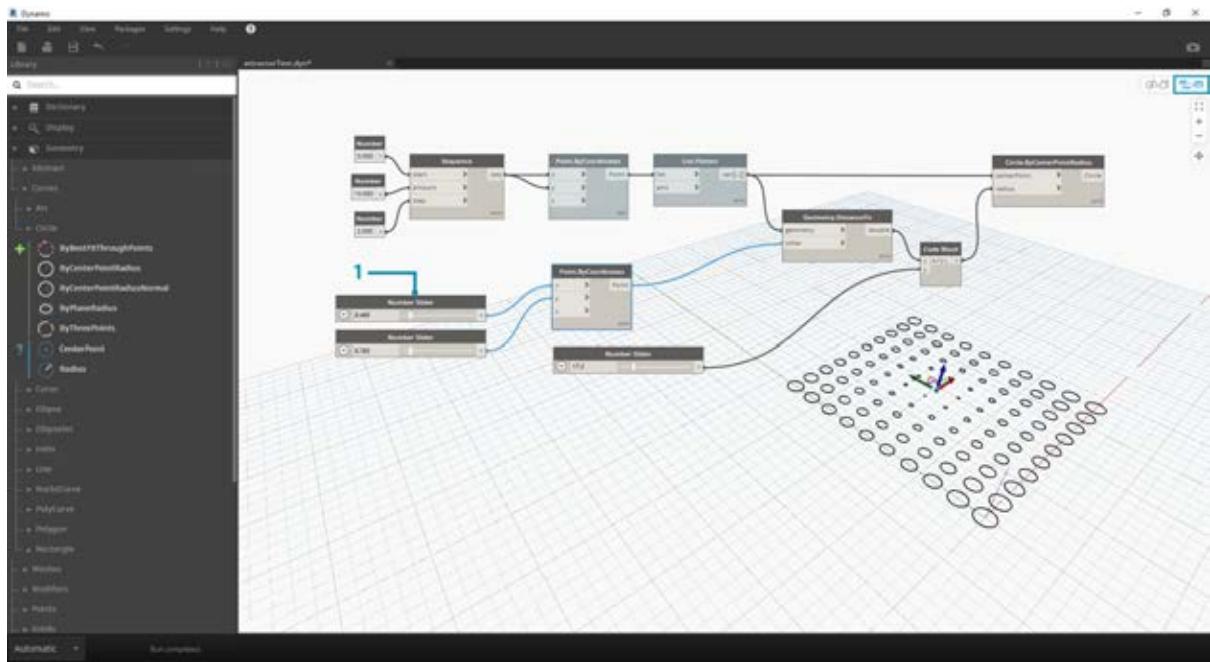
In manchen Fällen ist die numerische Bearbeitung nicht der richtige Ansatz. Jetzt können Sie beim Navigieren in der Hintergrund-3D-Vorschau Punktgeometrie manuell drücken und ziehen. Sie können auch andere Geometrie steuern, die durch einen Punkt konstruiert wurde. **Sphere.ByCenterPointRadius** kann beispielsweise ebenfalls direkt bearbeitet werden. Sie können die Position eines Punkts aus einer Reihe von X-, Y- und Z-Werten mit **Point.ByCoordinates** steuern. Mit dem Direktbearbeitungsansatz sind Sie jedoch in der Lage, die Werte der Schieberegler zu aktualisieren, indem Sie den Punkt im Modus **Navigation in 3D-Vorschau** manuell verschieben. Dieser Ansatz bietet eine intuitivere Methode zum Steuern von mehreren diskreten Werten, die eine Punktposition identifizieren.



1. Um die **Direktbearbeitung** zu verwenden, wählen Sie die Gruppe mit dem zu verschiebenden Punkt aus – über dem ausgewählten Punkt werden Pfeile angezeigt.
2. Wechseln Sie in den Modus **Navigation in 3D-Vorschau**.



1. Wenn Sie den Cursor über den Punkt bewegen, werden die X-, Y- und Z-Achsen angezeigt.
2. Klicken Sie, und ziehen Sie den farbigen Pfeil, um die entsprechende Achse zu verschieben. Die **Number Slider**-Werte werden live mit dem manuell verschobenen Punkt aktualisiert.

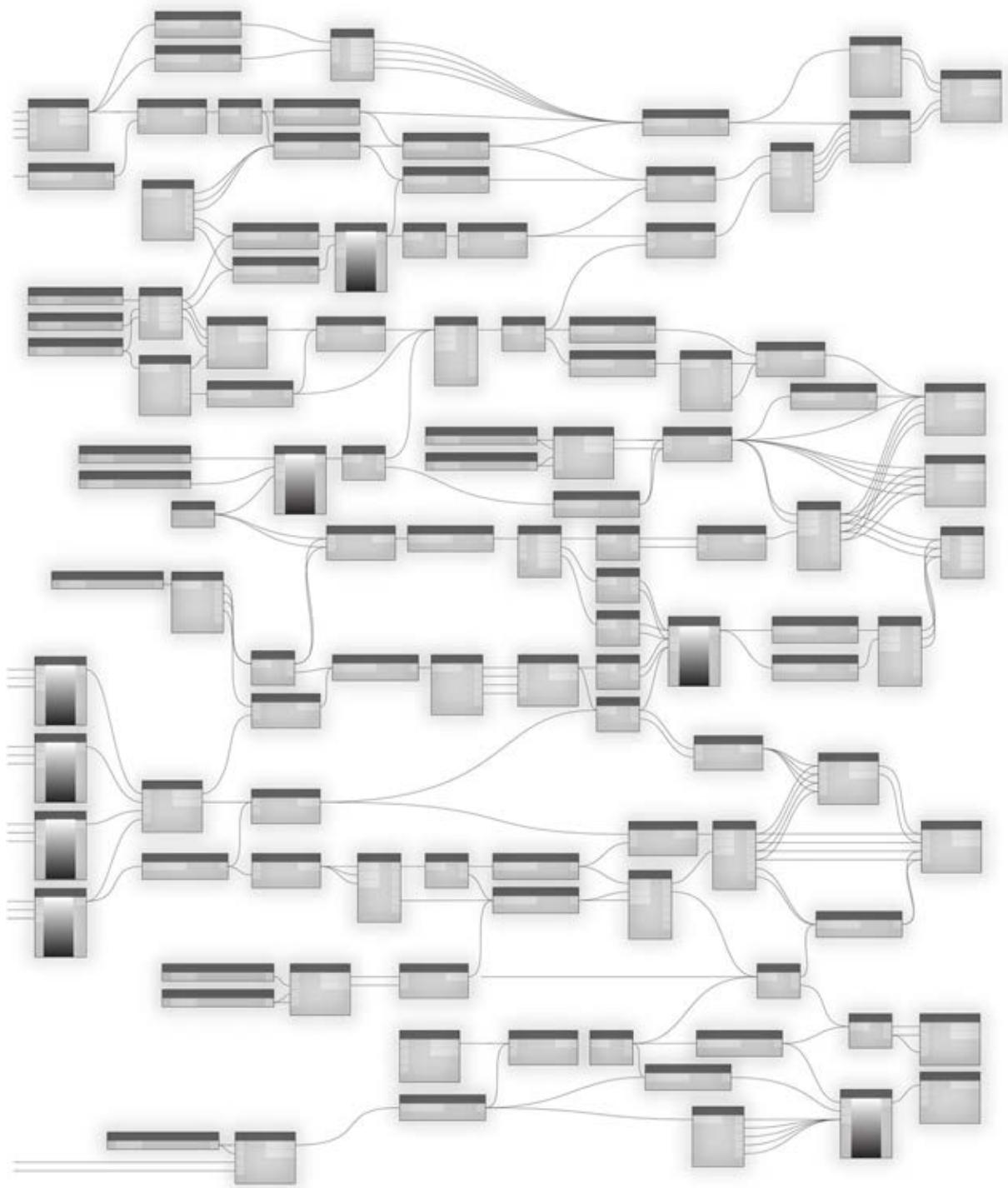


1. Beachten Sie, dass vor der **Direktbearbeitung** nur ein Schieberegler an die **Point.ByCoordinates**-Komponente angeschlossen war. Wenn Sie den Punkt manuell in X-Richtung verschieben, erstellt Dynamo automatisch einen neuen **Number Slider** für die X-Eingabe.

# **Die Anatomie visueller Programme**

## **Die Anatomie visueller Programme**

Dynamo ermöglicht das Erstellen visueller Programme in einem Arbeitsbereich, indem Blöcke mithilfe von Drähten verbunden werden, um den logischen Ablauf des resultierenden visuellen Programms festzulegen. Dieses Kapitel bietet eine Einführung in die Elemente visueller Programme, die Organisation der in der Bibliothek von Dynamo verfügbaren Blöcke, die Teile und Status von Blöcken und die optimalen Verfahren für Arbeitsbereiche.



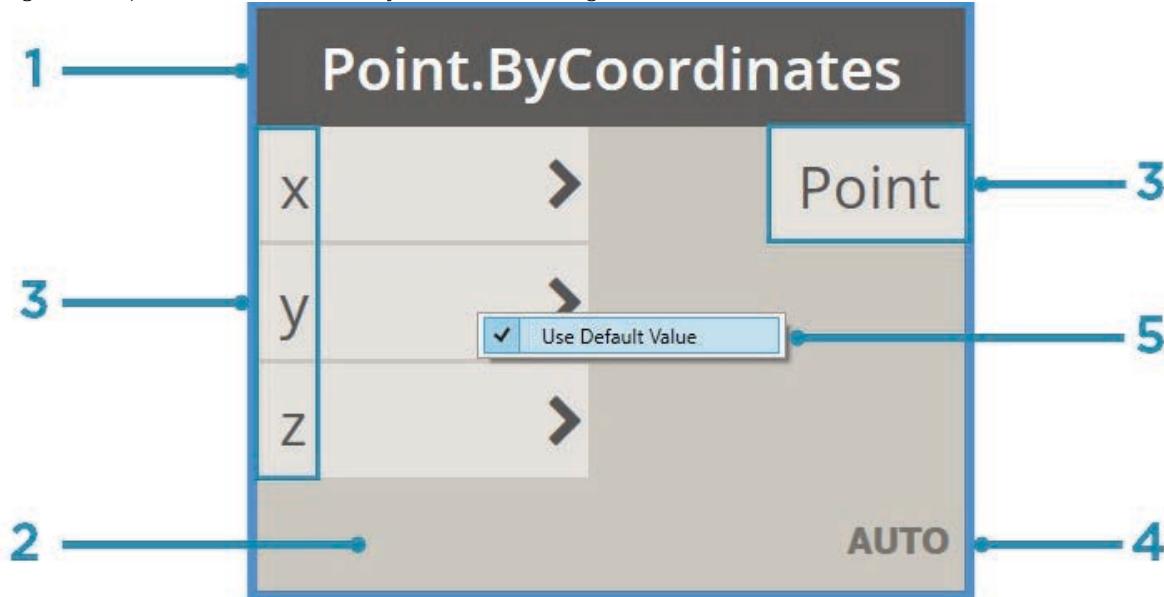
# Blöcke

## Blöcke

In Dynamo stellen **Blöcke** die Objekte dar, die zum Bilden eines visuellen Programms miteinander verbunden werden. Jeder **Block** führt einen Vorgang aus – vom einfachen Speichern einer Zahl bis hin zu komplexen Aktionen wie das Erstellen oder Abfragen von Geometrie.

### Anatomie von Blöcken

In Dynamo setzen sich die meisten Blöcke aus fünf Teilen zusammen. Abgesehen von einigen Ausnahmen (z. B. Eingabeblöcke) kann die Anatomie eines jeden Blocks wie folgt beschrieben werden:

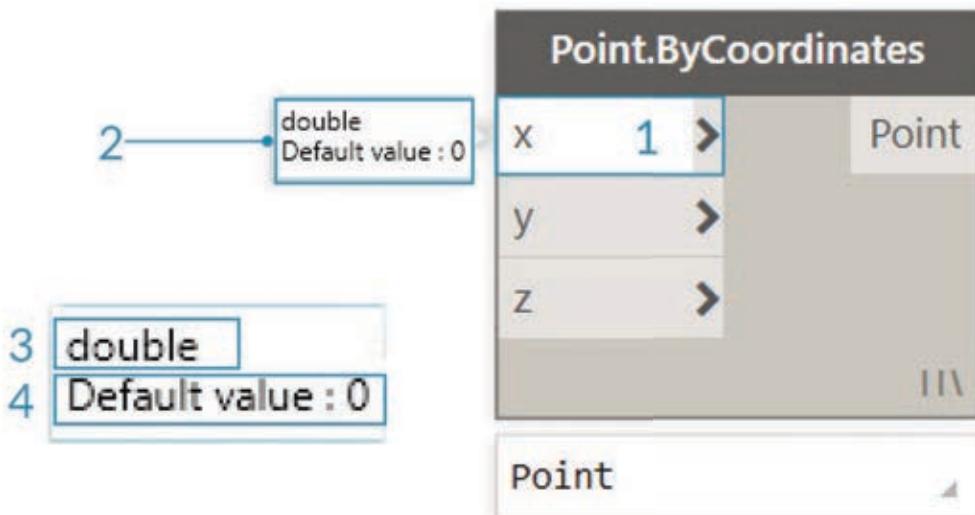


1. Name: Der Name des Blocks gemäß Category.Name-Benennungskonvention
2. Hauptbereich: Der Hauptkörper des Blocks. Durch Klicken mit der rechten Maustaste auf diesen Bereich werden Optionen für den gesamten Block angezeigt.
3. Anschlüsse (eingehend und ausgehend): Die Rezeptoren für Drähte, über die die eingegebenen Daten sowie die Ergebnisse von Blockaktionen an Blöcke geliefert werden.
4. Symbol Vergitterung: Zeigt die für die Zuordnung von Listeneingaben angegebene Vergitterungsoption an (mehr dazu später).
5. Vorgabewert: Klicken Sie mit der rechten Maustaste auf einen Eingabeanschluss. Einige Blöcke verfügen über Vorgabewerte, die verwendet werden können, aber nicht verwendet werden müssen.

### Anschlüsse

Die Eingaben und Ausgaben für Blöcke werden als **Anschlüsse** bezeichnet. Sie fungieren als Kontakte für Drähte. Daten gelangen über die Anschlüsse auf der linken Seite in Blöcke und strömen auf der rechten Seite wieder aus den Blöcken hinaus, nachdem der entsprechende Vorgang ausgeführt wurde. Anschlüsse erwarten Daten eines bestimmten Typs. Das Verbinden einer Zahl wie 2,75 mit den Anschlüssen eines Point By Coordinates-Blocks führt beispielsweise dazu, dass ein Punkt erfolgreich erstellt wird. Wenn jedoch "Rot" an denselben Anschluss liefert wird, tritt ein Fehler auf.

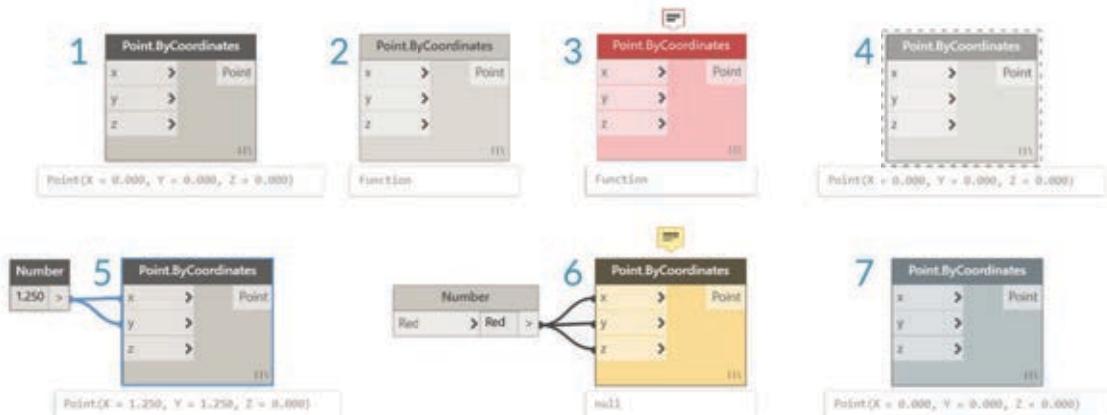
Tipp: Bewegen Sie den Cursor auf einen Anschluss, um eine QuickInfo mit dem erwarteten Datentyp aufzurufen.



1. Anschlussbezeichnung
2. QuickInfo
3. Datentyp
4. Vorgabewert

## Status

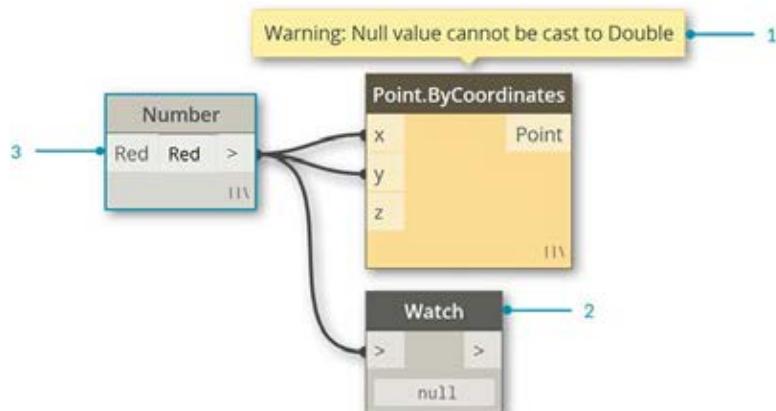
Dynamo gibt einen Hinweis auf den Status der Ausführung eines visuellen Programms aus, indem Blöcke mit unterschiedlichen Farbschemata basierend auf dem Status der einzelnen Blöcke gerendert werden. Darüber hinaus werden durch Bewegen des Cursors auf den Namen bzw. die Anschlüsse oder durch Klicken mit der rechten Maustaste darauf zusätzliche Informationen und Optionen angezeigt.



1. Aktiv: Blöcke, deren Namen einen dunkelgrauen Hintergrund aufweisen, sind ordnungsgemäß angeschlossen, d. h., alle Eingaben konnten erfolgreich verbunden werden.
2. Inaktiv: Graue Blöcke sind inaktiv und müssen mit Drähten verbunden werden, um in den Programmablauf im aktiven Arbeitsbereich integriert zu werden.
3. Fehlerstatus: Rot weist auf einen Fehlerstatus des Blocks hin.
4. Anhalten: Bei transparent dargestellten Blöcken ist Anhalten aktiviert, d. h., ihre Ausführung wurde unterbrochen.
5. Ausgewählt: Aktuell ausgewählte Blöcke weisen eine aquamarinblau hervorgehobenen Rand auf.
6. Warnung: Gelb markierte Blöcke befinden sich im Warnzustand, d. h., sie enthalten eventuell die falschen Datentypen.
7. Hintergrundvorschau: Dunkelgrau bedeutet, dass die Geometrievorschau deaktiviert ist.

Wenn Ihr visuelles Programm Warnungen oder Fehler aufweist, gibt Dynamo zusätzliche Informationen zu dem Problem an. Alle Blöcke, die in gelb angezeigt werden, verfügen auch über eine QuickInfo über dem Namen. Bewegen Sie den Cursor auf die QuickInfo, um sie zu erweitern.

Tipp: Untersuchen Sie vor dem Hintergrund dieser QuickInfo die vorgelagerten Blöcke, um zu sehen, ob der erforderliche Datentyp oder die erforderliche Datenstruktur fehlerhaft ist.



1. QuickInfo zu Warnung: "Null" oder keine Daten können nicht als Double verstanden werden, d. h. eine Zahl.
2. Verwenden Sie den Watch-Block, um die Eingabedaten zu untersuchen.
3. Der vorgelagerte Number-Block speichert "Rot", keine Zahl.

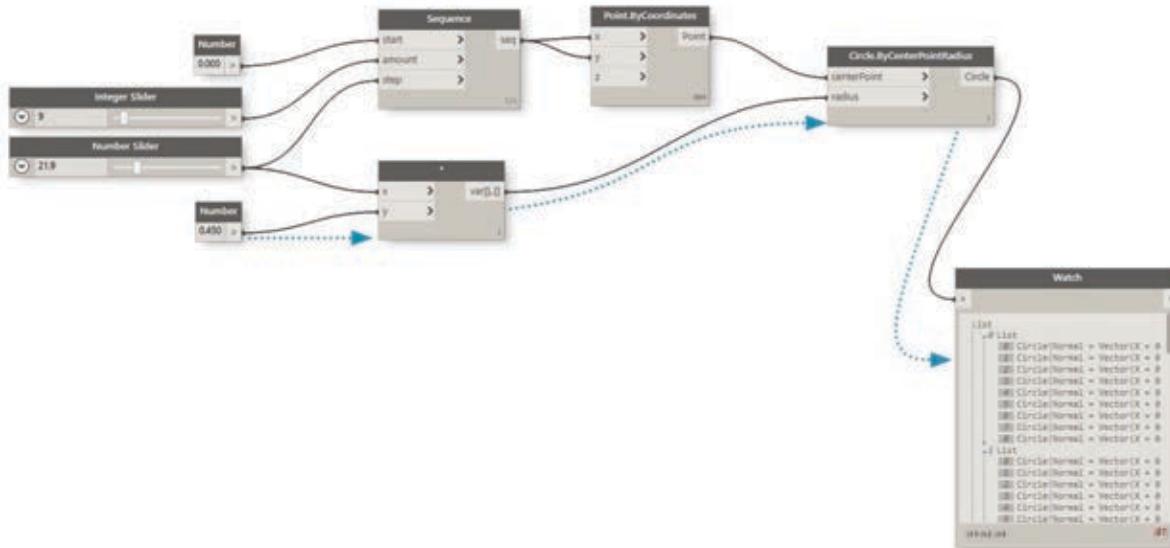
# Drähte

## Drähte

Drähte verbinden Blöcke miteinander, um Beziehungen zu erstellen und den Ablauf eines visuellen Programms festzulegen. Sie können sie sich buchstäblich als elektrische Drähte vorstellen, die Datenimpulse von einem Objekt zum nächsten transportieren.

### Programmablauf

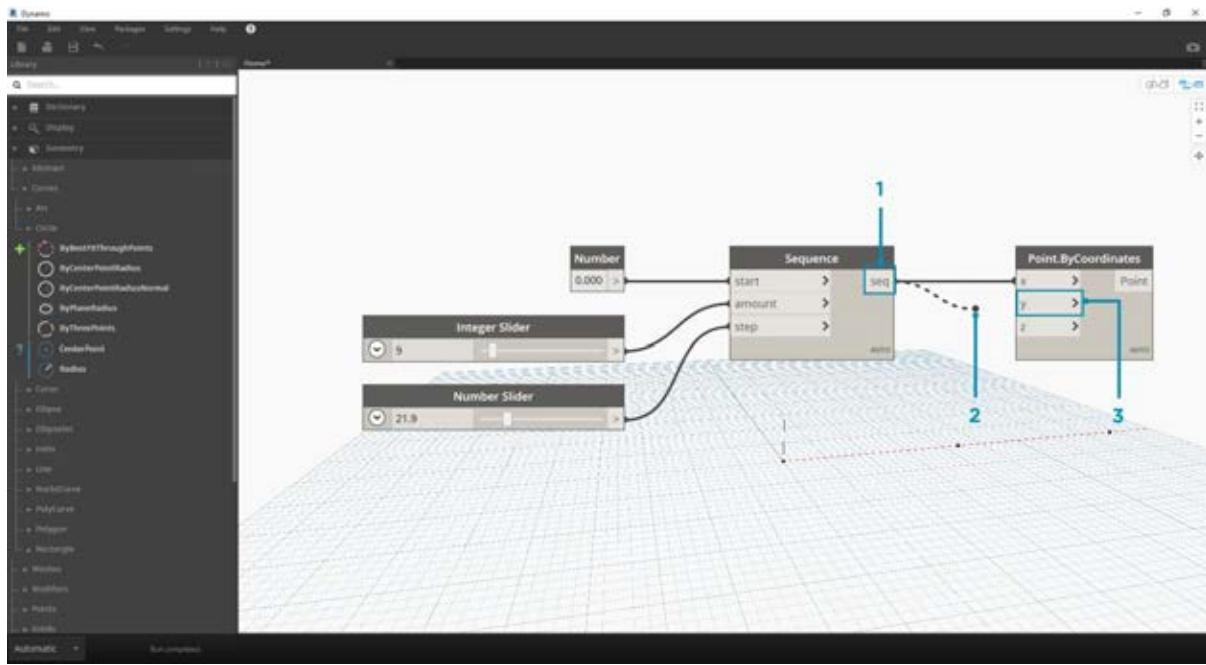
Drähte verbinden den Ausgabeanschluss eines Blocks mit dem Eingabeanschluss eines anderen Blocks. Diese Direktonalität legt den **Datenfluss** im visuellen Programm fest. Sie können die Blöcke zwar wie gewünscht im Arbeitsbereich anordnen, da sich die Ausgabeanschlüsse jedoch auf der rechten Seite der Blöcke und die Eingabeanschlüsse auf der linken Seite befinden, wird der Programmablauf allgemein als von links nach rechts betrachtet.



## Drähte erstellen

Sie erstellen einen Draht, indem Sie mit der linken Maustaste auf einen Anschluss und dann erneut mit der linken Maustaste auf den Anschluss eines anderen Blocks klicken, um eine Verbindung festzulegen. Während der Herstellung der Verbindung wird der Draht gestrichelt angezeigt. Nachdem die Verbindung erfolgreich hergestellt wurde, erscheint er als durchgezogene Linie. Die Daten fließen immer von Ausgabe zu Eingabe durch diesen Draht. Sie können den Draht jedoch in beliebiger Richtung erstellen, die dadurch definiert wird, in welcher Reihenfolge Sie auf die Anschlüsse klicken.

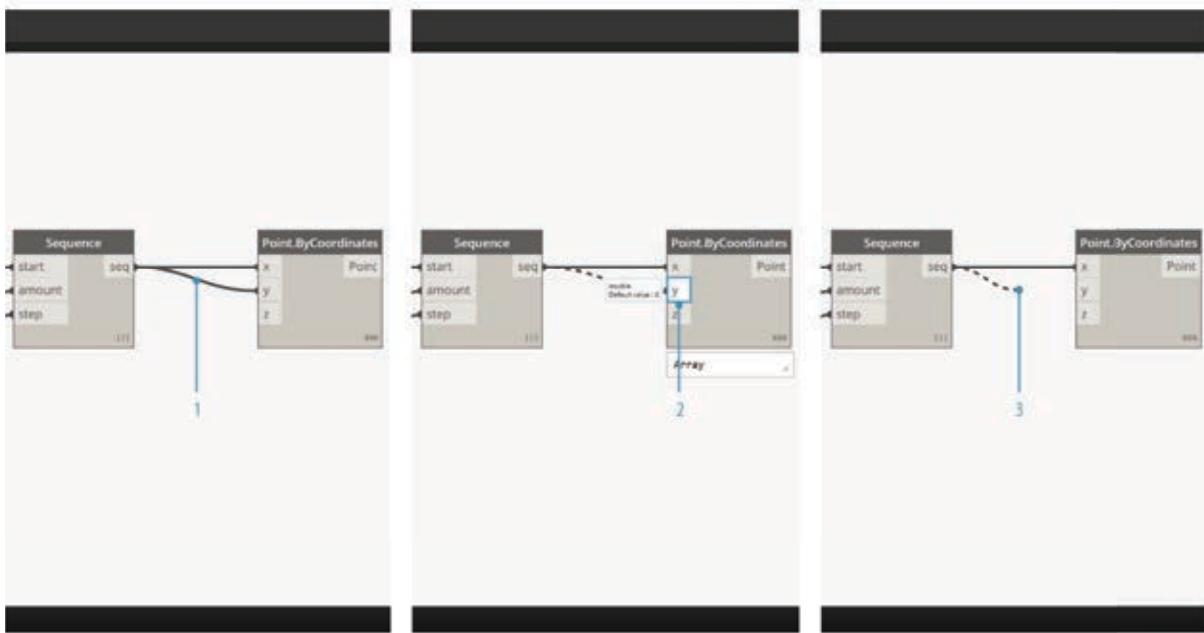
Tipp: Bevor Sie das Herstellen der Verbindung durch das zweite Klicken abschließen, lassen Sie den Draht an einem Anschluss einrasten und bewegen Sie den Cursor darauf, um die zugehörige QuickInfo anzuzeigen.



1. Klicken Sie auf den seq-Ausgabeanschluss des Number Sequence-Blocks.
2. Beim Verschieben der Maus zu einem anderen Anschluss wird der Draht gestrichelt angezeigt.
3. Klicken Sie auf den y-Eingabeanschluss des Point.ByCoordinate-Blocks, um die Verbindung herzustellen.

## Drähte bearbeiten

Es kommt häufig vor, dass Sie den Programmablauf in Ihrem visuellen Programm anpassen müssen, indem Sie die durch Drähte dargestellten Verbindungen bearbeiten. Um einen Draht zu bearbeiten, klicken Sie mit der linken Maustaste auf den Eingabeanschluss eines Blocks, der bereits verbunden ist. Sie haben jetzt zwei Möglichkeiten:

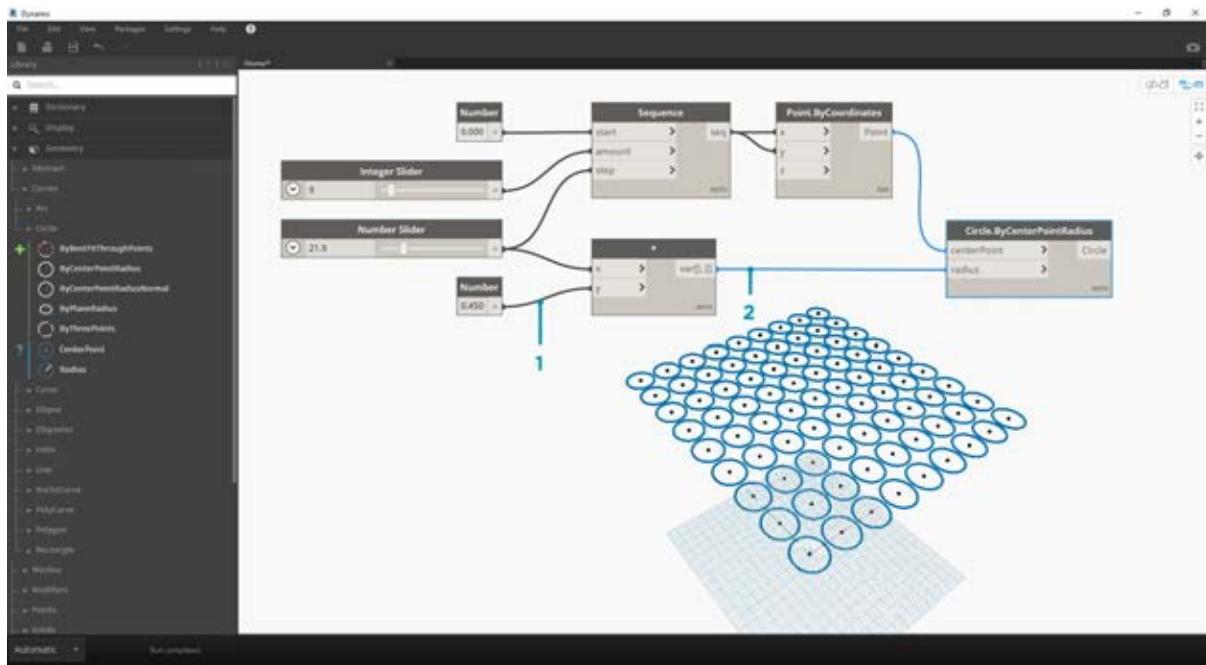


1. Vorhandener Draht
2. Um die Verbindung zu einem Eingabeanchluss zu ändern, klicken Sie mit der linken Maustaste auf einen anderen Eingabeanchluss.
3. Um den Draht zu entfernen, ziehen Sie den Draht weg und klicken Sie mit der linken Maustaste in den Arbeitsbereich.

\*Anmerkung: Sie können jetzt auch mehrere Drähte gleichzeitig verschieben. Diese Funktionalität wird hier behandelt:  
<http://dynamobim.org/dynamo-1-3-release/>

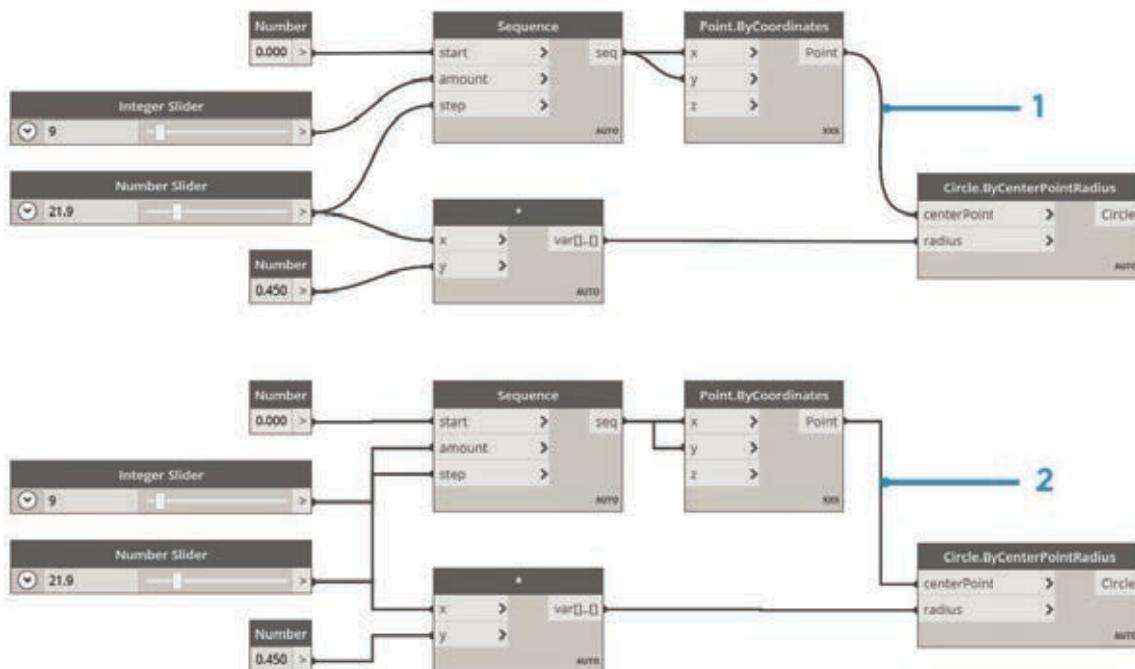
### Drahtvoransichten

Standardmäßig werden Drähte in der Vorschau mit einem grauen Strich angezeigt. Wenn ein Block ausgewählt wird, werden alle Verbindungsdrähte wie der Block in aquamarinblau hervorgehoben.



1. Standarddraht
2. Hervorgehobener Draht

In Dynamo können Sie über das Menü Ansicht > Connectors auch anpassen, wie Drähte im Arbeitsbereich angezeigt werden. Sie zwischen der Anzeige als Kurve oder Polylinie umschalten oder die Anzeige vollständig deaktivieren.



1. Connector-Typ: Kurven
2. Connector-Typ: Polylinien

# Dynamo-Bibliothek

## Dynamo-Bibliothek

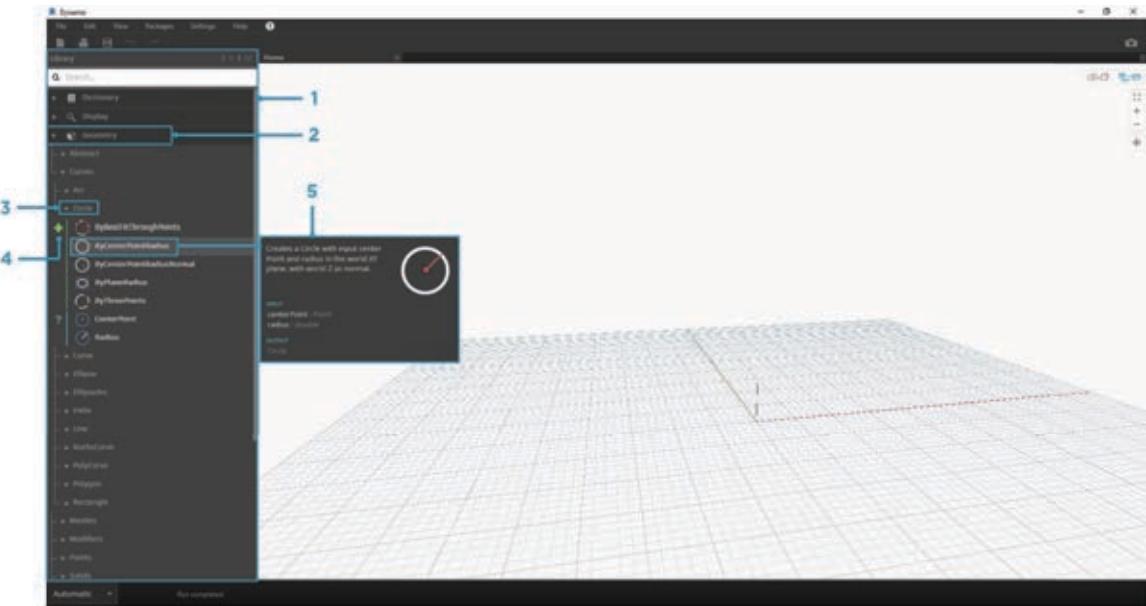
Die **Dynamo-Bibliothek** enthält die Blöcke, die Sie zum Arbeitsbereich hinzufügen können, um visuelle Programme zur Ausführung zu definieren. In der Bibliothek können Sie nach Blöcken suchen. Die hier enthaltenen Blöcke – die installierten Basisblöcke, Ihre benutzerdefinierten Blöcke und die von Ihnen zu Dynamo hinzugefügten Blöcke aus Package Manager – sind hierarchisch nach Kategorie organisiert. Im Folgenden werden Sie sich mit dieser Organisation und den wichtigsten Blöcken, die häufig von Ihnen verwendet werden, vertraut machen.

### Bibliothek der Bibliotheken

Die **Dynamo-Bibliothek**, mit der in der Anwendung eine Schnittstelle gebildet wird, entspricht einer Sammlung funktionaler Bibliotheken, die jeweils nach Kategorie gruppierte Blöcke enthalten. Auch wenn dies auf den ersten Blick überflüssig erscheinen mag, handelt es sich dabei um ein flexibles Framework für die Organisation von Blöcken, die zum Lieferumfang der Standardinstallation von Dynamo gehören – und deren Grundfunktionalität Sie noch um benutzerdefinierte Blöcke und zusätzliche Pakete erweitern können.

### Das Organisationsschema

Der Abschnitt **Bibliothek** der Dynamo-Benutzeroberfläche umfasst hierarchisch organisierte Bibliotheken. Um einen Block in der Bibliothek zu finden, müssen Sie tiefer in die Bibliothek eindringen und nacheinander die Bibliothek, die Kategorien der Bibliothek und die Unterkategorien einer Kategorie durchsuchen.

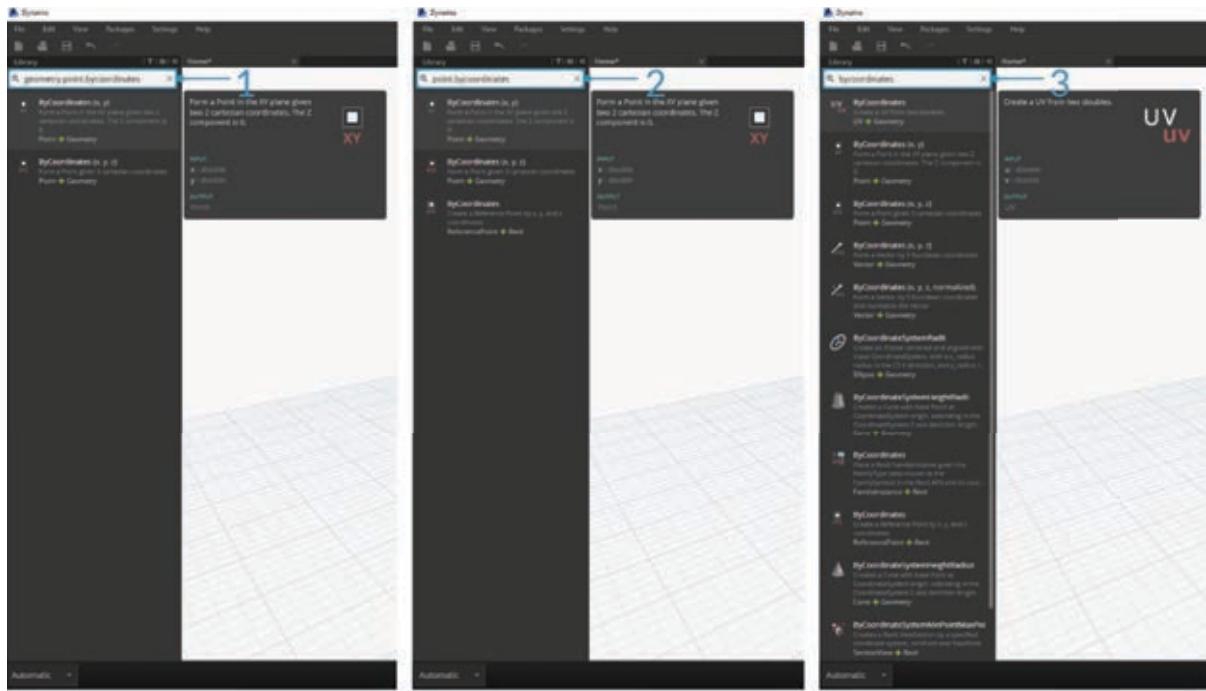


1. Die Bibliothek: Der Bereich der Dynamo-Benutzeroberfläche
2. Eine Bibliothek: Eine Sammlung verwandter Kategorien wie **Geometry**
3. Eine Kategorie: Eine Sammlung verwandter Blöcke wie alles in Verbindung mit **Circle**
4. Eine Unterkategorie: Aufschlüsselung der Blöcke innerhalb einer Kategorie, in der Regel in **Erstellen, Aktion** oder **Abfrage**
5. Ein Block: Die Objekte, die zum Arbeitsbereich hinzugefügt werden, um eine Aktion auszuführen

### Benennungskonventionen

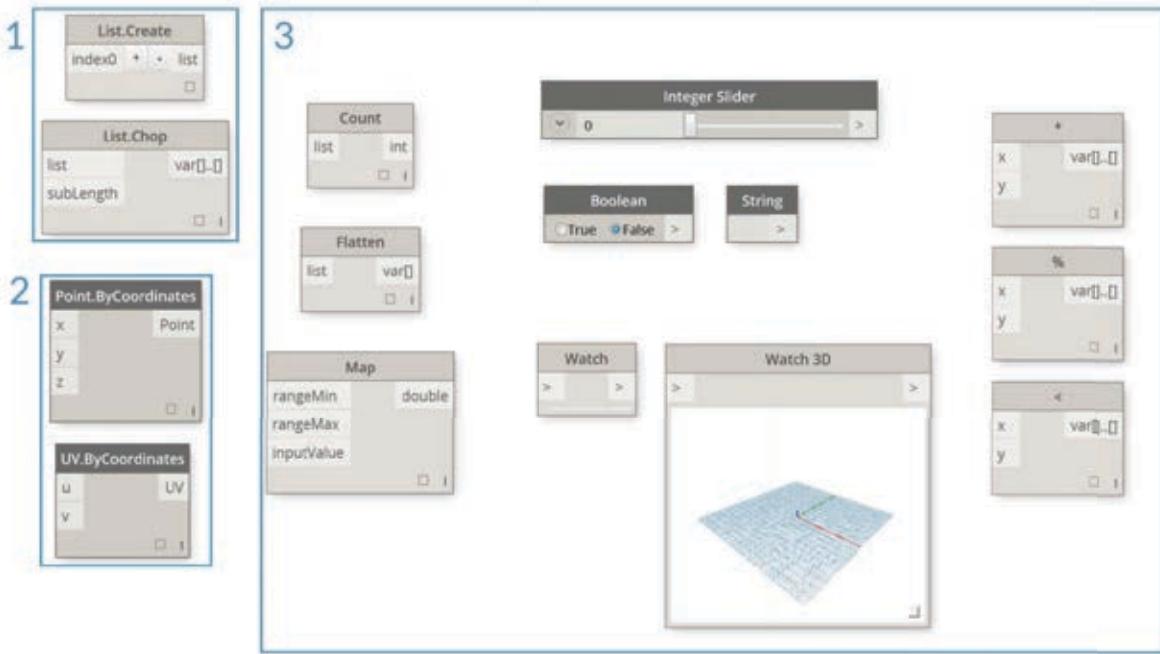
Die Hierarchie der jeweiligen Bibliothek spiegelt sich im Namen der Blöcke wider, die zum Arbeitsbereich hinzugefügt wurden und die Sie auch im Suchfeld oder mit Codeblöcken verwenden können (für die die *textuelle Sprache von Dynamo* verwendet wird). Neben der Verwendung von Schlüsselwörtern für die Suche nach Blöcken können Sie auch die Hierarchie getrennt durch einen Punkt eingeben.

Durch die Eingabe verschiedener Teile der Position des Blocks in der Bibliothekshierarchie im Format `bibliothek.kategorie.blockname` werden unterschiedliche Ergebnisse zurückgegeben:



1. `bibliothek.kategorie.blockname`
2. `kategorie.blockname`
3. `blockname` oder `schlüsselwort`

In der Regel wird der Name eines Blocks im Arbeitsbereich im Format `kategorie.blockname` gerendert, wobei einige Ausnahme insbesondere bei der Eingabe- und Ansichtskategorie bestehen. Beachten Sie bei ähnlich benannten Blöcken den Kategorieunterschied:



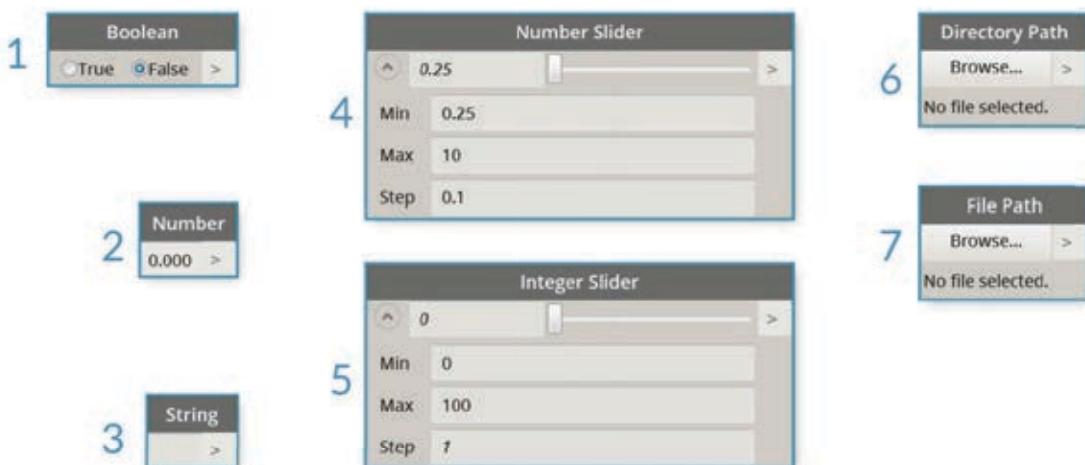
1. `Point.ByCoordinates` und `UV.ByCoordinates` weisen denselben Namen auf, stammen jedoch aus unterschiedlichen Kategorien.
2. Blöcke aus den meisten Bibliotheken schließen das Kategorieformat ein.
3. Zu den wichtigsten Ausnahmen gehören Built-in Functions, Core.Input, Core.View und Operators.

## Häufig verwendete Blöcke

Welche der zahlreichen Blöcke, die zum Lieferumfang der Basisinstallation von Dynamo gehören, sind für die Entwicklung visueller Programme von grundlegender Bedeutung? Konzentrieren Sie sich zunächst auf jene, mit denen Sie die Parameter Ihres Programms definieren (**Input**), die Ergebnisse der Aktion eines Blocks anzeigen (**Watch**) und die Eingaben oder Funktionen mithilfe einer Verknüpfung definieren (**Code Block**).

### Eingabeblocks (Input)

Eingabeblocks stellen das primäre Mittel für die Benutzer eines visuellen Programms – sowohl für Sie selbst als auch für andere Benutzer – zur Verwendung der Schlüsselparameter dar. Im Folgenden sind die Blöcke aufgeführt, die in der Kategorie Input der Core-Bibliothek verfügbar sind:

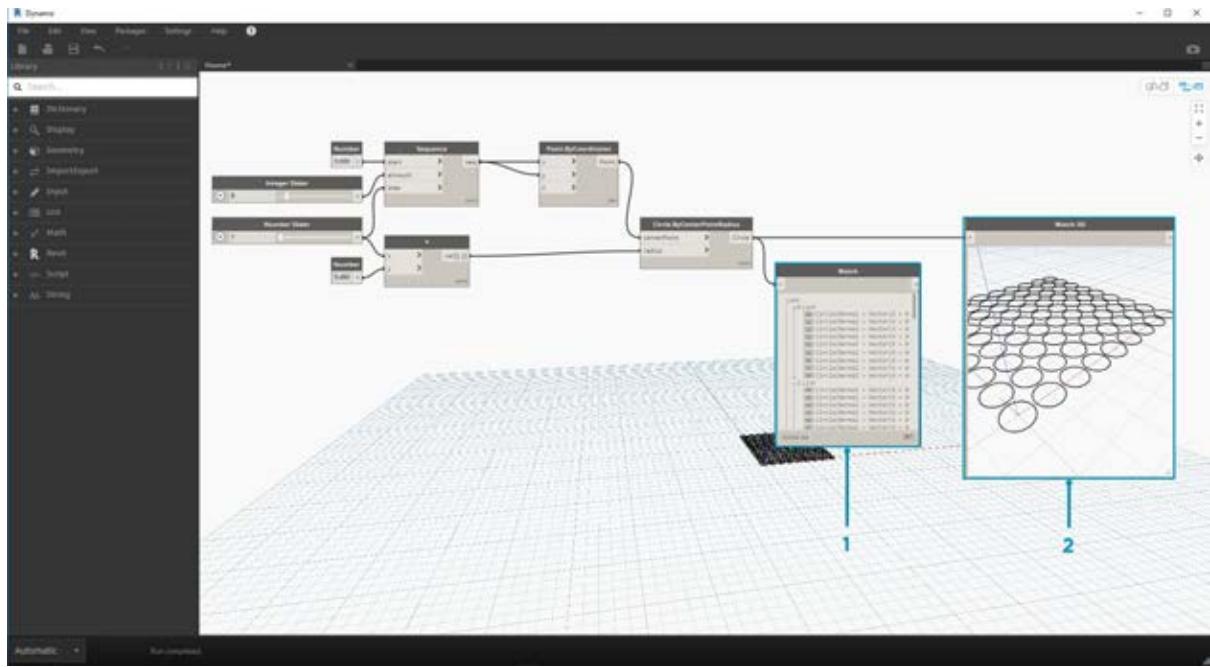


1. Boolean
2. Number
3. String
4. Number Slider
5. Integer Slider
6. Directory Path
7. File Path

### Beobachtungsblöcke (Watch)

Die Beobachtungsblöcke sind für die Verwaltung der Daten, die ein visuelles Programm durchlaufen, von grundlegender Bedeutung. Während Sie das Ergebnis eines Blocks in der Datenvorschau des Blocks anzeigen können, möchten Sie es möglicherweise in einem **Watch**-Block aufgedeckt lassen oder die Geometriergebnisse in einem **Watch3D**-Block anzeigen. Beide Blöcke sind in der Kategorie View der Core-Bibliothek enthalten.

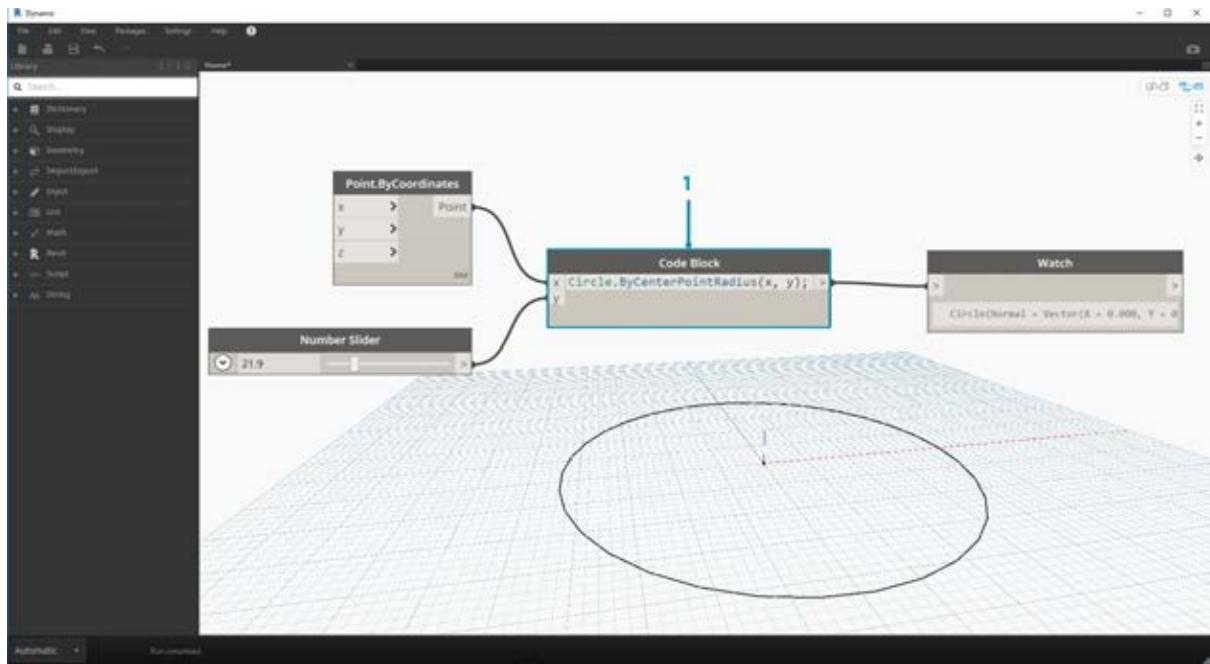
Tipp: Die 3D-Vorschau kann bisweilen unübersichtlich sein, wenn Ihr visuelles Programm viele Blöcke enthält. Ziehen Sie in diesem Fall in Betracht, im Einstellungsmenü die Option zum Anzeigen der Hintergrundvorschau zu deaktivieren und einen Watch3D-Block zu verwenden, um eine Vorschau der Geometrie anzuzeigen.



1. Watch: Beachten Sie bei Auswahl eines Elements im Watch-Block, dass das Element im Watch3D-Block und in 3D-Voransichten markiert wird.
2. Watch3D: Ändern Sie die Größe mithilfe des Griffes rechts unten, und navigieren Sie mit der Maus wie in der 3D-Vorschau.

### Codeblock

**Code Block**-Blöcke können verwendet werden, um einen Codeblock mit Linien durch Semikolons getrennt zu definieren. Dabei kann es sich einfach um X/Y handeln. Sie können auch Codeblöcke als Abkürzung verwenden, um einen Number Input-Block zu definieren oder eine andere Funktion des Blocks aufzurufen. Die Syntax hierfür entspricht der Benennungskonvention von DesignScript, der textuellen Sprache von Dynamo wie in Abschnitt 7.2 beschrieben. Versuchen Sie, mit diesem Kurzbefehl einen Kreis zu erstellen:



1. Zum Erstellen eines **Code Block**-Blocks doppelklicken
2. `Circle.ByCenterPointRadius(x, y);` eingeben
3. Durch Klicken in den Arbeitsbereich zum Löschen der Auswahl werden die Eingaben `x` und `y` automatisch hinzugefügt
4. Blöcke **Point.ByCoordinates** und **Number Slider** erstellen und anschließend mit den Eingaben des Codeblocks verbinden
5. Als Ergebnis der Ausführung des visuellen Programms wird in der 3D-Vorschau ein Kreis ausgegeben.

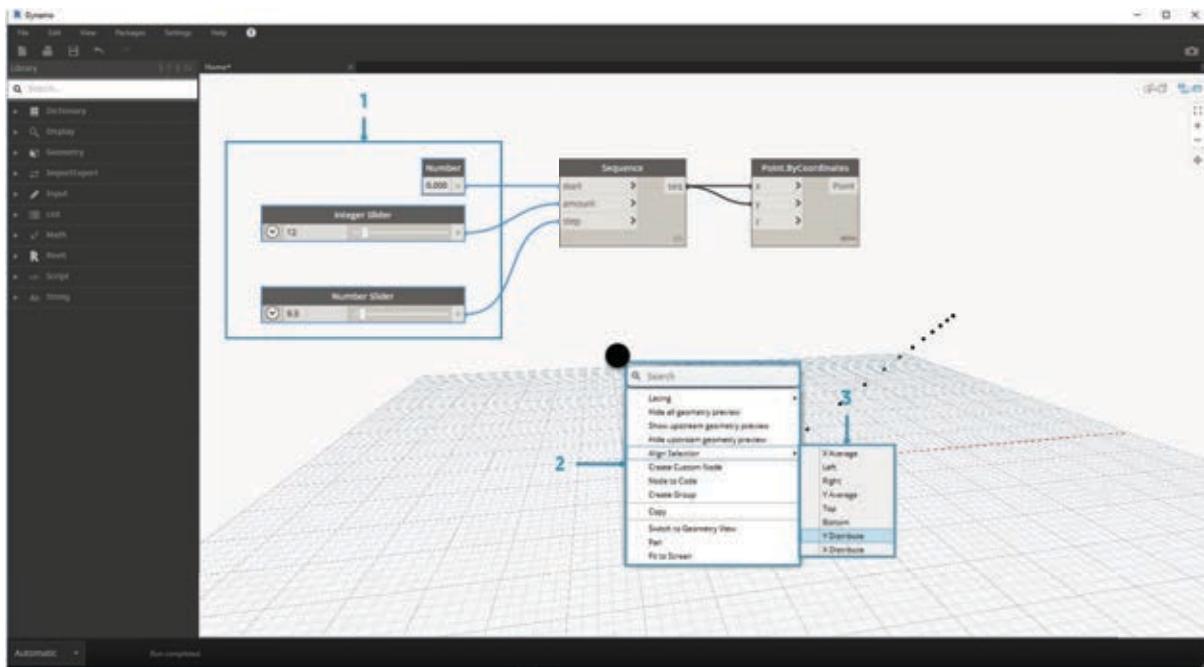
# **Programme verwalten**

## **Programme verwalten**

Der visuelle Programmierungsprozesses ist eine äußerst leistungsstarke, kreative Aktivität, wobei der Programmablauf und die wichtigsten Benutzereingaben jedoch schnell durch ihre Komplexität und/oder das Layout des Arbeitsbereichs unübersichtlich werden können. Machen Sie sich im Folgenden mit einigen bewährten Verfahren für die Verwaltung von Programmen vertraut.

### **Ausrichtung**

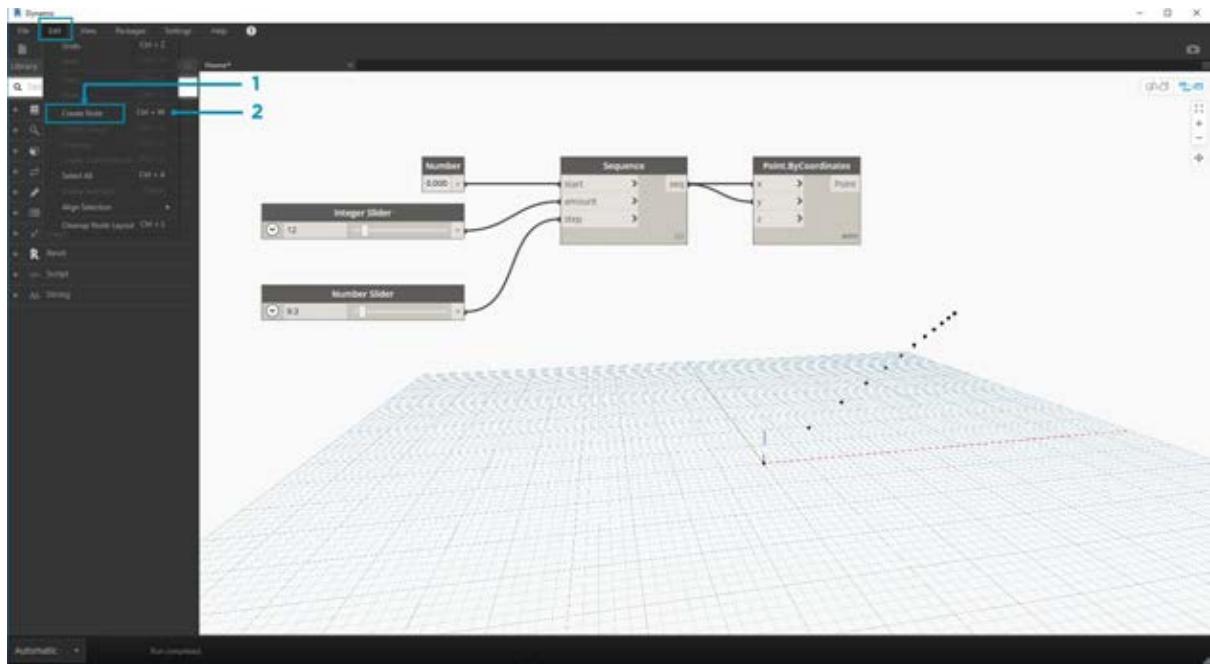
Nachdem Sie bereits zahlreiche Blöcke zum Arbeitsbereich hinzugefügt haben, möchten Sie sie möglicherweise neu anordnen, um das Layout übersichtlicher zu gestalten. Indem Sie mehrere Blöcke auswählen und mit der rechten Maustaste in den Arbeitsbereich klicken, wird ein Popup-Fenster mit dem Menü **Auswahl ausrichten** angezeigt, das Optionen zum Ausrichten und Verteilen in X- und Y-Richtung enthält.



1. Wählen Sie mehrere Blöcke aus.
2. Klicken Sie mit der rechten Maustaste in den Arbeitsbereich.
3. Verwenden Sie die Optionen von **Auswahl ausrichten**.

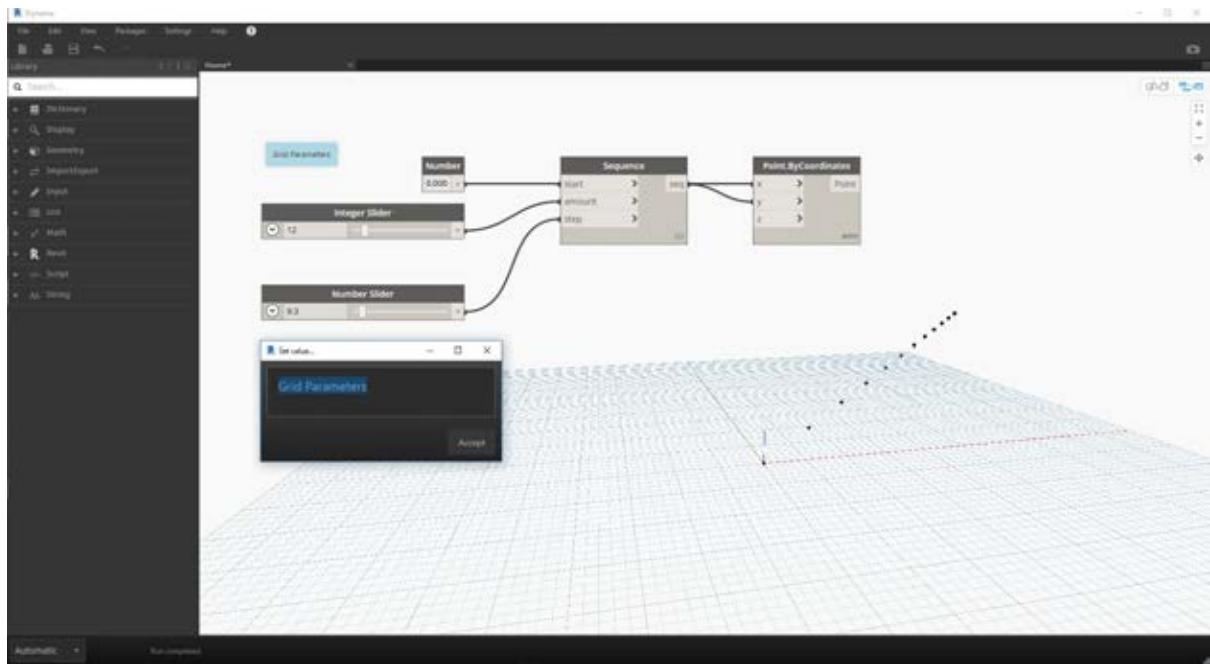
## Anmerkungen

Mit etwas Erfahrung werden Sie auch in der Lage sein, visuelle Programme zu "lesen", indem Sie die Blocknamen überprüfen und den Programmablauf verfolgen. Für Benutzer unterschiedlicher Erfahrungs niveaus hat es sich ebenfalls bewährt, aussagekräftige Beschriftungen und Beschreibungen einzufügen. In Dynamo ist hierfür ein **Notes**-Block mit einem bearbeitbaren Textfeld verfügbar. Für das Hinzufügen von Anmerkungen zum Arbeitsbereich bestehen zwei Möglichkeiten:



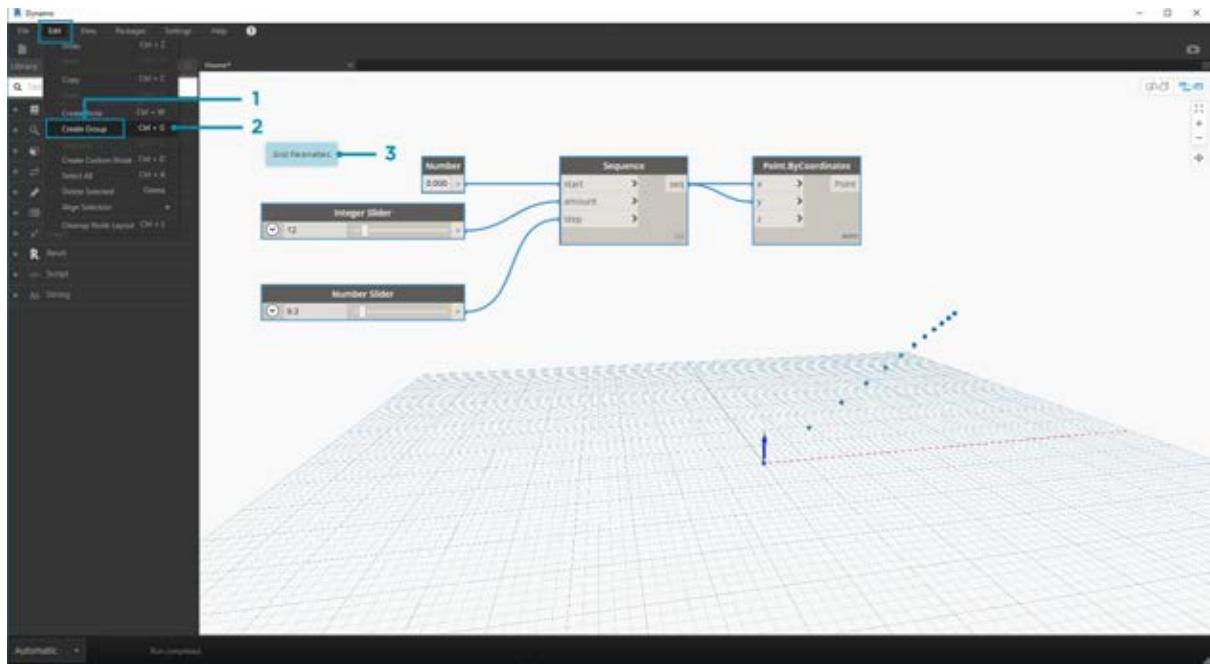
1. Navigieren Sie zum Menü Datei > Anmerkung erstellen.
2. Verwenden Sie die Tastenkombination Strg+W.

Nachdem Sie eine Anmerkung zum Arbeitsbereich hinzugefügt haben, wird ein Popup-Textfeld angezeigt, in dem Sie den Text für die Anmerkung bearbeiten können. Nach der Erstellung einer Anmerkung können Sie sie bearbeiten, indem Sie darauf doppelklicken oder mit der rechten Maustaste auf den Note-Block klicken.



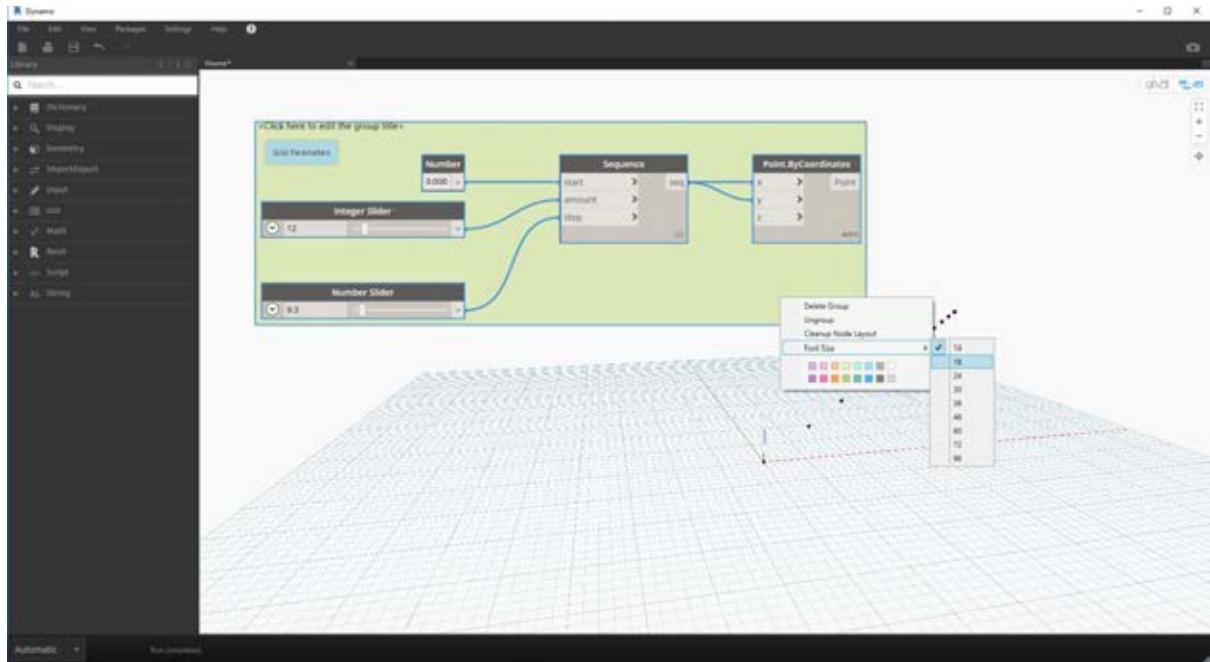
## Gruppieren

Je umfangreicher ein visuelles Programm wird, desto hilfreicher kann es sein, größere Schritte zu identifizieren, die ausgeführt werden. Sie können größere Sammlungen von Blöcken durch ein Rechteck mit farbigem Hintergrund und einen Titel zu einer **Gruppe** zusammenfassen. Für das Erstellen einer Gruppe mit mehreren ausgewählten Blöcken sind drei Möglichkeiten verfügbar:



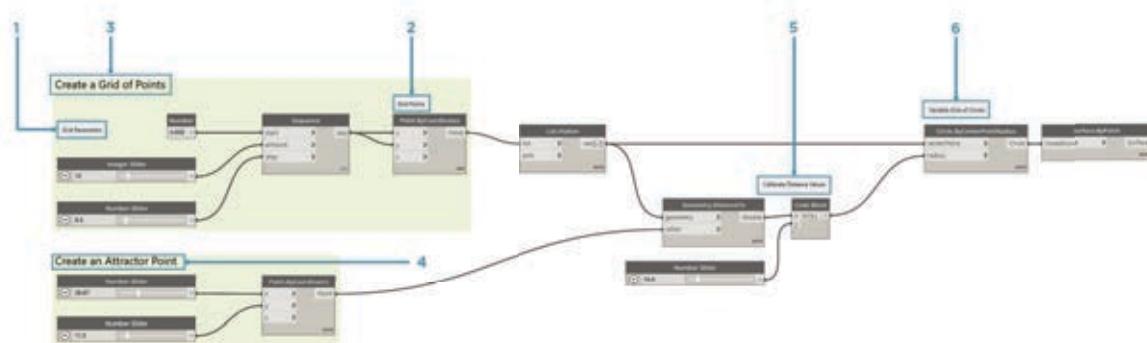
1. Navigieren Sie zum Menü Datei > Gruppe erstellen.
2. Verwenden Sie die Tastenkombination Strg+G.
3. Klicken Sie mit der rechten Maustaste in den Arbeitsbereich und wählen Sie "Gruppe erstellen".

Nachdem Sie eine Gruppe erstellt haben, können Sie deren Einstellungen wie den Titel und die Farbe bearbeiten.



Tipp: Verwenden Sie Anmerkungen und Gruppen auf effektive Weise, um Ihre Datei zu beschriften und die Lesbarkeit zu erhöhen.

Hier ist Ihr Programm aus Abschnitt 2.4 mit hinzugefügten Anmerkungen und Gruppen:



1. Anmerkung: "Rasterparameter"
2. Anmerkung: "Rasterpunkte"
3. Gruppe: "Raster aus Punkten erstellen"
4. Gruppe: "Attraktorpunkt erstellen"
5. Anmerkung: "Entfernungswerte kalibrieren"
6. Anmerkung: "Variables Raster von Kreisen"

## **Grundbausteine von Programmen**

### **Grundbausteine von Programmen**

Um die Entwicklung visueller Programme detaillierter kennenzulernen, benötigen Sie ein genaueres Verständnis der hierfür verwendeten Grundbausteine. In diesem Kapitel werden die Grundbegriffe zum Thema Daten vorgestellt: zu den Informationen, die durch die Verbindungen von Dynamo-Programmen geleitet werden.



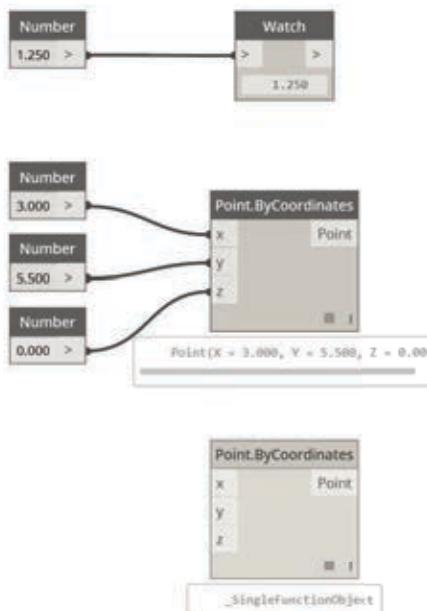
# Daten

## Daten

Daten sind das Material unserer Programme. Sie fließen als Eingaben durch "Drähte" in Blöcke, wo sie verarbeitet und in eine neue Form von Ausgabedaten umgewandelt werden. Hier sollen zunächst die Definition und Struktur von Daten betrachtet werden, bevor Sie damit beginnen, sie in Dynamo zu verwenden.

### Was sind Daten?

Unter Daten versteht man eine Gruppe von Werten für qualitative oder quantitative Variable. Die einfachste Form von Daten sind Zahlen wie z. B. 0, 3 . 14 oder 17. Es gibt jedoch eine Reihe unterschiedlicher Datentypen: Variable, die für veränderliche Zahlen stehen (Höhe); Zeichen (meinName); Geometrie (Kreis) oder eine Liste von Datenelementen (1, 2, 3, 5, 8, 13, ...). Daten müssen in die Eingabeanschlüsse der Blöcke in Dynamo eingegeben werden: Daten können ohne Aktionen existieren, aber die Aktionen, die durch die Blöcke dargestellt werden, können nur durchgeführt werden, wenn Daten vorhanden sind. Wenn Sie im Arbeitsbereich einen Block hinzufügen, ohne Eingaben bereitzustellen, ist das Ergebnis eine Funktion, nicht das Ergebnis der eigentlichen Aktion.



1. Einfache Daten
2. Daten und Aktion (Block): erfolgreiche Ausführung
3. Aktion (Block) ohne Daten gibt eine allgemeine Funktion zurück

## Vermeiden Sie Nullen

Der Typ 'null' steht für das Fehlen von Daten. Dies ist zwar ein abstraktes Konzept, dem Sie jedoch bei der Arbeit mit visueller Programmierung sehr wahrscheinlich begegnen werden. Wenn eine Aktion kein gültiges Ergebnis erstellt, gibt der Block Null zurück. Das Ermitteln und Entfernen von Nullwerten aus Datenstrukturen ist ein entscheidender Schritt zum Erstellen stabiler Programme.

**Symbol**      **Name/Syntax**      **Eingaben**      **Ausgaben**



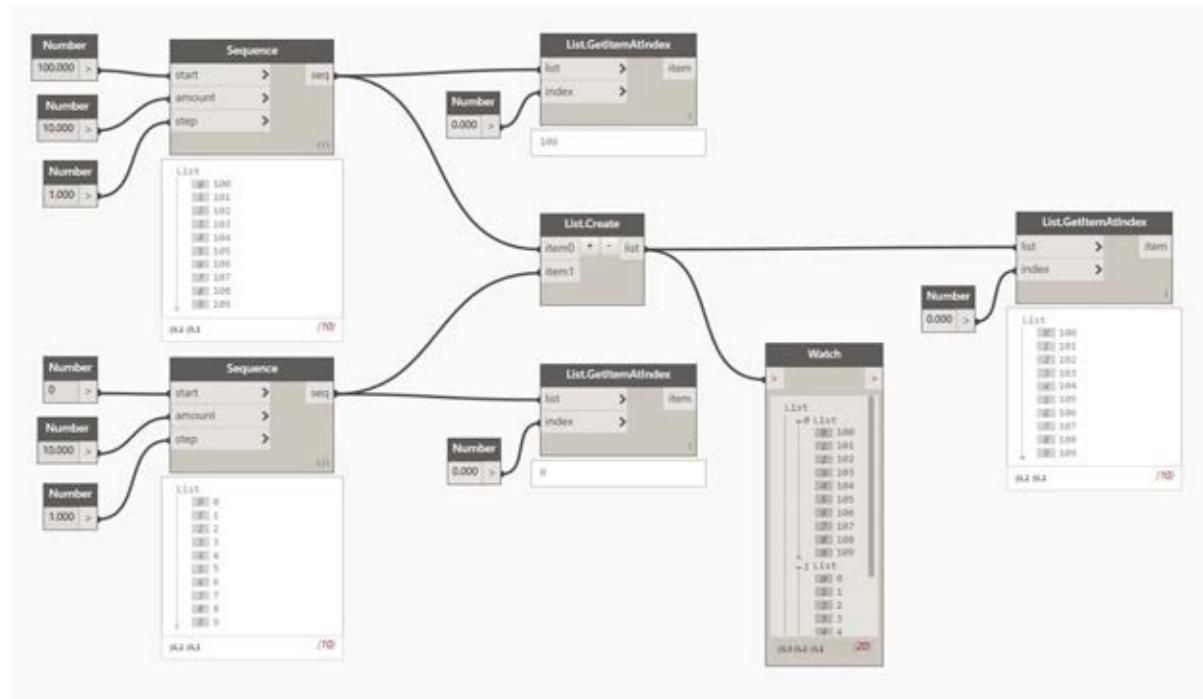
Object.IsNull obj bool

## Datenstrukturen

Bei der visuellen Programmierung können sehr schnell große Datenmengen generiert werden. Aus diesem Grund benötigen Sie ein Verfahren zur Verwaltung von deren Hierarchie. Dies ist die Funktion der Datenstrukturen, der organisatorischen Schemata, in denen Daten gespeichert werden. Die Charakteristika der Datenstrukturen und ihrer Verwendung sind von Programmiersprache zu Programmiersprache unterschiedlich. In Dynamo werden Daten mithilfe von Listen hierarchisch geordnet. Dies wird in den weiteren Kapiteln ausführlich erläutert. Am Anfang sollen einige einfache Beispiele stehen:

Eine Liste für eine Sammlung von Elementen, die in einer Struktur von Daten abgelegt wurden:

- Sie haben fünf Finger (*Elemente*) an einer Hand (*Liste*).
- Zehn Häuser (*Elemente*) stehen entlang einer Straße (*Liste*).



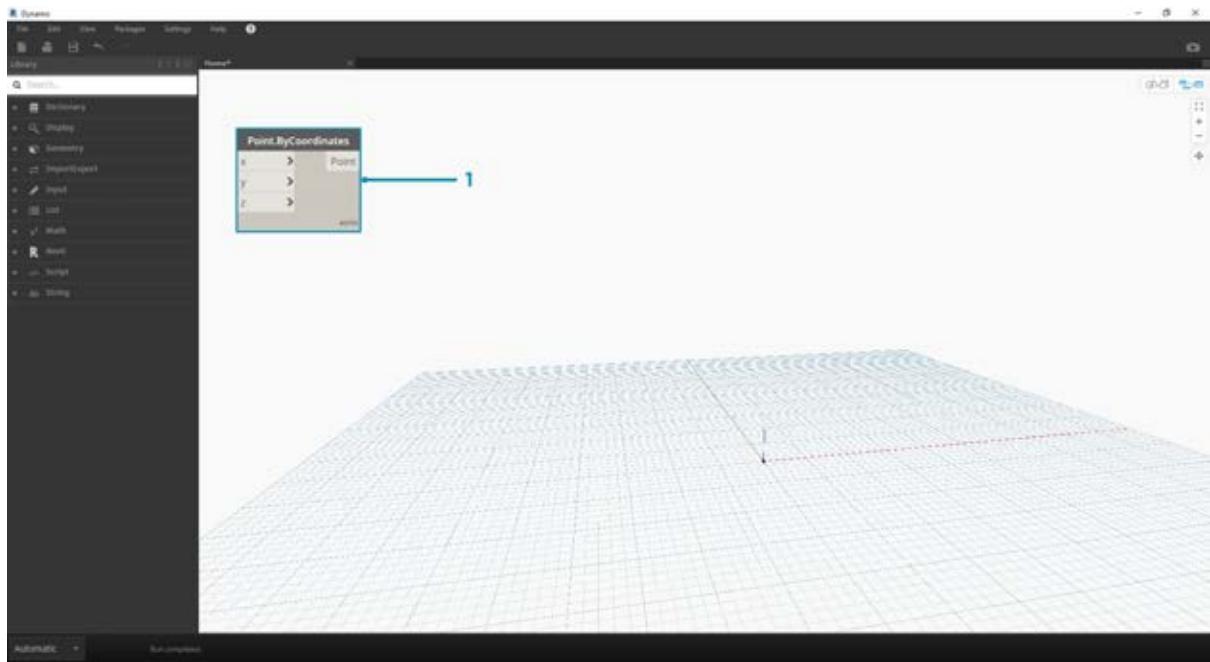
1. Ein **Number Sequence**-Block definiert eine Liste von Zahlen mithilfe der Eingaben *start*, *amount* und *step*. Mithilfe dieser beiden Blöcke wurden zwei separate Listen mit je zehn Zahlen – 100–109 und 0–9 erstellt.
2. Mithilfe des Blocks **List.GetItemAtIndex** wird das Element an einer bestimmten Indexposition ausgewählt. Wenn Sie 0 wählen, wird das erste Element in der Liste (in diesem Fall 100) abgerufen.
3. In der zweiten Liste erhalten Sie mit demselben Verfahren den Wert 0, das erste Element in der Liste.
4. Als Nächstes führen Sie die beiden Listen mithilfe eines **List.Create**-Blocks zu einer zusammen. Mit diesem Block wird eine *Liste von Listen erstellt*. Dies verändert die Struktur der Daten.
5. Wenn Sie **List.GetItemAtIndex** erneut mit dem Index 0 verwenden, erhalten Sie die erste der in der übergeordneten Liste enthaltenen Listen. Dieses Beispiel verdeutlicht, wie Listen behandelt werden: Sie gelten – anders als in anderen Skriptsprachen – als Objekte. Listenbearbeitung und Datenstruktur werden in den später folgenden Kapiteln genauer beschrieben.

Als wichtigstes Prinzip zum Verständnis der Datenhierarchie in Dynamo gilt: **Innerhalb der Datenstruktur gelten Listen als Elemente**. In anderen Worten: In Dynamo kommt ein hierarchisch von oben nach unten geordneter Prozess für Datenstrukturen zum Einsatz. Was bedeutet das? Das folgende Beispiel soll dies verdeutlichen:

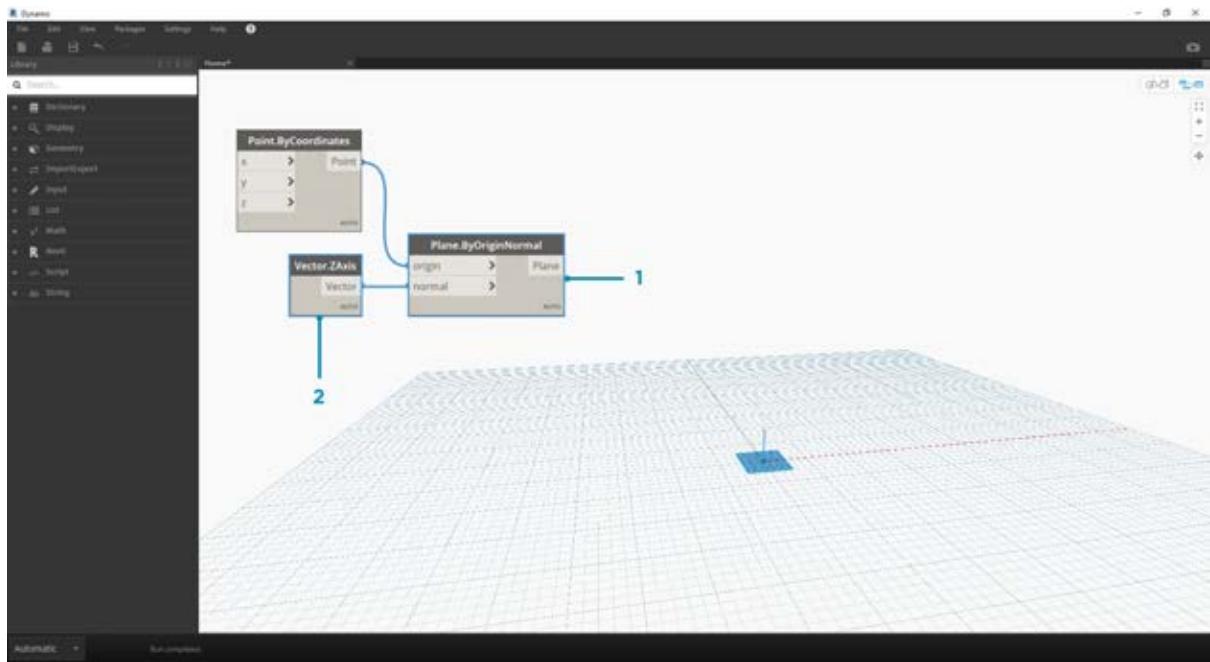
### Erstellen einer Kette von Zylindern mithilfe von Daten

Laden Sie die Beispieldatei für diese Übungslektion herunter (durch Rechtsklicken und Wahl von "Save Link As..."):  
[Building Blocks of Programs - Data.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

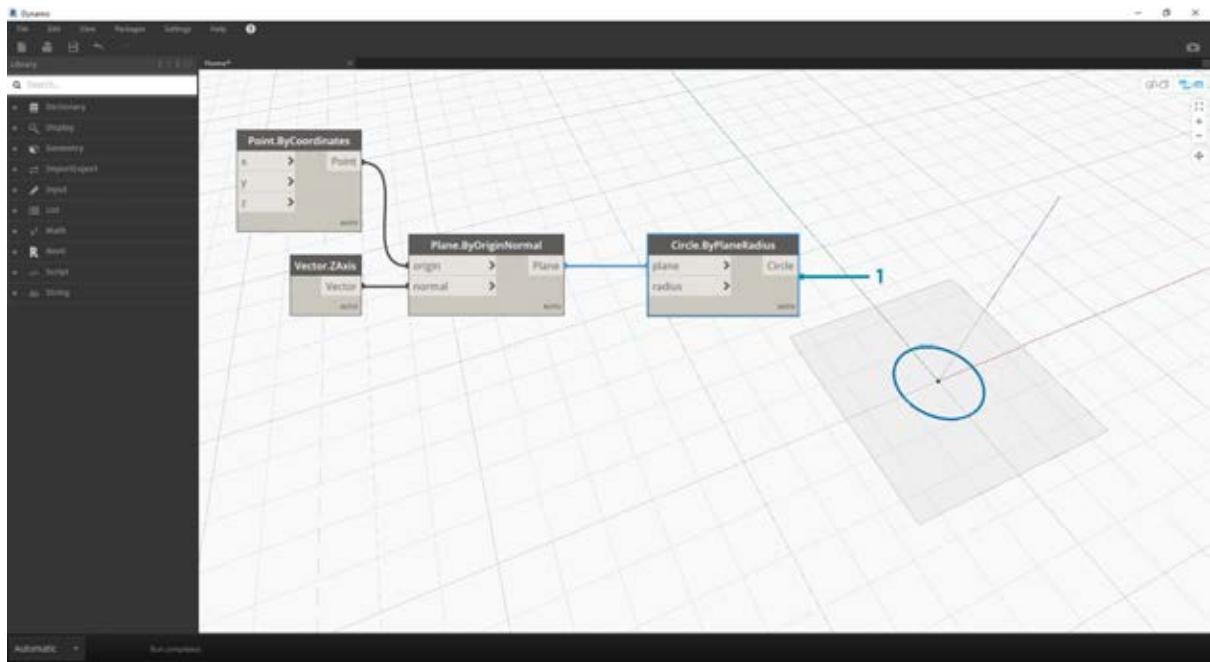
In diesem ersten Beispiel erstellen Sie zur Demonstration der in diesem Abschnitt behandelten Geometriehierarchie einen Zylinder und geben seine Wandstärke an.



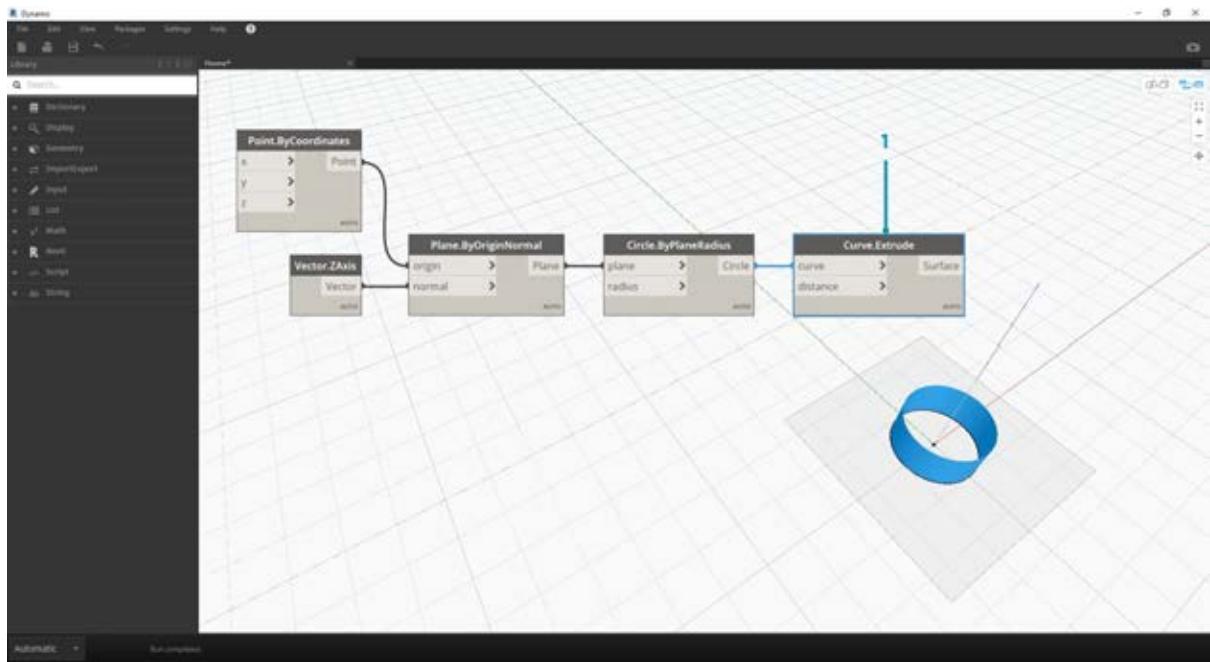
1. **Point.ByCoordinates:** Nachdem Sie diesen Block im Ansichtsbereich hinzugefügt haben, wird ein Punkt am Ursprung des Rasters in der Dynamo-Vorschau angezeigt. Die Vorgabewerte der Eingaben für  $x$ ,  $y$  und  $z$  sind  $0.0$ , wodurch ein Punkt an dieser Position definiert wird.



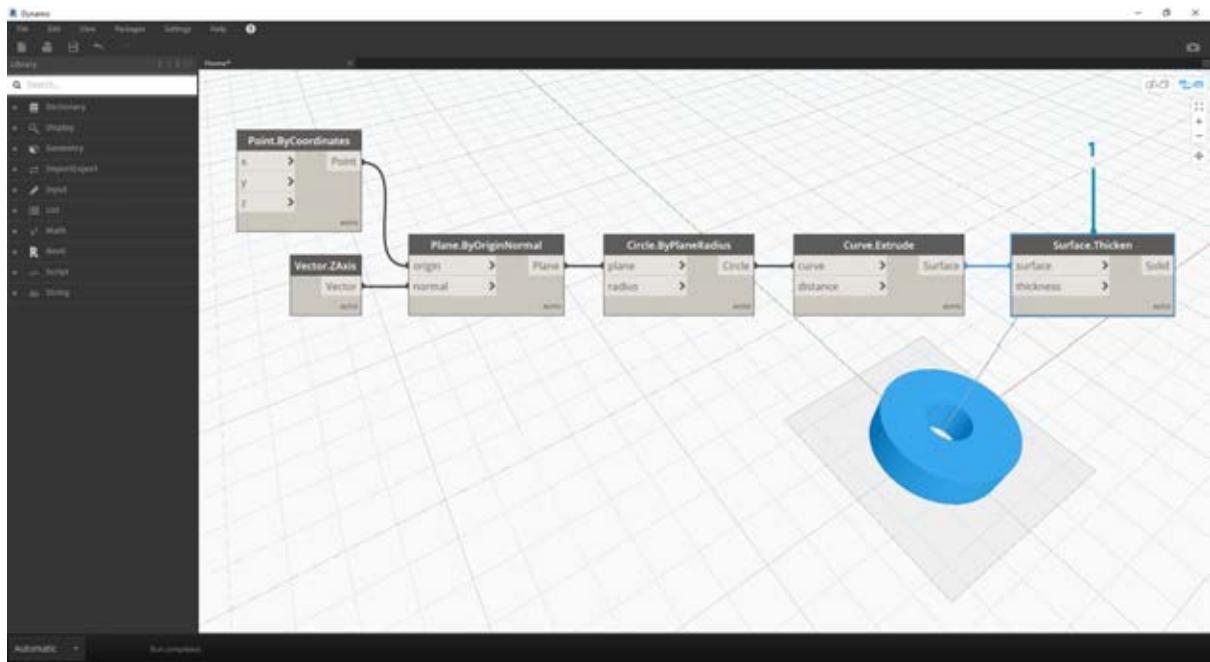
1. **Plane.ByOriginNormal:** Die nächste Stufe in der Geometriehierarchie ist eine Ebene. Es gibt mehrere Möglichkeiten für die Konstruktion einer Ebene: Hier werden ein Ursprung und eine Normale als Eingaben verwendet. Der Ursprung ist der Punkt, den Sie mithilfe des Blocks im vorigen Schritt erstellt haben.
2. **Vector.ZAxis:** Dies ist ein Vektor mit Einheiten in z-Richtung. Hier sind keine Eingaben, sondern nur ein Vektor mit dem Wert [0,0,1] vorhanden. Dieser wird für die *normal*-Eingabe des *Plane.ByOriginNormal*-Blocks verwendet. In der Dynamo-Vorschau wird daraufhin eine rechteckige Ebene angezeigt.



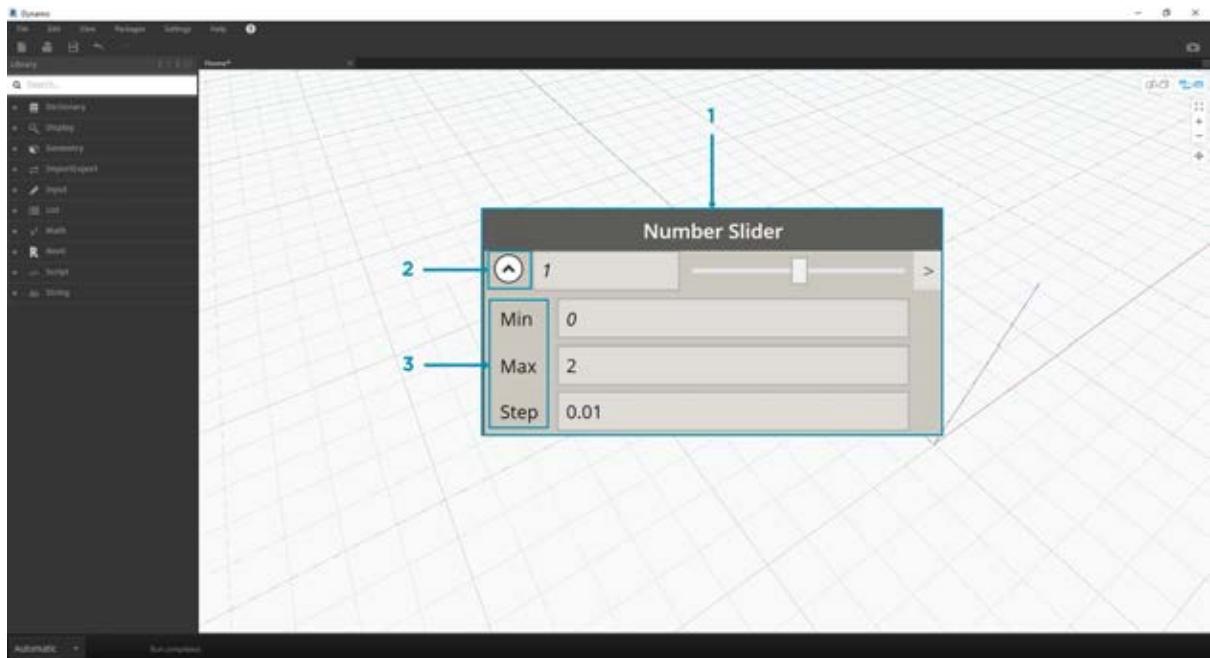
1. **Circle.ByPlaneRadius:** Eine Stufe höher in der Hierarchie erstellen Sie aus der Ebene, die Sie im letzten Schritt erstellt haben, eine Kurve. Nachdem Sie den Block verbunden haben, erhalten Sie einen Kreis um den Ursprungspunkt. Als Radius ist in diesem Block der Wert 1 vorgegeben.



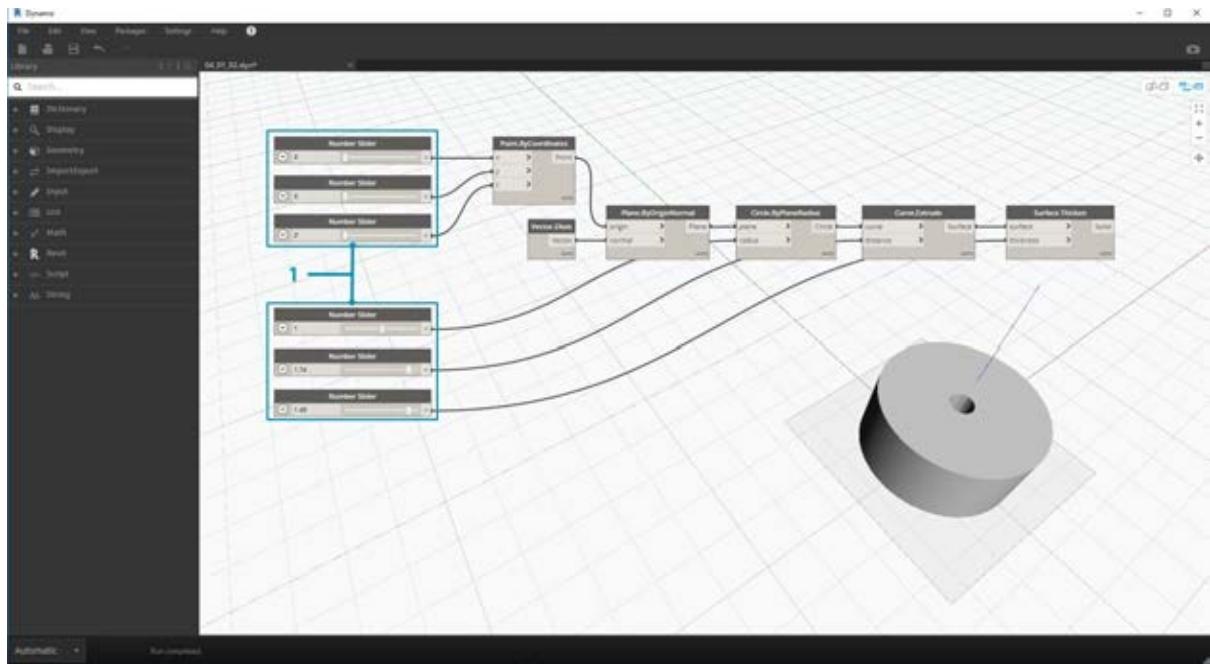
1. **Curve.Extrude:** In diesem Schritt wandeln Sie dieses Objekt in einen 3D-Körper um. Dieser Block erstellt durch Extrusion eine Oberfläche aus einer Kurve. Der vorgegebene Abstand im Block beträgt 1 und im Ansichtsfenster ist jetzt ein Zylinder zu sehen.



1. **Surface.Thicken**: Mit diesem Block erhalten Sie einen geschlossenen Körper, indem die Oberfläche um den angegebenen Wert versetzt und die entstehende Form geschlossen wird. Der Vorgabewert für die Verdickung ist **1**. Im Ansichtsfenster wird ein Zylinder mit der entsprechenden Wandstärke angezeigt.

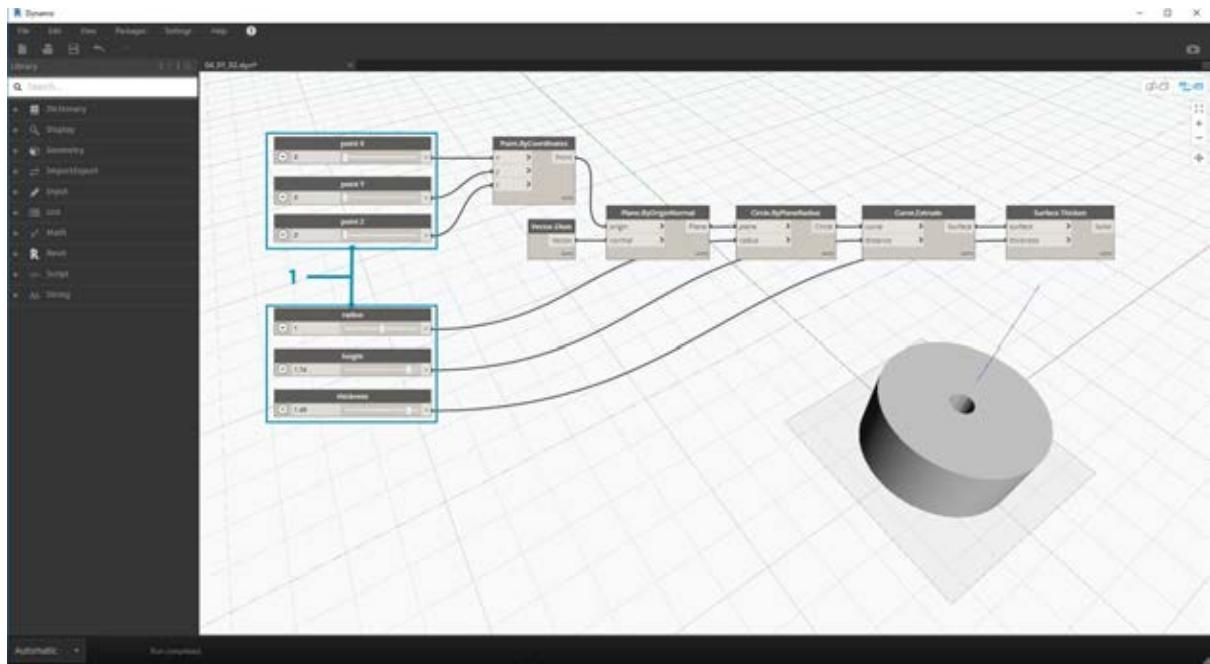


1. **Number Slider:** Anstatt die Vorgabewerte für alle diese Eingaben zu verwenden, können Sie dem Modell auch Funktionen zur parametrischen Steuerung hinzufügen.
2. **Domänenbearbeitung:** Nachdem Sie den Zahlen-Schieberegler im Ansichtsbereich hinzugefügt haben, klicken Sie auf die Pfeilspitze oben links, um die Domänenoptionen anzuzeigen.
3. **Min/Max/Step:** Ändern Sie die Werte für *min*, *max* und *step* in 0,2 und 0,01. Mit diesen Angaben steuern Sie die Gesamtgröße der Geometrie.



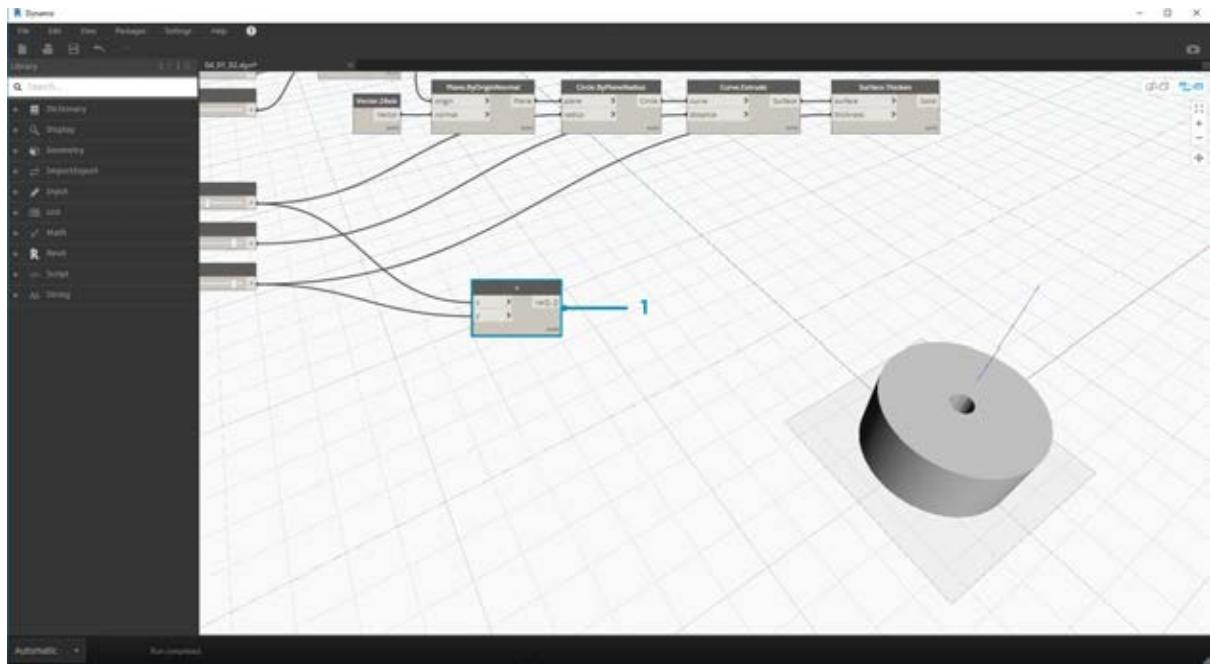
- Number Slider-Blöcke:** Erstellen Sie mehrere Kopien dieses Number Slider-Blocks (indem Sie ihn auswählen und anschließend zuerst Strg+C und dann Strg+V drücken), bis für alle Eingaben anstelle der bisherigen Vorgaben Verbindungen zu den Number Sliders verwendet werden. Manche Werte für diese Schieberegler müssen größer als Null sein, damit die Definition verwendbar ist. So wird beispielsweise eine Tiefe für die Extrusion benötigt, damit eine Oberfläche entsteht, die verdickt werden kann.

Sie haben jetzt mithilfe dieser Regler einen parametrischen Zylinder mit Angabe einer Wandstärke erstellt. Experimentieren Sie mit einigen dieser Parameter und beobachten Sie die dynamische Veränderung der Geometrie im Dynamo-Ansichtsfenster.

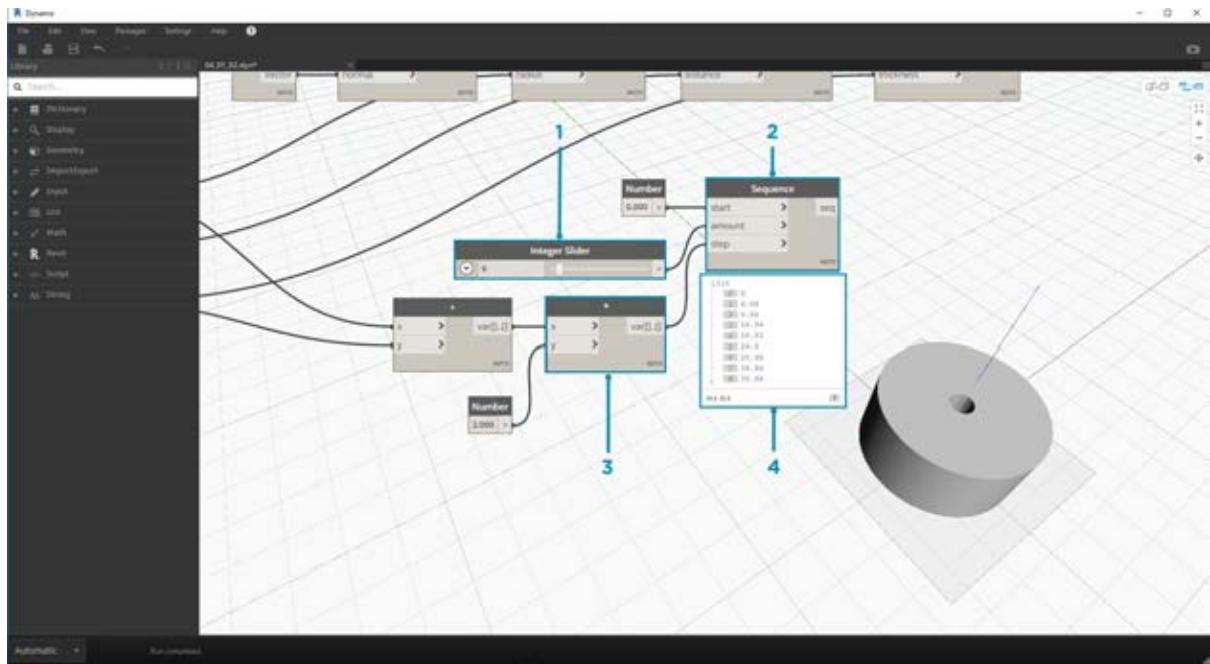


- Number Sliders-Blöcke:** In einem weiteren Schritt wurden etliche weitere Schiebereglern im Ansichtsbereich hinzugefügt und die Benutzeroberfläche des neu erstellten Werkzeugs muss übersichtlicher gestaltet werden. Klicken Sie mit der rechten Maustaste nacheinander auf die einzelnen Schiebereglern, wählen Sie Umbenennen und ändern Sie den Namen jedes Schiebereglers in den Namen des dazugehörigen Parameters. Die Namen können Sie der Abbildung oben entnehmen.

Damit haben Sie einen perfekten verdickten Zylinder erstellt. Dies ist jedoch nur ein einzelnes Objekt. Im nächsten Schritt wird gezeigt, wie Sie ein Array von Zylindern erstellen, die dynamisch miteinander verbunden bleiben. Zu diesem Zweck arbeiten Sie nicht mit einem einzelnen Element, sondern erstellen eine Liste von Zylindern.

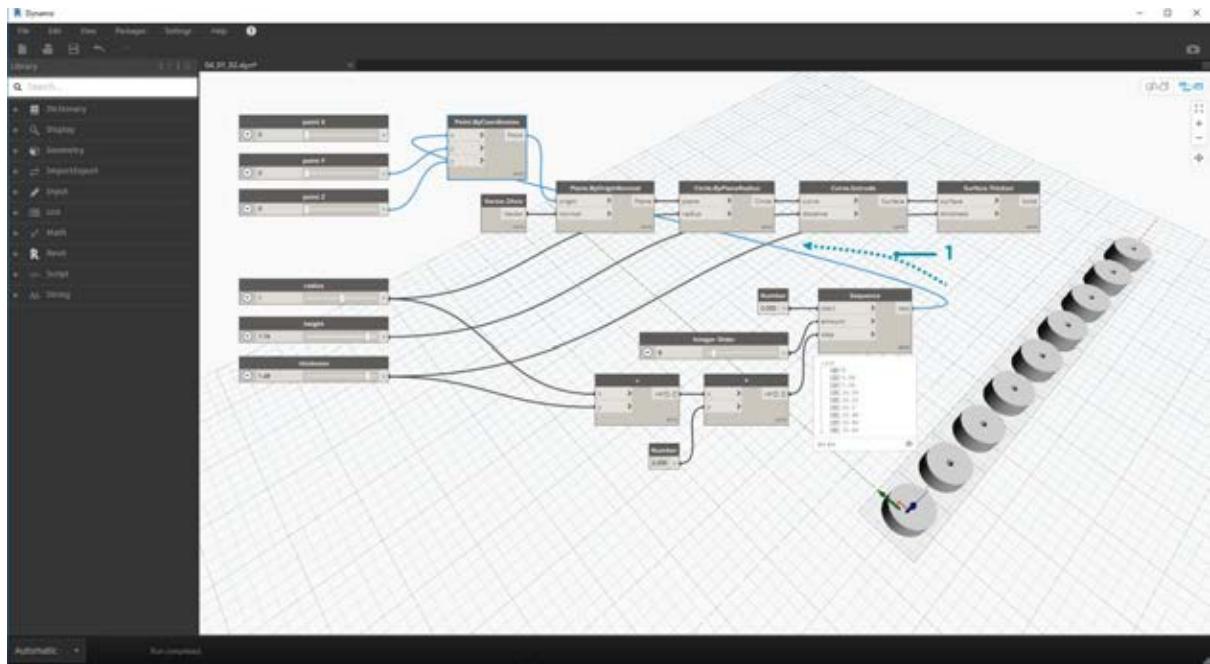


1. **Addition (+):** Neben dem eben erstellten Zylinder soll eine Reihe weiterer Zylinder hinzugefügt werden. Um einen Zylinder direkt neben dem vorhandenen hinzuzufügen, müssen Sie sowohl dessen Radius als auch die Stärke seiner Wand berücksichtigen. Diesen Wert erhalten Sie durch Addition der Werte aus den beiden Schieberegeln.

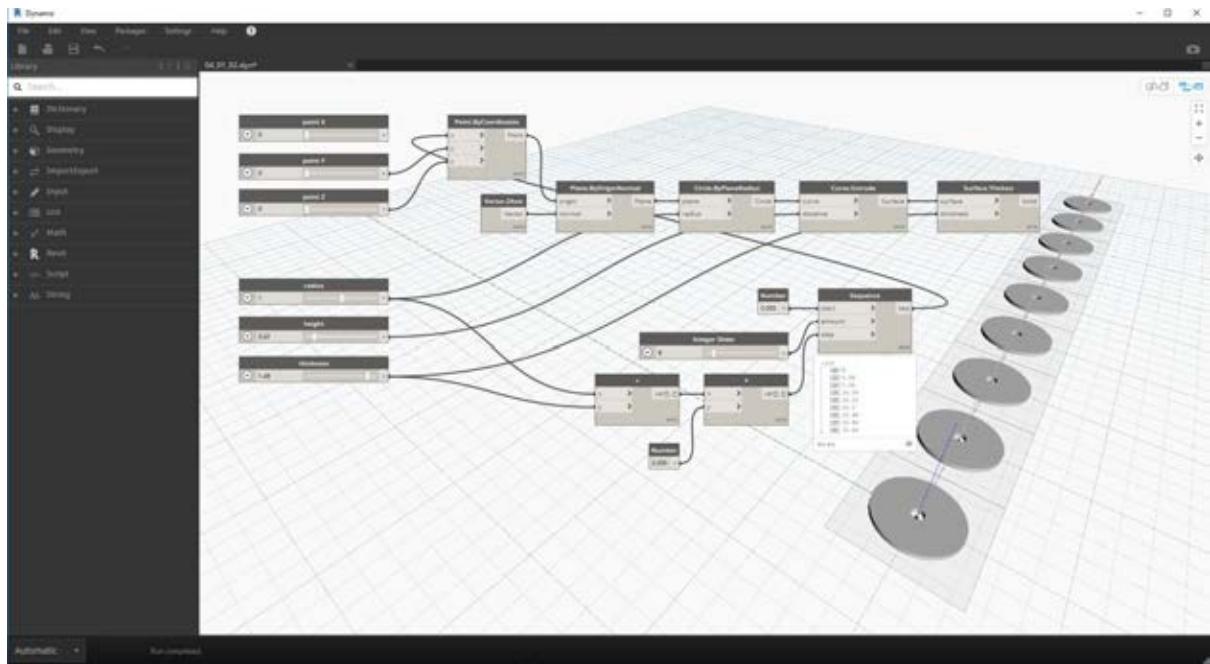


Dieser Schritt ist etwas komplexer und wird daher im Einzelnen erläutert: Erstellt werden soll eine Liste mit Zahlen, die die Positionen der einzelnen Zylinder in einer Reihe definieren.

- Multiplikation:** Als Erstes müssen Sie den Wert aus dem vorigen Schritt mit 2 multiplizieren. Der Wert aus dem vorherigen Schritt gibt den Radius an, der Zylinder muss jedoch um den vollständigen Durchmesser verschoben werden.
- Sequence:** Mithilfe dieses Blocks erstellen Sie ein Array von Zahlen. Die erste Eingabe ist der Wert des *Multiplikation*-Blocks aus dem vorigen Schritt in den *step*-Wert. Als *start*-Wert können Sie *0.0* festlegen. Verwenden Sie hierfür einen *number*-Block.
- Integer Slider:** Verbinden Sie für den *amount*-Wert einen Integer-Schieber. Diese Funktion definiert, wie viele Zylinder erstellt werden.
- Ausgabe:** Diese Liste zeigt den Abstand, um den die einzelnen Zylinder im Array versetzt werden. Sie wird parametrisch durch die ursprünglichen Schieberegler gesteuert.



1. Dieser Schritt ist einfach: Verbinden Sie die im letzten Schritt erstellte Sequenz mit der x-Eingabe des ursprünglichen Point.ByCoordinates. Dies ersetzt den Schieberegler pointX, den Sie löschen können. Im Ansichtsfenster wird jetzt ein Array von Zylindern angezeigt (achten Sie darauf, dass im Integer Slider ein Wert größer als Null angegeben ist).



Die Kette der Zylinder ist weiterhin dynamisch mit allen Schiebereglern verknüpft. Experimentieren Sie mit den Schiebereglern, und beobachten Sie, wie die Definition aktualisiert wird.

# Math

## Math

Zahlen stellen die einfachste Form von Daten dar und sind am einfachsten durch mathematische Operationen zu verknüpfen. Von einfachen Operatoren etwa zur Division über trigonometrische Funktionen bis hin zu komplexen Formeln: Mathematische Operationen eignen sich ausgezeichnet zur Analyse von Zahlenverhältnissen und -mustern.

### Arithmetische Operatoren

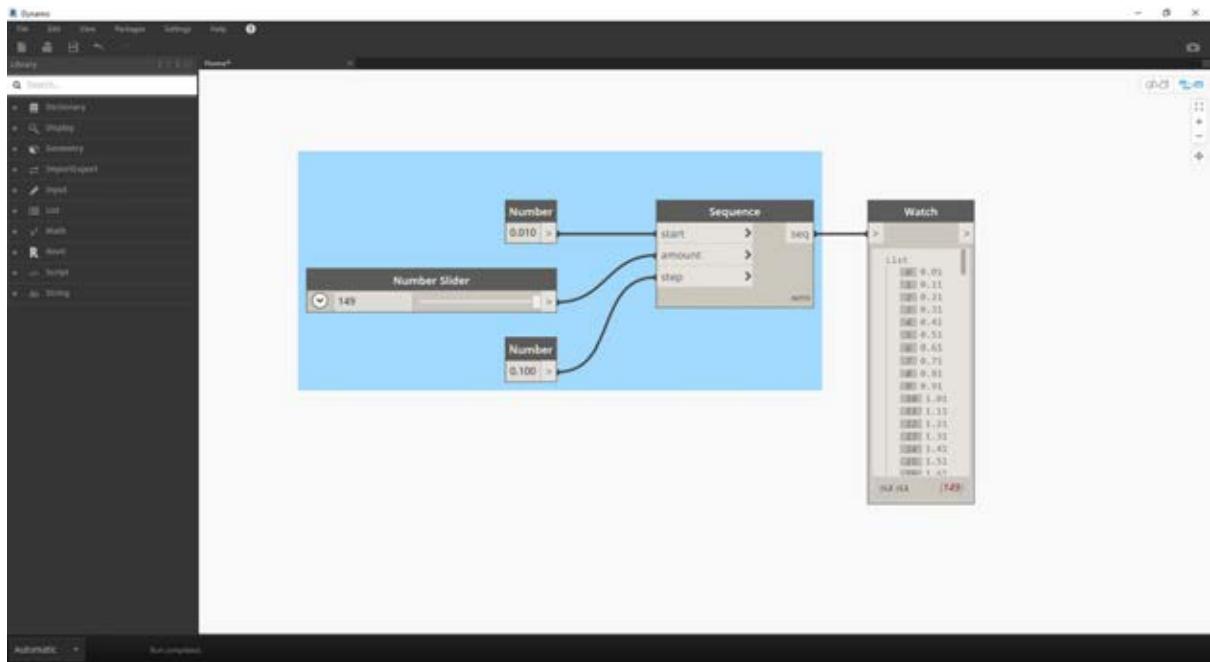
Operatoren sind Komponenten für algebraische Funktionen, die zwei Eingabewerte benötigen, um einen Ausgabewert zu erhalten (etwa bei der Addition, Subtraktion, Multiplikation, Division usw.). Sie finden die Operatoren unter Operators > Actions.

Symbol	Name	Syntax	Eingaben	Ausgaben
	Addition	+	var[...[], var[...[], var[...[]]	
	Subtraktion	-	var[...[], var[...[], var[...[]]	
	Multiplikation	*	var[...[], var[...[], var[...[]]	
	Division	/	var[...[], var[...[], var[...[]]	

### Parametrische Formeln

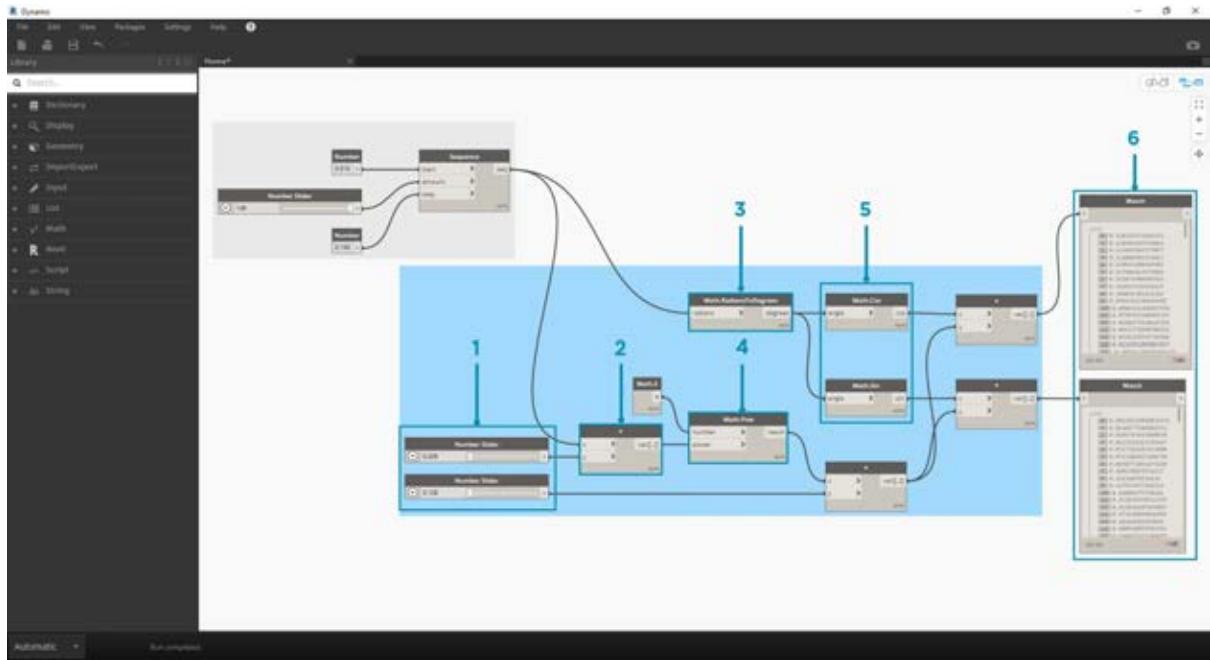
Laden Sie die Beispieldatei für diese Übungslektion herunter (durch Rechtsklicken und Wahl von "Save Link As..."):  
[Building Blocks of Programs - Math.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

Es ist logisch, im nächsten Schritt Operatoren und Variable in **Formeln** zu kombinieren, um komplexere Beziehungen zu erstellen. In diesem Beispiel erstellen Sie eine Formel, die durch Eingabeparameter, z. B. über Schieberegler, gesteuert werden kann.



1. **Sequence:** Definieren Sie eine Zahlenfolge mithilfe von drei Eingaben: *start*, *amount* und *step*. Diese Folge steht für die Angabe "t" in der parametrischen Gleichung. Sie benötigen daher eine Liste mit genügend Werten zum Definieren einer Spirale.

Mit dem oben beschriebenen Schritt haben Sie eine Liste von Zahlen erstellt, die die parametrische Domäne definieren. Die Goldene Spirale ist durch die Gleichungen  $x = r \cos \theta = a \cos \theta e^{b\theta}$  und  $y = r \sin \theta = a \sin \theta e^{b\theta}$  definiert. Die unten gezeigte Gruppe von Blöcken zeigt die Darstellung dieser Gleichung in visueller Programmierung.

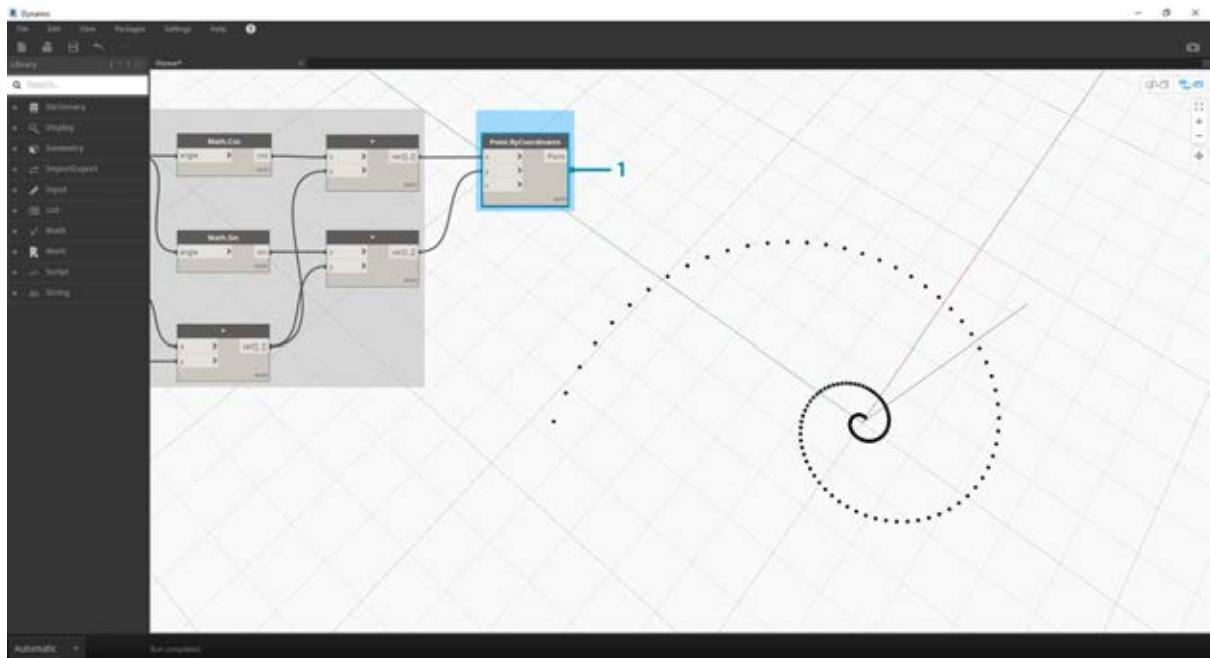


Beachten Sie bei der Betrachtung dieser Blockgruppe die Entsprechungen zwischen dem visuellen Programm und der schriftlichen Gleichung.

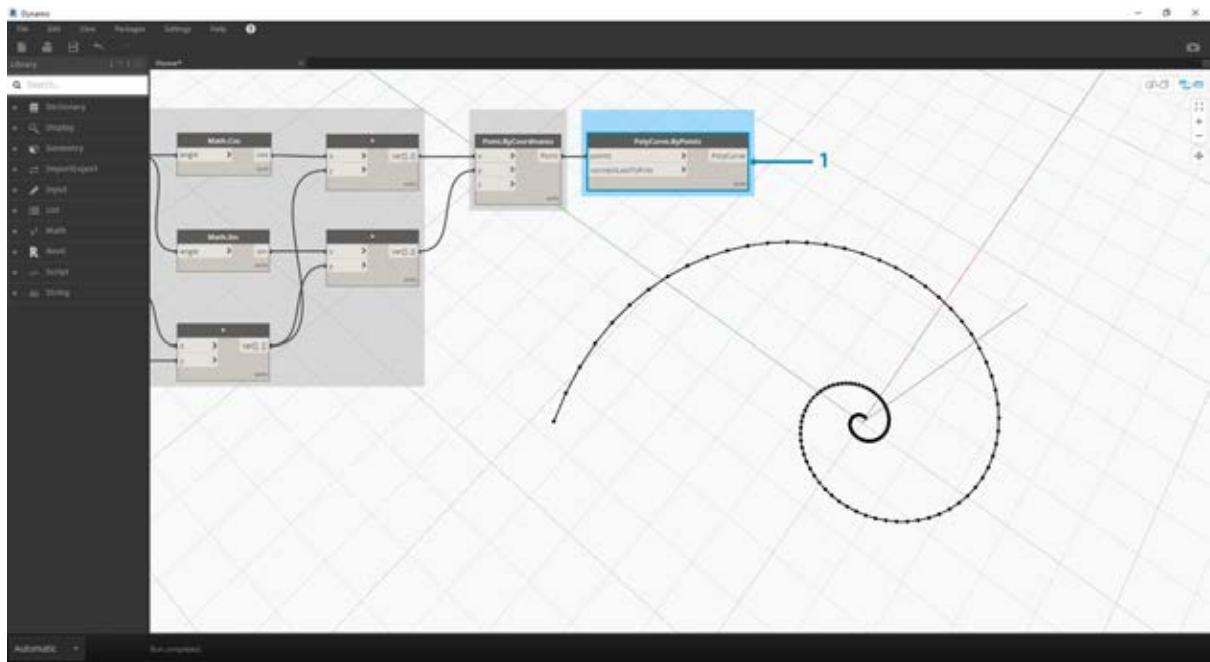
1. **Number Slider:** Fügen Sie im Ansichtsbereich zwei Zahlen-Schiebereglern ein. Diese Schieberegler steuern die Variablen  $a$  und  $b$  in der parametrischen Gleichung. Sie stehen für einstellbare Konstanten bzw. Parameter, die Sie anpassen können, um das gewünschte Ergebnis zu erhalten.
2. **\*:** Der Block für die Multiplikation wird mit einem Asterisk dargestellt. Er kommt hier mehrmals zum Einsatz, um Variable miteinander zu multiplizieren.
3. **Math.RadiansToDegrees:** Die Werte für " $t$ " müssen in Grad umgewandelt werden, damit sie in den trigonometrischen Funktionen verwendet werden können. Dynamo verwendet per Vorgabe Grad zur Auswertung dieser Funktionen.
4. **Math.Pow:** Als Funktion von " $t$ " und der Zahl " $e$ " erstellt dieser Block die Fibonacci-Folge.
5. **Math.Cos und Math.Sin:** Diese beiden trigonometrischen Funktionen differenzieren die x- und y-Koordinaten der einzelnen parametrischen Punkte.
6. **Watch:** Als Ausgabe erhalten Sie zwei Listen mit Werten für die x- und y-Koordinaten der Punkte, aus denen die Spirale erstellt wird.

## Erstellen von Geometrie aus der Formel

Die Gruppe von Blöcken aus dem letzten Schritt funktioniert einwandfrei, erfordert jedoch erheblichen Aufwand. Einen effizienteren Arbeitsablauf finden Sie unter **Codeblöcke** (Abschnitt 3.3.2.3). Dort wird beschrieben, wie Sie eine Reihe von Dynamo-Ausdrücken in ein und demselben Block definieren können. In den nächsten Schritten zeichnen Sie mithilfe der parametrischen Gleichung die Fibonacci-Spirale.



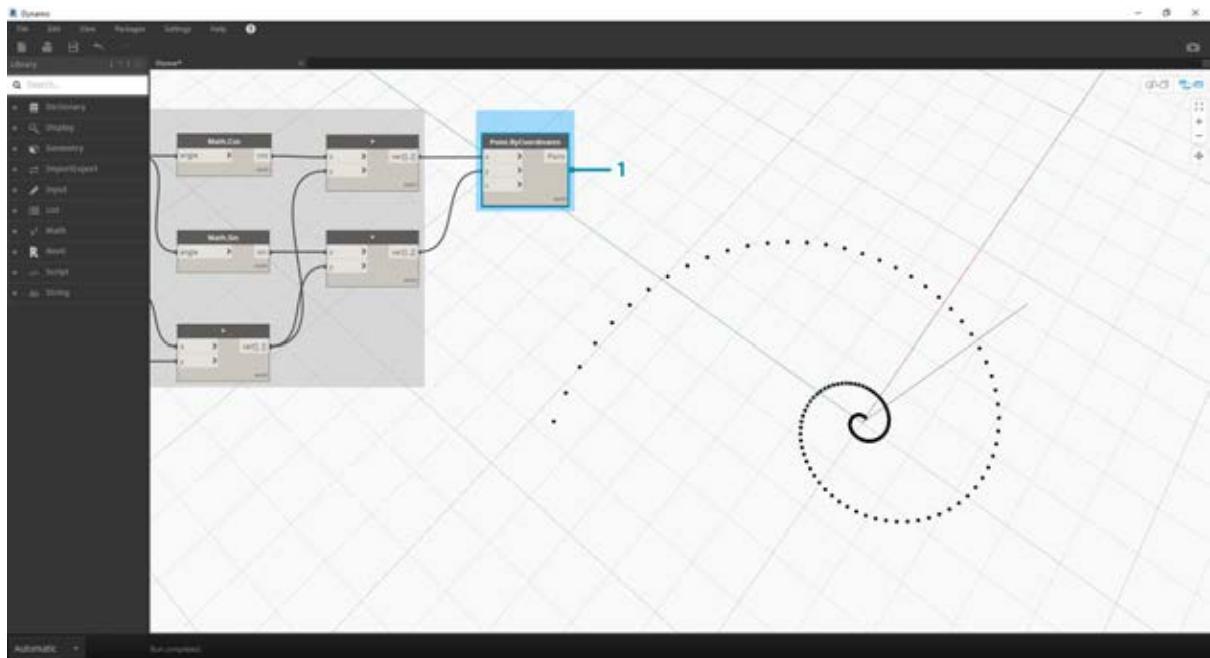
1. **Point.ByCoordinates:** Verbinden Sie den oberen Multiplikationsblock mit der x-Eingabe und den unteren mit der y-Eingabe. Dadurch wird auf dem Bildschirm eine parametrische Spirale aus Punkten angezeigt.



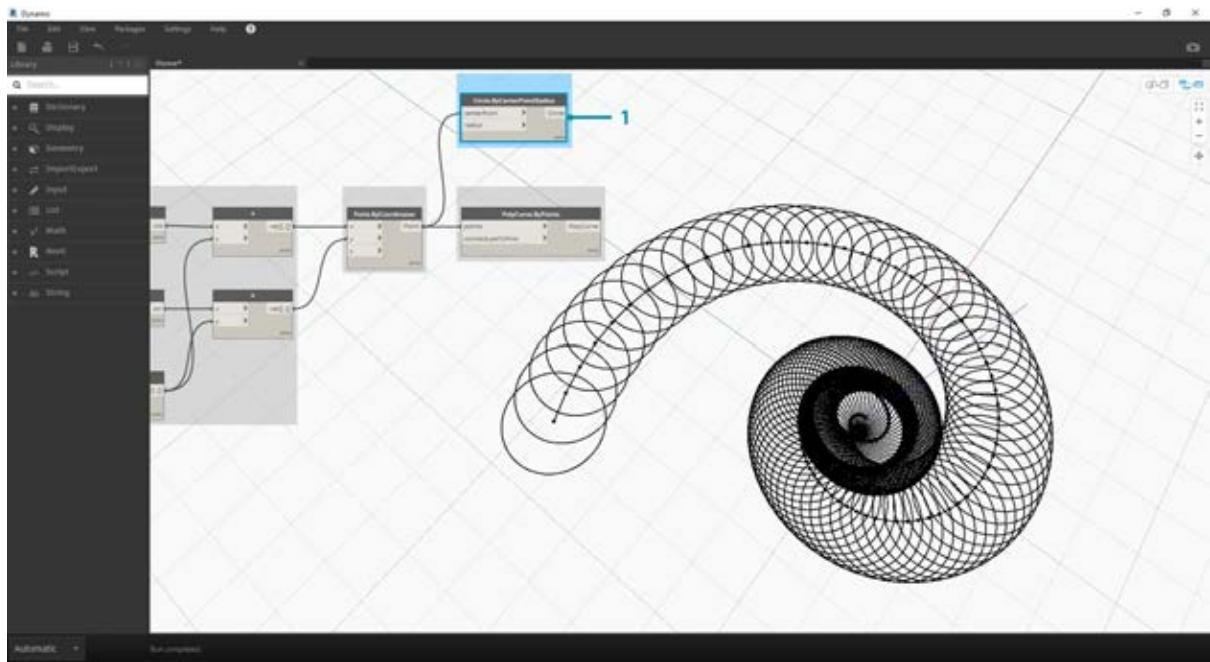
1. **Polycurve.ByPoints:** Verbinden Sie Point.ByCoordinates aus dem vorigen Schritt mit *points*. Für *connectLastToFirst* wird keine Eingabe benötigt, da Sie keine geschlossene Kurve erstellen. Dadurch wird eine durch die im vorigen Schritt erstellten Punkte verlaufende Spirale erstellt.

Damit haben Sie die Fibonacci-Spirale erstellt. Dies entwickeln Sie in zwei weiteren Übungen weiter, die hier als "Nautilus" und "Sonnenblume" bezeichnet werden. Dabei handelt es sich um Abstraktionen aus Systemen, die in der Natur vorkommen und gute Beispiele für zwei verschiedene Verwendungsweisen der Fibonacci-Spirale darstellen.

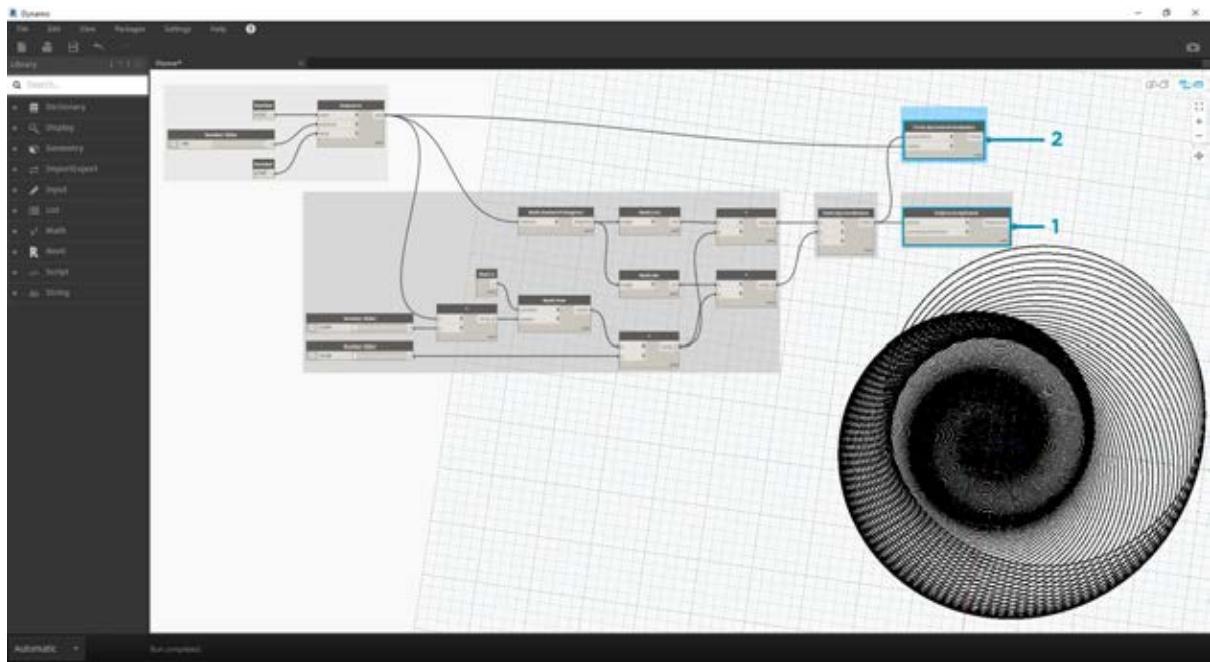
### Von der Spirale zum Nautilus



1. Beginnen Sie mit demselben Schritt wie in der vorigen Übung, d. h., indem Sie mithilfe des **Point.ByCoordinates**-Blocks ein spiralförmiges Array aus Punkten erstellen.



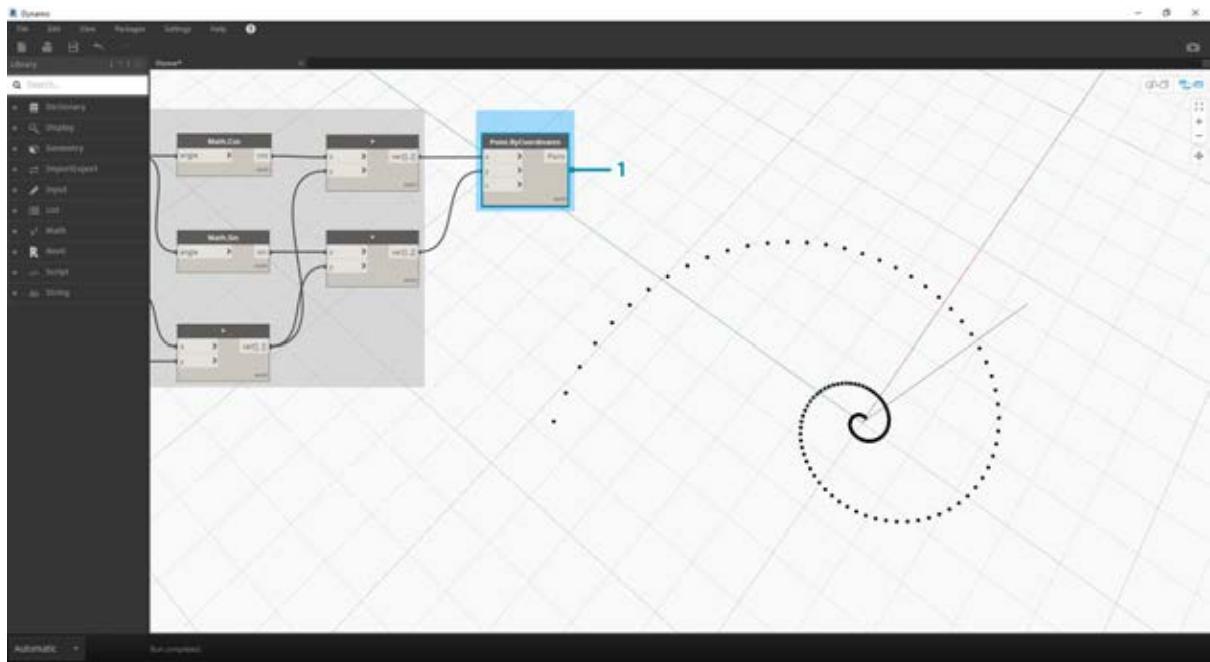
1. **Polycurve.ByPoints:** Auch dieser Block wurde in der vorigen Übung verwendet. Er dient hier als Referenz.
2. **Circle.ByCenterPointRadius:** Verwenden Sie hier einen Circle-Block mit denselben Eingaben wie im vorigen Schritt. Als Radius ist der Wert 1.0 vorgegeben, d. h., Sie sehen sofort die ausgegebenen Kreise. Die zunehmende Entfernung der Punkte vom Ursprung ist sofort ersichtlich.



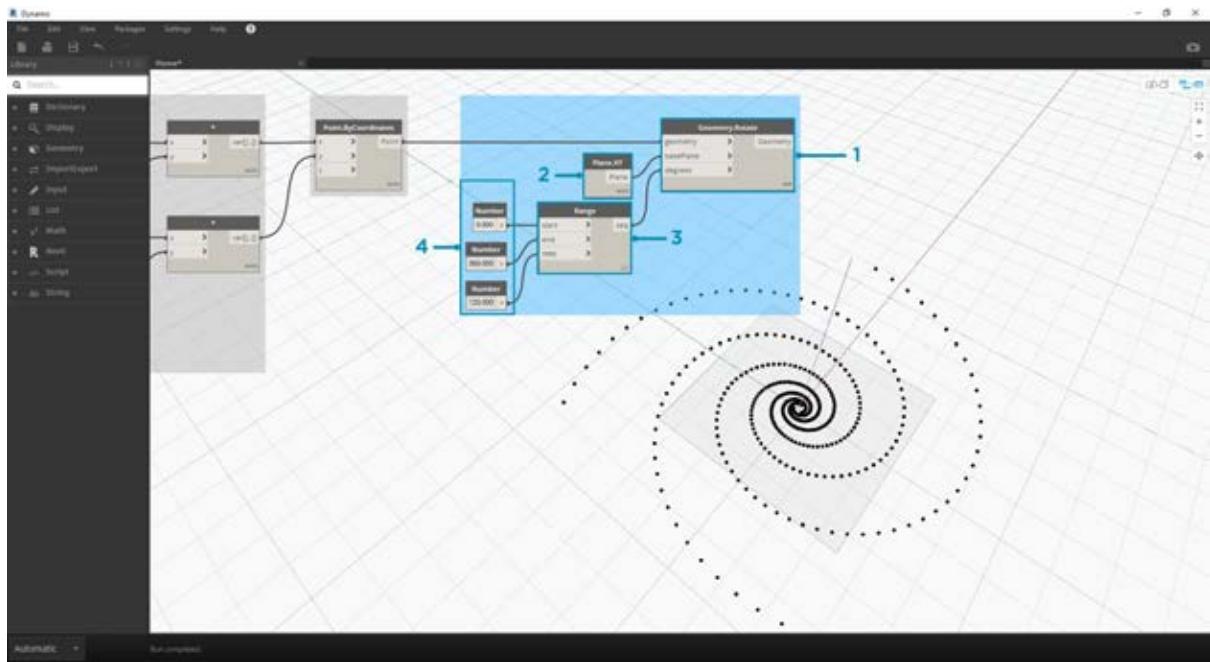
1. **Circle.ByCenterPointRadius:** Um das Array aus Kreisen dynamischer zu gestalten, verbinden Sie die ursprüngliche Zahlenfolge (die Folge der "t"-Werte) mit dem Radiuswert.
2. **Number Sequence:** Dies ist das Original-Array für "t". Die Verbindung mit dem Radiuswert bewirkt, dass die Mittelpunkte der Kreise sich nach wie vor vom Ursprung entfernen, wobei jedoch auch ihr Radius zunimmt. Sie erhalten eine recht originelle Fibonacci-Grafik. Versuchen Sie, dies in 3D darzustellen!

### Vom Nautilus zur Sonnenblume (Phyllotaxis)

Nachdem Sie eine Nautilusschale aus Kreisen erstellt haben, betrachten Sie jetzt parametrische Raster. Durch einfaches Drehen der Fibonacci-Spirale erstellen Sie ein Fibonacci-Raster. Das Ergebnis ist anhand des [Wachstums vom Sonnenblumensamen](#) modelliert.

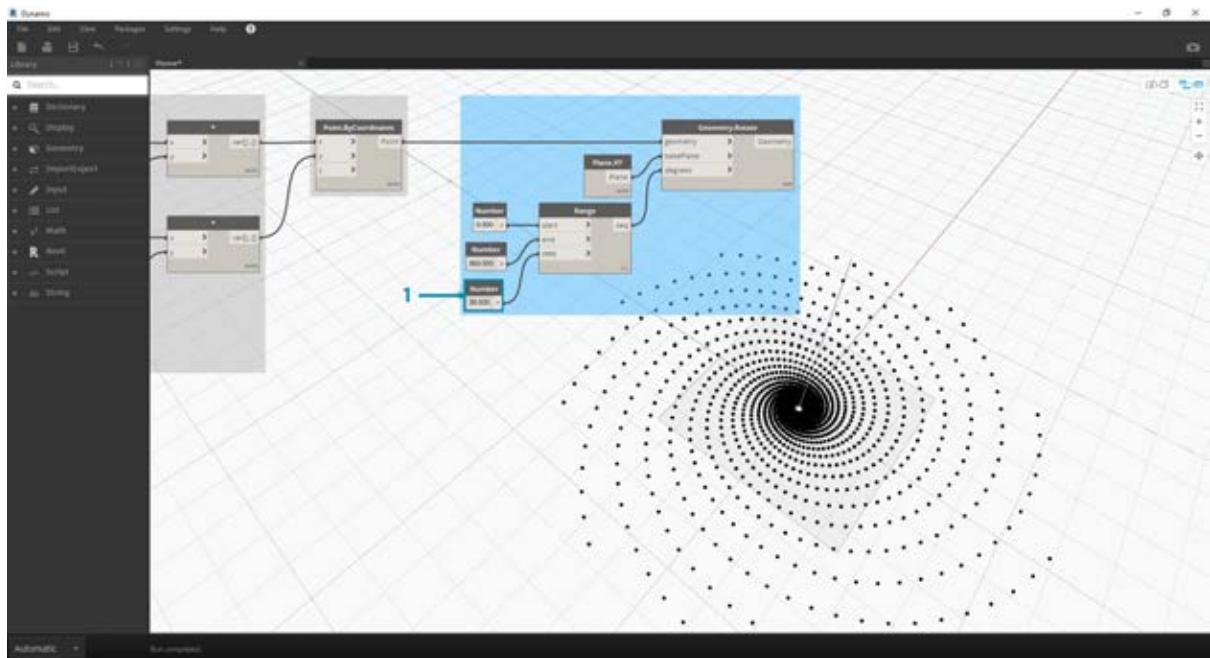


1. Beginnen Sie auch hier mit demselben Schritt wie in der vorigen Übung, d. h., indem Sie mithilfe des **Point.ByCoordinates**-Blocks ein spiralförmiges Array aus Punkten erstellen.

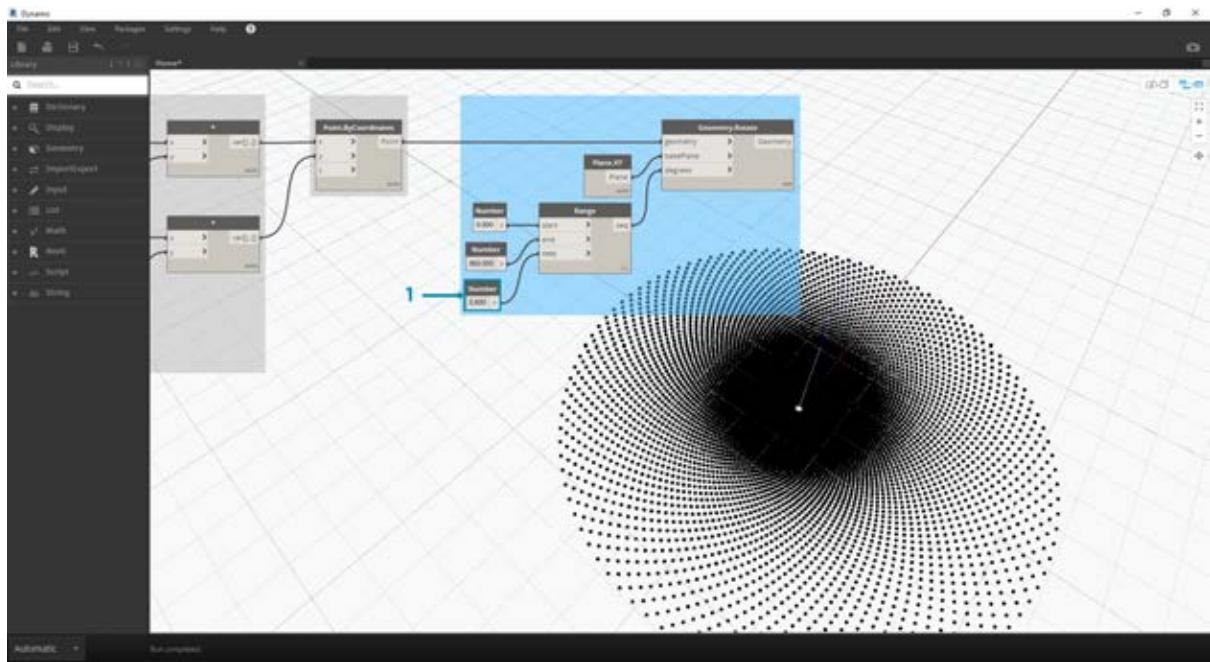


1. **Geometry.Rotate:** Es stehen mehrere Optionen für Geometry.Rotate zur Verfügung. Achten Sie darauf, den Block mit den Eingaben *geometry*, *basePlane* und *degrees* zu wählen. Verbinden Sie **Point.ByCoordinates** mit der *geometry*-Eingabe.
2. **Plane.XY:** Verbinden Sie dies mit der *basePlane*-Eingabe. Das Zentrum der Drehung ist der Ursprung, d. h. derselbe Punkt wie die Basis der Spirale.
3. **Range:** Sie benötigen mehrere Drehungen für die *degree*-Eingabe. Dies erreichen Sie schnell mit der Komponente für den Zahlenbereich. Verbinden Sie diese mit der *degrees*-Eingabe.
4. **Number:** Fügen Sie im Ansichtsbereich drei Zahlenblöcke übereinander ein, um den Zahlenbereich zu definieren. Weisen Sie diesen von oben nach unten die Werte 0.0, 360.0, und 120.0 zu. Diese Werte steuern die Drehung der Spirale. Beachten Sie die Ergebnisse der Ausgabe aus dem **Range**-Block, nachdem Sie die drei Zahlenblöcke mit ihm verbunden haben.

Die Ausgabe nimmt eine gewisse Ähnlichkeit mit einem Wirbel an. Passen Sie jetzt einige der für **Range** verwendeten Parameter an und beobachten Sie, wie sich die Ergebnisse verändern:



1. Ändern Sie die Schrittgröße für den **Range**-Block von **120.0** in **36.0**. Damit erhalten Sie mehr Drehungen und daher ein dichteres Raster.



1. Ändern Sie die Schrittgröße für den **Range**-Block von 36.0 in 3.6. Dadurch erhalten Sie ein wesentlich dichteres Raster und die Richtung der Spiralen ist nicht mehr erkennbar. Damit haben Sie ein Sonnenblumenmuster erstellt.

# Logik

## Logik

**Logik**, genauer **Bedingungslogik** ermöglicht die Angabe einer Aktion oder einer Gruppe von Aktionen unter Verwendung einer Prüfung. Die Auswertung der Prüfung ergibt einen booleschen Wert für **True** oder **False**, der zur Steuerung des Programmablaufs verwendet werden kann.

### Boolesche Werte

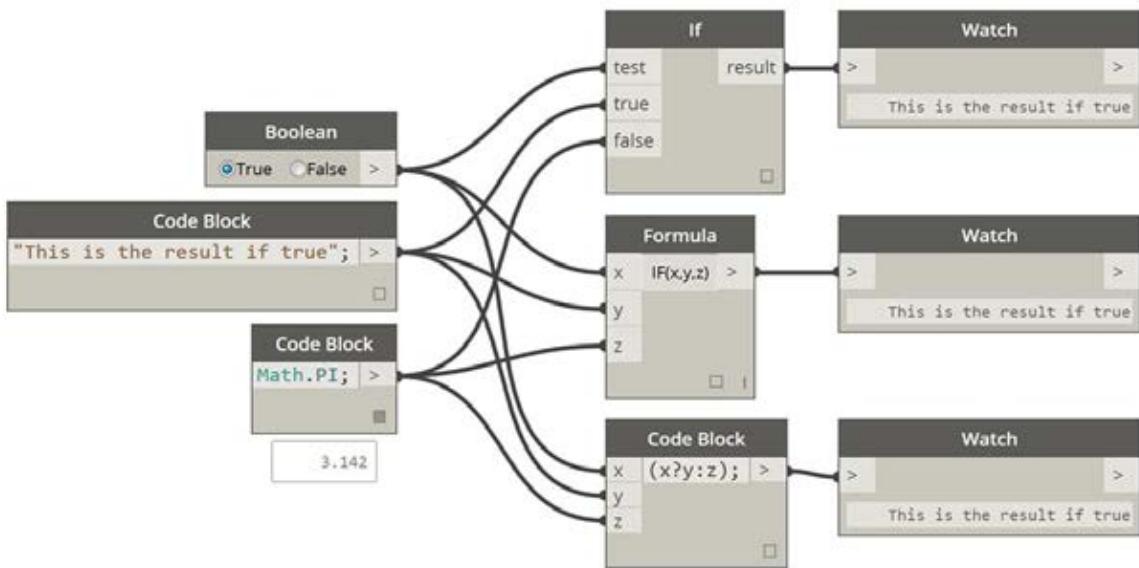
Numerische Variablen können für eine Vielzahl unterschiedlicher Zahlen stehen. Boolesche Variablen hingegen können nur zwei Werte annehmen, die als True oder False, Ja oder Nein, 1 oder 0 wiedergegeben werden. Booleschen Operationen werden wegen ihrer begrenzten Möglichkeiten nur selten zum Durchführen von Berechnungen verwendet.

### Bedingungsanweisungen

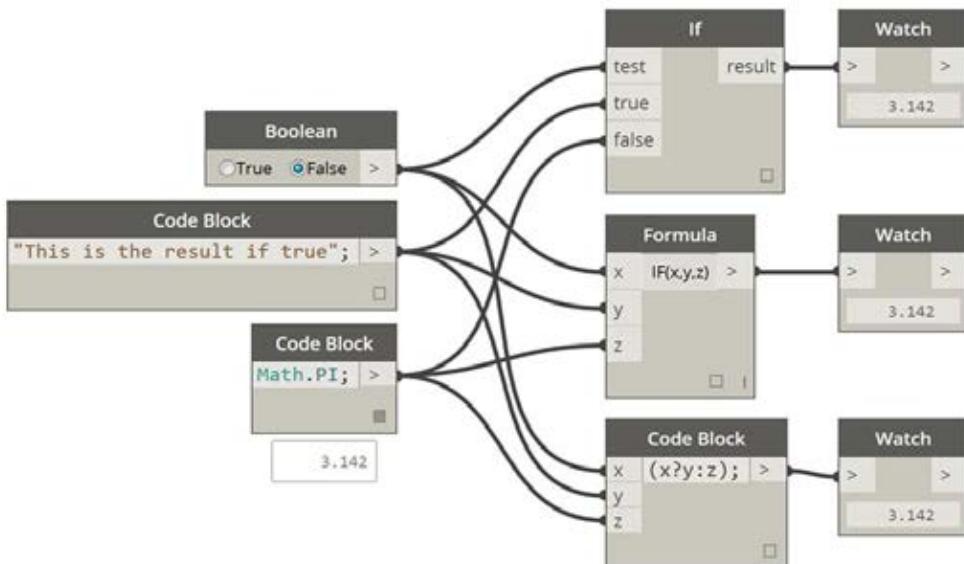
Die "If"-Anweisung ist ein wichtiges Konzept in der Programmierung: "Wenn *dies* zutrifft (True), dann geschieht *das*, andernfalls geschieht *etwas anderes*. Die aus der Anweisung resultierende Aktion wird durch einen booleschen Wert gesteuert. In Dynamo stehen mehrere Möglichkeiten zum Definieren von If-Anweisungen zur Verfügung:

Symbol	Name	Syntax	Eingaben	Ausgaben
	Wenn	If	test, true, false	result
	Formel	if (x, y,z)	x, y, z	result
	Codeblock	(x?y:z)	x, y, z	result

Die folgenden kurzen Beispiele zeigen die Funktionsweise dieser drei Blöcke in der If-Bedingungsanweisung.



In dieser Abbildung wurde im *Boolean* die Option *True*, eingestellt, d. h., das Ergebnis ist die Zeichenfolge: "this is the result if true". Die drei möglichen Blöcke, mit deren Hilfe die *If*-Anweisung erstellt werden kann, funktionieren hier auf dieselbe Weise.

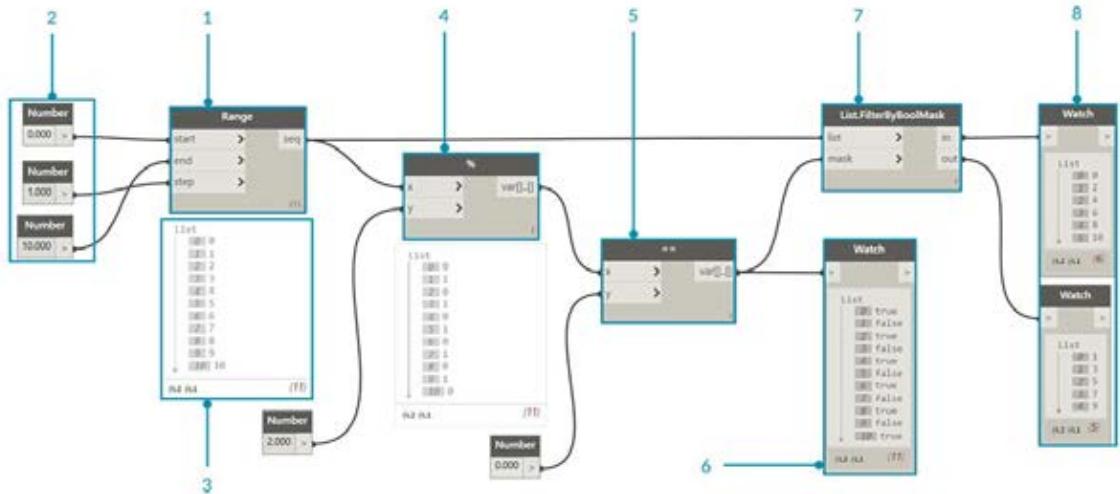


Auch in diesem Fall funktionieren die Blöcke auf dieselbe Weise. Wenn Sie den *Boolean*-Wert in *False* ändern, wird als Ergebnis die Zahl *Pi* ausgegeben, wie in der ursprünglichen *If*-Anweisung festgelegt.

## Filtern einer Liste

Laden Sie die Beispieldatei für diese Übungslektion herunter (durch Rechtsklicken und Wahl von Save Link As...): [Building Blocks of Programs - Logic.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

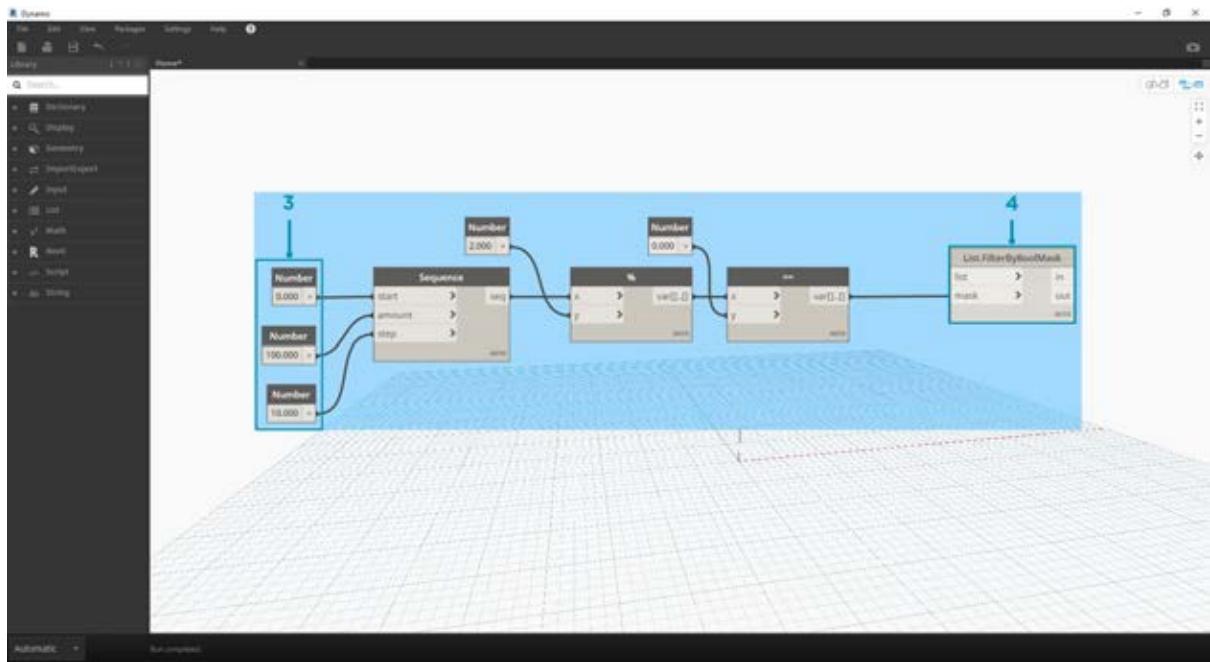
In diesem Beispiel teilen Sie eine Liste von Zahlen in eine Liste mit geraden und eine Liste ungeraden Zahlen auf.



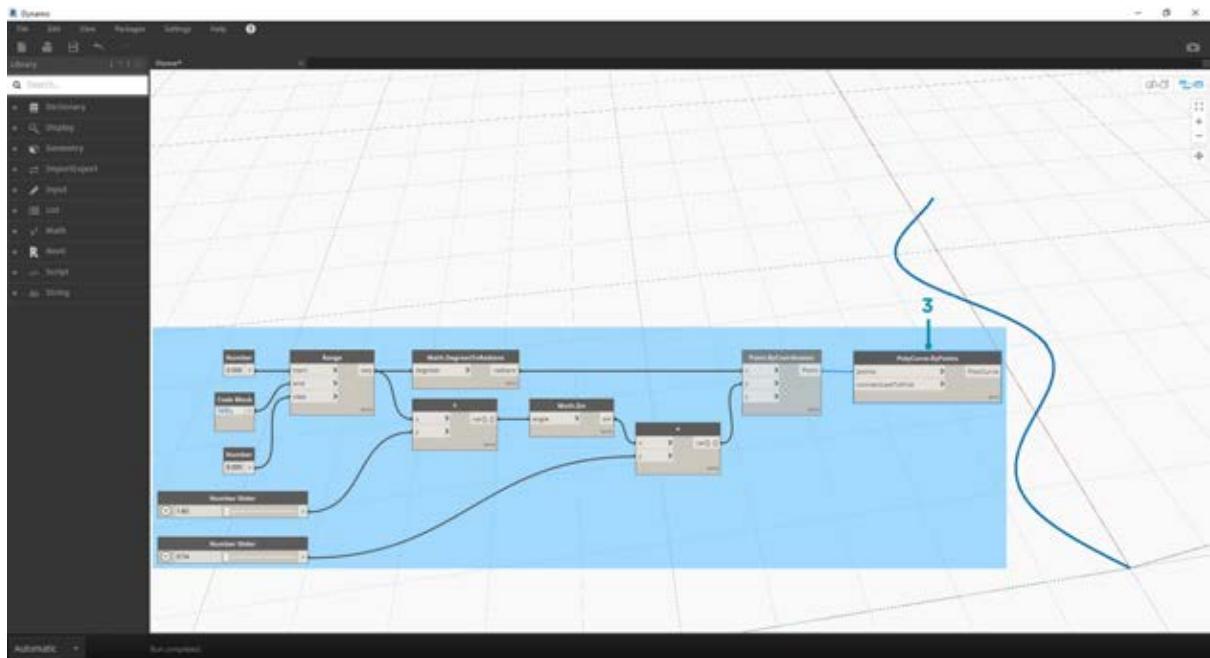
- Range:** Definieren Sie im Ansichtsbereich einen Zahlenbereich.
- Number-Blöcke:** Fügen Sie im Ansichtsbereich drei Number-Blöcke ein. Legen Sie in diesen Blöcken die folgenden Werte fest: `0.0` für `start`, `10.0` für `end` und `1.0` für `step`.
- Ausgabe:** Die Ausgabe zeigt eine Liste mit 11 Zahlen von 0-10.
- Modulus (%):** Verbinden Sie `Range` mit `x` und `2.0` mit `y`. Mit dieser Funktion wird für jede Zahl aus der Liste der Rest bei Division durch 2 berechnet. Die Ausgabe für diese Liste ist eine Liste, die abwechselnd die Werte 0 und 1 enthält.
- Gleichheitsprüfung (==):** Fügen Sie im Ansichtsbereich einen Block für die Gleichheitsprüfung hinzu. Verbinden Sie die Ausgabe des `Modulus`-Blocks mit der `x`-Eingabe und `0.0` mit der `y`-Eingabe.
- Watch:** Die Gleichheitsprüfung gibt eine Liste aus, die abwechselnd die Werte `true` und `false` enthält. Mithilfe dieser Werte werden die Einträge aus der Zahlenliste eingeordnet. Dabei steht `0` (bzw. `true`) für gerade und `(1` bzw. `false`) für ungerade Zahlen.
- List.FilterByBoolMask:** Dieser Block filtert die Werte anhand der eingegebenen Booleschen Operation in zwei Listen. Verbinden Sie den ursprünglichen `number range` mit der `list`-Eingabe und die Ausgabe der `Gleichheitsprüfung` mit der `mask`-Eingabe. Die `in`-Ausgabe enthält die `true`-Werte, die `out`-Ausgabe die `false`-Werte.
- Watch:** Als Ergebnis erhalten Sie eine Liste mit geraden und eine Liste mit ungeraden Zahlen. Damit haben Sie mithilfe logischer Operatoren Listen anhand eines Musters aufgeteilt.

## Aus Logik wird Geometrie

In diesem Schritt wenden Sie die in der ersten Übung erstellte Logik auf einen Modellierungsvorgang an.



1. Beginnen Sie mit den Blöcken aus der letzten Übung. Dabei bestehen jedoch einige Unterschiede:
2. Das Format wurde geändert.
3. Die Eingabewerte wurden geändert.
4. Die in-Listeneingabe für *List.FilterByBoolMask* wurde entfernt. Diese Blöcke werden momentan nicht benötigt, kommen aber weiter unten in dieser Übung zum Einsatz.



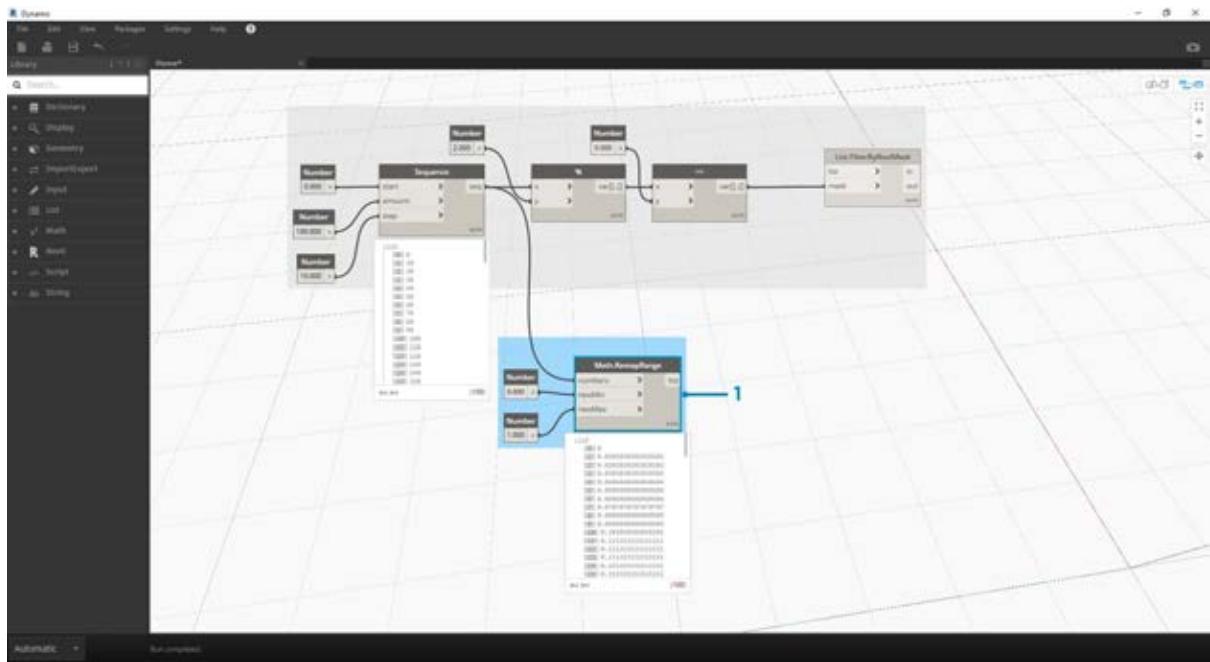
Beginnen Sie, indem Sie die Blöcke wie in der Abbildung oben gezeigt miteinander verbinden. Diese Gruppe von Blöcken stellt eine parametrische Gleichung zum Definieren einer Sinuskurve dar. Einige Hinweise:

1. Im **ersten Schieberegler** müssen als Mindestwert 1, als Höchstwert 4 und als Schritt 0.01 angegeben werden.
2. Im **zweiten Schieberegler** müssen als Mindestwert 0, als Höchstwert 1 und als Schritt 0.01 angegeben werden.
3. **PolyCurve.ByPoints:** Indem Sie das oben gezeigte Blockdiagramm kopieren, erhalten im Ansichtsfenster der Dynamo-Vorschau eine Sinuskurve.

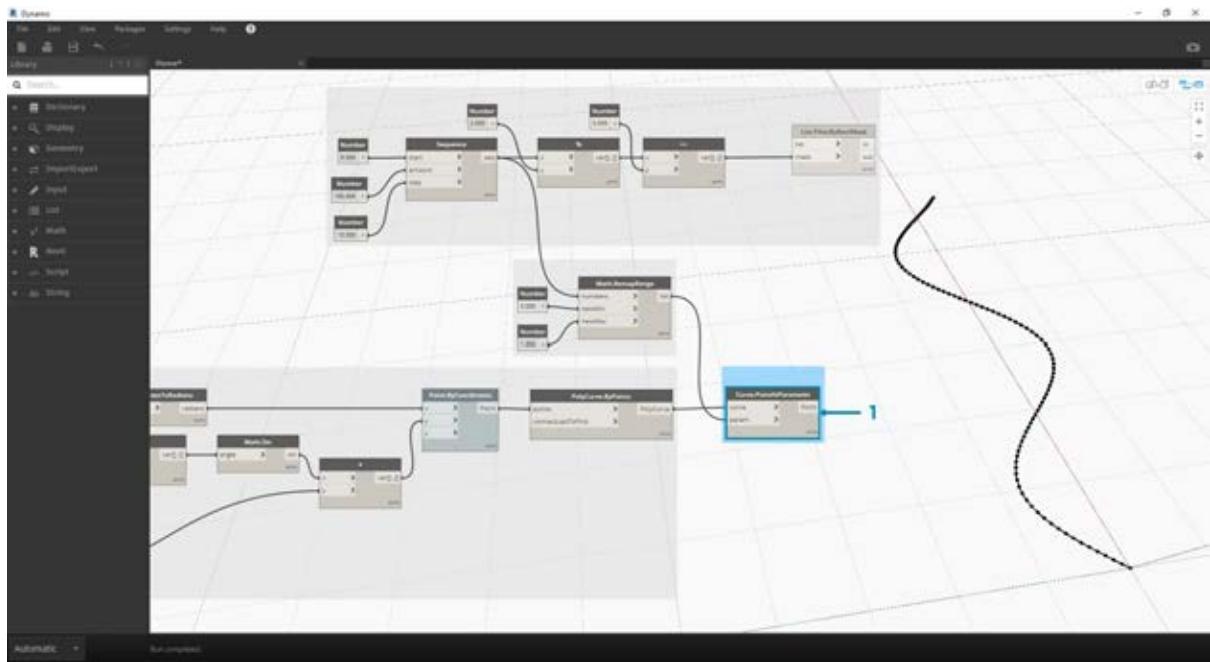
Für die Eingaben gilt hier folgende Regel: Verwenden Sie Zahlenblöcke für statische und Schieberegler für veränderliche Eigenschaften. Der anfangs definierte ursprüngliche Zahlenbereich soll erhalten bleiben. Für die Sinuskurve, die hier erstellt werden soll, wird jedoch mehr Flexibilität benötigt. Mithilfe dieser Schieberegler können Sie die Frequenz und Amplitude der Kurve ändern.



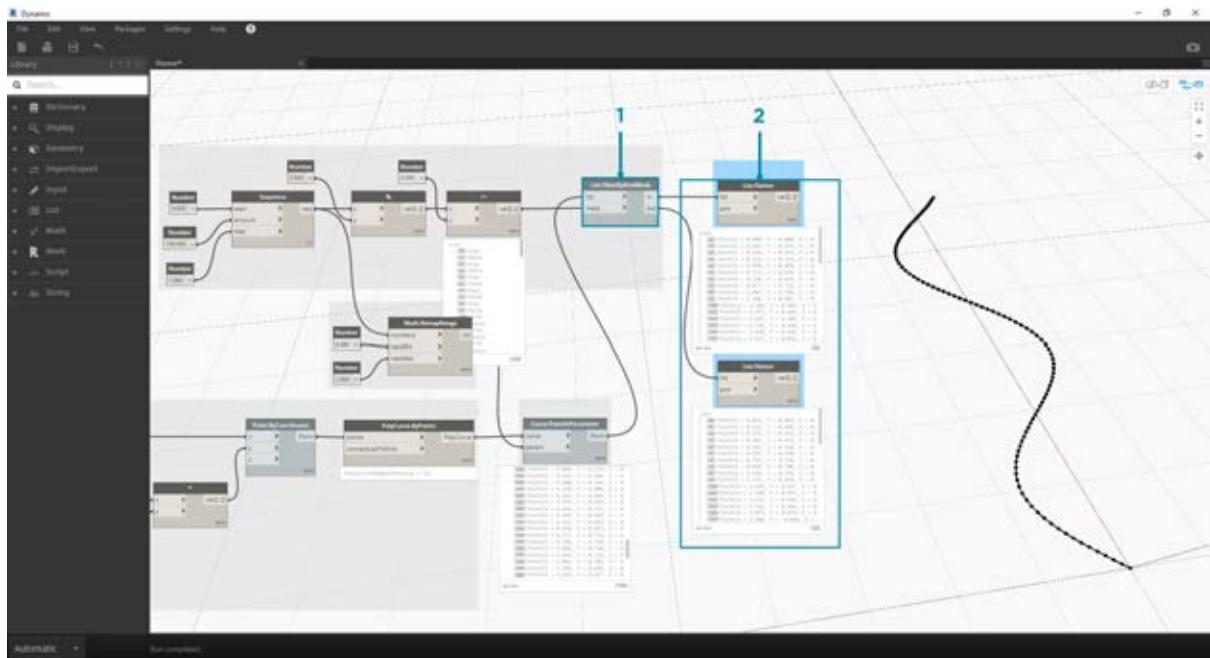
Die Schritte dieser Definition werden hier nicht nacheinander beschrieben. Hier wird zunächst das Endergebnis gezeigt, um eine Vorstellung der fertigen Geometrie zu vermitteln. Die ersten beiden Schritte wurden separat durchgeführt und müssen jetzt zusammengeführt werden. Die Position der reißverschlussähnlichen Bauteile soll durch die zugrunde liegende Sinuskurve gesteuert werden, wobei mithilfe der True/False-Logik abwechselnd große und kleine Quader eingefügt werden.



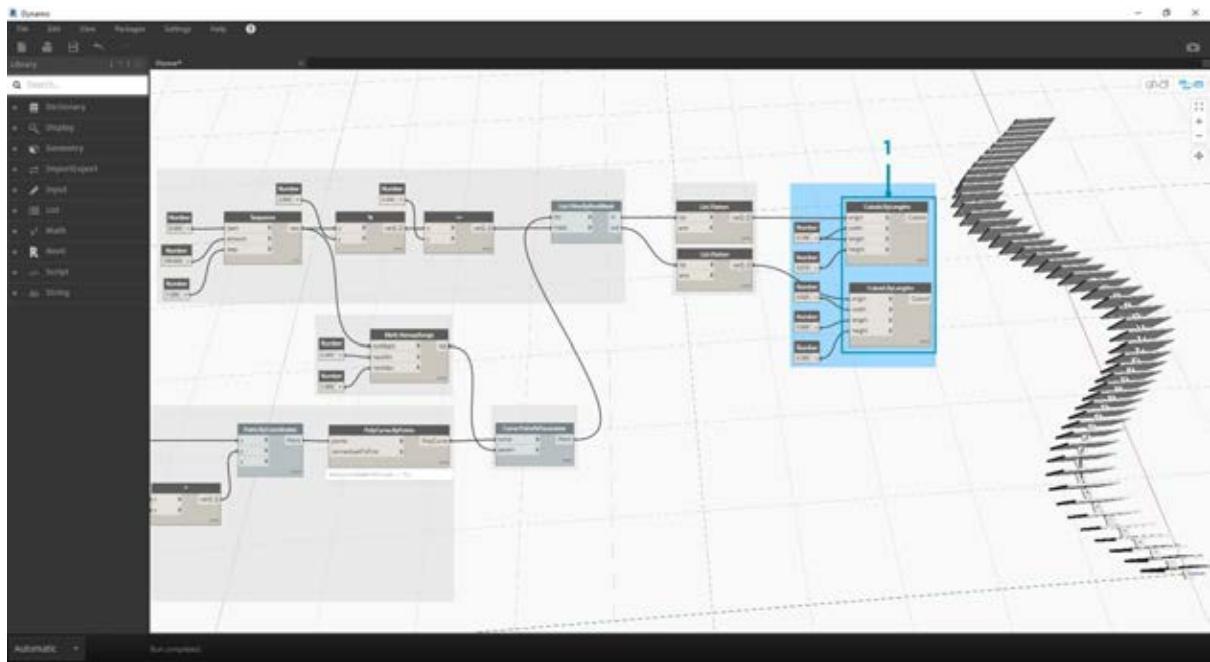
1. **Math.RemapRange:** Erstellen Sie aus der in Schritt 01 erstellten Zahlenfolge eine neue Zahlenfolge, indem Sie den Bereich neu zuordnen. In Schritt 01 wurden Zahlen von 0 – 100 festgelegt. Diese Zahlen liegen zwischen 0 und 1, wie mithilfe der Eingaben *newMin* und *newMax* festgelegt.



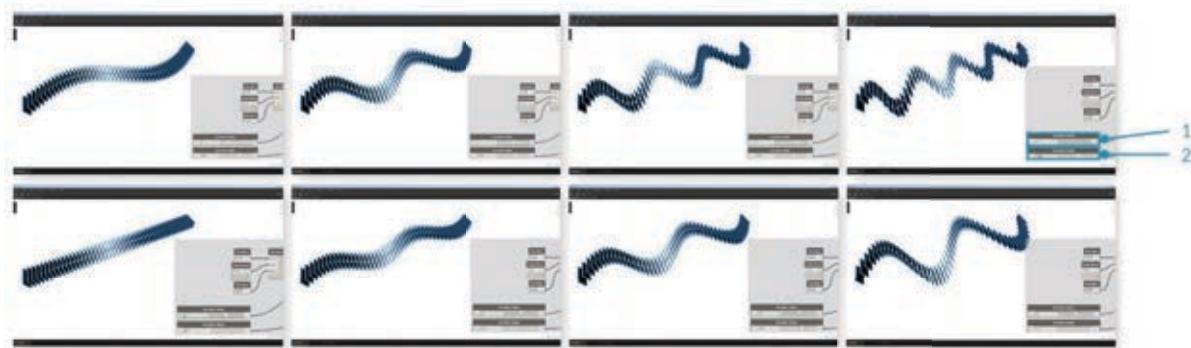
1. **Curve.PointAtParameter:** Verbinden Sie *Polycurve.ByPoints* (aus Schritt 2) mit *curve* und *Math.RemapRange* mit *param*. Mithilfe dieses Schritts erstellen Sie Punkte entlang der Kurve. Die Zahlen mussten dem Bereich 0 bis 1 neu zugeordnet werden, da als Eingabe für *param* Werte in diesem Bereich verlangt werden. Der Wert 0 steht für den Startpunkt, der Wert 1 für die Endpunkte. Die Auswertung aller dazwischen liegenden Zahlen ergibt Werte im Bereich [0,1].



1. **List.FilterByBoolMask** - Verbinden Sie *Curve.PointAtParameter* aus dem vorigen Schritt mit der *list*-Eingabe.
2. **Watch:** Die Watch-Blöcke für *in* und *out* zeigen, dass zwei Listen für gerade bzw. ungerade Indizes erstellt wurden. Diese Punkte sind auf dieselbe Weise entlang der Kurve angeordnet, wie im folgenden Schritt gezeigt.



1. **Cuboid.ByLengths:** Stellen Sie die in der Abbildung oben gezeigten Verbindungen wieder her, um eine reißverschlussähnliche Struktur entlang der Kurve zu erhalten. Sie erstellen hier einfache Quader, deren Größe jeweils durch ihren auf der Kurve liegenden Mittelpunkt definiert wird. Die Logik der Aufteilung in gerade und ungerade Werte wird damit im Modell deutlich.



1. **Number Slider:** Wenn Sie zum Anfang dieser Definition zurückkehren, können Sie mit den Werten im Schieberegler experimentieren und beobachten, wie sich der "Reißverschluss" verändert. Die obere Reihe der Abbildungen zeigt den Wertebereich für den oberen Zahlen-Schieberegler. Er steuert die Frequenz der Welle.
2. **Number Slider:** Die untere Reihe der Abbildungen zeigt den Wertebereich für den unteren Zahlen-Schieberegler. Er steuert die Amplitude der Welle.

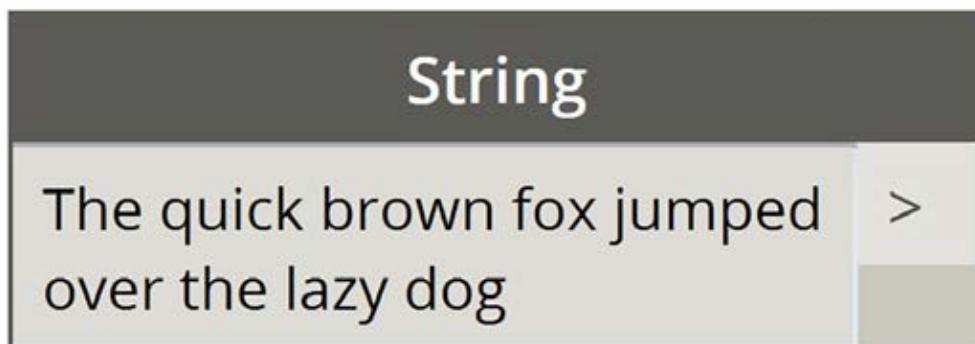
# Zeichenfolgen

## Zeichenfolgen

Eine **Zeichenfolge** ist in der Fachsprache eine Folge von Zeichen, die für eine Literalkonstante oder verschiedene Typen von Variablen steht. Im Zusammenhang mit Programmierung ist "Zeichenfolge" eine Bezeichnung für Text. Sie haben bereits Zahlen (Ganzzahlen und Dezimalzahlen) zur Steuerung von Parametern verwendet. Dasselbe ist auch mit Text möglich.

### Erstellen von Zeichenfolgen

Zeichenfolgen können auf vielfältige Weise eingesetzt werden. Dazu gehören das Definieren benutzerdefinierter Parameter, Beschriftungen für Dokumentation sowie die Analyse mithilfe textbasierter Datensätze. Der String-Block befindet sich in der Kategorie Core > Input.



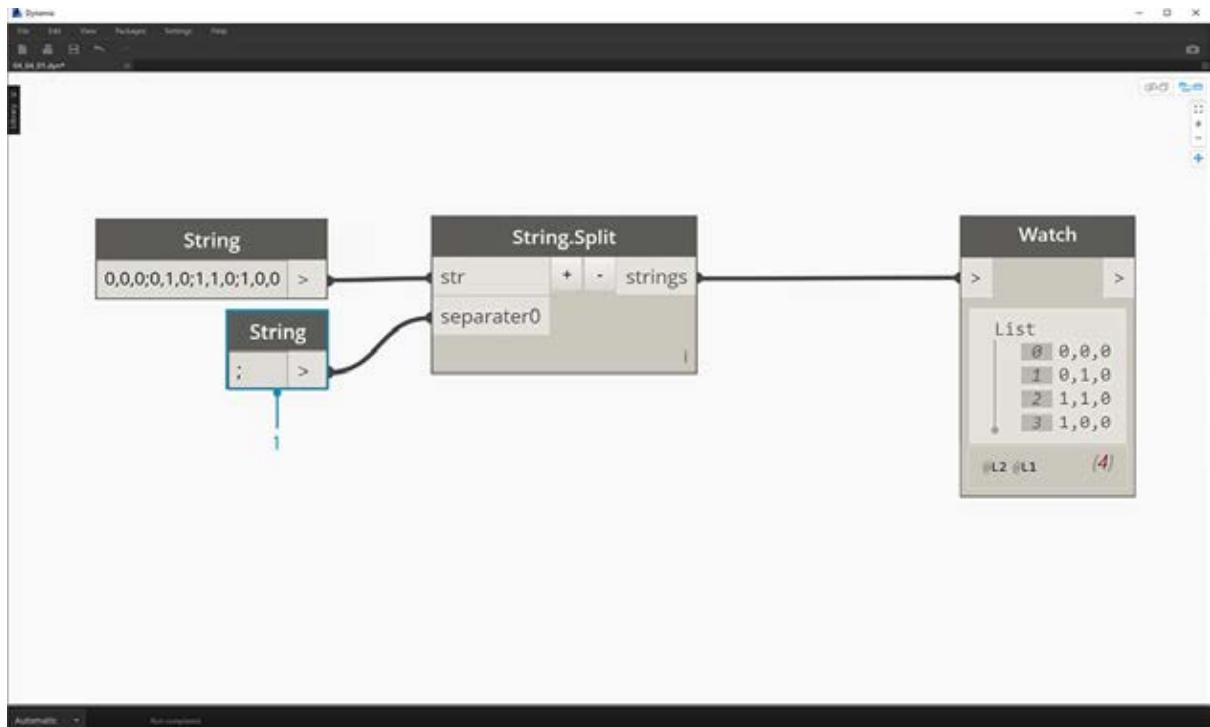
Die oben gezeigten Blöcke sind Zeichenfolgen. Zeichenfolgen können Zahlen, Buchstaben oder ganze Textabschnitte sein.

### Abfragen von Zeichenfolgen

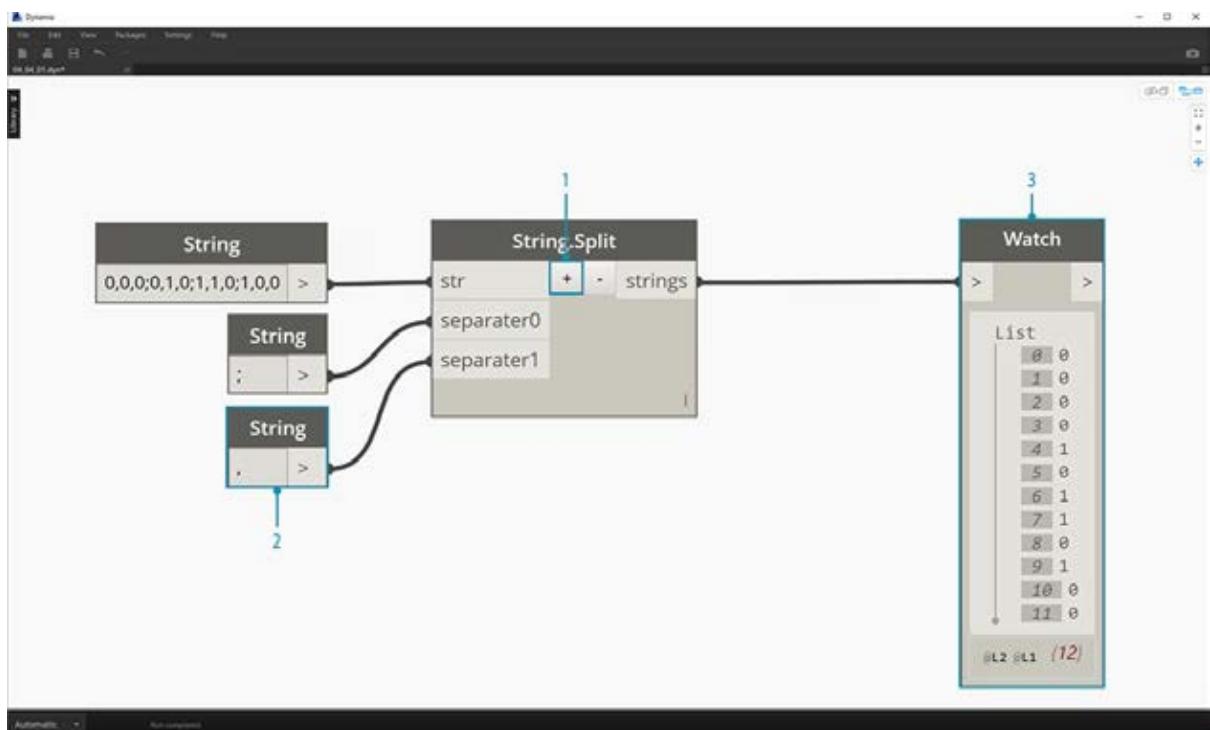
Laden Sie die Beispieldatei für diese Übungslektion herunter (durch Rechtsklicken und Wahl von "Save Link As..."):  
[Building Blocks of Programs - Strings.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

Sie können große Datenmengen rasch analysieren, indem Sie Zeichenfolgen abfragen. In diesem Abschnitt werden einige grundlegende Vorgänge behandelt, die Arbeitsabläufe beschleunigen und die Interoperabilität von Software verbessern können.

Die folgende Abbildung zeigt eine Folge von Daten aus einer externen Kalkulationstabelle. Diese Zeichenfolge steht für die Scheitelpunkt eines Rechtecks in der xy-Ebene. In dieser kurzen Übung wird die Zeichenfolge geteilt:

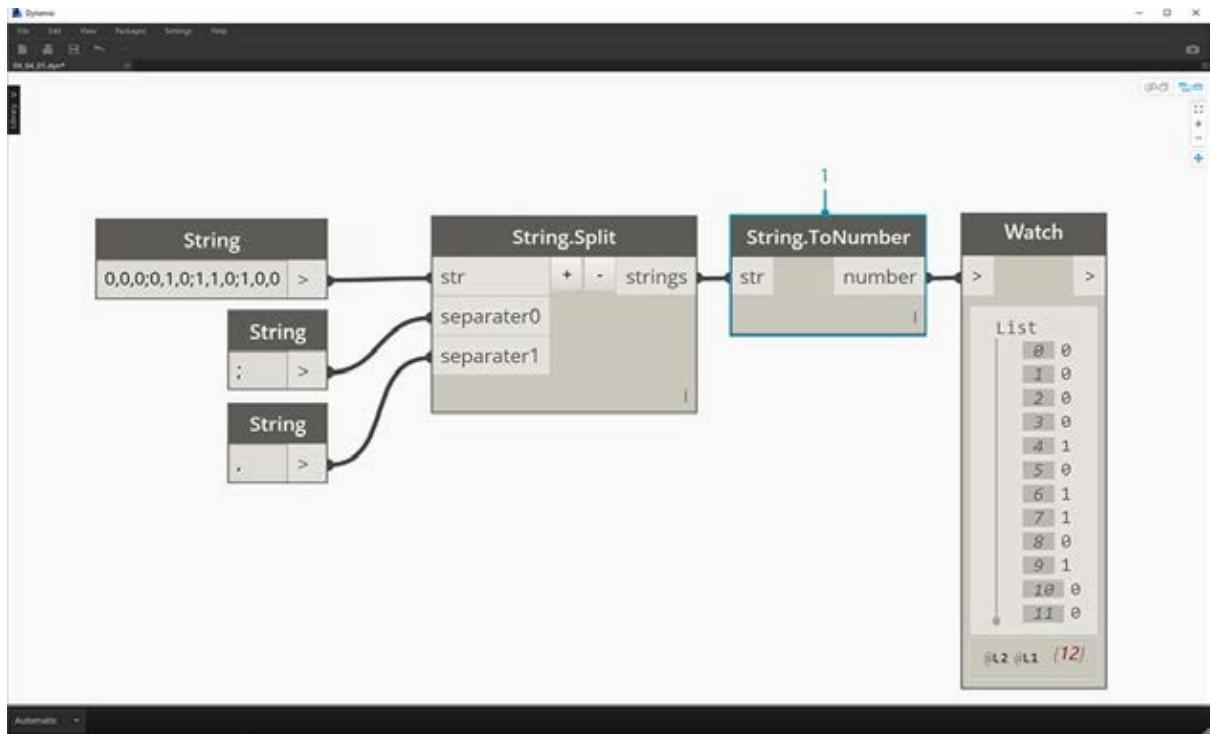


1. Das Trennzeichen ";" trennt die einzelnen Scheitelpunkte des Rechtecks voneinander. Dadurch erhalten Sie eine Liste mit vier Einträgen für die einzelnen Scheitelpunkte.

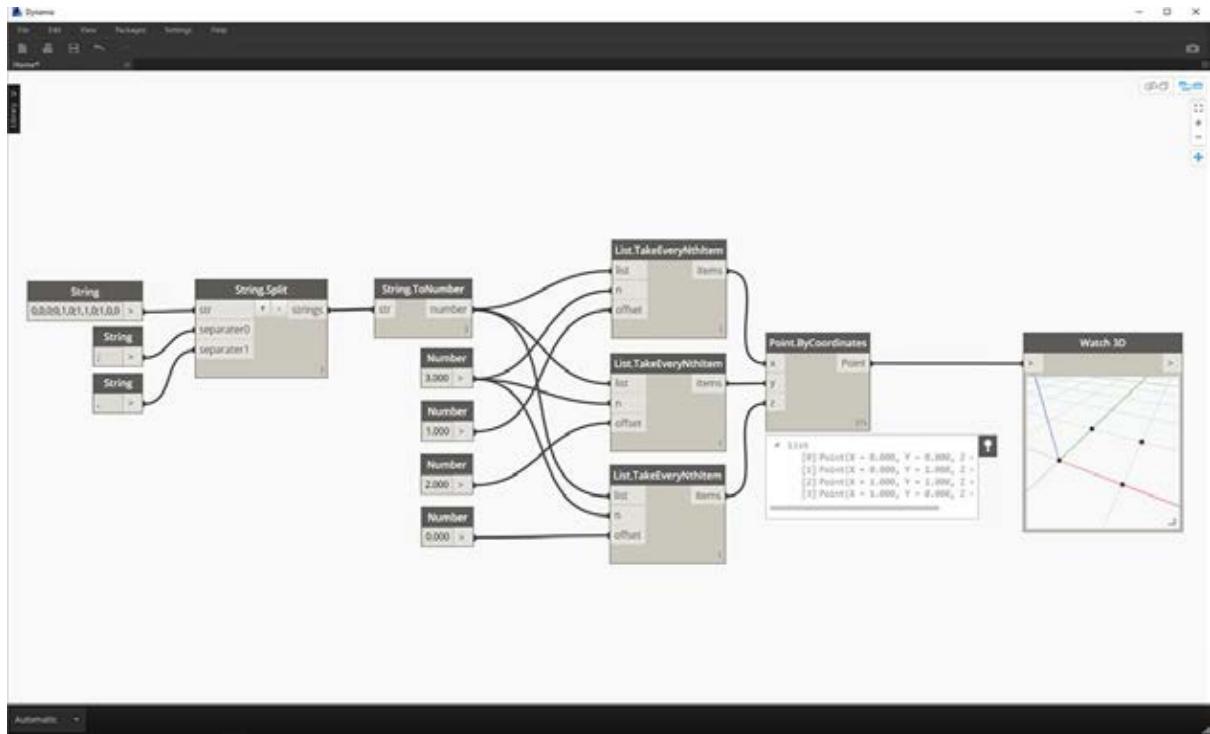


1. Durch Klicken auf "+" in der Mitte des Blocks erstellen Sie ein neues Trennzeichen.
2. Fügen Sie einen String-Block mit der Angabe "," in den Ansichtsbereich ein und verbinden Sie ihn mit der neuen separator-Eingabe.
3. Als Ergebnis erhalten Sie jetzt eine Liste mit zehn Einträgen. In diesem Block wird die Liste zuerst entsprechend *separator0* und dann entsprechend *separator1* geteilt.

Die Einträge in der oben gezeigten Liste sehen zwar wie Zahlen aus, werden jedoch in Dynamo nach wie vor als einzelne Zeichen betrachtet. Damit Sie Punkte erstellen können, müssen Sie ihren Datentyp aus Zeichenfolgen in Zahlen konvertieren. Dazu verwenden Sie den Block *String.ToNumber*.



1. Dieser Block ist einfach zu verwenden. Verbinden Sie das Ergebnis von String.Split mit der Eingabe. Die Ausgabe sieht genau so aus wie zuvor, der Datentyp ist jetzt jedoch *number* anstelle von *string*.

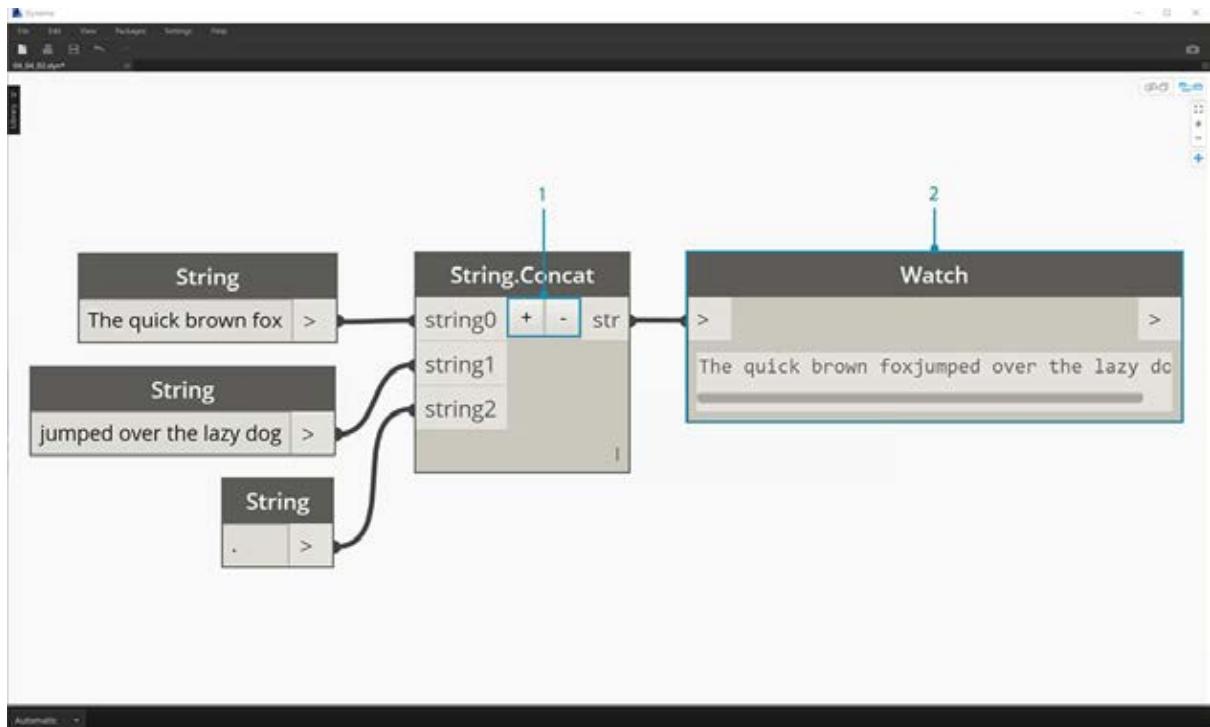


1. Mit einigen weiteren grundlegenden Operationen wird am Ursprungspunkt ein Rechteck gezeichnet, dem die Zeichenfolge aus der ursprünglichen Eingabe zugrunde liegt.

## Bearbeiten von Zeichenfolgen

Da Zeichenfolgen allgemeine Textobjekte sind, eignen sie sich für eine Vielzahl von Verwendungszwecken. In diesem Abschnitt werden einige der wichtigsten Aktionen in der Kategorie Core > String in Dynamo beschrieben:

Die folgende Methode verbindet zwei Zeichenfolgen in der angegebenen Reihenfolge. Dabei werden die einzelnen Literalzeichenfolgen in einer Liste zu einer einzigen Zeichenfolge zusammengeführt.

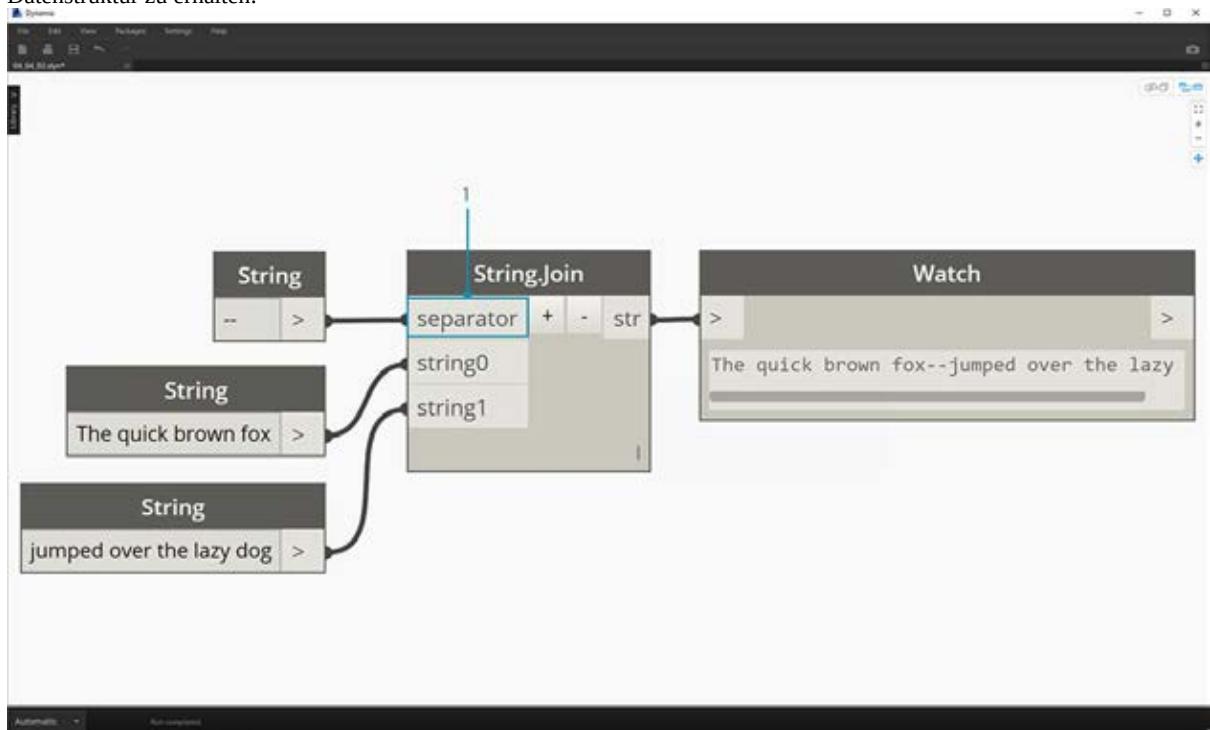


Die Abbildung oben zeigt die Verkettung dreier Zeichenfolgen.

1. Mithilfe der Schaltflächen + und - in der Mitte des Blocks können Sie der Verkettung weitere Zeichenfolgen hinzufügen oder sie daraus entfernen.
2. Die Ausgabe zeigt die aus der Verkettung resultierende Zeichenfolge einschließlich Leerzeichen und Satzzeichen.

Die Verbindung ist der Verkettung ähnlich, wobei das Ergebnis jedoch zusätzlich gegliedert wird.

Aus der Arbeit mit Excel sind Sie wahrscheinlich mit CSV-Dateien vertraut. Dies steht für kommagetrennte Werte. Sie könnten im Join-Block ein Komma (oder in diesem Fall zwei Bindestriche) als Trennzeichen verwenden, um eine ähnliche Datenstruktur zu erhalten:

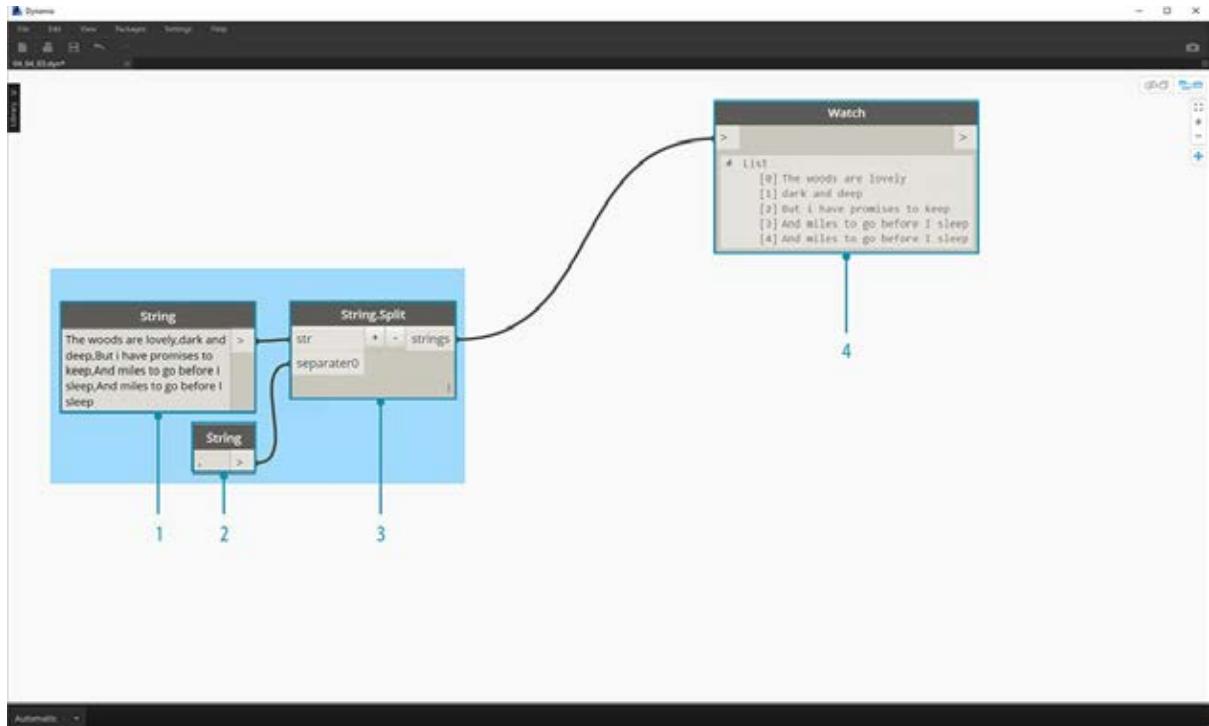


Die Abbildung oben zeigt die Verbindung zweier Zeichenfolgen:

1. Die separator-Eingabe ermöglicht es, eine Zeichenfolge zu erstellen, die als Trennzeichen zwischen den verbundenen Zeichenfolgen steht.

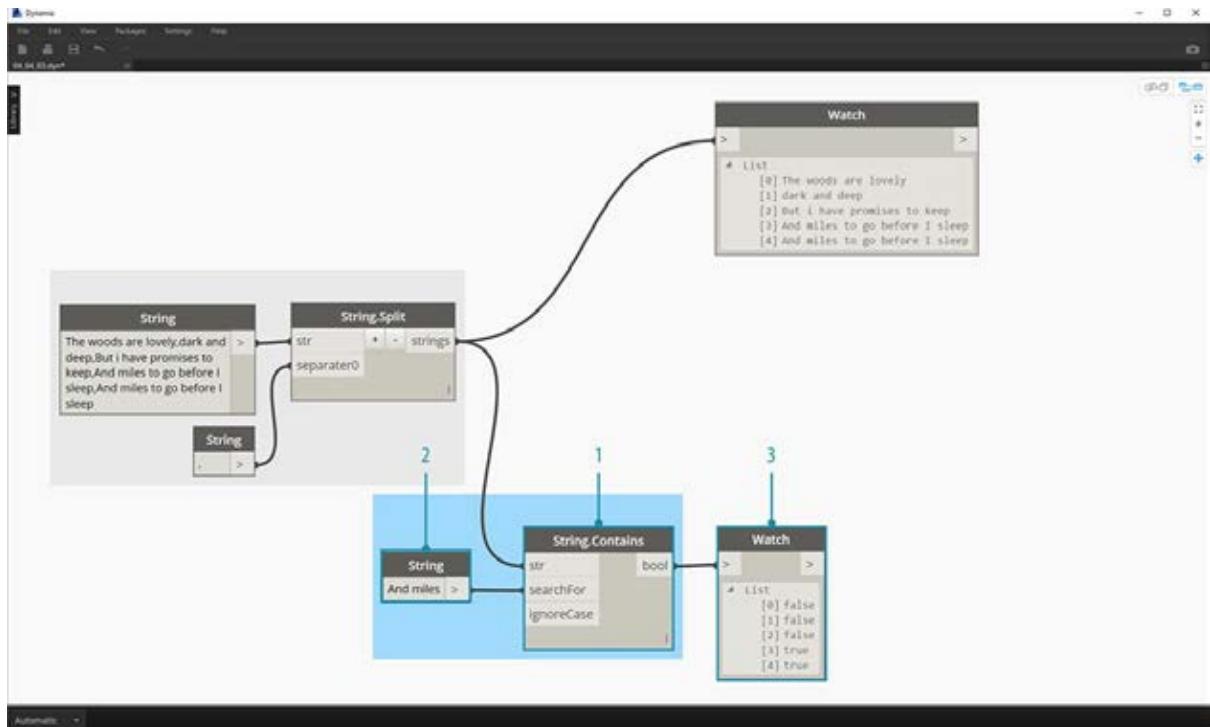
## Arbeiten mit Zeichenfolgen

In dieser Übungslektion zerlegen Sie die letzte Strophe von Robert Frossts Gedicht [Stopping By Woods on a Snowy Evening](#) mithilfe von Methoden zum Abfragen und Bearbeiten von Zeichenfolgen. Diese zugegebenermaßen nicht unbedingt praxisnahe Anwendung verdeutlicht die grundlegenden Aktionen für Zeichenfolgen durch Anwendung auf die Zeilen mit ihrem Rhythmus und Reim, wie sie im Gedicht erscheinen.



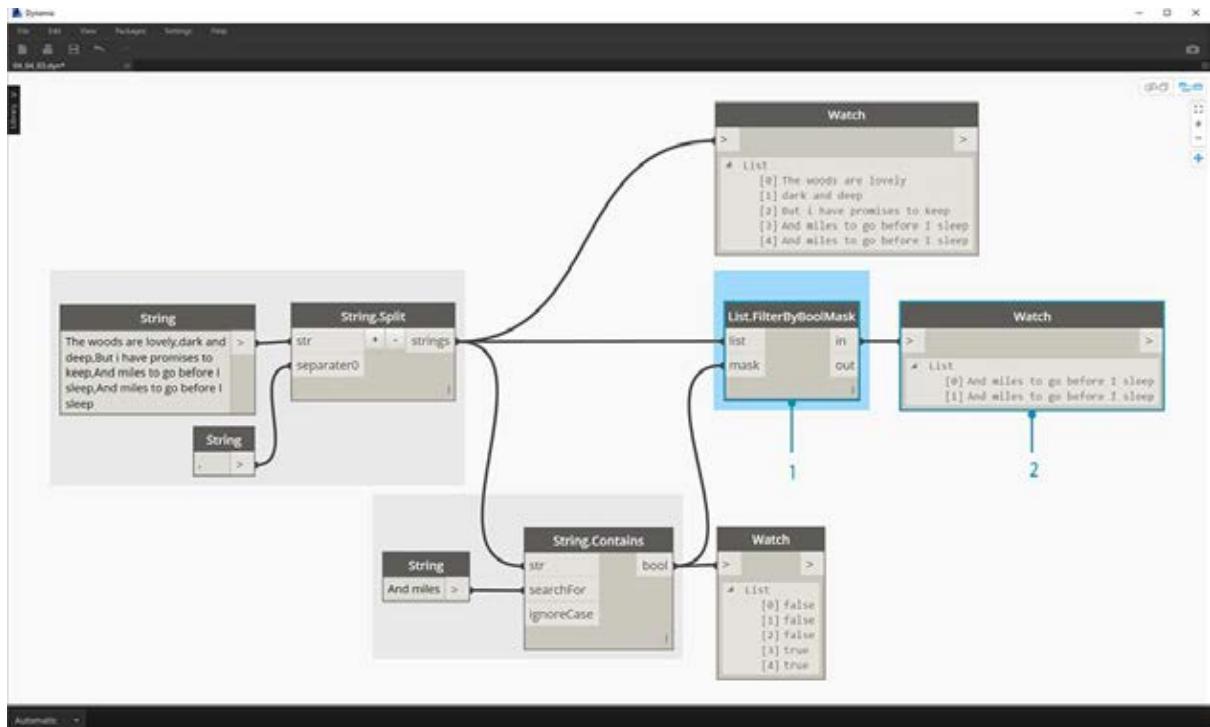
Als Erstes teilen Sie einfach die Zeichenfolge, die die Strophe bildet. Zunächst fällt auf, dass der Text durch Kommas gegliedert ist. Anhand dieses Formats definieren Sie jede Zeile als eigenen Eintrag.

1. Fügen Sie die Ausgangszeichenfolge in einen String-Block ein.
2. Legen Sie mithilfe eines zweiten String-Blocks das Trennzeichen fest. Verwenden Sie in diesem Fall ein Komma.
3. Fügen Sie im Ansichtsbereich einen String.Split-Block hinzu und verbinden Sie ihn mit den beiden String-Blöcken.
4. Die Ausgabe zeigt die einzelnen Zeilen als separate Einträge.

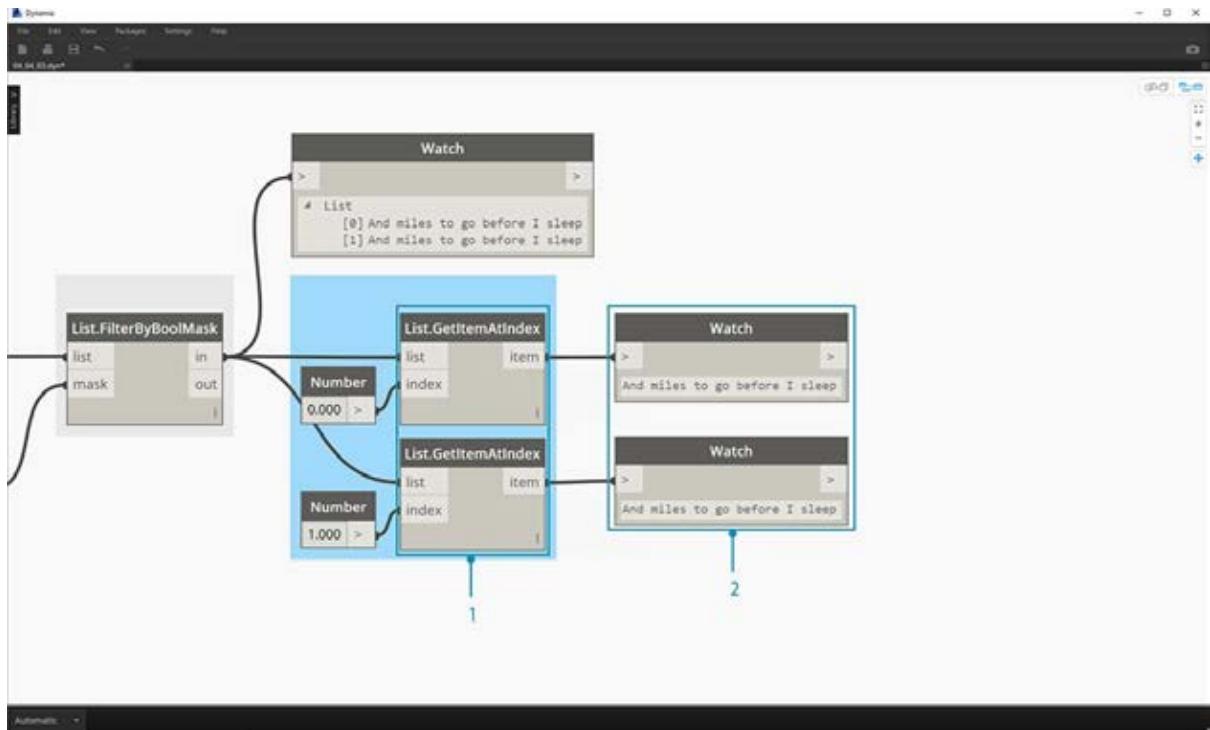


Betrachten Sie jetzt den besten Teil des Gedichts – die letzten beiden Zeilen. Die ursprüngliche Strophe war ein einziges Datenelement. Diese Daten haben Sie im ersten Schritt in einzelne Einträge aufgeteilt. Als Nächstes müssen Sie nach dem gewünschten Text suchen. In diesem Fall *können* Sie dies zwar erreichen, indem Sie die letzten beiden Einträge in der Liste auswählen. In einem ganzen Buch wäre es jedoch unrealistisch, dieses komplett durchzulesen und die Elemente manuell zu isolieren.

1. Verwenden Sie daher, anstatt manuell zu suchen, einen String.Contains-Block für die Suche nach einer Gruppe von Zeichen. Diese Funktion ist dem Befehl Suchen in Textverarbeitungsprogrammen ähnlich. In diesem Falle erhalten Sie True oder False als Ergebnis, je nachdem, ob die betreffende Teilzeichenfolge in einem Eintrag gefunden wurde.
2. Definieren Sie in der searchFor-Eingabe die Teilzeichenfolge, nach der Sie in der Strophe suchen. Verwenden Sie dazu einen String-Block mit dem Text "And miles".
3. Die Ausgabe zeigt eine Liste von True- und False-Werten. Im nächsten Schritt filtern Sie die Elemente mithilfe dieser Booleschen Logik.

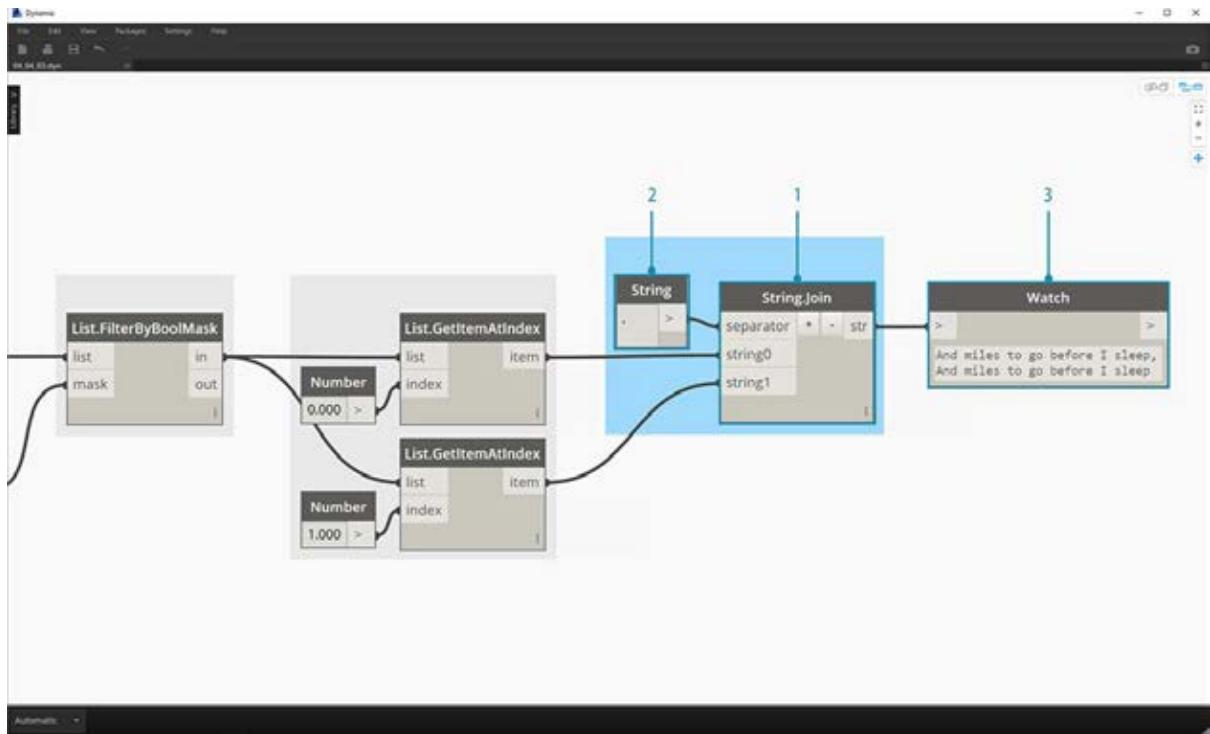


1. Mit einem List.FilterByBoolMask-Block bereinigen Sie die True- und False-Werte. Die in-Ausgabe gibt die Aussagen mit dem Wert True für die mask-Eingabe zurück, die out-Ausgabe diejenigen mit dem Wert False.
2. Die in-Ausgabe zeigt wie erwartet die letzten beiden Zeilen der Strophe.



Heben Sie jetzt die Wiederholung in der Strophe hervor, indem Sie diese beiden Zeilen zusammenführen. In der Ausgabe aus dem vorigen Schritt sehen Sie eine Liste mit zwei Einträgen.

1. Mithilfe zweier List.GetItemAtIndex-Blöcke können Sie die Einträge durch Angabe der Werte 0 bzw. 1 für die index-Eingabe isolieren.
2. Die Ausgabe jedes Blocks zeigt die letzten beiden Zeilen in der angegebenen Reihenfolge.



Verwenden Sie zum Zusammenführen der beiden Einträge einen String.Join-Block:

1. Nachdem Sie String.Join hinzugefügt haben, zeigt sich, dass ein Trennzeichen benötigt wird.
2. Fügen Sie zum Erstellen des Trennzeichens im Ansichtsbereich einen String-Block hinzu und geben Sie ein Komma ein.
3. In der letzten Ausgabe wurden beide Einträge zu einem zusammengeführt.

Dieser Vorgang zum Isolieren der letzten beiden Zeilen mag recht aufwendig erscheinen. Für Operationen mit Zeichenfolgen sind in der Tat zuweilen einige Vorarbeiten nötig. Diese Operationen sind jedoch skalierbar und können relativ leicht auf große Datensätze angewendet werden. Berücksichtigen Sie bei der parametrischen Arbeit mit Kalkulationstabellen und Interoperabilität die Operationen für Zeichenfolgen.

# Farbe

## Farbe

Der Datentyp Farbe eignet sich ausgezeichnet zum Erstellen beeindruckender Präsentationen sowie zur Darstellung von Unterschieden in der Ausgabe Ihres Grafikprogramms. Bei der Arbeit mit abstrakten Daten und veränderlichen Zahlen ist es zuweilen schwierig, zu erkennen, was sich in welchem Umfang verändert. Dies ist ein nützlicher Anwendungsbereich für Farben.

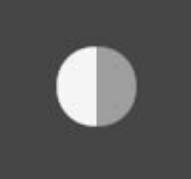
### Erstellen von Farben

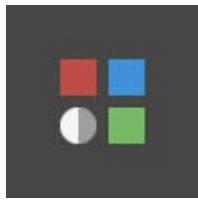
Farben werden in Dynamo mithilfe von ARGB-Eingaben erstellt. Dies entspricht den Angaben Alpha, Rot, Grün und Blau. Der Alpha-Kanal gibt die *Transparenz* der Farbe an, während die drei anderen Angaben als Primärfarben zur Darstellung des gesamten Farbspektrums verwendet werden.

Symbol	Name	Syntax	Eingaben	Ausgaben
	ARGB-Farbe	Color.ByARGB	A, R, G, B	color

### Abfragen von Farbwerten

Mithilfe der Farben in der Tabelle unten werden die Eigenschaften zum Definieren von Farben abgefragt: Alpha, Rot, Grün und Blau. Beachten Sie, dass der Block Color.Components alle vier unterschiedlichen Ausgaben bereitstellt. Diesem Block ist daher für die Abfrage der Eigenschaften einer Farbe der Vorzug zu geben.

Symbol	Name	Syntax	Eingaben	Ausgaben
	Alpha	Color.Alpha	color	A
	Rot	Color.Red	color	R
	Grün	Color.Green	color	G
	Blau	Color.Blue	color	B



Komponenten Color.Components color      A, R, G, B

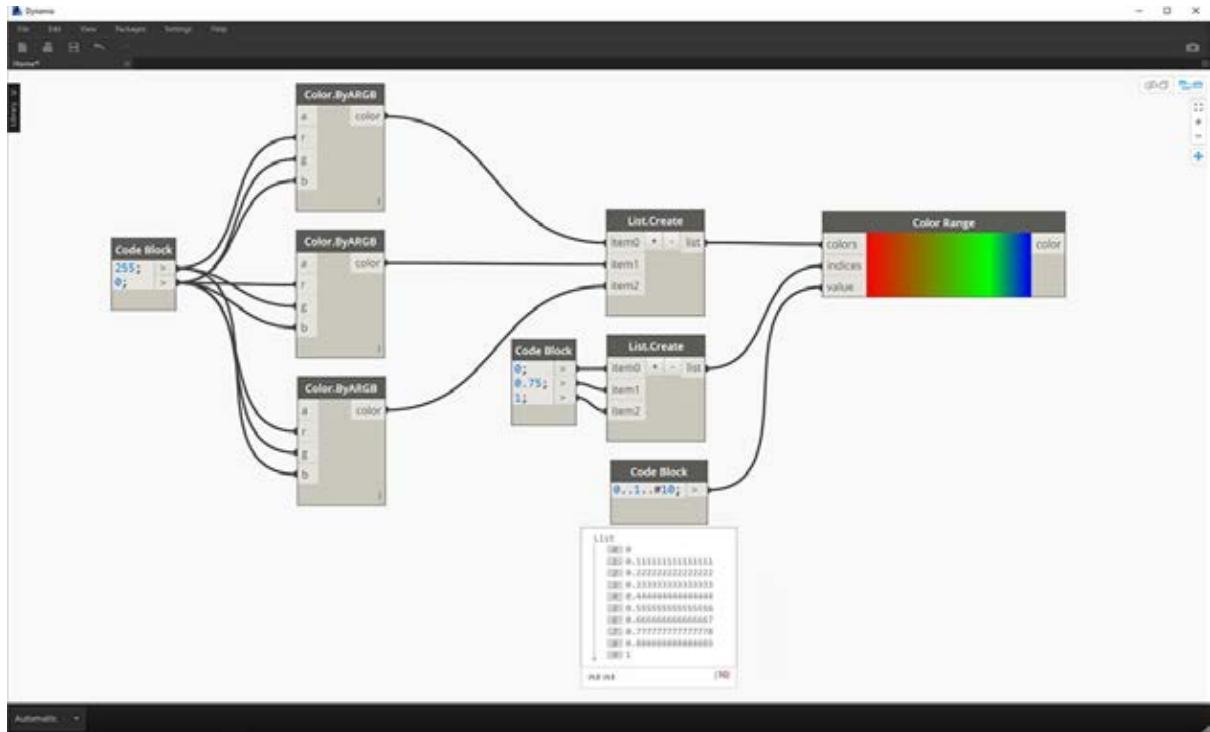
Die Farben in der Tabelle unten entsprechen dem **HSB-Farbraum**. Die Interpretation einer anhand von Farbton, Sättigung und Helligkeit definierten Farbe mag intuitiver scheinen: Welche Farbe ist gemeint? Wie intensiv soll sie sein? Wie hell oder dunkel soll die Farbe sein? Dies ist die Differenzierung nach Farbton, Sättigung und Helligkeit.

Symbol	Abfragename	Syntax	Eingaben	Ausgaben
	Farbton	Color.Hue	color	Hue
	Sättigung	Color.Saturation	color	Saturation
	Helligkeit	Color.Brightness	color	Brightness

## Farbbereich

Der Farbbereich ist dem **Remap Range**-Block aus Abschnitt 4.2 ähnlich: Eine Liste von Zahlen wird in einer anderen Domäne neu zugeordnet. Allerdings wird sie nicht einer *number*-Domäne, sondern anhand eingegebener Zahlenwerte zwischen 0 und 1 einem *Farbverlauf* zugeordnet.

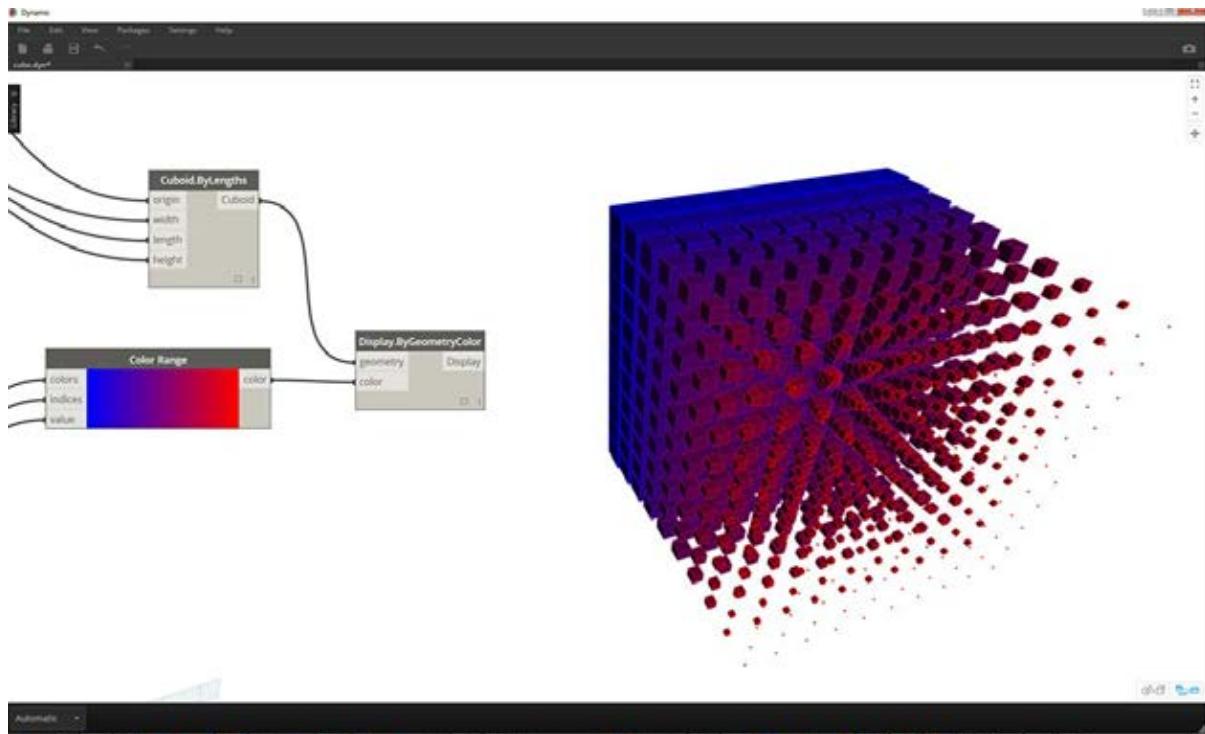
Der derzeit verwendete Block funktioniert problemlos, seine Funktionsweise ist jedoch anfangs möglicherweise sehr ungewohnt. Die beste Möglichkeit, mit dem Farbverlauf vertraut zu werden, besteht darin, ihn in der Praxis auszuprobieren. Die folgende kurze Übung zeigt, wie Sie einen Farbverlauf mit Ausgabe der Farben anhand von Zahlen erstellen können.



- Farben definieren:** Definieren Sie mithilfe eines Codeblock-Blocks *Rot*, *Grün* und *Blau*, indem Sie die jeweiligen Kombinationen von 0 und 255 verbinden.
- Liste erstellen:** Führen Sie die drei Farben zu einer Liste zusammen.
- Definieren Sie Indizes:** Erstellen Sie eine Liste zum Definieren der Griffpositionen für jede Farbe (von 0 bis 1). Beachten Sie, dass für Grün der Wert 0.75 festgelegt wurde. Dadurch wird die Farbe Grün an der Position bei 3/4 der Strecke des horizontalen Verlaufs im Schieberegler für den Farbbereich platziert.
- Code Block:** Geben Sie Werte (zwischen 0 und 1) zum Versetzen der Farben ein.

## Farbvorschau

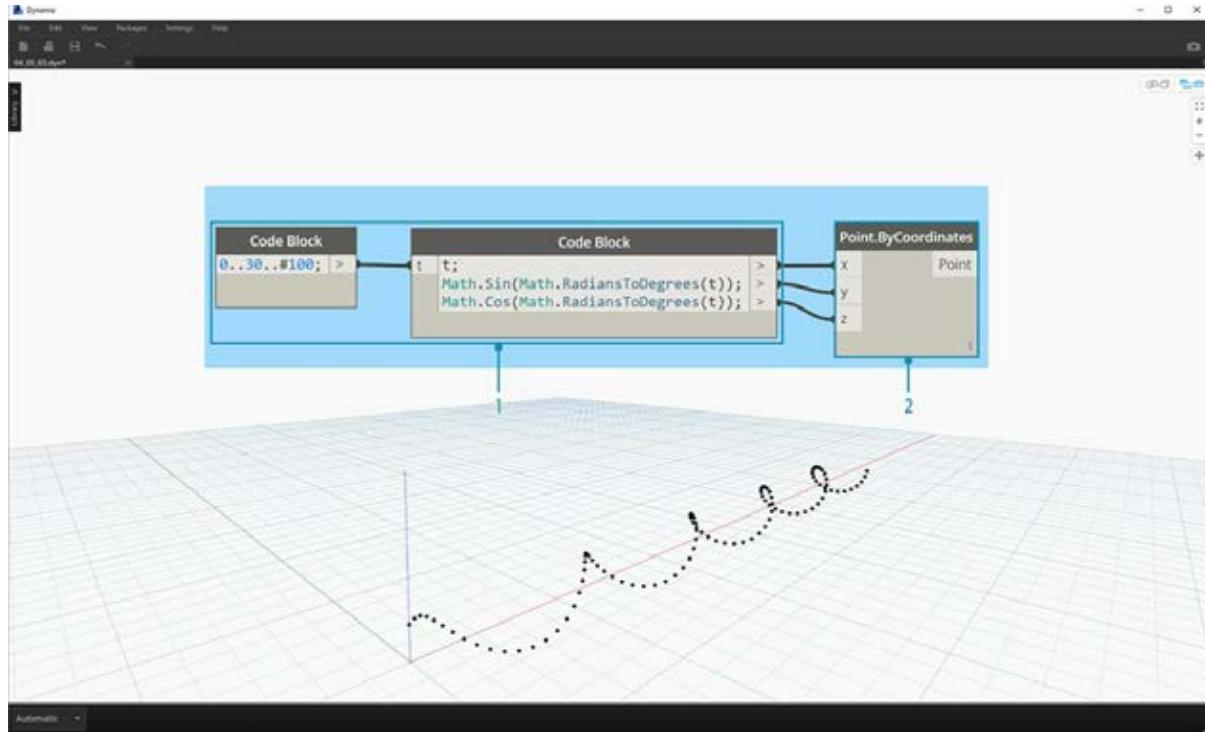
Der **Display.ByGeometry**-Block ermöglicht die farbige Darstellung von Geometrie im Ansichtsfenster. Dies ist hilfreich zur Unterscheidung verschiedenartiger Geometrie, zur Verdeutlichung eines parametrischen Konzepts oder zum Definieren einer Analyselegende für die Simulation. Die Eingaben sind einfach: *geometry* und *color*. Um einen Farbverlauf wie in der Abbildung oben gezeigt zu erstellen, wird die *color*-Eingabe mit dem **Color Range**-Block verbunden.



## Übung zu Farben

Laden Sie die Beispieldatei für diese Übungslektion herunter (durch Rechtsklicken und Wahl von "Save Link As..."):  
[Building Blocks of Programs - Color.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

In dieser Übung soll Farbe zusammen mit Geometrie parametrisch gesteuert werden. Die Geometrie ist eine einfache, wie unten gezeigt in einem **Code Block** (3.2.3) definierte Schraubenform. Dies ist ein schnelles und einfaches Verfahren zum Erstellen parametrischer Funktionen. Da das Thema dieses Abschnitts nicht Geometrie, sondern Farbe ist, wird die Helix auf effiziente Weise mithilfe des Codeblocks erstellt, ohne dass zu viel Platz im Ansichtsbereich beansprucht wird. Codeblöcke werden in den weiteren Kapiteln dieses Handbuchs bei der Behandlung komplexerer Themen häufiger verwendet.

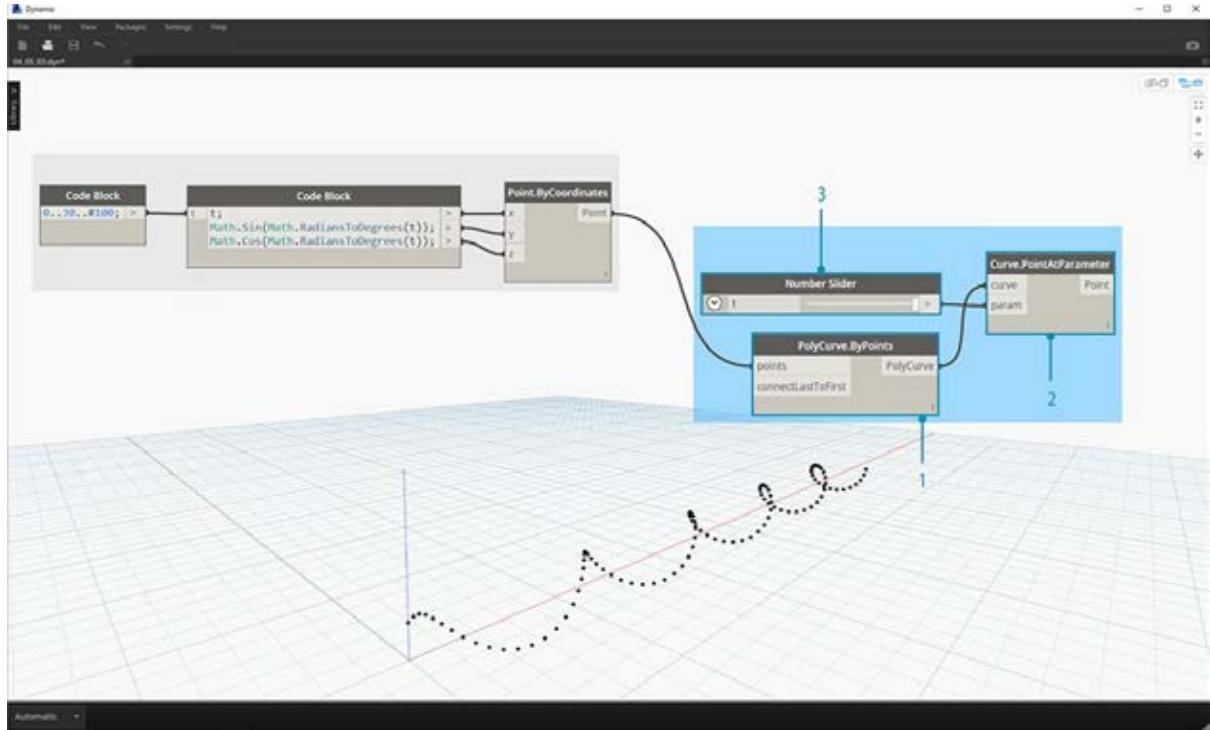


1. **Code Block:** Definieren Sie die beiden Codeblöcke mit den oben gezeigten Formeln. Dies ist eine schnelle

parametrische Methode zum Erstellen einer Helix.

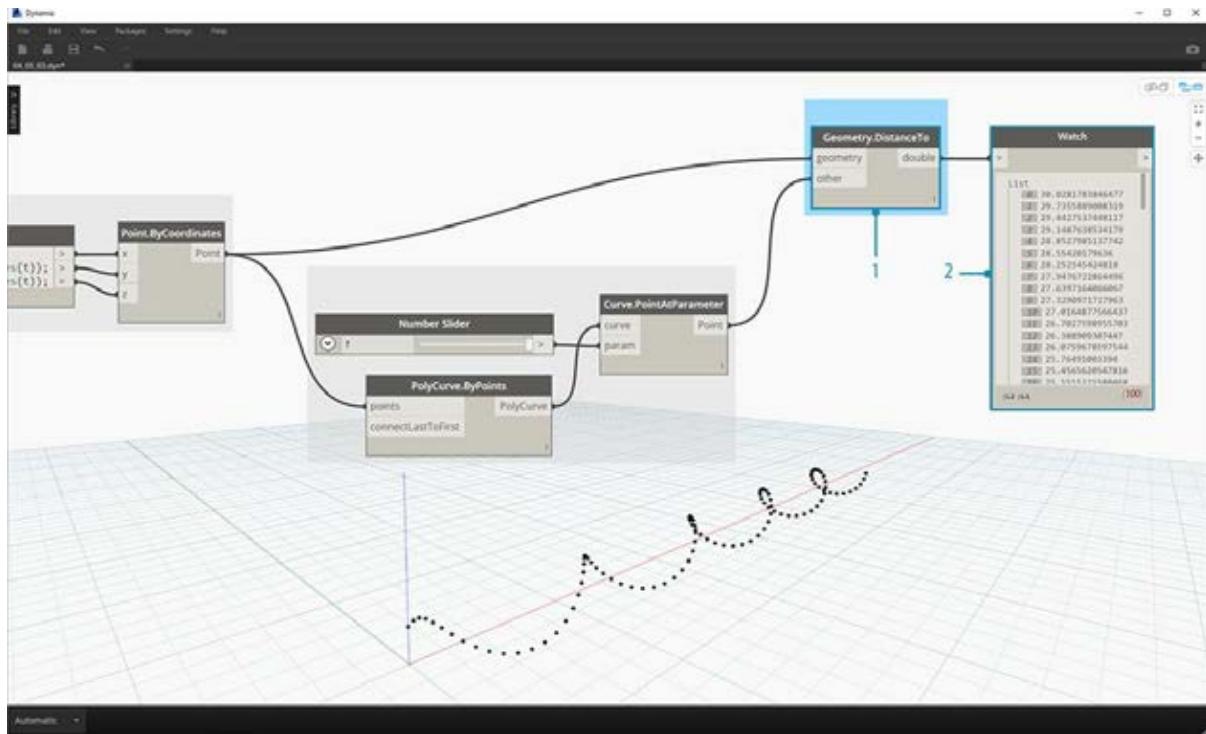
2. **Point.ByCoordinates:** Verbinden Sie die drei Ausgaben des Codeblocks mit den Koordinaten dieses Blocks.

Daraufhin wird ein Array aus Punkten angezeigt, das eine Helixform bildet. Im nächsten Schritt erstellen Sie zur Visualisierung der Helix eine Kurve durch diese Punkte.



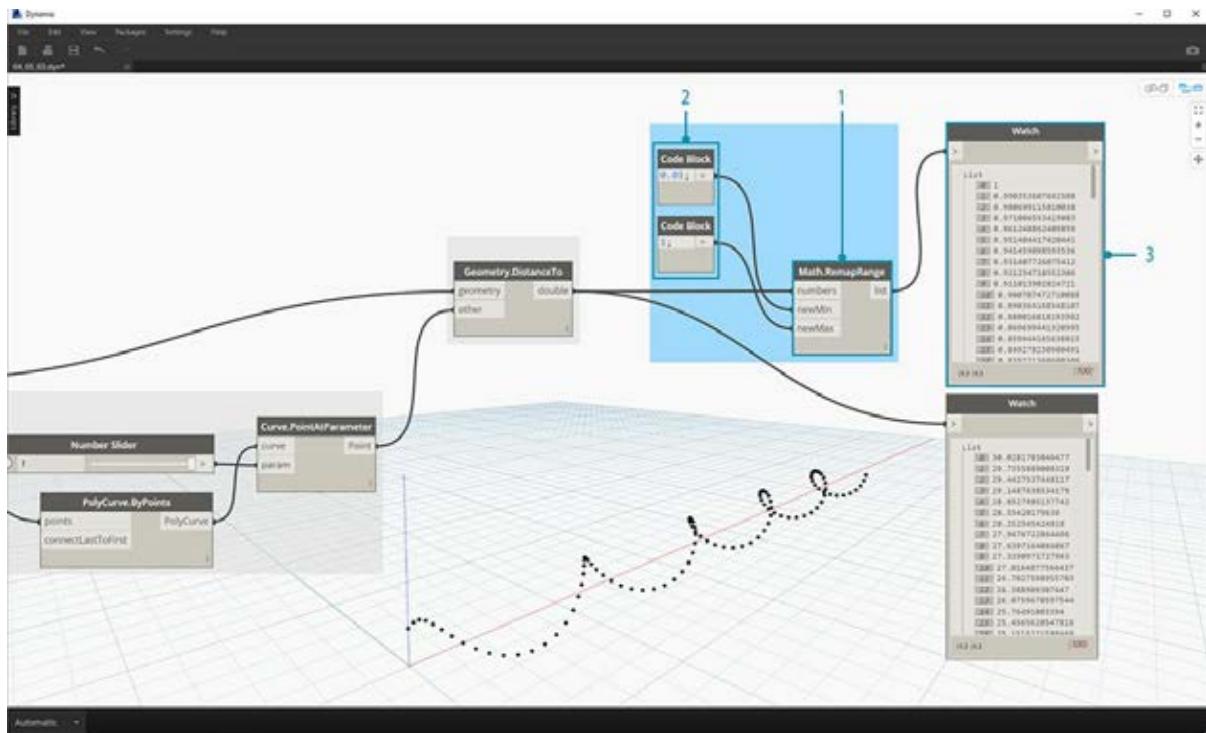
1. **PolyCurve.ByPoints:** Verbinden Sie die *Point.ByCoordinates*-Ausgabe mit der *points*-Eingabe für den Block. Sie erhalten eine schraubenförmige Kurve.
2. **Curve.PointAtParameter:** Verbinden Sie die *PolyCurve.ByPoints*-Ausgabe mit der *curve*-Eingabe. Mithilfe dieses Schritts erstellen Sie einen geometrischen Attraktorpunkt, der entlang der Kurve verschoben werden kann. Da die Kurve einen Punkt an einer Parameterposition auswertet, müssen Sie einen *param*-Wert zwischen 0 und 1 eingeben.
3. **Number Slider:** Nachdem Sie diesen Block im Ansichtsbereich hinzugefügt haben, ändern Sie seinen *min*-Wert in 0.0, den *max*-Wert in 1.0 und den *step*-Wert in .01. Verbinden Sie die Ausgabe des Schiebereglers mit der *param*-Eingabe für *Curve.PointAtParameter*. Daraufhin wird auf der Helix ein Punkt angezeigt, dessen Position durch die Prozentangabe im Schieberegler definiert wird (0 am Startpunkt, 1 am Endpunkt).

Nachdem Sie den Referenzpunkt erstellt haben, vergleichen Sie die Abstände vom Referenzpunkt zu den Originalpunkten, durch die die Helix definiert ist. Der Wert für diesen Abstand bestimmt sowohl die Geometrie als auch die Farbe.



1. **Geometry.DistanceTo:** Verbinden Sie die Ausgabe von *Curve.PointAtParameter* mit der *other*-Eingabe.  
Verbinden Sie *Point.ByCoordinates* mit der \*geometry-Eingabe.
2. **Watch:** Die Ausgabe zeigt als Ergebnis eine Liste der Abstände von jedem der Punkte auf der Helix zum Referenzpunkt.

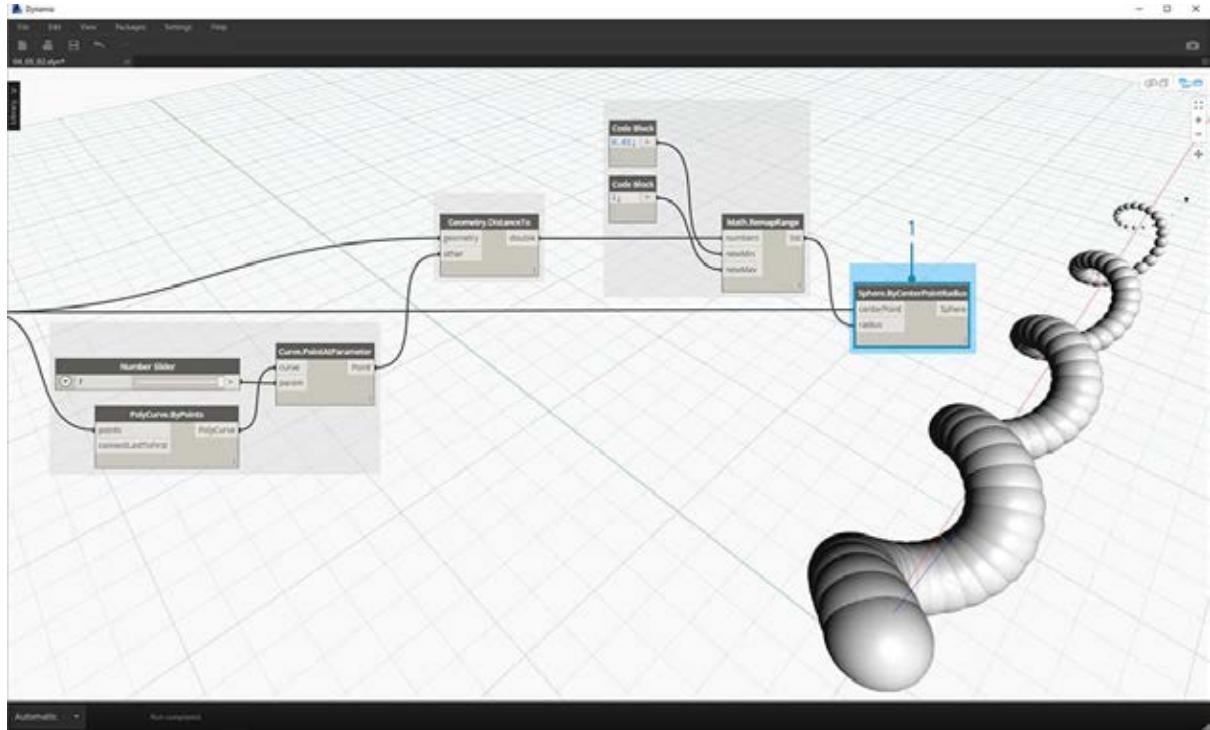
Im nächsten Schritt soll die Liste der Abstände zwischen den Helixpunkten und dem Referenzpunkt zur Steuerung von Parametern genutzt werden. Mithilfe dieser Werte werden die Radien einer Reihe von Kugeln entlang der Kurve definiert. Damit die Kugeln die richtige Größe annehmen, müssen Sie die Werte der Abstände mithilfe von *remap* neu zuordnen.



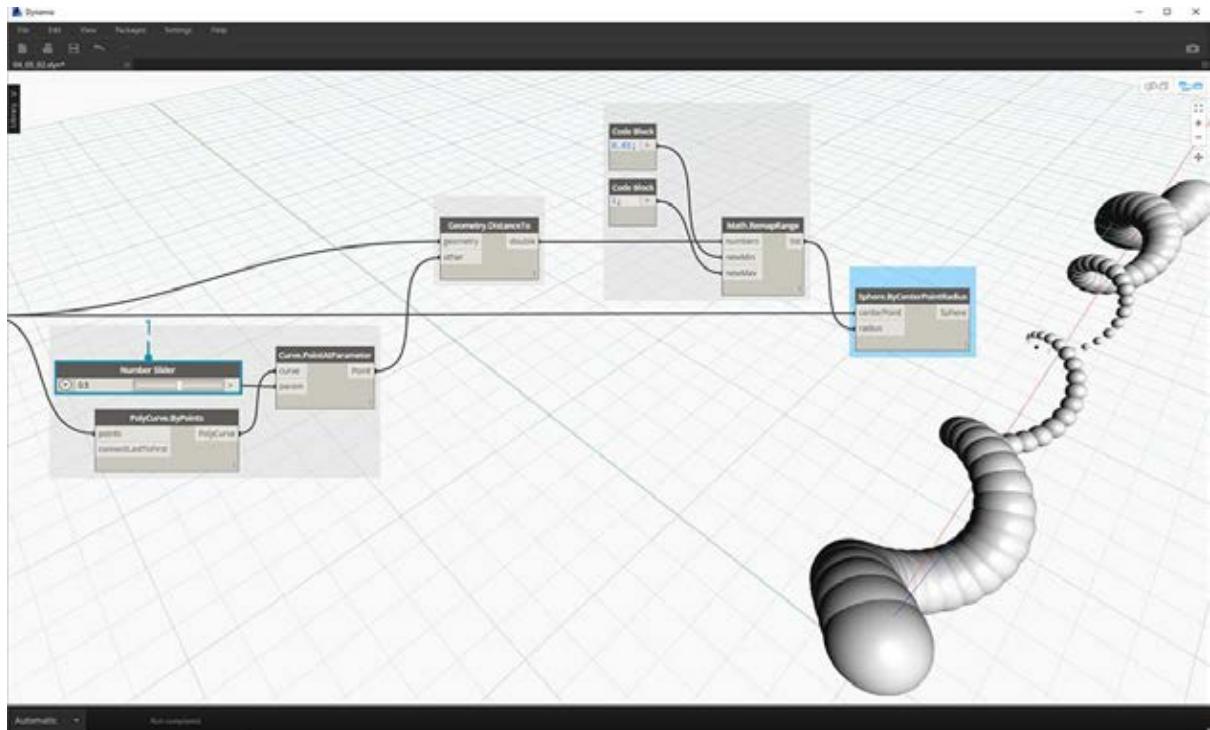
1. **Math.RemapRange:** Verbinden Sie die Ausgabe von *Geometry.DistanceTo* mit der *numbers*-Eingabe.
2. **Code Block:** Verbinden Sie einen Codeblock mit dem Wert 0.01 mit der *newMin*-Eingabe und einen zweiten Codeblock mit dem Wert 1 mit der *newMax*-Eingabe.
3. **Watch:** Verbinden Sie die Ausgabe von *Math.RemapRange* mit einem solchen Block und die Ausgabe von

*Geometry.DistanceTo* mit einem zweiten. Vergleichen Sie die Ergebnisse.

Mit diesem Schritt haben Sie die Liste der Abstände auf einen kleineren Bereich reduziert. Sie können die Werte für *newMin* und *newMax* wie gewünscht bearbeiten. Die Werte werden neu zugeordnet, wobei ihre *proportionale Verteilung* im Bereich erhalten bleibt.



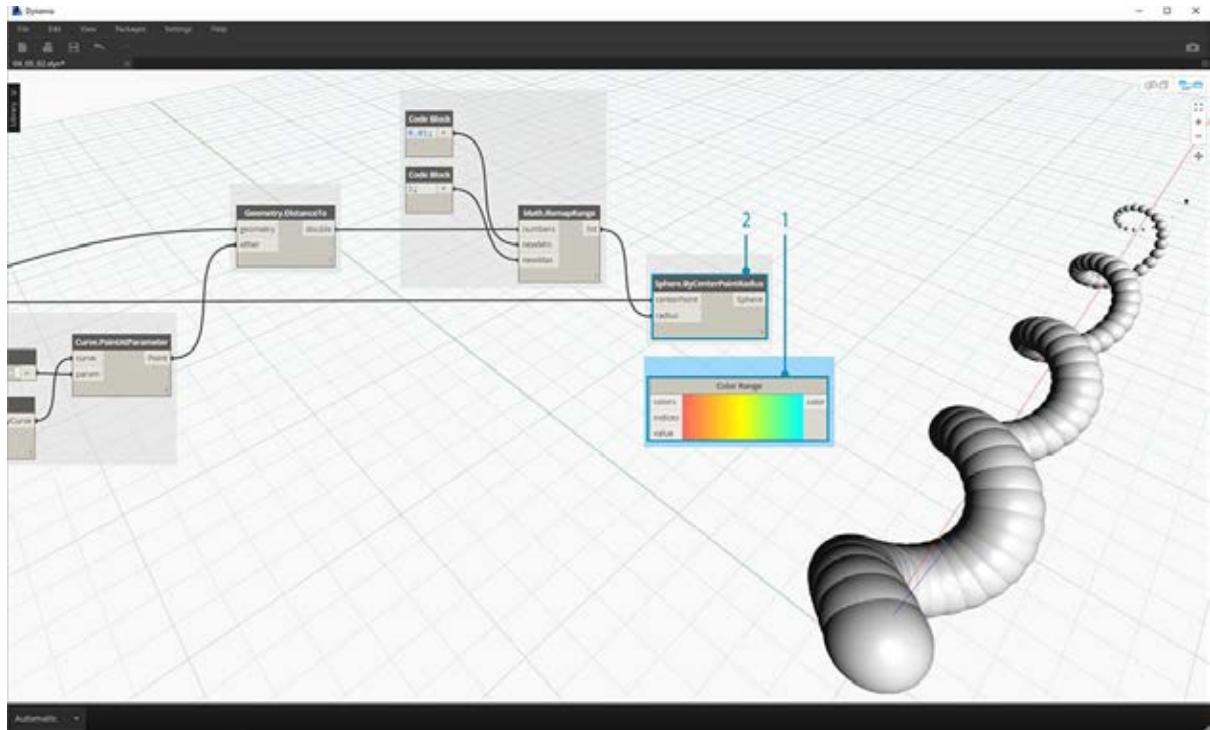
1. **Sphere.ByCenterPointRadius:** Verbinden Sie die Ausgabe von *Math.RemapRange* mit der *radius*-Eingabe und die ursprüngliche Ausgabe von *Point.ByCoordinates* mit der *centerPoint*-Eingabe.



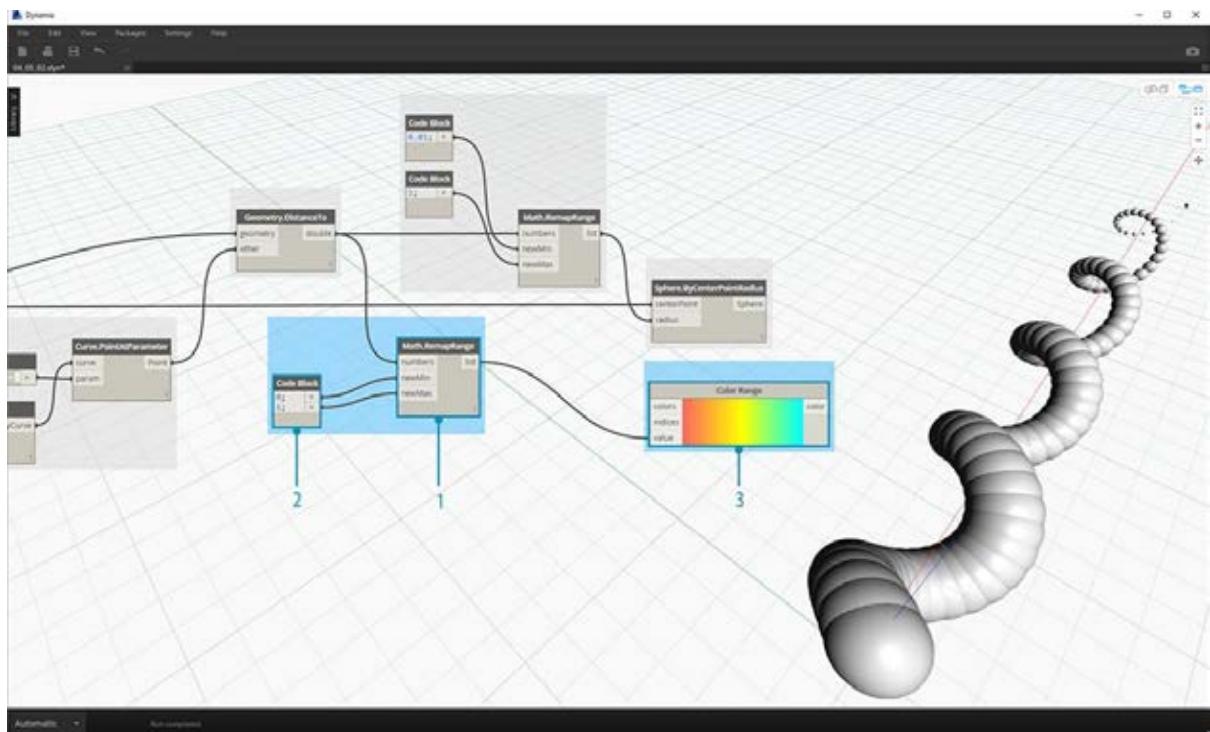
1. **Number Slider:** Ändern Sie den Wert im Schieberegler und beobachten Sie, wie die Größe der Kugeln aktualisiert wird. Damit haben Sie eine parametrische Schablone erstellt.

Die Größe der Kugeln zeigt das parametrische Array, das durch den Referenzpunkt entlang der Kurve definiert wird. Als

Nächstes soll die Farbe der Kugeln nach demselben Prinzip wie ihr Radius bestimmt werden.

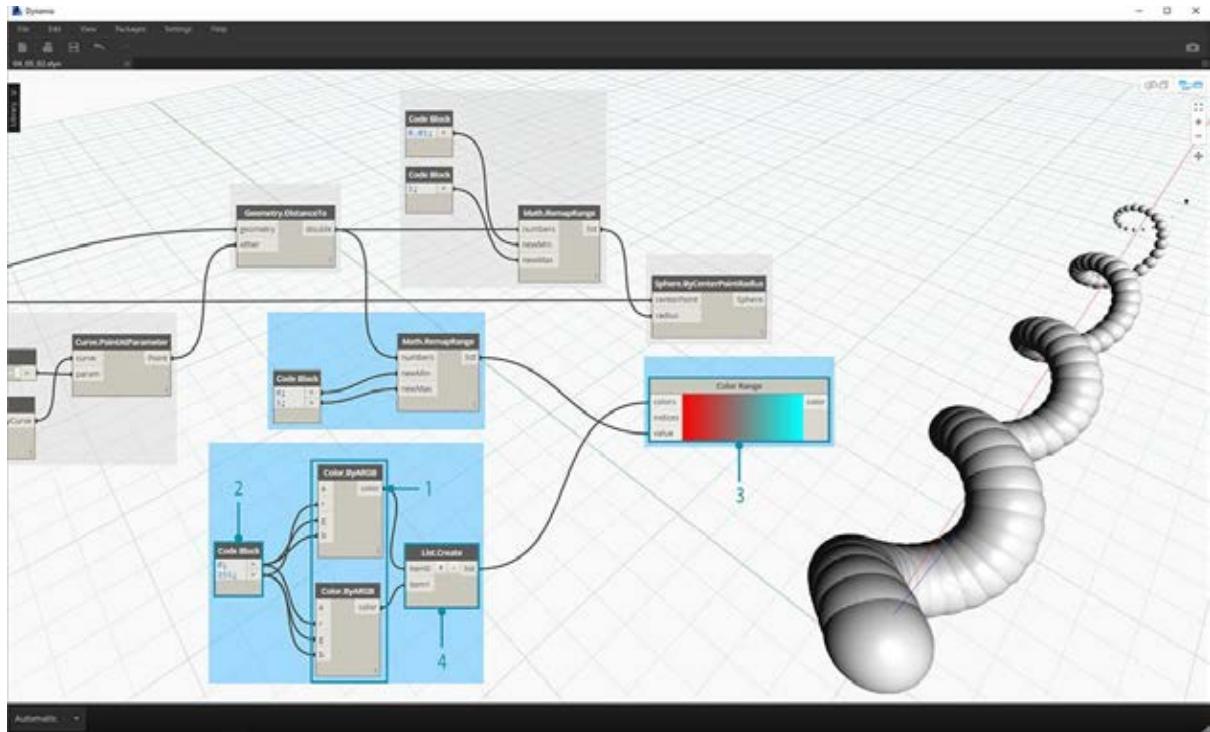


- Color Range:** Fügen Sie diesen Block im Ansichtsbereich hinzu. Wenn Sie den Mauszeiger auf die *value*-Eingabe setzen, fällt auf, dass die angeforderten Zahlenwerte zwischen 0 und 1 liegen. Sie müssen die Zahlen aus der *Geometry.DistanceTo*-Auscgabe neu zuordnen, damit sie mit diesem Bereich kompatibel sind.
- Sphere.ByCenterPointRadius:** Deaktivieren Sie vorübergehend die Vorschau für diesen Block (*Rechtsklick > Vorschau*).

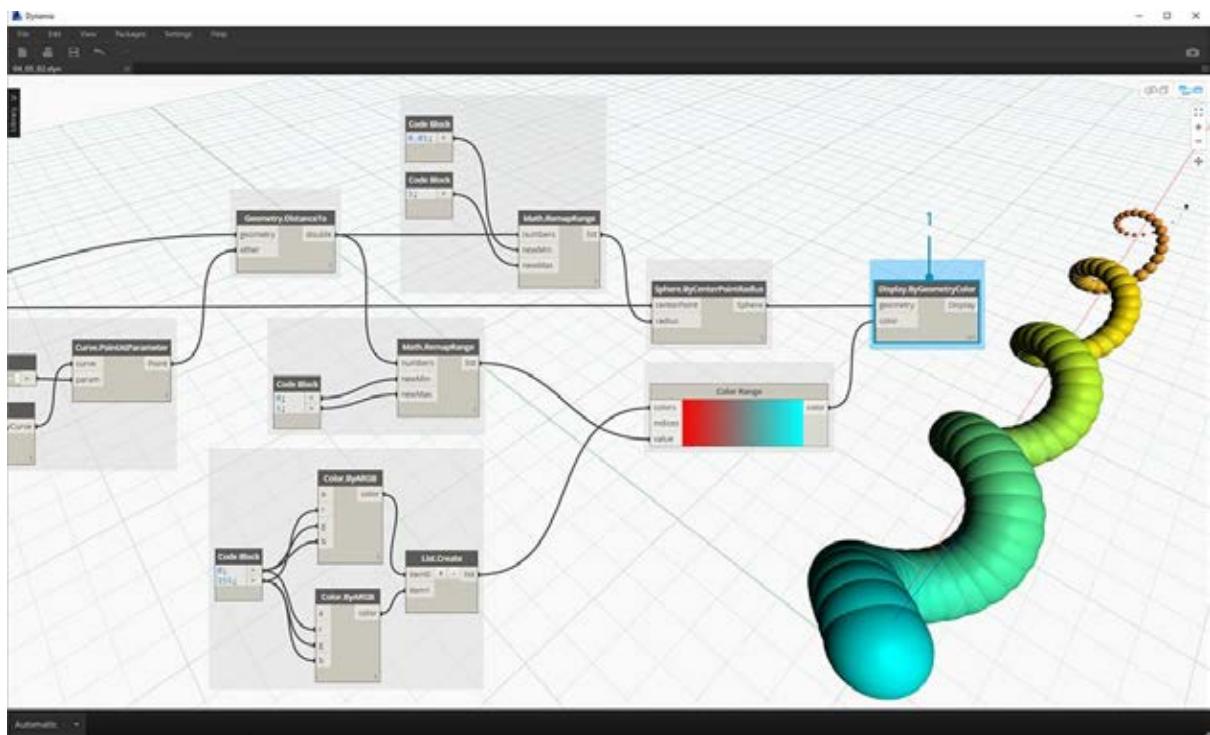


- Math.RemapRange:** Dieser Vorgang ist inzwischen bekannt. Verbinden Sie die Ausgabe von *Geometry.DistanceTo* mit der *numbers*-Eingabe.
- Code Block:** Erstellen Sie ähnlich wie in einem der vorigen Schritte den Wert 0 für die *newMin*-Eingabe und den Wert 1 für die *newMax*-Eingabe. In diesem Fall können Sie zwei Ausgaben aus demselben Codeblock erstellen.

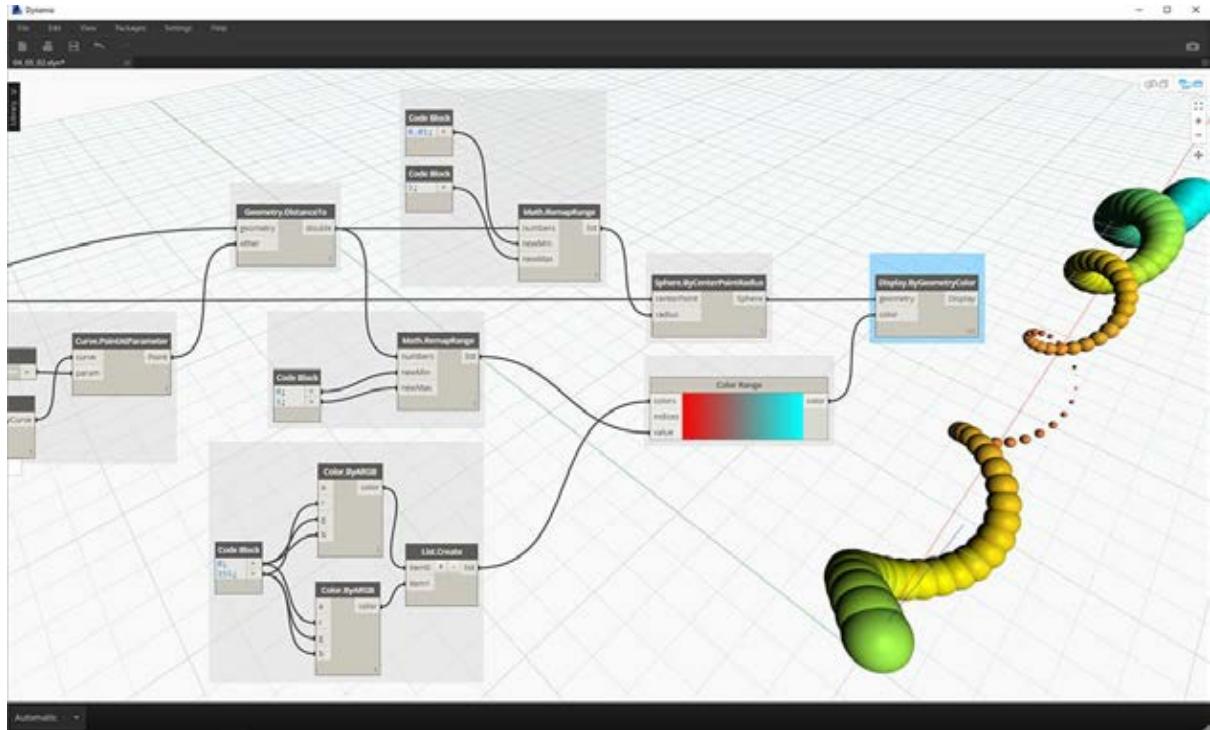
**3. Color Range:** Verbinden Sie die *Math.RemapRange*-Ausgabe mit der *value*-Eingabe.



- 1. Color.ByARGB:** Mithilfe dieses Blocks erstellen Sie zwei Farben. Dieser Prozess wirkt eventuell etwas umständlich, unterscheidet sich jedoch nicht von den RGB-Farben in anderen Programmen. Er wird hier lediglich mithilfe visueller Programmierung durchgeführt.
- 2. Code Block:** Erstellen Sie die beiden Werte 0 und 255. Verbinden Sie die beiden Ausgaben mit den Eingaben der beiden *Color.ByARGB*-Blöcke wie in der Abbildung oben gezeigt (oder definieren Sie ganz nach Wunsch Ihre eigenen Farben).
- 3. Color Range:** Für die *colors*-Eingabe ist eine Liste von Farben erforderlich. Diese Liste müssen Sie aus den beiden im vorigen Schritt erstellten Farben erstellen.
- 4. List.Create:** Führen Sie die beiden Farben zu einer Liste zusammen. Verbinden Sie die Ausgabe mit der *colors*-Eingabe von *Color Range*.



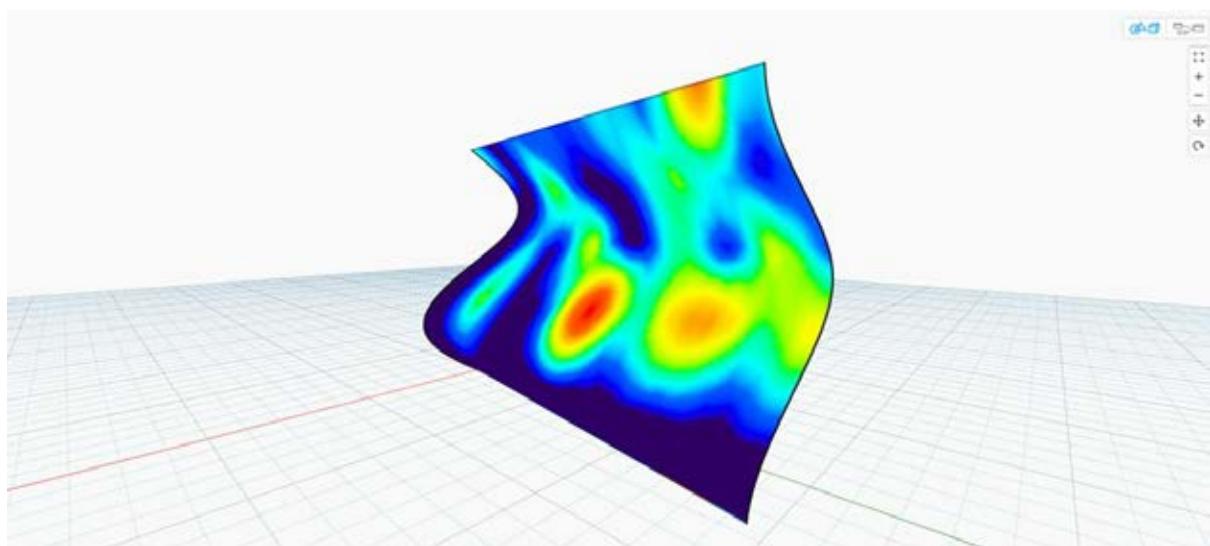
1. **Display.ByGeometryColor:** Verbinden Sie *Sphere.ByCenterPointRadius* mit der *geometry*-Eingabe und *Color Range* mit der *color*-Eingabe. Damit erhalten Sie einen fließenden Farbübergang über die Länge der Kurve.



Wenn Sie den Wert im *Number Slider* aus einem früheren Schritt in der Definition ändern, werden die Farben und Größen aktualisiert. Zwischen den Farben und den Radien besteht in diesem Fall ein direkter Zusammenhang: Damit haben Sie eine visuelle Verknüpfung zweier Parameter erstellt.

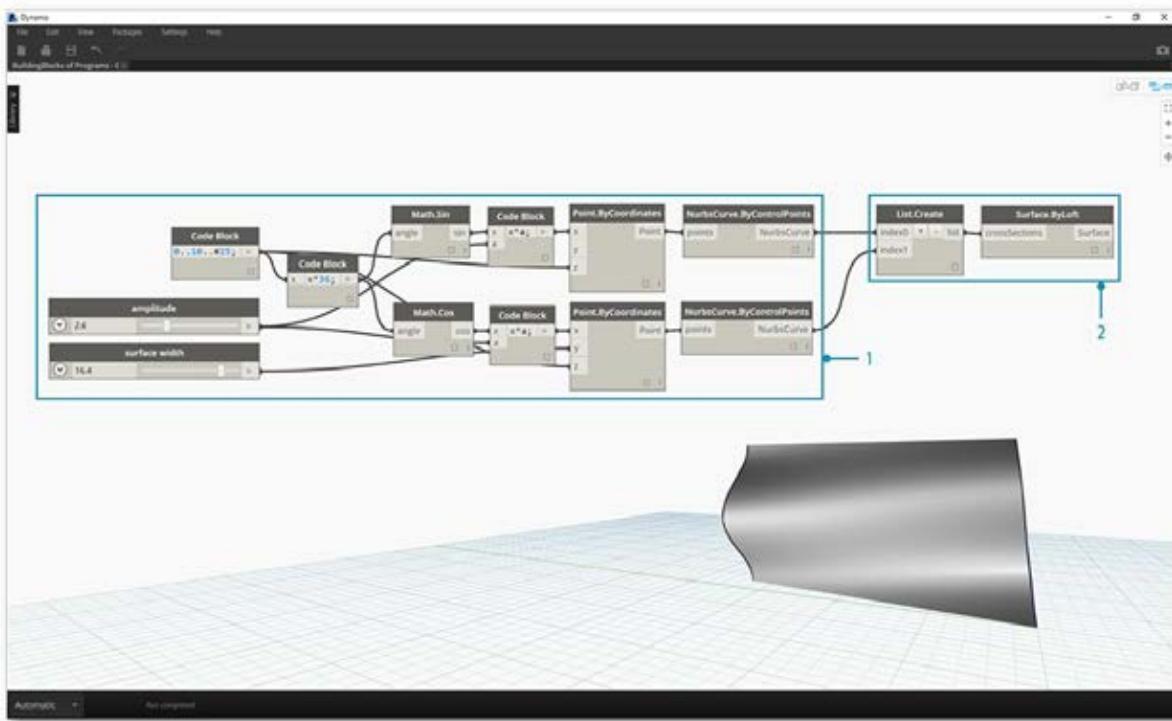
### Farbe auf Oberflächen

Der **Display.BySurfaceColors**-Block ermöglicht die Zuordnung von Daten auf einer Oberfläche mithilfe von Farben. Mithilfe dieser Funktion können Sie durch diskrete Auswertung gewonnene Daten, etwa zu Sonneneinstrahlung, Energie und Entfernung, überzeugend veranschaulichen. Die Anwendung von Farben auf eine Oberfläche in Dynamo ist der Anwendung von Texturen auf Materialien in anderen CAD-Umgebungen ähnlich. Die folgende kurze Übung demonstriert die Verwendung dieses Werkzeugs.



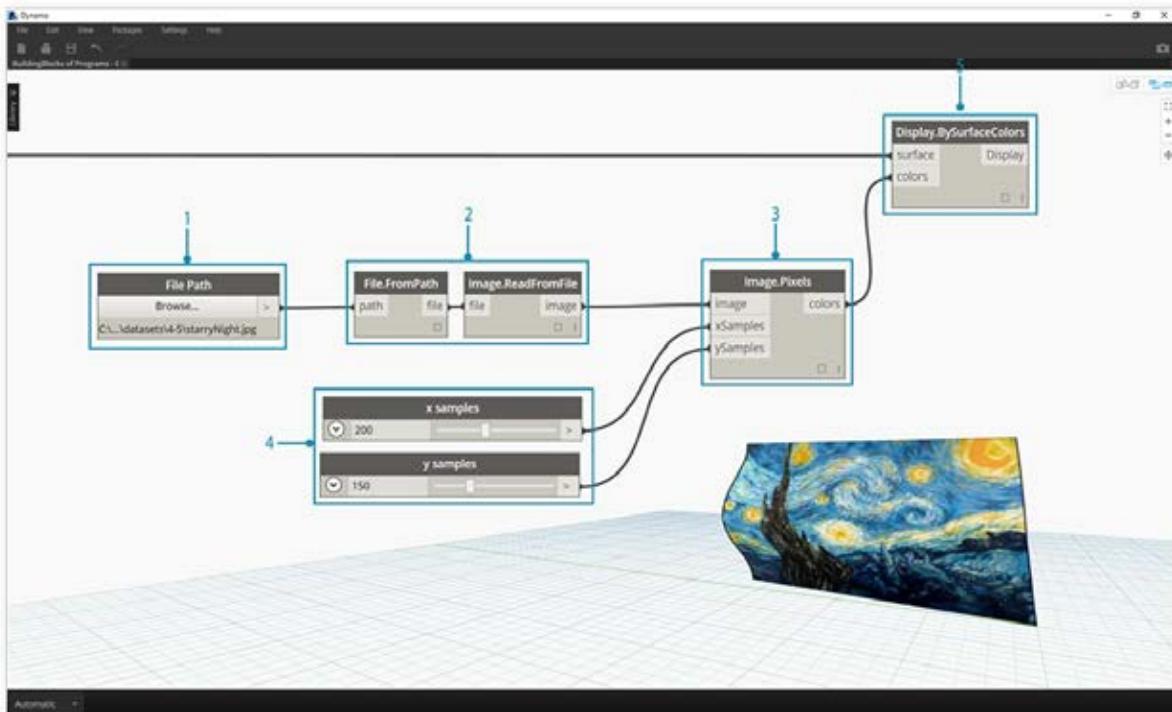
### Übungslektion: Farbe auf Oberflächen

Laden Sie die Beispieldatei für diese Übungslektion herunter (durch Rechtsklicken und Wahl von "Save Link As..."):  
[Building Blocks of Programs - ColorOnSurface.zip](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



Als Erstes müssen Sie eine Oberfläche erstellen (oder referenzieren), die als Eingabe für den **Display.BySurfaceColors**-Block verwendet werden soll. In diesem Beispiel wird dies durch Erhebung zwischen einer Sinus- und einer Kosinuskurve erreicht.

1. Die oben gezeigte **Gruppe** von Blöcken erstellt Punkte entlang der z-Achse und verschiebt sie anschließend unter Verwendung von Sinus- und Kosinusfunktionen. Aus den beiden Punktlisten werden anschließend Nurbs-Kurven erstellt.
2. **Surface.ByLoft**: Dieser Block erstellt eine interpolierte Oberfläche zwischen den Nurbs-Kurven in der Kurvenliste.



1. **File Path**: Wählen Sie die Bilddatei, aus der Samples für die in den folgenden Schritten zu verwendenden Pixeldaten entnommen werden sollen.

2. Konvertieren Sie mithilfe von **File.FromPath** den Dateipfad in eine Datei und übergeben Sie diese an **Image.ReadFromFile**, sodass ein Bild für das Sampling ausgegeben wird.
3. **Image.Pixels**: Geben Sie ein Bild ein und legen Sie die Sample-Werte für die x- und y-Richtung des Bilds fest.
4. **Schieberegler**: Verwenden Sie diese zum Angeben der Sample-Werte für **Image.Pixels**.
5. **Display.BySurfaceColors**: Dieser Block ordnet das Array der Farbwerte den Positionen auf der Oberfläche in x- und y-Richtung zu.

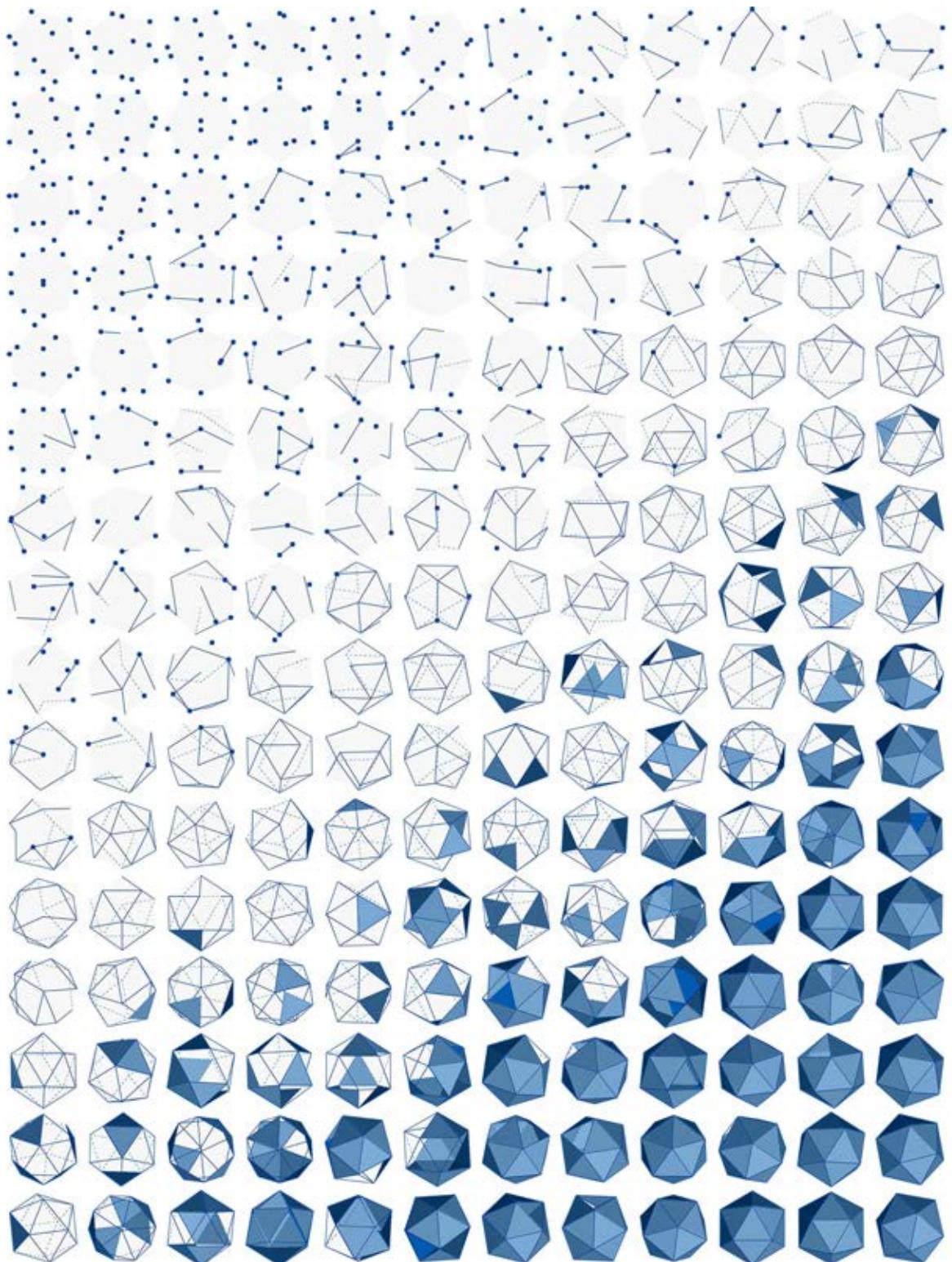


Detailansicht: Vorschau der ausgegebenen Oberfläche mit der Auflösung 400 x 300 Samples

# **Geometrie für Computational Design**

## **Geometrie für Computational Design**

Als visuelle Programmierumgebung bietet Dynamo Ihnen die Möglichkeit, die Art und Weise, in der Daten verarbeitet werden, selbst zu gestalten. Daten sind Zahlen oder Text. Geometrie gehört ebenfalls dazu. Aus der Perspektive des Computers ist Geometrie – auch als Computational Geometry bezeichnet – nicht anderes als die Daten, aus denen Sie wunderschöne, komplexe oder funktionsorientierte Modelle erstellen können. Hierzu müssen Sie die Eigenschaften der verschiedenen Typen von Geometrie kennen, die zur Verfügung stehen.



# Geometrie – Überblick

## Geometrie – Überblick

**Geometrie** ist die Sprache der Konstruktion. Wenn eine Programmiersprache oder Programmierungsumgebung in seinem Kern einen geometrischen Kernel aufweist, können Sie die Möglichkeiten für das Konstruktionspräziser und robuster Modelle, die Automatisierung von Konstruktionsroutinen und die Generierung von Konstruktionsiterationen mit Algorithmen erschließen.

### Die Grundlagen

Die Geometrie ist nach traditioneller Definition die Studie der Form, Größe, relativen Position von Zahlen und der Eigenschaften im Raum. Dieser Bereich weist eine reiche Geschichte auf, die Tausende von Jahren zurückreicht. Mit dem Aufkommen und der Verbreitung des Computers verfügen Sie über ein leistungsstarkes Werkzeug für die Definition, Erforschung und Generierung von Geometrie. Es ist heute ein Leichtes, das Ergebnis komplexerer geometrischer Interaktionen zu berechnen. Die Tatsache, dass dies getan wird, ist fast transparent.



Wenn Sie neugierig sind und mithilfe Ihres Computers herausfinden wollen, wie vielfältig und komplex Geometrie sein kann, führen Sie eine schnelle Internetsuche nach dem Stanford Bunny durch – einem kanonischen Modell zum Testen von Algorithmen.

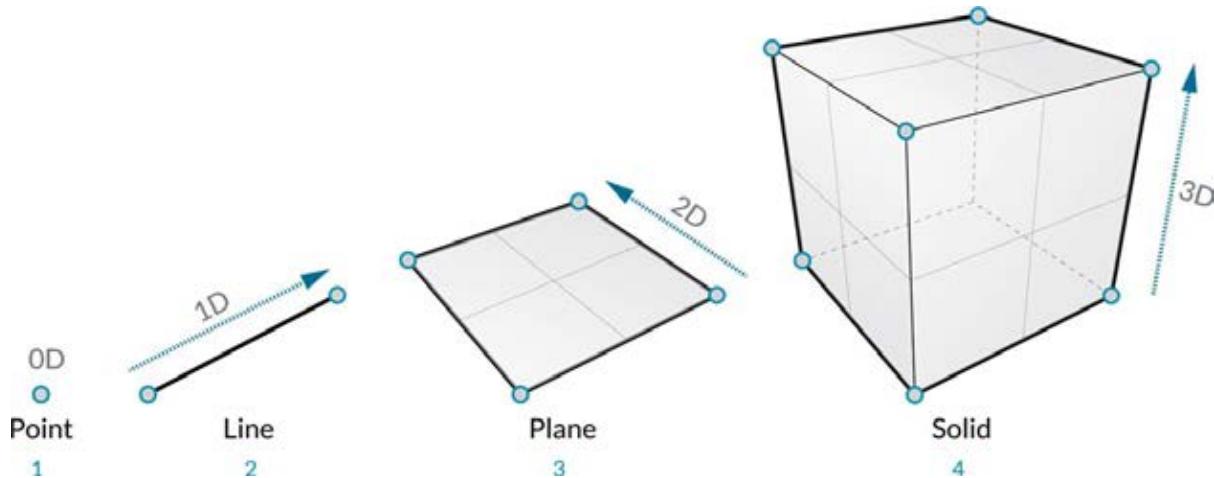
Das Verstehen von Geometrie im Kontext von Algorithmen, Berechnungen und Komplexität kann wie eine gewaltige Aufgabe erscheinen, es gibt jedoch einige wichtige und relativ einfachen Prinzipien, die als Grundlagen für weitergehende Anwendungen etabliert werden können:

1. Geometrie sind **Daten**: Für den Computer und Dynamo unterscheidet sich ein Bunny nicht wesentlich von einer Zahl.
2. Geometrie basiert auf **Abstraktion** – grundsätzlich werden geometrische Elemente durch Zahlen, Beziehungen und Formeln innerhalb eines bestimmten räumlichen Koordinatensystems beschrieben.
3. Geometrie verfügt über eine **Hierarchie** – Punkte werden verbunden, um Linien zu bilden, Linien werden verbunden, um Flächen zu bilden, usw.
4. Geometrie beschreibt gleichzeitig sowohl **das Bauteil als auch das Ganze** – bei einer Kurve handelt es sich sowohl um die Form als auch um alle möglichen Punkte entlang der Kurve

In der Praxis bedeutet dies, dass uns bewusst sein muss, womit wir arbeiten (welche Art von Geometrie, wie sie erzeugt wurde usw.), damit wir fließend unterschiedliche Geometrien zusammenstellen, zerlegen und neu zusammensetzen können, um komplexere Modelle zu entwickeln.

### Durchlaufen der Hierarchie

Nehmen Sie sich etwas Zeit, um die Beziehung zwischen der abstrakten und hierarchischen Beschreibung von Geometrie näher zu betrachten. Da diese beiden Konzepte miteinander verbunden, aber nicht immer auf den ersten Blick ersichtlich sind, können Sie schnell in eine konzeptuelle Sackgasse gelangen, sobald Sie damit beginnen, tiefergehende Arbeitsabläufe oder Modelle zu entwickeln. Verwenden Sie zunächst Dimensionalität als eine einfache Beschreibung des "Zeugs", das Sie modellieren. Die Anzahl der Bemaßungen, die erforderlich sind, um eine Form zu beschreiben, verdeutlicht, wie Geometrie hierarchisch aufgebaut ist.



1. Ein **Punkt** (definiert durch Koordinaten) verfügt über keine Bemaßungen, sondern weist nur Zahlen auf, die die einzelnen Koordinaten beschreiben.
2. Eine **Linie** (definiert durch zwei Punkte) verfügt jetzt über *eine* Bemaßung – Sie können sich vorwärts (in positiver Richtung) oder rückwärts (in negativer Richtung) entlang der Linie bewegen.
3. Eine **Ebene** (definiert durch zwei Linien) verfügt über *zwei* Bemaßungen – Sie können sich jetzt weiter nach links oder nach rechts bewegen.
4. Ein **Quader** (definiert durch zwei Ebenen) verfügt über *drei* Bemaßungen – Sie können eine Position relativ zu oben oder unten definieren.

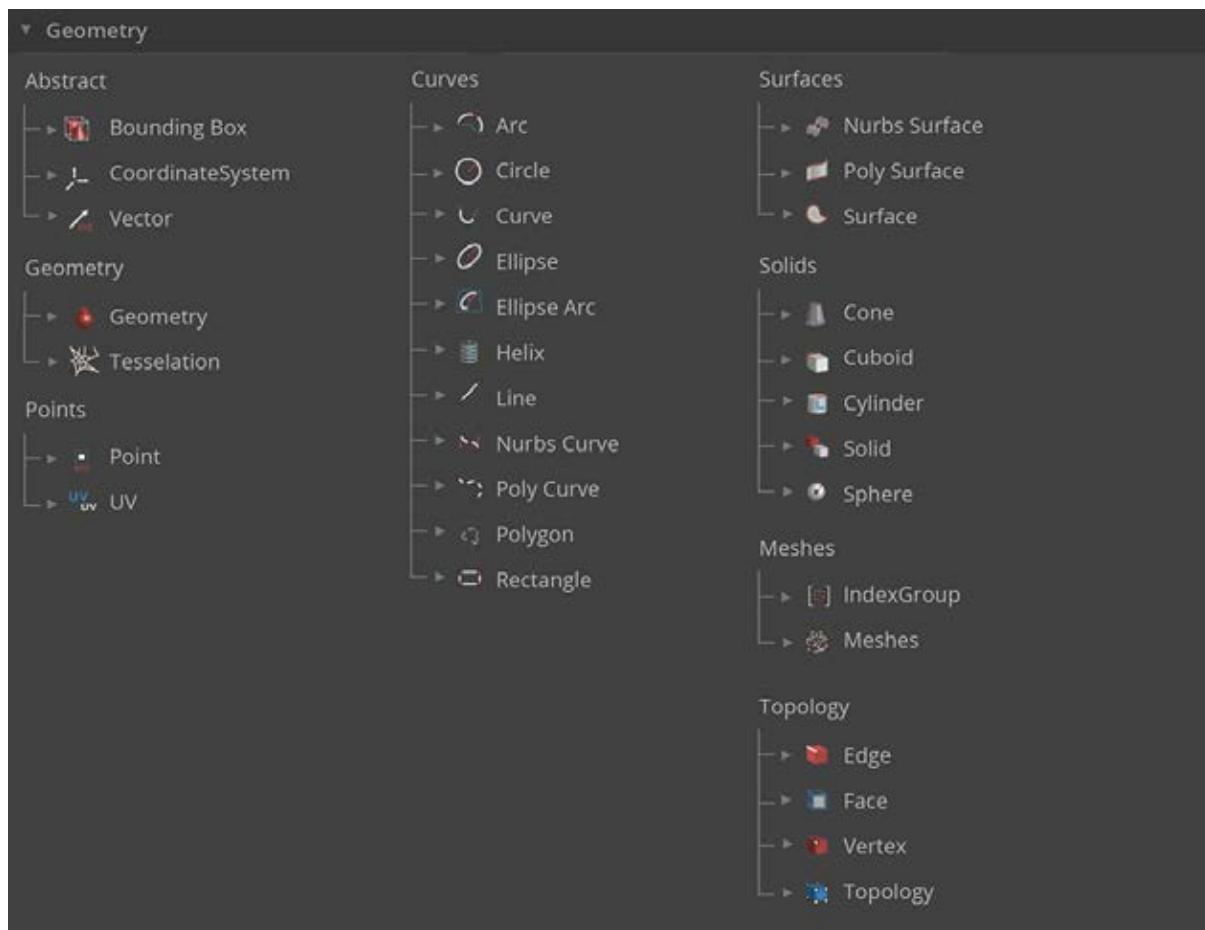
Dimensionalität stellt eine praktische Möglichkeit zum Kategorisieren von Geometrie dar, jedoch nicht unbedingt die beste. Schließlich verwenden wir zum Modellieren nicht nur Punkte, Linien, Ebenen und Quader, sondern auch mal etwas Gekrümmtes? Darüber hinaus gibt es eine vollkommen andere Kategorie der geometrischen Typen, die vollständig abstrakt sind, d. h. die Eigenschaften wie Ausrichtung, Volumen und Beziehungen zwischen Bauteilen definieren. Ein Vektor ist nicht wirklich greifbar. Wie kann er also relativ zu dem definiert werden, was im Raum angezeigt wird? Eine detailliertere Kategorisierung der geometrischen Hierarchie sollte den Unterschied zwischen abstrakten Typen und "Helfern" berücksichtigen, die jeweils danach gruppiert werden können, welche Schritte sie unterstützen, und nach den Typen, die die Beschreibung der Form von Modellelementen unterstützen.

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"> <li>• Vector</li> <li>• Plane</li> <li>• Coordinate System</li> </ul>	<ul style="list-style-type: none"> <li>• Bounding Box</li> </ul>	<ul style="list-style-type: none"> <li>• Vertex</li> <li>• Edge</li> <li>• Face</li> </ul>	<ul style="list-style-type: none"> <li>• XYZ Coordinate</li> <li>• UV Coordinate</li> </ul>	<ul style="list-style-type: none"> <li>• Line</li> <li>• Polygon</li> <li>• Arc</li> <li>• Circle</li> <li>• Ellipse</li> <li>• NURBS Curve</li> <li>• PolyCurve</li> </ul>	<ul style="list-style-type: none"> <li>• NURBS Surface</li> <li>• Polysurface</li> </ul>	<ul style="list-style-type: none"> <li>• Cuboid</li> <li>• Sphere</li> <li>• Cone</li> <li>• Cylinder</li> </ul>	<ul style="list-style-type: none"> <li>• Mesh</li> </ul>

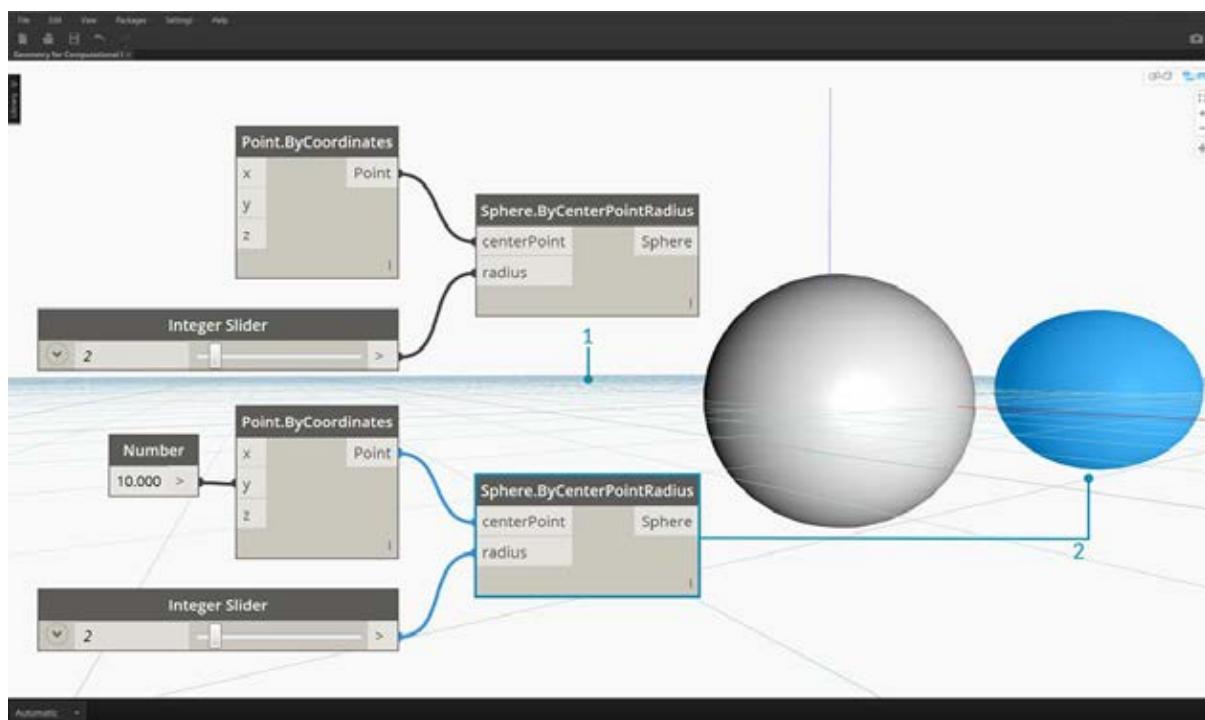
## Geometrie in Dynamo Studio

Was bedeutet dies für die Verwendung von Dynamo? Durch das Verstehen der Geometrietypen und der Beziehungen, die sie zueinander aufweisen, können Sie leichter durch die Sammlung der **Geometrieknoten** navigieren, die in der Bibliothek für Sie verfügbar sind. Die Geometrieknoten sind in alphabetischer Reihenfolge im Gegensatz zu hierarchischen angeordnet. Sie

werden hier also ähnlich wie in ihrem Layout in der Dynamo Benutzeroberfläche angezeigt.



Darüber hinaus sollte das Erstellen von Modellen in Dynamo und das Verbindung der Vorschau in der Hintergrundvorschau mit dem Datenstrom in unserem Diagramm im Laufe der Zeit intuitiver werden.



1. Beachten Sie das angenommene Koordinatensystem, das durch das Raster und die farbigen Achsen dargestellt

wird.

2. Die ausgewählten Knoten rendern die entsprechende Geometrie (wenn der Knoten Geometrie erstellt) im Hintergrund in der Hervorhebungsfarbe. Laden Sie die Beispieldatei für dieses Bild herunter (durch Rechtsklicken und Wahl von "Save Link As..."): [Geometry for Computational Design - Geometry Overview.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

## Weitere Informationen zu Geometrie

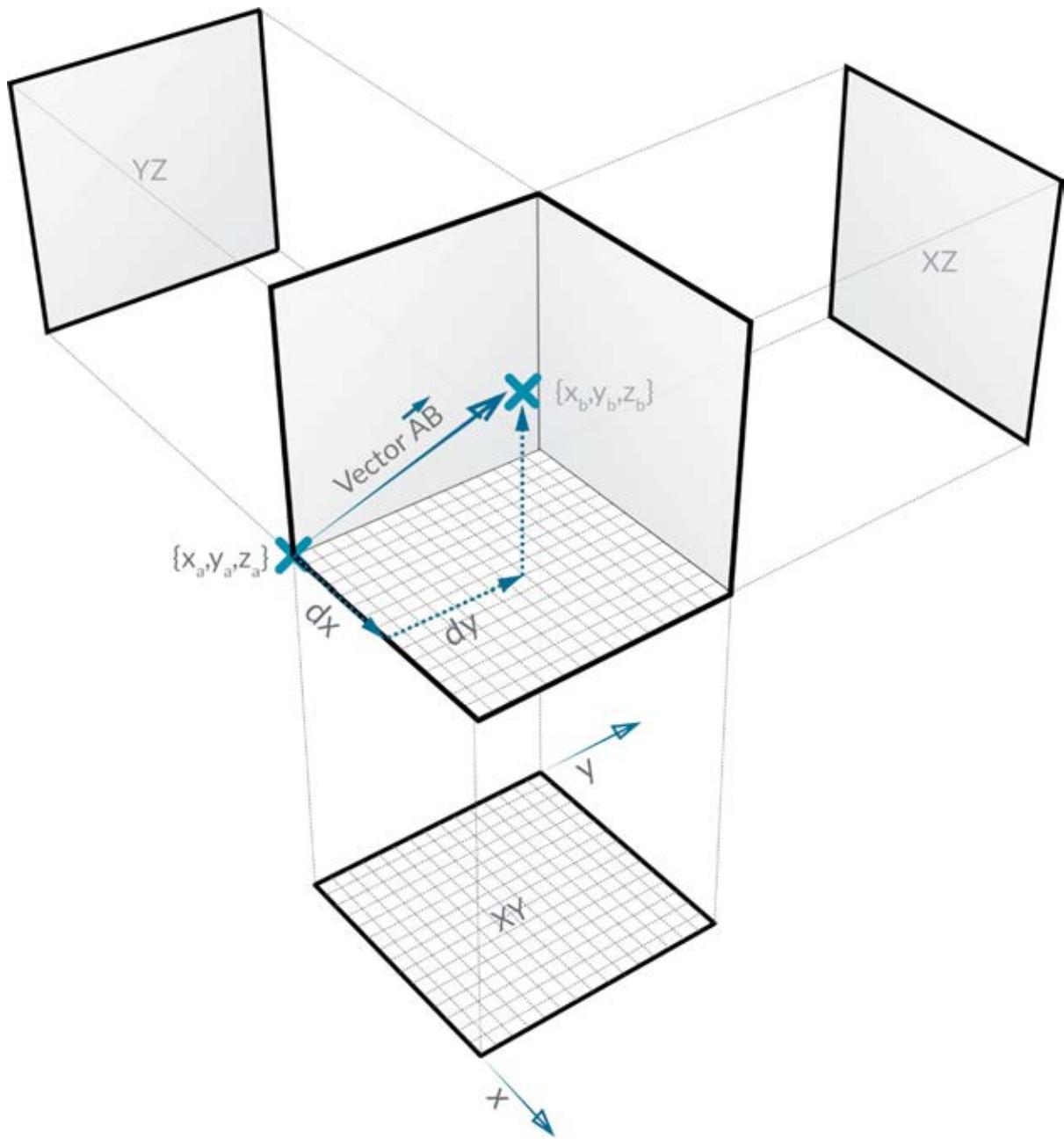
Das Erstellen von Modellen in Dynamo ist nicht darauf beschränkt, was mit Knoten generiert werden kann. Im Folgenden sind einige wichtige Möglichkeiten aufgeführt, wie Sie Ihren Prozess mit Geometrie auf die nächste Stufe stellen können:

1. Dynamo ermöglicht das Importieren von Dateien. Versuchen Sie, CSV-Dateien für Punktwolken zu verwenden, oder SAT-Dateien für das Einbeziehen von Flächen.
2. Bei Verwendung von Revit können Revit-Elemente für die Verwendung in Dynamo referenziert werden.
3. Der Dynamo Package Manager bietet zusätzliche Funktionen wie das [Mesh Toolkit](#) für erweiterte Geometrietypen und Vorgänge.

# **Vektoren, Ebenen und Koordinatensysteme**

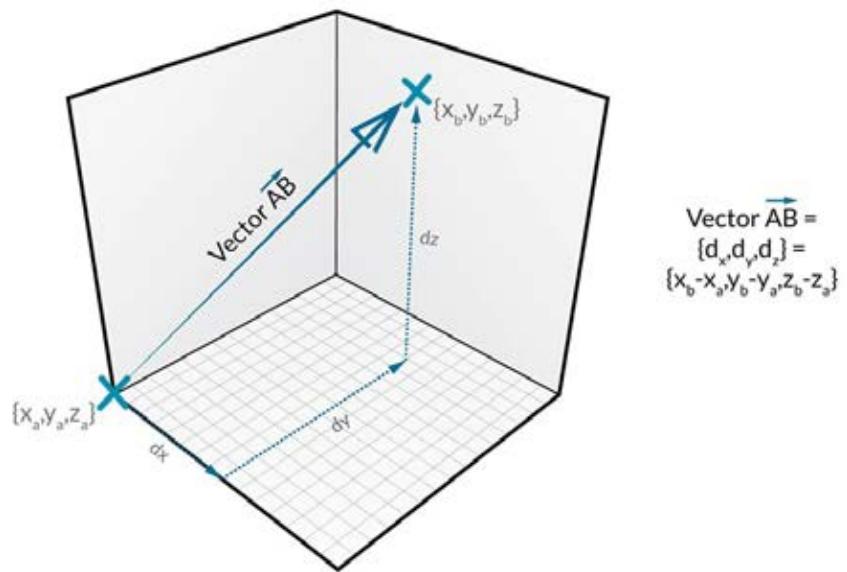
## **Vektoren, Ebenen und Koordinatensysteme**

Vektoren, Ebenen und Koordinatensysteme bilden die primäre Gruppe der abstrakten Geometrietypen. Sie helfen uns dabei, die Position und Ausrichtung sowie den räumlichen Kontext für andere Geometrien zu definieren, die Formen beschreiben. Wenn Sie sagen, dass Sie sich in New York City an der Kreuzung zwischen der 42nd Street und dem Broadway (Koordinatensystem) auf Straßenniveau (Ebene) befinden und nach Norden (Vektor) blicken, habe Sie gerade diese "Helfer" verwendet, um zu definieren, wo Sie stehen. Dasselbe gilt für das Gehäuse eines Telefons oder einen Wolkenkratzer – Sie benötigen diesen Kontext für die Entwicklung eines Modells.



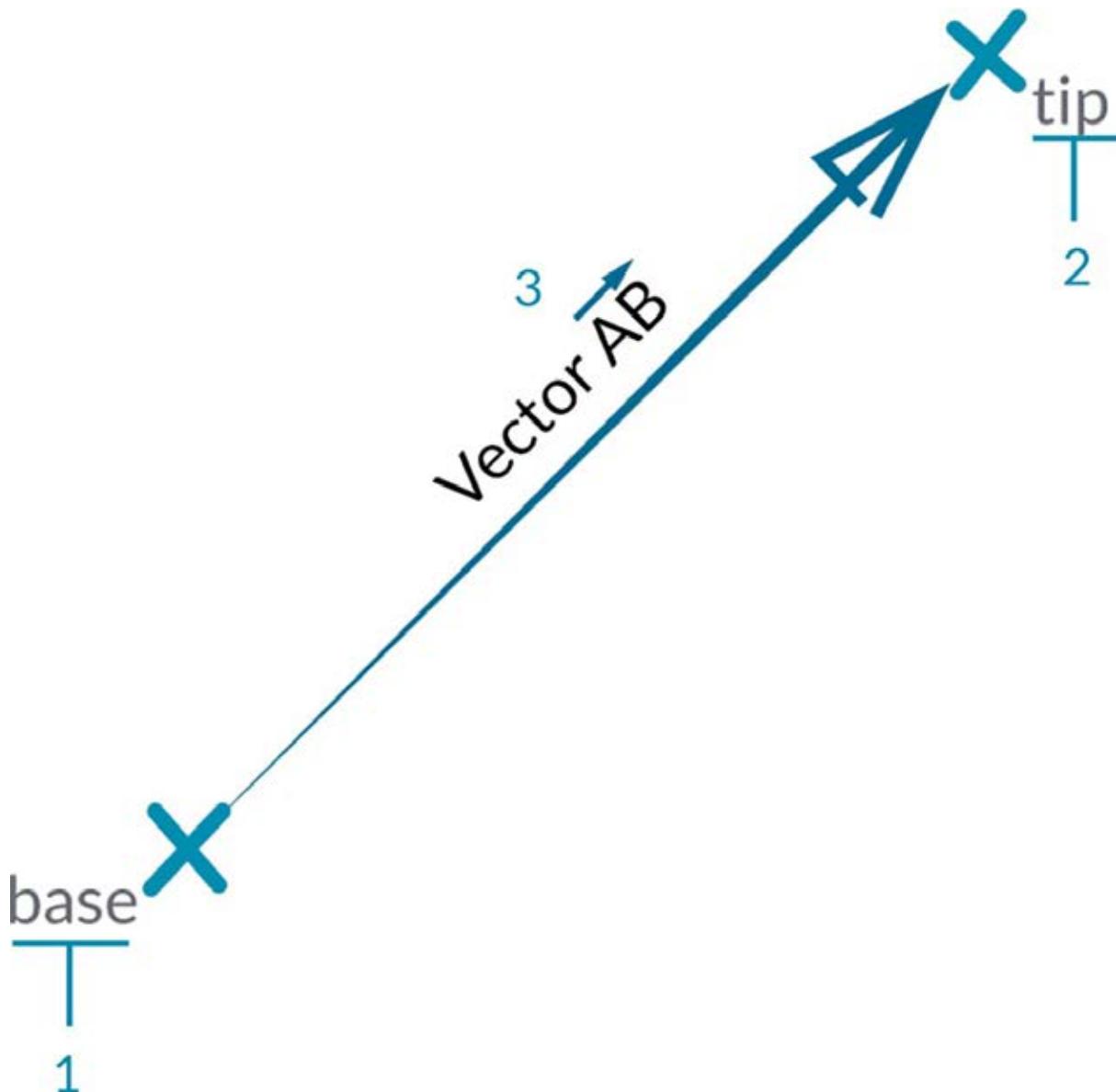
### Was ist ein Vektor?

Ein Vektor ist eine geometrische Größe, die die Richtung und den Betrag beschreibt. Vektoren sind abstrakt, d. h., dass Sie eine Größe darstellen, kein geometrisches Element. Vektoren können leicht mit Punkten verwechselt werden, da beide aus Wertelisten bestehen. Es gibt jedoch einen wesentlichen Unterschied: Punkte beschreiben eine Position in einem bestimmten Koordinatensystem, während Vektoren einen relativen Positionsunterschied beschreiben, also die "Richtung".



Wenn Ihnen die Idee des relativen Unterschieds verwirrend erscheint, stellen Sie sich einen Vektor AB folgendermaßen vor:  
 Sie stehen an Punkt A und sehen zu Punkt B. Die Richtung von A zu B entspricht Ihrem Vektor.

Aufgliedern von Vektoren in Ihre Bestandteile mit derselben AB-Notation:

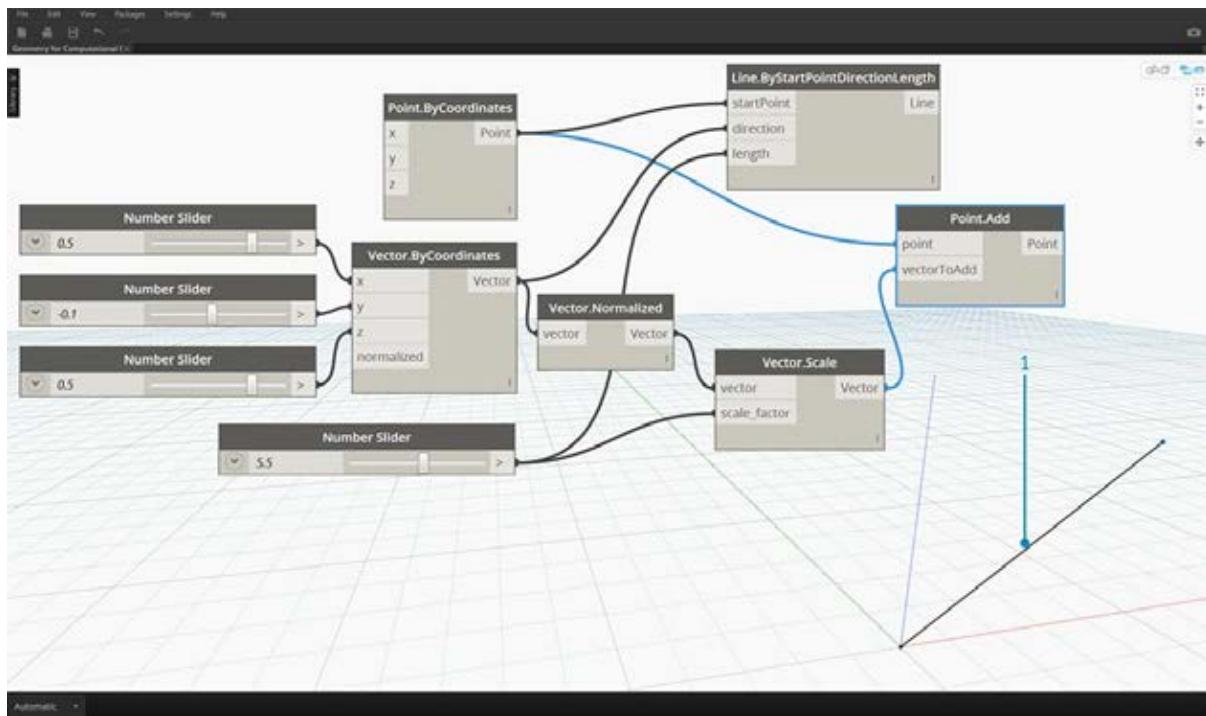


1. Der **Startpunkt** von Vektoren wird als **Basis** bezeichnet.
2. Der **Endpunkt** von Vektoren wird als **Spitze** oder **Ausrichtung** bezeichnet.
3. Vektor  $AB$  entspricht nicht Vektor  $BA$ , der in die entgegengesetzte Richtung weist.

Wenn Sie in Bezug auf Vektoren (und ihrer abstrakten Definition) jemals einer komischen Entlastung bedürfen, sehen Sie sich die klassische Komödie "Die unglaubliche Reise in einem verrückten Flugzeug" an und hören auf die häufig zitierte, humorvolle Aussage:

*Roger, Roger. Was ist unser Vektor, Viktor?*

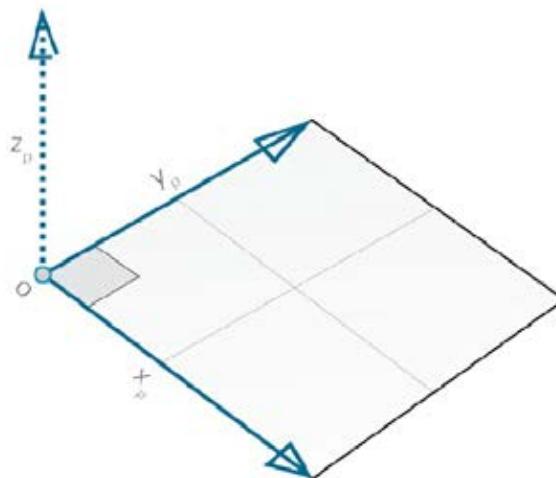
Vektoren stellen eine wichtige Komponente für Modelle in Dynamo dar. Beachten Sie, dass Sie zur abstrakten Kategorie der "Helfer" gehören. Wenn Sie also einen Vektor erstellen, wird nichts in der Hintergrundvorschau angezeigt.



1. Sie können eine Linie zur Darstellung einer Vektorvorschau verwenden. Laden Sie die Beispieldatei für dieses Bild herunter (durch Rechtsklicken und Wahl von "Save Link As..."): [Geometry for Computational Design - Vectors.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

### Was ist eine Ebene?

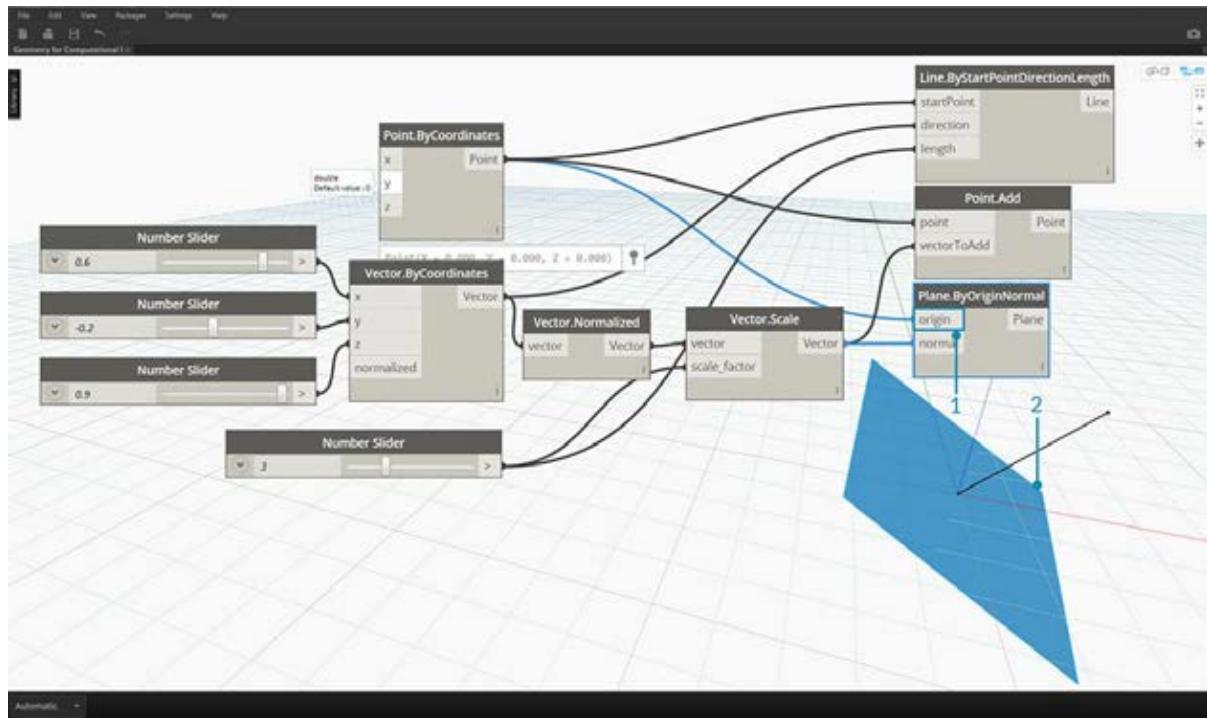
Ebenen sind zweidimensionale abstrakte "Helfer". Genauer gesagt sind Ebenen konzeptuell gesehen "flach" und erstrecken sich unendlich in zwei Richtungen. In der Regel werden sie als ein kleineres Rechteck in der Nähe ihres Ursprungs gerendert.



Sie denken möglicherweise: "Stopp! Ursprung? Das klingt nach einem Koordinatensystem, das ich auch zum Modellieren in meiner CAD-Software verwende!"

Und Sie haben recht! In Modellierungssoftware werden häufig Konstruktionsebenen verwendet, um einen lokalen, zweidimensionalen Kontext zu definieren, in dem Entwürfe erstellt werden können. XY-, XZ-, YZ- bzw. Nord- oder

Südostebene klingen möglicherweise vertrauter. Dies sind alles Ebenen, die einen unendlichen "flachen" Kontext definieren. Ebenen weisen keine Tiefe auf, unterstützen uns jedoch bei der Beschreibung einer Richtung – jede Ebene weist einen Ursprung, eine X-Richtung, eine Y-Richtung und eine Z-Richtung (nach oben) auf.

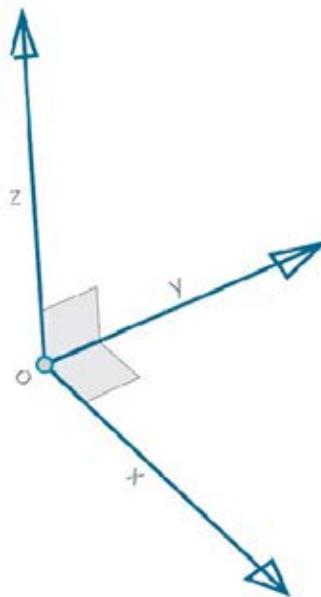


1. Ebenen sind zwar abstrakt, verfügen aber über eine Ursprungsposition, damit sie im Raum lokalisiert werden können.
2. In Dynamo werden Ebenen in der Hintergrundvorschau gerendert. Laden Sie die Beispieldatei für dieses Bild herunter (durch Rechtsklicken und Wahl von "Save Link As..."): [Geometry for Computational Design - Planes.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

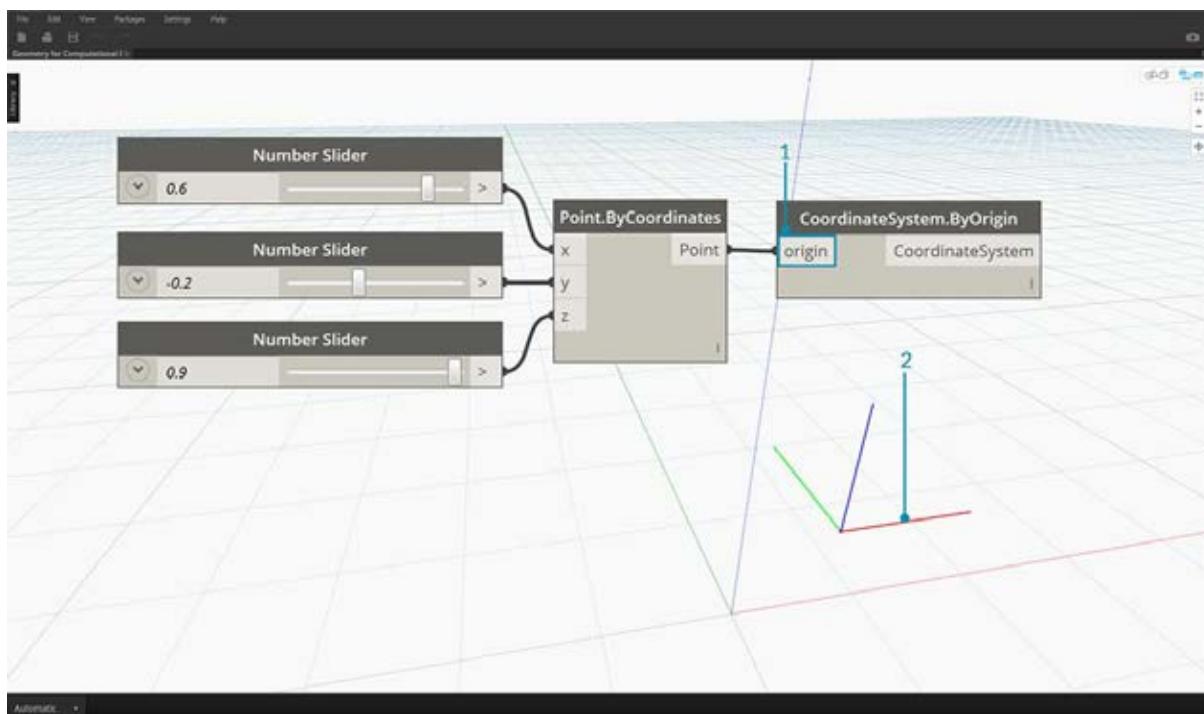
## Was ist ein Koordinatensystem?

Sobald Sie mit Ebenen vertraut sind, ist es nur noch ein kleiner Schritt hin zu Koordinatensystemen. Eine Ebene weist dieselben Bestandteile wie ein Koordinatensystem auf, solange es sich um ein "euklidisches" oder "XYZ"-Koordinatensystem handelt.

Darüber hinaus gibt es jedoch auch alternative Koordinatensysteme wie Zylinder- oder Kugelkoordinatensysteme. Wie Sie in späteren Abschnitten sehen werden, können Koordinatensysteme auch auf andere Geometrietypen angewendet werden, um eine Position in der Geometrie zu definieren.



Hinzufügen alternativer Koordinatensysteme – Zylinder- oder Kugelkoordinatensystem

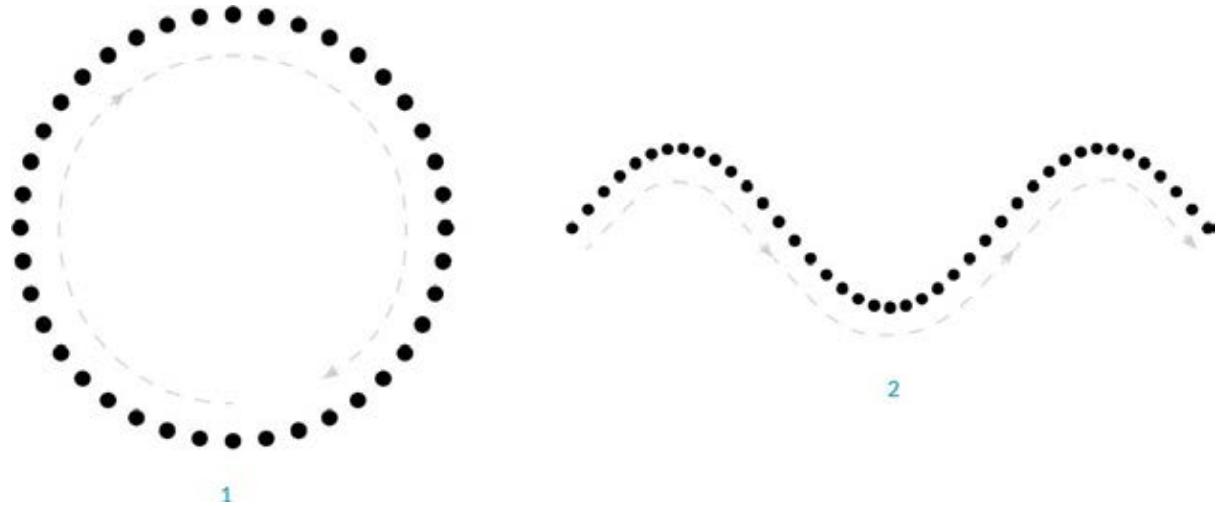


1. Koordinatensysteme sind zwar abstrakt, verfügen aber über eine Ursprungsposition, damit sie im Raum lokalisiert werden können.
2. In Dynamo werden Koordinatensysteme in der Hintergrundvorschau als Punkt (Ursprung) und Linien gerendert, die die Achsen definieren (gemäß folgender Konvention: X ist rot, Y ist grün und Z ist blau). Laden Sie die Beispieldatei für dieses Bild herunter (durch Rechtsklicken und Wahl von "Save Link As..."): [Geometry for Computational Design - Coordinate System.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

# Punkte

## Punkte

Wenn Geometrie gewissermaßen die Sprache für ein Modell ist, sind Punkte das Alphabet. Punkte sind die Grundlage für die Erstellung aller anderen Geometrie: Sie benötigen mindestens zwei Punkte, um eine Kurve zu erstellen, mindestens drei Punkte für ein Polygon oder eine Netzfläche usw. Indem Sie die Position, Anordnung und Beziehung zwischen Punkten angeben (z. B. mithilfe einer Sinusfunktion), können Sie Geometrie höherer Ordnung definieren, die etwa als Kreise oder Kurven zu erkennen ist.



1. Kreis mit den Funktionen  $x=r\cos(t)$  und  $y=r\sin(t)$
2. Sinuskurve mit den Funktionen  $x=(t)$  und  $y=r\sin(t)$

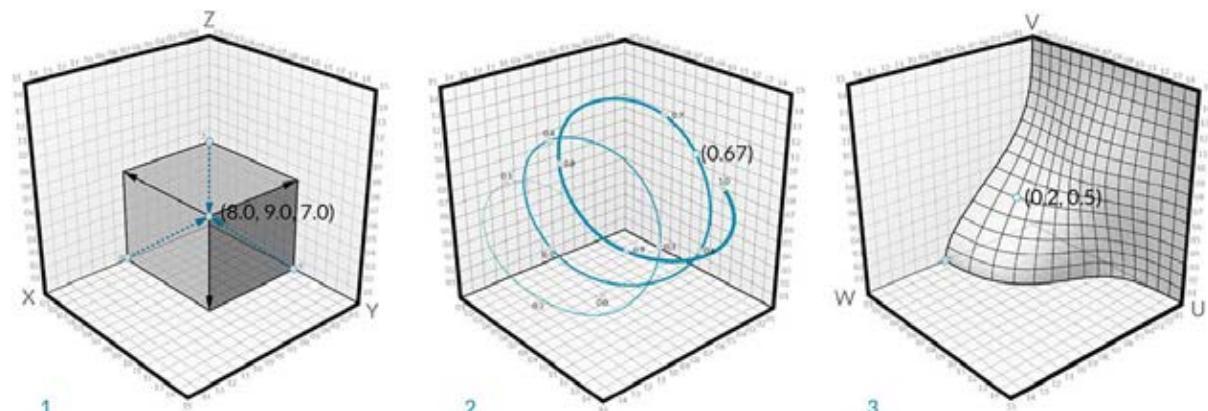
## Was ist ein Punkt?

Ein Punkt wird lediglich durch einen oder mehrere Werte, die sogenannten Koordinaten, definiert. Die Anzahl der Koordinatenwerte, die zum Definieren des Punkts benötigt werden, ist vom Koordinatensystem oder Kontext abhängig, in dem er sich befindet. In Dynamo werden größtenteils Punkte verwendet, die sich im dreidimensionalen Weltkoordinatensystem befinden und drei Koordinaten [x,y,z] aufweisen.



## Punkt als Koordinaten

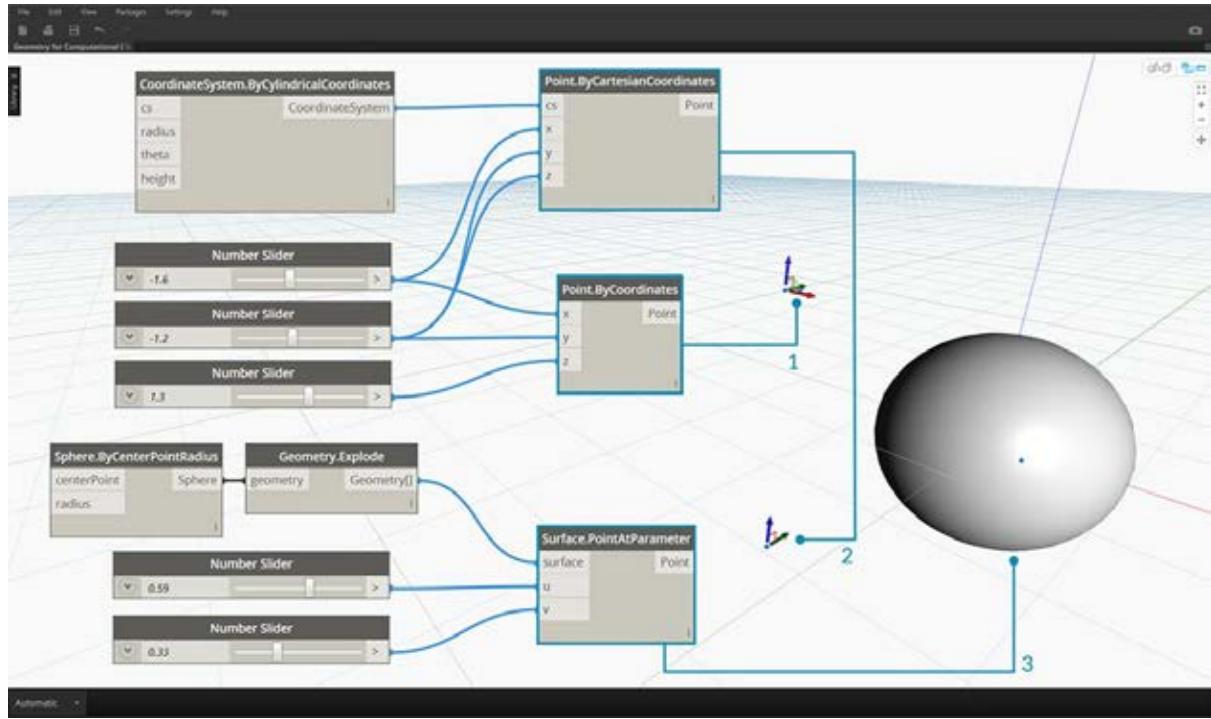
Punkte können auch in zweidimensionalen Koordinatensystemen vorhanden sein. Für unterschiedliche Räume bestehen unterschiedliche Notationskonventionen: So wird etwa bei einer Ebene [X,Y], bei einer Oberfläche jedoch [U,V] verwendet.



1. Punkt in euklidischen Koordinatensystem: [x,y,z]
2. Punkt in einem Koordinatensystem mit Kurvenparametern: [t]

### 3. Punkt in Koordinatensystem mit Oberflächenparametern: [U,V]

Parameter für Kurven und Oberflächen sind – auch wenn dies nicht intuitiv verständlich wirkt – kontinuierlich und erstrecken sich über die Grenzen der gegebenen Geometrie hinaus. Da die Formen, die den Parameterraum definieren, sich im dreidimensionalen Weltkoordinatensystem befinden, können parametrische Koordinaten jederzeit in Weltkoordinaten konvertiert werden. Der Punkt [0.2, 0.5] auf der Oberfläche entspricht beispielsweise dem Punkt [1.8, 2.0, 4.1] in Weltkoordinaten.

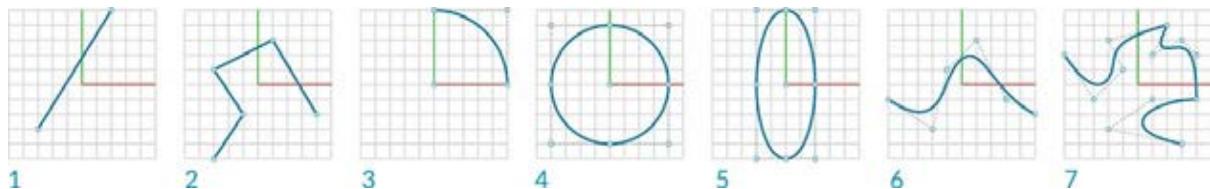


1. Punkt in angenommenen Weltkoordinaten (xyz)
2. Punkt relativ zu einem angegebenen Koordinatensystem (zylindrisch)
3. Punkt in UV-Koordinaten auf einer Oberfläche Laden Sie die Beispieldatei für dieses Bild herunter (durch Rechtsklicken und Wahl von "Save Link As..."): [Geometry for Computational Design - Points.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

# Kurven

## Kurven

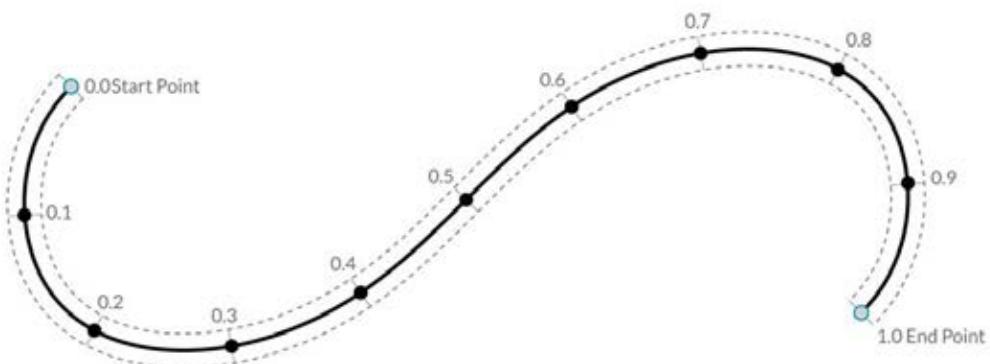
Kurven sind der erste geometrische Datentyp, den wir abgedeckt haben, die einen vertrauteren Satz an beschreibenden Eigenschaften für die Form aufweisen: Wie kurvig oder gerade? Wie lange oder kurz? Und denken Sie daran, dass Punkte weiterhin unsere Bausteine für die Definition von allem darstellen, von einer Linie zu einem Spline und zu allen Kurvenarten dazwischen.



1. Linie
2. Polylinie
3. Bogen
4. Kreis
5. Ellipse
6. NURBS-Kurve
7. Polykurve

## Was ist eine Kurve?

Der Begriff **Kurve** ist im Allgemeinen ein Oberbegriff für all die unterschiedlichen Typen von gekrümmten (auch geraden) Formen. "C"-Kapitalkurve ist die übergeordnete Kategorisierung für all diese Formen – Linien, Kreise, Splines usw. Technisch betrachtet beschreibt eine Kurve jeden möglichen Punkt, der durch die Eingabe von "t" in eine Sammlung von Funktionen gefunden werden kann, von einfachen Funktionen wie  $x = -1.26*t$ ,  $y = t$  bis hin zu Funktionen mit Calculi. Unabhängig davon, mit welcher Art von Kurve Sie arbeiten, handelt es sich bei dem **Parameter** mit dem Namen "t" um eine Eigenschaft, die bewertet werden kann. Darüber hinaus haben alle Kurven unabhängig vom Aussehen der Form einen Startpunkt und einen Endpunkt, die zufälligerweise mit dem Mindest- und dem Höchstwert von t zusammenfallen, die zum Erzeugen der Kurve verwendet wurden. Dies hilft Ihnen auch dabei, ihre Direktionalität zu verstehen.

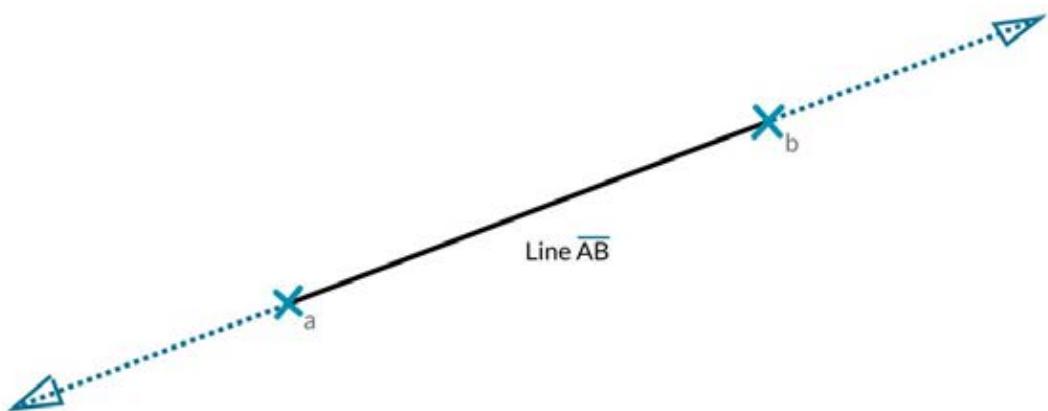


Es ist wichtig zu beachten, dass Dynamo davon ausgeht, dass die Domäne der Werte "t" für eine Kurve 0.0 bis 1.0 beträgt.

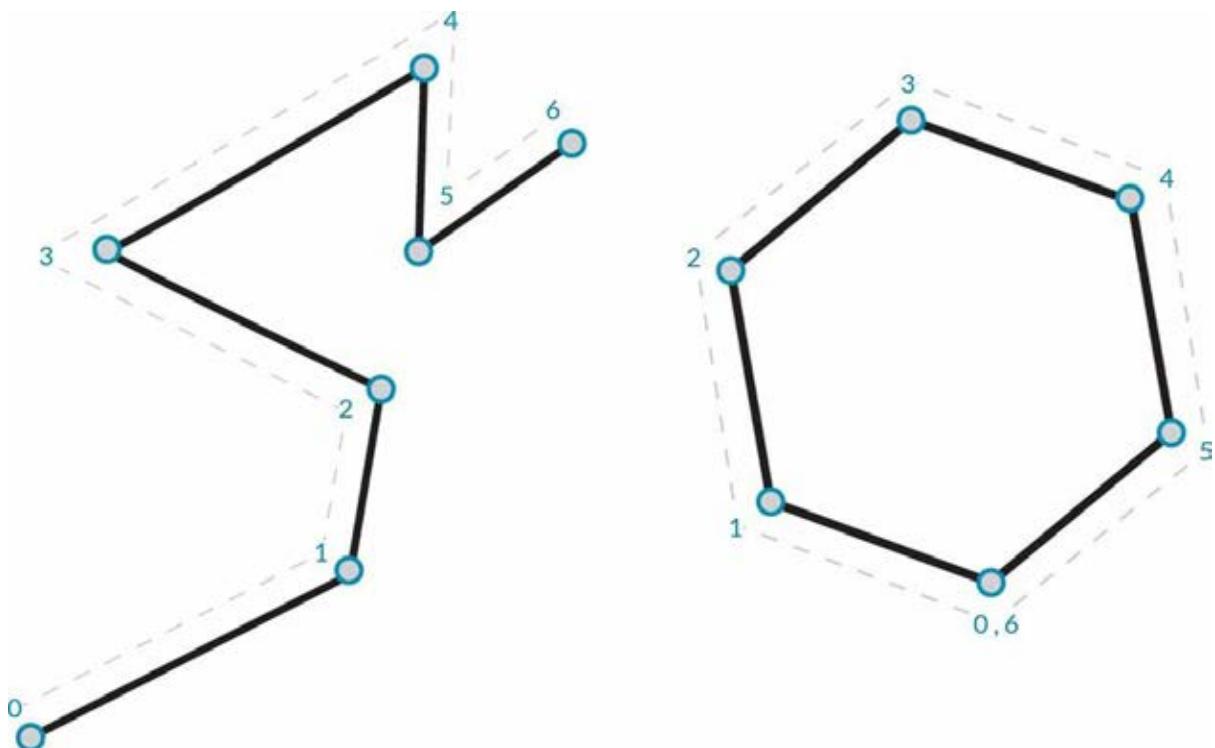
Alle Kurven besitzen auch eine Reihe von Eigenschaften oder Merkmalen, die verwendet werden können, um sie zu beschreiben oder zu analysieren. Wenn der Abstand zwischen dem Start- und dem Endpunkt null beträgt, ist die Kurve "geschlossen". Außerdem weist jede Kurve eine Reihe von Steuerpunkten auf. Wenn sich diese Punkte alle in derselben Ebene befinden, ist die Kurve "planar". Einige Eigenschaften gelten für die Kurve als Ganzes, während andere nur für bestimmte Punkte auf der Kurve gelten. Die Ebenheit ist beispielsweise eine globale Eigenschaft, während ein tangentialer Vektor an einem bestimmten Wert  $t$  eine lokale Eigenschaft ist.

## Linien

**Linien** sind die einfachste Form von Kurven. Sie sehen möglicherweise nicht gekrümmt aus, sind jedoch tatsächlich Kurven – nur ohne Krümmung. Es gibt mehrere Möglichkeiten zum Erstellen von Linien, die intuitivste davon zwischen den Punkten A und B. Die Form der Linie AB erstreckt sich zwischen den beiden Punkten, mathematisch betrachtet geht sie jedoch in beide Richtungen unendlich weiter.

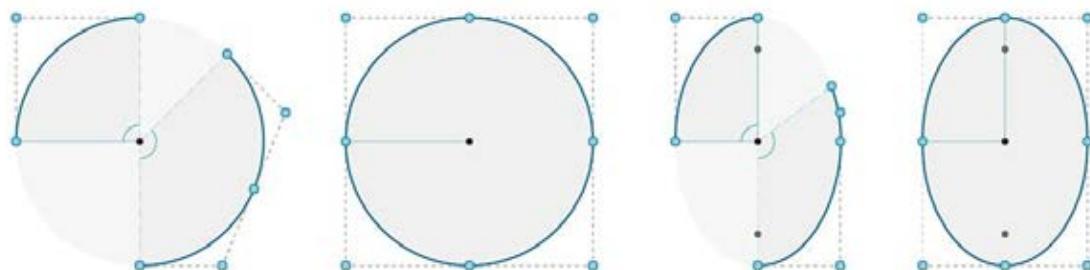


Wenn Sie zwei Linien miteinander verbinden, erhalten Sie eine **Polylinie**. Im Folgenden ist auf einfache Weise dargestellt, was ein Steuerpunkt ist. Durch das Bearbeiten der Position eines dieser Punkte ändert sich die Form der Polylinie. Wenn die Polylinie geschlossen ist, erhalten Sie ein Polygon. Wenn die Kantenlängen des Polygons gleich sind, wird von einem regelmäßigen Polygon gesprochen.



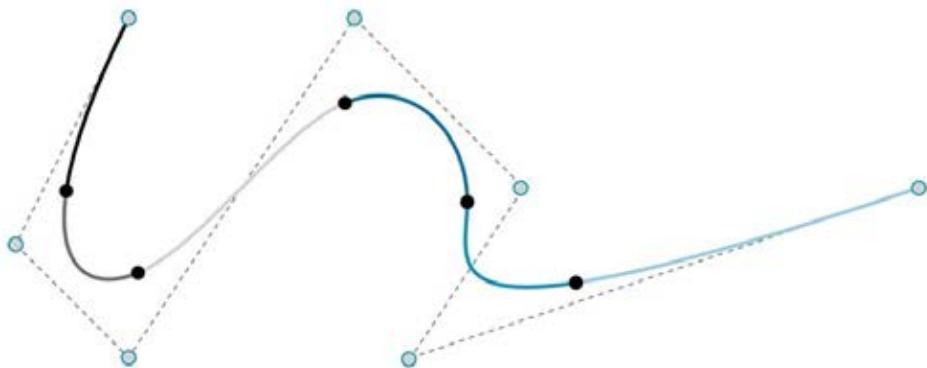
### Bogen, Kreise, Ellipsenbögen und Ellipsen

Um mehr Komplexität zu den parametrischen Funktionen hinzuzufügen, die eine Form definieren, können Sie mit einem weiteren Schritt aus einer Linie einen **Bogen**, **Kreis**, **Ellipsenbogen** oder eine **Ellipse** erstellen, indem Sie einen oder zwei Radien beschreiben. Dabei besteht der Unterschied zwischen der Bogenversion sowie der Kreis- oder Ellipsenversion darin, ob die Form offen oder geschlossen ist.



### NURBS + Polykurven

**NURBS** (nicht-uniforme rationale B-Splines) sind mathematische Darstellungen, mit denen jede beliebige Form von einfachen zweidimensionalen Linien, Kreisen, Bogen oder Rechtecken bis hin zu den komplexesten dreidimensionalen organischen Freiformkurven präzise modelliert werden kann. Aufgrund ihrer Flexibilität (relativ wenige Steuerpunkte, aber eine glatte Interpolation basierend auf Gradeinstellungen) und Genauigkeit (durch eine robuste Mathematik) können NURBS-Modelle in jedem beliebigen Prozess von der Illustration über die Animation bis hin zur Fertigung verwendet werden.

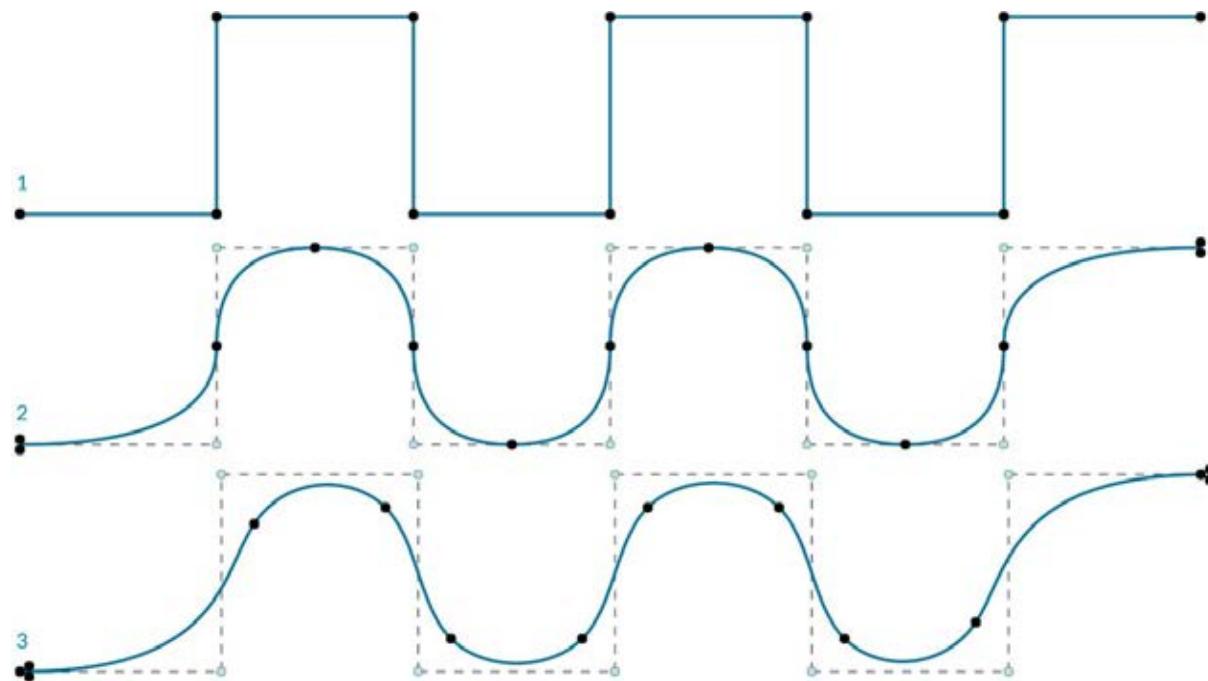


**Grad:** Der Grad einer Kurve bestimmt den Einflussbereich, den die Steuerpunkte auf eine Kurve haben. Dabei gilt: je höher der Grad, desto größer der Bereich. Der Grad ist eine positive ganze Zahl. Er lautet normalerweise 1, 2, 3 oder 5, es kann sich dabei aber um jede beliebige ganze Zahl handeln. NURBS-Linien und -Polylinien weisen normalerweise den Grad 1 und die meisten Freiformkurven den Grad 3 oder 5 auf.

**Steuerpunkte:** Steuerpunkte sind eine Liste mit mindestens einem Grad und einem Punkt. Eine der einfachsten Möglichkeiten zum Ändern der Form einer NURBS-Kurve besteht im Verschieben der Steuerpunkte.

**Gewichtung:** Steuerpunkten ist eine Zahl zugewiesen, die als Gewichtung bezeichnet wird. Gewichtungen werden in der Regel als positive Zahlen angegeben. Wenn die Steuerpunkte einer Kurve alle dieselbe Gewichtung aufweisen (normalerweise 1), wird die Kurve als nichtrational, andernfalls als rational bezeichnet. Die meisten NURBS-Kurven sind nichtrational.

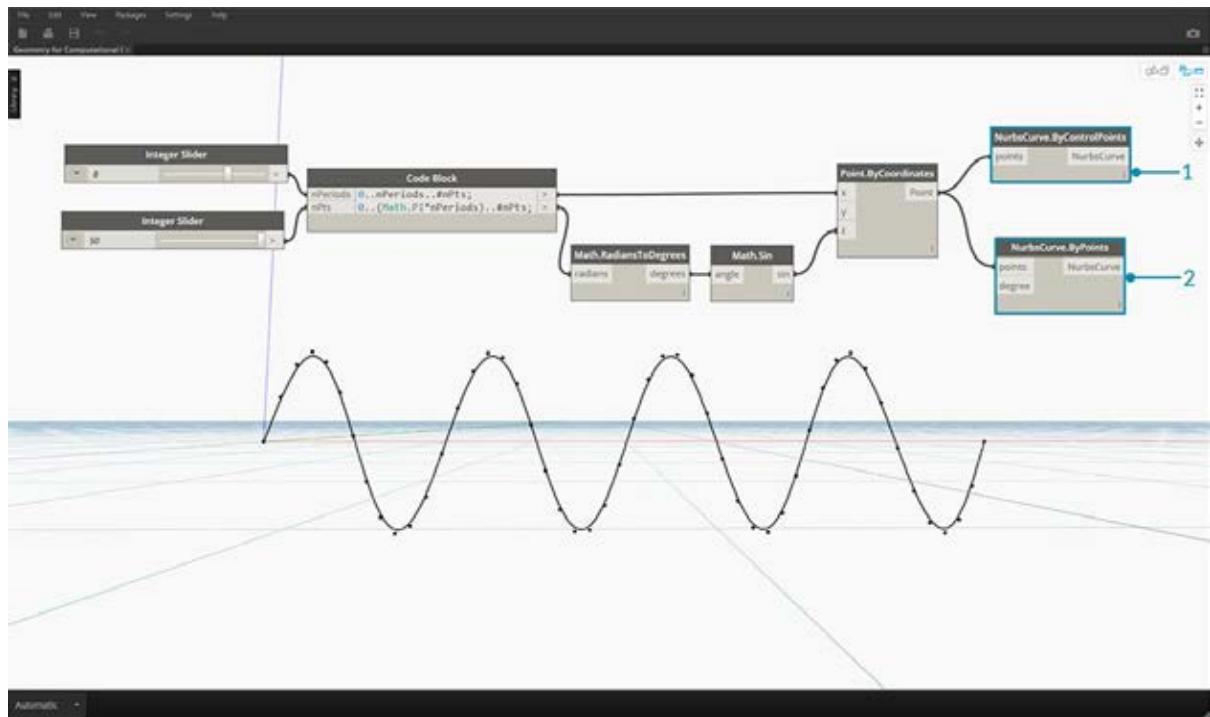
**Knoten:** Knoten sind eine Liste mit  $(\text{Grad}+N-1)$  Zahlen, wobei  $N$  der Anzahl an Steuerpunkten entspricht. Die Knoten werden zusammen mit den Gewichtungen verwendet, um den Einfluss der Steuerpunkte auf die resultierende Kurve zu steuern. Eine Verwendung von Knoten besteht darin, Knicke an bestimmten Punkten in der Kurve zu erzeugen.



1. Grad = 1
2. Grad = 2
3. Grad = 3

Beachten Sie, dass je höher der Gradwert ist, desto mehr Steuerpunkte werden verwendet, um die resultierende Kurve zu interpolieren.

Erstellen Sie in Dynamo eine Sinuskurve mit zwei unterschiedlichen Methoden, um NURBS-Kurven zu erstellen, und vergleichen Sie die Ergebnisse.



1. *NurbsCurve.ByControlPoints* verwendet die Liste der Punkte als Steuerpunkte.
2. *NurbsCurve.ByPoints* zeichnet eine Kurve durch die Liste der Punkte. Laden Sie die Beispieldatei für dieses Bild herunter (durch Rechtsklicken und Wahl von "Save Link As..."): [Geometry for Computational Design - Curves.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

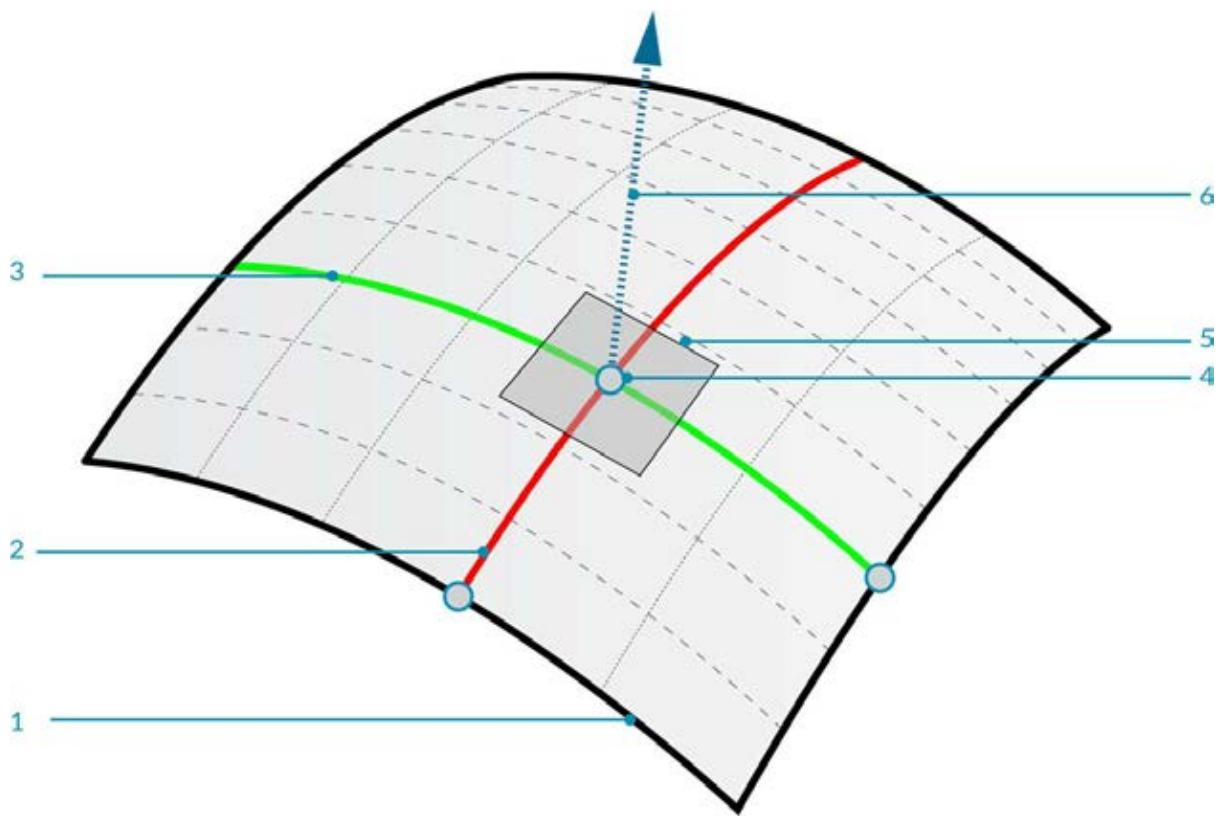
# Oberflächen

## Oberflächen

Mit dem Übergang von Kurven zu Oberflächen in einem Modell beginnen Sie, Objekte darzustellen, die in der dreidimensionalen Realität sichtbar sind. Kurven sind nicht immer planar (d. h., sie sind dreidimensional), der durch sie definierte Raum ist jedoch immer an eine Dimension gebunden. Mit Oberflächen kommen eine weitere Dimension und damit eine Reihe weiterer Eigenschaften hinzu, die Sie in anderen Modellierungsvorgängen nutzen können.

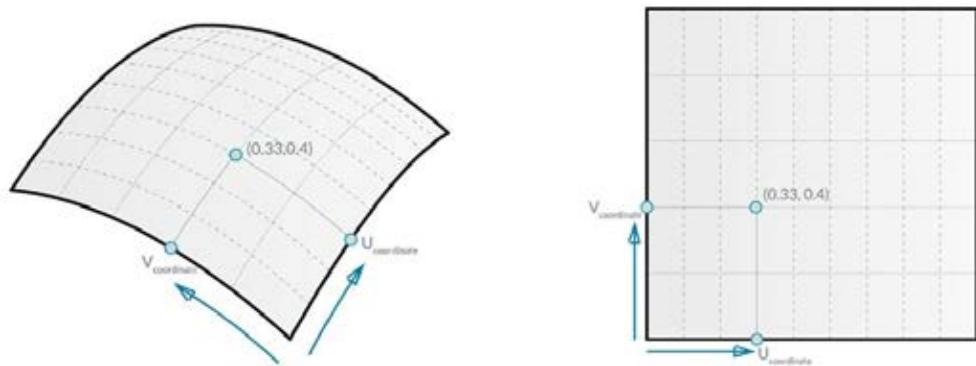
### Was ist eine Oberfläche?

Eine Oberfläche ist eine durch eine Funktion und zwei Parameter definierte mathematische Form. Der entsprechende Parameterraum wird nicht wie bei Kurven durch  $t$ , sondern durch  $U$  und  $V$  beschrieben. Das bedeutet, dass bei der Arbeit mit dieser Art von Geometrie mehr geometrische Daten genutzt werden können. So sind z. B. bei Kurven Tangentenvektoren und Normalenebenen (die über die Länge der Kurve hinweg gedreht werden können), bei Oberflächen hingegen Normalenvektoren und Tangentialebenen vorhanden, deren Ausrichtung unverändert bleibt.



1. Oberfläche
2. U-Isokurve
3. V-Isokurve
4. UV-Koordinaten
5. Senkrechte Ebene
6. Normalenvektor

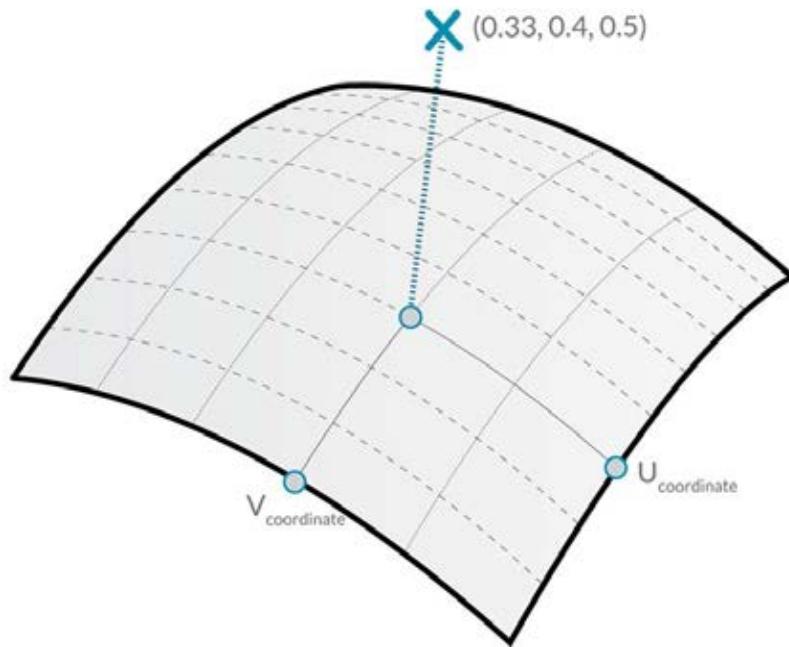
**Oberflächendomäne:** Die Domäne einer Oberfläche ist als der Bereich von UV-Parametern definiert, die als dreidimensionale Punkte auf der Oberfläche ausgewertet werden können. Die Domäne für jede der Dimensionen ( $U$  oder  $V$ ) wird normalerweise in Form zweier Zahlen ( $U \text{ Min}$  bis  $U \text{ Max}$ ) und ( $V \text{ Min}$  bis  $V \text{ Max}$ ) beschrieben.



Die Kontur der Oberfläche ist dem Augenschein nach nicht unbedingt "rechteckig" und das Netz der Isokurven kann lokal eng- oder weitmaschiger sein; der durch die Domäne definierte "Raum" ist jedoch immer zweidimensional. In Dynamo wird immer angenommen, dass die Domäne einer Oberfläche durch den Mindestwert 0.0 und den Höchstwert 1.0 sowohl in U- als auch in V-Richtung definiert ist. Bei planaren oder gestützten Oberflächen sind andere Domänen möglich.

**Isokurve** (oder isoparametrische Kurve): Eine durch einen konstanten U- oder V-Wert auf der Oberfläche und eine Domäne für die Werte in der dazugehörigen U- bzw. V-Richtung definierte Kurve.

**UV-Koordinaten:** Punkt im UV-Parameterraum, definiert durch U, V und manchmal W.

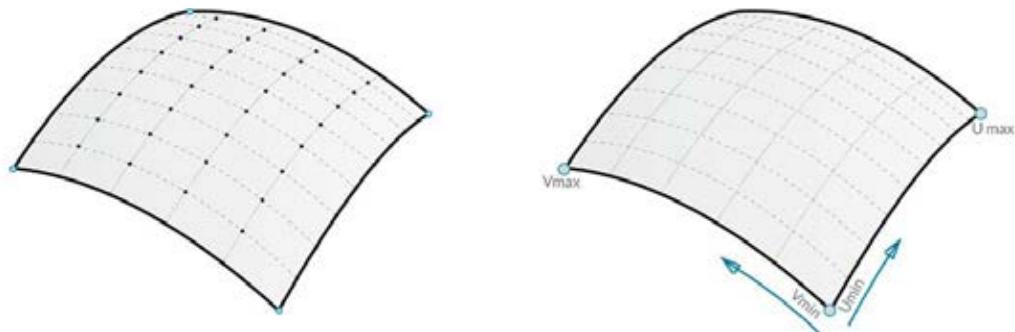


**Senkrechte Ebene:** Ebene, die an einer gegebenen UV-Koordinatenposition sowohl zur U- als auch zur V-Isokurve senkrecht steht.

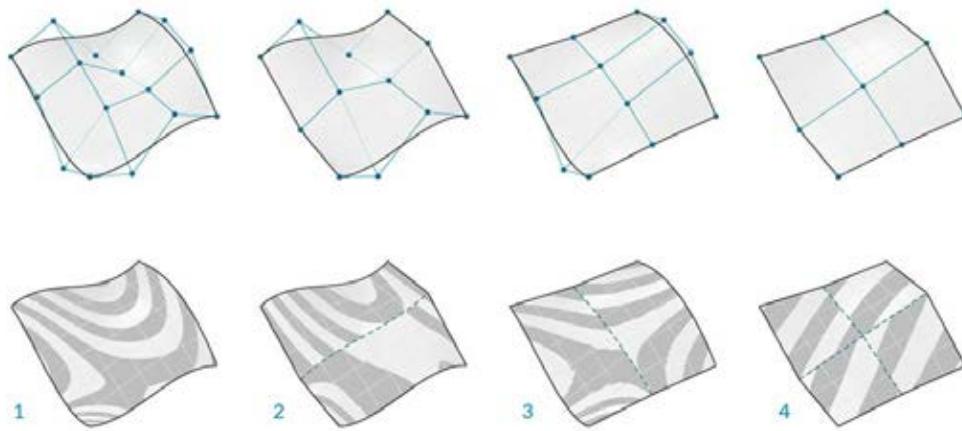
**Normalenvektor:** Vektor, der die Aufwärtsrichtung relativ zur senkrechten Ebene definiert.

## NURBS-Oberflächen

**NURBS-Oberflächen** sind NURBS-Kurven sehr ähnlich. NURBS-Oberflächen sind vorstellbar als aus NURBS-Kurven gebildete Raster mit zwei Richtungen. Die Form einer NURBS-Oberfläche wird durch eine Reihe von Steuerpunkten und den Grad der Oberfläche in U- und V-Richtung definiert. Dieselben Algorithmen zur Berechnung von Form, Normalen, Tangenten, Krümmungen und anderer Eigenschaften mithilfe von Steuerpunkten, Gewichtungen und Grad kommen auch hier zum Einsatz.



Bei NURBS-Oberflächen werden zwei Richtungen für die Geometrie angenommen, da diese Oberflächen ungeachtet der sichtbaren Form rechtwinklige Raster von Steuerpunkten sind. Diese Richtungen liegen relativ zum Weltkoordinatensystem oft beliebig. Sie werden dennoch häufig zur Analyse von Modellen oder zum Generieren weiterer Geometrie auf Basis der Oberfläche verwendet.

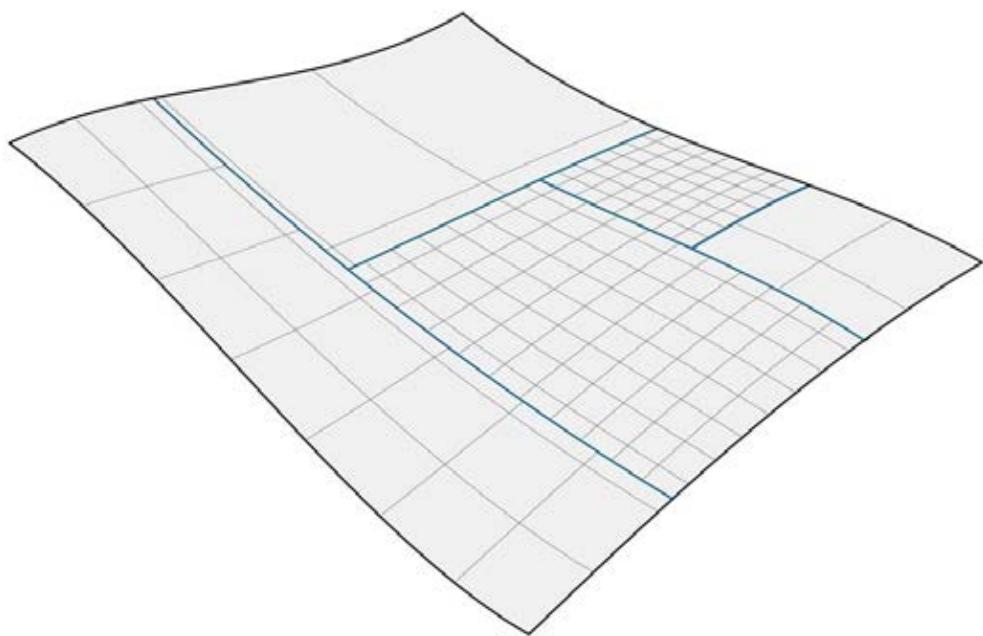


1.  $\text{Grad } (U,V) = (3,3)$
2.  $\text{Grad } (U,V) = (3,1)$
3.  $\text{Grad } (U,V) = (1,2)$
4.  $\text{Grad } (U,V) = (1,1)$

## PolySurfaces

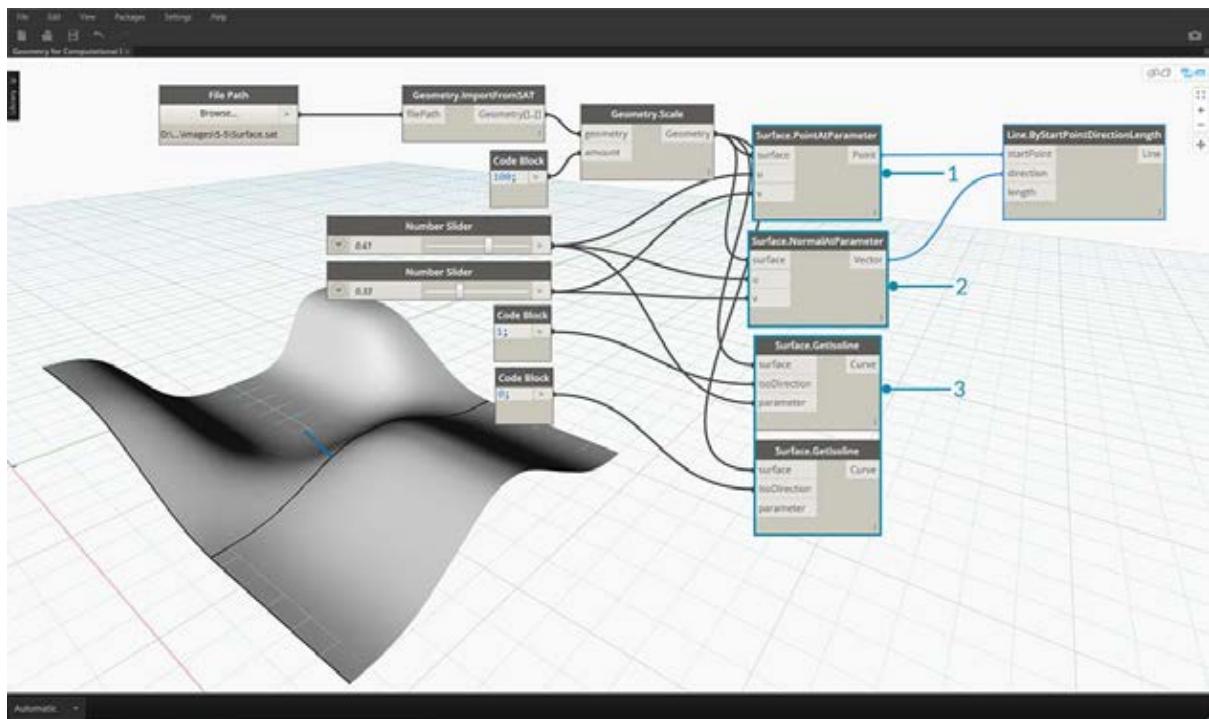
**PolySurfaces** setzen sich aus an einer Kante verbundenen Oberflächen zusammen. PolySurfaces bieten mehr als zweidimensionale UV-Definitionen: Sie können sich jetzt anhand der Topologie durch die verbundenen Formen bewegen.

"Topologie" beschreibt in der Regel die Verbindungen und Beziehungen zwischen Teilen. In Dynamo ist Topologie darüber hinaus auch ein Typ von Geometrie. Sie ist, genauer, die übergeordnete Kategorie für Oberflächen, PolySurfaces und Körper.



Durch Zusammenfügen von Oberflächen (manchmal als "Pflasterung" bezeichnet) können komplexere Formen erstellt und Details entlang der Naht definiert werden. Beispielsweise können Sie die Kanten einer PolySurface mit Abrundungen oder Fasen versehen.

Importieren Sie eine Oberfläche in Dynamo und werten Sie sie an einer Parameterposition aus, um zu sehen, welche Informationen Sie extrahieren können.



1. *Surface.PointAtParameter* gibt den Punkt an der angegebenen UV-Koordinatenposition zurück.
2. *Surface.NormalAtParameter* gibt den Normalenvektor an der angegebenen UV-Koordinatenposition zurück.
3. *Surface.GetIsoline* gibt die isoparametrische Kurve an der U- oder V-Koordinatenposition zurück. Beachten Sie die isoDirection-Eingabe. Laden Sie die zu diesem Bild gehörige Beispieldatei herunter (durch Rechtsklicken).

und Wahl der Option Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

4. [Geometry for Computational Design - Surfaces.dyn](#)
5. [Surface.sat](#)

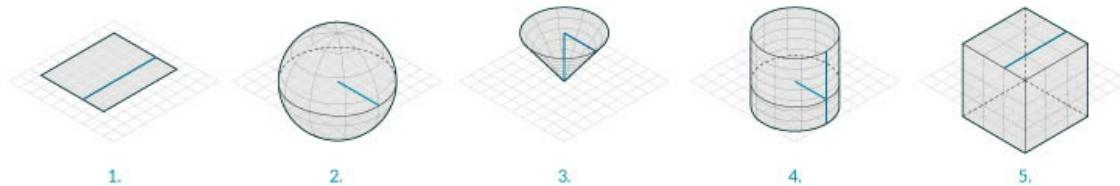
# Volumenkörper

## Volumenkörper

Wenn wir komplexere Modelle erstellen möchten, die nicht aus einer einzelnen Fläche erstellt werden können, oder wenn wir ein explizites Volumen definieren möchten, müssen wir uns in den Bereich der Volumenkörper (und Flächenverbände) vorwagen. Selbst ein einfacher Würfel ist so komplex, dass er sechs Oberflächen erfordert, eine pro Seite. Volumenkörper ermöglichen den Zugriff auf zwei wichtige Konzepte, den Oberflächen nicht bieten – eine verfeinerte topologische Beschreibung (Flächen, Kanten, Scheitelpunkte) und boolesche Operationen.

### Was ist ein Volumenkörper?

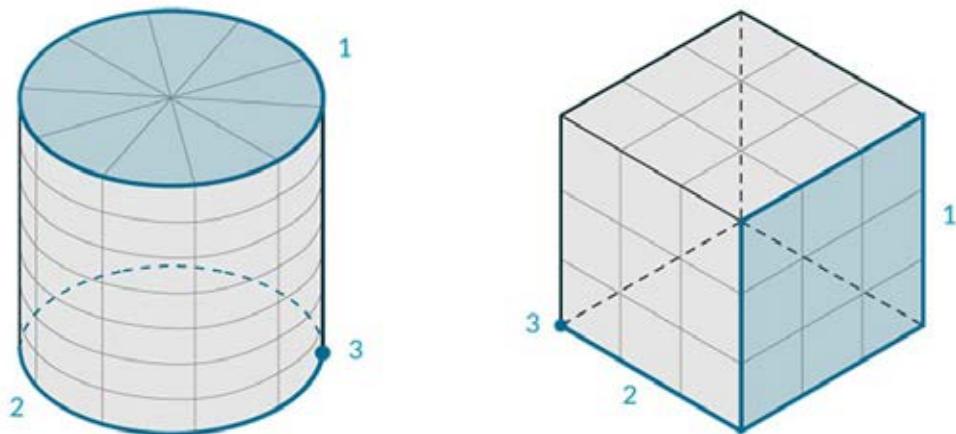
Volumenkörper bestehen aus einer oder mehreren Oberflächen, die ein Volumen durch eine geschlossene Berandung enthalten, die "drinnen" oder "draußen" definiert. Unabhängig davon, wie viele dieser Oberflächen vorhanden sind, müssen Sie ein "wasserliches" Volumen bilden, um als Volumenkörper zu gelten. Volumenkörper können erstellt werden, indem Oberflächen oder Flächenverbände miteinander verbunden werden oder durch Verwendung von Vorgängen wie Ausformung, Extrusion und Drehung. Die Grundkörper Kugel, Würfel, Kegel und Zylinder sind ebenfalls Volumenkörper. Ein Würfel, von dem mindestens eine Fläche entfernt wurde, gilt als Flächenverband mit ähnlichen Eigenschaften wie ein Volumenkörper, aber nicht mehr als Volumenkörper selbst.



1. Eine Ebene besteht aus einer einzelnen Oberfläche und ist kein Volumenkörper.
2. Eine Kugel besteht aus einer einzelnen Oberfläche, aber ist ein Volumenkörper.
3. Ein Kegel besteht aus zwei Oberflächen, die miteinander verbunden sind, um einen Volumenkörper zu bilden.
4. Ein Zylinder besteht aus drei Oberflächen, die miteinander verbunden sind, um einen Volumenkörper zu bilden.
5. Ein Würfel besteht aus sechs Oberflächen, die miteinander verbunden sind, um einen Volumenkörper zu bilden.

## Topologie

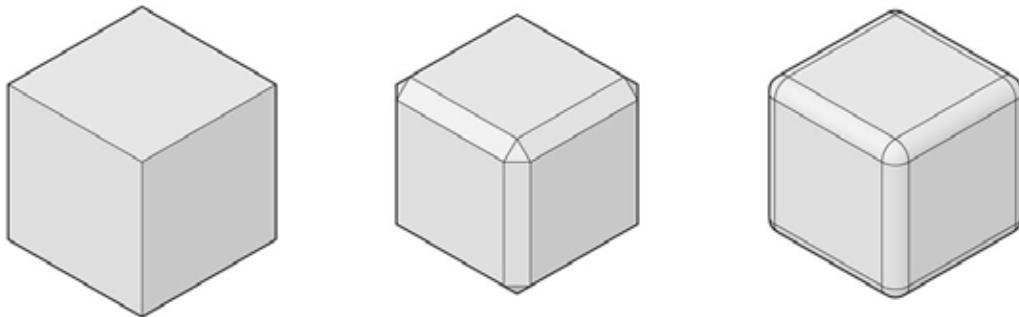
Volumenkörper bestehen aus drei Typen von Elementen: Scheitelpunkten, Kanten und Flächen. Flächen sind die Oberflächen, die einen Volumenkörper bilden. Kanten sind die Kurven, die die Verbindung zwischen angrenzenden Flächen definieren, und Scheitelpunkte sind die Start- und Endpunkte der Kurven. Diese Elemente können mit den Topologieknoten abgefragt werden.



1. Flächen
2. Kanten
3. Scheitelpunkte

### Vorgänge

Volumenkörper können geändert werden, indem ihre Kanten abgerundet oder gefast werden, um scharfe Ecken und Winkel zu entfernen. Durch den Fasvorgang wird eine Regeloberfläche zwischen zwei Flächen erzeugt, während durch eine Abrundung ein Übergang zwischen Flächen erzeugt wird, um Tangentialität beizubehalten.



1. Volumenkörperwürfel
2. Gefaster Würfel
3. Abgerundeter Würfel

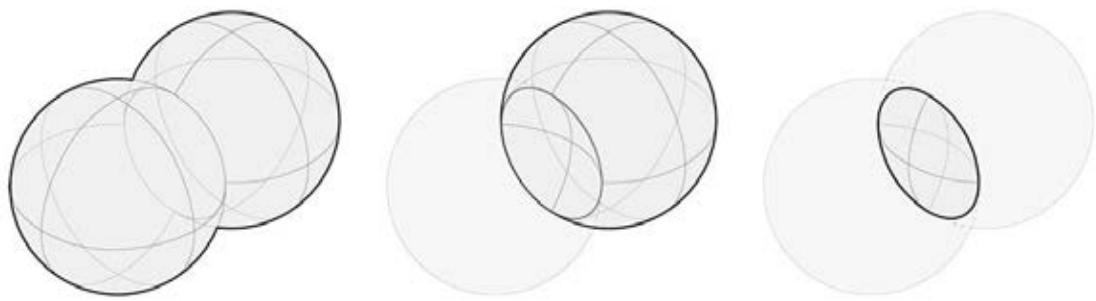
### Boolesche Operationen

Boolesche Operationen für Volumenkörper sind Methoden zum Kombinieren von zwei oder mehr Volumenkörpern. Bei einer einzelnen booleschen Operation werden eigentlich vier Vorgänge durchgeführt:

1. Zwei oder mehr Objekte **überschneiden**.
2. Die Objekte an den Schnittpunkten **teilen**.
3. Unerwünschte Teile der Geometrie **löschten**.
4. Alles wieder miteinander **verbinden**.

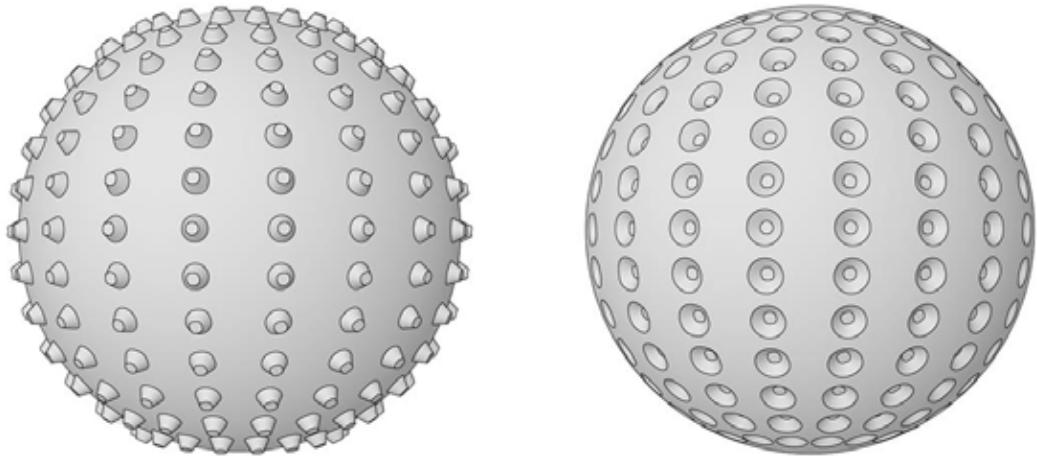
Dadurch werden boolesche Operationen für Volumenkörper zu einem leistungsstarken und zeitsparenden Prozess. Es gibt

drei boolesche Operationen für Volumenkörper, die unterscheiden, welche Teile der Geometrie beibehalten werden.



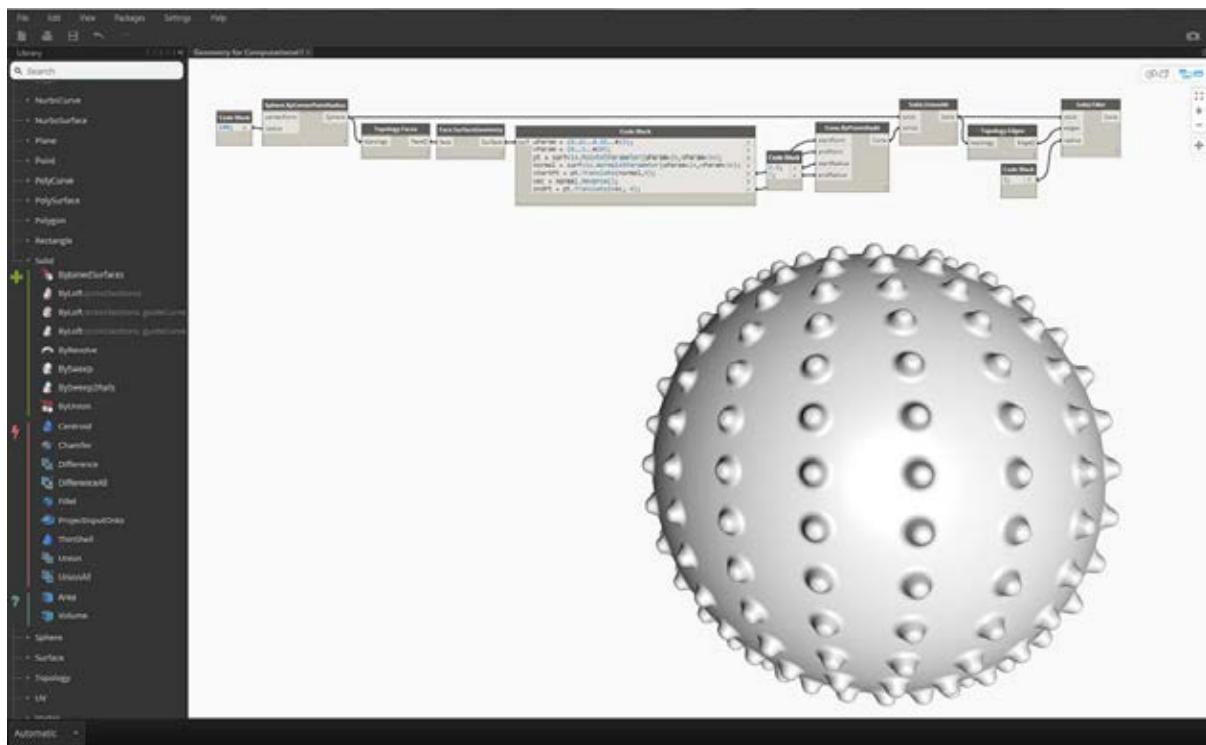
1. **Vereinigung:** Die überlappendenden Teile der Volumenkörper werden entfernt und sie werden zu einem einzelnen Volumenkörper verbunden.
2. **Differenz:** Ein Volumenkörper wird von einem anderen abgezogen. Der abzuziehende Volumenkörper wird als Werkzeug bezeichnet. Beachten Sie, dass Sie umschalten können, bei welchem Volumenkörper es sich um das Werkzeug handelt, um das inverse Volumen beizubehalten.
3. **Schnitt:** Nur das überschneidende Volumen der beiden Volumenkörper wird beibehalten.

Zusätzlich zu diesen drei Vorgänge sind in Dynamo die Knoten **Solid.DifferenceAll** und **Solid.UnionAll** verfügbar, mit denen Differenz- und Schnittvorgänge mit mehreren Volumenkörpern ausgeführt werden können.



1. **UnionAll :** Vereinigungsvorgang mit Kugel und nach außen gerichteten Kegeln
2. **DifferenceAll :** Differenzvorgang mit Kugel und nach innen gerichteten Kegeln

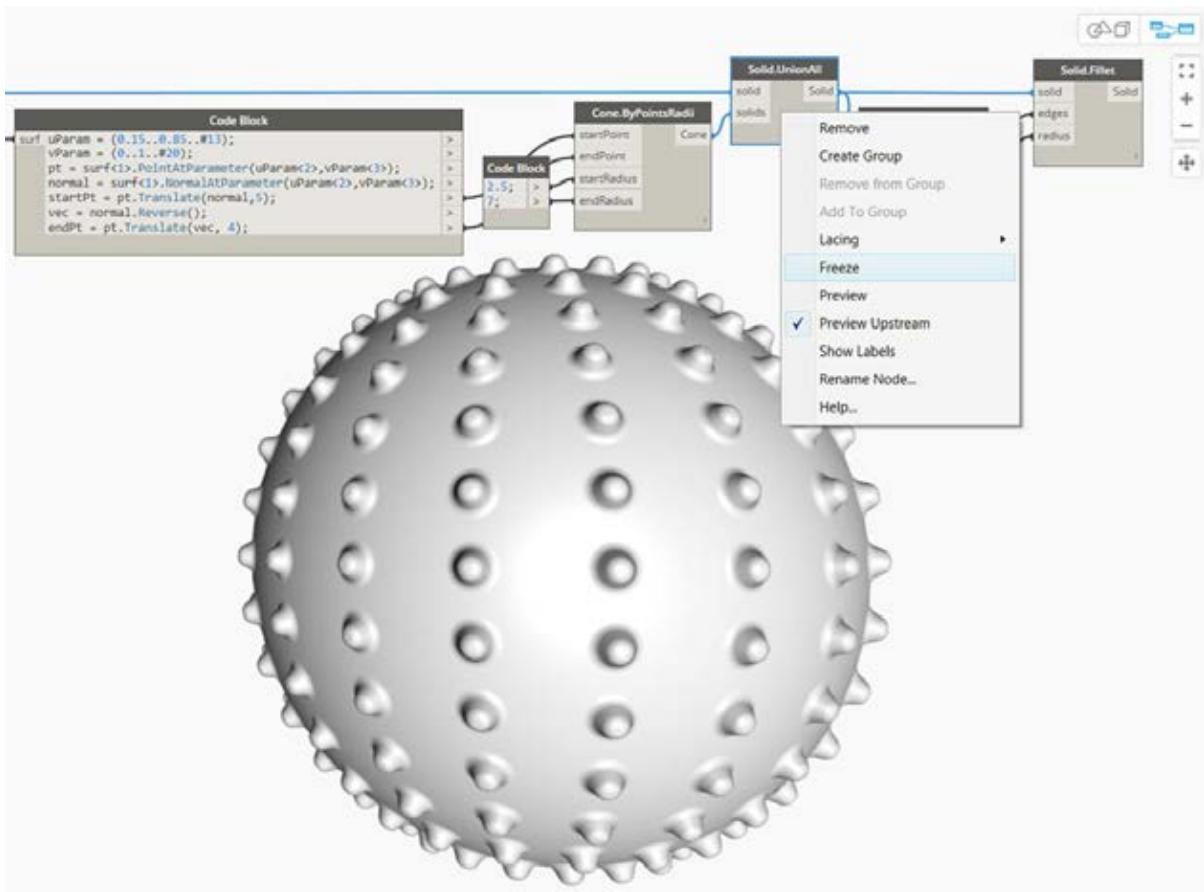
Führen Sie mehrere boolesche Operationen aus, um einen Noppenball zu erstellen.



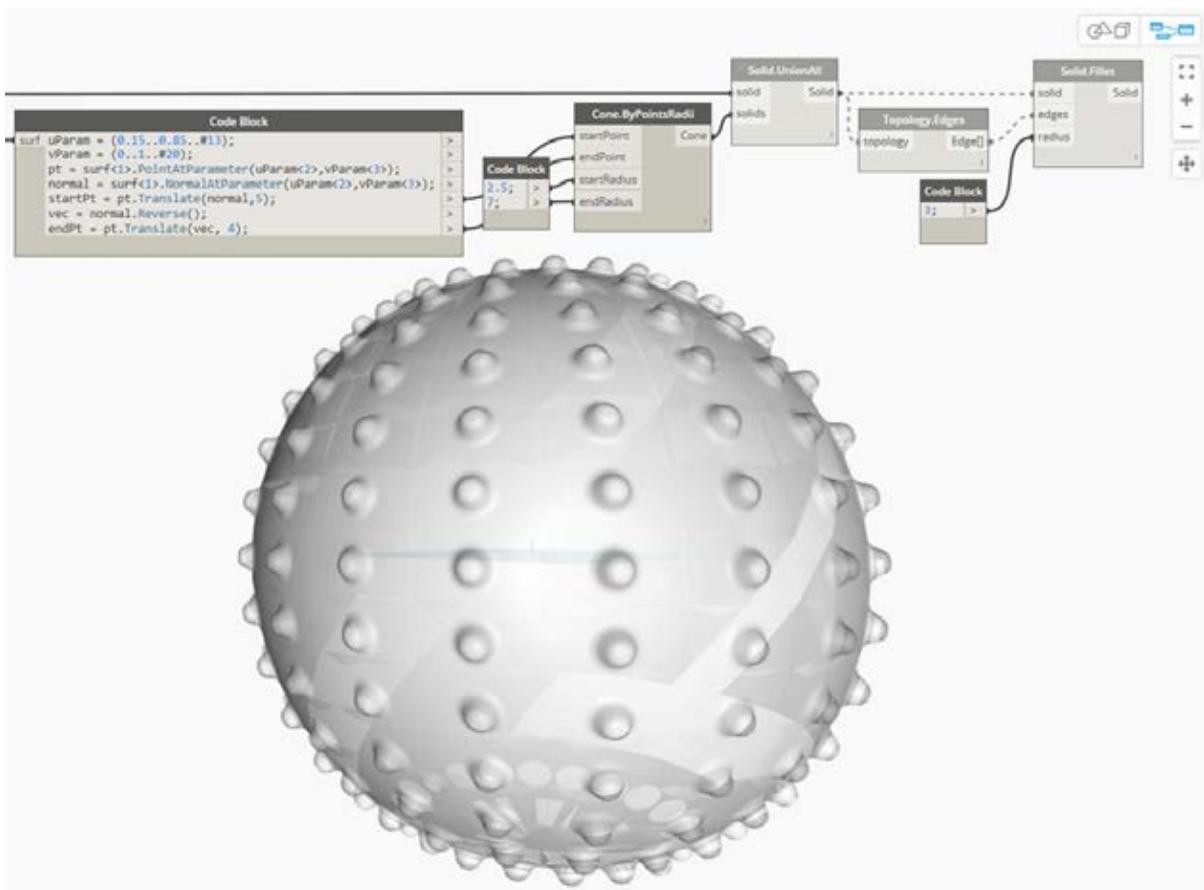
1. **Sphere.ByCenterPointRadius:** Der Basisvolumenkörper wird erstellt.
2. **Topology.Faces, Face.SurfaceGeometry:** Die Flächen des Volumenkörpers werden abgefragt und die Oberflächengeometrie wird konvertiert – in diesem Fall weist die Kugel nur eine Fläche auf.
3. **Cone.ByPointsRadii:** Mithilfe von Punkten auf der Oberfläche werden Kegel konstruiert.
4. **Solid.UnionAll:** Die Kegel und die Kugel werden vereinigt.
5. **Topology.Edges:** Die Kanten des neuen Volumenkörpers werden abgefragt.
6. **Solid.Fillet:** Die Kanten des Noppenballs werden abgerundet. Laden Sie die zu diesem Bild gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Geometry for Computational Design - Solids.dyn](#)

## Anhalten

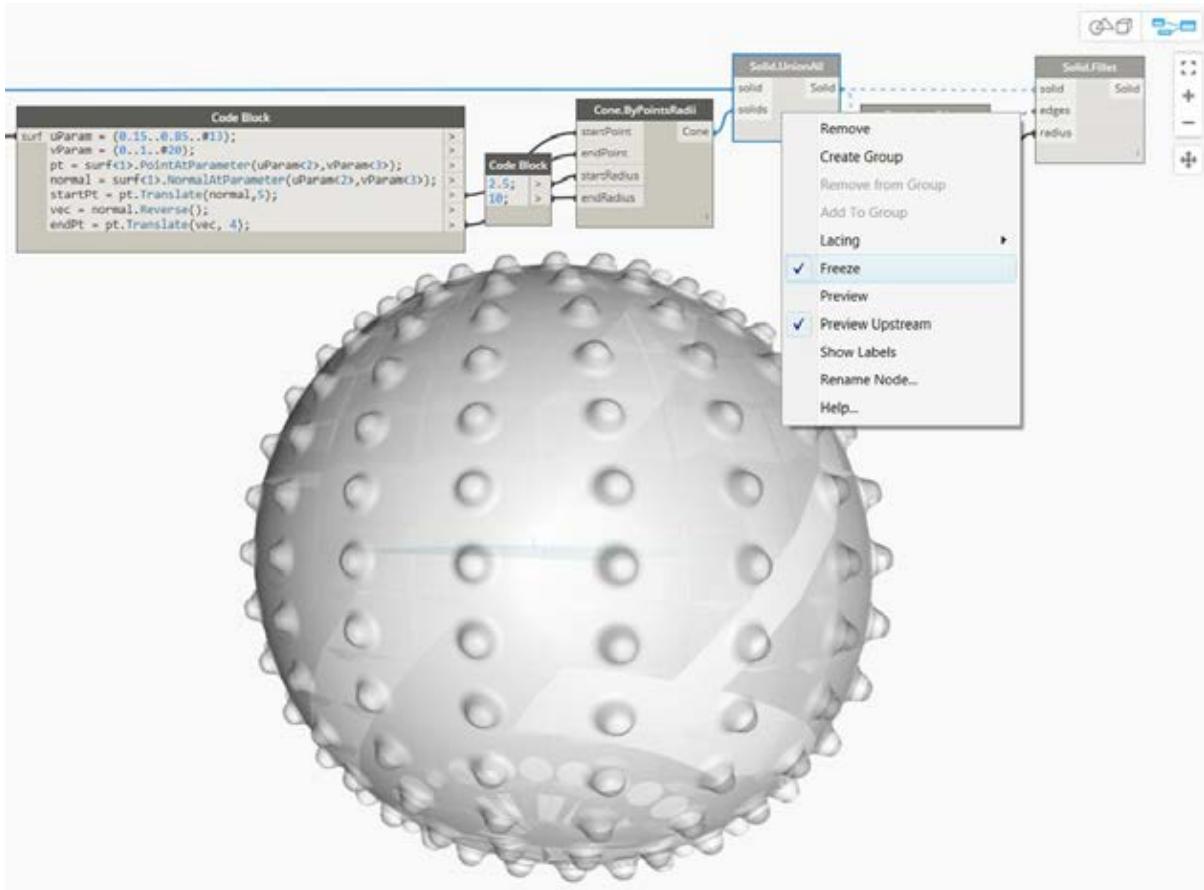
Boolesche Operationen sind sehr komplex und ihre Berechnung kann möglicherweise viel Zeit in Anspruch nehmen. Verwenden Sie die Anhaltfunktion, um die Ausführung der ausgewählten Knoten und der betroffenen untergeordneten Knoten zu unterbrechen.



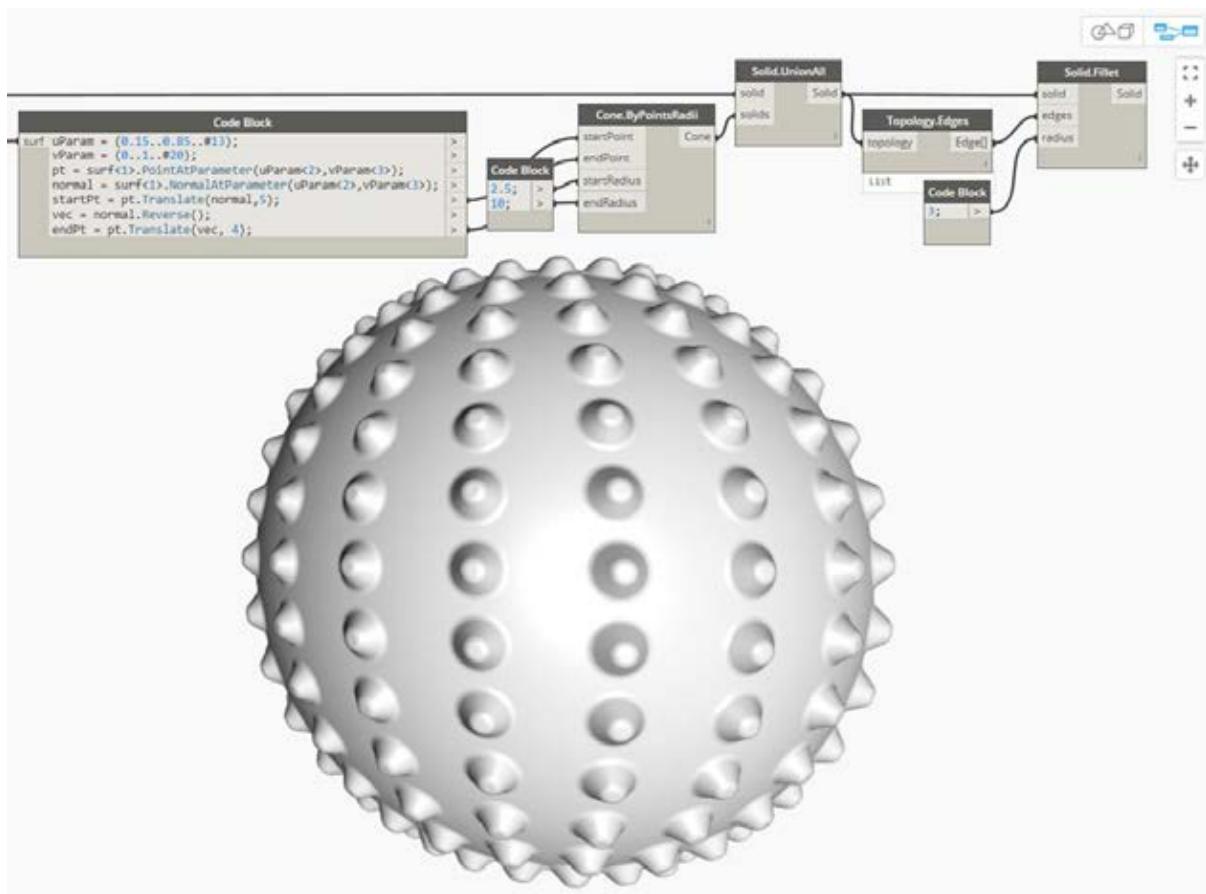
Verwenden Sie das Kontextmenü, um den Vorgang "Vereinigung" für einen Volumenkörper anzuhalten.



Der ausgewählte Knoten und alle untergeordneten Knoten werden in einem hellgrauen halbtransparenten Modus in einer Vorschau angezeigt und die betroffenen Drähte werden als gestrichelte Linien angezeigt. Die betroffene Geometrievorschau wird ebenfalls halbtransparent angezeigt. Sie können jetzt vorgelagerte Werte ändern, ohne die boolesche Vereinigung zu berechnen.



Um die Ausführung der Knoten fortzusetzen, klicken Sie mit der rechten Maustaste und deaktivieren "Anhalten".



Alle betroffenen Knoten und die zugehörigen Geometrievorschauen werden aktualisiert und wieder im standardmäßigen Vorschaumodus angezeigt.

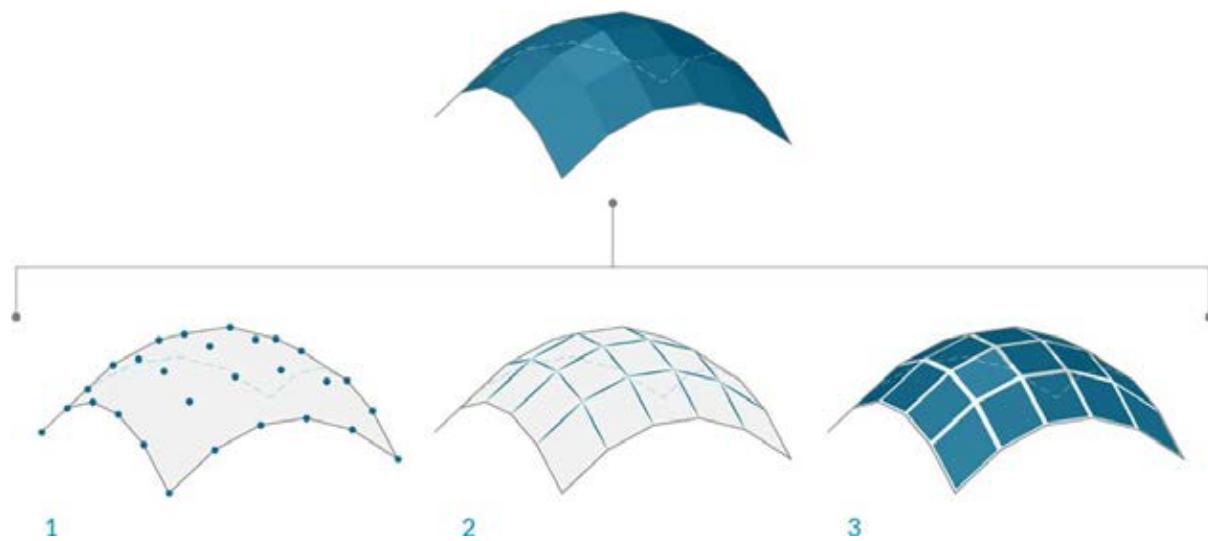
# Netze

## Netze

Im Bereich der computergestützten Modellierung stellen Netze eine der am weitesten verbreiteten Formen für die Darstellung von 3D-Geometrie dar. Netzgeometrie kann eine einfache und flexible Alternative zum Arbeiten mit NURBS darstellen. Darüber hinaus werden Netze in allem verwendet – von Renderings über Visualisierungen bis hin zur digitalen Fertigung und 3D-Druck.

### Was ist ein Netz?

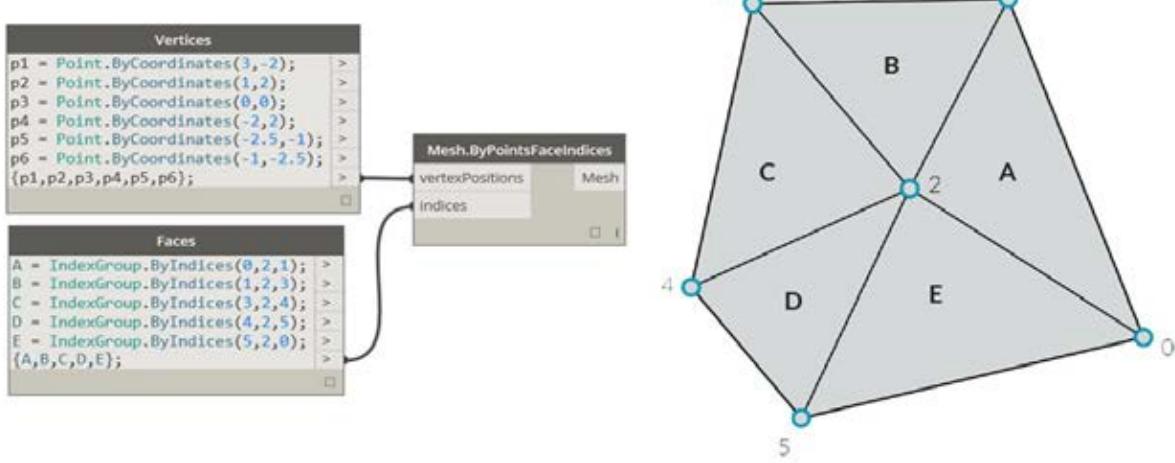
Ein Netz ist eine Sammlung von Vierecken und Dreiecken, die eine Oberfläche oder einen Volumenkörper darstellt. Wie bei Volumenkörpern enthält auch die Struktur von Netzobjekten Scheitelpunkte, Kanten und Flächen. Darüber hinaus gibt es weitere Eigenschaften, die Netze eindeutig machen, z. B. Normalen.



1. Netzscheitelpunkte
2. Netzkanten \*Kanten mit nur einer angrenzenden Fläche werden als "nackt" bezeichnet. Alle anderen Kanten sind "angezogen".
3. Netzflächen

## Netzelemente

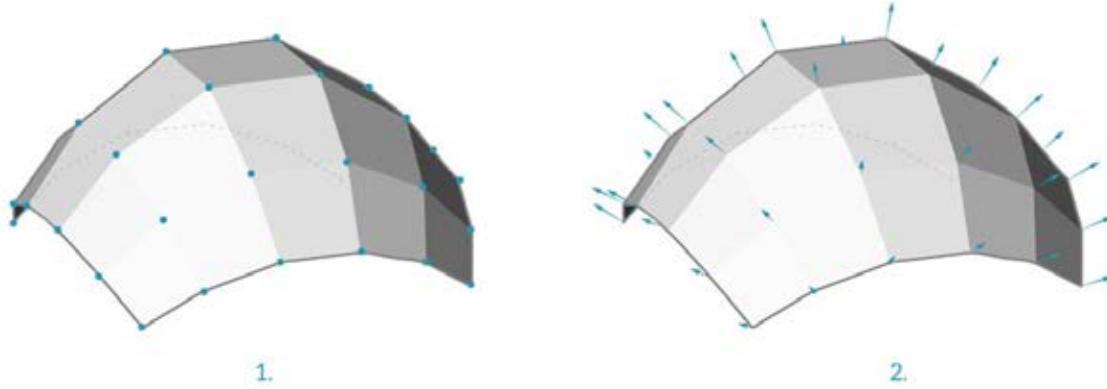
Dynamo definiert Netze mit einer Flächen-Scheitelpunkt-Datenstruktur. Auf elementarster Ebene handelt es sich bei dieser Struktur einfach um eine Sammlung von Punkten, die in Polygone gruppiert sind. Die Punkte eines Netzes werden als Scheitelpunkte bezeichnet, während die oberflächenartigen Polygone als Flächen bezeichnet werden. Um ein Netz zu erstellen, benötigen Sie eine Liste von Scheitelpunkten und ein System für die Gruppierung dieser Scheitelpunkte in Flächen. Dies wird auch als Indexgruppe bezeichnet.



1. Liste von Scheitelpunkten
2. Liste von Indexgruppen zum Definieren von Flächen

### Scheitelpunkte + Scheitelpunktnormalen

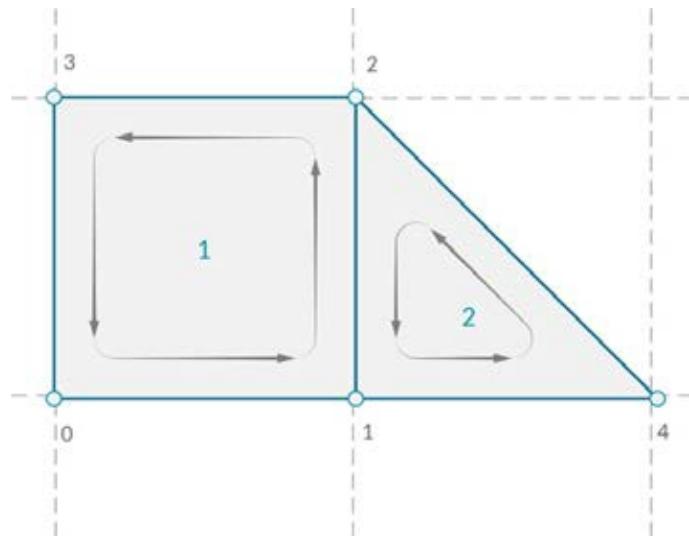
Die Scheitelpunkte eines Netzes entsprechen einfach einer Liste von Punkten. Der Index der Scheitelpunkte ist beim Konstruieren eines Netzes oder Abrufen von Informationen über die Struktur eines Netzes sehr wichtig. Für jeden Scheitelpunkt gibt es auch eine entsprechende Scheitelpunktnormale (Vektor), die die durchschnittliche Richtung der verbundenen Flächen beschreibt und Sie dabei unterstützt, die nach innen und nach außen gerichtete Orientierung des Netzes zu verstehen.



1. Scheitelpunkte
2. Scheitelpunktnormalen

### Flächen

Eine Fläche ist eine geordnete Liste von drei oder vier Scheitelpunkten. Die "Oberflächendarstellung" einer Netzfläche ist deshalb gemäß der Position der indizierten Scheitelpunkte impliziert. Sie verfügen bereits über die Liste der Scheitelpunkte, die ein Netz bilden. Statt also individuelle Punkte anzugeben, um eine Fläche zu definieren, verwenden Sie einfach den Index der Scheitelpunkte. Dies ermöglicht Ihnen auch die Verwendung desselben Scheitelpunkts in weiteren Flächen.



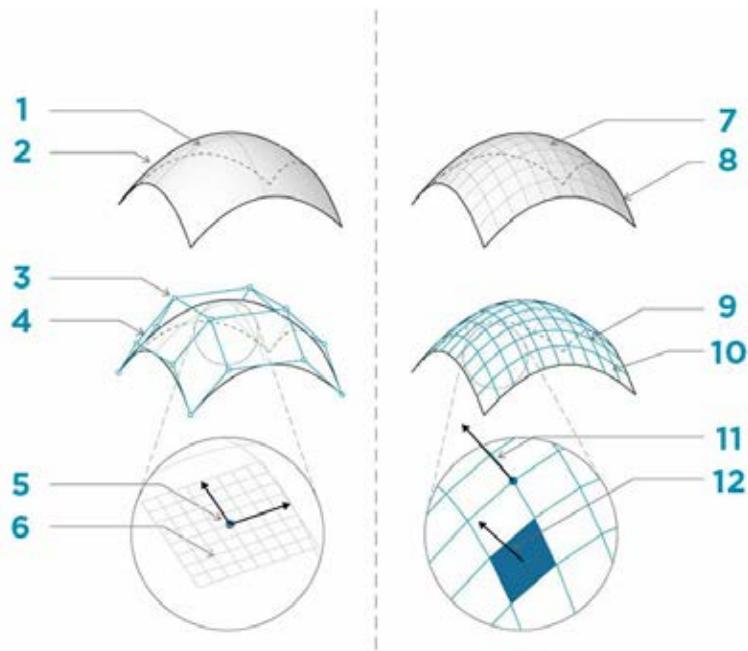
1. Quadratische Fläche, die aus den Indizes 0, 1, 2 und 3 erstellt wurde
2. Dreieckige Fläche, die aus den Indizes 1, 4 und 2 erstellt wurde. Beachten Sie, dass die Indexgruppen in ihrer Reihenfolge verschoben werden können – solange die Sequenz gegen den Uhrzeigersinn angeordnet ist, ist die Fläche korrekt definiert

## Netze und NURBS-Oberflächen im Vergleich

Welche Unterschiede bestehen zwischen Netz- und NURBS-Geometrie? Wann möchten Sie die eine Geometrie anstelle der anderen verwenden?

### Parametrisierung

In einem früheren Kapitel haben wir gesehen, dass NURBS-Oberflächen durch eine Reihe von NURBS-Kurven in zwei Richtungen definiert werden. Diese Richtungen werden als U und V bezeichnet und ermöglichen, dass eine NURBS-Oberfläche gemäß einer zweidimensionalen Oberflächendomäne parametrisiert wird. Die Kurven selbst werden als Gleichungen im Computer gespeichert, sodass die resultierenden Oberflächen auf einen beliebigen, verhältnismäßig kleinen Genauigkeitsbereich berechnet werden können. Es kann jedoch schwierig sein, mehrere NURBS-Oberflächen miteinander zu kombinieren. Das Verbinden von zwei NURBS-Oberflächen führt zu einem Flächenverband, in dem verschiedene Bereiche der Geometrie unterschiedliche UV-Parameter und Kurvendefinitionen aufweisen.



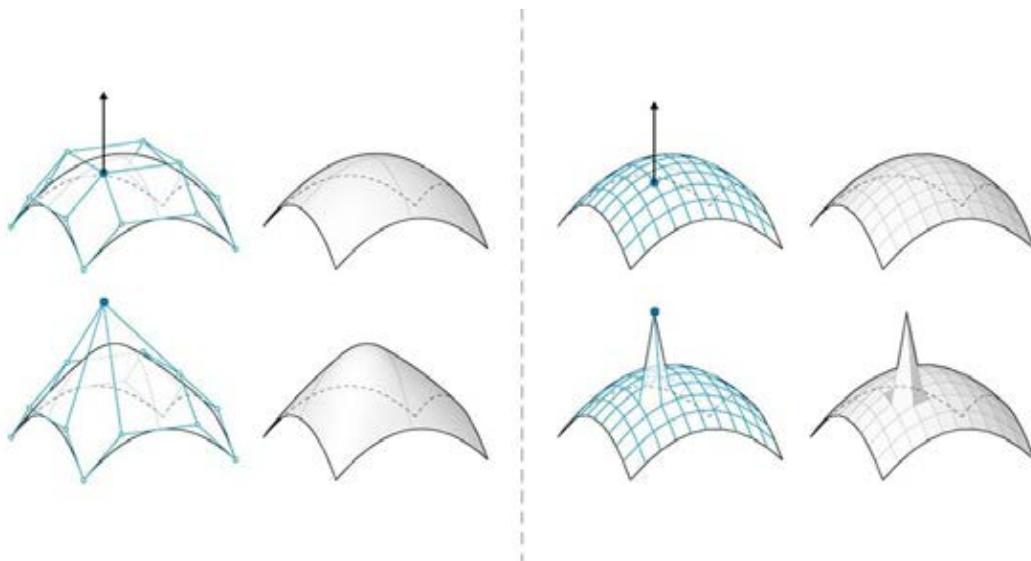
1. Oberfläche
2. Isoparametrische (Isoparm) Kurve

3. Steuerpunkt der Oberfläche
4. Steuerpunkt des Flächenverbands
5. Isoparametrischer Punkt
6. Oberflächenrahmen
7. Netz
8. Nackte Kante
9. Maschennetz
10. Netzkanten
11. Scheitelpunktnormale
12. Netzfläche/Netzflächennormale

Netze auf der anderen Seite bestehen aus einer diskreten Anzahl von genau definierten Scheitelpunkten und Flächen. Das Netzwerk von Scheitelpunkten kann im Allgemeinen nicht durch einfache UV-Koordinaten definiert werden. Da die Anzahl an Flächen diskret ist, bestimmt sich daraus auch der Genauigkeitsgrad des Netzes, der nur geändert werden kann, indem das Netz neu definiert und weitere Flächen hinzugefügt werden. Das Fehlen der mathematischen Beschreibungen ermöglicht Netzen die flexiblere Handhabung komplexer Geometrie innerhalb eines einzelnen Netzes.

### Lokaler und globaler Einfluss im Vergleich

Ein weiterer wichtiger Unterschied ist das Ausmaß, in dem sich eine lokale Änderung der Netz- oder NURBS-Geometrie auf die gesamte Form auswirkt. Das Verschieben von einem Scheitelpunkt eines Netzes wirkt sich nur auf die an diesen Scheitelpunkt angrenzenden Flächen aus. In NURBS-Oberflächen ist das Ausmaß des Einflusses wesentlich komplizierter und richtet sich sowohl nach dem Grad der Oberfläche als auch nach den Gewichtungen und Knoten der Steuerpunkte. Allgemein wird durch das Verschieben eines einzelnen Steuerpunkts in einer NURBS-Oberfläche eine glattere, umfassendere Änderungen in der Geometrie erzeugt.



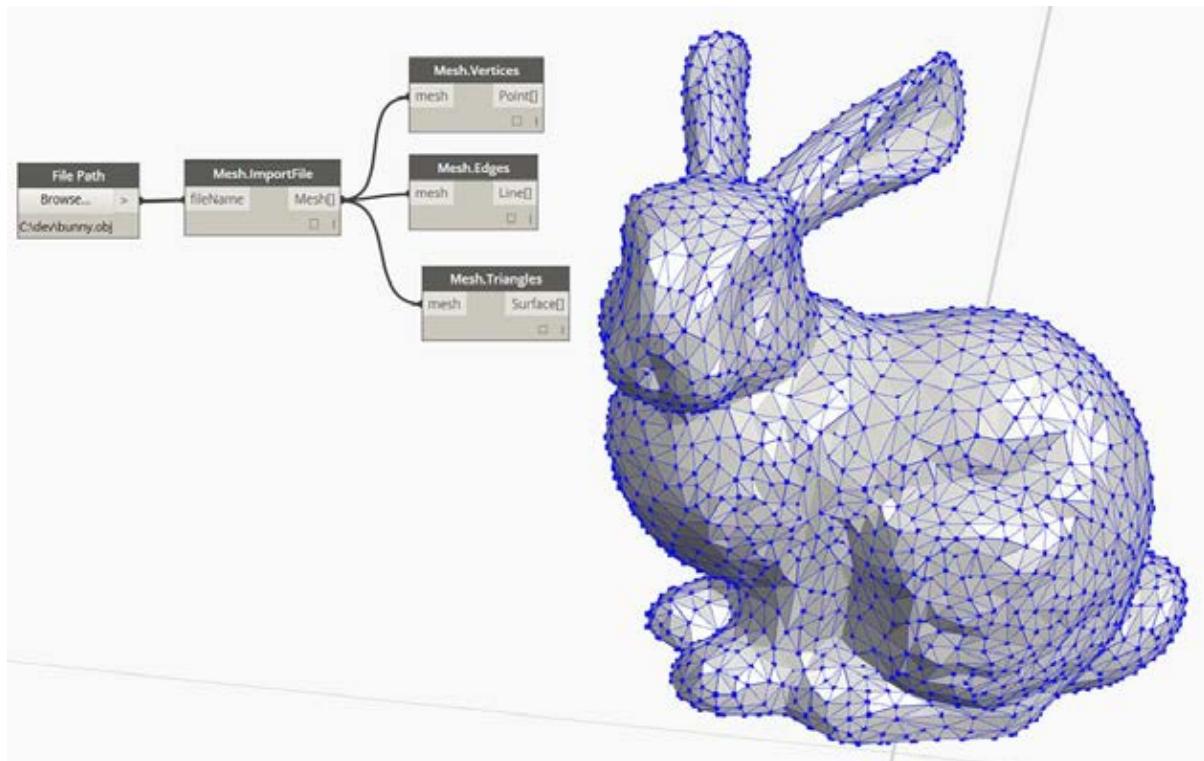
1. NURBS-Oberfläche: Das Verschieben eines Steuerpunkts wirkt sich über die Form hinaus aus.
2. Netzgeometrie – Das Verschieben eines Scheitelpunkts wirkt sich nur auf die angrenzenden Elemente aus.

Eine Analogie, die hilfreich sein kann, besteht im Vergleich eines Vektorbilds (bestehend aus Linien und Kurven) mit einem Rasterbild (bestehend aus einzelnen Pixeln). Wenn Sie die Anzeige eines Vektorbilds vergrößern, sind die Kurven weiterhin klar und deutlich zu sehen, während das Vergrößern eines Rasterbilds dazu führt, dass die einzelnen Pixel größer werden. In dieser Analogie können NURBS-Oberflächen mit einem Vektorbild verglichen werden, da eine glatte mathematische Beziehung besteht, während sich ein Netz ähnlich wie ein Rasterbild mit einer festgelegten Auflösung verhält.

### Mesh Toolkit

Der Funktionsumfang in Bezug auf Netze von Dynamo kann durch die Installation des Pakets [Mesh Toolkit](#) erweitert werden. Das Dynamo Mesh Toolkit bietet Werkzeuge zum Importieren von Netzen aus externen Dateiformaten, zum Erstellen von Netzen aus Dynamo Geometrieobjekten und zum manuellen Erstellen von Netzen aus ihren Scheitelpunkten und Indizes. Die Bibliothek enthält auch Werkzeuge zum Ändern und Reparieren von Netzen sowie zum Extrahieren horizontaler Scheiben zur Verwendung in der Fertigung.

Siehe Kapitel 10.2 für ein Beispiel zur Verwendung von Mesh Toolkit.



# Importieren von Geometrie

## Importieren von Geometrie

Es gibt mehrere Möglichkeiten zum Importieren von Geometrie in Dynamo. Der Import von Netzen mithilfe von *Mesh Toolkit* wurde im vorigen Abschnitt bereits gezeigt. Es ist ebenfalls möglich, Körpermodelle aus SAT-Dateien zu importieren. Mit diesen Verfahren können Sie Geometrie in anderen Plattformen entwickeln, sie in Dynamo laden und mithilfe visueller Programmierung parametrische Operationen anwenden.

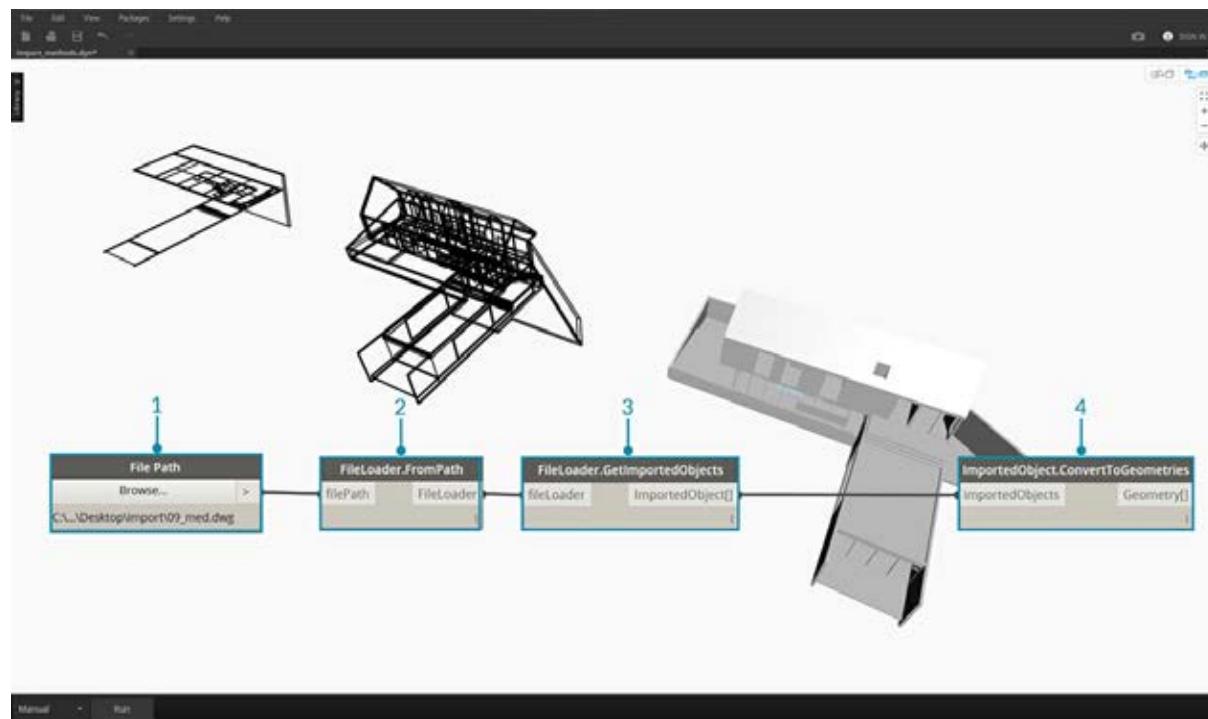
Eine andere Methode für den Import von Geometrie nutzt einen Prozess namens *ATF-Translation*. In diesem Fall können Sie nicht nur die Geometrie, sondern auch die Struktur einer Datei importieren. So können Sie beispielsweise wählen, welche Layer aus einer DWG-Datei importiert werden sollen, anstatt das gesamte Modell zu importieren. Dies wird im Folgenden genauer beschrieben.

### Importieren von Geometrie aus einer DWG-Datei

Blöcke zum Importieren von DWG-Dateien in die Dynamo-Umgebung finden Sie unter *Translation*. (Anmerkung: Diese Tools stehen nur in [Dynamo Studio](#) zur Verfügung.) Die folgenden Beispiele zeigen eine Reihe von Komponenten, mit deren Hilfe Sie nach einer Datei suchen, ihren Inhalt importieren und diesen in für Dynamo verwendbare Geometrie konvertieren können. In Dynamo können Sie darüber hinaus bestimmte Objekte für den Import aus einer DWG-Datei filtern und auswählen, was im Folgenden gezeigt wird. Weitere Informationen zum Importieren von Geometrie aus einer DWG-Datei finden Sie in Ben Gohs [Blog-Post hier](#).

### Importierte Objekte abrufen

Die einfachste Möglichkeit zum Importieren von DWG-Dateien in Dynamo Studio besteht darin, die gesamte Datei in den Arbeitsbereich zu importieren:



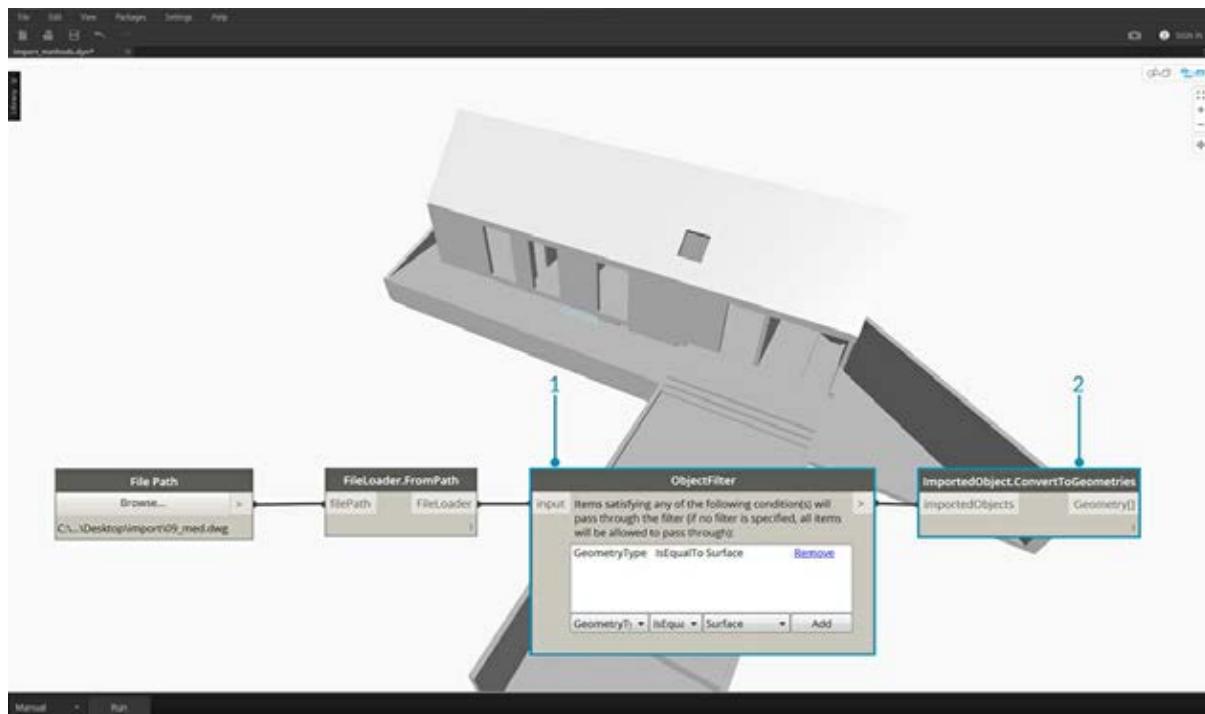
1. Suchen Sie mithilfe der File Path-Komponente nach der DWG-Datei, die in Dynamo importiert werden soll.
2. Verbinden Sie dies mit **FileLoader.FromPath**, um die Datei zu lesen.
3. Verwenden Sie die **FileLoader.GetImportedObjects**-Komponente zum Parsen der Geometrie für Dynamo Studio.
4. Mit **ImportedObject.ConvertToGeometries** werden die Objekte in Geometrie konvertiert, die Sie im Arbeitsbereich von Dynamo verwenden können.

Wie die Abbildung oben zeigt, werden sämtliche Typen von Geometrie – Oberflächen, Netze, Kurven und Linien – in Dynamo importiert.

## Objektfilter

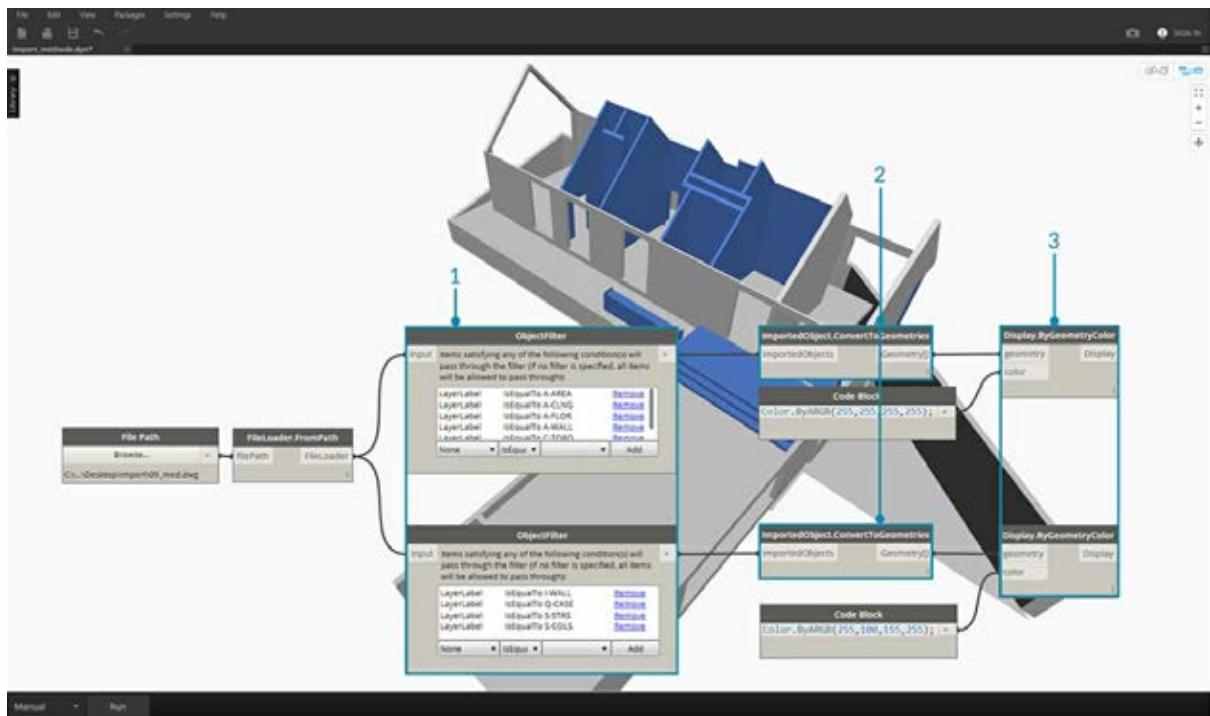
Um festzulegen, welche Geometrieelemente aus der DWG-Datei importiert werden sollen, können Sie der Definition zusätzliche **ObjectFilter**-Blöcke hinzufügen. Der **ObjectFilter**-Block ist kompatibel entweder mit **FileLoader** oder mit einer Liste von **ImportedObject** und gibt eine **ImportedObject**-Liste aus.

Die folgenden Abbildungen zeigen die Bedingungsanweisungen in den einzelnen **ObjectFilter**-Blöcken. Jedes **ImportedObject**, das eine der Bedingungen in der Liste erfüllt, passiert den Filter. Sie können nach Layer-Bezeichnung (d. h. Layer-Namen), Geometriertyp, Streufarbe usw. filtern sowie Filter kombinieren, um die Auswahl zu präzisieren.



1. Ersetzen Sie **FileLoader.GetImportedObjects** durch **ObjectFilter**, um die DWG-Datei anhand bestimmter Bedingungen zu durchsuchen. In diesem Fall wird nur Oberflächengeometrie importiert. Die im vorigen Bild sichtbare Kurven- und Liniengeometrie wird entfernt.
2. Verbinden Sie den Filter mit **ImportedObject.ConvertToGeometries**, um die gefilterte Geometrie zu importieren.

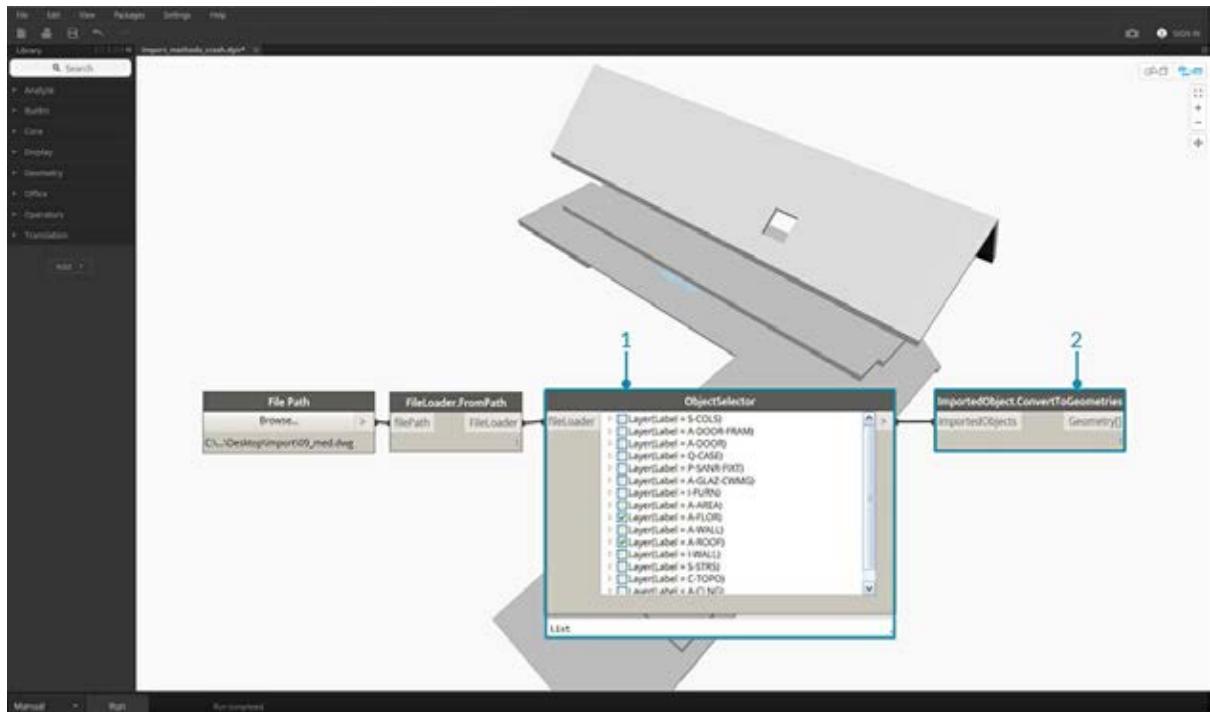
Indem Sie zwei Filter mit unterschiedlichen Bedingungsanweisungen hinzufügen, können Sie die Liste der Geometrie in mehrere Gruppen unterteilen:



1. Ersetzen Sie **FileLoader.GetImportedObjects** durch zwei **ObjectFilter**-Module mit unterschiedlichen Bedingungsanweisungen. Die aus derselben Datei stammende Geometrie wird dadurch in zwei verschiedene Gruppen unterteilt.
2. Verbinden Sie den Filter mit **ImportedObject.ConvertToGeometries**, um die gefilterte Geometrie zu importieren.
3. Verbinden Sie **ImportedObject.ConvertToGeometries** mit **Display.ByGeometryColor**, um die Gruppen in verschiedenen Farben zu visualisieren.

## Explizite Objektauswahl

Der **ObjectSelector**-Block bietet eine alternative Methode zum Importieren von Objekten aus der DWG-Datei. Bei dieser Methode werden keine Filter verwendet, sondern Sie können gezielt wählen, welche Objekte und Layer in Dynamo importiert werden sollen.

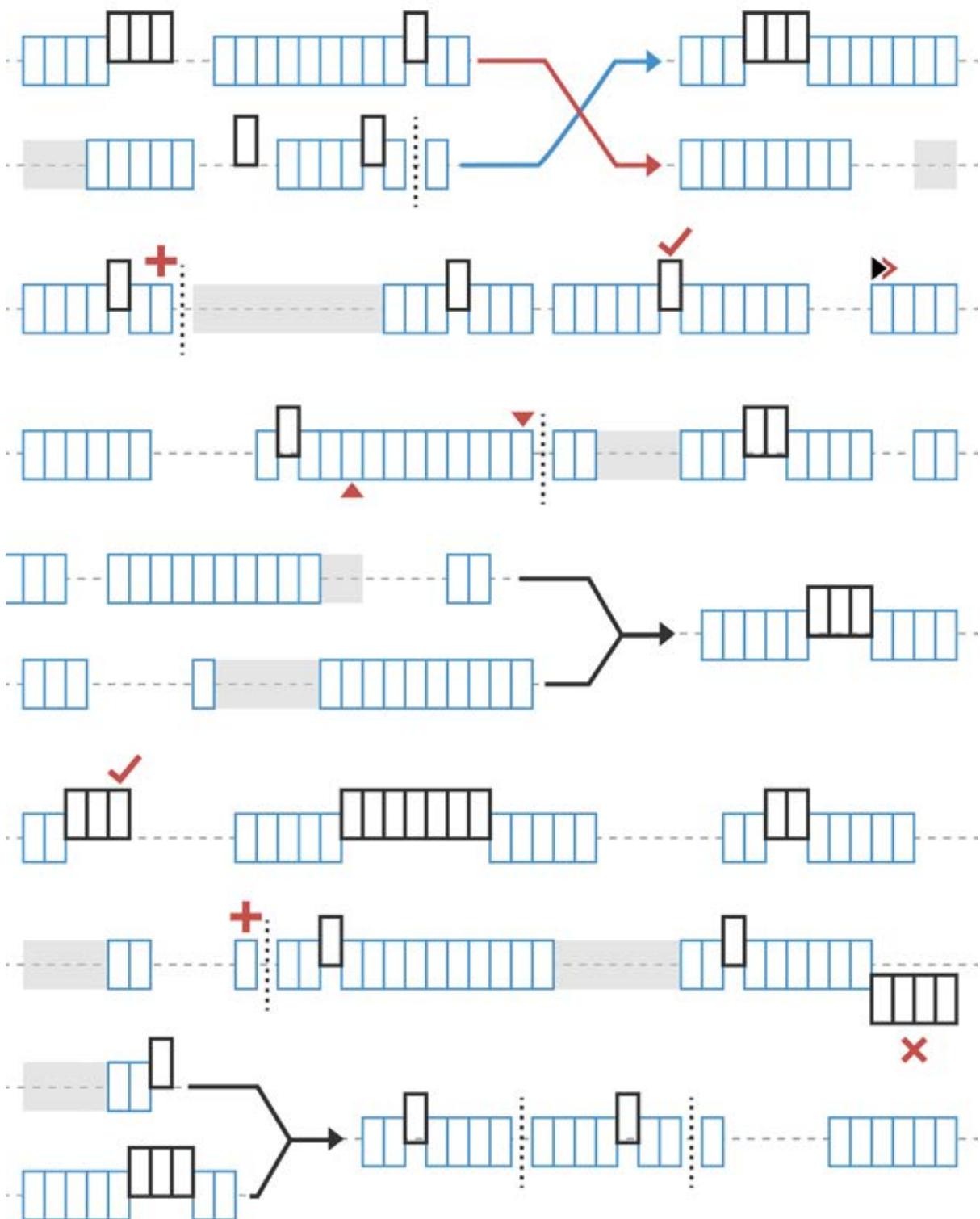


1. Ersetzen Sie **FileLoader.GetImportedObjects** durch **ObjectSelector**, um bestimmte Layer und Objekte in einer DWG-Datei abzurufen.
2. Verbinden Sie Filter mit **ImportedObject.ConvertToGeometries**.

## **Entwerfen mit Listen**

### **Entwerfen mit Listen**

Die Daten werden mithilfe von Listen geordnet. Im Betriebssystem des Computers befinden sich Dateien und Ordner. In Dynamo können diese als Einträge bzw. Listen betrachtet werden. Wie in Ihrem Betriebssystem stehen viele Möglichkeiten zum Erstellen, Ändern und Abfragen von Daten zur Verfügung. In diesem Kapitel wird die Verwaltung von Listen in Dynamo im Einzelnen beschrieben.



# Was ist eine Liste?

## Was ist eine Liste?

Eine Liste ist eine Sammlung von Elementen oder Einträgen. Ein Beispiel kann z. B. ein Bündel Bananen sein. Jede Banane ist ein Eintrag in der Liste (bzw. im Bündel). Ein Bündel Bananen ist leichter aufzuheben als die einzelnen Bananen. Dasselbe gilt für die Gruppierung von Elementen durch parametrische Beziehungen in einer Datenstruktur.



Foto [Augustus Binu](#).

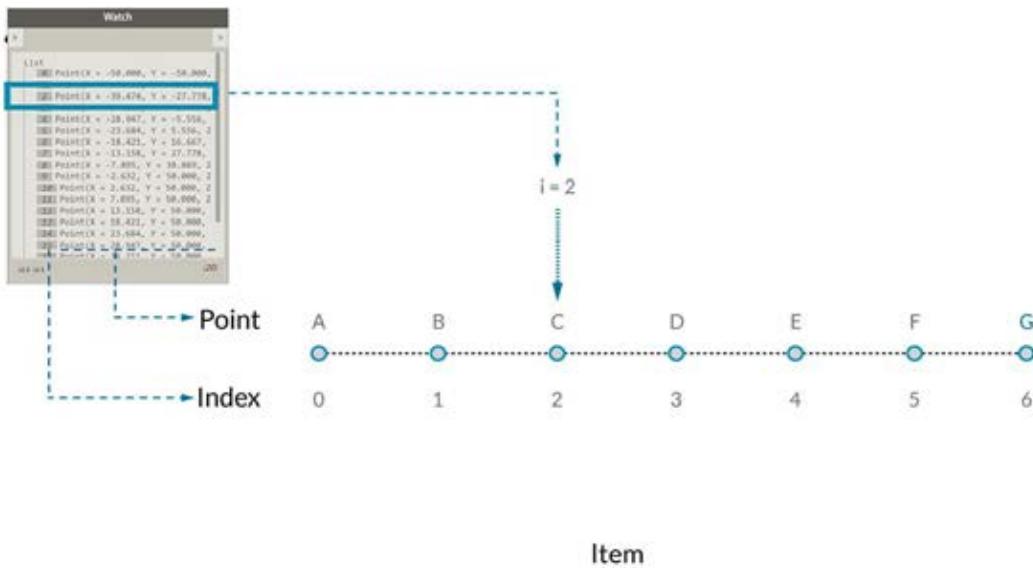
Beim Einkaufen packen Sie alle gekauften Artikel in eine Tasche. Auch diese Tasche ist eine Liste. Angenommen, Sie benötigen drei Bündel Bananen, um (eine *große Menge*)Bananenbrot zu backen. Damit stellt die Tasche eine Liste mit Bananenbündeln und jedes Bündel eine Liste mit Bananen dar. Die Tasche ist eine Liste von Listen (zweidimensional) und jedes Bananenbündel ist eine Liste (eindimensional).

In Dynamo sind Listendaten geordnet und der erste Eintrag in einer Liste hat immer den Index "0". Weiter unten wird beschrieben, wie Listen in Dynamo definiert werden und welche Beziehungen zwischen mehreren Listen möglich sind.

## Nullbasierte Indizes

Auf den ersten Blick scheint es ungewohnt, dass der erste Index einer Liste immer 0 und nicht 1 lautet. Wenn also vom ersten Eintrag in einer Liste die Rede ist, ist damit der Eintrag mit dem Index 0 gemeint.

Wenn Sie etwa die Finger an Ihrer rechten zählen, würden Sie von 1 bis 5 zählen. In einer Liste in Dynamo hätten Ihre Finger jedoch die Indizes 0–4. Für Einsteiger in die Programmierung ist dies eventuell zunächst ungewohnt. Nullbasierte Indizes sind jedoch die in den meisten Rechensystemen gebräuchliche Praxis.



### Item

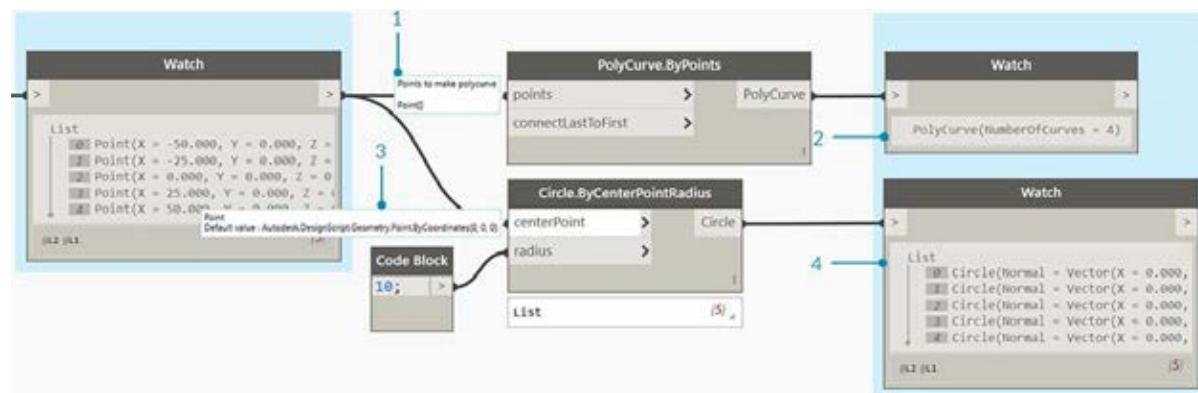
Beachten Sie, dass die Liste nach wie vor 5 Einträge enthält, sie werden nur beginnend mit 0 gezählt. Die Einträge in Listen müssen nicht unbedingt Zahlen sein. Vielmehr können alle in Dynamo unterstützten Datentypen verwendet werden: Punkte, Kurven, Oberflächen, Familien usw.

Die einfachste Möglichkeit, den Typ der in einer Liste enthaltenen Daten sichtbar zu machen, besteht oft darin, einen Watch-Block mit der Ausgabe eines anderen Blocks zu verbinden. Im Watch-Block werden per Vorgabe alle Indizes automatisch links und die Datenelemente rechts in der Liste angezeigt.

Diese Indizes sind ein entscheidendes Element bei der Arbeit mit Listen.

## Eingaben und Ausgaben

Ein- und Ausgaben behandeln Listen abhängig vom verwendeten Block unterschiedlich. In diesem Beispiel wird die ausgegebene Liste mit fünf Punkten mit zwei verschiedenen Dynamo-Blöcken verbunden: *PolyCurve.ByPoints* und *Circle.ByCenterPointRadius*:



1. Die *points*-Eingabe von *PolyCurve.ByPoints* sucht nach "*Point[]*". Dies entspricht einer Liste von Punkten.
2. Die Ausgabe von *PolyCurve.ByPoints* ist eine einzelne PolyCurve, die aus den fünf Punkten aus der Liste erstellt wird.
3. Die *centerPoint*-Eingabe für *Circle.ByCenterPointRadius* verlangt "*Point*".
4. Die Ausgabe für *Circle.ByCenterPointRadius* ist eine Liste mit fünf Kreisen, deren Mittelpunkte den Punkten aus der ursprünglichen Liste entsprechen.

In *PolyCurve.ByPoints* und in *Circle.ByCenterPointRadius* wurden dieselben Daten eingegeben; mit PolyCurve erhalten Sie jedoch nur eine Polykurve, während der Circle-Block fünf Kreise um die einzelnen Punkte ausgibt. Intuitiv ist dies einleuchtend: Die Polykurve wird als Kurve gezeichnet, die die fünf Punkte verbindet, bei den Kreisen hingegen wird um jeden Punkt ein eigener Kreis erstellt. Was geschieht dabei mit den Daten?

Wenn Sie den Mauszeiger auf die *points*-Eingabe von *Polycurve.ByPoints* setzen, sehen Sie, dass "*Point[]*" als Eingabe

verlangt wird. Beachten Sie die Klammern am Ende. Dies steht für eine Liste von Punkten. Um eine einzelne Polykurve zu erstellen, wird eine Liste benötigt. Das bedeutet, dass dieser Block jede eingegebene Liste zu einer PolyCurve zusammenfasst.

Im Gegensatz dazu verlangt die *centerPoint*-Eingabe für *Circle.ByCenterPointRadius* den Typ "Point". Dieser Block sucht nach einem einzelnen Punkt als einem eigenständigen Eintrag, um den Mittelpunkt des Kreises zu definieren. Aus diesem Grund erhalten Sie fünf Kreise aus den Eingabedaten. Die Kenntnis dieser Unterschiede bei den Eingaben in Dynamo erleichtert das Verständnis der Funktionsweise der Blöcke bei der Verarbeitung von Daten.

## Vergitterung

Für die Zuordnung von Daten gibt es keine eindeutige Lösung. Dieses Problem tritt auf, wenn in einem Block Eingaben von unterschiedlicher Größe verwendet werden. Die Verwendung unterschiedlicher Algorithmen zur Datenzuordnung kann zu äußerst unterschiedlichen Ergebnissen führen.

Angenommen, ein Block erstellt Liniensegmente zwischen Punkten (*Line.ByStartEndPoint*). Hierfür werden zwei Eingabeparameter verwendet. Beide stellen Punktkoordinaten bereit:

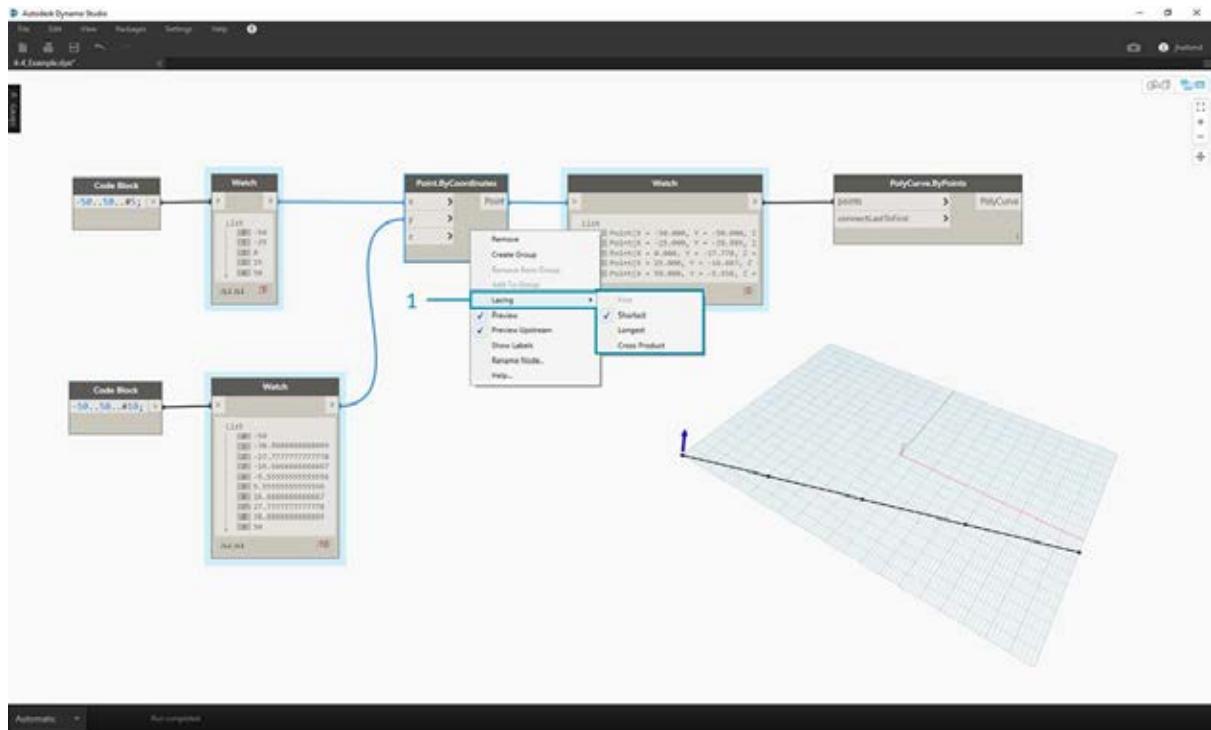


Es ist leicht zu erkennen, dass es mehrere Möglichkeiten gibt, Linien zwischen diesen Punktgruppen zu zeichnen. Um die Vergitterungsoptionen aufzurufen, klicken Sie mit der rechten Maustaste in die Mitte eines Blocks und wählen das Menü Vergitterung.

## Basisdatei

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [Lacing.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

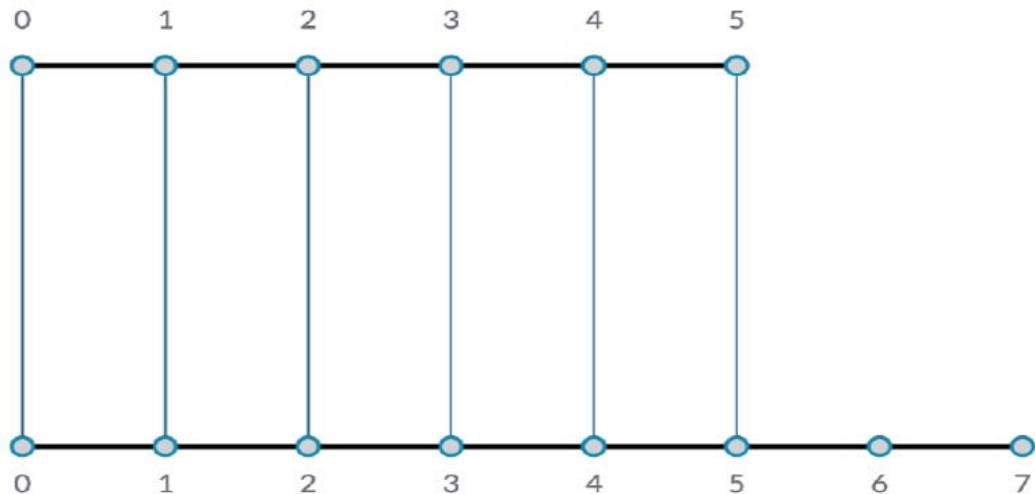
Zur Demonstration der unten beschriebenen Vergitterungsoptionen werden anhand dieser Basisdatei die kürzeste und die längste Liste sowie das Kreuzprodukt definiert.

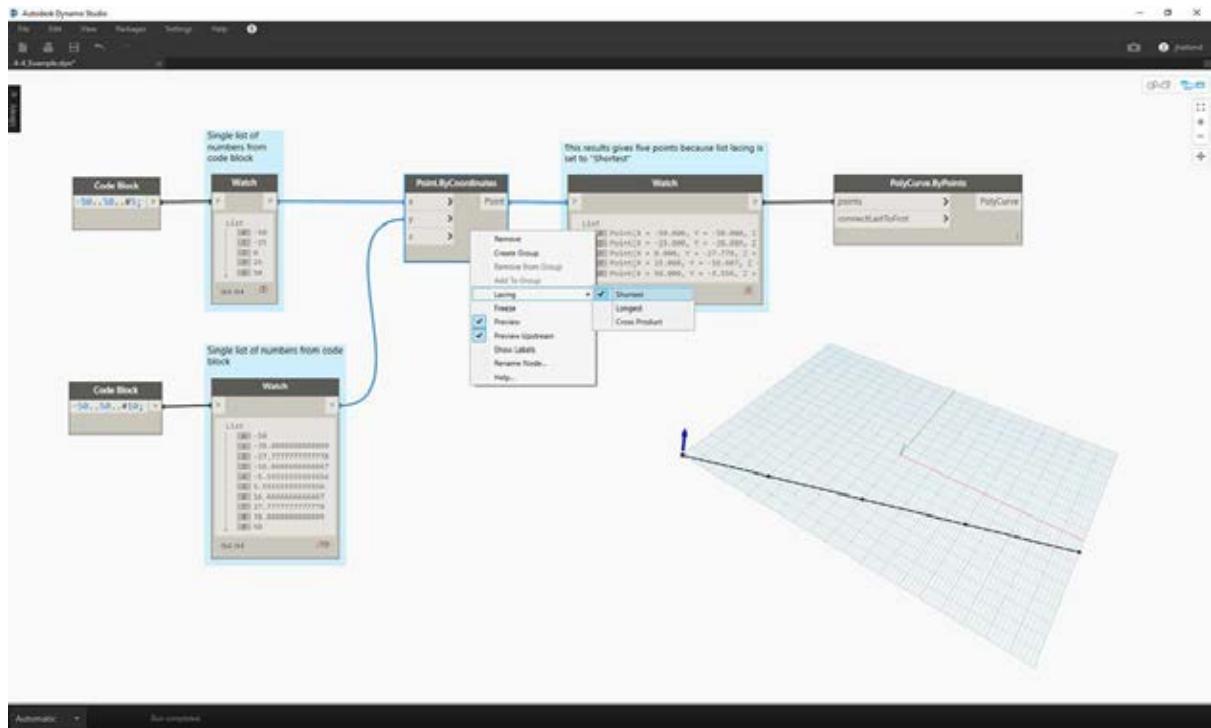


1. Dabei ändern Sie die Vergitterung für *Point.ByCoordinates*, nehmen jedoch keine weiteren Änderungen im oben gezeigten Diagramm vor.

### Kürzeste Liste

Die einfachste Möglichkeit besteht darin, jedem Wert genau einen Wert aus der anderen Eingabe zuzuordnen, bis das Ende einer der Folgen erreicht ist. Dieser Algorithmus wird als "Kürzeste Liste" bezeichnet. Dies ist das vorgegebene Verhalten in Dynamo-Blöcken:

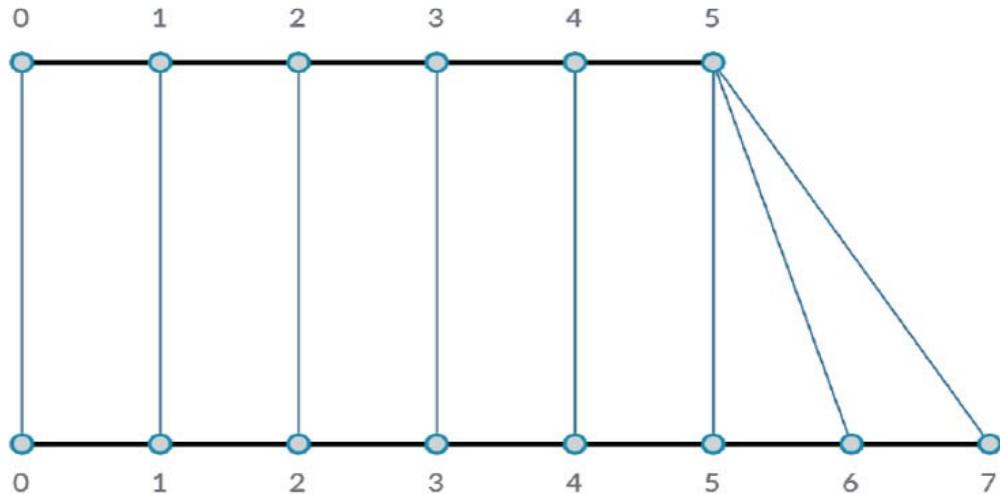


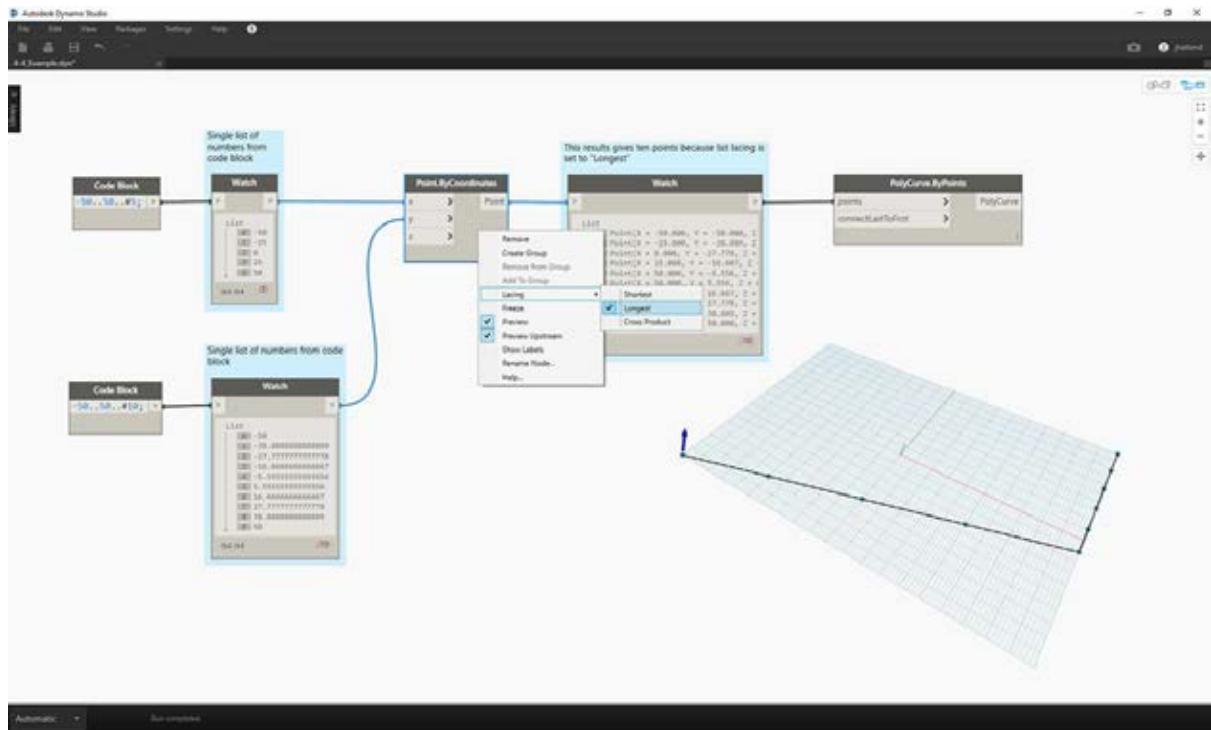


Mit der Vergitterung *Kürzeste Liste* erhalten Sie eine einfache diagonale Linie durch fünf Punkte. Die kürzere Liste umfasst fünf Einträge. Aus diesem Grund endet die Vergitterung *Kürzeste Liste*, sobald das Ende dieser Liste erreicht ist.

### Längste Liste

Der Algorithmus "Längste Liste" verbindet weiterhin Eingaben und verwendet gegebenenfalls Elemente mehrfach, bis alle Folgen aufgebraucht sind.

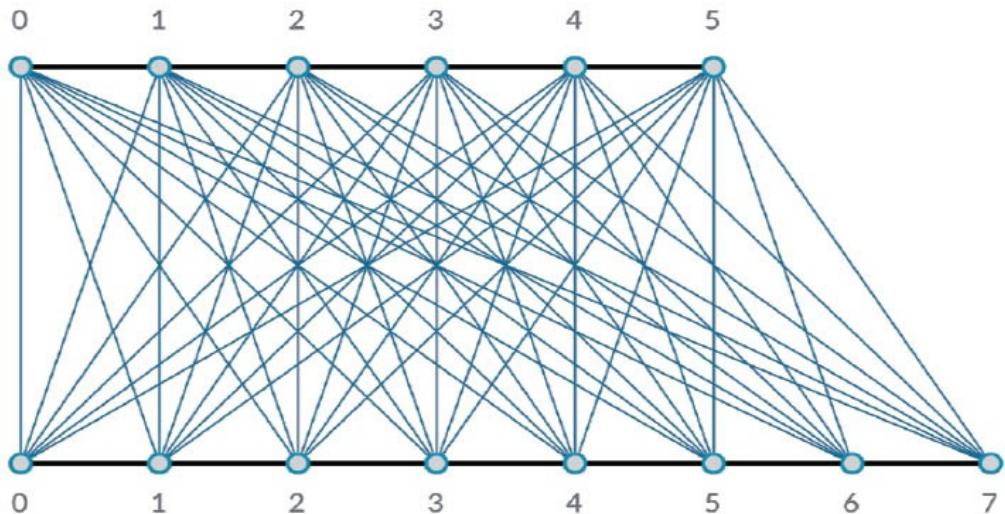


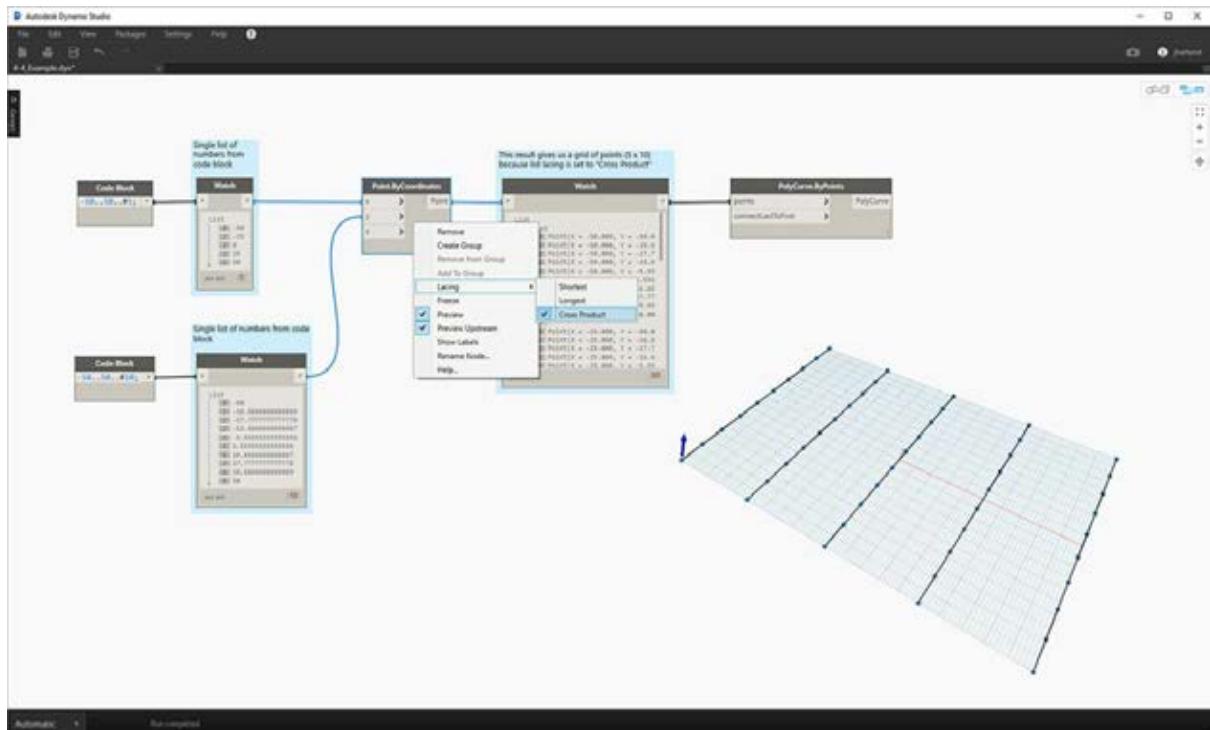


Mit der Vergitterung *Längste Liste* erhalten Sie eine diagonale Linie, die vertikal endet. Der letzte Eintrag in der 5 Einträge langen Liste wird genau wie im Übersichtsdiagramm so lange wiederholt, bis auch das Ende der längeren Liste erreicht ist.

### Kreuzprodukt

Mit der Methode "Kreuzprodukt" werden sämtliche möglichen Verbindungen hergestellt.





Bei der Vergitterung *Kreuzprodukt* erhalten Sie jede mögliche Kombination der beiden Listen. Dadurch entsteht ein Raster aus  $5 \times 10$  Punkten. Diese Datenstruktur entspricht der Darstellung des Kreuzprodukts im Übersichtsdiagramm oben, allerdings wurden die Daten dabei in eine Liste von Listen umgewandelt. Durch Verbinden einer PolyCurve wird sichtbar, dass jede Liste durch ihren x-Wert definiert ist. Damit entsteht eine Reihe mit fünf vertikalen Linien.

# Arbeiten mit Listen

## Arbeiten mit Listen

Im vorigen Kapitel wurde definiert, was unter einer Liste zu verstehen ist. Thema des folgenden Kapitels sind die Vorgänge, die für Listen durchgeführt werden können. Eine Liste lässt sich mit einem Stapel Spielkarten vergleichen. Der Stapel ist die Liste und jede Spielkarte steht für ein Element.



Foto von [Christian Gidlöf](#)

Welche **Abfragen** sind in der Liste möglich? Dies ermöglicht den Zugriff auf vorhandene Eigenschaften.

- Anzahl der Karten im Stapel? 52.
- Anzahl der Farben? 4.
- Material? Papier.
- Länge? 3.5" bzw. 89 mm.
- Breite? 2.5" bzw. 64 mm.

Welche **Aktionen** können für die Liste durchgeführt werden? Diese bewirken Änderungen in der Liste entsprechend der angegebenen Operation.

- Sie können den Stapel neu mischen.
- Sie können den Stapel nach Wert sortieren.
- Sie können den Stapel nach Farben sortieren.
- Sie können den Stapel teilen.
- Sie können den Stapel unterteilen, indem Sie die Karten an verschiedene Spieler ausgeben.
- Sie können eine bestimmte Karte im Stapel auswählen.

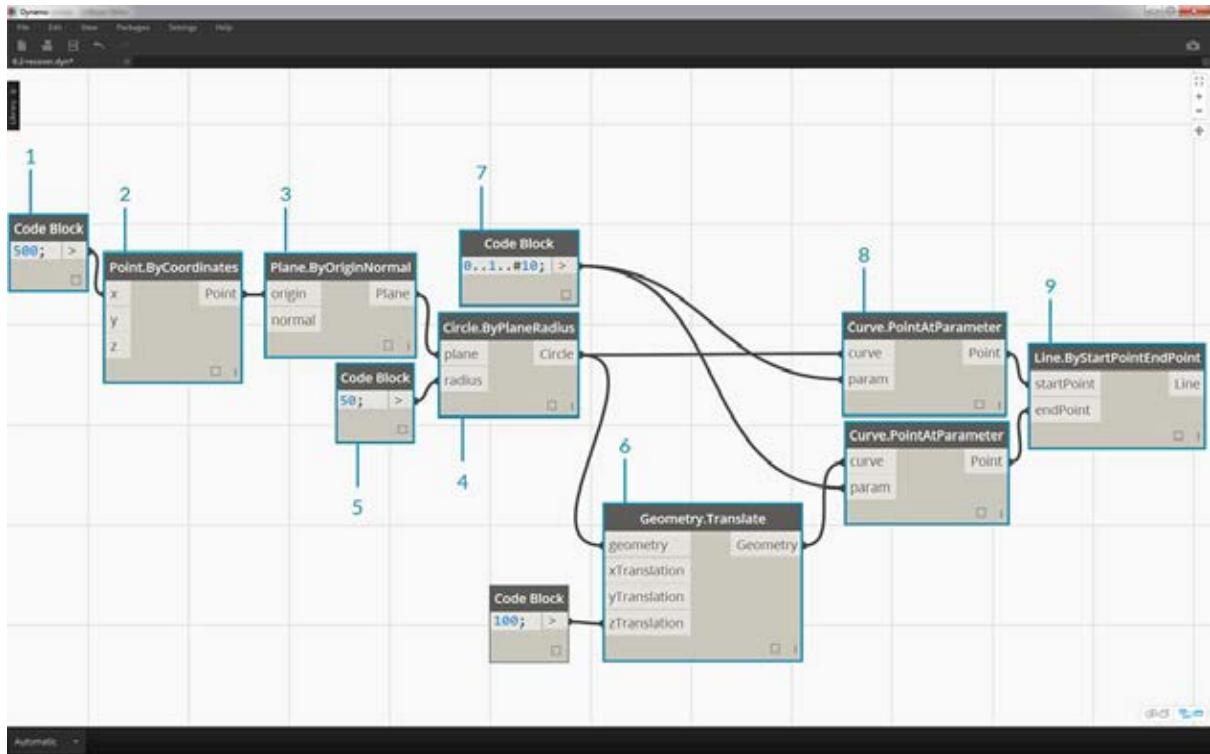
In Dynamo sind Blöcke vorhanden, die den oben genannten Vorgängen entsprechen und die Arbeit mit Listen von allgemeinen Daten ermöglichen. Die Lektionen weiter unten demonstrieren einige der grundlegenden Operationen, die für Listen durchgeführt werden können.

## Operationen für Listen

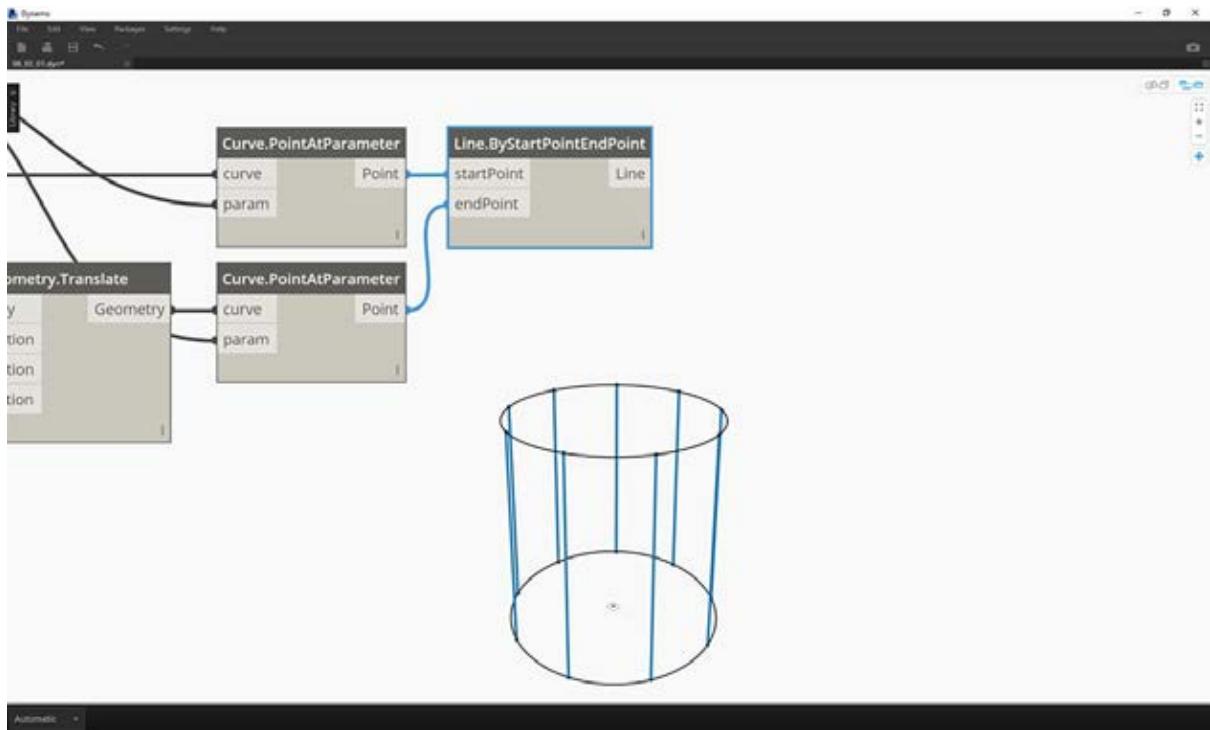
Das folgende Bild zeigt das Basisdiagramm, an dem grundlegende Operationen für Listen gezeigt werden. Dabei wird die Verwaltung von Daten in einer Liste betrachtet, und die visuellen Ergebnisse werden angezeigt.

### Übungslektion – Operationen für Listen

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [List-Operations.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

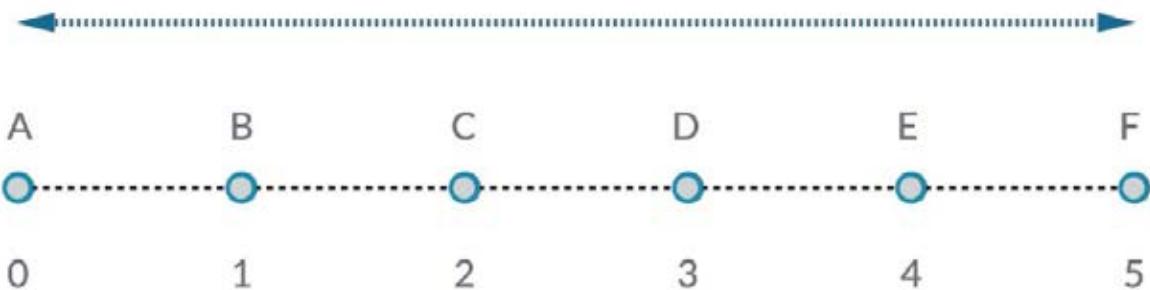


1. Beginnen Sie mit einem *Code Block* mit dem Wert 500.
2. Verbinden Sie ihn mit der x-Eingabe eines *Point.ByCoordinates*-Blocks.
3. Verbinden Sie den Block aus dem vorigen Schritt mit der origin-Eingabe eines *Plane.ByOriginNormal*-Blocks.
4. Verbinden Sie in einem *Circle.ByPlaneRadius*-Block den Block aus dem vorigen Schritt mit dem plane-Eingang.
5. Legen Sie mithilfe eines *Code Block* den Wert 50; als Radius fest. Dadurch erstellen Sie den ersten Kreis.
6. Verschieben Sie mithilfe eines *Geometry.Translate*-Blocks den Kreis um 100 Einheiten nach oben in z-Richtung.
7. Definieren Sie mithilfe eines *Code Block*-Blocks einen Bereich von 10 Zahlen zwischen 0 und 1, wobei Sie die folgende Zeile verwenden: `0..1..#10;`.
8. Verbinden Sie den Codeblock aus dem vorigen Schritt mit der param-Eingabe zweier *Curve.PointAtParameter*-Blöcke. Verbinden Sie *Circle.ByPlaneRadius* mit der curve-Eingabe des oberen Blocks und *Geometry.Translate* mit der curve-Eingabe des Blocks darunter.
9. Verbinden Sie in einem *Line.ByStartPointEndPoint* die beiden *Curve.PointAtParameter*-Blöcke.



1. Ein *Watch3D*-Block zeigt die Ergebnisse aus *Line.ByStartPointEndPoint* an. Hier werden Linien zwischen zwei Kreisen gezeichnet, um einfache Listenoperationen darzustellen. Dieses Dynamo-Basisdiagramm wird zur Demonstration der unten genannten Listenoperationen verwendet.

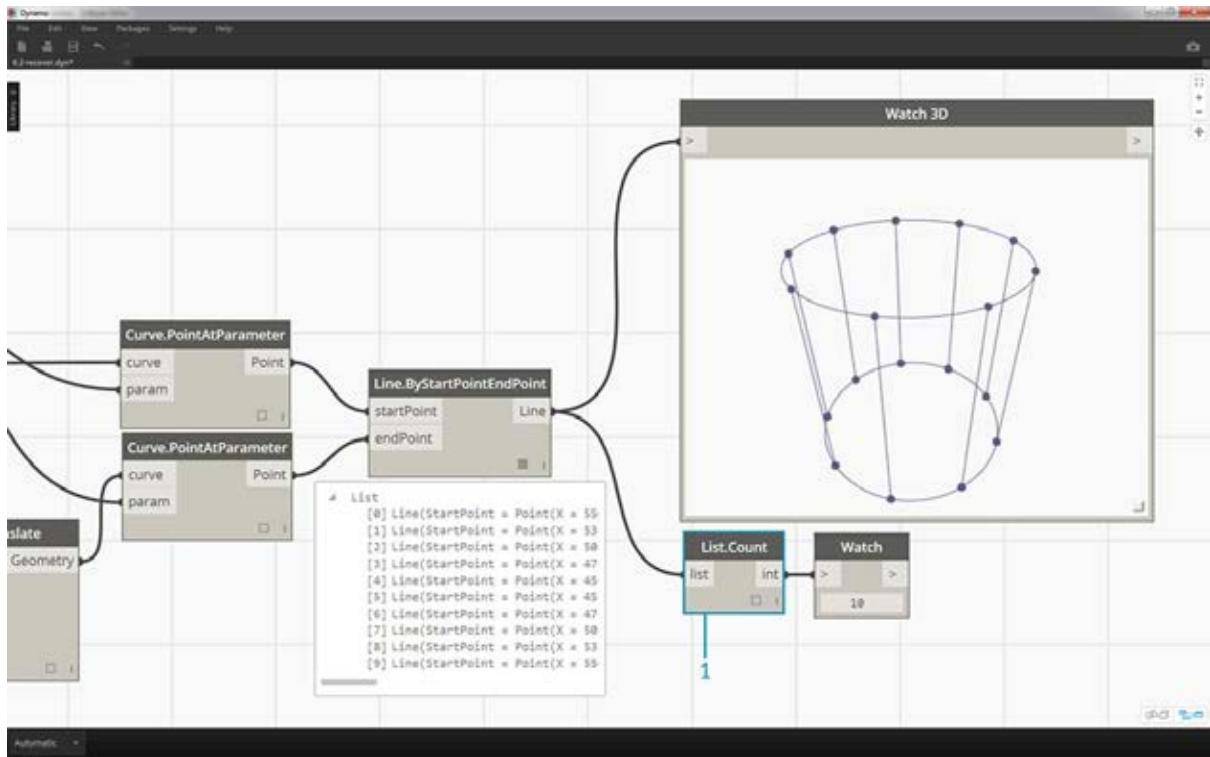
### List.Count



Der Block *List.Count* ist einfach: Er zählt die in einer Liste enthaltenen Werte und gibt ihre Anzahl zurück. Bei der Arbeit mit Listen von Listen gestaltet sich die Verwendung dieses Blocks differenzierter. Dies wird in weiter unten folgenden Abschnitten gezeigt.

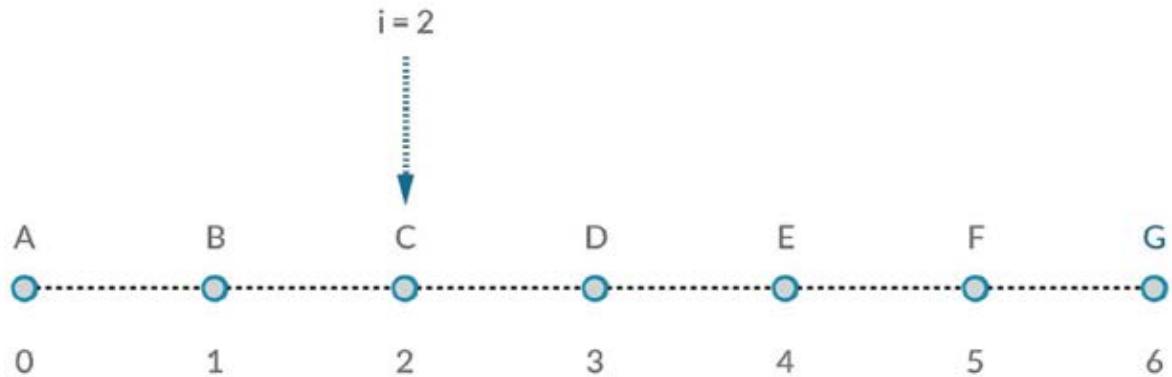
### Übungslektion – List.Count

Laden Sie die zu dieser Übungslektion gehörige Beispielfdatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [List-Count.dyn](#). Eine vollständige Liste der Beispielfdateien finden Sie im Anhang.



- Der *List.Count*-Block gibt die Anzahl der Linien aus dem *Line.ByStartPointEndPoint*-Block zurück. In diesem Fall beträgt dieser Wert 10. Dies stimmt mit der Anzahl der Punkte überein, die mithilfe des ursprünglichen *Code Block*-Blocks erstellt wurden.

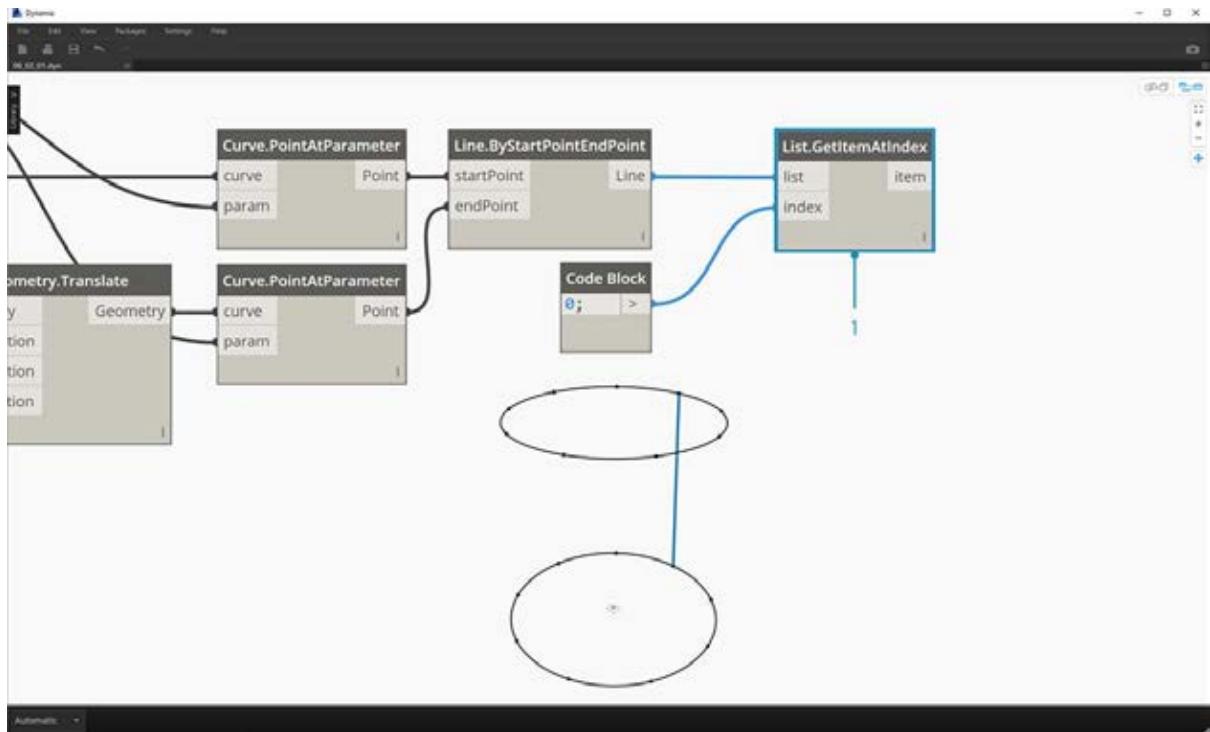
### List.GetItemAtIndex



*List.GetItemAtIndex* ist ein grundlegendes Verfahren zum Abrufen von Elementen in der Liste. In der Abbildung oben wird mithilfe des Indexwerts "2" der Punkt mit der Bezeichnung "C" abgerufen.

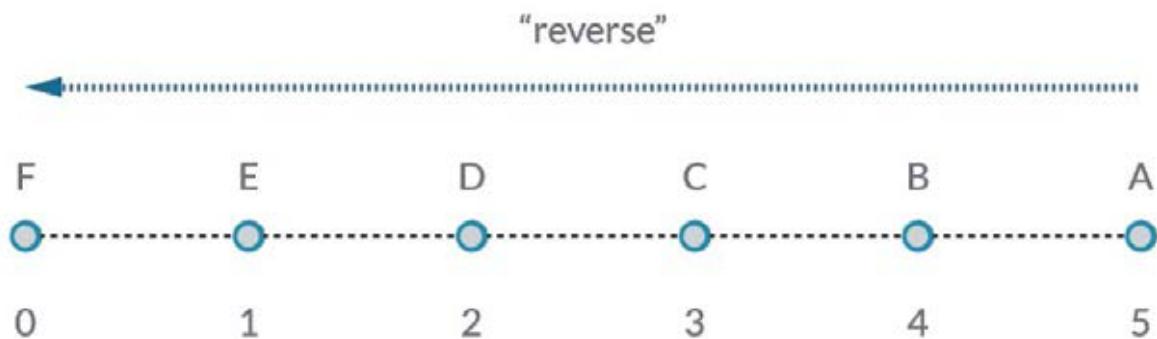
### Übungslektion – List.GetItemAtIndex

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [List-GetItemAtIndex.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



1. Mithilfe von *List.GetItemAtIndex* wird der Index "0" bzw. das erste Element in der Liste der Linien ausgewählt.
2. Der *Watch3D*-Block zeigt an, dass genau eine Linie ausgewählt wurde. Anmerkung: Um die Abbildung oben zu erhalten, müssen Sie die Vorschau von *Line.ByStartPointEndPoint* deaktivieren.

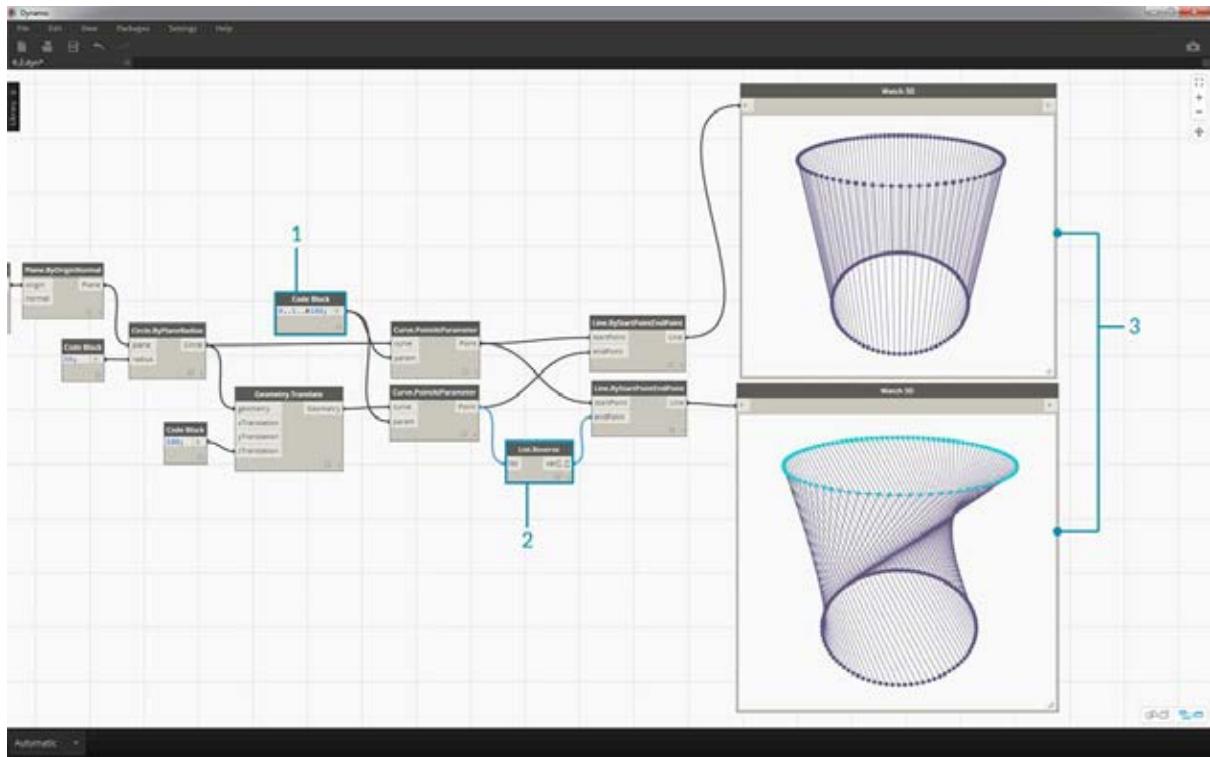
### List.Reverse



*List.Reverse* kehrt die Reihenfolge aller Elemente in der Liste um.

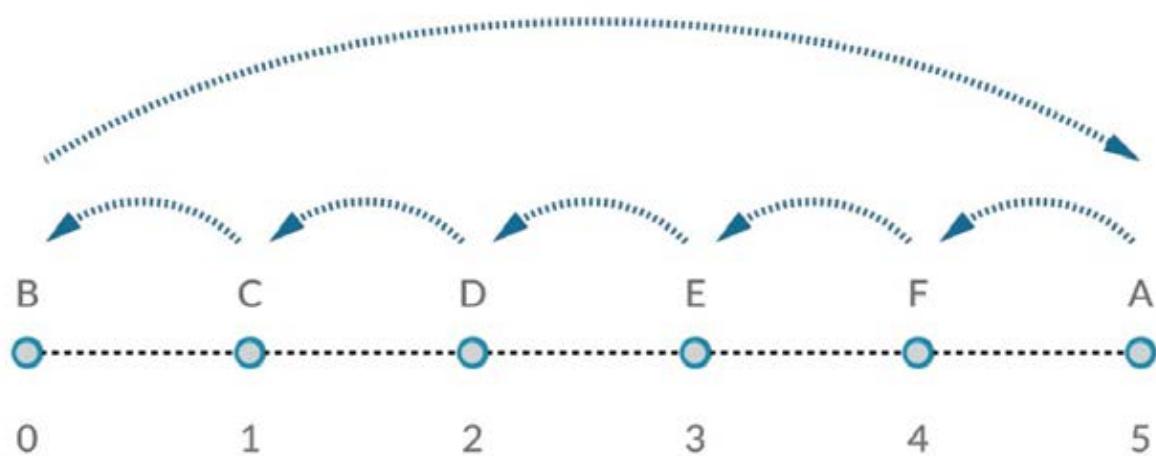
### Übungslektion – List.Reverse

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [List-Reverse.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



1. Erstellen Sie für eine deutlichere Darstellung der Linienliste in umgekehrter Reihenfolge weitere Linien, indem Sie den Codeblock in `0..1..#100`; ändern.
2. Fügen Sie einen `List.Reverse`-Block zwischen `Curve.PointAtParameter` und `Line.ByStartPointEndPoint` für eine der beiden Punktlisten ein.
3. Die `Watch3D`-Blöcke zeigen zwei unterschiedliche Ergebnisse. Der erste zeigt das Ergebnis ohne umgekehrte Liste. Die Linien verlaufen vertikal und verbinden benachbarte Punkte. Die Umkehrung einer der Listen bewirkt jedoch, dass alle Punkte in entgegengesetzter Reihenfolge mit Punkten in der anderen Liste verbunden werden.

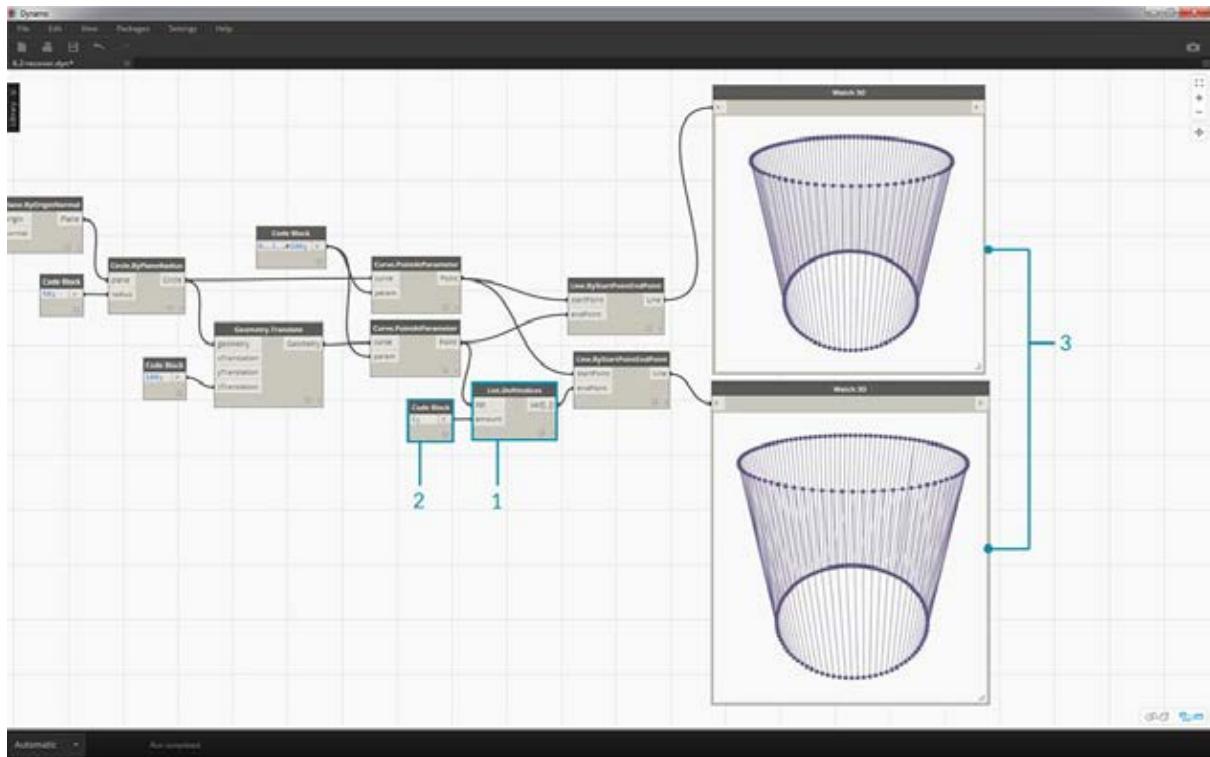
### List.ShiftIndices



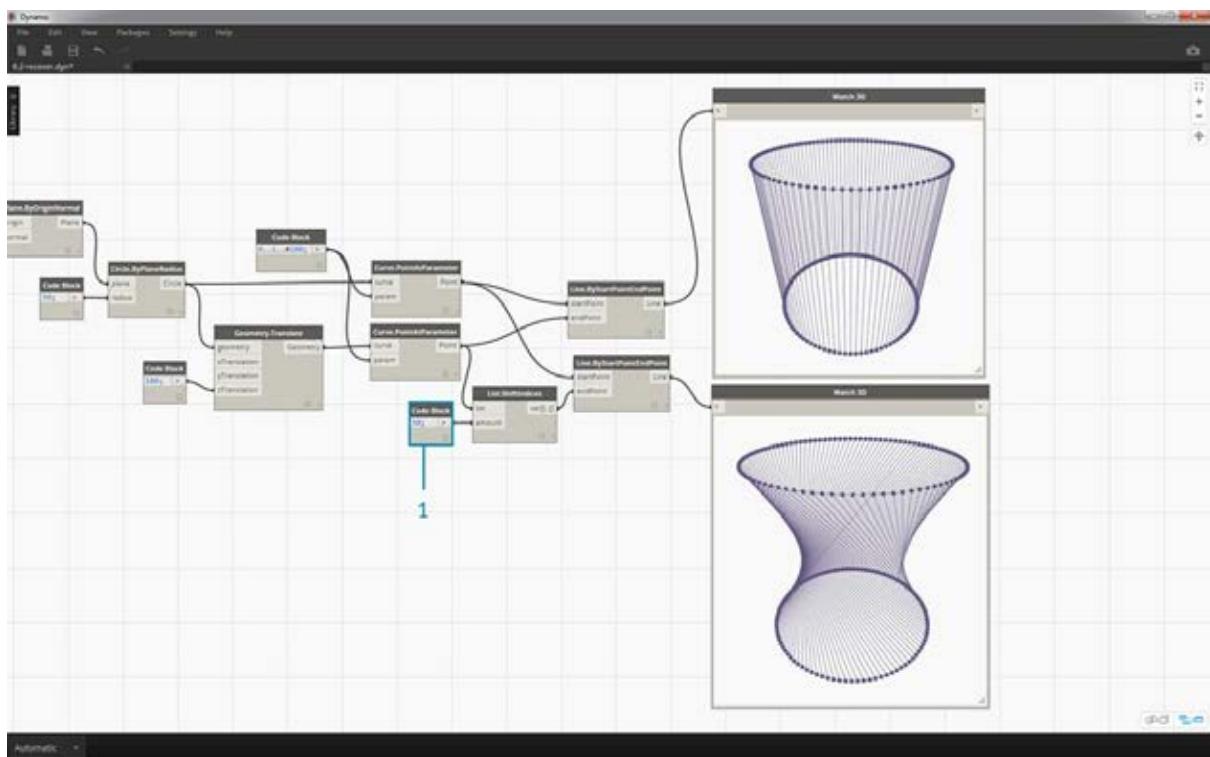
`List.ShiftIndices` ist ein geeignetes Werkzeug zum Erstellen verdrehter oder schraubenförmiger Muster oder für ähnliche Datenverarbeitungen. Dieser Block verschiebt die Elemente in einer Liste um die angegebene Anzahl von Indexpositionen.

### Übungslektion – List.ShiftIndices

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [List-ShiftIndices.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

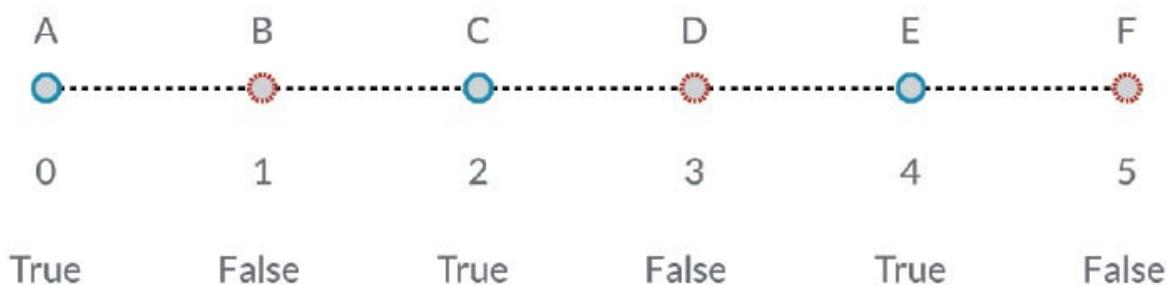


1. Fügen Sie auf dieselbe Weise wie beim Umkehren der Liste einen *List.ShiftIndices*-Block zwischen *Curve.PointAtParameter* und *Line.ByStartPointEndPoint* ein.
2. Verwenden Sie einen *Code Block*, dem Sie den Wert "1" zuweisen, zum Verschieben der Liste um eine Indexposition.
3. Sie erhalten keine extreme Veränderung, aber sämtliche Linien im unteren *Watch3D*-Block wurden bei der Verbindung mit der anderen Punktgruppe um eine Indexposition versetzt.



1. Wenn Sie im *Code Block* einen größeren Wert, z. B. "30" festlegen, ist ein deutlicher Unterschied in den diagonalen Linien zu erkennen. Die Verschiebung hat in diesem Fall dieselbe Wirkung wie die Irisblende einer Kamera und bewirkt eine Verdrehung der ursprünglichen Zylinderform.

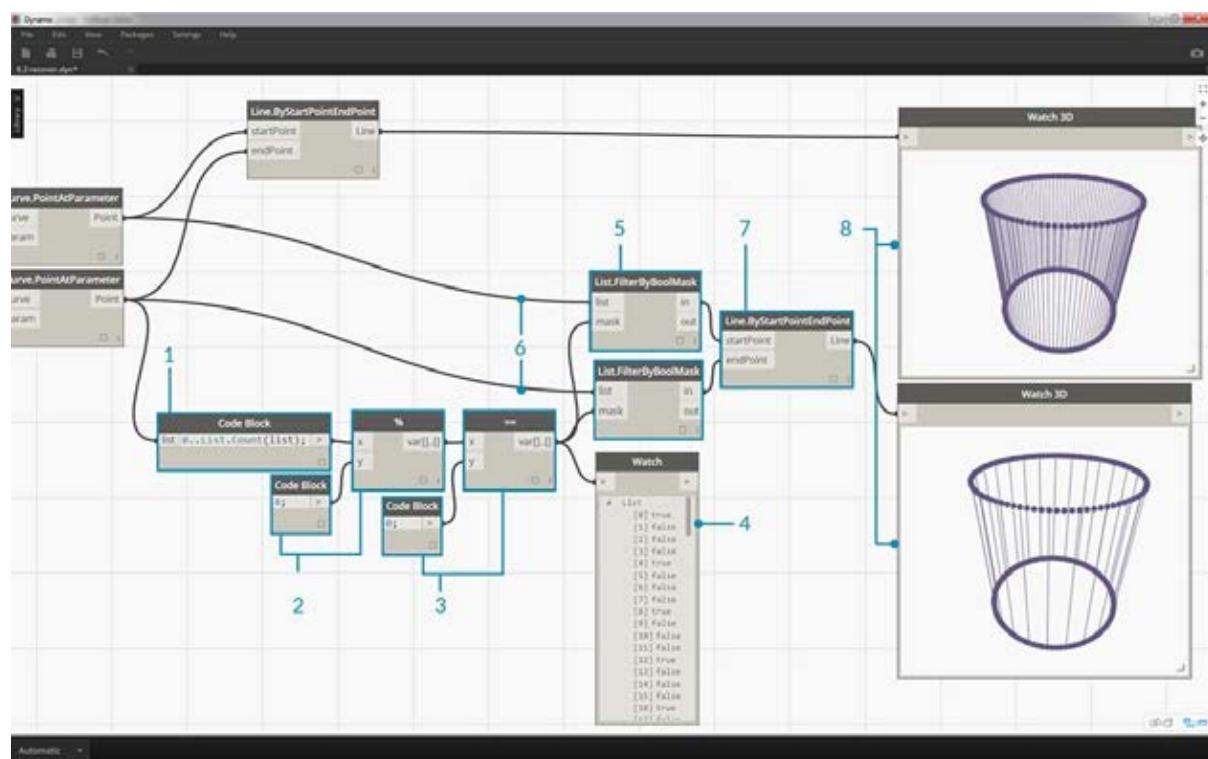
## List.FilterByBooleanMask



*List.FilterByBooleanMask* entfernt bestimmte Elemente anhand einer Liste boolescher Werte bzw. der Werte "true" oder "false".

### Übungslektion – List.FilterByBooleanMask

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As): [List-FilterByBooleanMask.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



Um eine Liste mit true- und false-Werten zu erstellen, sind einige weitere Schritte erforderlich.

1. Definieren Sie mithilfe eines *Code Block* einen Ausdruck mit der folgenden Syntax:  
0..List.Count(list);. Verbinden Sie den *Curve.PointAtParameter*-Block mit der *list*-Eingabe. Diese Einrichtung wird im Kapitel zu Codeblöcken genauer behandelt. In diesem Fall erhalten Sie mit dieser Codezeile eine Liste mit sämtlichen Indizes aus dem *Curve.PointAtParameter*-Block.
2. Fügen Sie einen Modulo-Block ("%"") ein und verbinden Sie die Ausgabe des *Code Block* mit der *x*-Eingabe und den Wert 4 mit der *y*-Eingabe. Dadurch erhalten Sie den Rest bei der Division der Indizes in der Liste durch 4. Die Modulo-Funktion ist sehr hilfreich beim Erstellen von Mustern. Alle Werte werden als mögliche Reste für 4 ausgegeben: 0, 1, 2, 3.
3. Aus dem *Modulo*-Block ergibt sich, dass Indizes mit dem Wert 0 durch 4 teilbar sind (0, 4, 8...). Mithilfe eines " $=$ "-Blocks kann die Teilbarkeit durch Vergleich mit dem Wert 0 geprüft werden.
4. Der *Watch*-Block zeigt genau dieses Ergebnis, d. h. das folgende true/false-Muster: *true,false,true,false,...*
5. Verbinden Sie die Ausgabe mit diesem true/false-Muster mit der *mask*-Eingabe zweier *List.FilterByBooleanMask*-Blöcke.
6. Verbinden Sie jeweils den *Curve.PointAtParameter*-Block mit der *list*-Eingaben des

*List.FilterByBooleanMask*-Blocks.

7. Die Ausgaben von *Filter.ByBooleanMask* lauten "in" und "out". "In" steht für Werte mit dem Maskenwert "true", "out" für Werte mit dem Wert "false". Indem Sie die "in"-Ausgaben mit den *startPoint*- und *endPoint*-Eingaben eines *Line.ByStartPointEndPoint*-Blocks verbinden, erhalten Sie eine gefilterte Liste von Linien.
8. Im *Watch3D*-Block ist zu erkennen, dass weniger Linien als Punkte vorhanden sind. Durch Filtern ausschließlich der true-Werte wurden lediglich 25 % der Punkte ausgewählt.

# Listen von Listen

## Listen von Listen

In diesem Abschnitt kommt eine weitere Ebene zur Hierarchie hinzu. Für den Einstieg wurde ein Stapel Karten als Beispiel betrachtet. Angenommen, eine Schachtel enthält mehrere solche Stapel. In diesem Fall entspricht die Schachtel einer Liste von Stapeln und jeder Stapel einer Liste von Karten. Dies ist eine Liste von Listen. Als Analogie für diesen Abschnitt dient die unten gezeigte rote Kiste eine Liste von Münzstapeln, wobei jeder Stapel einer Liste von Münzen entspricht.



Foto von [Dori](#).

Welche **Abfragen** sind in der Liste von Listen möglich? Dies ermöglicht den Zugriff auf vorhandene Eigenschaften.

- Anzahl von Münzarten? 2.
- Wert der Münzarten? 0,01 \$ und 0,25 \$.
- Material der Vierteldollars? 75 % Kupfer und 25 % Nickel.
- Material der Cents? 97,5 % Zink und 2,5 % Kupfer.

Welche **Aktionen** können in der Liste von Listen durchgeführt werden? Diese Aktionen bewirken Änderungen in der Liste von Listen entsprechend der jeweiligen Operation.

- Wählen Sie einen bestimmten Stapel von Vierteldollars oder Cents aus.
- Wählen Sie einen bestimmten Vierteldollar oder Cent aus.
- Ordnen Sie den Stapel aus Vierteldollars und Cents neu.
- Mischen Sie die Stapel zusammen.

Auch in diesem Fall steht in Dynamo für jeden der oben genannten Vorgänge ein entsprechender Block zur Verfügung. Da allerdings keine realen Objekte, sondern abstrakte Daten verarbeitet werden, sind Regeln zum Navigieren in der Datenhierarchie erforderlich.

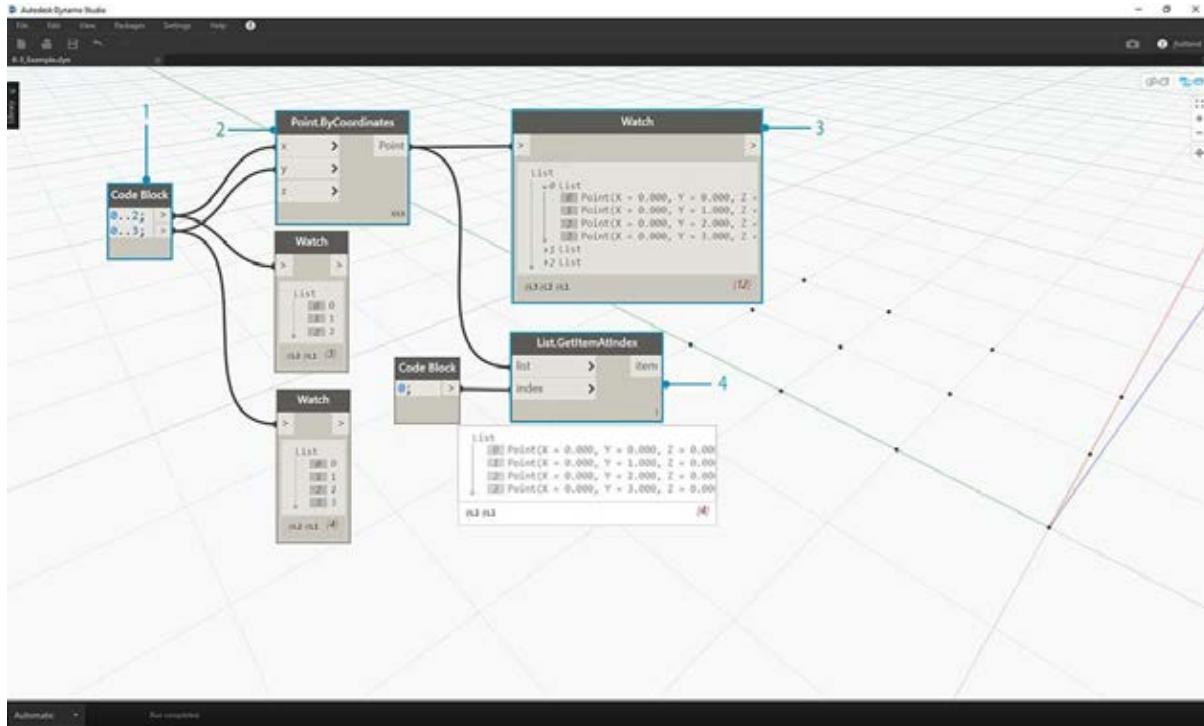
Die bei der Arbeit mit Listen von Listen verwendeten Daten sind komplex und umfassen mehrere Ebenen, was aber auch die Möglichkeit für äußerst wirkungsvolle parametrische Operationen eröffnet. Im Folgenden werden zunächst die Grundlagen im Einzelnen erläutert und in weiteren Lektionen einige weitere Operationen betrachtet.

## Hierarchie von oben nach unten

Das in diesem Abschnitt eingeführte Grundprinzip lautet: **Dynamo behandelt auch Listen als Objekte**. Diese Hierarchie von oben nach unten wurde im Hinblick auf objektorientierte Programmierung entwickelt. Anstatt Unterelemente mithilfe eines Befehls wie etwa List.GetItemAtIndex auszuwählen, wählt Dynamo den entsprechenden Index in der Hauptliste innerhalb der Datenstruktur. Dieses Element kann eine weitere Liste sein. Ein Beispieldbild soll dies verdeutlichen:

### Übung – Hierarchie von oben nach unten

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Top-Down-Hierarchy.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



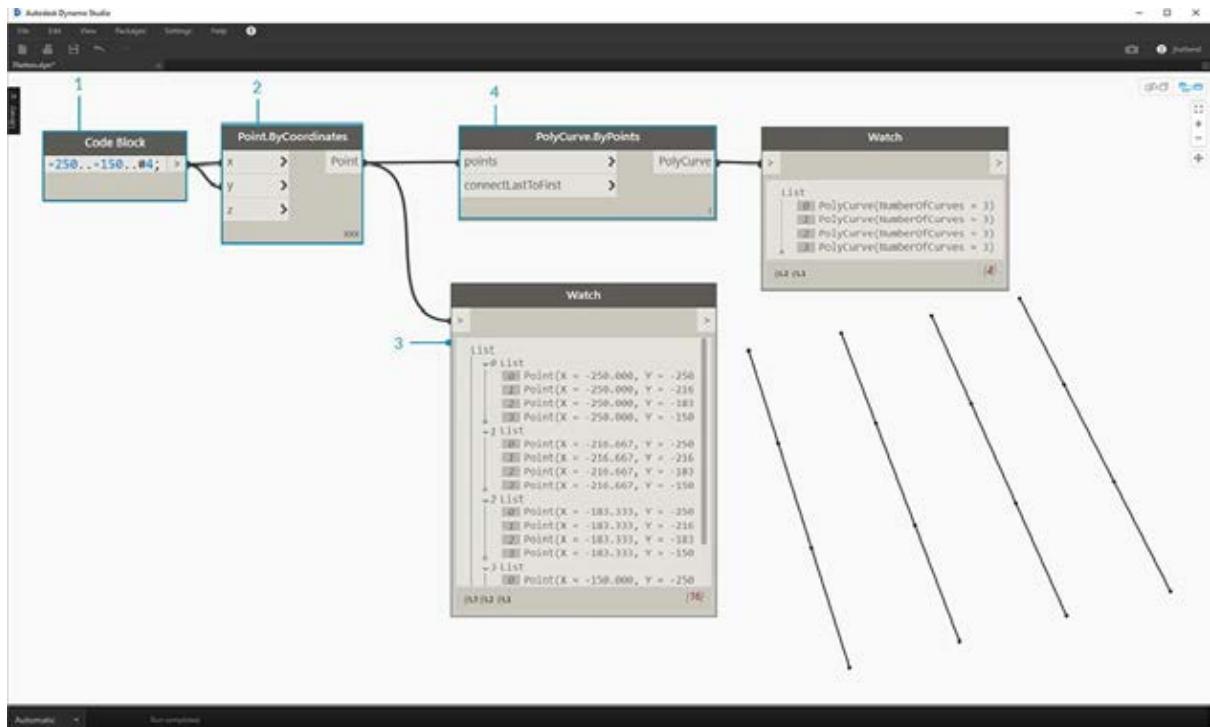
1. Im *Codeblock* wurden zwei Bereiche definiert: `0..2; 0..3;`
2. Diese Bereiche sind mit einem *Point.ByCoordinates*-Block unter Verwendung der Vergitterung "Kreuzprodukt" verbunden. Dadurch wird ein Raster von Punkten erstellt und zugleich wird eine Liste von Listen als Ausgabe zurückgegeben.
3. Beachten Sie, dass im *Watch*-Block 3 Listen mit jeweils 4 Elementen angezeigt werden.
4. Bei Verwendung von *List.GetItemAtIndex* mit dem Index 0 wählt Dynamo die erste Liste sowie ihren gesamten Inhalt aus. In anderen Programmen wird eventuell das erste Element jeder Liste in der Datenstruktur ausgewählt, Dynamo verwendet jedoch eine von oben nach unten geordnete Hierarchie zur Verarbeitung der Daten.

### Flatten und List.Flatten

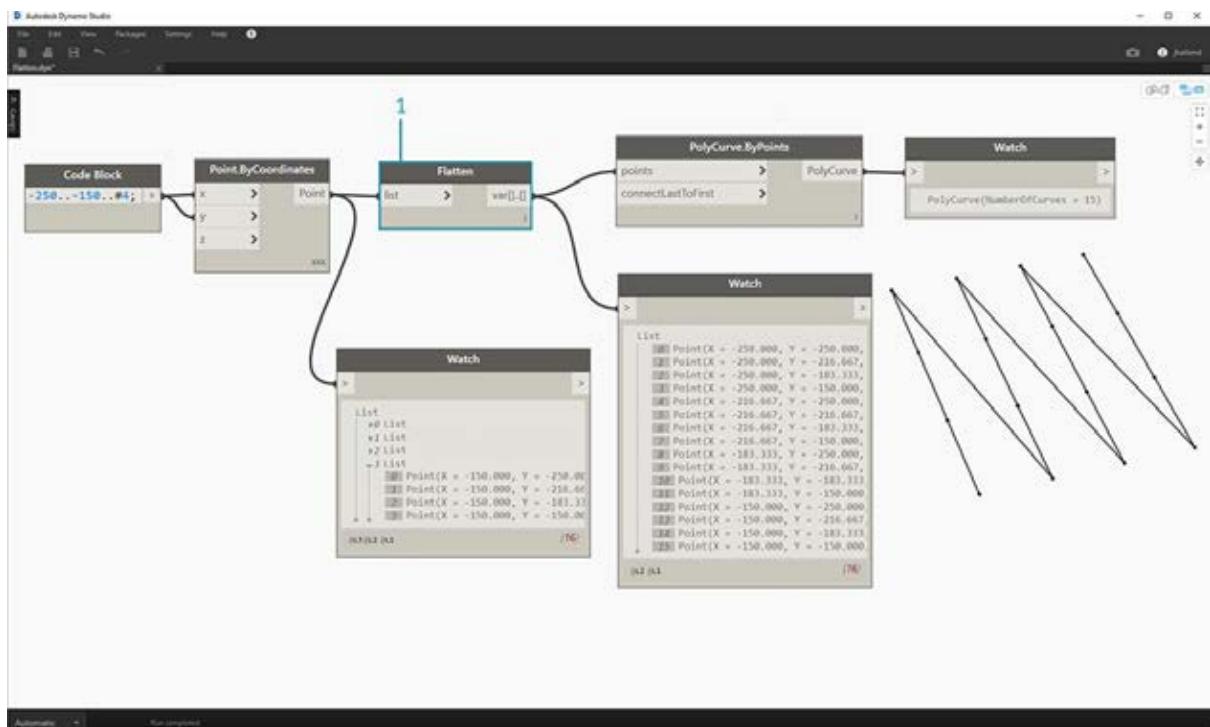
Der Befehl Flatten entfernt alle Datenebenen aus einer Datenstruktur. Dies ist hilfreich, wenn Sie für Ihre Operationen keine Datenhierarchien benötigen, birgt jedoch auch Risiken, da Informationen entfernt werden. Das folgende Beispiel zeigt das Ergebnis nach der Vereinfachung einer Liste mit Daten.

### Übungslektion – Flatten

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Flatten.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



1. Fügen Sie im *Codeblock* eine Zeile mit Code zum Definieren eines Bereichs ein: `-250.. -150.. #4;`
2. Der *Codeblock* wird mit den *x*- und *y*-Eingaben eines *Point.ByCoordinates*-Blocks verbunden, wobei die Vergitterung "Kreuzprodukt" verwendet wird, um ein Raster aus Punkten zu erhalten.
3. Im *Watch*-Block wird angezeigt, dass eine Liste von Listen erstellt wurde.
4. Ein *PolyCurve.ByPoints*-Block referenziert die einzelnen Listen und erstellt die entsprechenden Polykurven. Die Dynamo-Vorschau zeigt vier Polykurven für die Zeilen im Raster.



1. Durch Einfügen von *flatten* vor dem Block für die Polykurven entsteht eine einzelne Liste, die sämtliche Punkte enthält. Der Block für die Polykurven referenziert diese Liste und erstellt nur eine Kurve. Da alle Punkte in derselben Liste enthalten sind, entsteht eine zickzackförmige Polykurve, die durch sämtliche Punkten aus der Liste verläuft.

Darüber hinaus stehen auch Optionen zum Vereinfachen isolierter Datenebenen zur Verfügung. Mithilfe des Blocks

List.Flatten können Sie festlegen, wie viele Datenebenen unterhalb der ersten Hierarchieebene vereinfacht werden sollen. Dies ist sehr hilfreich bei komplexen Datenstrukturen, die für Ihren Arbeitsablauf nicht unbedingt erforderlich sind. Eine weitere Möglichkeit besteht darin, den Flatten-Block als Funktion in List.Map einzusetzen. [List.Map](#) wird weiter unten ausführlicher beschrieben.

## Chop

Bei der parametrischen Modellierung ist es zuweilen erforderlich, die Datenstruktur einer bestehenden Liste präziser zu differenzieren. Hierfür stehen ebenfalls zahlreiche Blöcke zur Verfügung, wobei Chop die einfachste Version darstellt. Mit Chop können Sie eine Liste in Unterlisten mit der angegebenen Anzahl Elemente unterteilen.

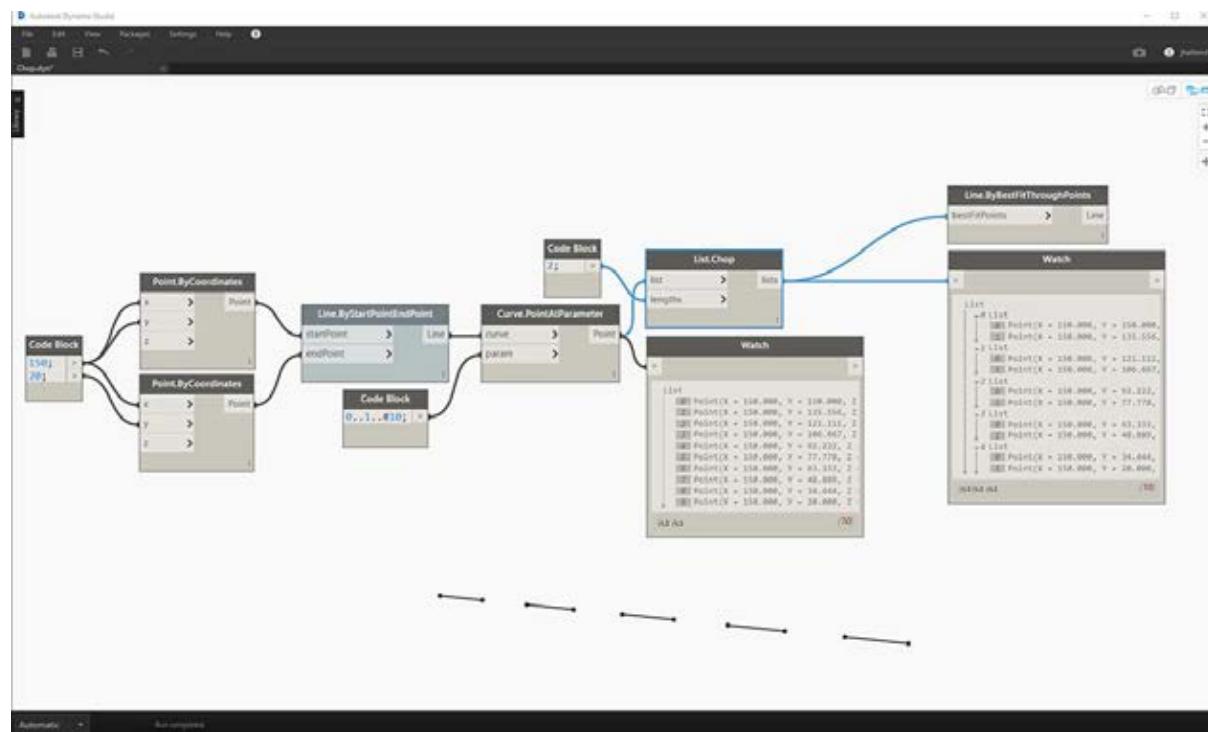
### Übungslektion – List.Chop

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Chop.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im [Appendix](#).



Mit *List.Chop* und dem *\_subLength*-Wert 2 entstehen vier Listen mit je zwei Elementen.

Der Befehl Chop teilt Listen gemäß der angegebenen Listenlänge. In gewisser Weise stellt Chop das Gegenteil zu Flatten dar: Der Datenstruktur werden neue Ebenen hinzugefügt, anstatt dass vorhandene entfernt werden. Dieses Werkzeug ist hilfreich für geometrische Operationen wie im Beispiel unten gezeigt.



## List.Map und List.Combine

Mit List.Map/Combine wird eine angegebene Funktion auf die eingegebene Liste angewendet, allerdings geschieht dies eine Ebene tiefer in der Hierarchie. Kombinationen entsprechen Maps, wobei für Kombinationen allerdings mehrere Eingaben entsprechend der Eingabe einer gegebenen Funktion möglich sind.

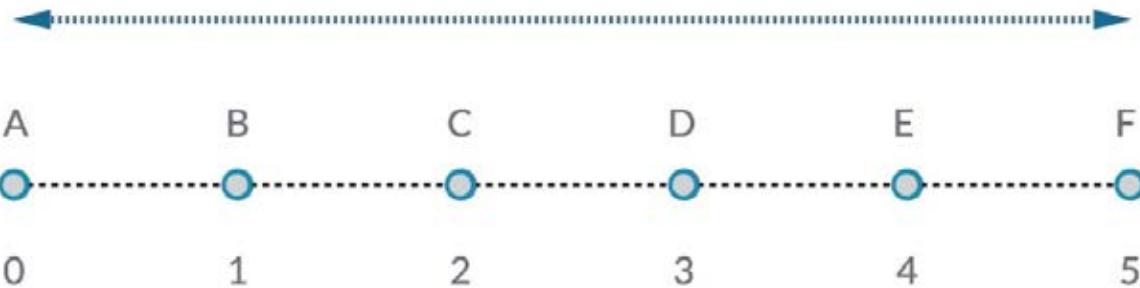
### Übungslektion – List.Map

Anmerkung: Diese Übung wurde mit einer früheren Version von Dynamo erstellt. Die Funktion List.Map lässt sich großenteils durch die neu hinzugefügte Funktion List@Level ersetzen. Weitere Informationen finden Sie weiter unten unter

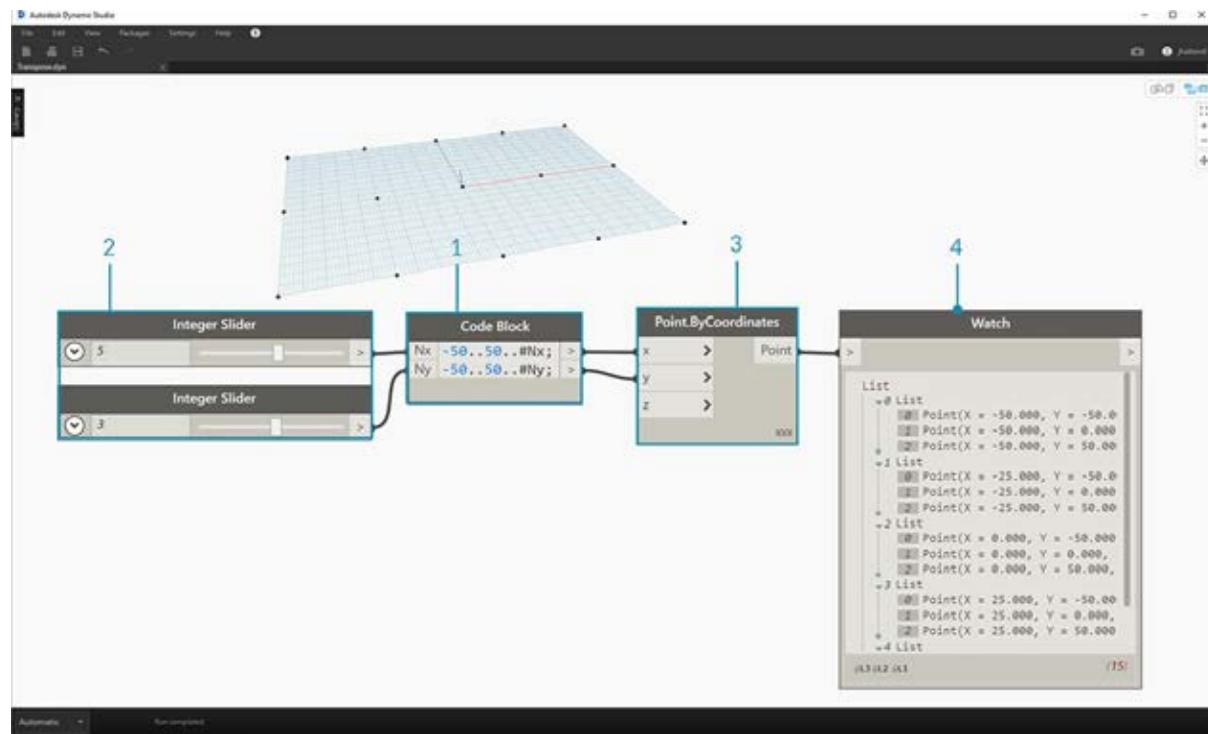
## List@Level.

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Map.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im [Appendix](#).

Ziehen Sie als kurze Einführung den List.Count-Block aus dem vorigen Abschnitt heran.



Der *List.Count*-Block zählt alle Elemente in einer Liste. An diesem Beispiel wird die Funktionsweise von *List.Map* gezeigt.

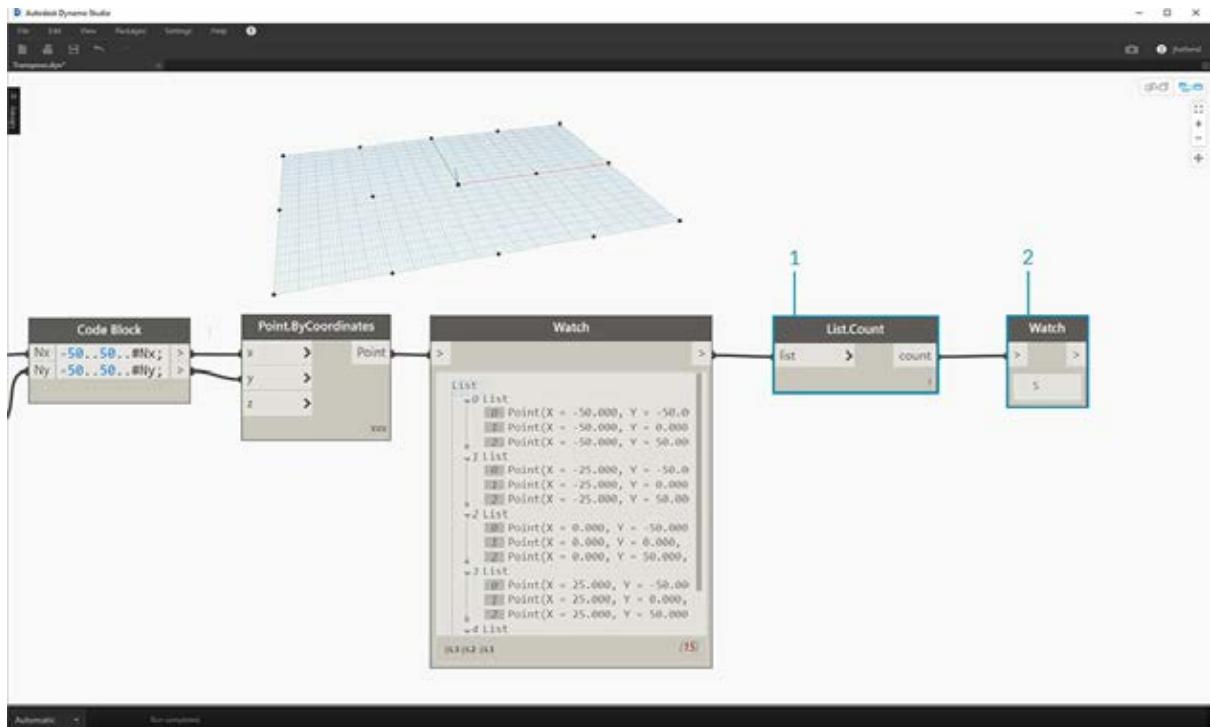


1. Fügen Sie zwei Zeilen Code in den *Codeblock* ein:

```
-50..50..#Nx;  
-50..50..#Ny;
```

Nach der Eingabe dieses Codes werden im *Codeblock* zwei Eingaben für Nx und Ny erstellt.

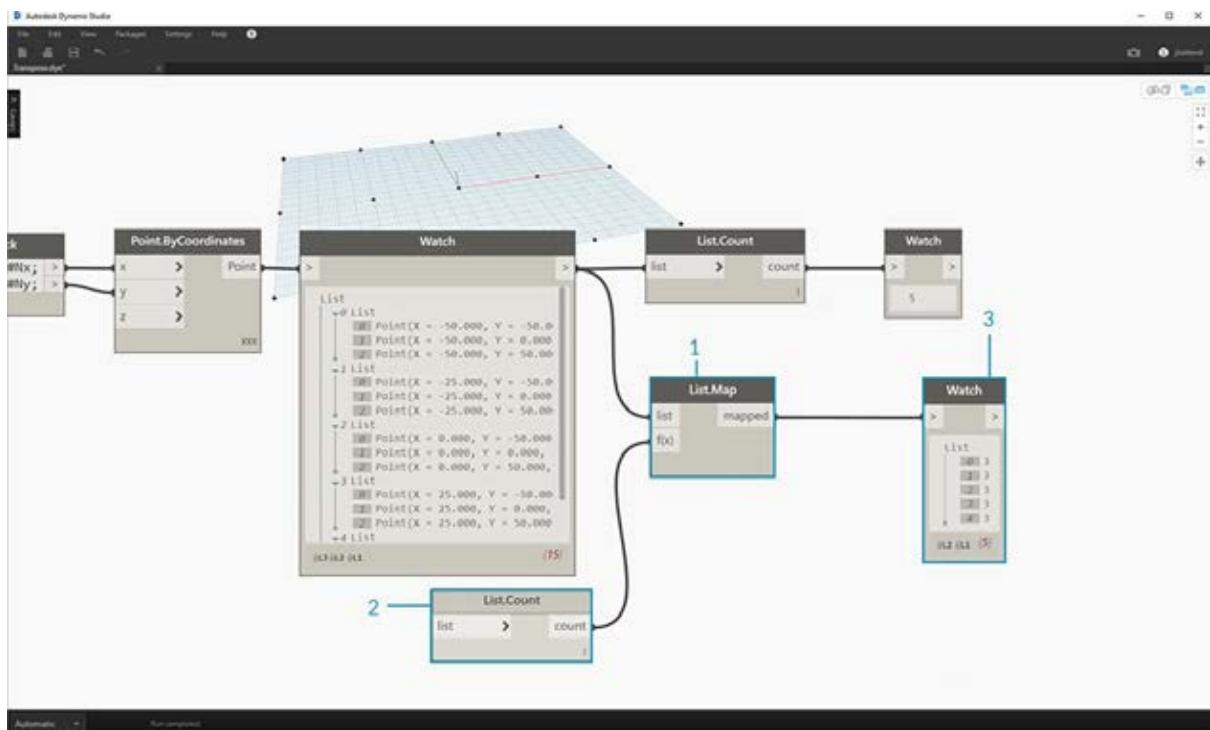
1. Definieren Sie mit zwei *Integer Slidern* die Werte für Nx und Ny, indem Sie die Schiebereglern mit dem *Codeblock* verbinden.
2. Verbinden Sie jede Zeile des *Codeblocks* mit der entsprechenden X- bzw. Y-Eingabe eines *Point.ByCoordinates*-Blocks. Klicken Sie mit der rechten Maustaste auf den Block und > wählen Sie zuerst Vergitterung und dann "Kreuzprodukt". Dadurch wird ein Raster aus Punkten erstellt. Da der Bereich zwischen -50 und 50 definiert wurde, umfasst er das vorgabemäßige Raster von Dynamo.
3. In einem *Watch*-Block werden die erstellten Punkte angezeigt. Beachten Sie die Datenstruktur. Es wurde eine Liste von Listen erstellt. Jede Liste entspricht einer Reihe von Punkten im Raster.



1. Verbinden Sie einen *List.Count*-Block mit der Ausgabe des *Watch*-Blocks aus dem vorigen Schritt.
2. Verbinden Sie einen *Watch*-Block mit der Ausgabe des *List.Count*-Blocks.

Der *List.Count*-Block gibt den Wert 5 aus. Dieser Wert ist gleich dem Wert der im Codeblock definierten Variablen Nx. Dies geschieht aus dem folgenden Grund:

- Im *Point.ByCoordinates*-Block wird die x-Eingabe als primäre Eingabe zum Erstellen von Listen verwendet. Für Nx = 5 und Ny = 3 ergibt sich daher eine Liste, die fünf Listen mit je 3 Elementen enthält.
- Da Listen in Dynamo als Objekte behandelt werden, wird der *List.Count*-Block auf die Hauptliste in der Hierarchie angewendet. Das Ergebnis ist der Wert 5, d. h. die Anzahl der Listen in der Hauptliste.



1. Mithilfe eines *List.Map*-Blocks bewegen Sie sich eine Ebene tiefer in die Hierarchie und führen dort eine "Funktion" aus.

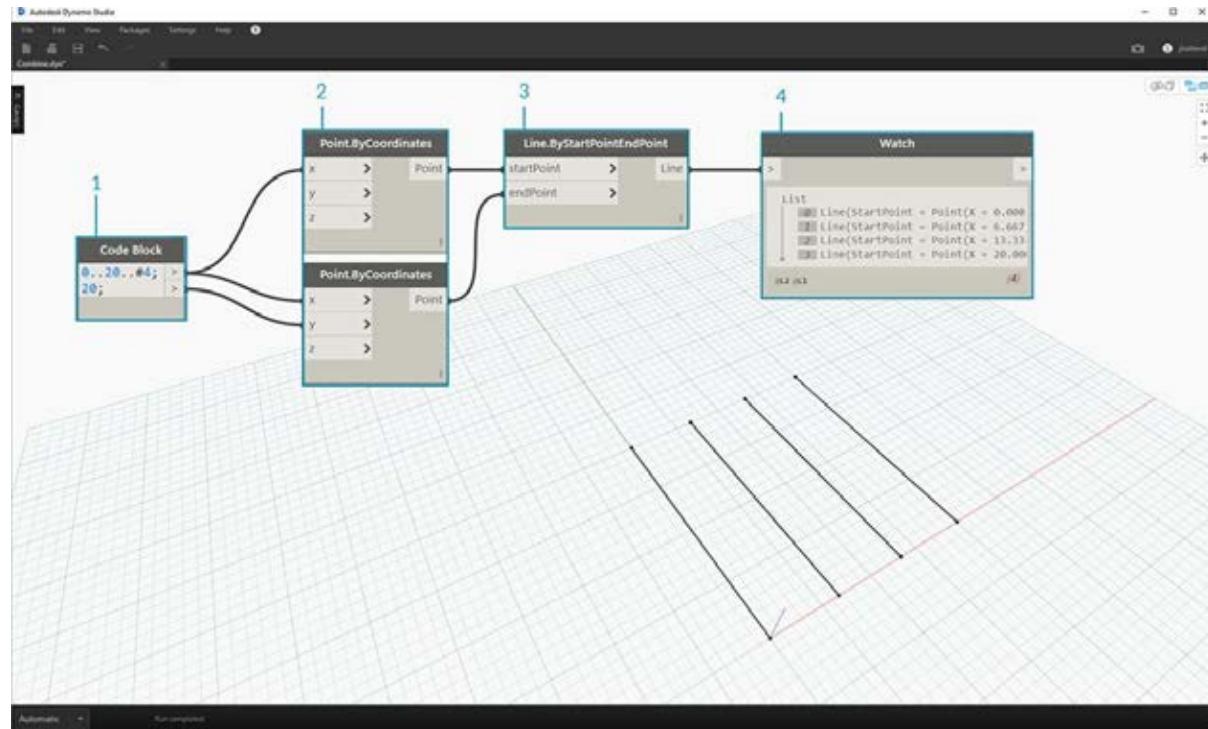
2. Beachten Sie, dass der *List.Count*-Block keine Eingabe hat. Er wird als Funktion eingesetzt, d. h., der *List.Count*-Block wird auf jede einzelne Liste eine Ebene unterhalb der Hauptliste in der Hierarchie angewendet. Die leere Eingabe von *List.Count* entspricht der Listeneingabe von *List.Map*.
3. Als Ergebnis von *List.Count* erhalten Sie jetzt eine Liste mit fünf Elementen, jeweils mit dem Wert 3. Dies entspricht der Länge jeder Unterliste.

### Übungslektion – List.Combine

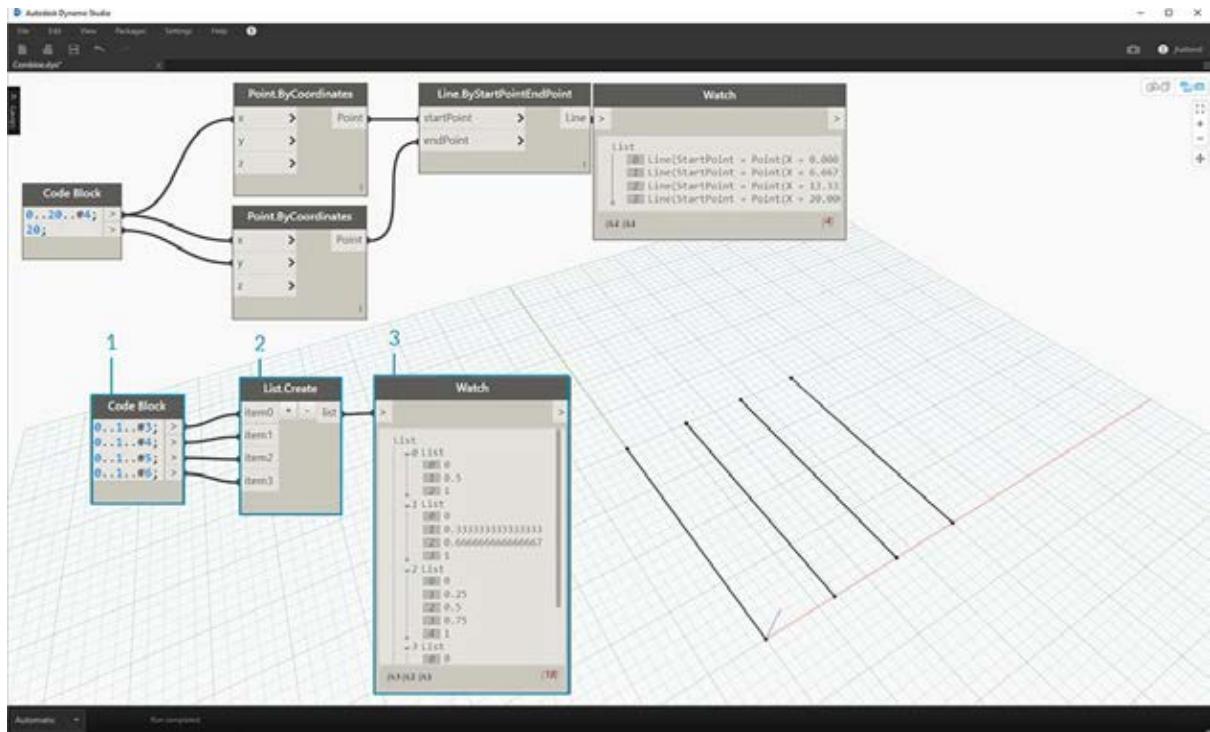
Anmerkung: Diese Übung wurde mit einer früheren Version von Dynamo erstellt. Die Funktionalität von *List.Combine* lässt sich großenteils durch die neu hinzugefügte Funktion *List@Level* ersetzen. Weitere Informationen finden Sie weiter unten unter [List@Level](#).

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Combine.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im [Appendix](#).

In dieser Übung wird eine ähnliche Logik verwendet wie für *List.Map*, wobei allerdings mehrere Elemente einbezogen werden. In diesem Fall soll jede der Kurven in einer Liste durch eine andere Anzahl von Punkten unterteilt werden.



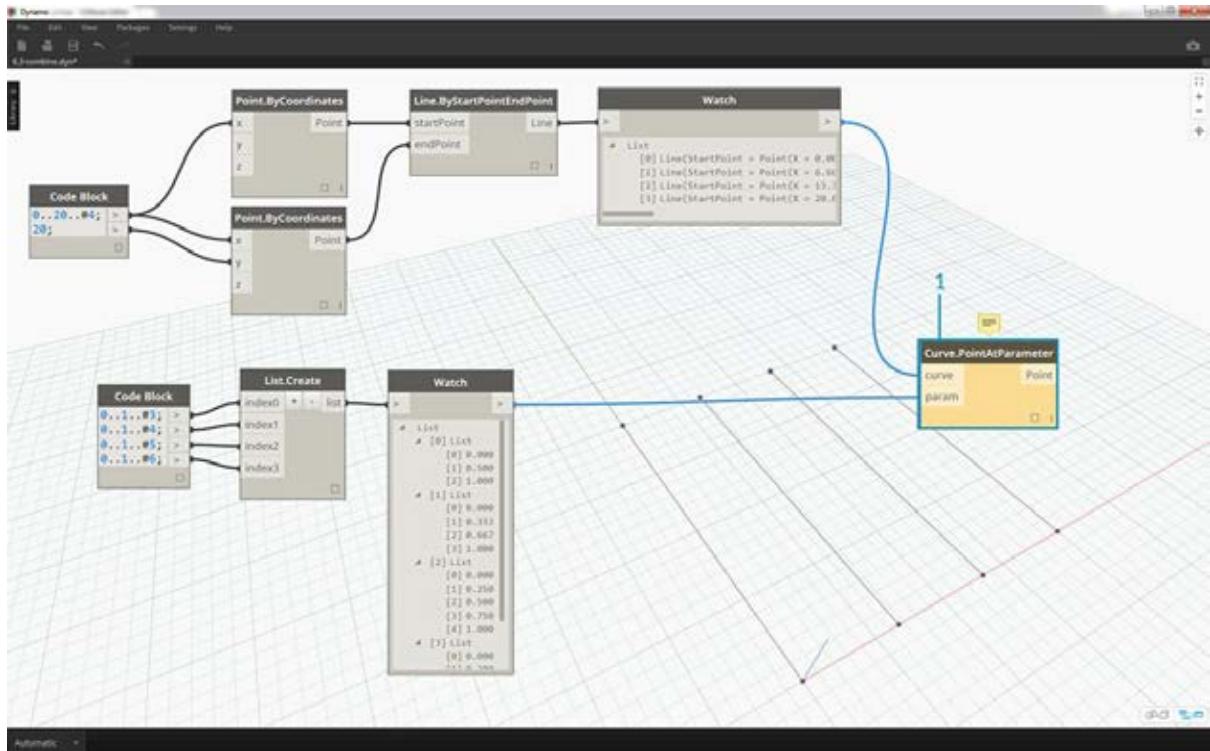
1. Definieren Sie im *Codeblock* einen Bereich mit der Syntax `0..20..#4;` und dem Wert `20`; in der Zeile darunter.
2. Verbinden Sie den *Codeblock* mit zwei *Point.ByCoordinates*-Blöcken.
3. Erstellen Sie eine *Line.ByStartPointEndPoint* aus den *Point.ByCoordinates*-Blöcken.
4. Der *Watch*-Block zeigt vier Linien.



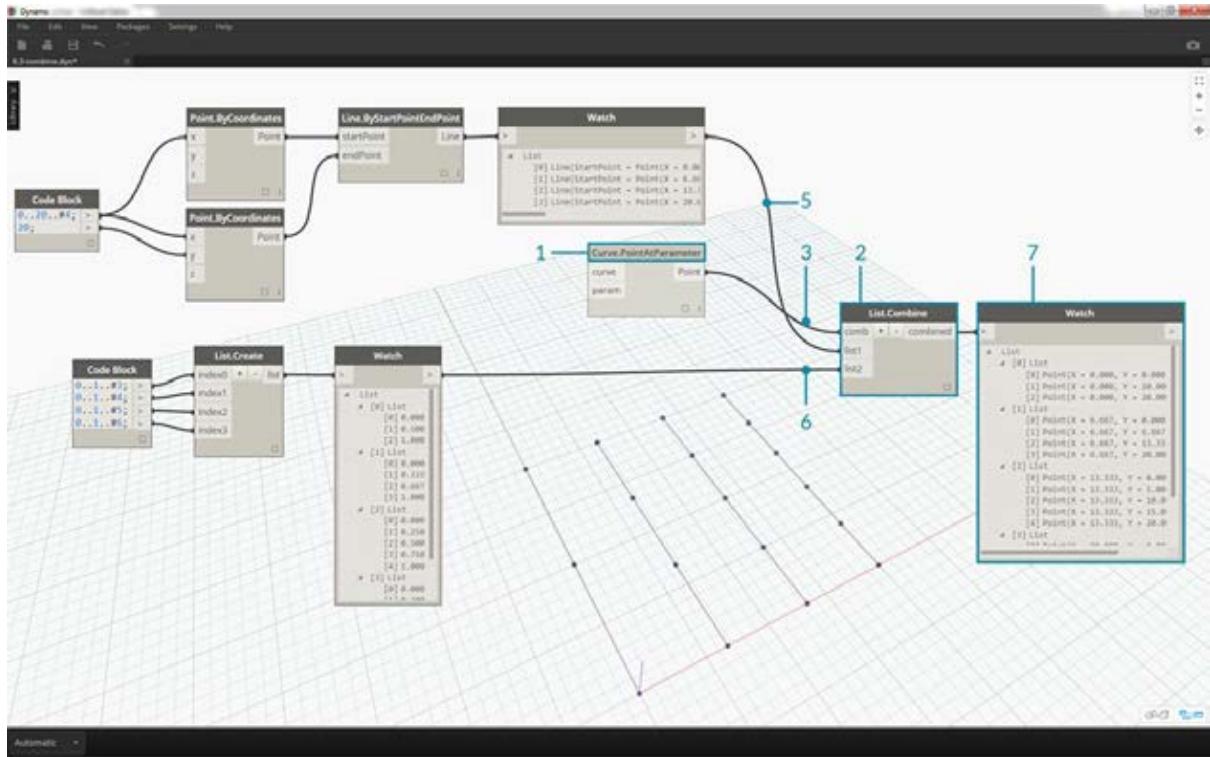
- Unter dem Diagramm für die Linierstellung sollen mithilfe eines *code block* vier separate Bereiche zum Unterteilen der einzelnen Linien erstellt werden. Verwenden Sie dazu die folgenden Codezeilen:

```
0..1..#3;
0..1..#4;
0..1..#5;
0..1..#6;
```

- Mithilfe eines *List.Create*-Blocks werden die vier Zeilen aus dem *Codeblock* in ein und derselben Liste zusammengefasst.
- Der *Watch*-Block zeigt eine Liste von Listen.



1. *Curve.PointAtParameter* kann nicht durch direkte Verbindung der Linien mit den *parameter*-Werten angewendet werden. Dies ist nur auf der nächsttieferen Ebene der Hierarchie möglich. Verwenden Sie hierfür *List.Combine*.



Mithilfe von *List.Combine* werden die einzelnen Linien unter Verwendung der angegebenen Bereiche unterteilt. Da dieser Vorgang etwas komplexer ist, wird er hier im Detail beschrieben.

1. Fügen Sie als Erstes einen *Curve.PointAtParameter*-Block im Ansichtsbereich hinzzu. Dies ist die „Funktion“ bzw. der „Kombinator“, der auf den *List.Combine*-Block angewendet wird. Dies wird etwas weiter unten genauer beschrieben.
2. Fügen Sie im Ansichtsbereich einen *List.Combine*-Block hinzzu. Durch Klicken auf "+" oder "-" können Sie Eingaben hinzufügen oder entfernen. Verwenden Sie in diesem Fall die vorgegebenen zwei Eingaben für den Block.
3. Verbinden Sie den *Curve.PointAtParameter*-Block mit der "comb"-Eingabe von *List.Combine*. Eine weitere wichtige Eingabe in einem der Blöcke: Achten Sie darauf, mit der rechten Maustaste auf die *param-Eingabe* von *Curve.PointAtParameter* zu klicken und *Vorgabewert verwenden* zu deaktivieren. Wenn ein Block als Funktion ausgeführt wird, müssen Vorgabewerte in Dynamo-Eingaben entfernt werden. In anderen Worten: Es wird grundsätzlich angenommen, dass Vorgabewerte mit zusätzlichen Blöcken verbunden sind. Aus diesem Grund müssen Sie in diesem Fall die Vorgabewerte entfernen.
4. Es sind zwei Eingaben vorhanden: die Linien und die Parameter zum Erstellen der Punkte. Wie und in welcher Reihenfolge werden diese mit den *List.Combine*-Eingaben verbunden?
5. Die Reihenfolge der leeren Eingaben in *Curve.PointAtParameter* (von oben nach unten) muss auch für die Eingaben im Kombinator eingehalten werden. Die Linien müssen daher mit *List1* von *List.Combine* verbunden werden.
6. Dementsprechend werden die Parameterwerte mit der *list2*-Eingabe von *List.Combine* verbunden.
7. Im *Watch*-Block und in der Dynamo-Vorschau sind vier Linien zu erkennen und jede dieser Linien ist entsprechend den Bereichen aus dem *Code Block* unterteilt.

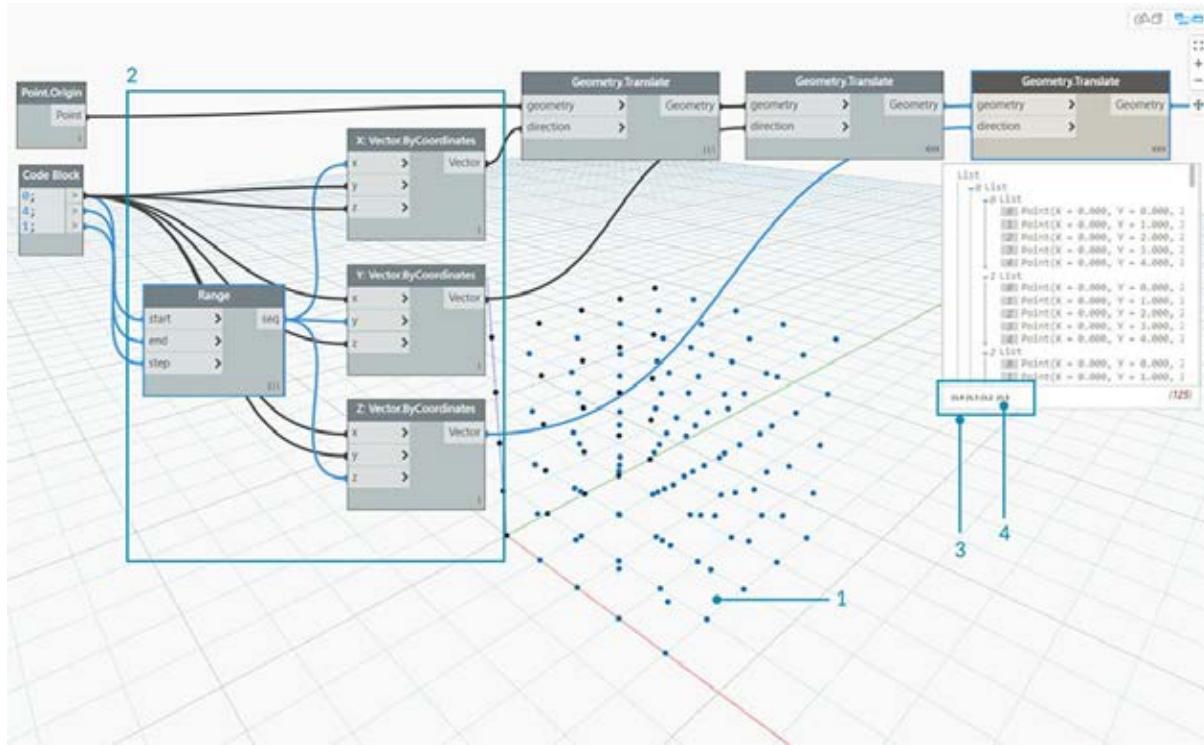
## List@Level

Die Funktion *List@Level* kann als Alternative zu *List.Map* eingesetzt werden und ermöglicht es, die Listenebene, mit der Sie arbeiten möchten, direkt am Eingabeanschluss auszuwählen. Diese Funktion kann auf jeden Eingabeanschluss eines Blocks angewendet werden und ermöglicht einen schnelleren und einfacheren Zugriff auf die Ebenen in Listen als andere Methoden. Geben Sie im Block einfach an, welche Ebene der Liste Sie als Eingabe verwenden möchten, und überlassen Sie die Ausführung dem Block.

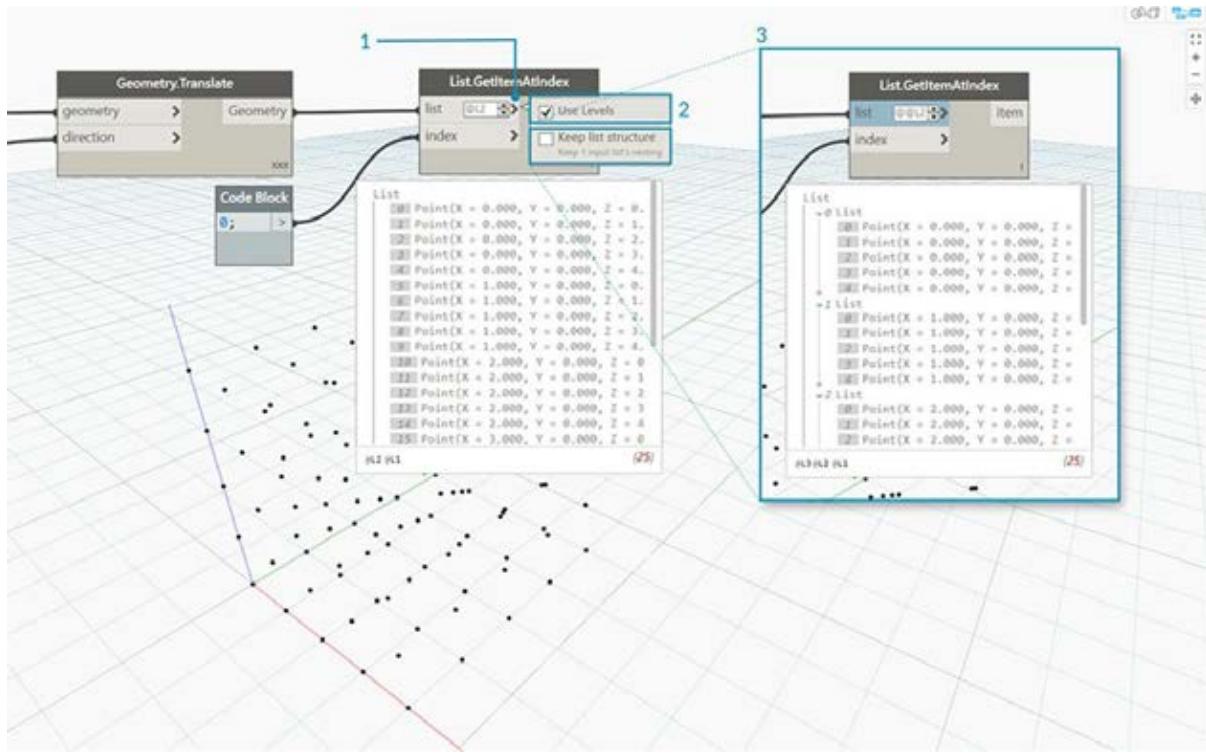
## Übungslektion – List@Level

In dieser Übung isolieren Sie mithilfe der Funktion List@Level eine bestimmte Datenebene.

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [List@Level](#). Eine vollständige Liste der Beispieldateien finden Sie im [Appendix](#).

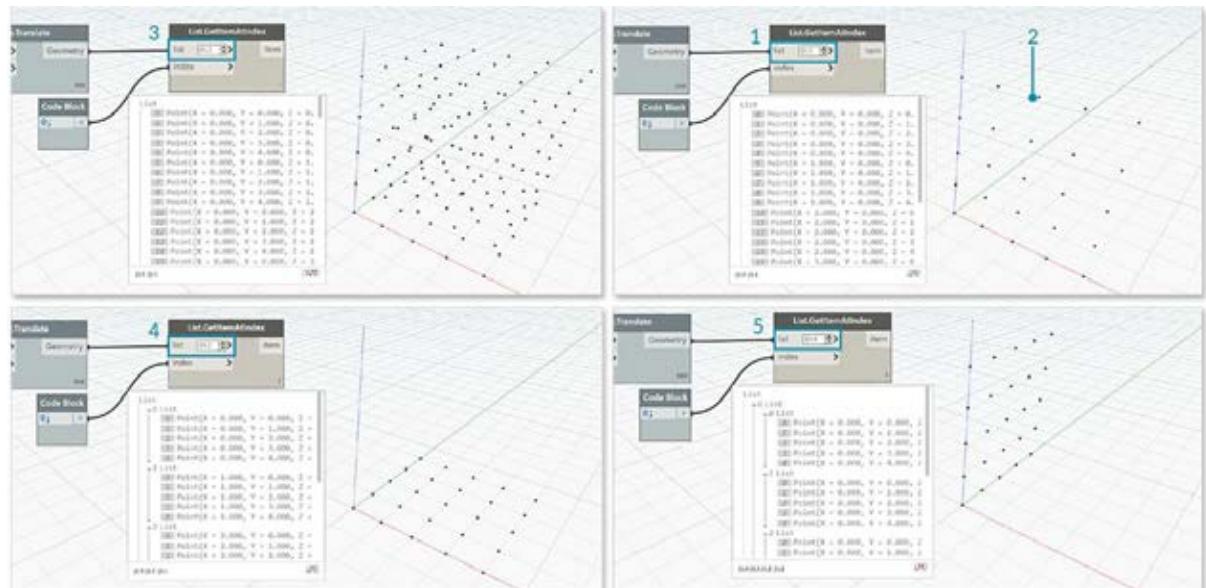


1. Sie beginnen mit einem einfachen 3D-Raster von Punkten.
2. Da das Raster mit Bereichen für X, Y und Z konstruiert wurde, ist bekannt, dass drei Datenstufen vorhanden ist, jeweils eine Liste für X, Y und Z.
3. Diese Stufen entsprechen verschiedenen **Ebenen**. Die Ebenen werden im unteren Bereich des Vorschaufensters angezeigt. Die Spalten für die Listenebenen entsprechen den oben angegebenen Listendaten und erleichtern es, die Ebene zu finden, auf der Sie arbeiten möchten.
4. Die Ebenen sind in umgekehrter Reihenfolge geordnet, d. h., die unterste Datenebene ist immer "L1". Dadurch wird sichergestellt, dass Ihre Diagramme wie geplant funktionieren, auch wenn vorangehende Schritte geändert werden.



- Um die Funktion List@Level zu verwenden, klicken Sie auf '>'. In diesem Menü werden zwei Kontrollkästchen angezeigt.
- Ebenen verwenden:** Aktiviert die Funktion List@Level. Nach dem Klicken auf diese Option können Sie durch die Eingabelistenebenen navigieren und die Ebene auswählen, die der Block verwenden soll. Über dieses Menü können Sie durch Navigieren nach oben oder unten rasch Optionen für verschiedene Ebenen testen.
- Listenstruktur beibehalten:** Ist dies aktiviert, haben Sie die Möglichkeit, die Ebenenstruktur der Eingabe beizubehalten. In manchen Fällen haben Sie eventuell Ihre Daten absichtlich in Unterlisten geordnet. Indem Sie diese Option aktivieren, können Sie Ihre Listenstruktur beibehalten, sodass keine Informationen verloren gehen.

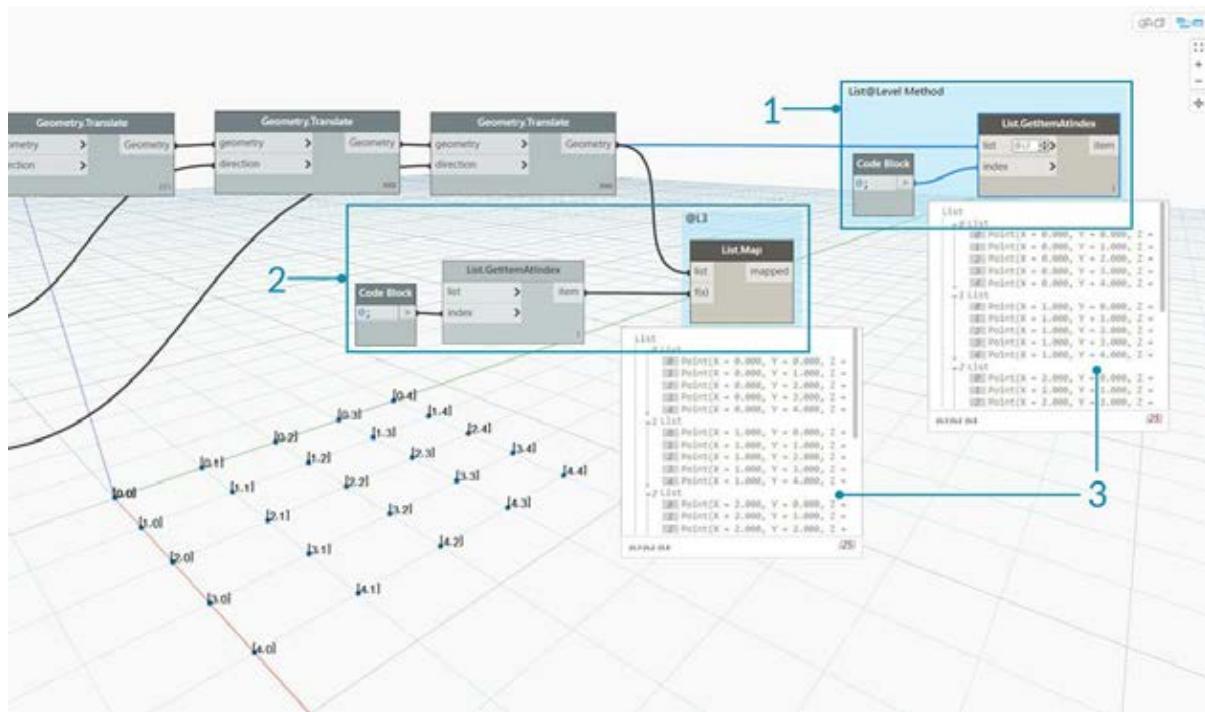
In diesem einfachen 3D-Raster können Sie auf die Listenstruktur zugreifen und sie visualisieren, indem Sie durch die Listenebenen blättern. Mit jeder Kombination aus Listenebene und Index erhalten Sie eine andere Untergruppe von Punkten aus der ursprünglichen 3D-Punktmenge.



- Mit der Angabe "@L2" in DesignScript wählen Sie ausschließlich die Liste auf Ebene 2 aus.
- Die Liste auf Ebene 2 mit dem Index 0 enthält nur die Gruppe von Punkten mit dem ersten Y-Wert, d. h. nur das XZ-Raster.
- Wenn Sie den Ebenenfilter in "L1" ändern, sind sämtliche Daten auf der ersten Listenebene sichtbar. Die Liste

- auf Ebene 1 mit dem Index 0 enthält alle 3D-Punkte in einer unstrukturierten Liste.
4. Mit "L3" werden nur die Punkte auf der dritten Listenebene angezeigt. Die Liste auf Ebene 3 mit dem Index 0 enthält nur die Gruppe von Punkten mit dem ersten Z-Wert, d. h. nur das XY-Raster.
  5. Mit "L4" werden nur die Punkte auf der dritten Listenebene angezeigt. Die Liste auf Ebene 4 mit dem Index 0 enthält nur die Gruppe von Punkten mit dem ersten X-Wert, d. h. nur das YZ-Raster.

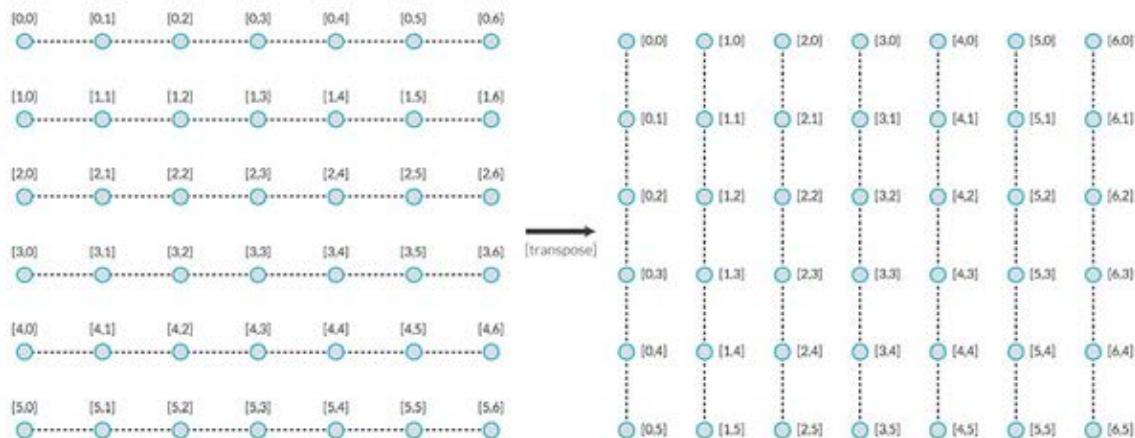
Obwohl dieses Beispiel auch mithilfe von List.Map erstellt werden kann, bietet List@Level eine wesentlich einfachere Interaktion und erleichtert dadurch den Zugriff auf die Daten des Blocks. Unten sehen Sie die Methoden List.Map und List@Level im Vergleich:



1. Beide Methoden ermöglichen den Zugriff auf dieselben Punkte, mit List@Level ist jedoch der mühelose Wechsel zwischen Datenebenen innerhalb desselben Blocks möglich.
2. Für den Zugriff auf ein Punktraster mithilfe von List.Map benötigen Sie außer dem eigentlichen List.Map-Block einen List.GetItemAtIndex-Block. Für jeden Schritt auf eine tiefere Listenebene benötigen Sie einen weiteren List.Map-Block. Dies könnte je nach Komplexität Ihrer Listen bedeuten, dass Sie Ihrem Diagramm eine größere Anzahl von List.Map-Blöcken hinzufügen müssen, um die richtige Informationsebene zu erreichen.
3. In diesem Beispiel gibt der List.GetItemAtIndex-Block zusammen mit dem List.Map-Block dieselbe Punktgruppe mit derselben Listenstruktur zurück wie List.GetItemAtIndex mit der Auswahl "L3".

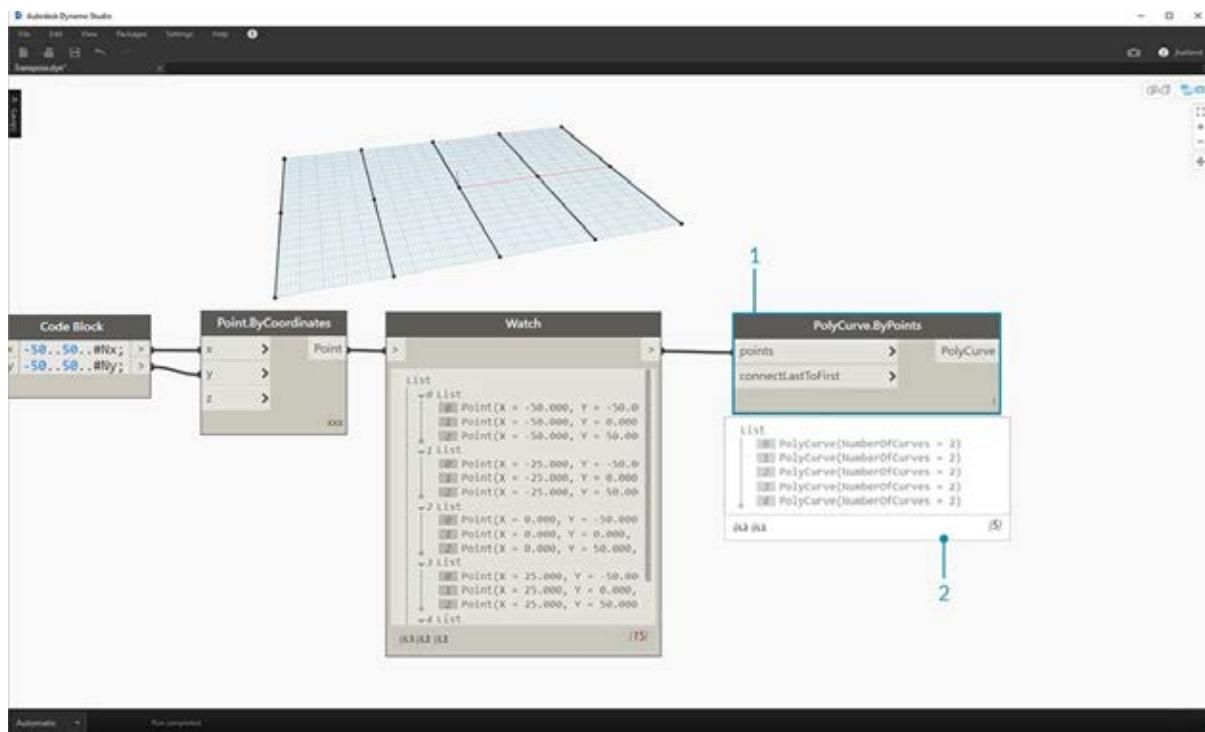
## Transpose

Transpose gehört zu den Grundfunktionen bei der Arbeit mit Listen von Listen. Mit Transpose werden genau wie in Tabellenkalkulationsprogrammen die Spalten und Zeilen einer Datenstruktur vertauscht. Dies wird in einer einfachen Matrix unten gezeigt. Im darauf folgenden Abschnitt wird beschrieben, wie Sie mithilfe von Transpose geometrische Beziehungen erstellen können.



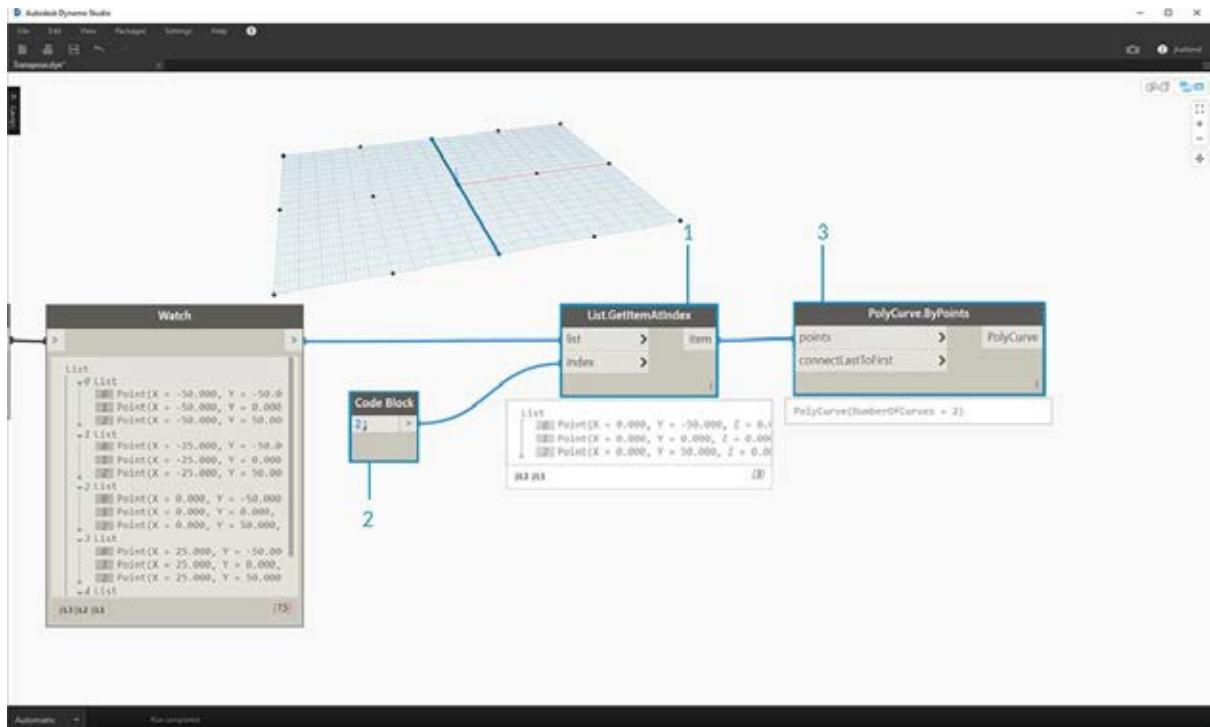
## Übungslektion – List.Transpose

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [Transpose.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

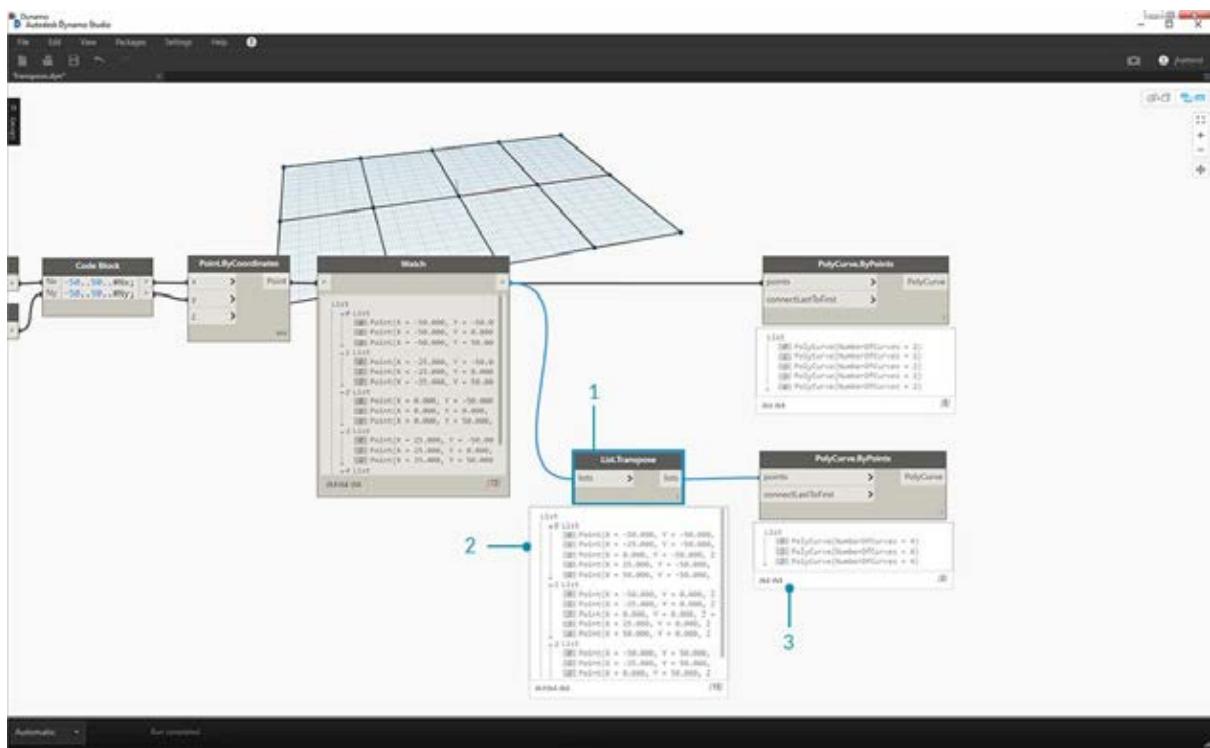


Löschen Sie die *List.Count*-Blöcke aus der vorhergegangenen Übungslektion, sodass Sie mit Geometrie arbeiten und die Struktur der Daten sehen können.

1. Verbinden Sie einen *PolyCurve.ByPoints*-Block mit der Ausgabe des Beobachtungsblocks für *Point.ByCoordinates*.
2. Die Ausgabe zeigt 5 Polykurven, die auch in der Dynamo-Vorschau angezeigt werden. Der Dynamo-Block sucht nach einer Liste von Punkten (bzw. in diesem Fall nach einer Liste von Listen von Punkten) und erstellt jeweils eine Polykurve. Dies bedeutet, dass jede Liste in eine Kurve in der Datenstruktur umgewandelt wurde.



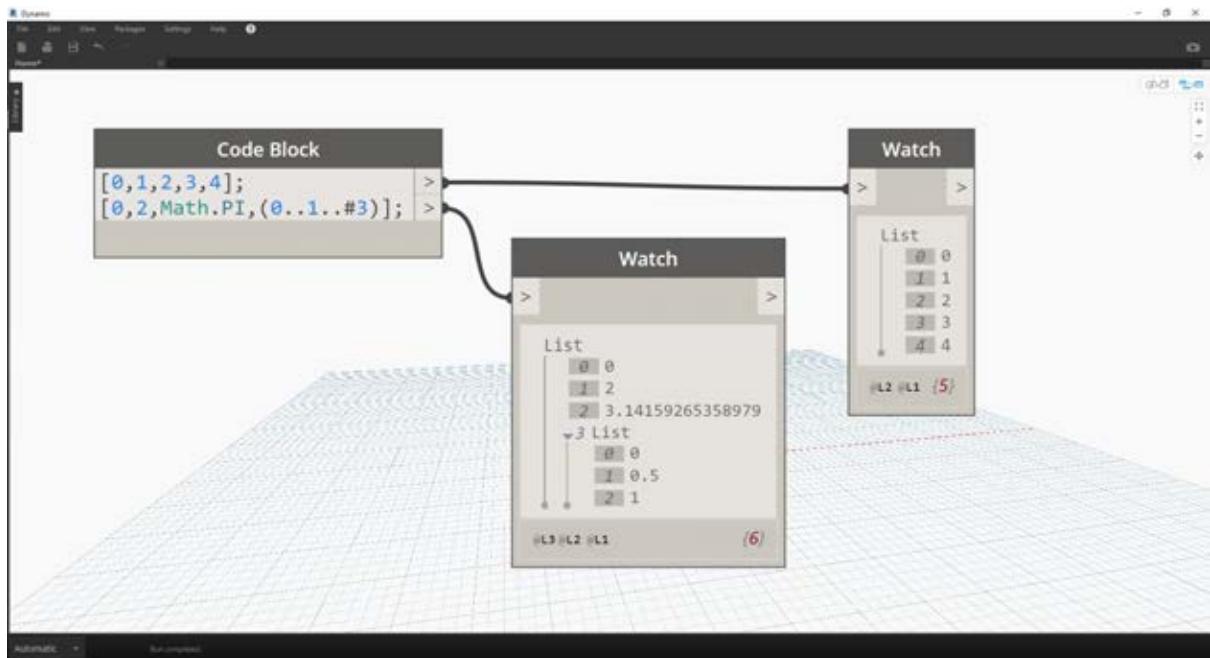
1. Um eine solche Kurve zu isolieren, verwenden Sie den *List.GetItemAtIndex*-Block.
2. Fragen Sie mithilfe des *Codeblock*-Werts 2 das dritte Element aus der Hauptliste ab.
3. *PolyCurve.ByPoints* gibt nur eine Kurve aus, da nur eine Liste mit dem Block verbunden ist.



1. Ein *List.Transpose*-Block vertauscht sämtliche Elemente mit allen Listen in einer Liste von Listen. Dies wirkt kompliziert, folgt aber derselben Logik wie die Funktion Transponieren in Microsoft Excel: Spalten und Zeilen in einer Datenstruktur werden vertauscht.
2. Im abstrakten Ergebnis ist zu sehen, dass *Transpose* die Listenstruktur aus 5 Listen mit je 3 Elementen in 3 Listen mit je 5 Elementen umgewandelt hat.
3. Das geometrische Ergebnis aus *PolyCurve.ByPoints* zeigt 3 Polykurven, die rechtwinklig zu den Originalkurven liegen.

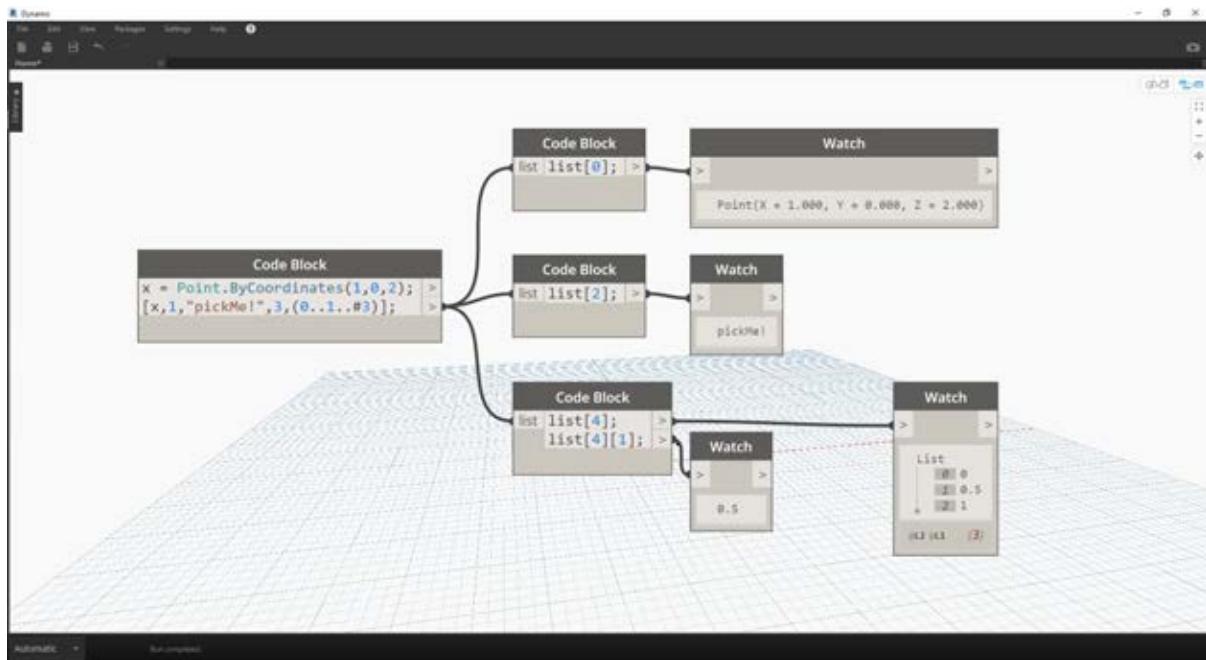
## **Erstellen mithilfe von Codeblöcken**

In der Codeblock-Notation wird "[]" zum Definieren einer Liste verwendet. Mit diesem Verfahren können Sie Listen wesentlich schneller und flexibler erstellen als mithilfe des List.Create-Blocks. Genauere Informationen zu Codeblöcken finden Sie in Kapitel 7. Die folgende Abbildung zeigt, wie Sie eine Liste mit mehreren Ausdrücken in einem Codeblock definieren können.



### Codeblock-Abfrage

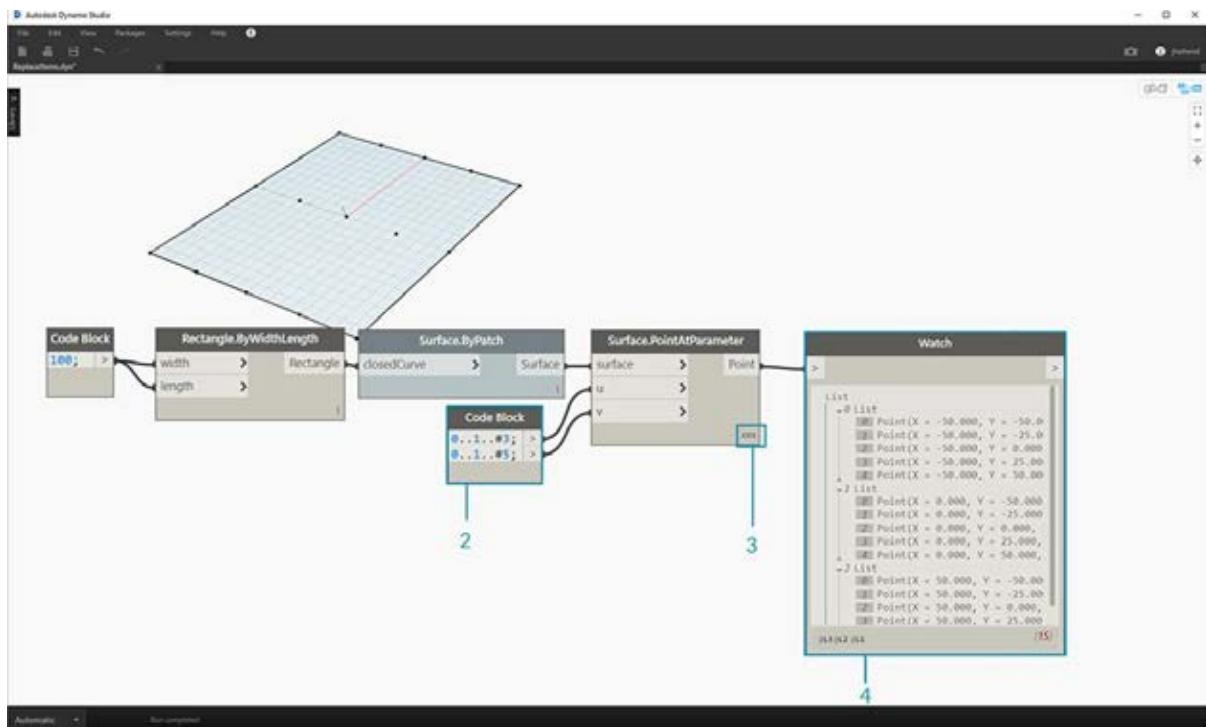
Die Codeblock-Notation verwendet eckige Klammern ("[]") als schnelle und einfache Möglichkeit zur Auswahl bestimmter Elemente, die aus einer komplexen Datenstruktur abgerufen werden sollen. Codeblöcke werden in Kapitel 7 detaillierter erläutert. Die folgende Abbildung zeigt, wie Sie mithilfe von Codeblöcken eine Liste mit mehreren Datentypen abfragen können.



## Übungslektion: Abfragen und Einfügen von Daten

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [ReplaceItems.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

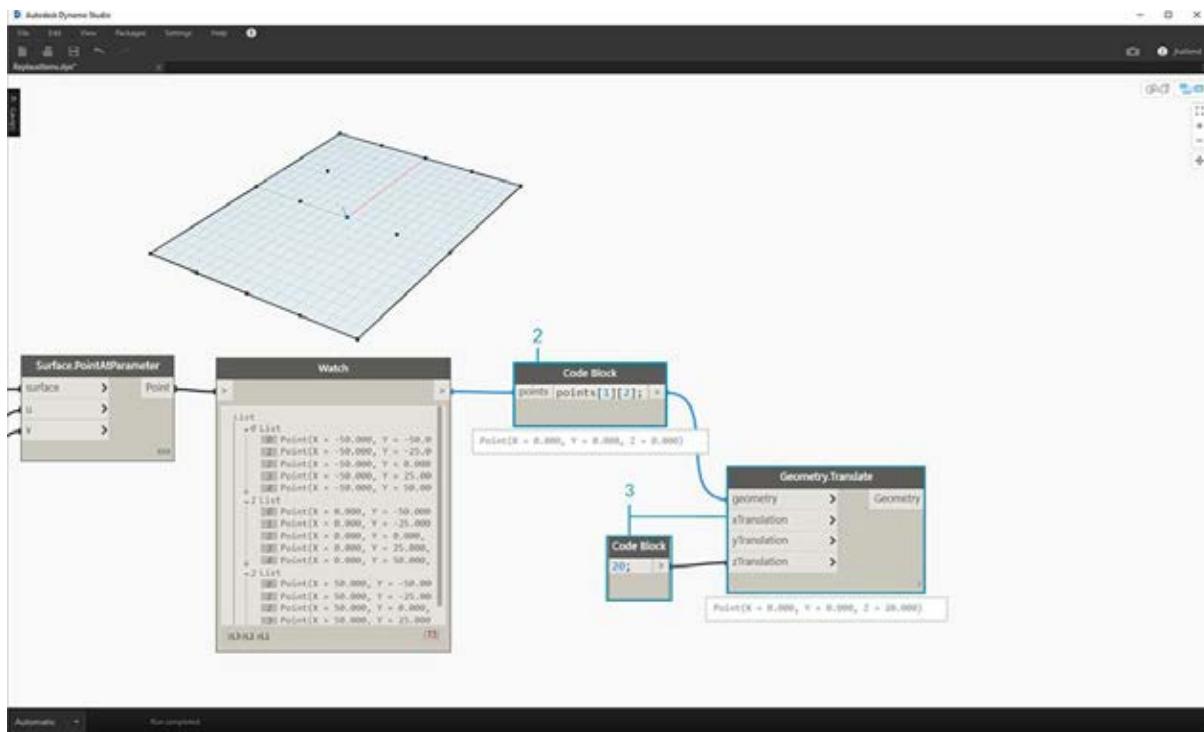
In dieser Übungslektion wird ein Teil der in der vorigen Lektion erstellten Logik zum Bearbeiten einer Oberfläche verwendet. Die Navigation in der Datenstruktur ist etwas komplexer, obwohl das angestrebte Ergebnis intuitiv wirkt. Die Oberfläche soll durch Verschieben eines Steuerpunkts strukturiert werden.



1. Beginnen Sie mit der oben gezeigten Folge von Blöcken. Dadurch wird eine einfache, das vorgegebene Dynamo-Raster umfassende Oberfläche erstellt.
2. Fügen Sie mithilfe eines *Codeblock* diese beiden Codezeilen ein und verbinden Sie sie mit den *u-* und *v-*Eingaben von *Surface.PointAtParameter*.

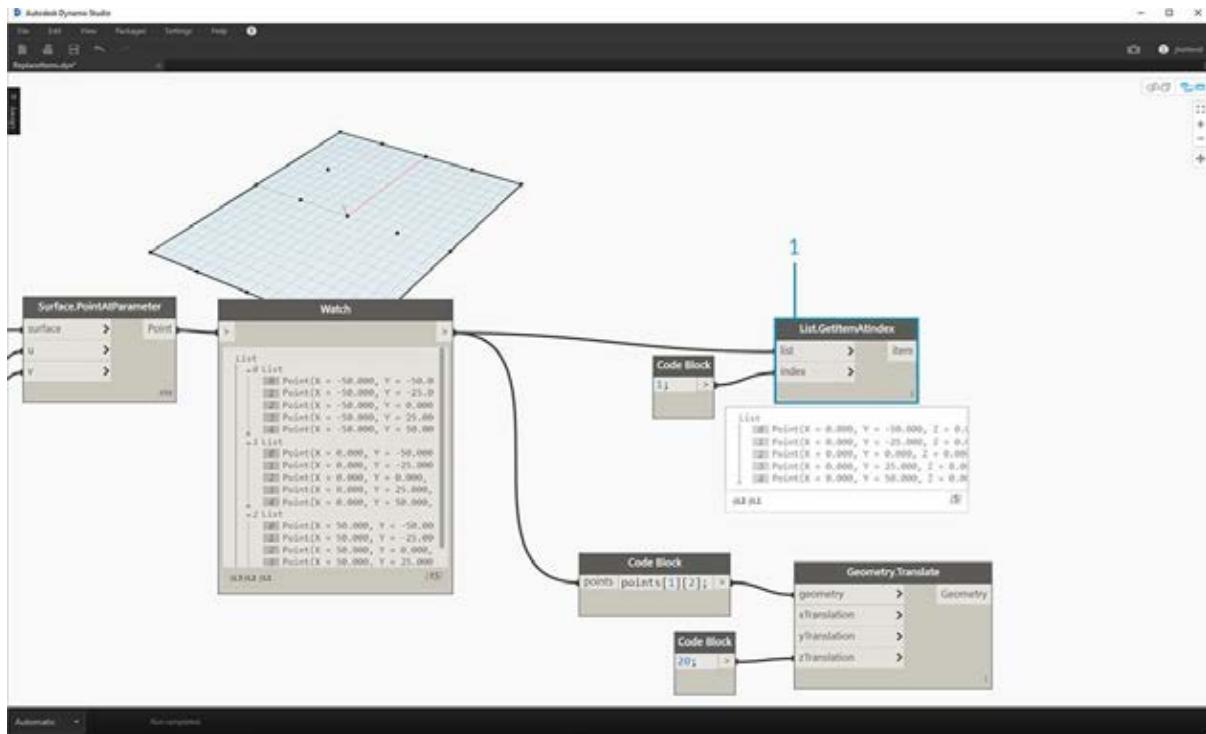
```
-50..50..#3;
-50..50..#5;
```

3. Achten Sie darauf, als Vergitterung von *Surface.PointAtParameter* die Option *Kreuzprodukt* zu wählen.
4. Im *Watch*-Block wird eine Liste aus drei Listen mit je fünf Elementen angezeigt.

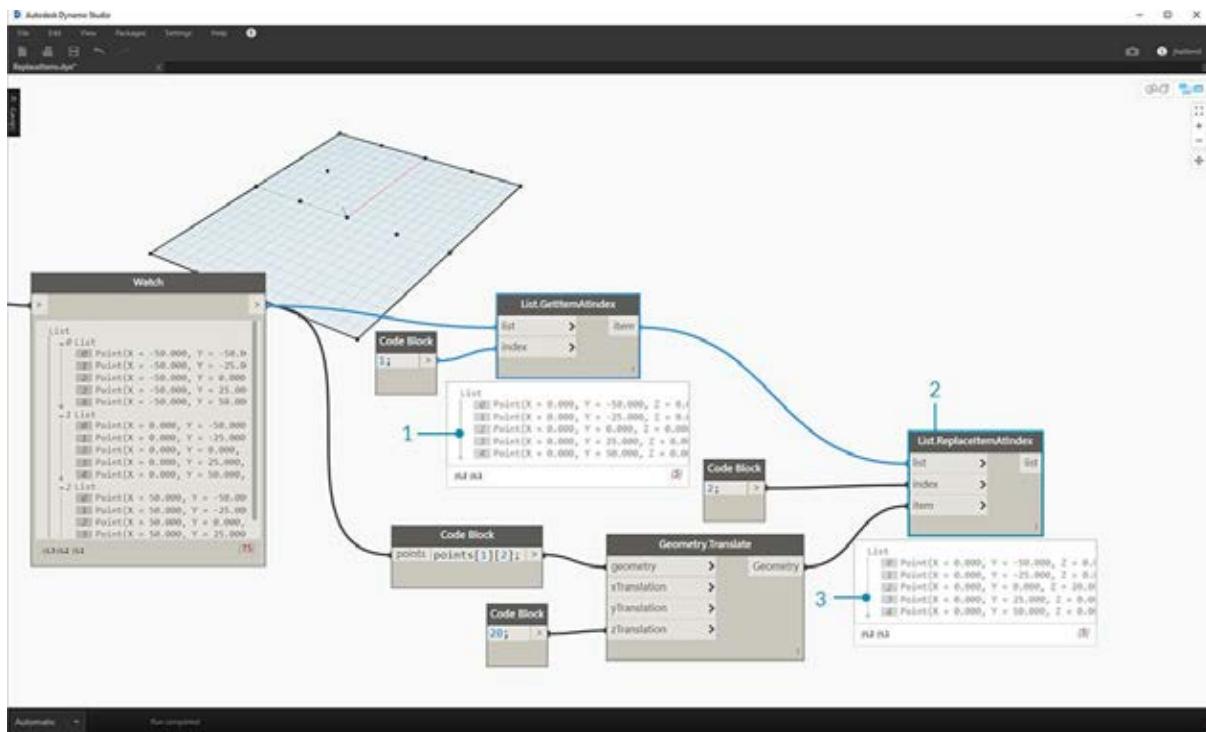


In diesem Schritt fragen Sie den Mittelpunkt des eben erstellten Rasters ab. Dazu wird die mittlere Zeile der mittleren Liste ausgewählt, wie es nahe liegt.

1. Sie können auch nacheinander auf die Elemente im *Watch*-Block klicken, um sicherzustellen, dass es sich um den richtigen Punkt handelt.
2. Schreiben Sie im *Codeblock* eine einfache Codezeile zum Abfragen einer Liste von Listen:  
`points[1][2];`
3. Verschieben Sie mithilfe von *Geometry.Translate* den ausgewählten Punkt in Z-Richtung um 20 Einheiten nach oben.

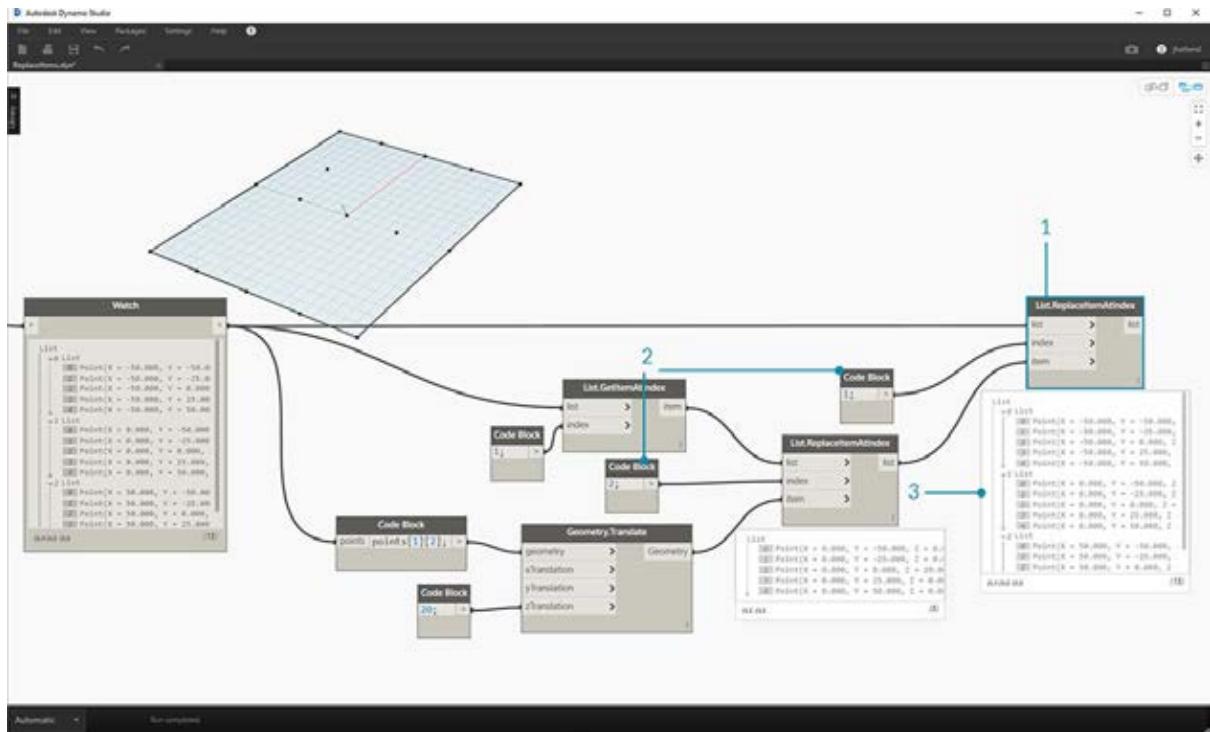


- Wählen Sie mithilfe eines *List.GetItemAtIndex*-Blocks auch die mittlere Reihe von Punkten aus. Anmerkung:  
Ähnlich wie im vorigen Schritt können Sie die Liste auch mithilfe eines *Codeblock* abfragen, wobei Sie die Zeile `points[1];` eingeben.



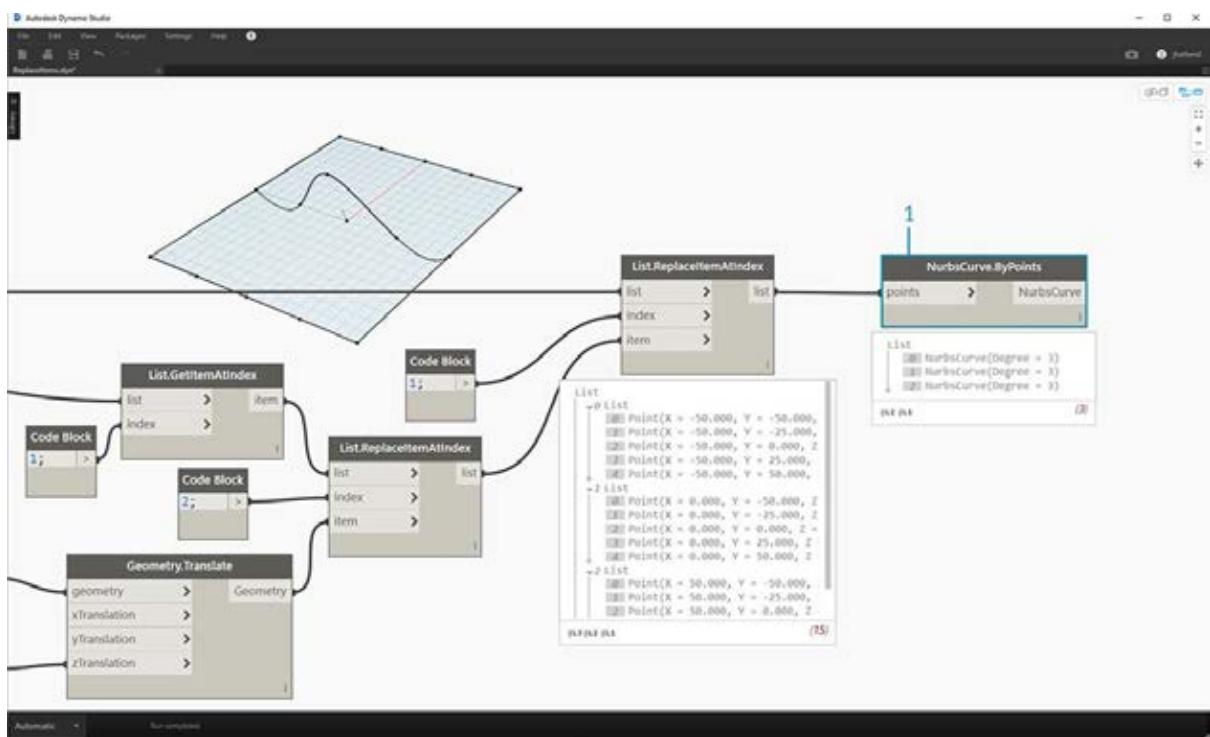
Damit haben Sie den Mittelpunkt abgefragt und ihn nach oben verschoben. Als Nächstes müssen Sie den verschobenen Punkt in die ursprüngliche Datenstruktur einfügen.

- Dazu ersetzen Sie zuerst das Listenelement, das Sie zuvor isoliert haben.
- Mit *List.ReplaceItemAtIndex* ersetzen Sie das mittlere Element mit dem Index "2" durch das Ersatzelement, das mit dem verschobenen Punkt (*Geometry.Translate*) verbunden ist.
- Die Ausgabe zeigt, dass der verschobene Punkt in das mittlere Element der Liste eingegeben wurde.



Jetzt müssen Sie die geänderte Liste in die ursprüngliche Datenstruktur, d. h. die Liste von Listen, einfügen.

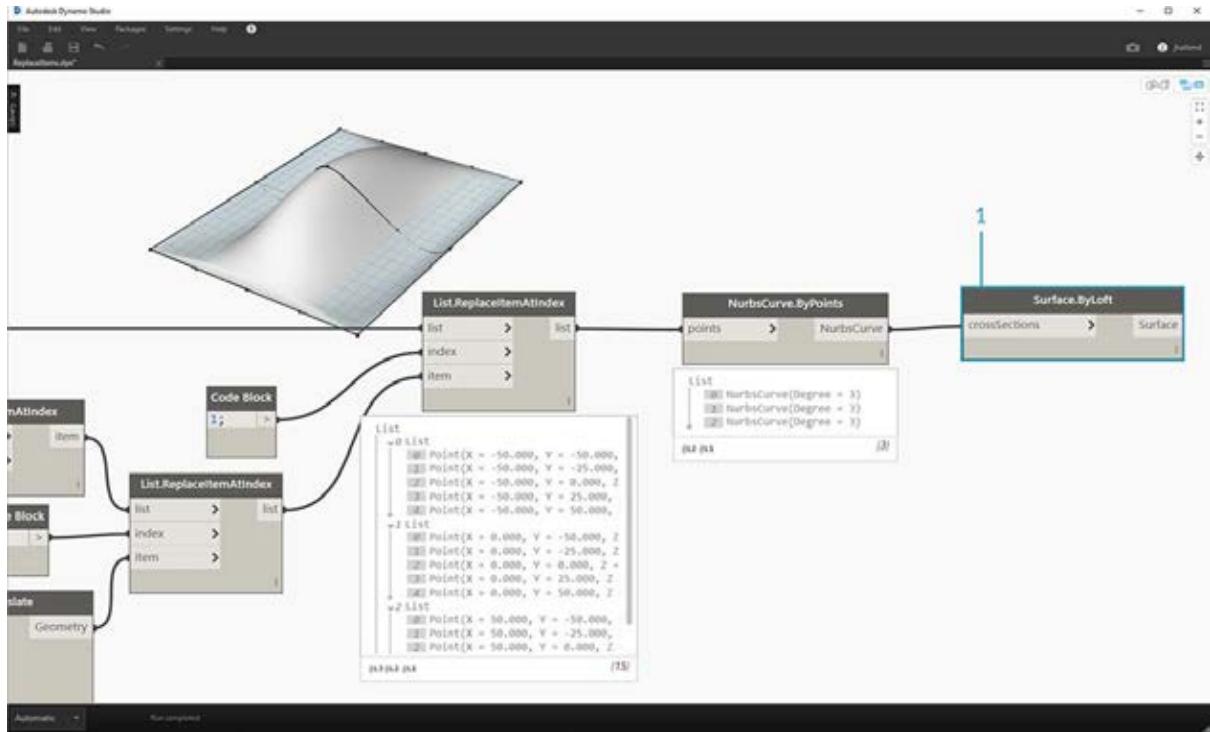
1. Ersetzen Sie nach demselben Prinzip die mittlere Liste mithilfe von `List.ReplaceItemAtIndex` durch die geänderte Liste.
2. Beachten Sie, dass in den `Codeblocks`, die die Indizes für diese beiden Blöcke definieren, die Werte 1 und 2 angegeben wurde, was der ursprünglichen Abfrage aus dem `Codeblock` (`points[1][2]`) entspricht.
3. Wenn Sie die Liste mit `index 1` auswählen, wird die Datenstruktur in der Dynamo-Vorschau hervorgehoben. Damit haben Sie den verschobenen Punkt in die ursprüngliche Datenstruktur aufgenommen.



Es gibt viele Möglichkeiten zum Erstellen einer Oberfläche aus dieser Punktgruppe. In diesem Fall erstellen Sie die Oberfläche durch Erheben und Verbinden von Kurven.

1. Erstellen Sie einen `NurbsCurve.ByPoints`-Block und verbinden Sie ihn mit der neuen Datenstruktur, um drei

Nurbs-Kurven zu erstellen.



1. Verbinden Sie einen **Surface.ByLoft**-Block mit der Ausgabe von **NurbsCurve.ByPoints**. Damit haben Sie eine geänderte Oberfläche erstellt. Sie können den ursprünglichen Z-Wert der Geometrie ändern. Beobachten Sie, wie die Geometrie durch diese Verschiebung aktualisiert wird.

# **n-dimensionale Listen**

## **n-dimensionale Listen**

Sie können der Hierarchie weitere untergeordnete Ebenen hinzufügen. Datenstrukturen können Listen von Listen mit weit mehr als zwei Dimensionen umfassen. Da Listen in Dynamo als Objekte behandelt werden, können Sie Daten mit so vielen Dimensionen erstellen, wie es möglich ist.

Als Metapher für diese Funktionen eignen sich ineinander verschachtelte russische Matrjoschka-Puppen. Jede Liste kann ein Container betrachtet werden, der mehrere Elemente enthält. Jede Liste verfügt über eigene Eigenschaften und wird als eigenständiges Objekt angesehen.



Die verschachtelten Matjoschka-Puppen (Foto: [Zeta](#)) sind eine Metapher für n-dimensionale Listen. Jede Ebene steht für eine Liste und jede Liste enthält Elemente. In Dynamo kann jeder Container seinerseits mehrere Container enthalten (die den Elementen der jeweiligen Liste entsprechen).

Die visuelle Veranschaulichung n-dimensionaler Listen ist schwierig. In diesem Kapitel finden Sie jedoch eine Reihe von Übungslektionen für die Arbeit mit Listen, die mehr als zwei Dimensionen enthalten.

## Mapping und Kombinationen

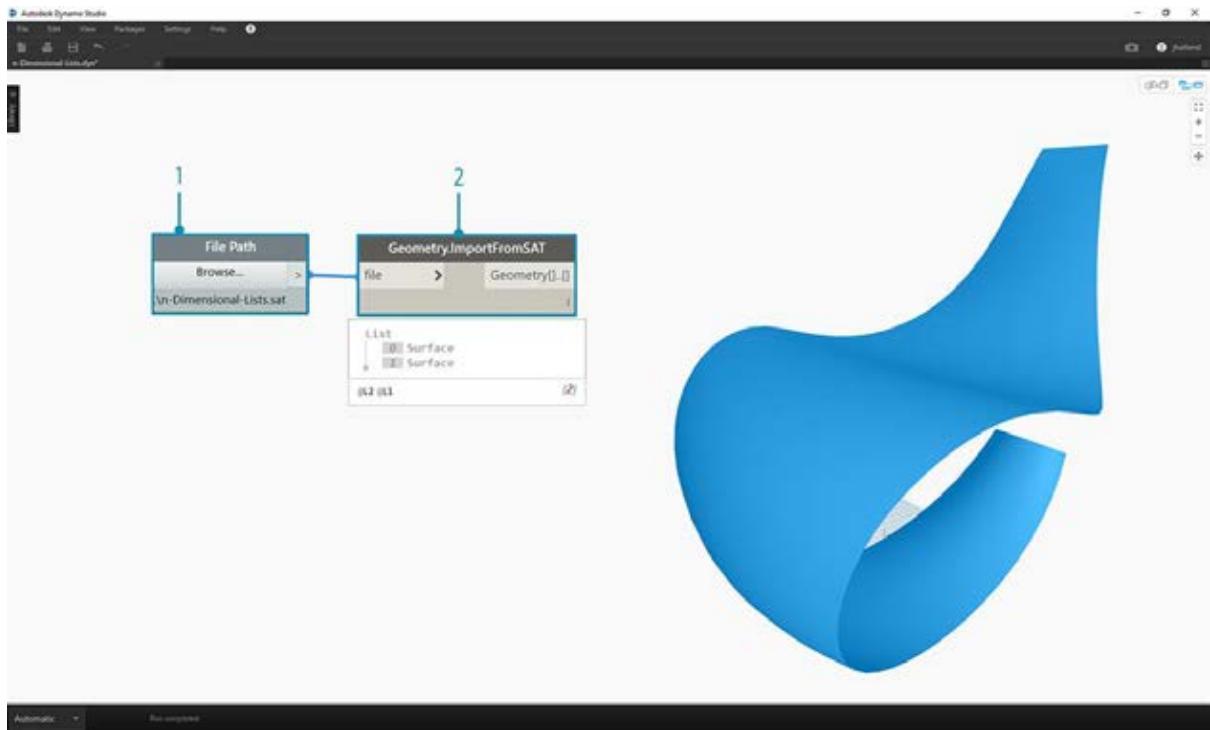
Mapping ist der wohl komplexeste Aspekt der Datenverwaltung in Dynamo. Bei der Arbeit mit komplexen Listenhierarchien kommt ihm besondere Bedeutung zu. In den folgenden Übungslektionen wird gezeigt, wann Mapping und Kombinationen für mehrdimensionale Daten eingesetzt werden sollten.

Eine vorbereitende Einführung zu List.Map und List.Combine finden Sie im vorigen Abschnitt. In der letzten Übungslektion werden diese Blöcke für eine komplexe Datenstruktur angewendet.

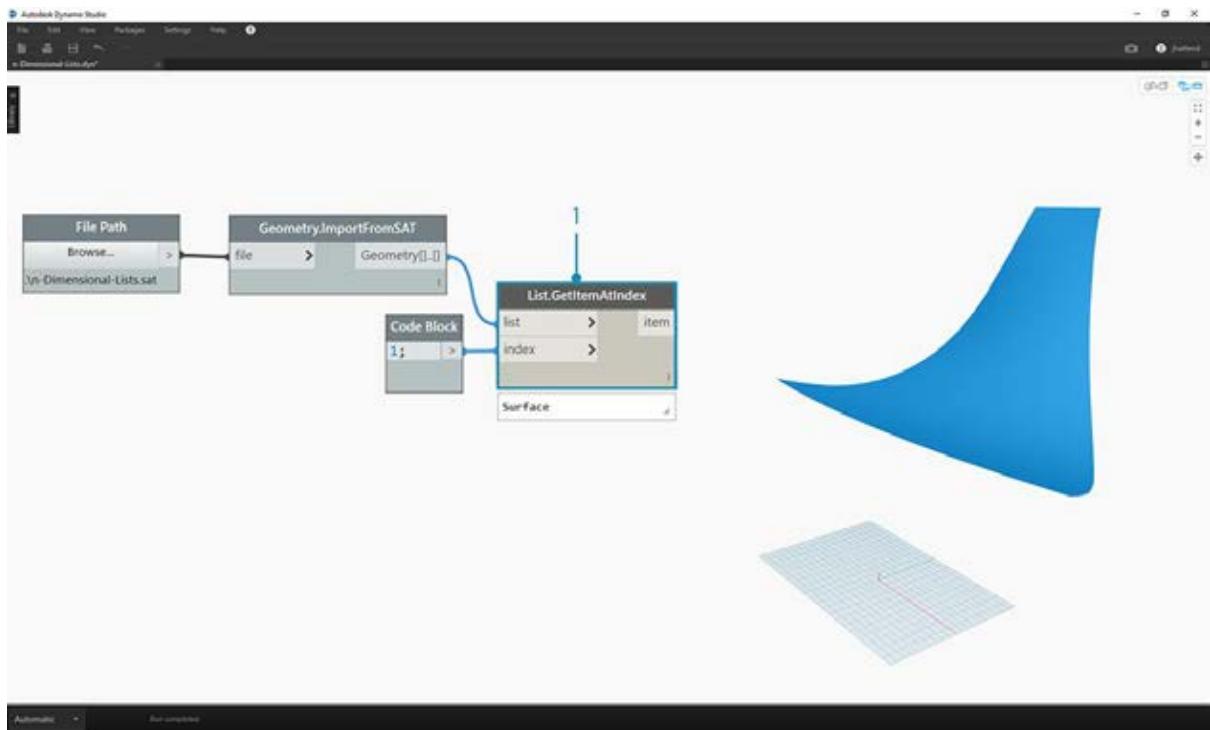
### Übungslektion: 2D-Listen – Grundlagen

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. 1.[n-Dimensional-Lists.dyn](#) 2.[n-Dimensional-Lists.sat](#)

Dies ist die erste von drei Übungslektionen zur Strukturierung importierter Geometrie. In den Teilen dieser Übungsreihe werden nach und nach komplexere Datenstrukturen verwendet.

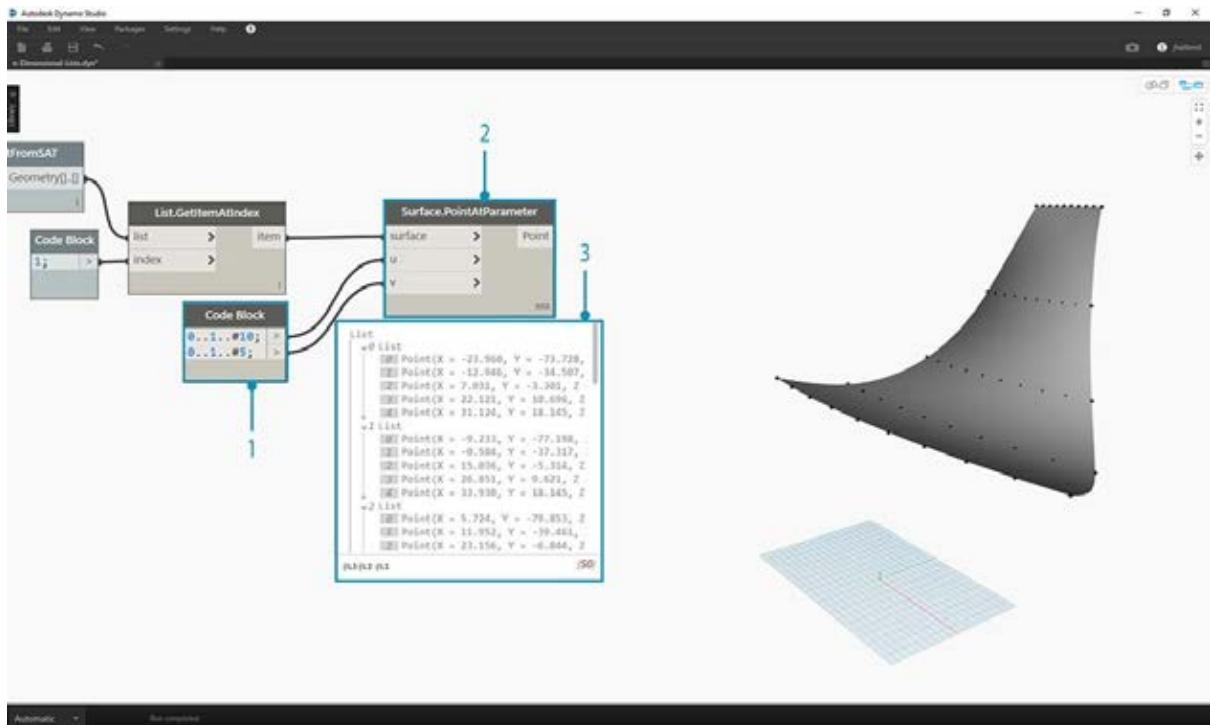


1. Sie beginnen mit der SAT-Datei aus dem Ordner mit den Übungsdateien. Diese Datei können Sie mithilfe des *File Path*-Blocks abrufen.
2. Mit *Geometry.ImportFromSAT* wird die Geometrie in Form zweier Oberflächen in die Dynamo-Vorschau importiert.



In dieser Übung wird der Einfachheit halber nur eine der Oberflächen verwendet.

1. Wählen Sie den Index 1, um die obere Oberfläche auszuwählen. Verwenden Sie dazu den *List.GetItemAtIndex*-Block.

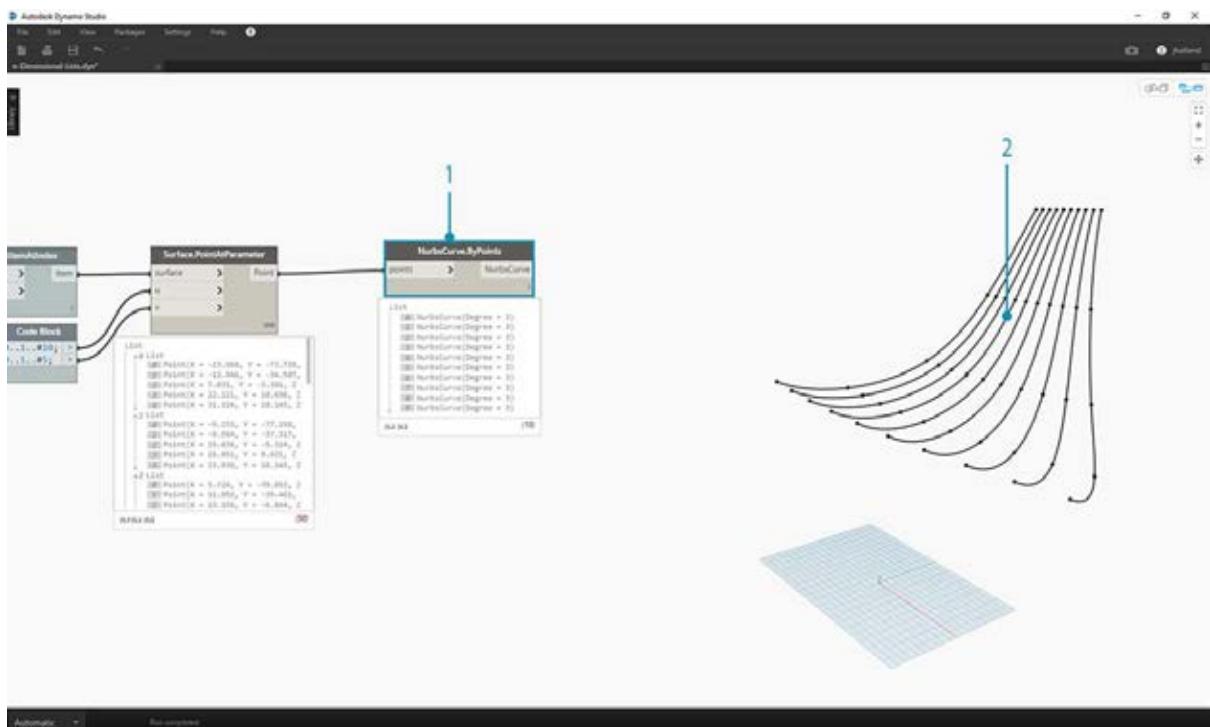


Im nächsten Schritt unterteilen Sie die Oberfläche mithilfe eines Rasters aus Punkten.

- Fügen Sie in einem *Code Block* die beiden folgenden Codezeilen ein:

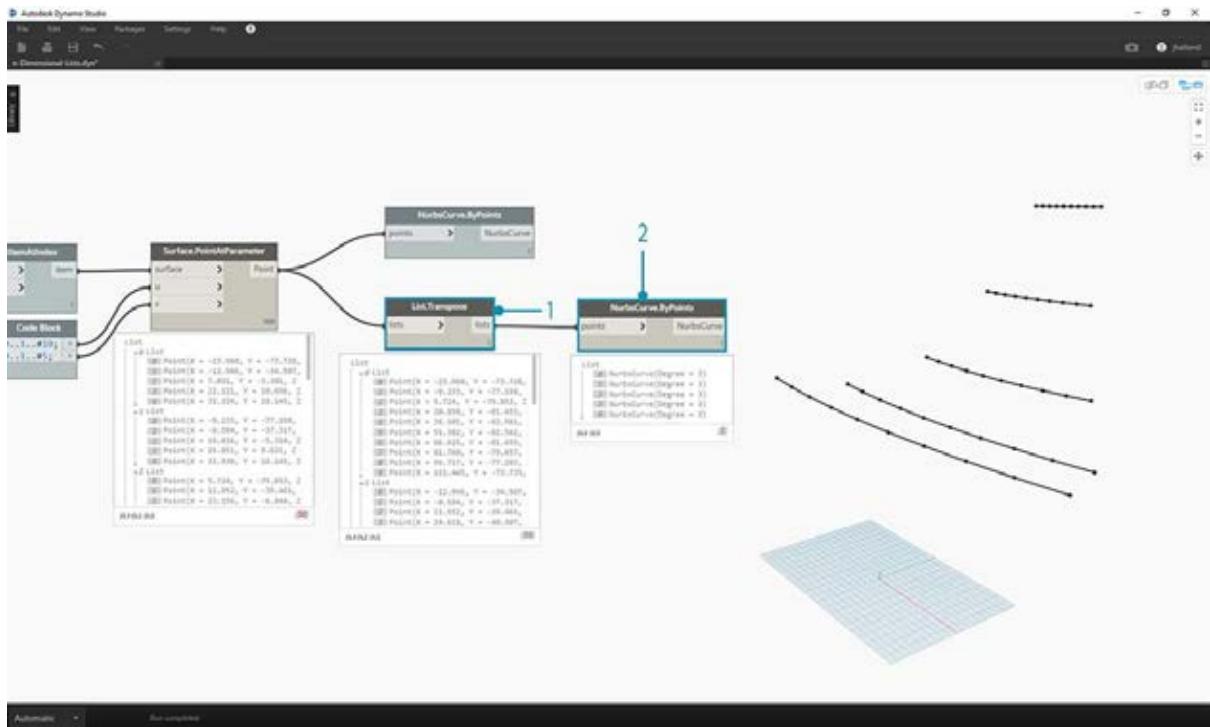
```
0..1..#10;
0..1..#5;
```

- Verbinden Sie in *Surface.PointAtParameter* die beiden Codeblock-Werte mit *u* und *v*. Ändern Sie die *Vergitterung* dieses Knotens in "Kreuzprodukt".
- In der Ausgabe und in der Dynamo-Vorschau wird die Datenstruktur angezeigt.



- Um den Aufbau der Datenstruktur zu verdeutlichen, verbinden Sie einen *NurbsCurve.ByPoints*-Block mit der Ausgabe von *Surface.PointAtParameter*.

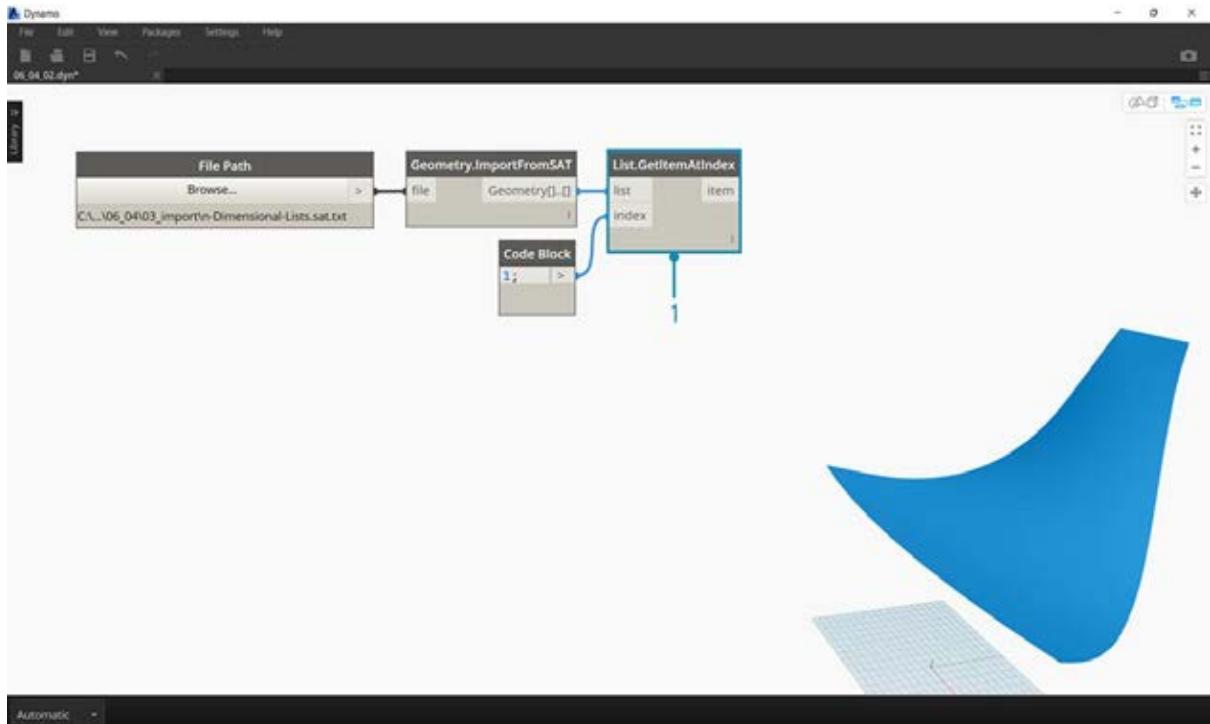
2. Sie erhalten zehn Kurven, die vertikal entlang der Oberfläche verlaufen.



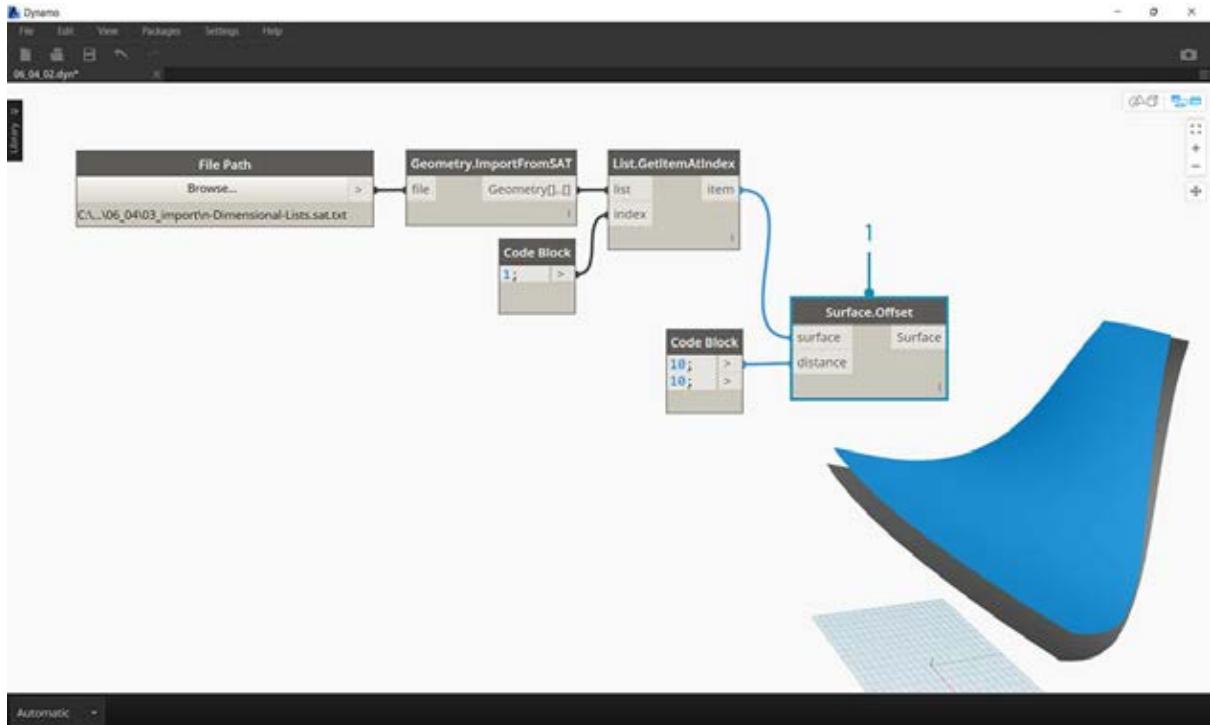
1. Mit einer einfachen *List.Transpose*-Operation vertauschen Sie die Spalten und Zeilen einer Liste von Listen.
2. Indem Sie die Ausgabe des *List.Transpose*-Blocks mit einem *NurbsCurve.ByPoints*-Blocks verbinden, erhalten Sie fünf Kurven, die horizontal auf der Oberfläche verlaufen.

### Übungslektion: 2D-Listen – Weiterführend

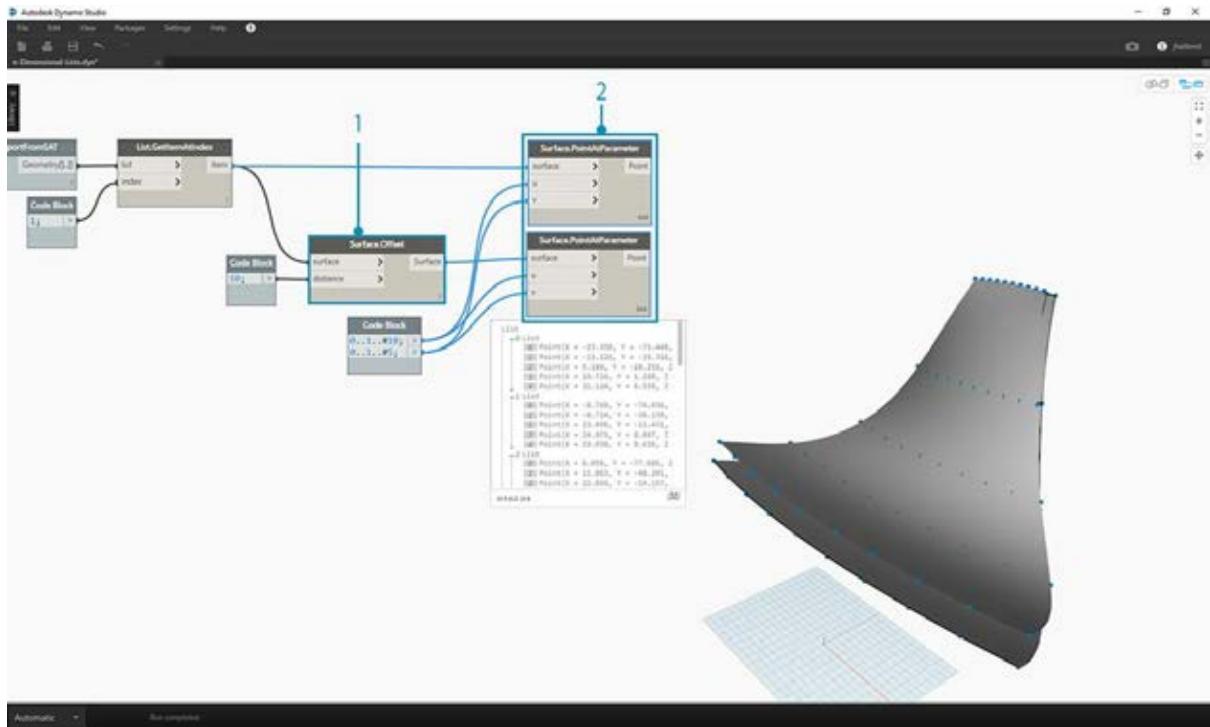
In diesem Schritt erhöhen Sie die Komplexität. Angenommen, an den Kurven aus der vorigen Übungslektion soll eine Operation durchgeführt werden. Vielleicht sollen die Kurven auf eine andere Oberfläche bezogen und eine Erhebung zwischen ihnen erstellt werden. Die hierfür erforderliche Datenstruktur erfordert mehr Aufwand, wobei jedoch dieselbe Logik zugrunde liegt.



- Beginnen Sie mit demselben Schritt wie in der vorherigen Übung, indem Sie die obere der beiden Oberflächen der importierten Geometrie mithilfe des *List.GetItemAtIndex*-Blocks isolieren.



- Versetzen Sie die Oberfläche mithilfe von *Surface.Offset* um den Wert 10.

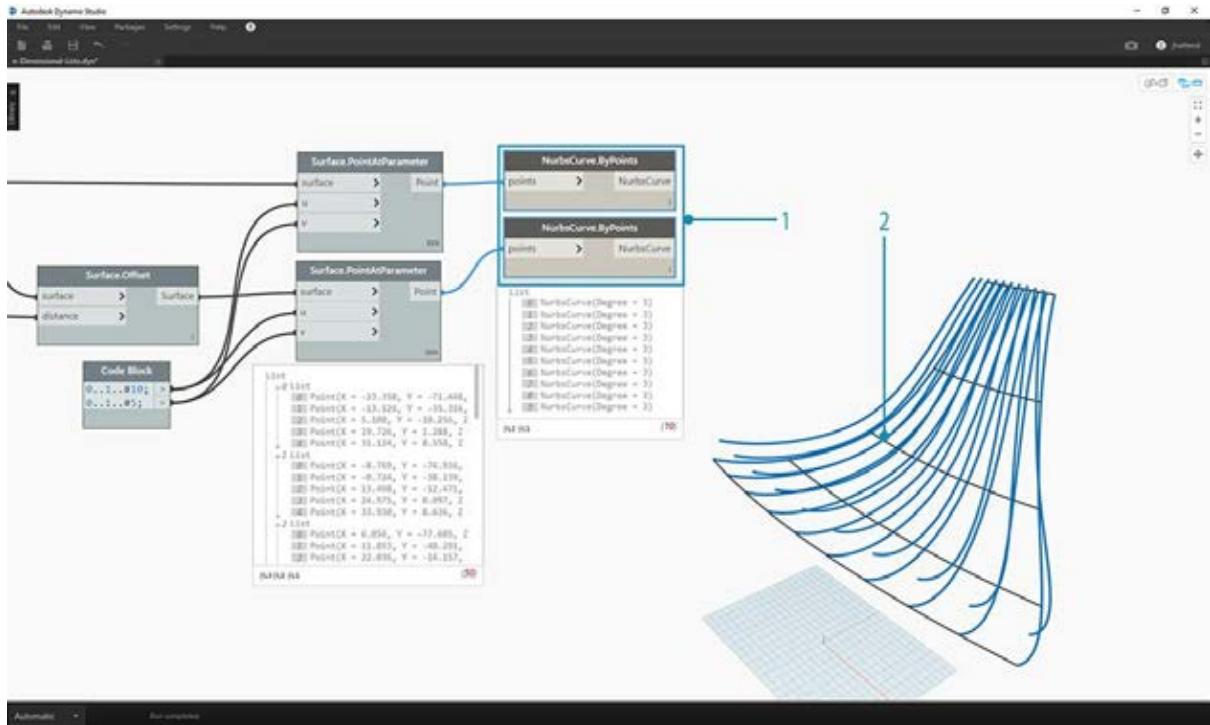


- Definieren Sie auf dieselbe Weise wie in der vorigen Übung einen *Code Block* mit den folgenden beiden Codezeilen:

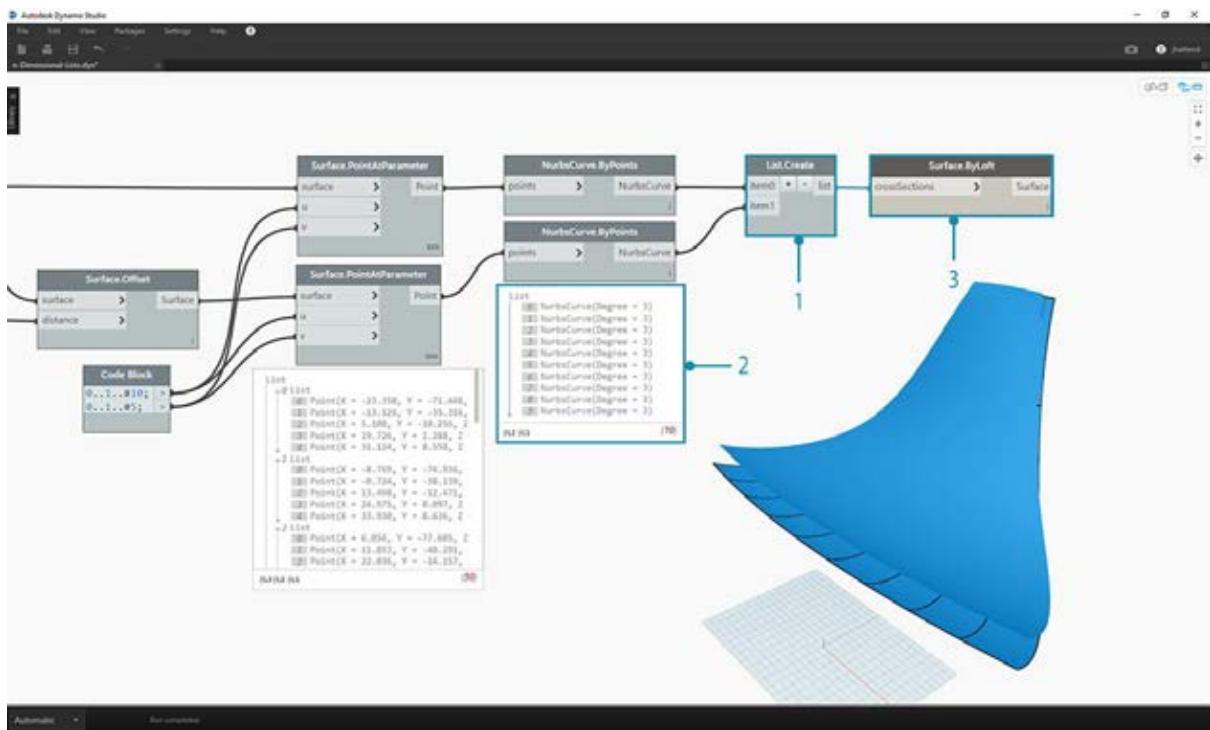
```
0..1..#10;  
0..1..#5;
```

- Verbinden Sie diese Ausgaben mit zwei *Surface.PointAtParameter*-Blöcken, jeweils mit der *Vergitterung* "Kreuzprodukt". Verbinden Sie einen dieser Blöcke mit der ursprünglichen und den anderen mit der versetzten

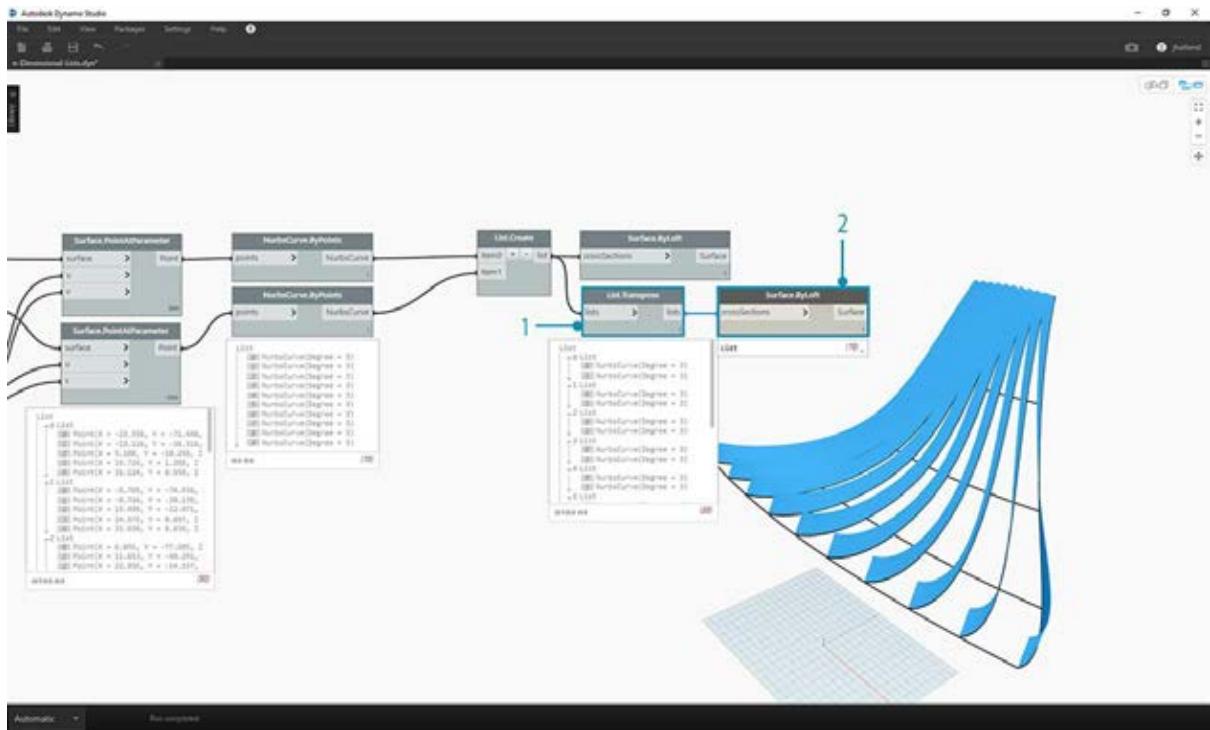
## Oberfläche.



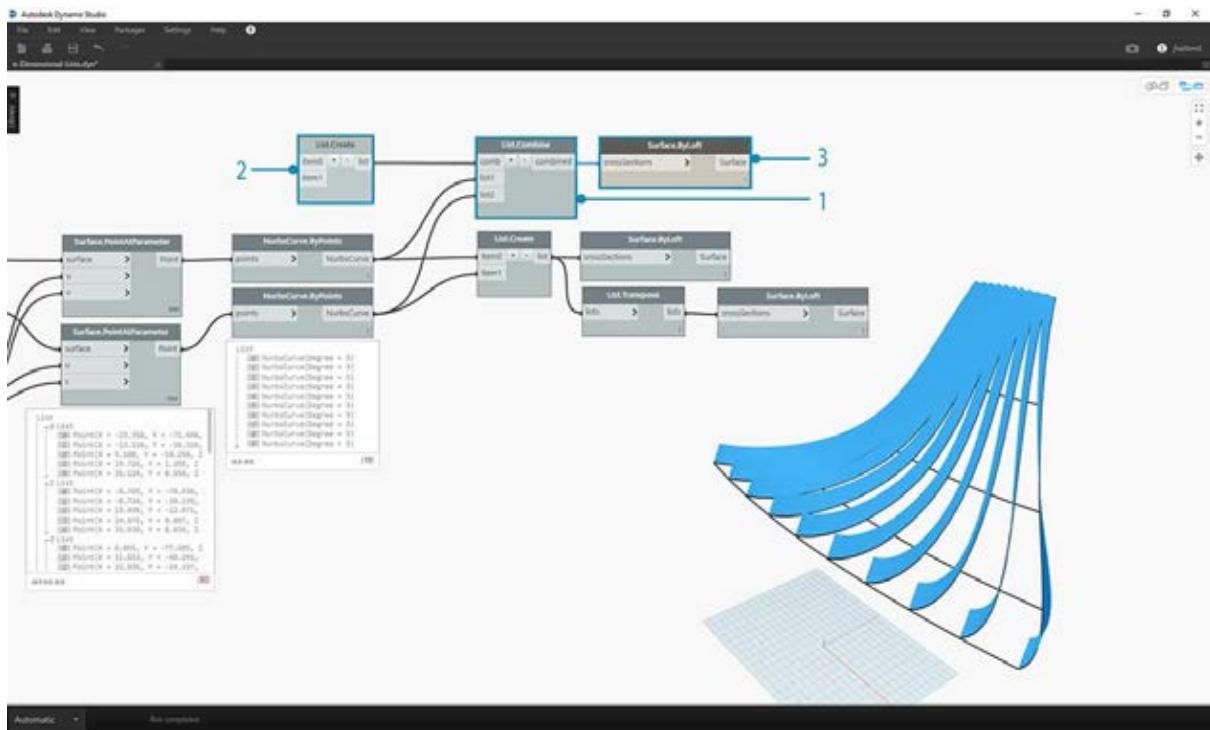
1. Verbinden Sie wie in der vorigen Übungslektion die Ausgaben mit zwei *NurbsCurve.ByPoints*-Blöcken.
2. Die Dynamo-Vorschau zeigt zwei Kurvengruppen für die beiden Oberflächen.



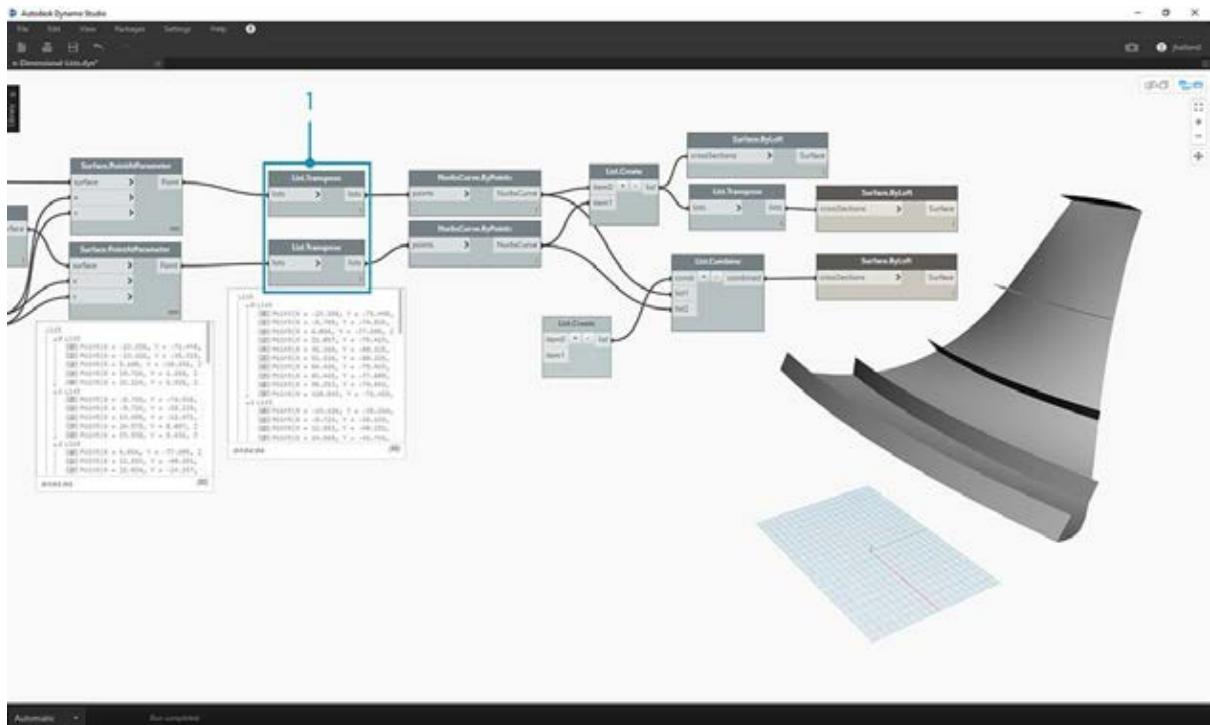
1. Mithilfe von *List.Create* können Sie die beiden Kurvengruppen in einer Liste von Listen kombinieren.
2. Die Ausgabe zeigt zwei Listen mit je zehn Elementen für die beiden Gruppen verbundener Nurbs-Kurven.
3. Mithilfe von *Surface.ByLoft* können Sie diese Datenstruktur visuell verdeutlichen. Mithilfe dieses Blocks verbinden Sie sämtliche Kurven in jeder der beiden Unterlisten durch eine Erhebung.



1. Mithilfe von *List.Transpose* werden die Spalten und Zeilen vertauscht. Dadurch werden zwei Listen mit je zehn Kurven in zehn Listen mit je zwei Kurven umgewandelt. Damit haben Sie für jede Nurbs-Kurve eine Beziehung zu ihrer Nachbarkurve auf der anderen Oberfläche erstellt.
2. Mit *Surface.ByLoft* erhalten Sie eine gerippte Struktur.



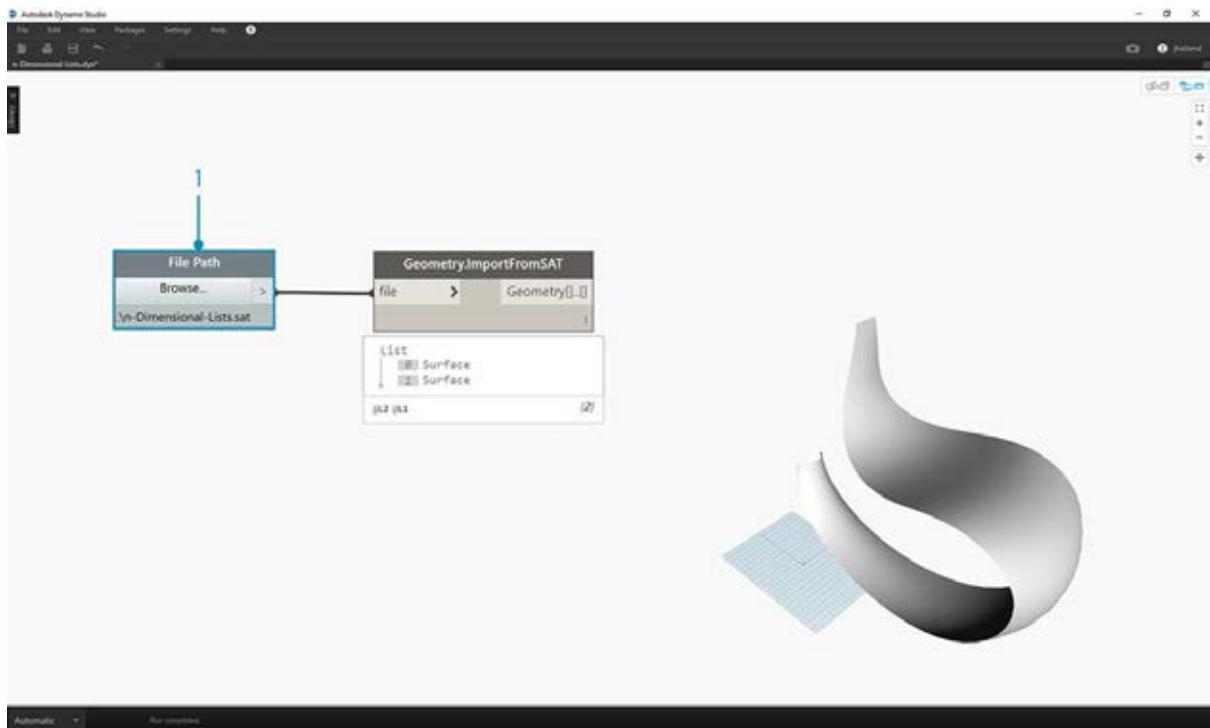
1. Als Alternative zu *List.Transpose* können Sie *List.Combine* verwenden. Damit wird ein "Kombinator" für die beiden Unterlisten ausgeführt.
2. In diesem Fall wird *List.Create* als "Kombinator" verwendet, um eine Liste der Elemente innerhalb der Unterlisten zu erstellen.
3. Mithilfe des *Surface.ByLoft*-Blocks erhalten Sie dieselben Oberflächen wie im vorigen Schritt. Transpose ist in diesem Falle einfacher zu verwenden, bei komplexeren Datenstrukturen erweist sich *List.Combine* jedoch als zuverlässiger.



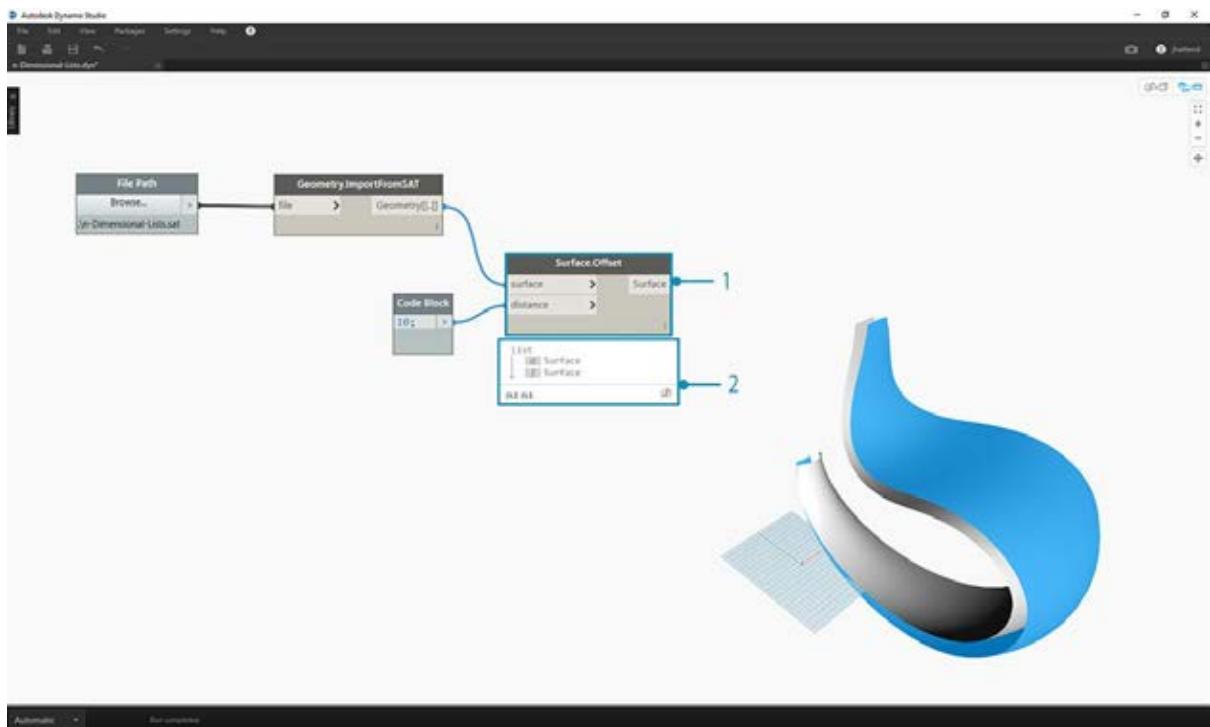
1. Sie können in einem der vorigen Schritte die Richtung der Kurven in der gerippten Struktur ändern, indem Sie List.Transpose vor der Verbindung zu NurbsCurve.ByPoints einfügen. Dadurch werden Spalten und Zeilen vertauscht und Sie erhalten 5 horizontale Rippen.

### Übungslektion: 3D-Listen

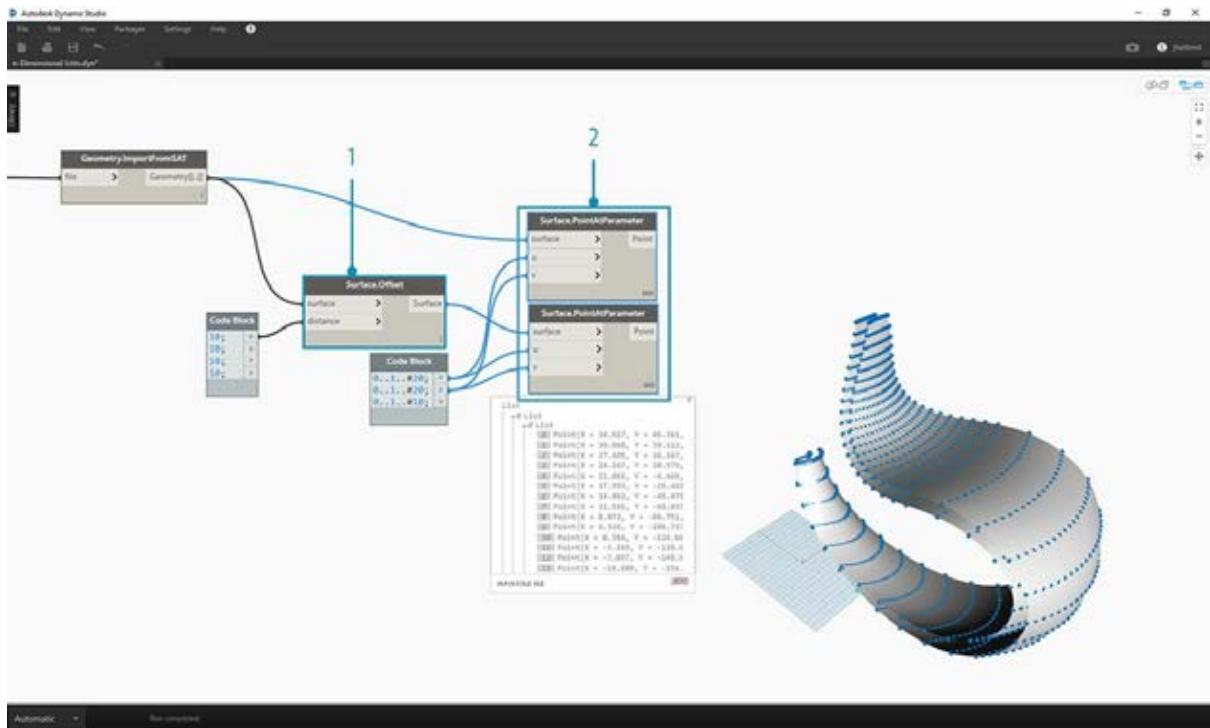
In diesem Abschnitt gehen Sie einen Schritt weiter. Sie arbeiten in dieser Übungslektion mit beiden importierten Oberflächen und erstellen eine komplexe Datenhierarchie. Dabei soll dieselbe Operation nach wie vor unter Verwendung derselben zugrunde liegenden Logik durchgeführt werden.



1. Beginnen Sie mit der Datei aus der vorherigen Übungslektion.



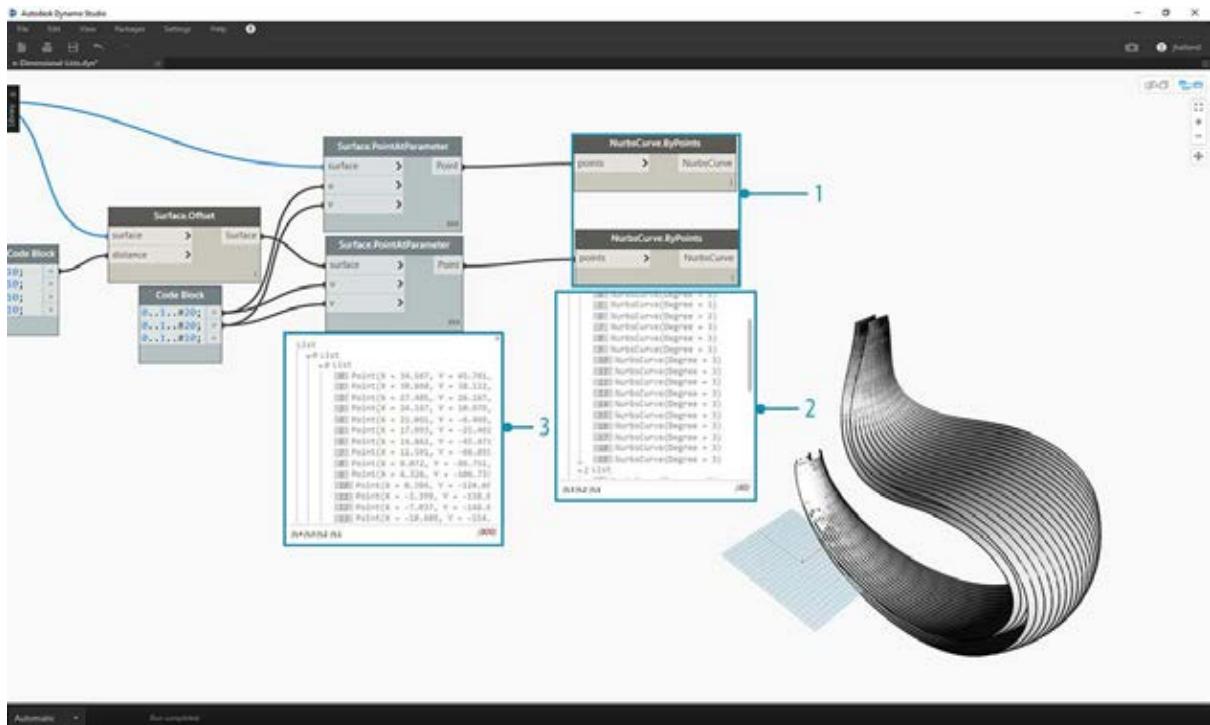
1. Verwenden Sie wie in der vorigen Übungslektion den *Surface.Offset*-Block, um einen Versatz mit dem Wert 10 zu erstellen.
2. Die Ausgabe zeigt, dass Sie mithilfe des Versatzblocks zwei Oberflächen erstellt haben.



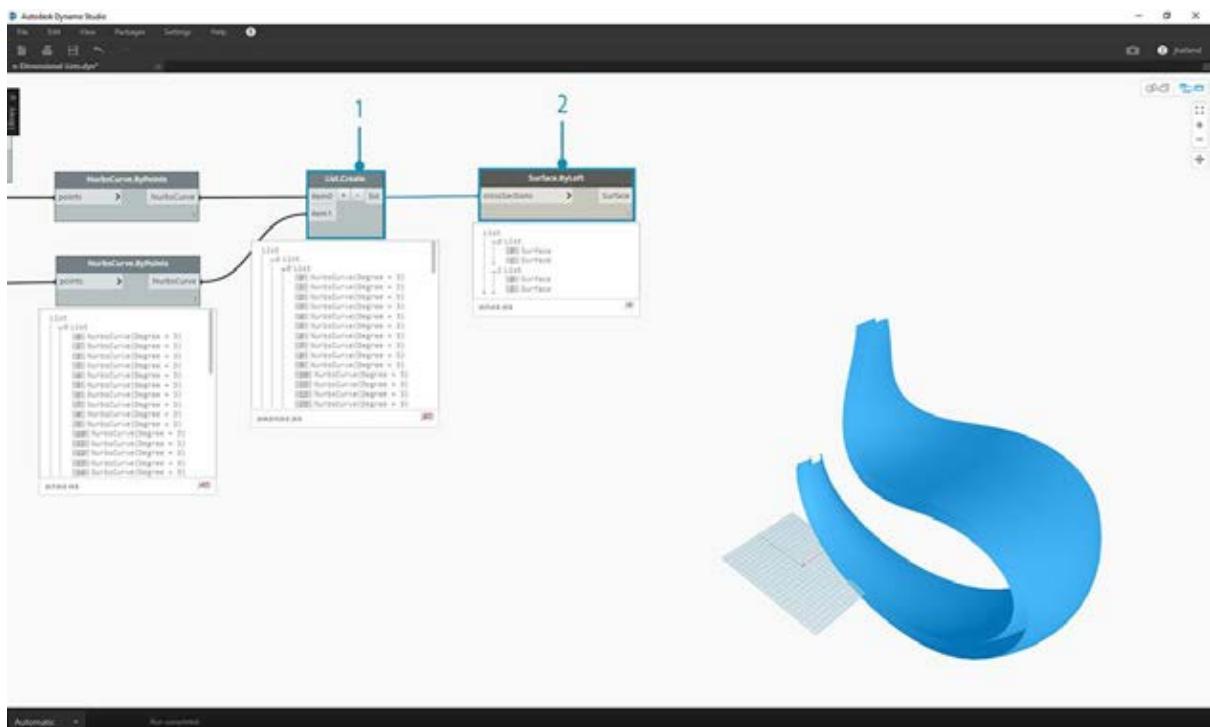
1. Definieren Sie auf dieselbe Weise wie in der vorigen Übungslektion einen Codeblock mit den folgenden beiden Codezeilen:

```
0..1..#20;  
0..1..#10;
```

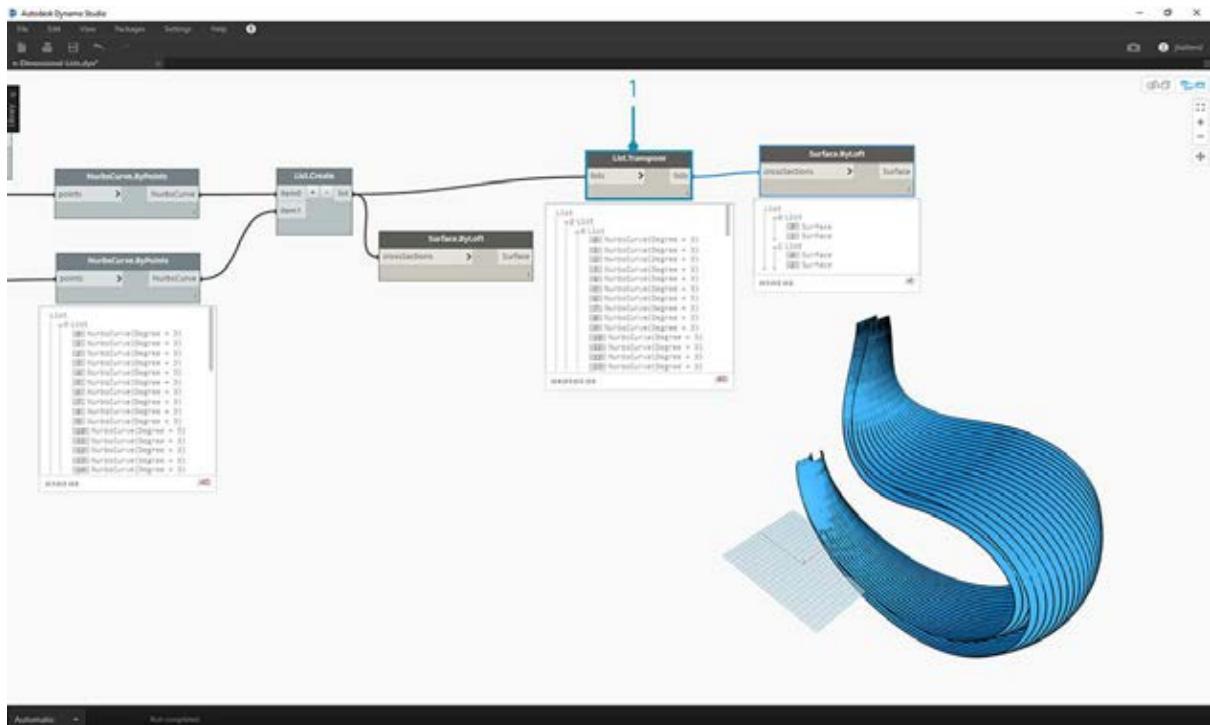
1. Verbinden Sie diese Ausgaben mit zwei *Surface.PointAtParameter*-Blöcken, jeweils mit der Vergitterung "Kreuzprodukt". Verbinden Sie einen dieser Blöcke mit den ursprünglichen und den anderen mit den versetzten Oberflächen.



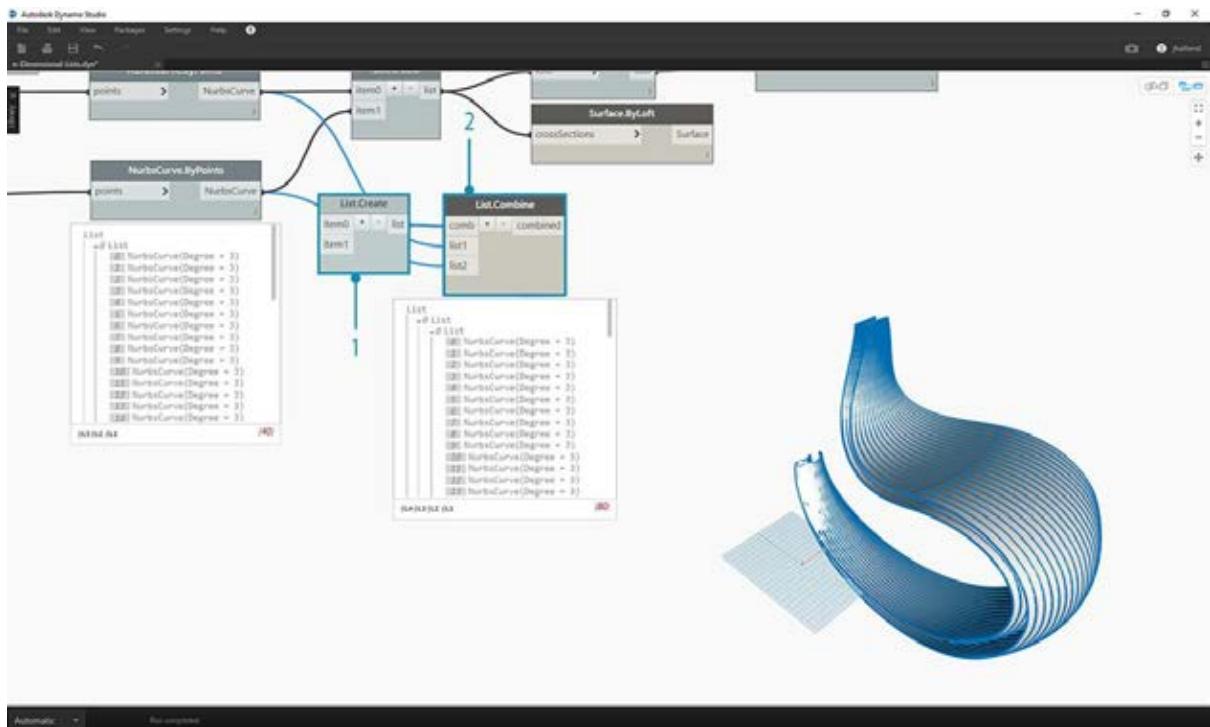
1. Verbinden Sie wie in der vorigen Übungslektion die Ausgaben mit zwei NurbsCurve.ByPoints-Blöcken.
2. Die Ausgabe der NurbsCurve.ByPoints-Blöcke ist eine Liste aus zwei Listen mit komplexerer Struktur als bei denjenigen in der vorigen Übungslektion. Die Daten werden durch die zugrunde liegende Oberfläche kategorisiert: Damit haben Sie der Datenstruktur eine weitere Ebene hinzugefügt.
3. Im Surface.PointAtParameter-Block ist eine komplexere Struktur zu erkennen: Sie haben eine Liste aus Listen von Listen erstellt.



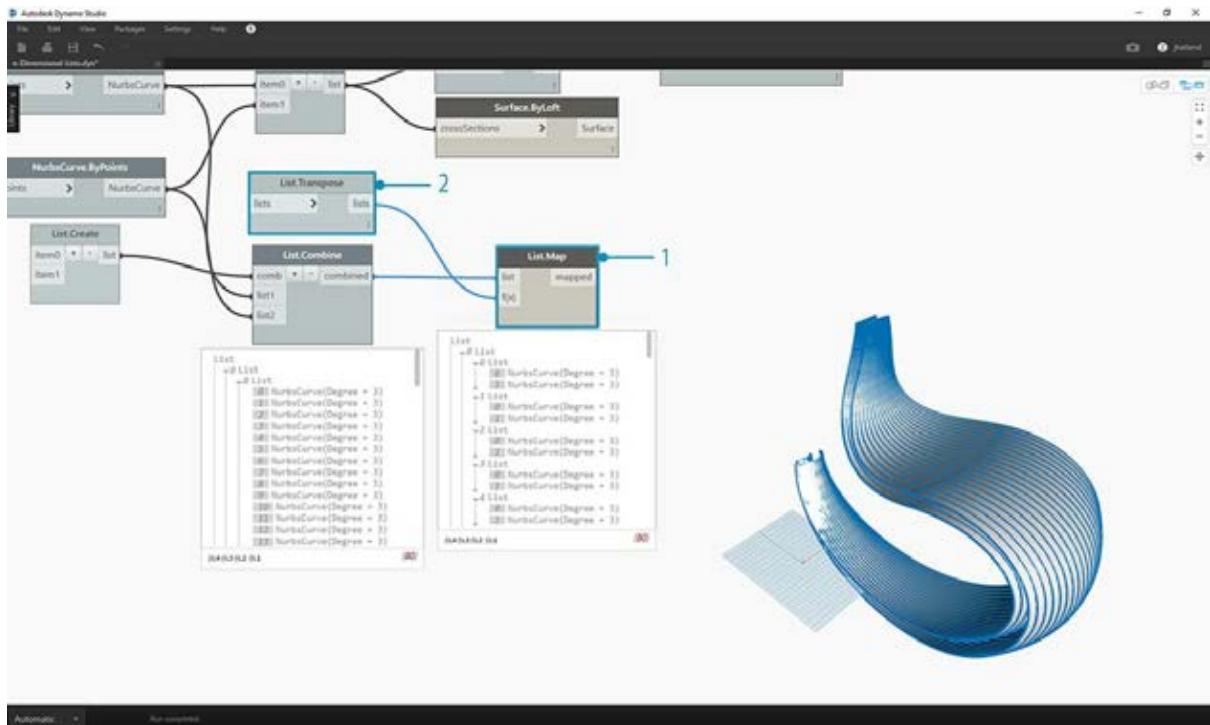
1. Führen Sie mithilfe des List.Create-Blocks die Nurbs-Kurven zu ein und derselben Datenstruktur zusammen, wobei eine Liste aus Listen von Listen entsteht.
2. Durch Verbinden eines Surface.ByLoft-Blocks erhalten Sie eine Version der ursprünglichen Oberflächen, die in ihrer eigenen Liste wie aus der ursprünglichen Datenstruktur erstellt erhalten bleiben.



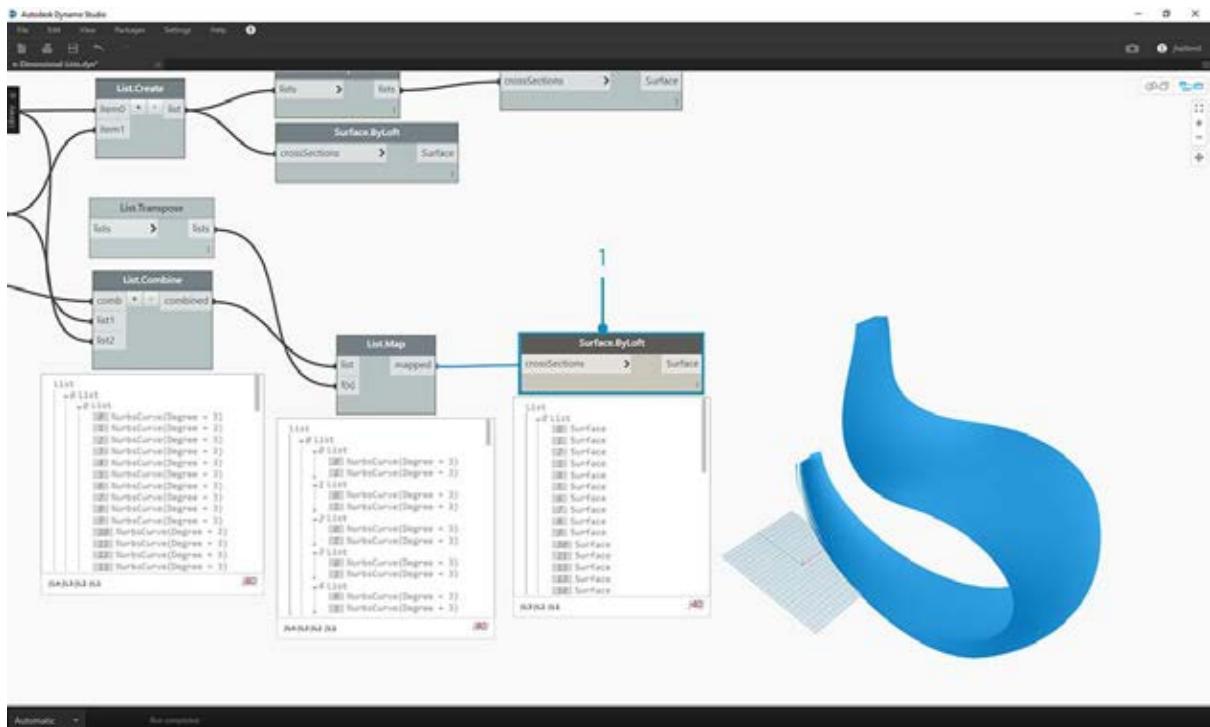
1. In der vorigen Übung konnten Sie mithilfe von *List.Transpose* eine gerippte Struktur erstellen. Dies ist hier nicht möglich. Die Funktion *Transpose* ist für zweidimensionale Listen vorgesehen. Hier liegt eine dreidimensionale Liste vor, die ein "Vertauschen von Spalten und Zeilen" nicht ohne Weiteres zulässt. Da Listen Objekte sind, können mit *List.Transpose* zwar die Listen mit den Unterlisten vertauscht werden, die Nurbs-Kurven, die sich eine Ebene tiefer in der Hierarchie befinden, bleiben dabei jedoch unverändert.



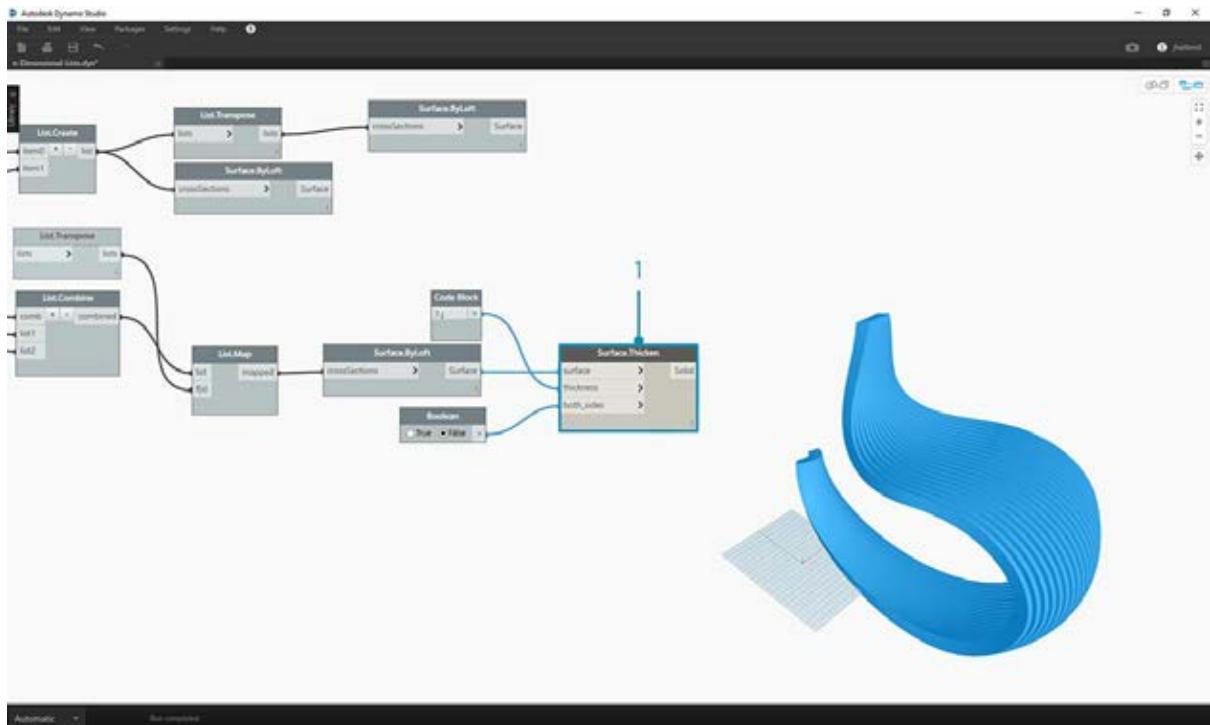
1. In diesem Fall ist *List.Combine* besser geeignet. Für komplexere Datenstrukturen sollten *List.Map*- und *List.Combine*-Blöcke zum Einsatz kommen.  
2. Mit *List.Create* als "Kombinator" erhalten Sie eine Datenstruktur, die in diesem Fall besser verwendbar ist.



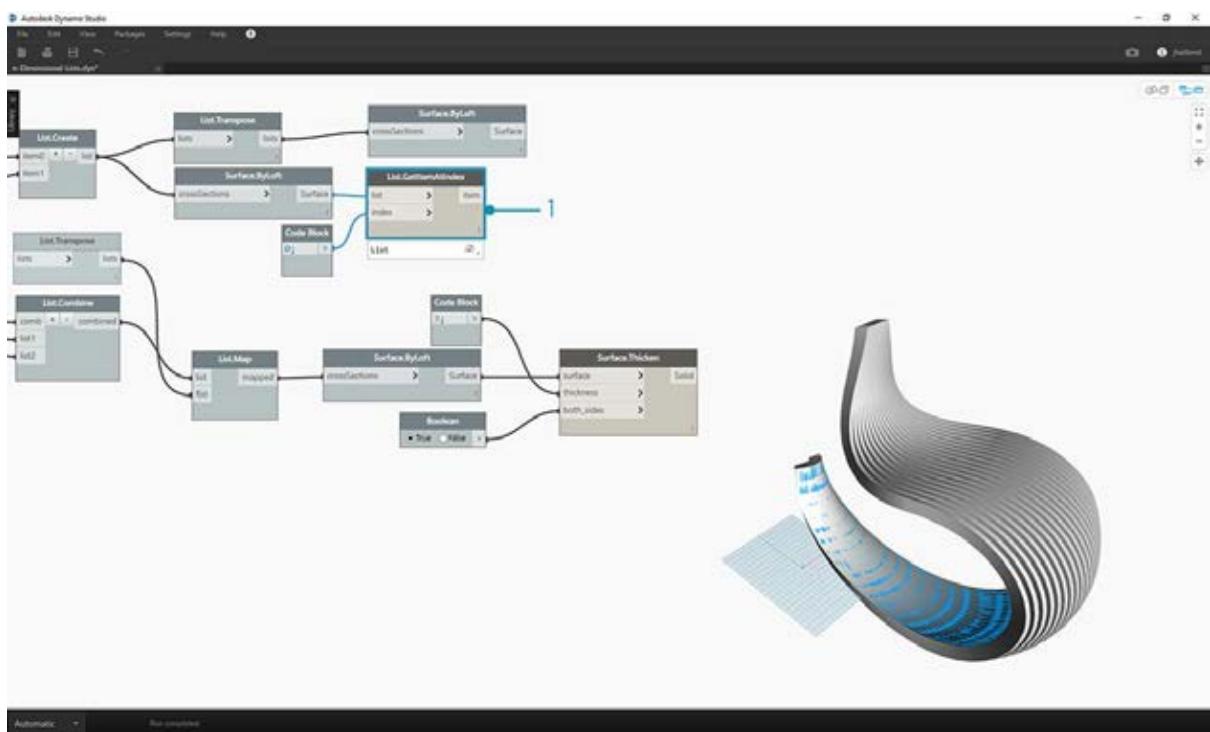
1. Die Datenstruktur muss auf der nächsttieferen Ebene der Hierarchie nach wie vor vertauscht werden. Verwenden Sie hierfür *List.Map*. Dies funktioniert auf dieselbe Weise wie *List.Combine*, wobei jedoch nur eine Liste anstelle mehrerer eingegeben wird.
2. Als Funktion für *List.Map* wird *List.Transpose* eingegeben, wodurch die Spalten und Zeilen der Unterlisten innerhalb der Hauptliste vertauscht werden.



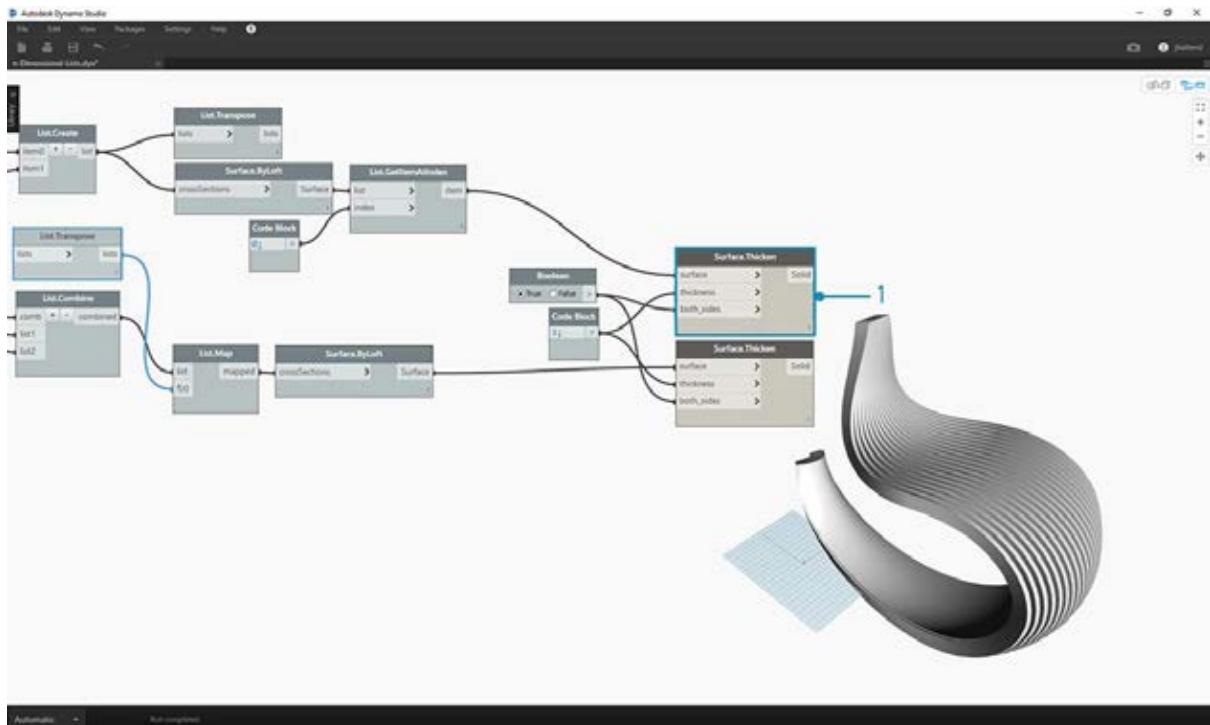
1. Schließlich können Sie die Nurbs-Kurven in der vorgesehenen Datenhierarchie durch eine Erhebung miteinander verbinden und erhalten eine gerippte Struktur.



- Fügen Sie der Geometrie mithilfe eines *Surface.Thicken*-Blocks Tiefe hinzzu.



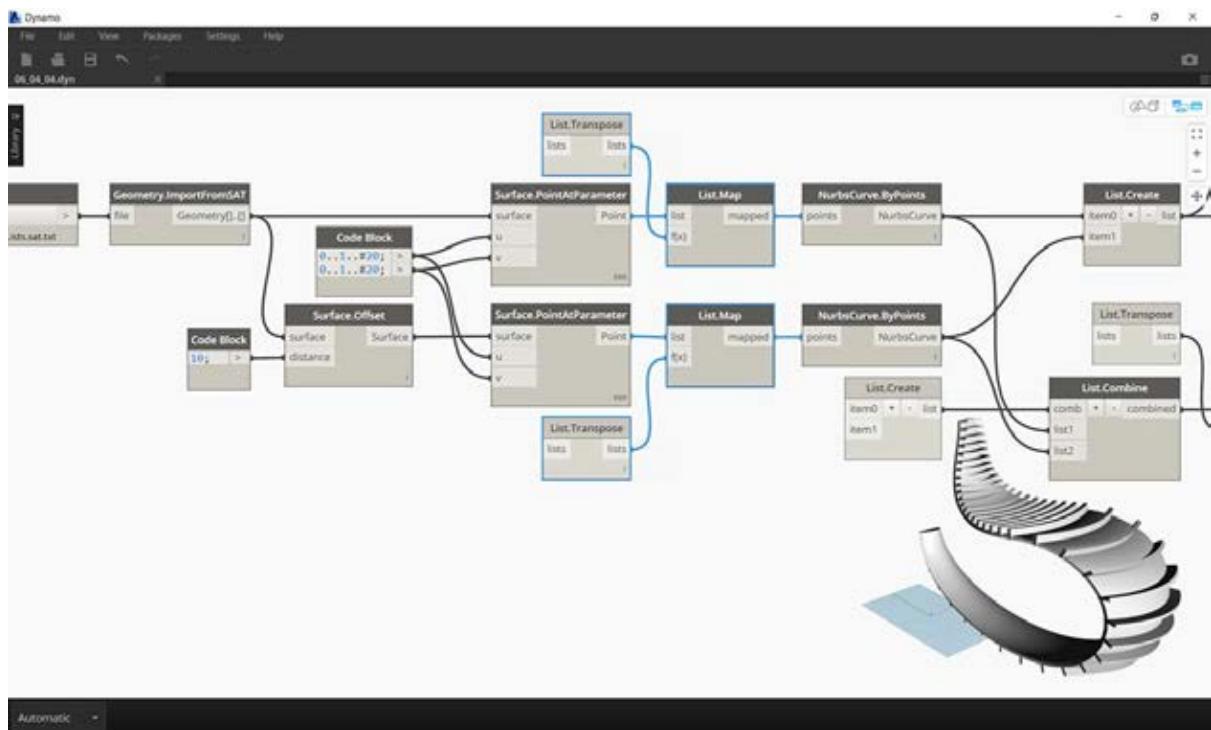
- Eine Deckschicht an der Rückseite dieser Struktur scheint passend. Wählen Sie daher mithilfe von *List.GetItemAtIndex* die hintere der beiden durch die Erhebung verbundenen Oberflächen aus den vorigen Schritten aus.



1. Mit der Verstärkung dieser ausgewählten Oberflächen ist die Untergliederung vollständig.

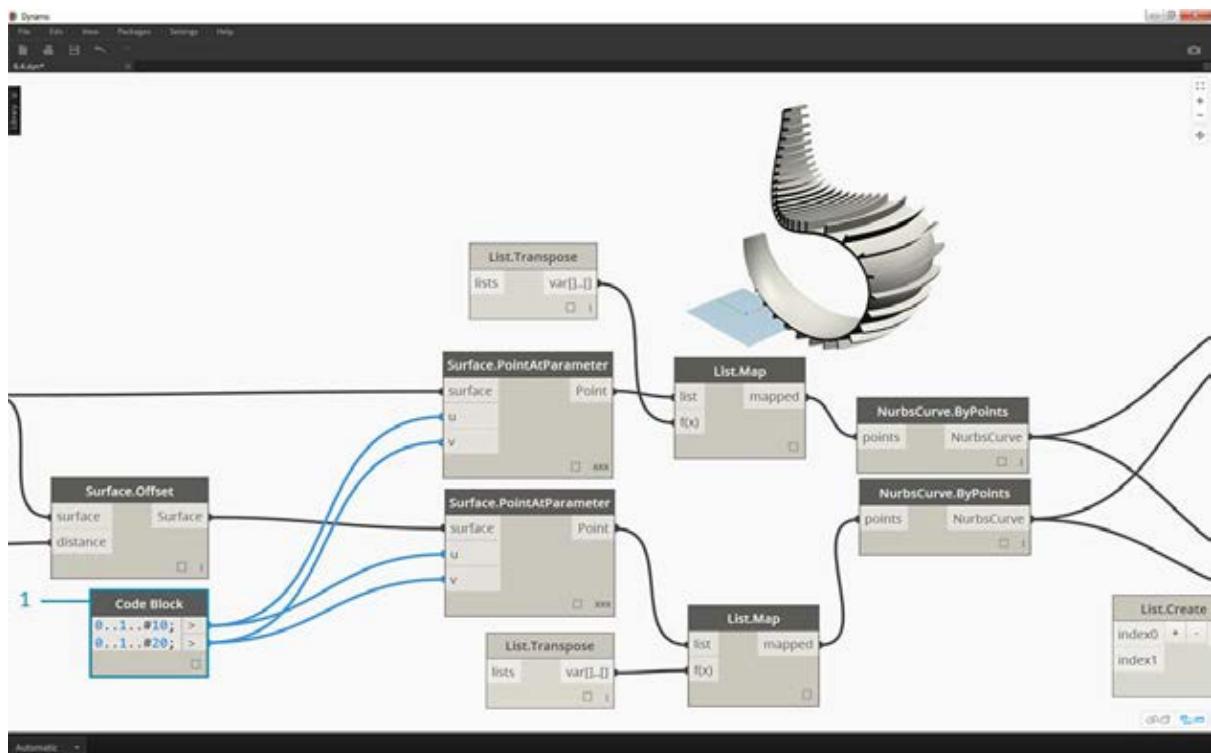


Das Ergebnis ist vielleicht kein sonderlich bequemer Schaukelstuhl, aber er enthält erhebliche Datenmengen.



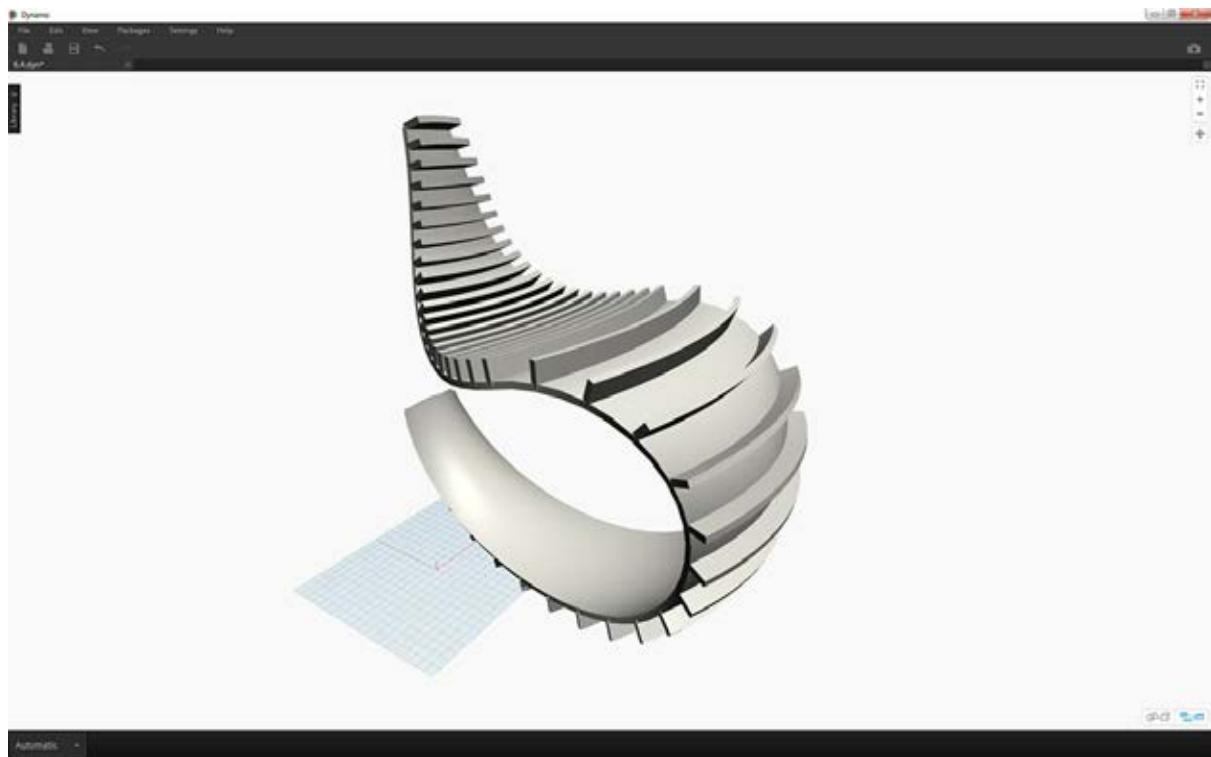
Im letzten Schritt kehren Sie die Richtung der Rippen um. Bei diesem Vorgang wird in ähnlicher Weise wie in der vorigen Übungslektion die Funktion Transpose verwendet.

1. Da in der Hierarchie eine weitere Ebene vorhanden ist, müssen Sie *List.Map* zusammen mit *ListTranspose* als Funktion verwenden, um die Richtung der Nurbs-Kurven zu ändern.



1. Es ist eventuell sinnvoll, die Anzahl der Rippen zu erhöhen. Ändern Sie daher den Codeblock wie folgt:

```
0..1..#20;  
0..1..#10;
```

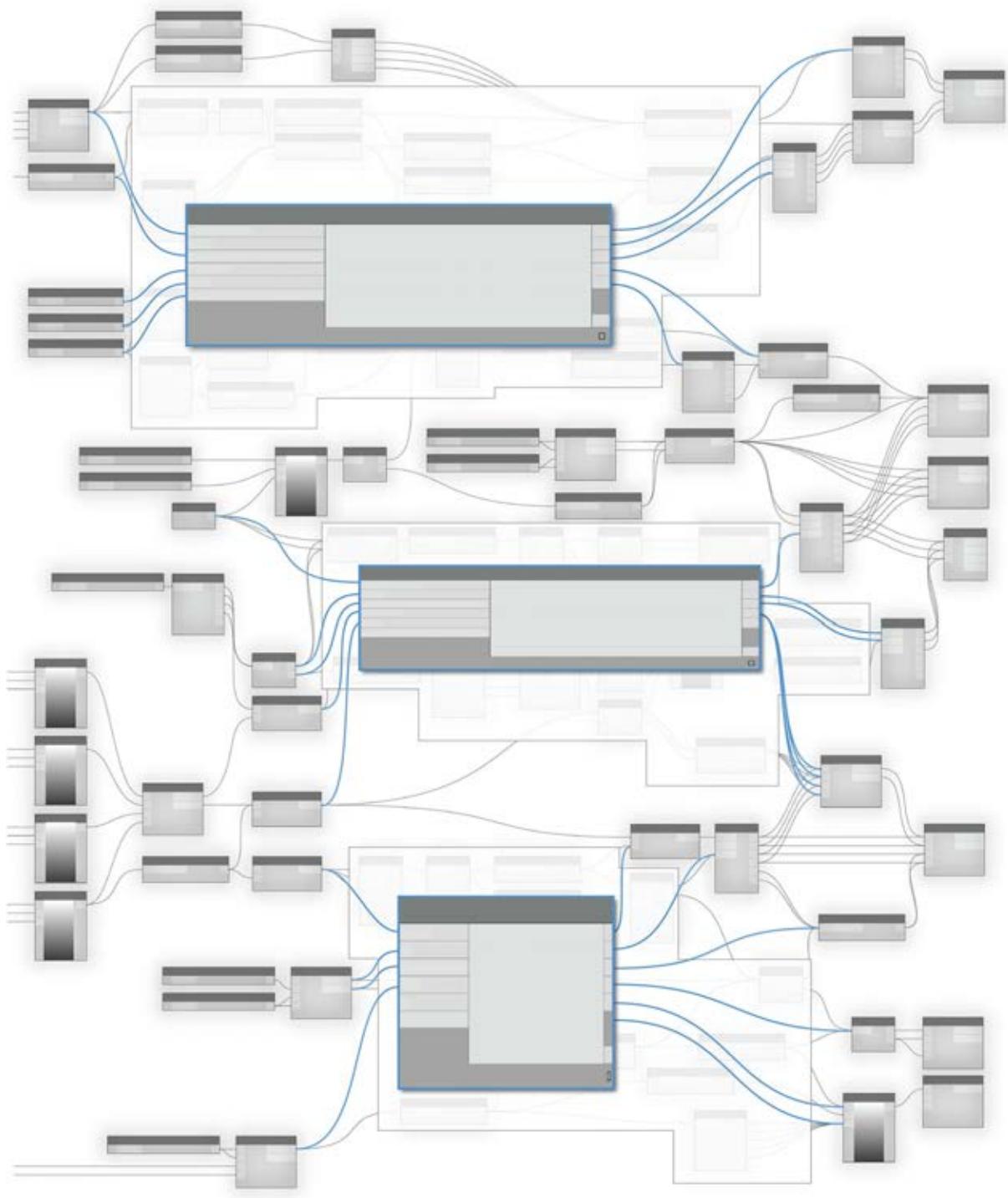


Während die erste Version des Schaukelstuhls eher elegant wirkte, punktet das zweite Modell durch robuste, sportliche Qualitäten.

## **Codeblöcke und DesignScript**

### **Codeblöcke und DesignScript**

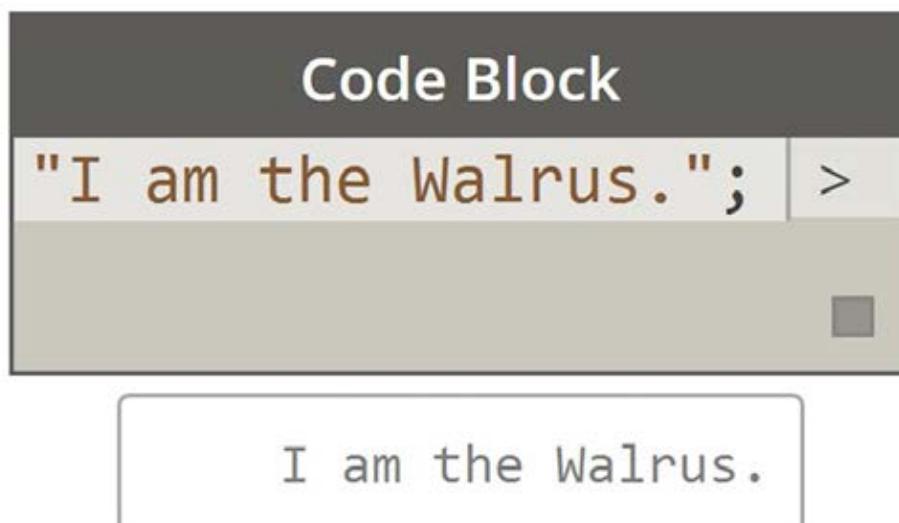
Codeblöcke sind eine spezielle Funktion in Dynamo, die eine dynamische Verbindung zwischen visueller und textbasierter Programmierumgebung darstellt. In einem Codeblock haben Sie Zugriff auf sämtliche Dynamo-Blöcke und können ein komplettes Diagramm in einem einzigen Block definieren. Lesen Sie die Informationen in diesem Kapitel sorgfältig, da Codeblöcke zu den wichtigsten Komponenten von Dynamo gehören.



# Was ist ein Codeblock?

## Was ist ein Codeblock?

Codeblöcke geben Zugang zu DesignScript, der Programmiersprache, die Dynamo zugrunde liegt. DesignScript ist eine völlig neu entwickelte, leicht lesbare und knappe Programmiersprache speziell für experimentelle Arbeitsabläufe, die sowohl sofortiges Feedback für kleine Codeabschnitte als auch Skalierungsmöglichkeiten für umfangreiche und komplexe Interaktionen bietet. DesignScript ist zugleich das Kernstück der Engine, die die meisten Funktionen von Dynamo "hinter den Kulissen" steuert. Für fast jede Funktion in Dynamo-Blöcken und -Interaktionen ist eine entsprechende Funktion in der Skriptsprache vorhanden. Aus diesem Grund stehen einzigartige Möglichkeiten für einen nahtlosen Übergang zwischen Block-Interaktionen und Skripterstellung zur Verfügung.

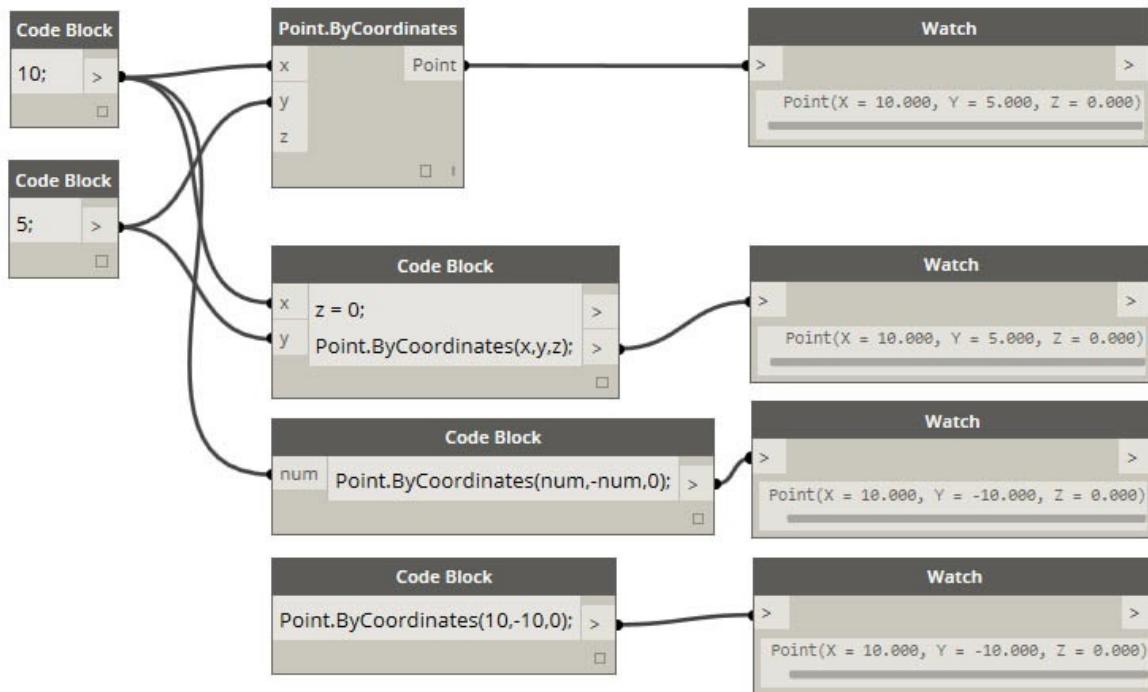


Blöcke können automatisch in Textsyntax konvertiert werden, etwa um Anfängern den Einstieg in DesignScript zu erleichtern oder um ganz einfach größere Abschnitte eines Diagramms auf kleinerem Raum zusammenzufassen. Hierfür steht die Funktion Block zu Code zur Verfügung, die im Abschnitt [DesignScript-Syntax](#) genauer beschrieben wird. Benutzer mit umfassender Erfahrung können in Codeblöcken unter Verwendung vieler Standardparadigmen der Codeerstellung benutzerdefinierte Mashups bestehender Funktionen sowie eigene Beziehungen erstellen. Benutzer, die über einige Erfahrung verfügen, aber keine Experten sind, finden zahlreiche Shortcuts und Codeabschnitte, mit deren Hilfe sie schneller an ihren Entwürfen arbeiten können. Der Begriff "Codeblock" mag zwar für Benutzer ohne Programmierkenntnisse etwas zu fachspezifisch wirken, die Codeblöcke selbst sind jedoch benutzerfreundlich und robust. Für den Einstieg können Codeblöcke mit einem Minimum an Codeerstellung effizient eingesetzt werden, während Benutzer mit fortgeschrittenen Kenntnissen Skriptdefinitionen definieren und gegebenenfalls an anderer Stelle in einer Dynamo-Definition erneut aufrufen können.

## Codeblock: kurzer Überblick

Codeblöcke sind, kurz zusammengefasst, eine Oberfläche für Textskripts innerhalb einer Umgebung für visuelles Skripting. Sie können für Zahlen, Zeichenfolgen, Formeln und andere Datentypen verwendet werden. Die Codeblock-Funktion wurde für Dynamo entwickelt. Sie können daher beliebige Variable im Codeblock definieren, die anschließend automatisch den Eingaben des Blocks hinzugefügt werden:

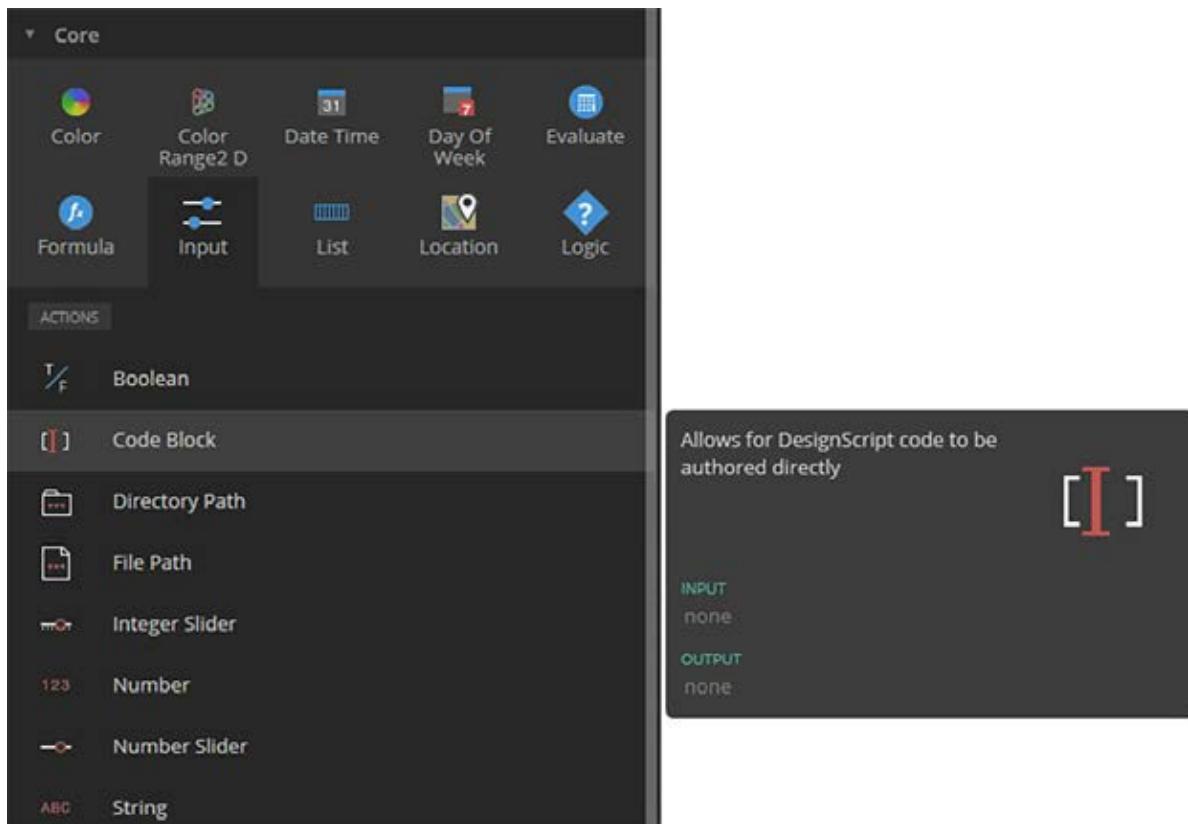
Bei Codeblöcken kann der Benutzer flexibel entscheiden, wie die Eingaben festgelegt werden sollen. Die folgenden Abbildungen zeigen verschiedene Möglichkeiten zum Erstellen eines einfachen Punkts mit den Koordinaten (10, 5, 0):



Während Sie weitere Funktionen aus der Bibliothek kennenlernen, erweist sich eventuell die Eingabe von "Point.ByCoordinates" als leichter und schneller als die Suche nach dem passenden Block in der Bibliothek. Wenn Sie beispielsweise "Point." eingeben, zeigt Dynamo eine Liste möglicher Funktionen an, die auf einen Punkt angewendet werden können. Dadurch gestaltet sich die Skripterstellung intuitiver und die Anwendung von Funktionen in Dynamo ist leichter zu erlernen.

### **Erstellen von Codeblock-Blöcken**

Der Codeblock befindet sich unter *Core > Input > Actions > Code Block*. Sie können den Codeblock jedoch auch wesentlich schneller durch einfaches Doppelklicken im Ansichtsbereich aufrufen. Dieser Block wird so häufig verwendet, dass ihm uneingeschränkte Doppelklickfunktionen zugewiesen wurden.



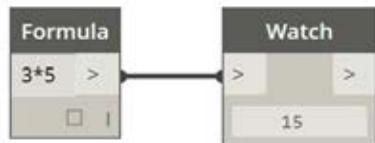
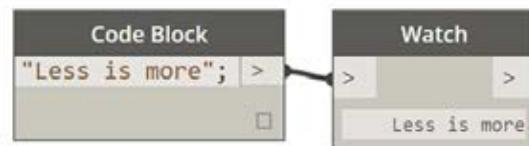
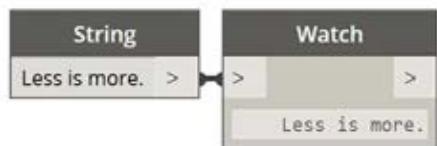
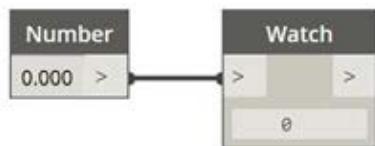
## Zahlen, Zeichenfolgen und Formeln

Codeblöcke können auch flexibel für unterschiedliche Datentypen eingesetzt werden. Die Benutzer können rasch Zahlen, Zeichenfolgen und Formeln definieren und der Codeblock liefert die gewünschte Ausgabe.

Die folgende Abbildung zeigt, dass der "herkömmliche" Ablauf für diese Angaben etwas umständlich ist: Der Benutzer sucht in der Benutzeroberfläche nach dem gewünschten Block, fügt diesen im Ansichtsbereich hinzu und gibt die Daten ein. Einen Codeblock hingegen kann der Benutzer durch Doppelklicken im Ansichtsbereich aufrufen, um anschließend den benötigten Datentyp in der entsprechenden Syntax einzugeben.

"Old School"

Code Blocks



Die Blöcke *number*, *string* und *formula* sind drei Beispiele für Dynamo-Blöcke, die im Vergleich zu *Code Block* als veraltet betrachtet werden könnten.

# DesignScript-Syntax

## DesignScript-Syntax

Die Namen von Blöcken in Dynamo weisen ein gemeinsames Merkmal auf: Sie verwenden eine Syntax mit Punkt "." ohne Leerzeichen. Dies ist der Fall, da die tatsächliche Skript-Syntax als Text ganz oben im jeweiligen Block verwendet wird. Der Punkt "." (bzw. die *Punktnotation*) dient dazu, ein Element von den möglichen Methoden abzugrenzen, die aufgerufen werden können. Dies vermittelt auf einfache Weise zwischen visueller und textbasierter Skripterstellung.



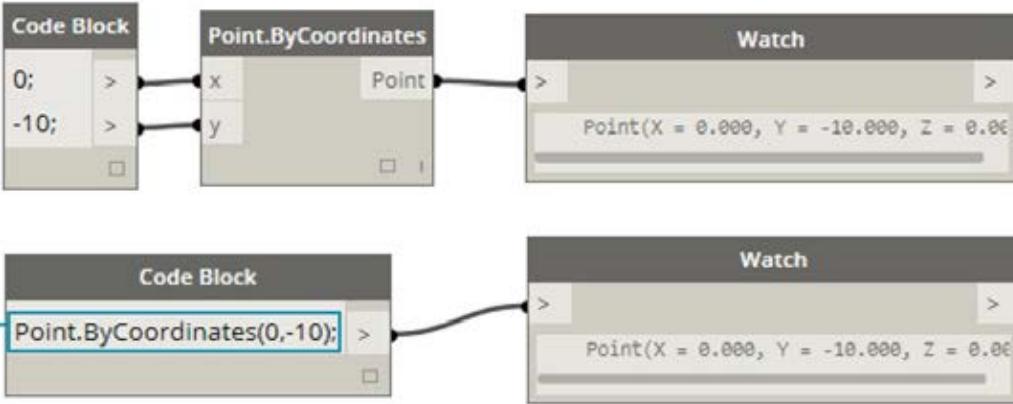
Zur allgemeinen Veranschaulichung der Punktnotation dient hier die parametrische Behandlung eines Apfels in Dynamo. Im Folgenden werden einige "Methoden" betrachtet, die Sie für den Apfel ausführen können (bevor Sie ihn essen).  
(Anmerkung: Diese Methoden sind keine echten Dynamo-Methoden.)

Klartext	Punktnotation	Ausgabe
Welche Farbe hat der Apfel?	Apfel.Farbe	Rot
Ist der Apfel reif?	Apfel.istReif	Wahr
Wie viel wiegt der Apfel?	Apfel.Gewicht	170 g
Woher kommt der Apfel?	Apfel.Übergeordnet	Baum
Was kann aus dem Apfel entstehen?	Apfel.Untergeordnet	Kerne
Wurde der Apfel hier in der Gegend angebaut?	Apfel.EntfernungVonObstgarten	100 km

Die Ausgabedaten in der Tabelle lassen darauf schließen, dass dieser Apfel schmackhaft ist. Ich glaube, ich *Apfel.Essen()*.

## Punktnotation in Codeblöcken

Behalten Sie dieses anschauliche Beispiel im Gedächtnis, wenn jetzt anhand von *Point.ByCoordinates* gezeigt wird, wie Sie mithilfe von Codeblöcken einen Punkt erstellen können:



Die *Code Block*-Syntax `Point.ByCoordinates(0,10);` gibt dasselbe Ergebnis aus wie der *Point.ByCoordinates*-Block in Dynamo, allerdings genügt ein Block, um den Punkt zu erstellen. Dieses Verfahren ist effizienter als die Verbindung separater Blöcke mit *x* und *y*.

1. Indem Sie *Point.ByCoordinates* im Codeblock verwenden, legen Sie die Eingabewerte in derselben Reihenfolge fest wie im vorgegebenen Block (*x*, *y*).

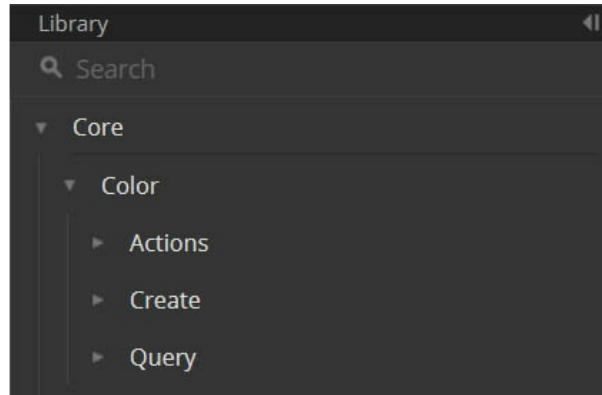
## Aufrufen von Blöcken

Sie können beliebige Blöcke aus der Bibliothek mithilfe eines Codeblocks aufrufen. Ausgenommen hiervon sind spezielle *Blöcke für die Benutzeroberfläche*, d. h. Blöcke mit einer speziellen Funktion in der Benutzeroberfläche. So können Sie beispielsweise *Circle.ByCenterPointRadius* aufrufen; einen *Watch 3D*-Block aufzurufen, wäre jedoch wenig sinnvoll.

Es gibt generell drei Typen normaler Blöcke (dies umfasst die meisten Blöcke in der Bibliothek):

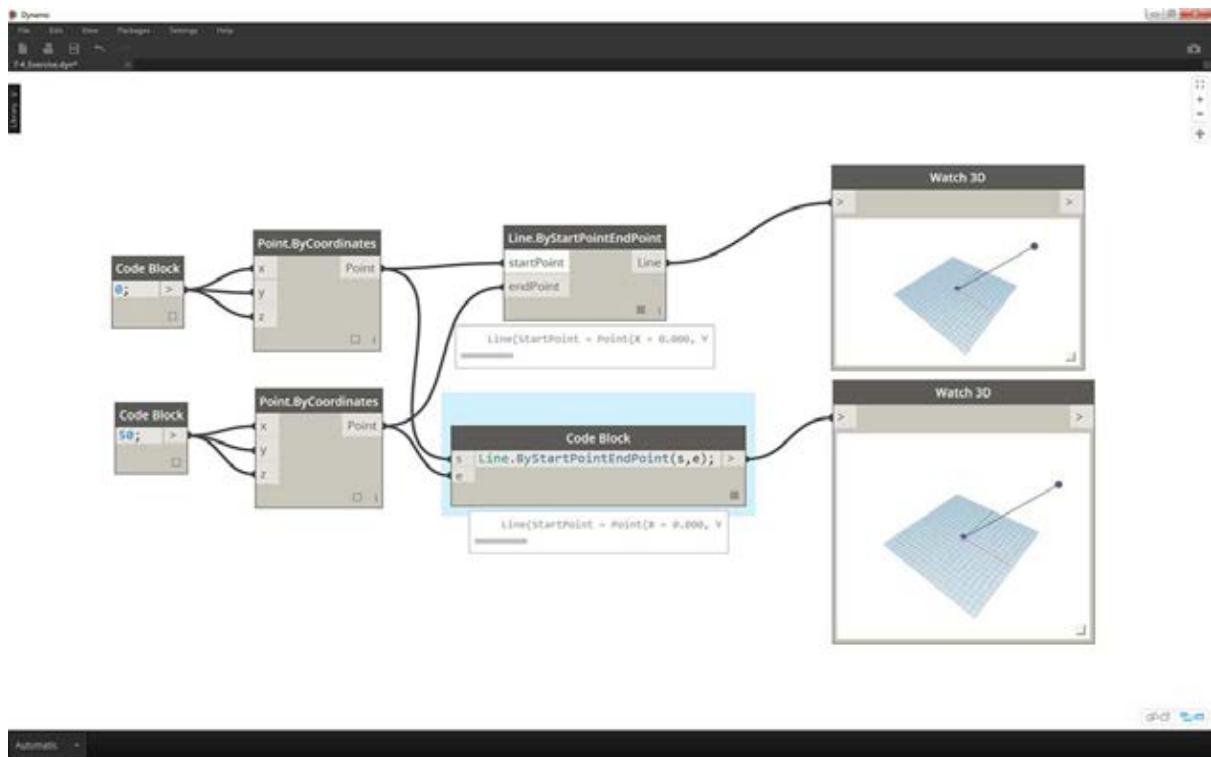
- **Erstellen:** Erstellt bzw. konstruiert ein Objekt.
- **Aktion:** Führt einen Vorgang für ein Objekt aus.
- **Abfrage:** Ruft eine Eigenschaft eines bestehenden Objekts ab.

Die Bibliothek ist anhand dieser Kategorien geordnet. Methoden bzw. Blöcke dieser drei Typen werden beim Aufruf in einem Codeblock unterschiedlich behandelt.



### Erstellen

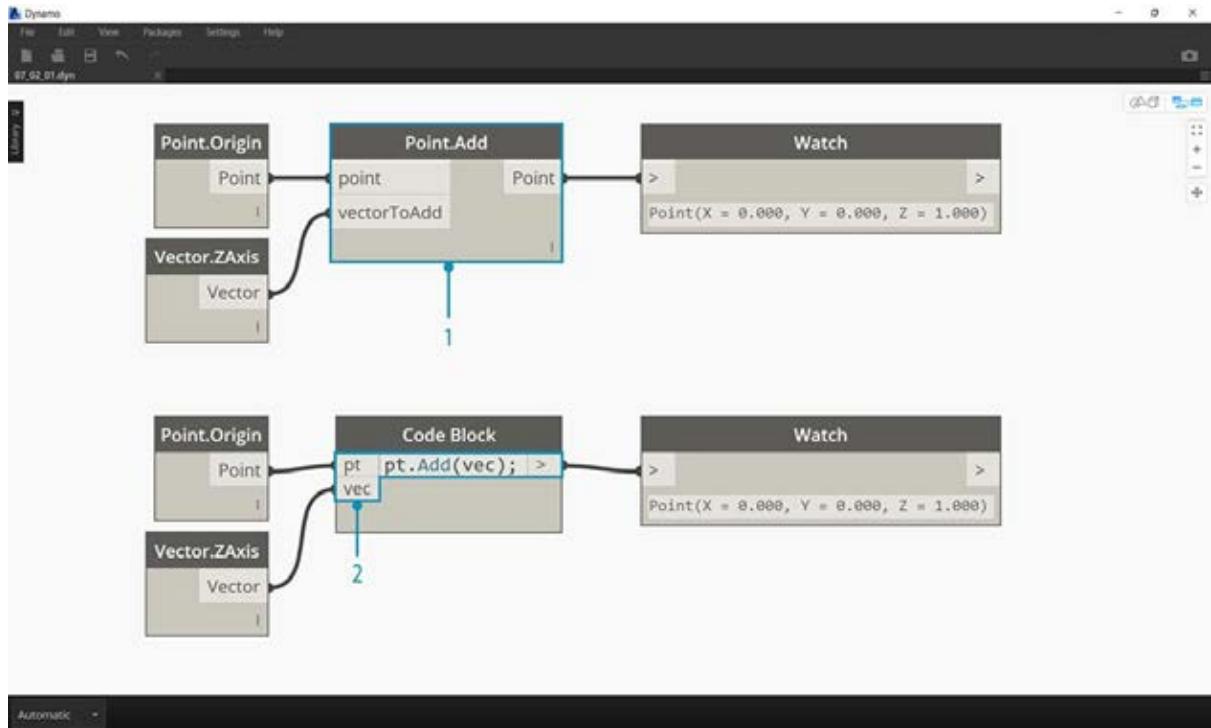
Mit Blöcken aus der Kategorie Erstellen wird völlig neue Geometrie konstruiert. Die Werte werden von links nach rechts in den Codeblock eingegeben. Dabei wird dieselbe Reihenfolge für die Eingaben eingehalten wie im eigentlichen Block von oben nach unten:



Der Vergleich des *Line.ByStartPointEndPoint*-Blocks und der entsprechenden Syntax im Codeblock zeigt, dass Sie dieselben Ergebnisse erhalten.

## Aktion

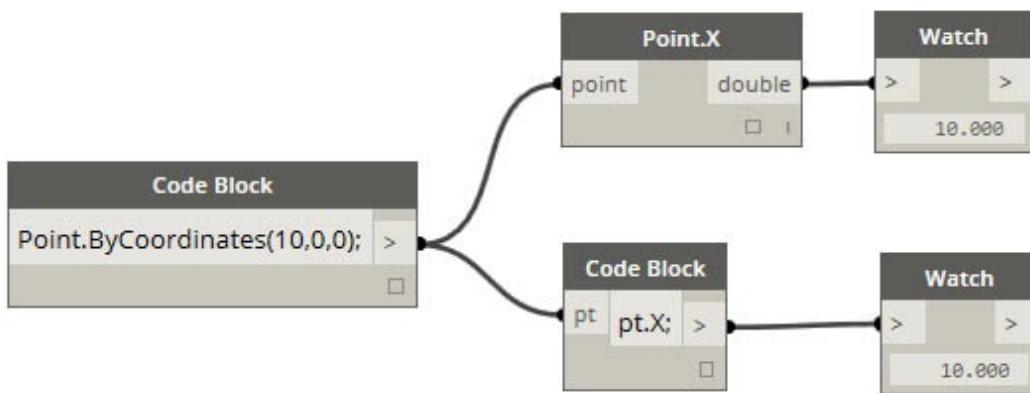
Eine Aktion ist ein Vorgang, den Sie für ein Objekt des gegebenen Typs ausführen. In Dynamo wird die vielen Programmiersprachen gemeinsame *Punktnotation* zum Anwenden von Aktionen auf Objekte verwendet. Dabei geben Sie zuerst das Objekt, dann einen Punkt und schließlich den Namen der Aktion ein. Die Eingabewerte für Aktionsmethoden werden genau wie bei Erstellungsmethoden in Klammern gesetzt, wobei Sie jedoch die erste Eingabe aus dem entsprechenden Block nicht angeben müssen. Stattdessen geben Sie das Element an, für Sie die Aktion durchzuführen:



- Der *Point.Add*-Block ist ein Aktionsblock, d. h., die Syntax funktioniert anders als zuvor.
- Die Eingaben sind: 1. der *point* und 2. der hinzuzufügende *vector*. Im *Code Block* hat der Punkt (das Objekt) die Bezeichnung "pt". Um einen Vektor "vec" zu "pt" hinzuzufügen, müssen Sie *pt.Add(vec)* schreiben: Objekt, Punkt, Aktion Add benötigt nur eine Eingabe, d. h. alle Eingaben aus dem *Point.Add*-Block ausgenommen die erste. Die erste Eingabe für den *Point.Add*-Block ist der Punkt selbst.

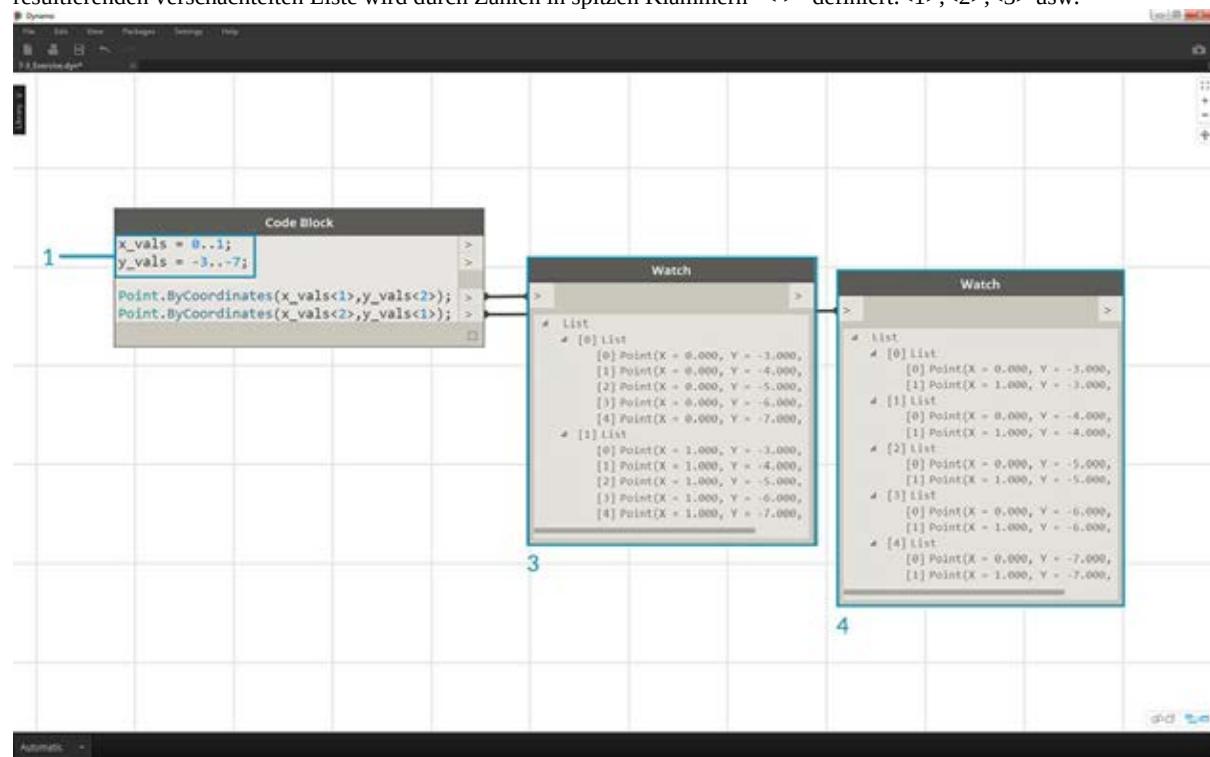
### Abfrage

Abfragemethoden rufen eine Eigenschaft eines Objekts ab. Da das Objekt selbst die Eingabe ist, müssen Sie keine weiteren Eingaben angeben. Es sind keine Klammern erforderlich.



### Wie funktioniert die Vergitterung?

Die Funktionsweise der Vergitterung in Blöcken unterscheidet sich von derjenigen in Codeblöcken. In Blöcken klickt der Benutzer mit der rechten Maustaste auf den jeweiligen Block und wählt die benötigte Vergitterungsoption. Codeblöcke bieten wesentlich mehr Kontrolle über die Strukturierung der Daten. In der Codeblock-Kurzschriftweise wird die Zuordnung mehrerer eindimensionaler Listen mithilfe von *Replikationsanleitungen* festgelegt. Die Hierarchie innerhalb der resultierenden verschachtelten Liste wird durch Zahlen in spitzen Klammern "<>" definiert: <1>, <2>, <3> usw.



- In diesem Beispiel werden mithilfe der Kurzschriftweise zwei Bereiche definiert. (Genauere Informationen zu Kurzschriftweisen erhalten Sie im folgenden Abschnitt dieses Kapitels.) Kurz gefasst:  $0..1$ ; entspricht  $\{0, 1\}$  und  $-3..-7$  entspricht  $\{-3, -4, -5, -6, -7\}$ . Als Ergebnis erhalten Sie Listen mit zwei x- und 5 y-

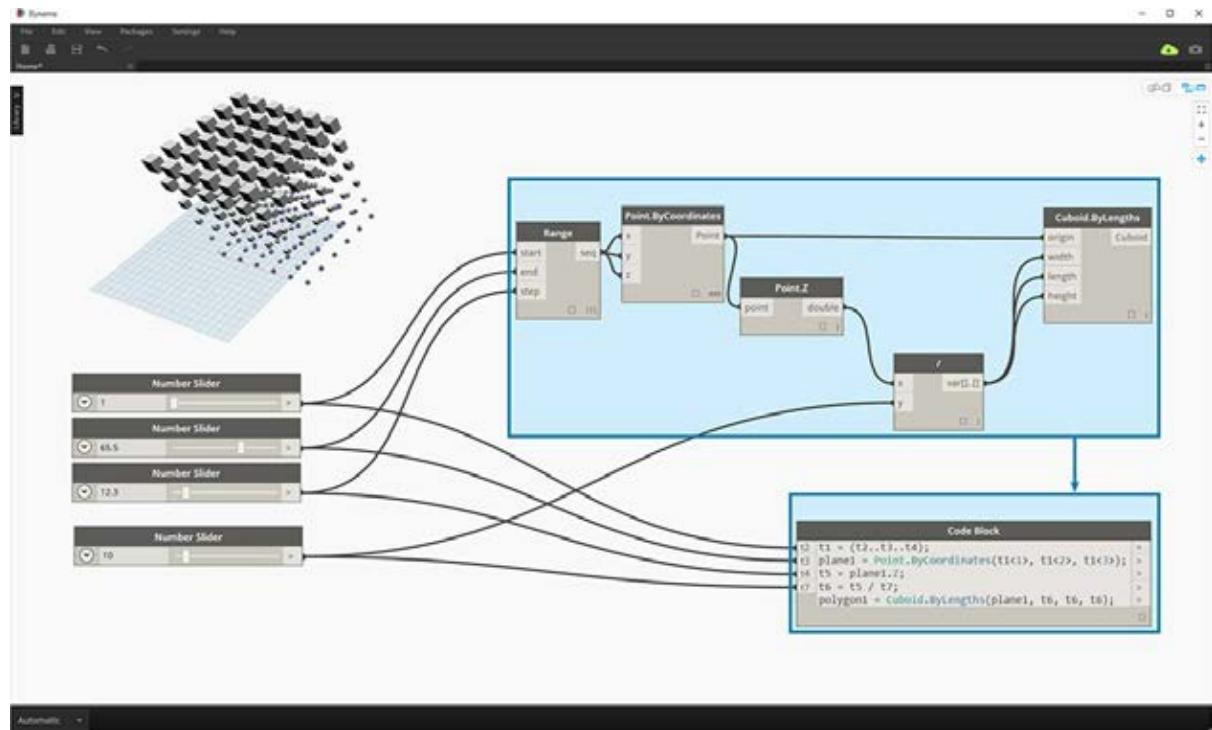
Werten. Ohne Replikationsanleitungen wird aus diesen nicht übereinstimmenden Listen eine Liste mit zwei Punkten erstellt, was der Länge der kürzesten Liste entspricht. Mithilfe der Replikationsanleitungen werden sämtliche möglichen Kombinationen aus zwei und fünf Koordinaten (d. h. ihr **Kreuzprodukt**) ermittelt.

2. Mit der Syntax `Point.ByCoordinates(x_vals<1>, y_vals<2>)`; erhalten Sie **zwei** Listen mit je **fünf** Einträgen.
3. Mit der Syntax `Point.ByCoordinates(x_vals<2>, y_vals<1>)`; erhalten Sie **fünf** Listen mit je **zwei** Einträgen.

Diese Notation ermöglicht es außerdem, welche der Listen der anderen übergeordnet sein soll, d. h., ob zwei Listen mit fünf Einträgen oder fünf Listen mit zwei Einträgen ausgegeben werden sollen. In diesem Beispiel bewirkt die Änderung der Reihenfolge der Replikationsanleitungen, dass als Ergebnis eine Liste entweder mit Zeilen von Punkten oder mit Spalten von Punkten ausgegeben wird.

## Block zu Code

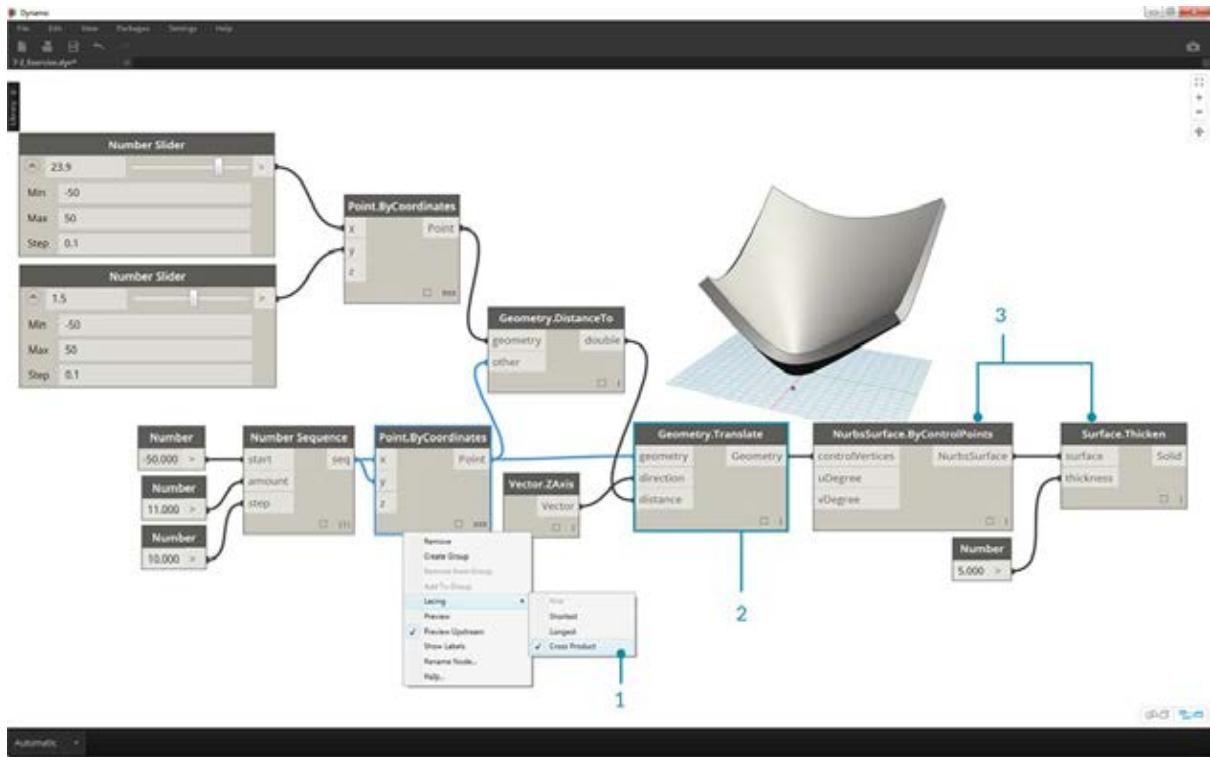
Für die oben beschriebenen Codeblock-Methoden ist eventuell eine gewisse Einarbeitung nötig. In Dynamo steht die Funktion Block zu Code zur Verfügung, die dies erleichtert. Um diese Funktion zu verwenden, wählen Sie eine Gruppe von Blöcken in Ihrem Dynamo-Diagramm aus, klicken mit der rechten Maustaste in den Ansichtsbereich und wählen Block zu Code. Dynamo fasst daraufhin diese Blöcke einschließlich aller Ein- und Ausgaben in einem Codeblock zusammen. Dies ist nicht nur ideal zum Erlernen von Codeblock, sondern ermöglicht darüber hinaus die Arbeit mit effizienteren und stärker parametrischen Dynamo-Diagrammen. Die unten folgende Übungslektion wird mit einem Abschnitt zu Block zu Code abgeschlossen. Sie sollten sie daher vollständig bearbeiten.



## Übungslektion

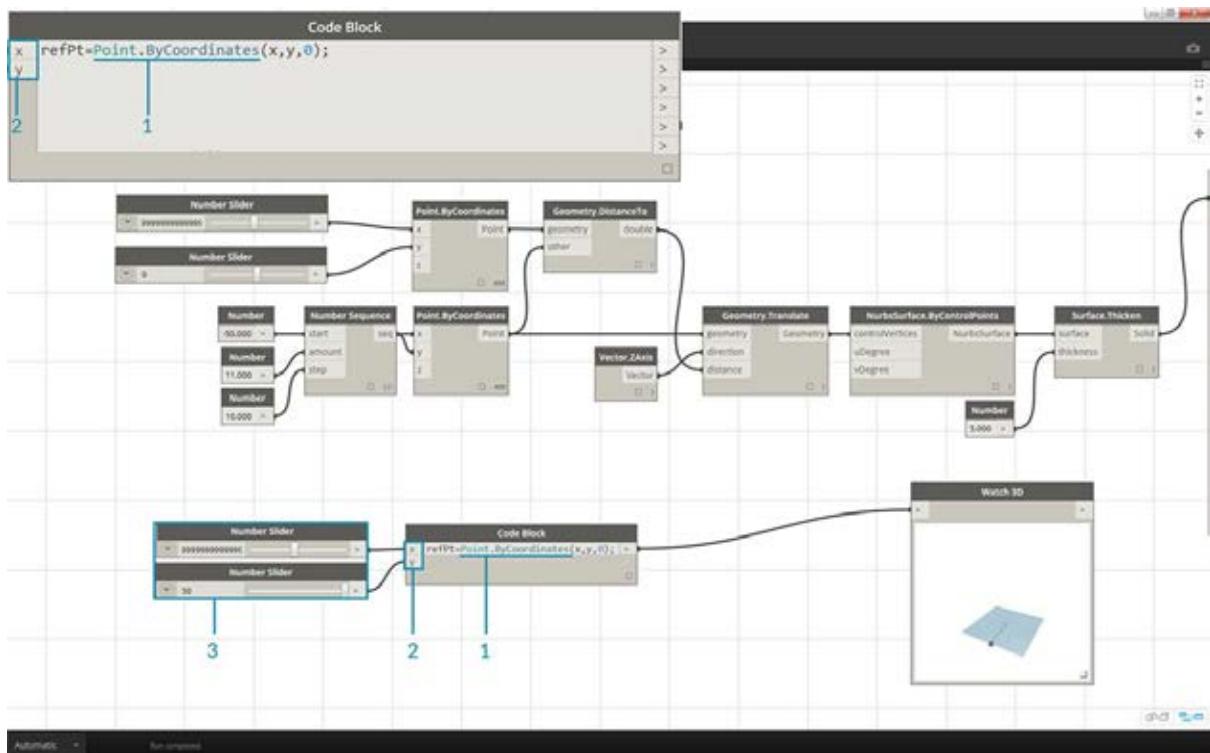
Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Dynamo-Syntax Attractor-Surface.dyn](#)

Zur Demonstration der Effizienz von Codeblock wird hier eine bestehende Definition eines Attraktorfelds in Codeblockform übertragen. Die Verwendung einer bestehenden Definition zeigt die Beziehung zwischen Codeblock und visuellem Skript und erleichtert das Erlernen der Design Script-Syntax.



Erstellen Sie zuerst die in der Abbildung oben gezeigte Definition (oder öffnen Sie die Beispieldatei).

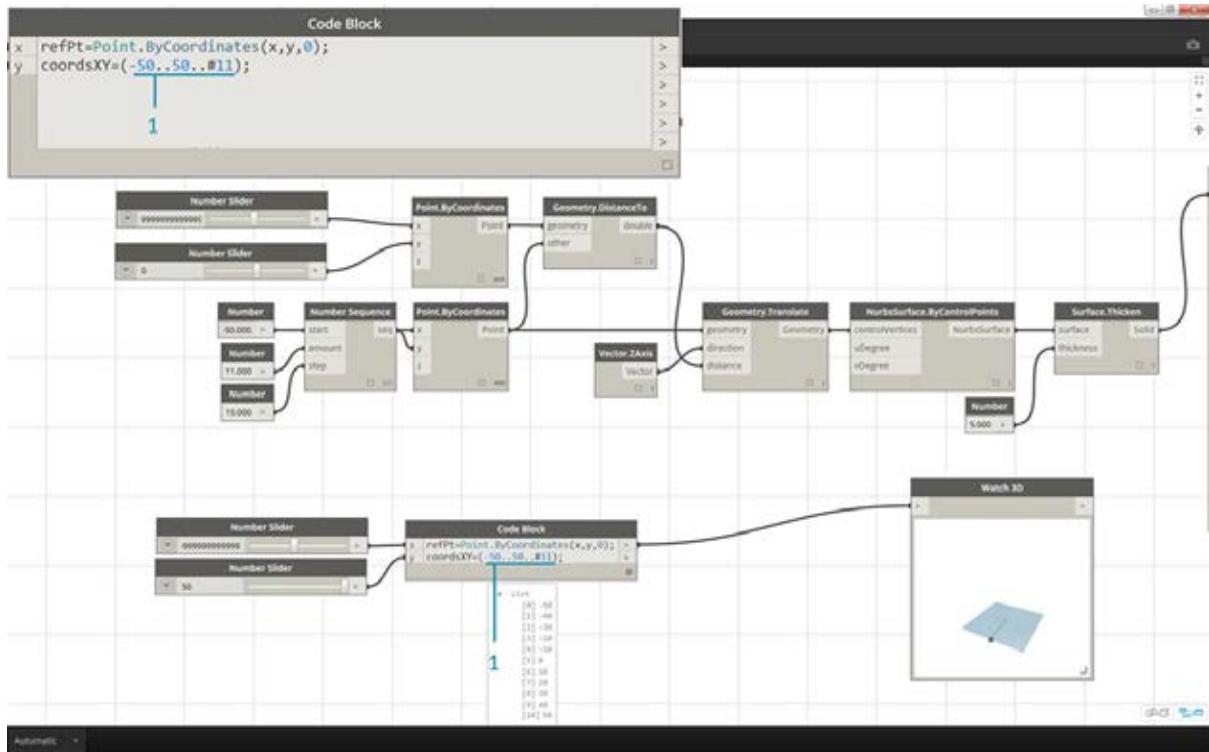
1. Als Vergitterung für *Point.ByCoordinates* wurde Kreuzprodukt eingestellt.
2. Die einzelnen Punkte eines Rasters werden in Abhängigkeit von ihrer Entfernung zum Referenzpunkt in z-Richtung nach oben verschoben.
3. Eine Oberfläche wird erstellt und verdickt, sodass die Geometrie relativ zur Entfernung vom Referenzpunkt ausgebeult wird.



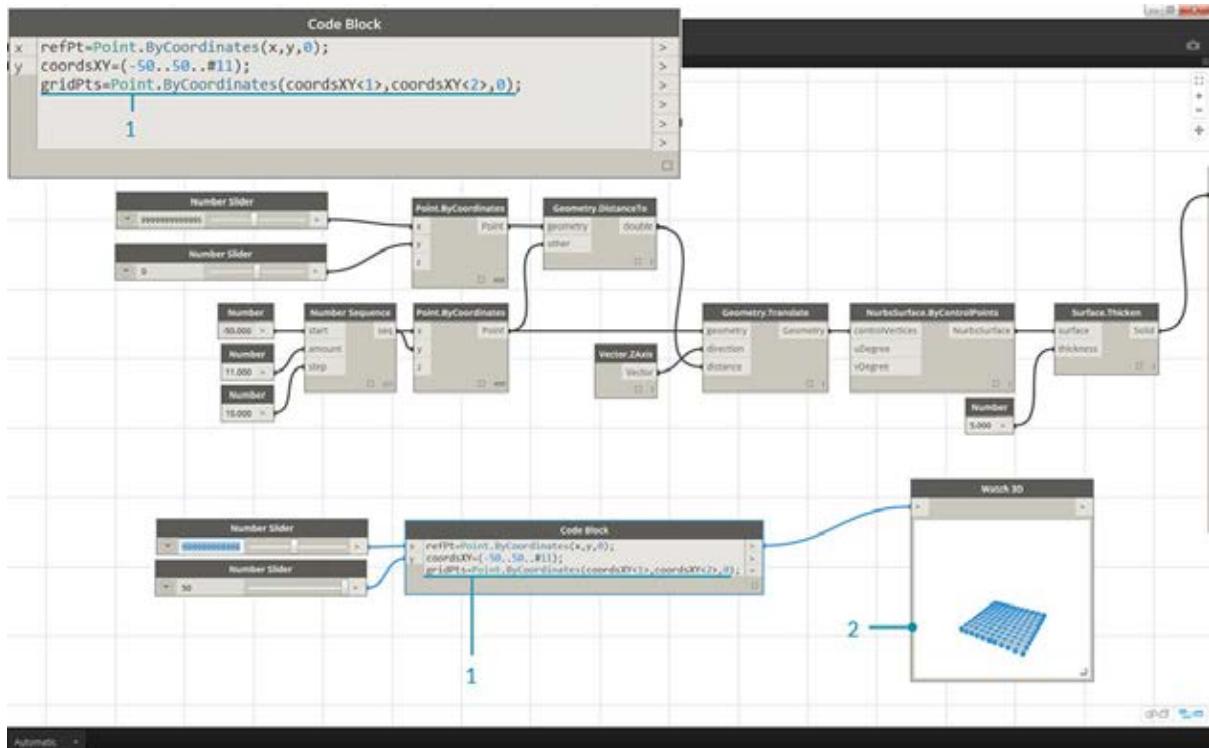
1. Als Erstes wird der Referenzpunkt definiert: `Point .ByCoordinates(x, y, 0)`. Dabei kommt dieselbe Syntax für *Point.ByCoordinates* zum Einsatz, die auch oben im Block für den Referenzpunkt angegeben ist.
2. Die Variablen x und y werden in den Codeblock eingefügt, sodass sie mithilfe von Schiebereglern dynamisch

aktualisiert werden können.

- Fügen Sie Schieberegler für die Eingaben im *Code Block* hinzu, mit denen Bereiche von -50 bis 50 definiert werden. Dies erstreckt sich über das gesamte Vorgaberaster von Dynamo.



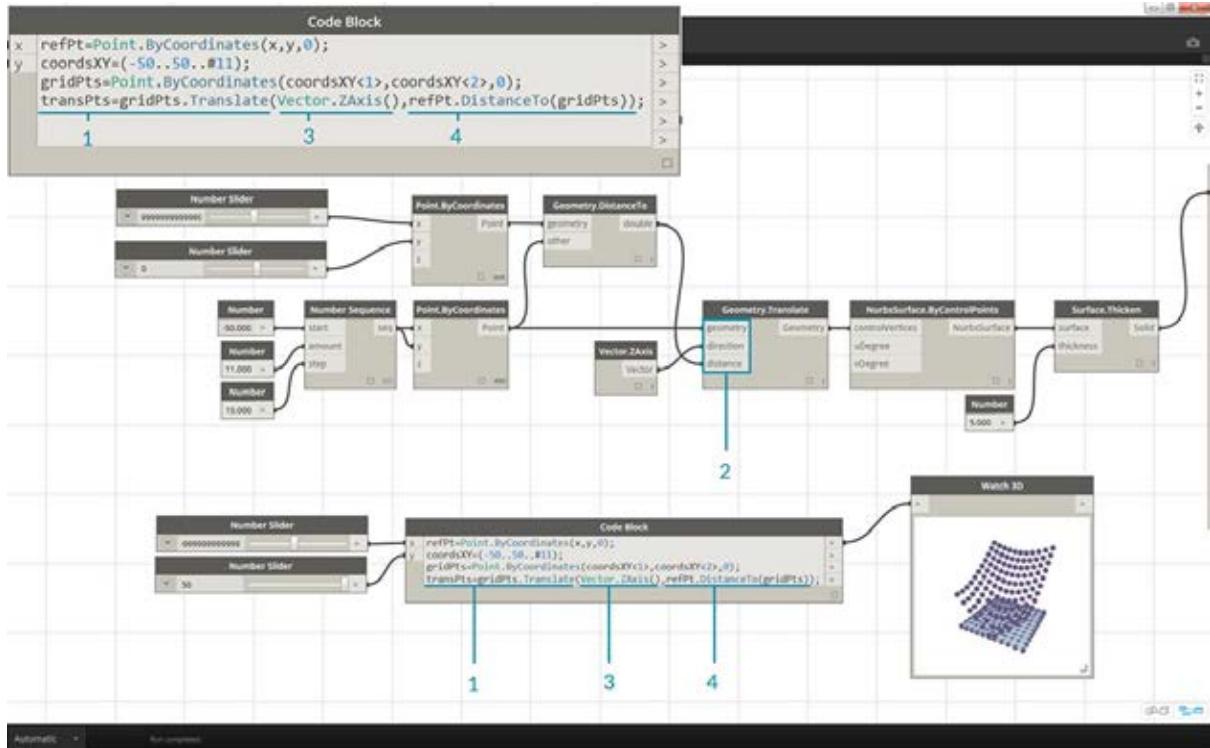
- Definieren Sie in der zweiten Zeile des *Codeblocks* eine Kurzschreibweise für den Zahlenfolgenblock: `coordsXY = (-50..50..#11);`. Dies wird im nächsten Abschnitt genauer behandelt. Halten Sie zunächst fest, dass diese Kurzschreibweise dem *Number Sequence*-Block im visuellen Skript entspricht.



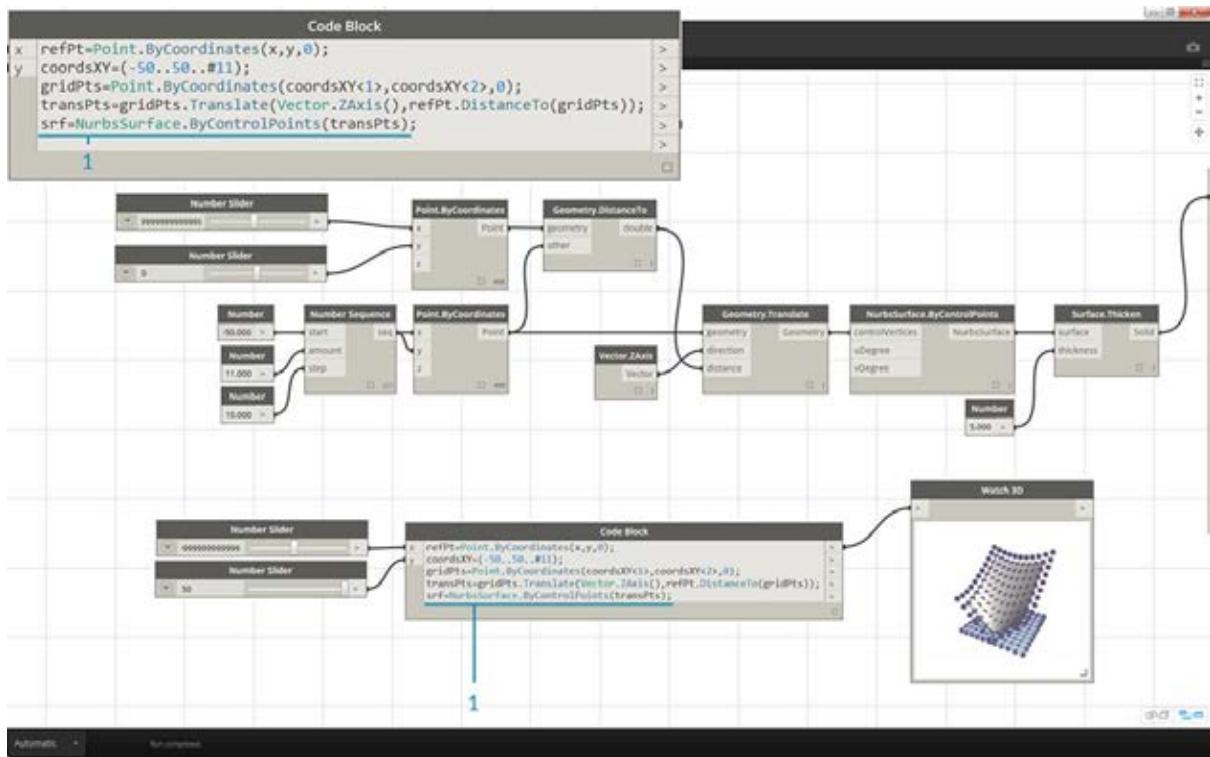
- Als Nächstes erstellen Sie ein Raster aus Punkten aus der *coordsXY*-Folge. Hierfür soll die Syntax *Point.ByCoordinates* verwendet werden, Sie müssen jedoch außerdem genau wie im visuellen Skript das Kreuzprodukt aus der Liste erstellen. Geben Sie zu diesem Zweck die folgende Zeile ein: `gridPts =`

`Point.ByCoordinates(coordsXY<1>, coordsXY<2>, 0);`. Die spitzen Klammern geben die Referenz im Kreuzprodukt an.

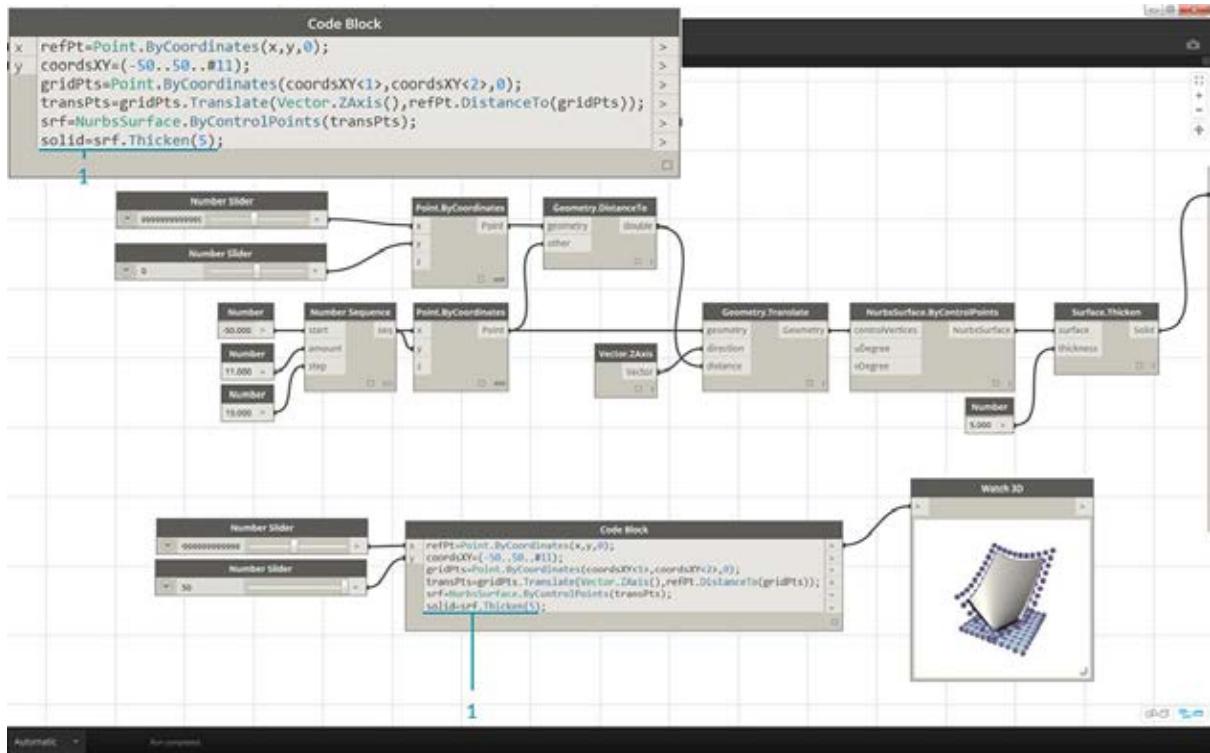
2. Im Watch3D-wird jetzt ein Raster von Punkten über das gesamte Dynamo-Raster angezeigt.



1. Jetzt folgt der schwierige Teil: Die Punkte des Rasters sollen in Abhängigkeit von ihrer Entfernung zum Referenzpunkt nach oben verschoben werden. Diese neue Punktgruppe soll den Namen `transPts` erhalten. Eine Verschiebung ist eine Aktion, die für ein bestehendes Element durchgeführt wird. Verwenden Sie daher nicht `Geometry.Translate...`, sondern `gridPts.Translate`.
2. Im Block, der im Ansichtsbereich gezeigt wird, sind drei Eingaben zu sehen. Die zu verschiebende Geometrie ist bereits deklariert, da die Aktion für dieses Element durchgeführt wird (mithilfe von `gridPts.Translate`). Die beiden verbleibenden Eingaben werden in die Klammern der Funktion eingegeben: `direction` und `distance`.
3. Die Richtung ist leicht anzugeben: Sie geben `Vector.ZAxis()` für die vertikale Verschiebung an.
4. Die Abstände zwischen dem Referenzpunkt und den einzelnen Rasterpunkten müssen jedoch noch berechnet werden. Dies erreichen Sie auf dieselbe Weise wie zuvor durch eine Aktion für den Referenzpunkt:  
`refPt.DistanceTo(gridPts)`
5. Mit der letzten Codezeile erhalten Sie die verschobenen Punkte: `transPts = gridPts.Translate(Vector.ZAxis(), refPt.DistanceTo(gridPts));`



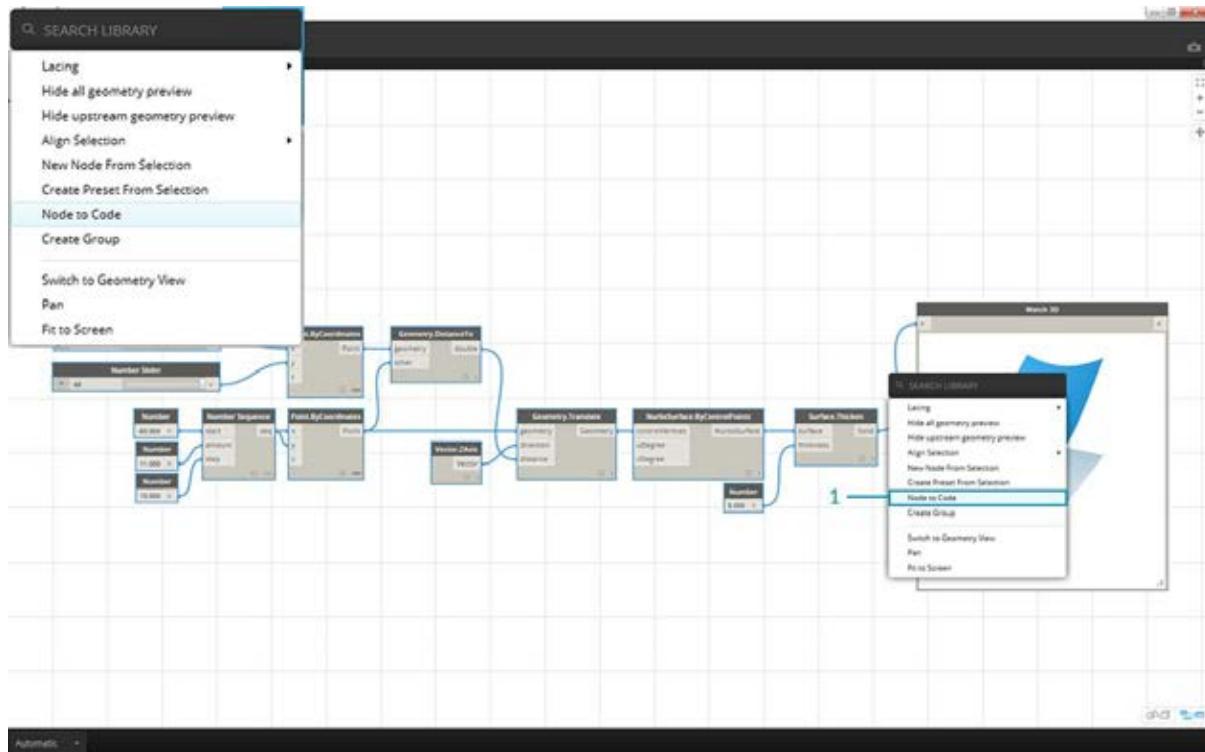
1. Damit haben Sie ein Raster aus Punkten mit einer geeigneten Datenstruktur zum Erstellen einer Nurbs-Oberfläche erstellt. Diese Oberfläche konstruieren Sie mithilfe von `srf = NurbsSurface.ByControlPoints(transPts);`



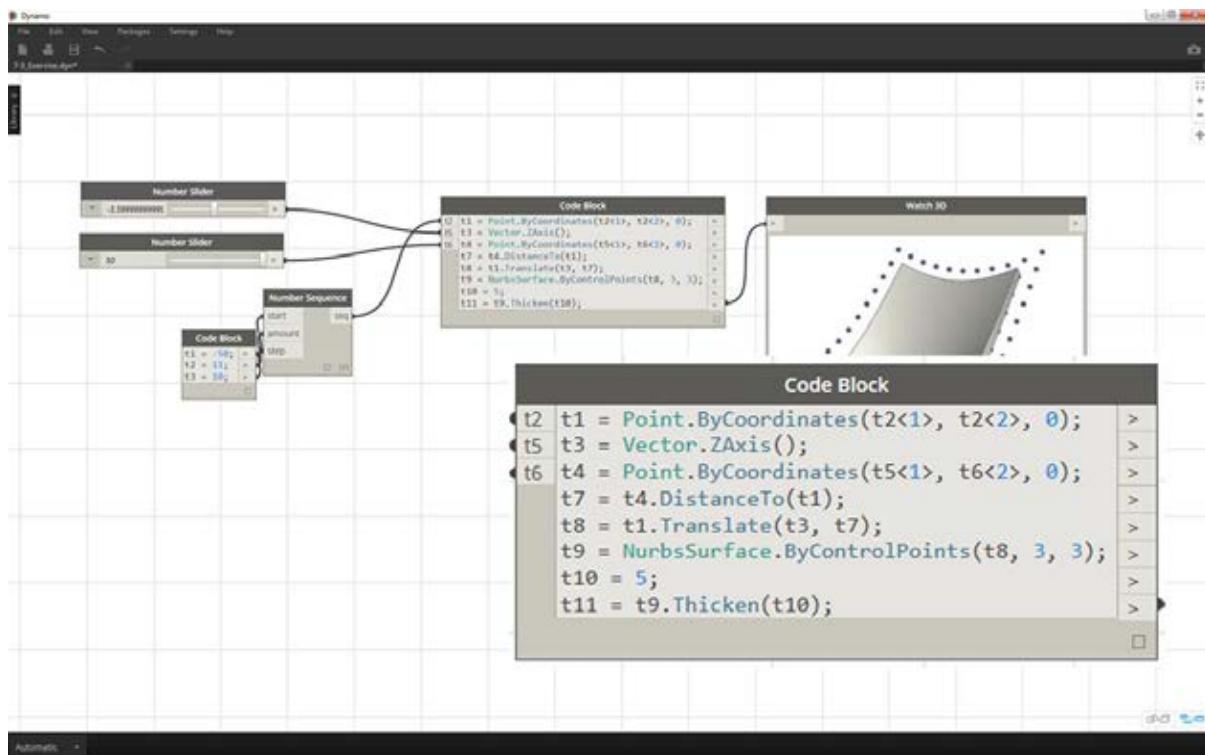
1. Schließlich fügen Sie der Oberfläche Tiefe hinzu: Dazu konstruieren Sie mithilfe von `solid = srf.Thicken(5)`; einen Volumenkörper. In diesem Fall wurde die Oberfläche im Code um 5 Einheiten verdickt, dies kann jedoch jederzeit auch als Variable (etwa mit dem Namen `thickness`) deklariert werden, deren Wert durch einen Schieberegler gesteuert wird.

## Vereinfachen des Diagramms mit Block zu Code

Die Funktion Block zu Code führt die gesamte eben behandelte Übung durch einen einfachen Mausklick aus. Sie ermöglicht damit nicht nur die Erstellung benutzerdefinierter Definitionen und wiederverwendbarer Codeblöcke, sondern ist auch ein äußerst nützliches Hilfsmittel beim Erlernen der Skripterstellung in Dynamo:



1. Beginnen Sie mit dem bestehenden visuellen Skript aus Schritt 1 dieser Übungslektion. Wählen Sie sämtliche Blöcke aus, klicken Sie mit der rechten Maustaste in den Ansichtsbereich und wählen Sie *Block zu Code*. Dieser einfache Schritt genügt.



Dynamo hat automatisch eine Textversion des visuellen Diagramms einschließlich Vergitterung und aller anderen Angaben erstellt. Probieren Sie dies mit Ihren visuellen Skripts aus und schöpfen Sie die Möglichkeiten von Codeblöcken voll aus!

# Kurzschreibweisen

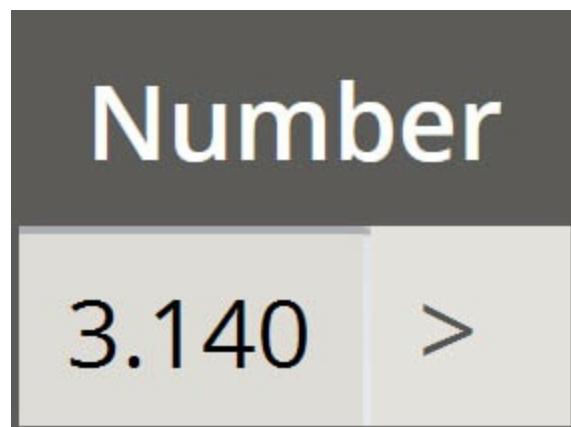
## Kurzschreibweisen

Für Codeblöcke stehen einige einfache Kurzschreibweisen zur Verfügung, die, einfach ausgedrückt, die Arbeit mit den Daten *erheblich* erleichtern. Im Folgenden werden die Grundlagen genauer erläutert und beschrieben, wie die jeweilige Kurzschreibweise zum Erstellen und Abfragen von Daten verwendet werden kann.

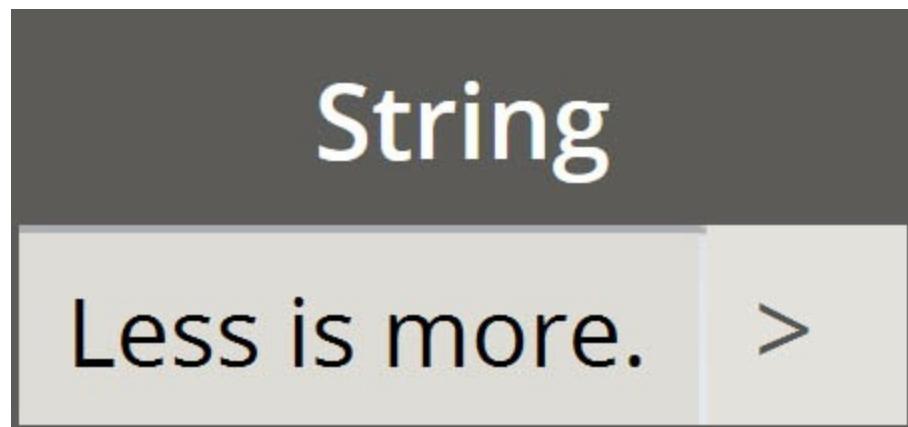
Datentyp

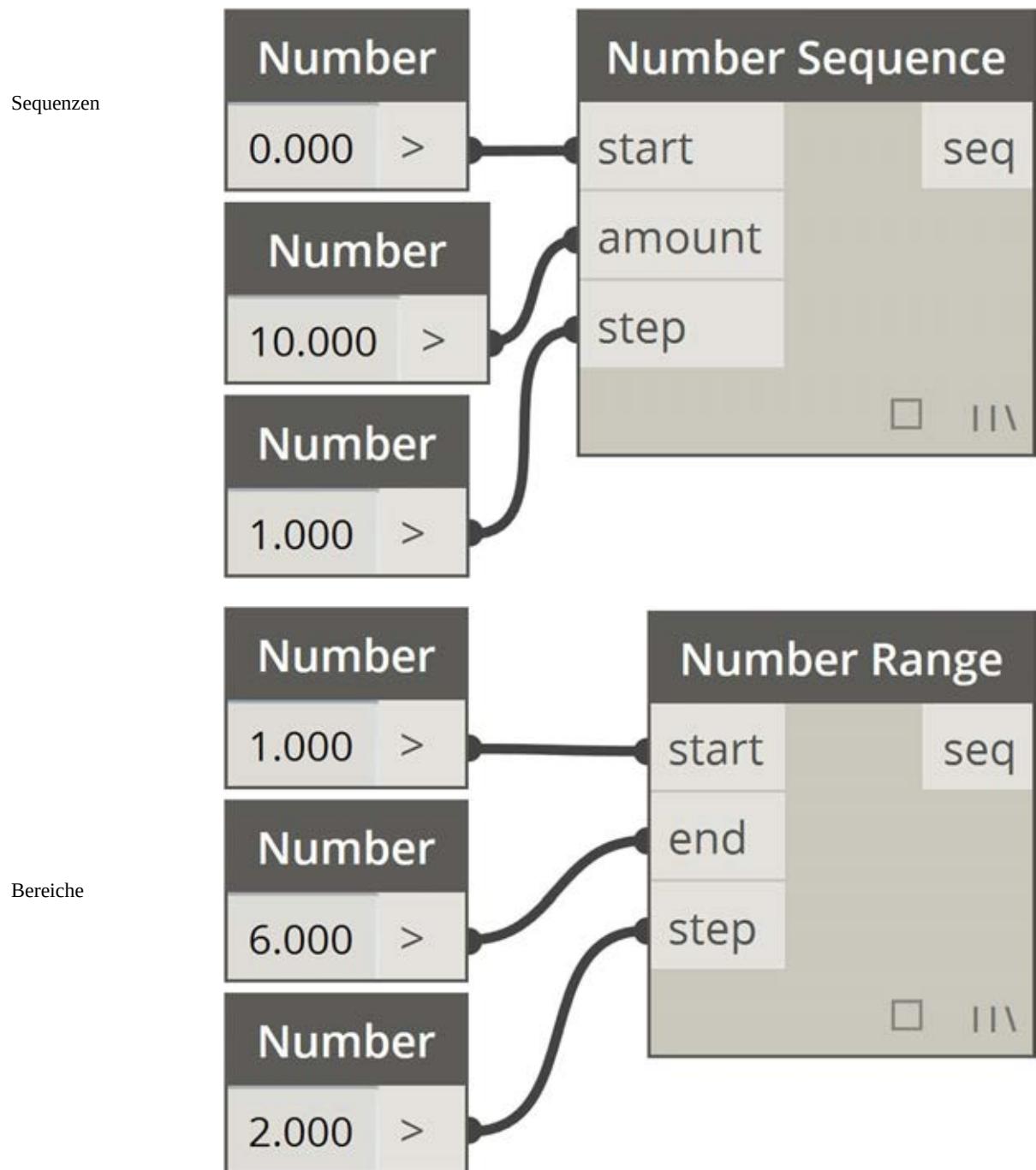
Dynamo-Standarddarstellung

Zahlen

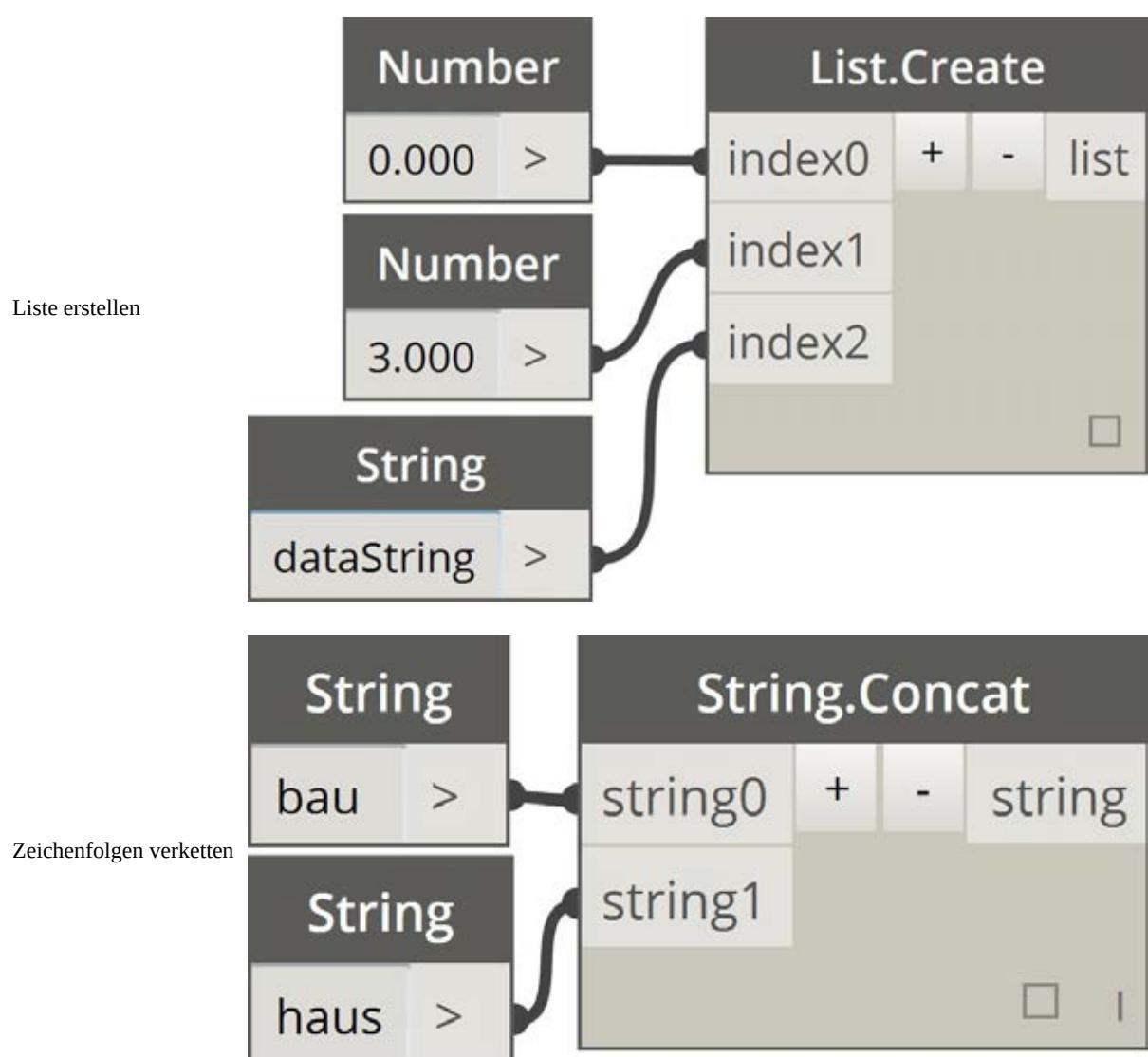


Zeichenfolgen

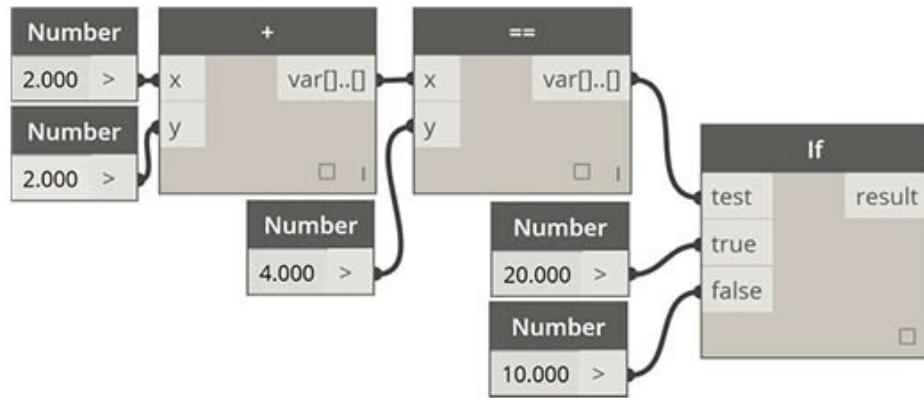




Eintrag an Indexposition  
abrufen



Bedingungsanweisungen



## Zusätzliche Syntax

Block/Blöcke	Codeblock-Entsprechung	Anmerkung
Beliebiger Operator (+, &&, >=, !, usw.)	+, &&, >=, !, usw.	Beachten Sie, dass "Not" (Nicht) durch "!" ersetzt wird, der Block jedoch zur Unterscheidung von "Fakultät" nach wie vor "Not" heißt.
Boolescher Wert True	true;	Anmerkung: Kleinbuchstaben
Boolescher Wert False	false;	Anmerkung: Kleinbuchstaben

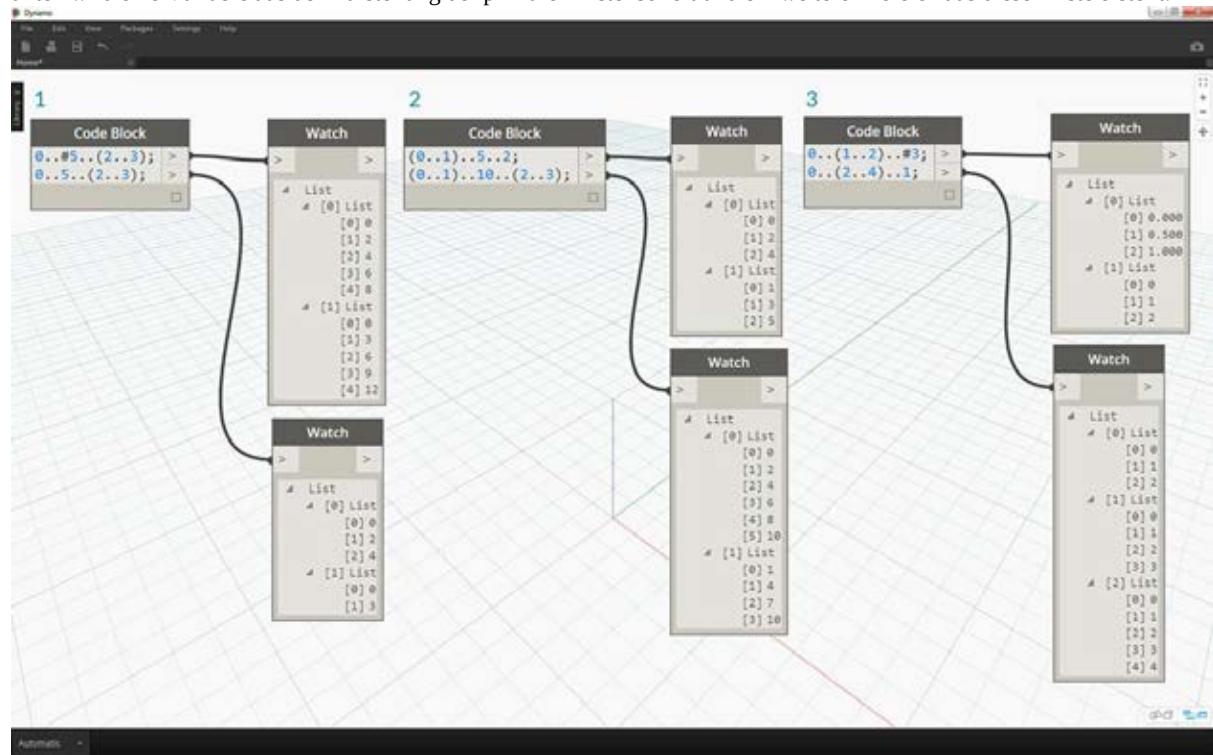
## Bereiche

Die Methoden zum Definieren von Bereichen und Sequenzen können in einfachen Kurzschreibweisen ausgedrückt werden. Die folgende Abbildung bietet eine Anleitung zum Definieren einer Liste mit numerischen Daten mithilfe der Syntax ".." in Codeblöcken. Nachdem Sie sich mit dieser Notation vertraut gemacht haben, können Sie numerische Daten äußerst effizient erstellen:

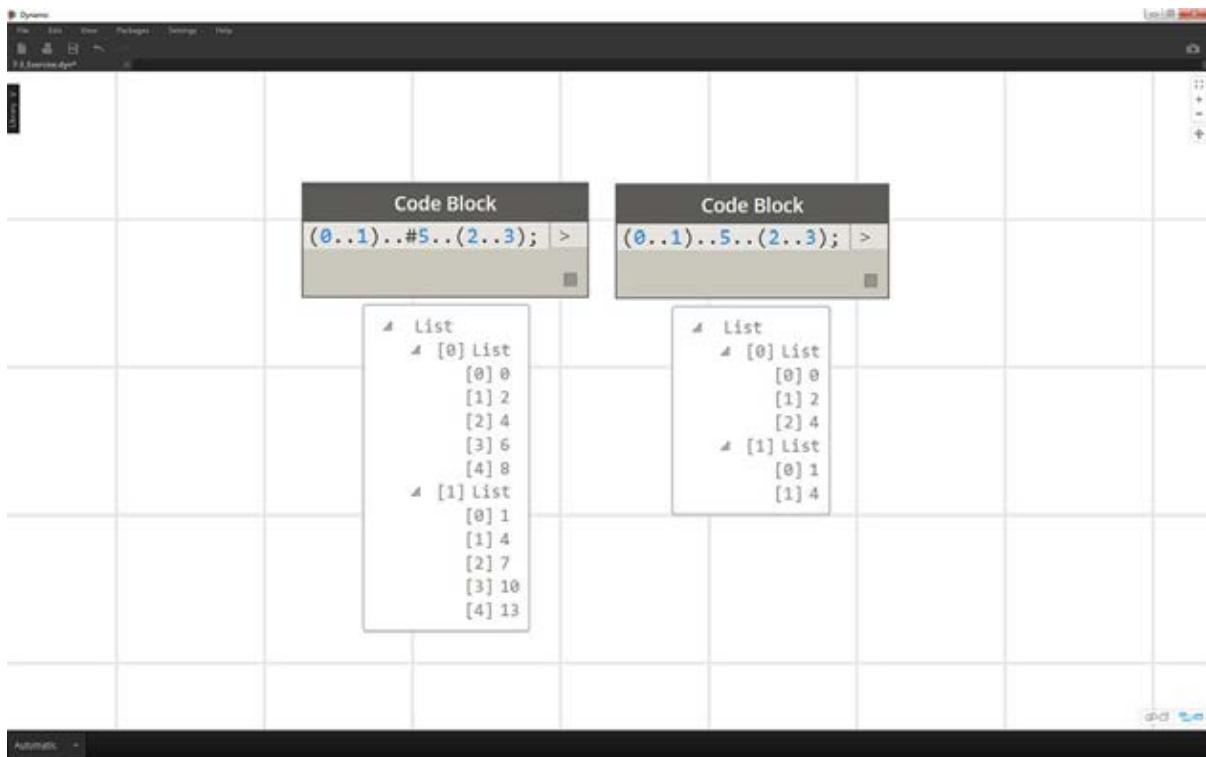
1. In diesem Beispiel wird ein Zahlenbereich durch einfache Codeblock-Syntax mit Angaben für **Start..Ende..Schrittgröße**; ersetzt. In numerischer Darstellung erhalten Sie die Werte **0..10..1**;
2. Beachten Sie, dass die Syntax **0..10..1**; der Syntax **0..10**; entspricht. Der Wert 1 wird in der Kurzschreibweise für die Schrittgröße vorgegeben. Mit **0..10**; erhalten Sie daher eine Folge von 0 bis 10 mit der Schrittgröße 1.
3. Das Beispiel für die **Zahlenfolge** ist ähnlich, allerdings wird hier mithilfe eines # -Zeichens angegeben, dass die Liste nicht beim Wert 15 enden, sondern 15 Werte enthalten soll. In diesem Fall definieren Sie: **Anfang..Anzahl Schritte..Schrittgröße:**. Die Syntax für die Folge lautet **0..#15..2**.
4. Platzieren Sie das #-Zeichen aus dem vorigen Schritt jetzt im Bereich für die **Schrittgröße** der Syntax. Damit haben Sie einen **Zahlenbereich** vom *Anfang* zum *Ende* erstellt. Die Notation für die **Schrittgröße** verteilt die angegebene Anzahl Werte gleichmäßig zwischen diesen Angaben: **Anfang..Ende..Anzahl Schritte**.

## Erweiterte Bereiche

Indem Sie erweiterte Bereiche erstellen, können Sie auf einfache Weise mit Listen von Listen arbeiten. In den Beispielen unten wird eine Variable aus der Darstellung der primären Liste isoliert und ein weiterer Bereich aus dieser Liste erstellt.



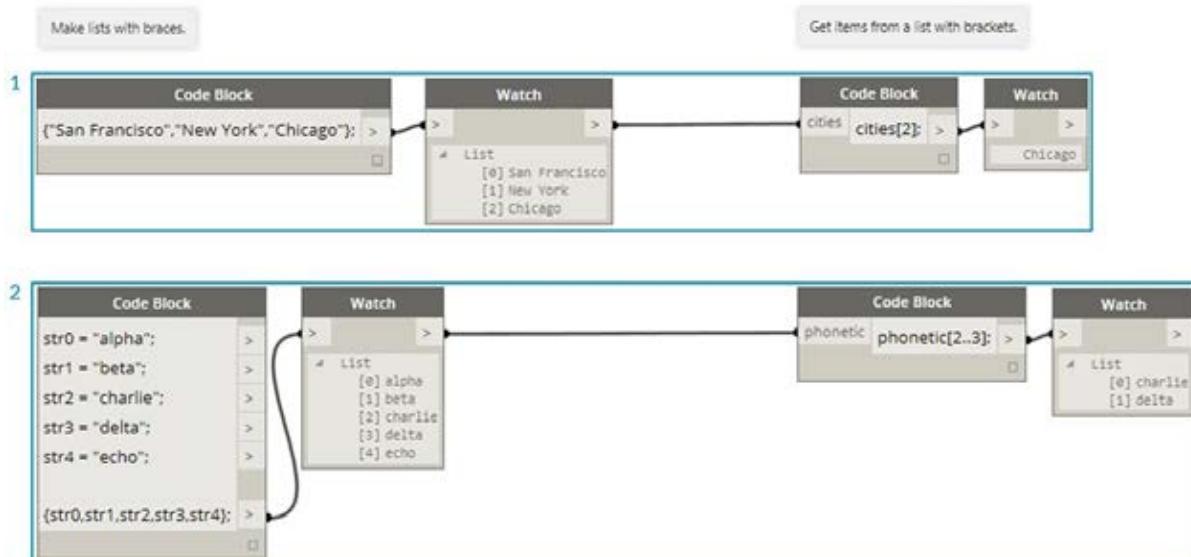
1. Vergleichen Sie die Notation mit und ohne #-Zeichen bei der Erstellung verschachtelter Bereiche. Dabei gilt dieselbe Logik wie bei einfachen Bereichen, die Angaben sind jedoch etwas komplexer.
2. Sie können an beliebiger Stelle des primären Bereichs einen Unterbereich erstellen. Es ist auch möglich, zwei Unterbereiche zu verwenden.
3. Mithilfe des Werts für *end* in einem Bereich erstellen Sie weitere Bereiche unterschiedlicher Länge.



Vergleichen Sie als Übung zu dieser Logik die beiden oben gezeigten Kurzschreibweisen und testen Sie, wie *Unterbereiche* und das # -Zeichen sich auf das Ergebnis auswirken.

### Erstellen von Listen und Abrufen von Einträgen aus Listen

Sie können Listen nicht nur mithilfe von Kurzschreibweisen, sondern auch ad hoc erstellen. Solche Listen können eine Vielfalt von Elementtypen enthalten und können abgefragt werden (da Listen ihrerseits Objekte sind). Kurz zusammengefasst: In einem Codeblock verwenden Sie zum Erstellen von Listen geschweifte und zum Abfragen der Listen eckige Klammern.



1. Sie können Listen schnell aus Zeichenfolgen erstellen und über die Indizes der Einträge abfragen.
2. Sie können Listen mit Variablen erstellen und sie über die Kurzschreibweisen für Bereiche abfragen.

Verschachtelte Listen werden auf ähnliche Weise verwaltet. Beachten Sie dabei die Reihenfolge der Listen und verwenden Sie mehrere Paare eckiger Klammern:

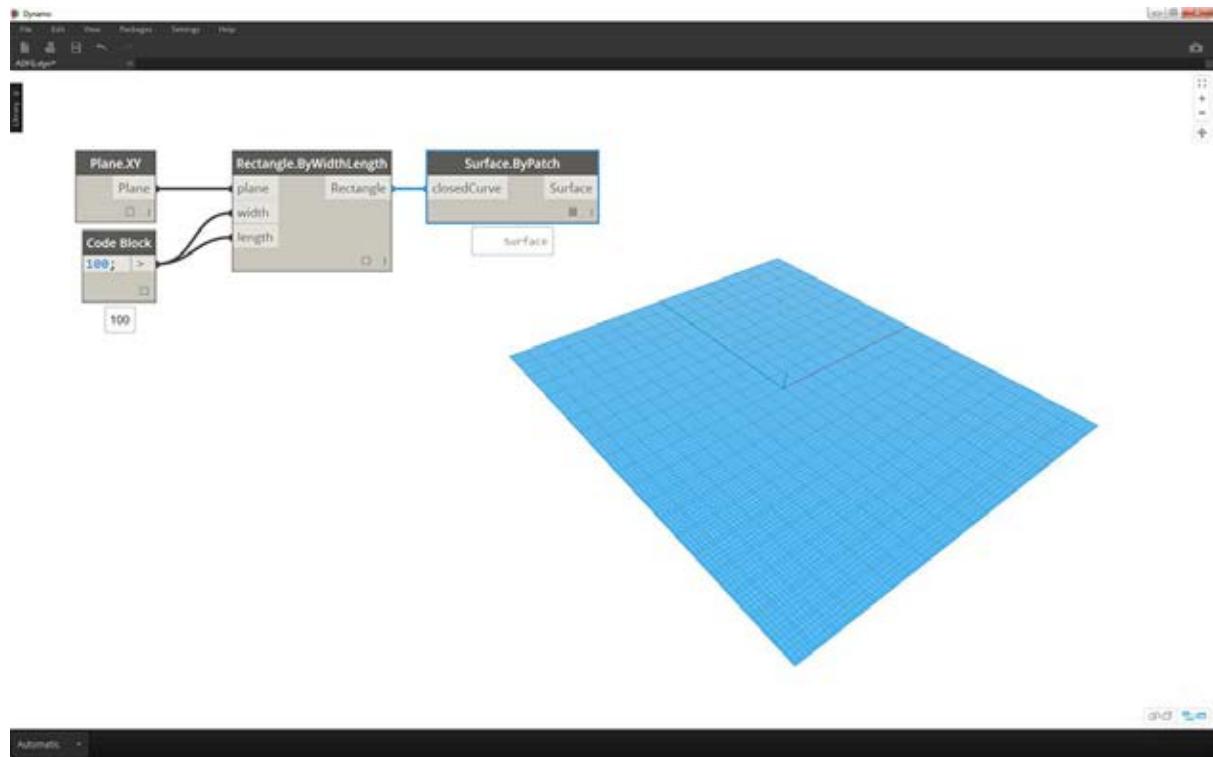


1. Definieren Sie eine Liste von Listen.
2. Fragen Sie eine Liste mit einer Angabe in eckigen Klammern ab.
3. Fragen Sie einen Eintrag mit zwei Angaben in eckigen Klammern ab.

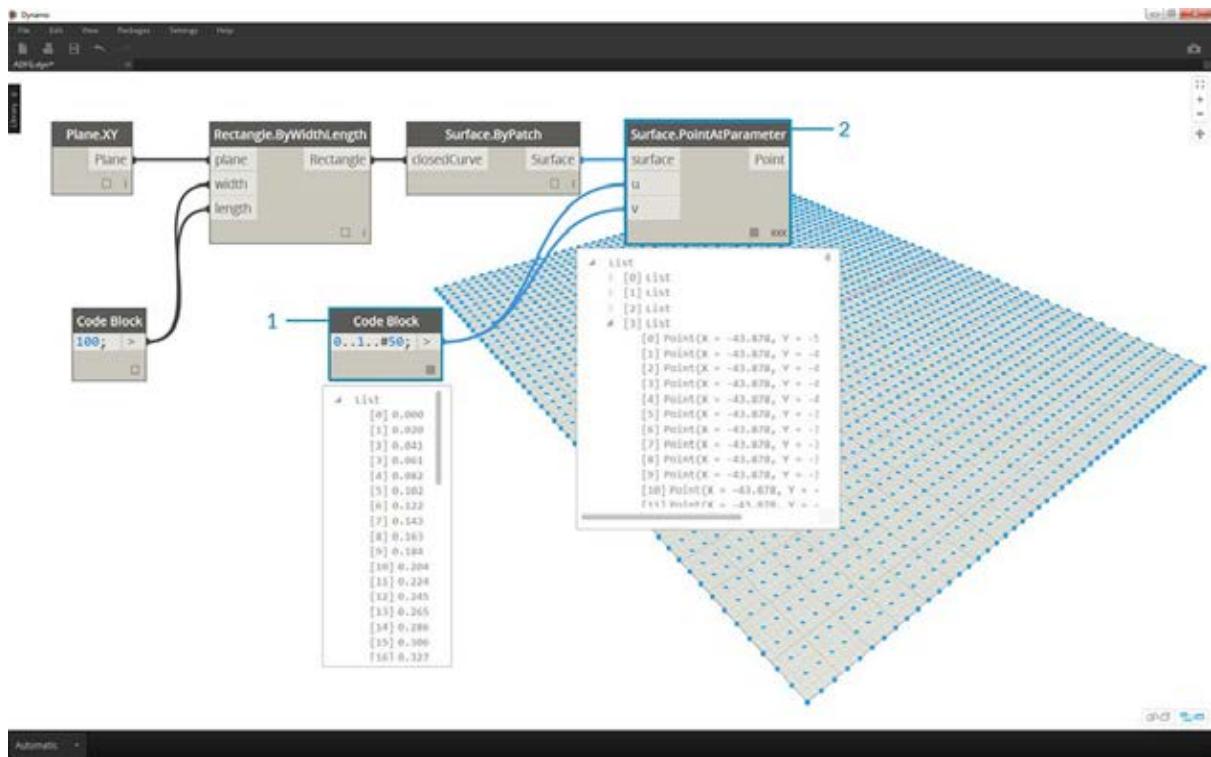
## Übungslektion

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Obsolete-Nodes\\_Sine-Surface.dyn](#)

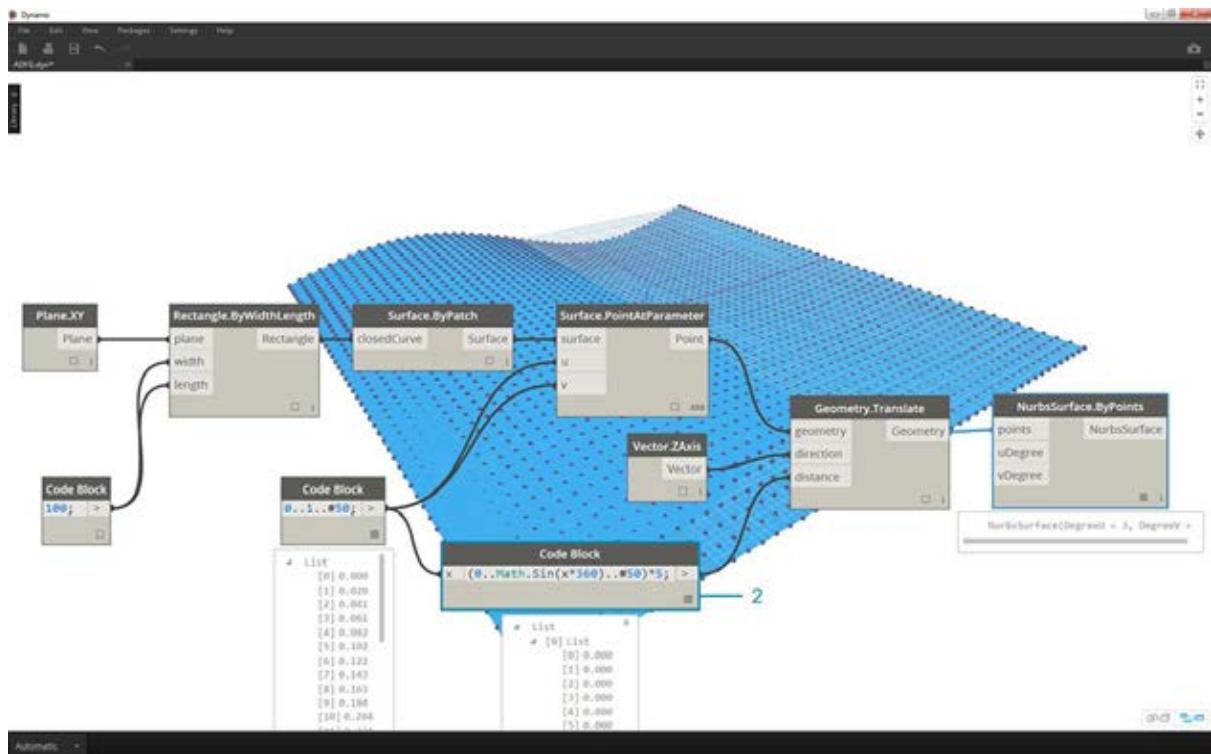
In dieser Übung wenden Sie Ihre Kenntnisse der Kurzschreibweise an und erstellen eine originelle gewölbte Oberfläche, die Sie mithilfe von Bereichen und Formeln definieren. Beachten Sie in dieser Übung, wie Codeblöcke und bestehende Dynamo-Blöcke zusammenwirken: Für umfangreiche Datenverarbeitungen kommen Codeblöcke zum Einsatz, durch die visuelle Darstellung der Dynamo-Blöcke ist die Definition leichter zu lesen.



Erstellen Sie zunächst eine Oberfläche, indem Sie die oben gezeigten Blöcke verbinden. Verwenden Sie zum Definieren der Breite und Länge keinen Number-Block, sondern doppelklicken Sie in den Ansichtsbereich und geben Sie den Wert **100;** in einen Codeblock ein.



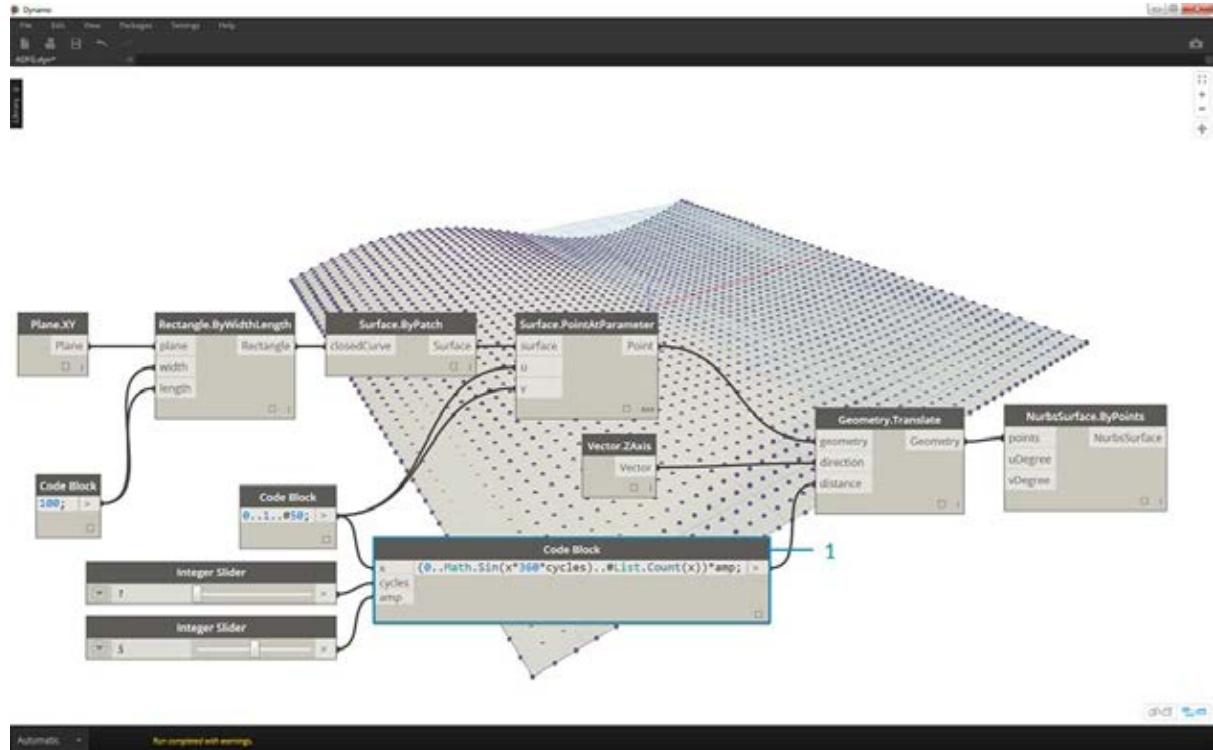
1. Definieren Sie einen Bereich zwischen 0 und 1 mit 50 Unterteilungen, indem Sie  $0..1..#50$  in einen Codeblock eingegeben.
2. Verbinden Sie den Bereich mit *Surface.PointAtParameter*. Dieser Block benötigt *u*- und *v*-Werte zwischen 0 und 1 für die gesamte Oberfläche. Sie müssen die *Vergitterung* in *Kreuzprodukt* ändern. Klicken Sie dazu mit der rechten Maustaste auf den *Surface.PointAtParameter*-Block.



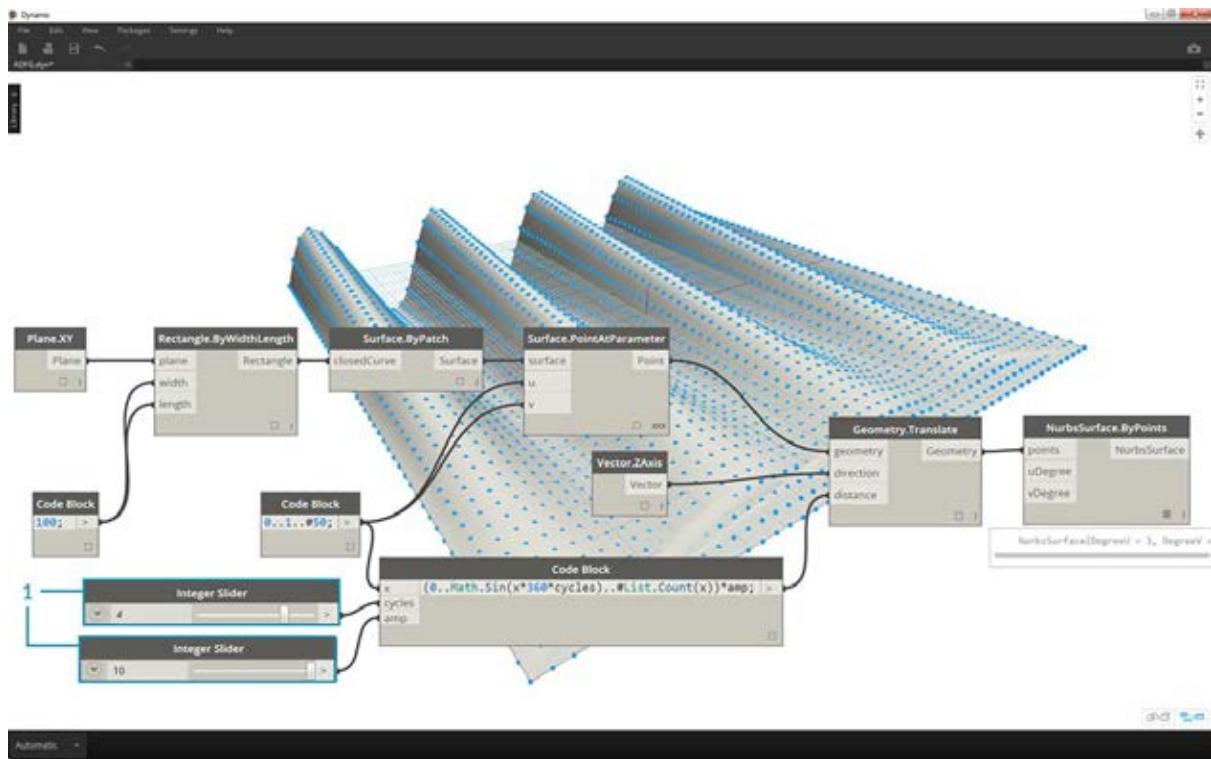
In diesem Schritt verschieben Sie das Raster aus Punkten mithilfe der ersten Funktion in z-Richtung nach oben. Dieses Raster steuert die zu erstellende Oberfläche anhand der zugrunde liegenden Funktion.

1. Fügen Sie die visuellen Blöcke wie in der Abbildung oben gezeigt in den Ansichtsbereich ein.
2. Verwenden Sie anstelle eines Formelblocks einen Codeblock mit der folgenden Zeile:

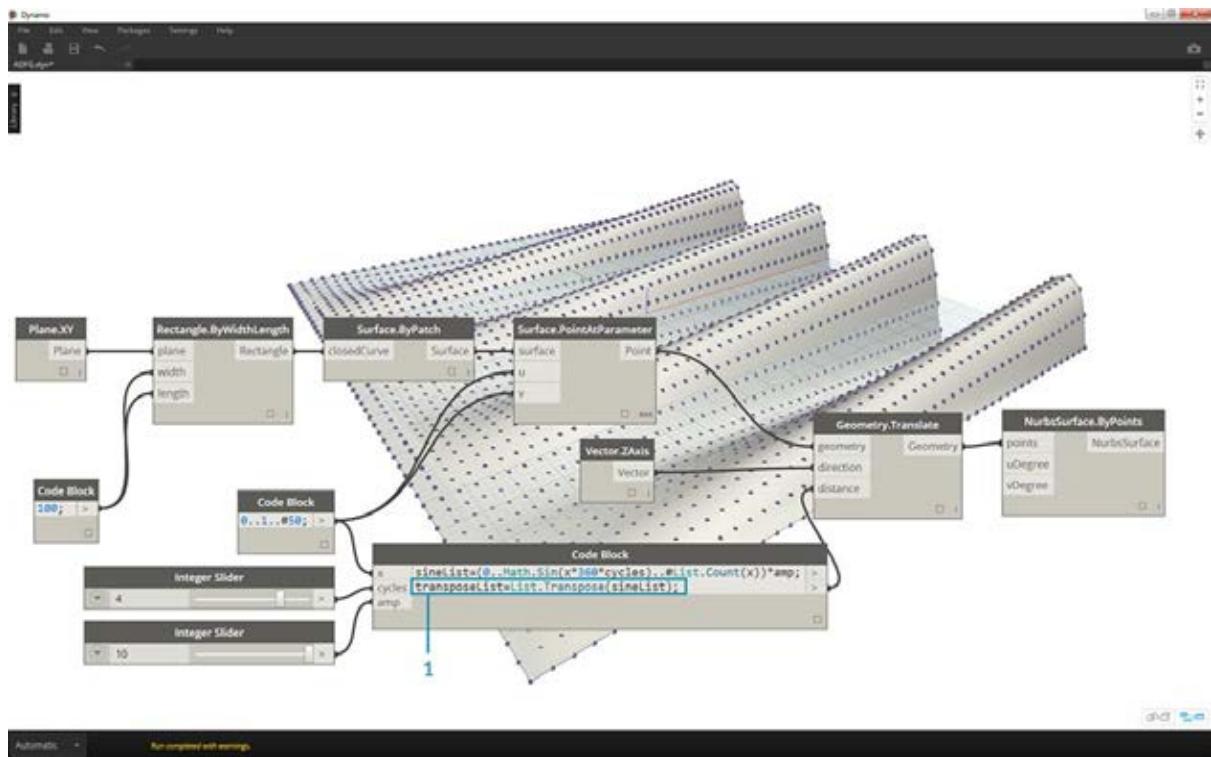
`(0..Math.Sin(x*360)..#50)*5;`. Dadurch definieren Sie, kurz zusammengefasst, einen Bereich, der eine Formel enthält. Diese Formel ist die Sinusfunktion. Da für die Sinusfunktion in Dynamo Werte in Grad eingegeben werden müssen, multiplizieren Sie die x-Werte (d. h. den eingegebenen Bereich zwischen 0 und 1) mit 360, um eine vollständige Sinuskurve zu erhalten. Als Nächstes wird dieselbe Anzahl Unterteilungen als Rastersteuerpunkte für die einzelnen Reihen benötigt. Definieren Sie daher 50 Unterteilungen mit #50. Der Multiplikator 5 schließlich verstärkt die Verschiebung, sodass deren Wirkung in der Dynamo-Vorschau deutlich zu sehen ist.



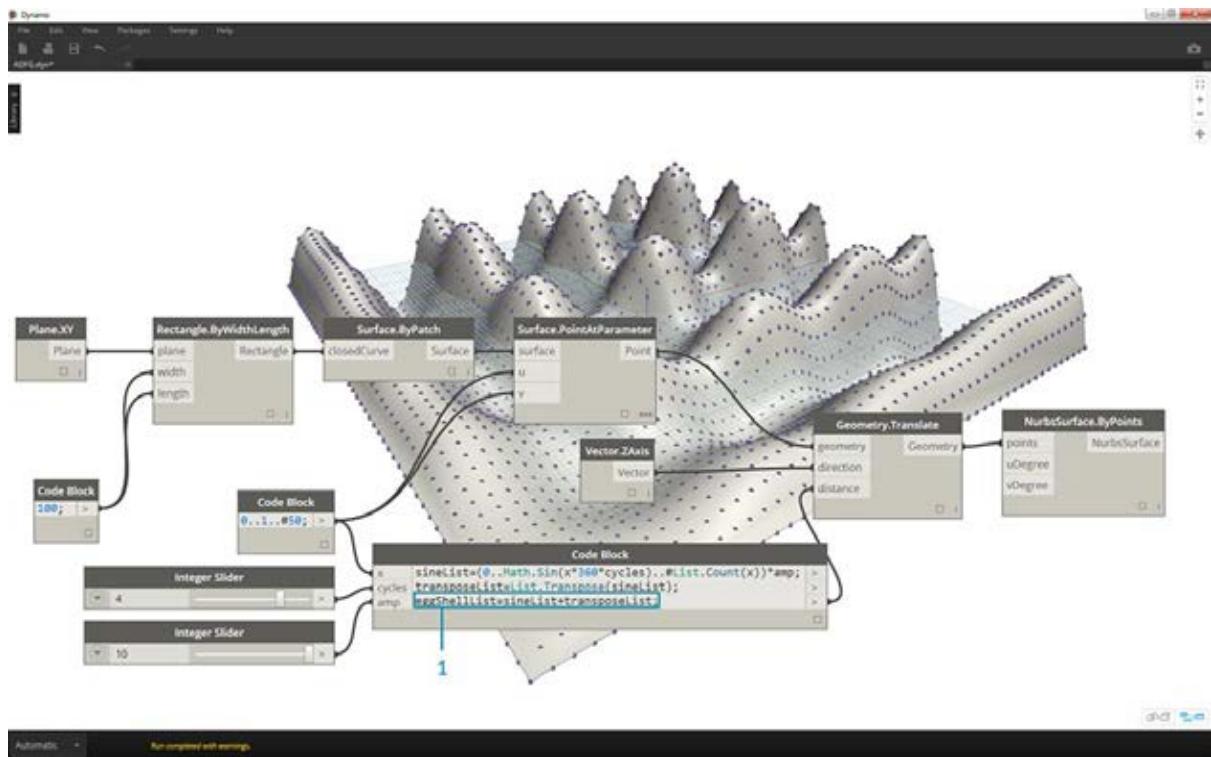
- Der bis hierher verwendete Codeblock erfüllte seine Funktion, war jedoch nicht vollständig parametrisch. Seine Parameter sollen dynamisch gesteuert werden. Ersetzen Sie daher die Zeile aus dem vorigen Schritt durch `(0..Math.Sin(x*360*cycles)..#List.Count(x))*amp;`. Dadurch erhalten Sie die Möglichkeit, diese Werte anhand Ihrer Eingaben zu definieren.



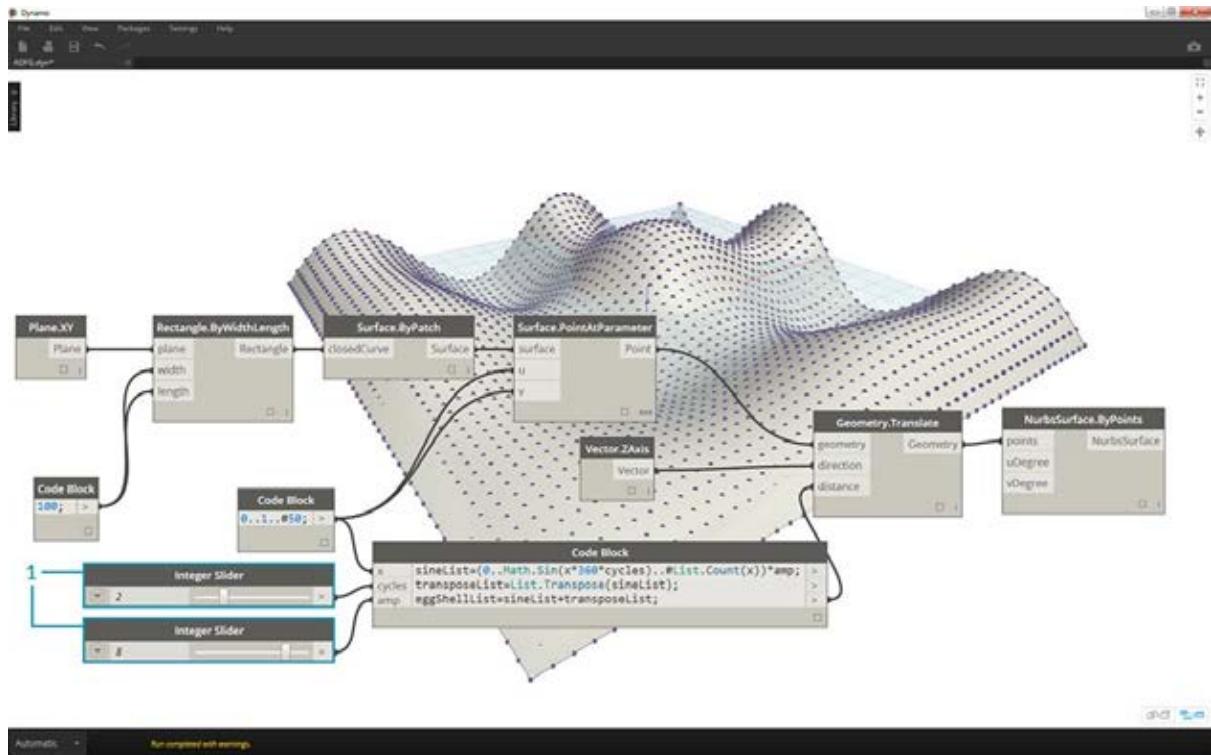
1. Indem Sie die Werte der Schieberegler (zwischen 0 und 10) ändern, erhalten Sie interessante Ergebnisse.



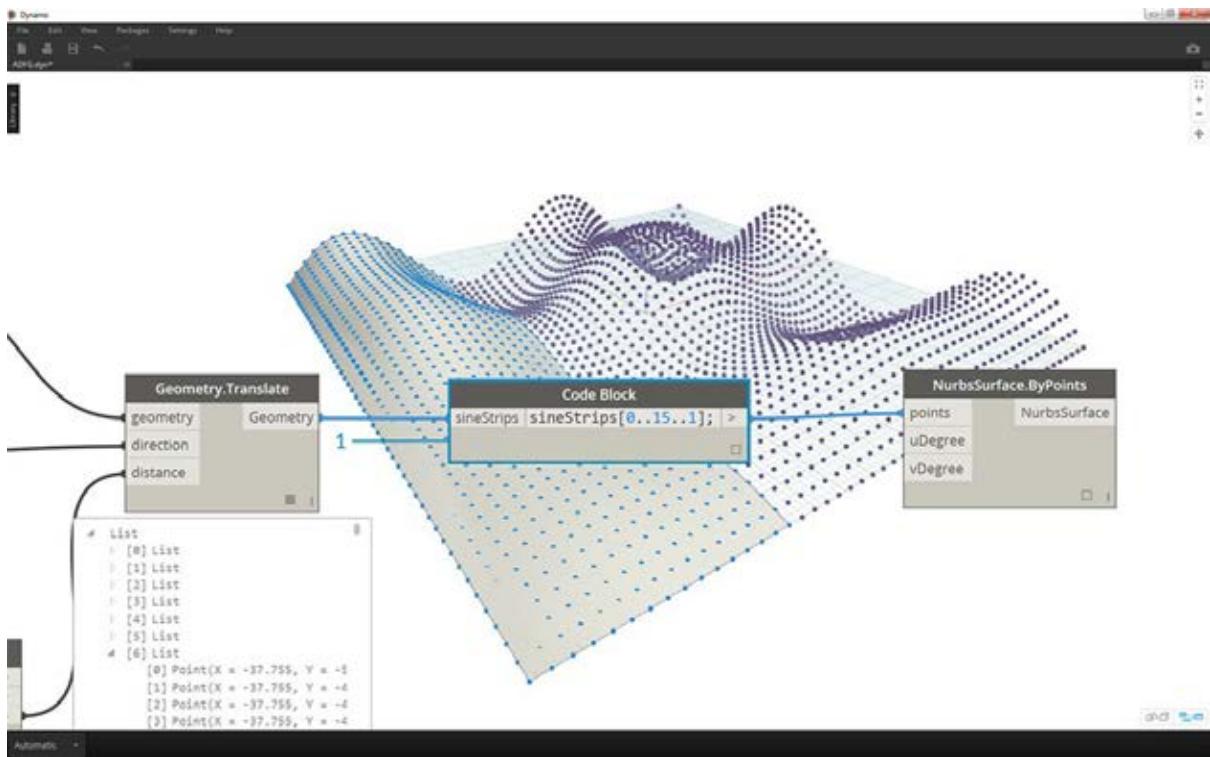
1. Indem Sie Transpose auf den Zahlenbereich anwenden, kehren Sie die Richtung der Wellen um:  
`transposeList = List.Transpose(sineList);`



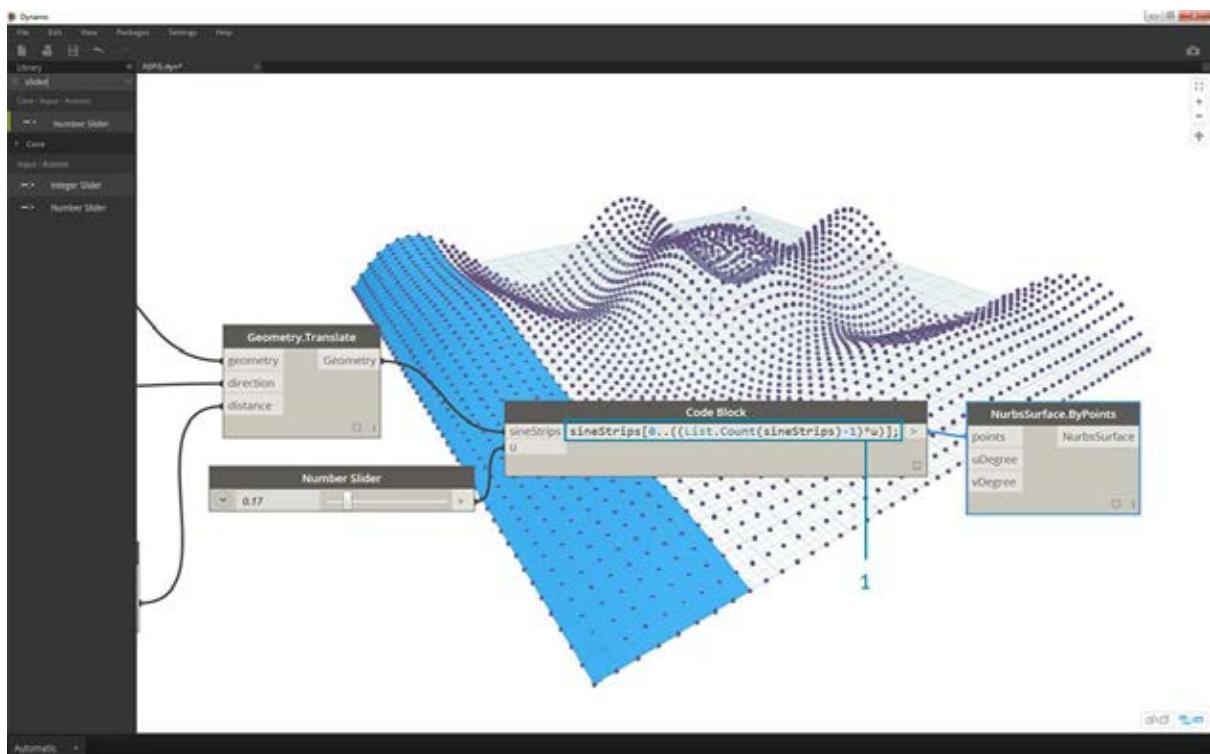
1. Durch Addieren von sineList und transposeList erhalten Sie eine verzerrte Hülle: `eggShellList = sineList+transposeList;`



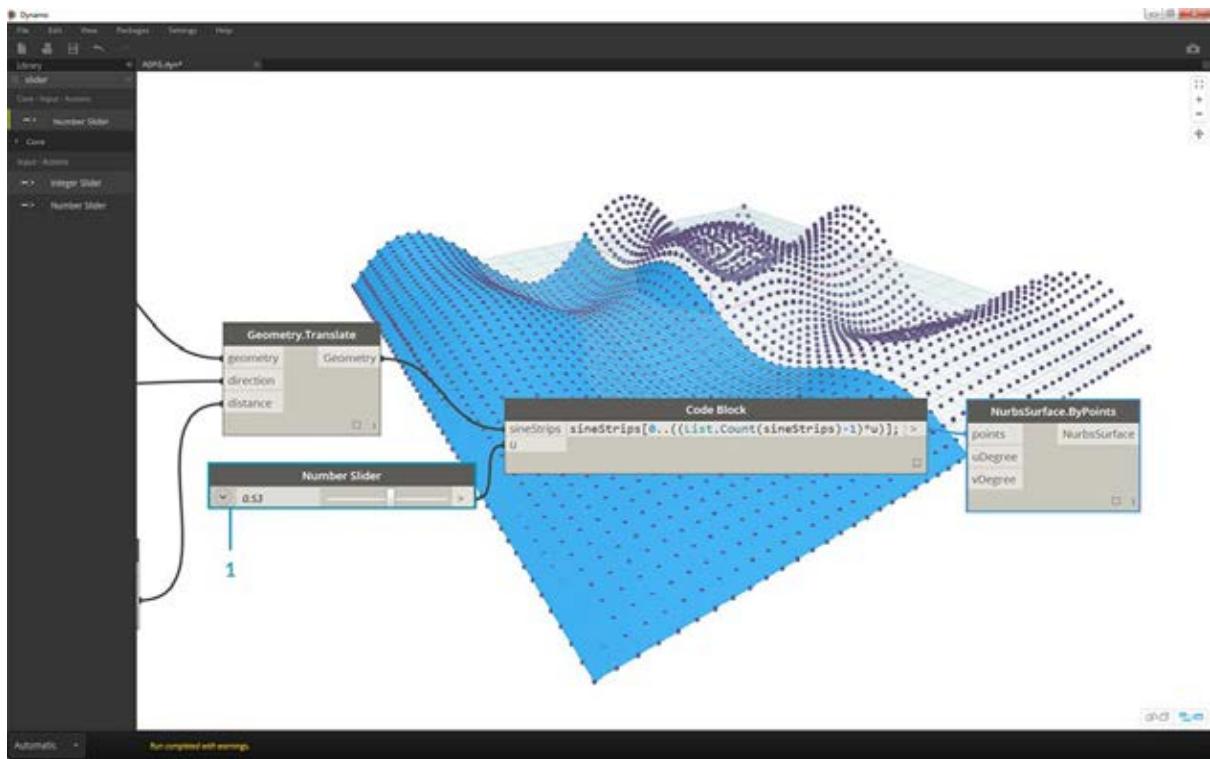
1. Ändern Sie die Werte in den Schiebereglern erneut, um ein ruhigeres Muster zu erhalten.



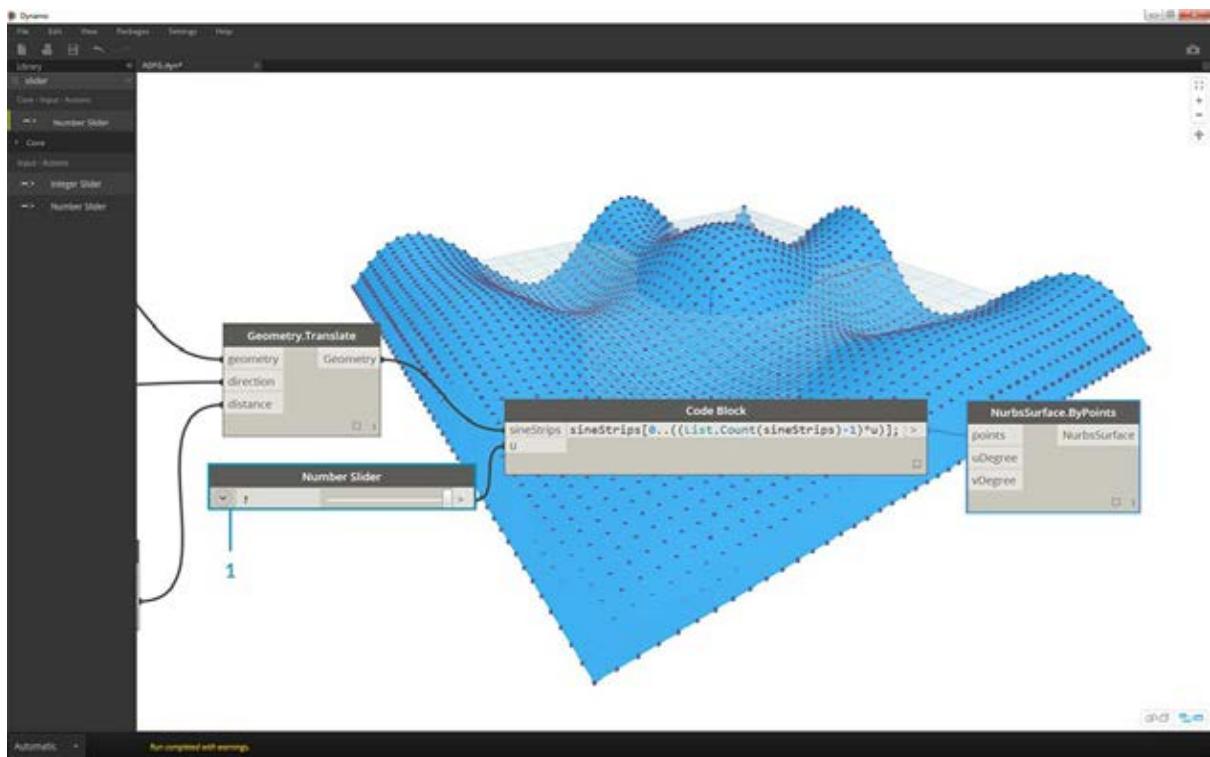
1. In dieser letzten Übung fragen Sie isolierte Teile der Daten mithilfe eines Codeblocks ab. Fügen Sie den oben gezeigten Codeblock zwischen dem *Geometry.Translate*- und dem *NurbsSurface.ByPoints*-Block ein, um die Oberfläche aus einem bestimmten Bereich von Punkten neu zu erstellen. Der Codeblock enthält die folgende Textzeile: `sineStrips[0..15..1];`. Dadurch werden die ersten 16 (von 50) Punktreihen ausgewählt. Wenn die Oberfläche neu erstellt wird, ist zu erkennen, dass ein isolierter Teil des Punktrasters generiert wurde.



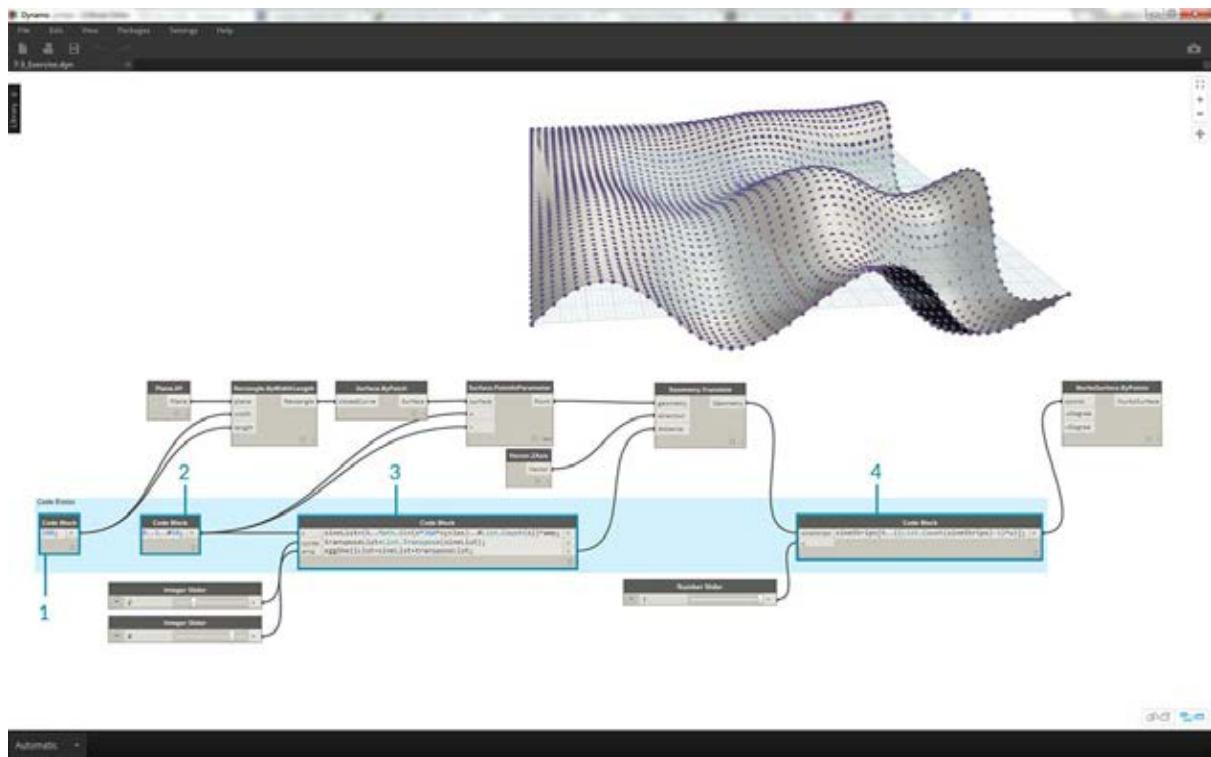
1. Im letzten Schritt erweitern Sie den Codeblock um parametrische Funktionen, indem Sie einen Schieberegler mit dem Bereich von 0 bis 1 zur Steuerung der Abfrage verwenden. Verwenden Sie hierfür die folgende Codezeile: `sineStrips[0..((List.Count(sineStrips)-1)*u)];`. Dies mag etwas verwirrend wirken. Diese Codezeile ermöglicht jedoch eine schnelle Skalierung der Länge der Liste über einen Multiplikator zwischen 0 und 1.



1. Mithilfe des Werts .53 im Schieberegler erstellen Sie eine Oberfläche, die sich knapp über die Mitte des Rasters hinaus erstreckt.



1. Mit dem Wert 1 im Schieberegler wird erwartungsgemäß eine Oberfläche aus dem gesamten Punktraster erstellt.



Im resultierenden visuellen Diagramm können Sie die einzelnen Codeblöcke markieren und ihre Funktionen sehen.

1. Der erste Codeblock ersetzt den *Number*-Block.
2. Der zweite Codeblock ersetzt den *Number Range*-Block.
3. Der dritte Codeblock ersetzt den *Formula*-Block (sowie *List.Transpose*, *List.Count* und *Number Range*).
4. Der vierte Codeblock ruft eine Liste von Listen ab und ersetzt den *List.GetItemAtIndex*-Block.

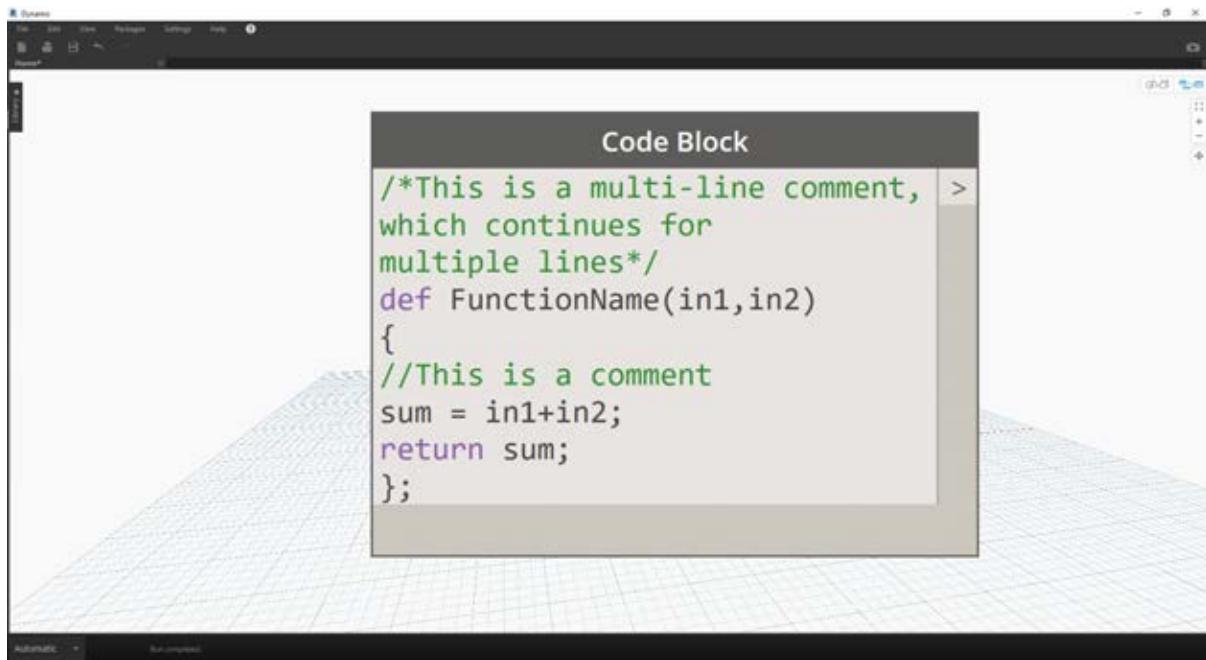
# **Codeblock-Funktionen**

## **Codeblock-Funktionen**

Funktionen können in einem Codeblock erstellt und an anderer Stelle in einer Dynamo-Definition erneut aufgerufen werden. Dadurch erhalten Sie eine weitere Steuerungsmöglichkeit in parametrischen Dateien. Sie stellt eine Textversion eines benutzerdefinierten Blocks dar. In diesem Fall ist der übergeordnete Block problemlos verfügbar und kann sich an beliebiger Stelle im Diagramm befinden. Hierfür sind keine Verbindungen erforderlich.

### **Übergeordneter Block**

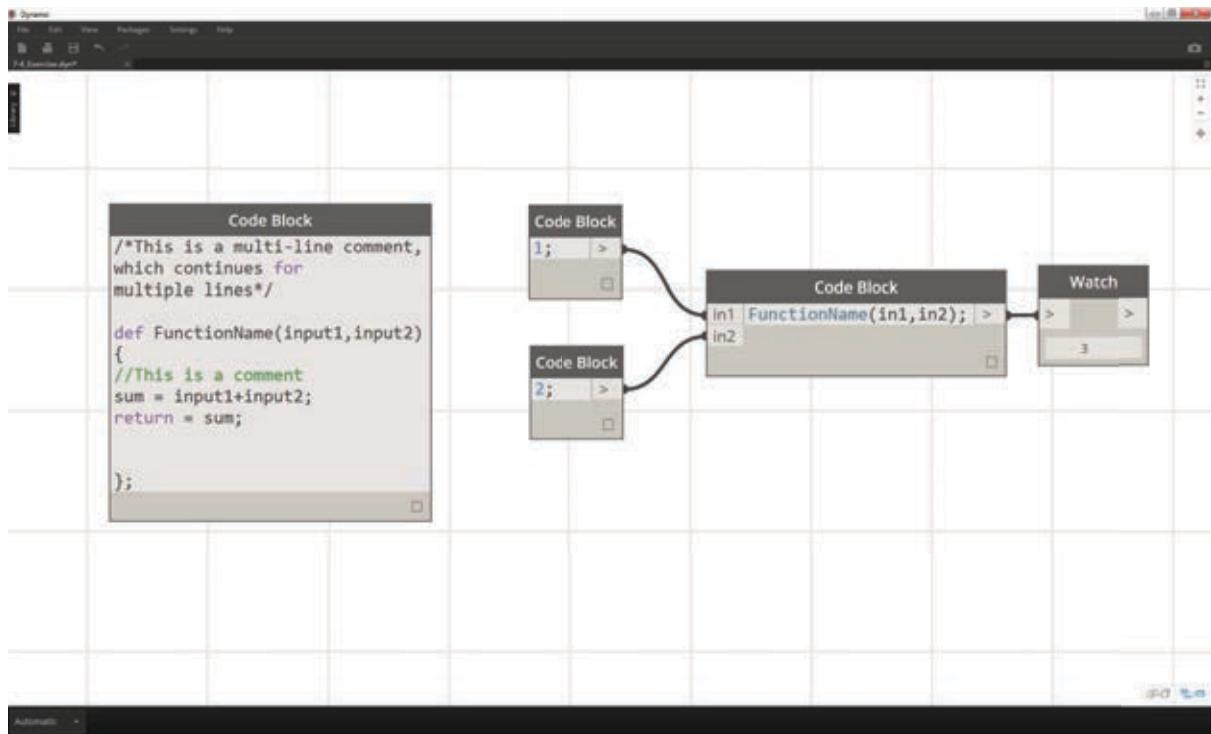
Die erste Zeile besteht aus dem Schlüsselwort "def" und dem Namen der Funktion mit den Namen der Eingaben in Klammern. Der Hauptteil der Funktion wird in geschweiften Klammern angegeben. Verwenden Sie zum Ausgeben eines Werts "return =". In Codeblöcken, die eine Funktion definieren, sind keine Eingabe- oder Ausgabeverbindungen vorhanden, da sie über andere Codeblöcke aufgerufen werden.



```
/*This is a multi-line comment,
which continues for
multiple lines*/
def FunctionName(in1,in2)
{
//This is a comment
sum = in1+in2;
return sum;
};
```

### Untergeordnete Blöcke

Sie können die Funktion in einem anderen Codeblock in derselben Datei aufrufen, indem Sie ihren Namen und dieselbe Anzahl von Argumenten angeben. Dies funktioniert auf dieselbe Weise wie die vordefinierten Blöcke aus der Bibliothek.

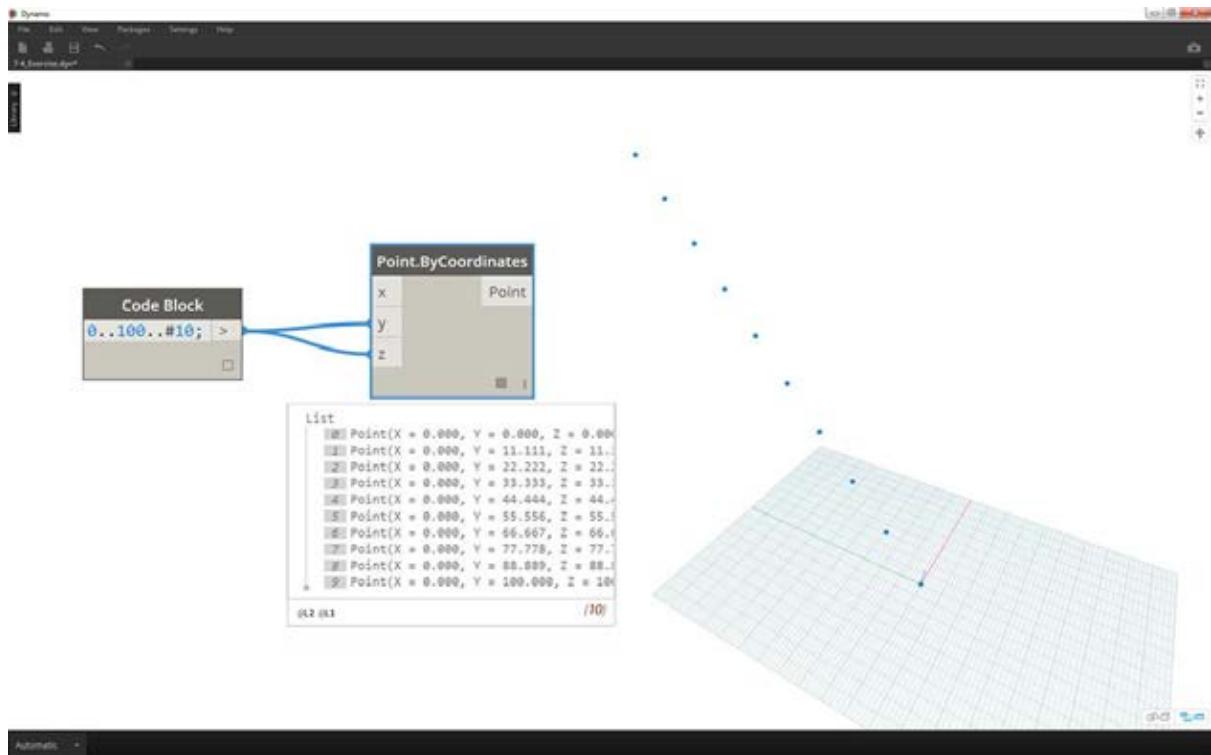


FunctionName(in1,in2);

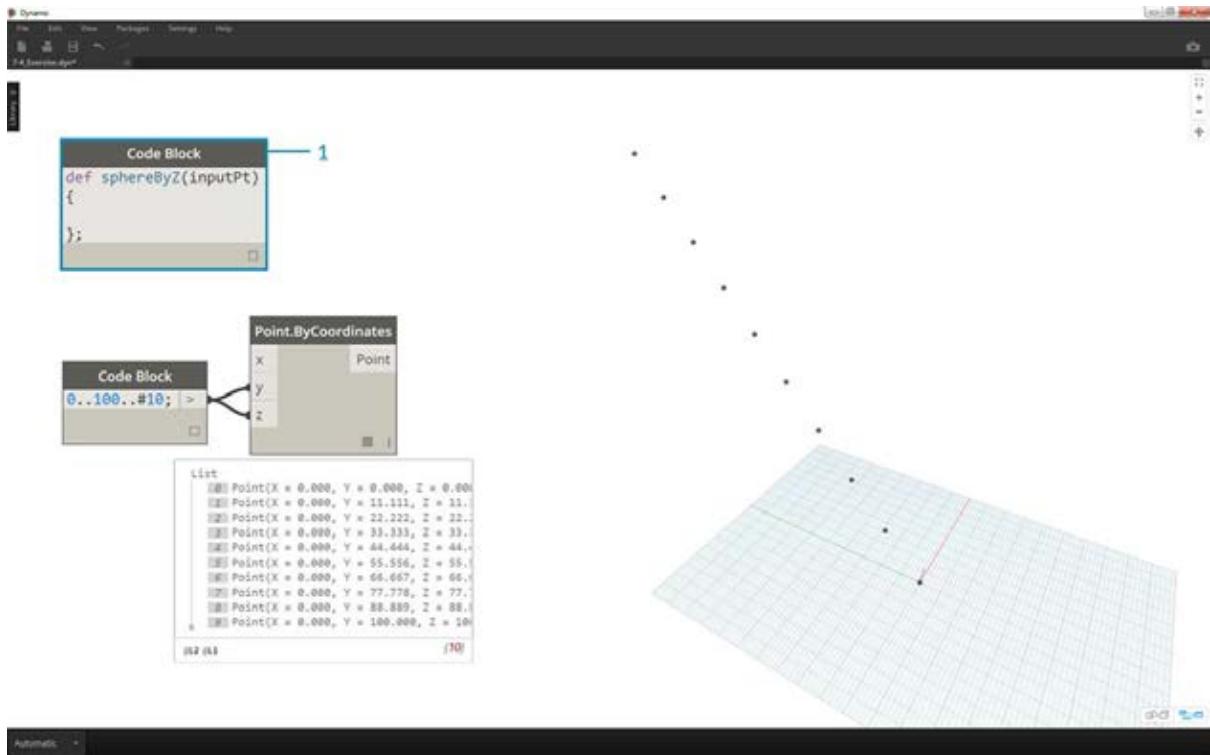
## Übungslektion

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Functions\\_SphereByZ.dyn](#)

In dieser Übung erstellen Sie eine allgemeine Definition zum Erstellen von Kugeln aus einer eingegebenen Liste von Punkten. Der Radius dieser Kugeln wird durch die z-Eigenschaft des jeweiligen Punkts gesteuert.



Sie beginnen mit einem Zahlenbereich, der zehn Werte von 0 bis 100 umfasst. Verbinden Sie diesen mit einem *Point.ByCoordinates*-Block, um eine diagonale Linie zu erhalten.



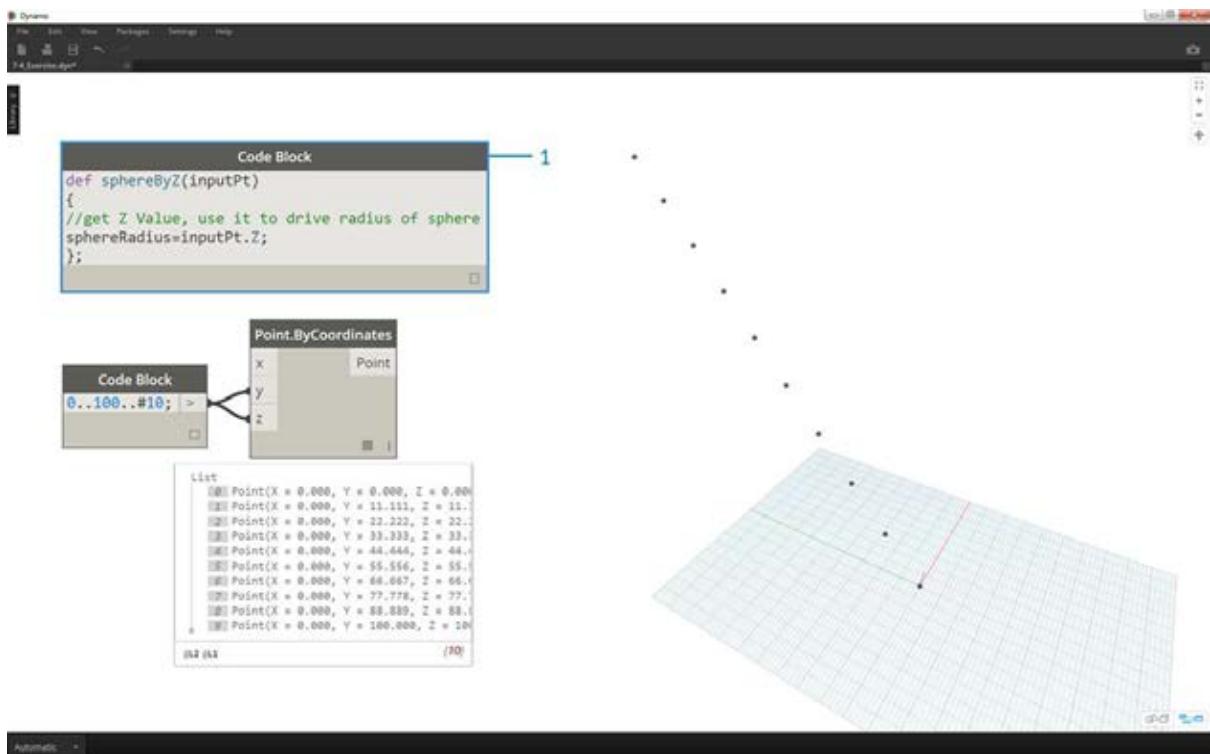
1. Erstellen Sie einen *Code Block* und geben Sie mithilfe der folgenden Codezeile die Definition ein:

```

def sphereByZ(inputPt){
};

```

Dabei ist *inputPt* der Name, den Sie für die Punkte zuweisen, über die die Funktion gesteuert werden soll. Bis hierher hat die Funktion keine Wirkung. Dies entwickeln Sie jedoch in den folgenden Schritten.

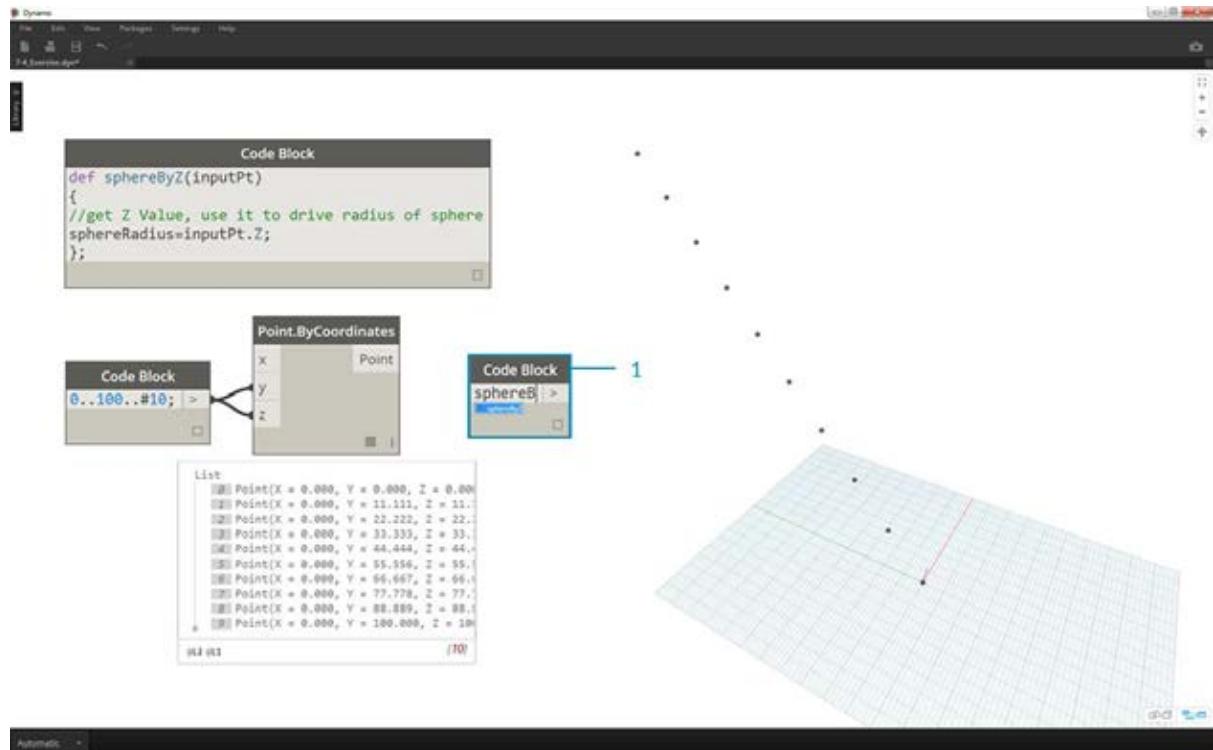


1. Fügen Sie der *Code Block*-Funktion einen Kommentar und die Variable *sphereRadius* hinzu, die die Z-Position der einzelnen Punkte abfragt. Beachten Sie, dass die Methode *inputPt.Z* keine Klammern benötigt. Da es sich um eine *Abfrage* von Eigenschaften eines bestehenden Elements handelt, sind keine Eingaben erforderlich.

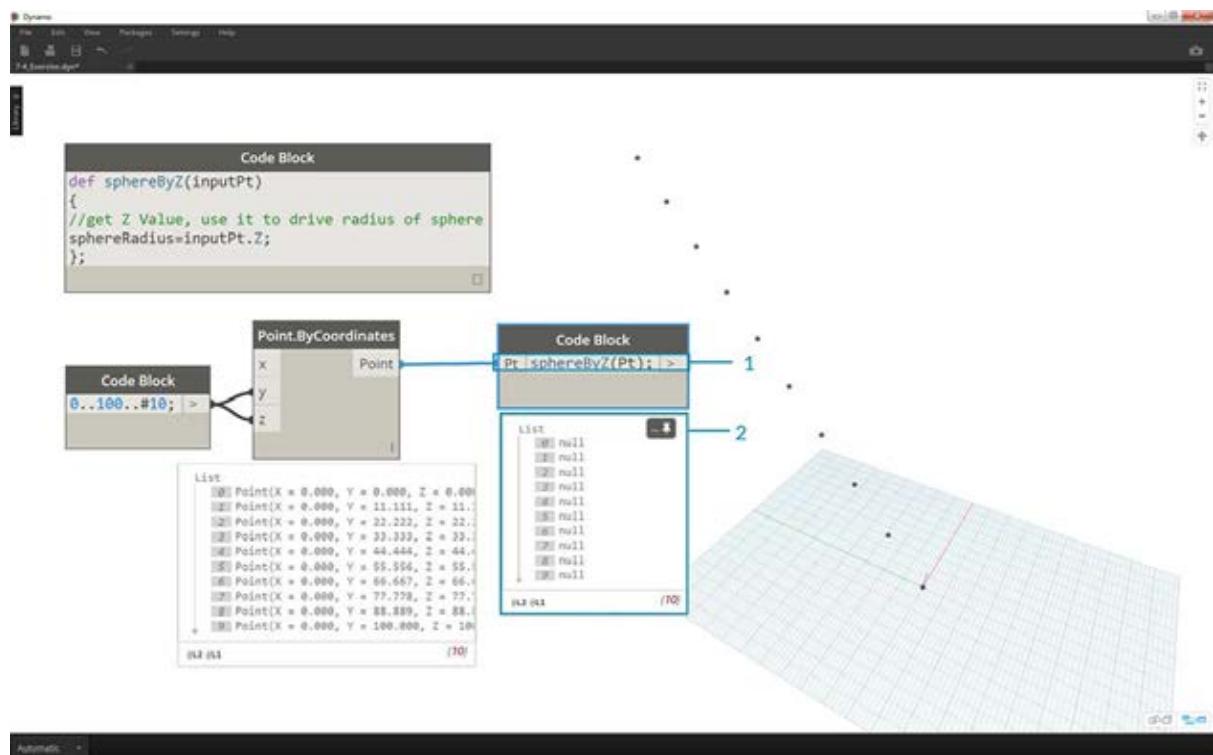
```

def sphereByZ(inputPt, radiusRatio)
{
    //get Z Value, use it to drive radius of sphere
    sphereRadius=inputPt.Z;
};

```



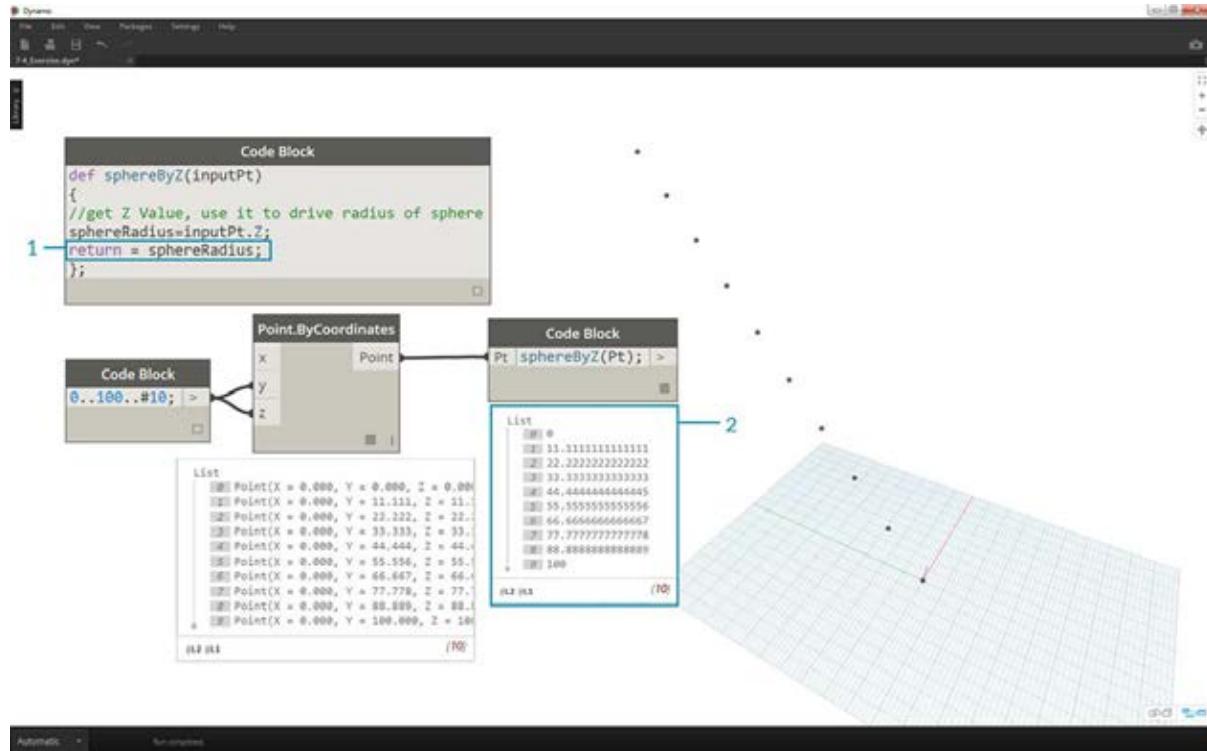
- Als Nächstes rufen Sie die eben erstellte Funktion in einem anderen *Code Block* auf. Wenn Sie im Ansichtsbereich doppelklicken, um einen neuen *Code Block* zu erstellen, und *sphereB* eingeben, schlägt *Dynamo* die eben definierte *sphereByZ*-Funktion vor: Die Funktion wurde der IntelliSense-Bibliothek hinzugefügt. Dieser Mechanismus ist sehr hilfreich.



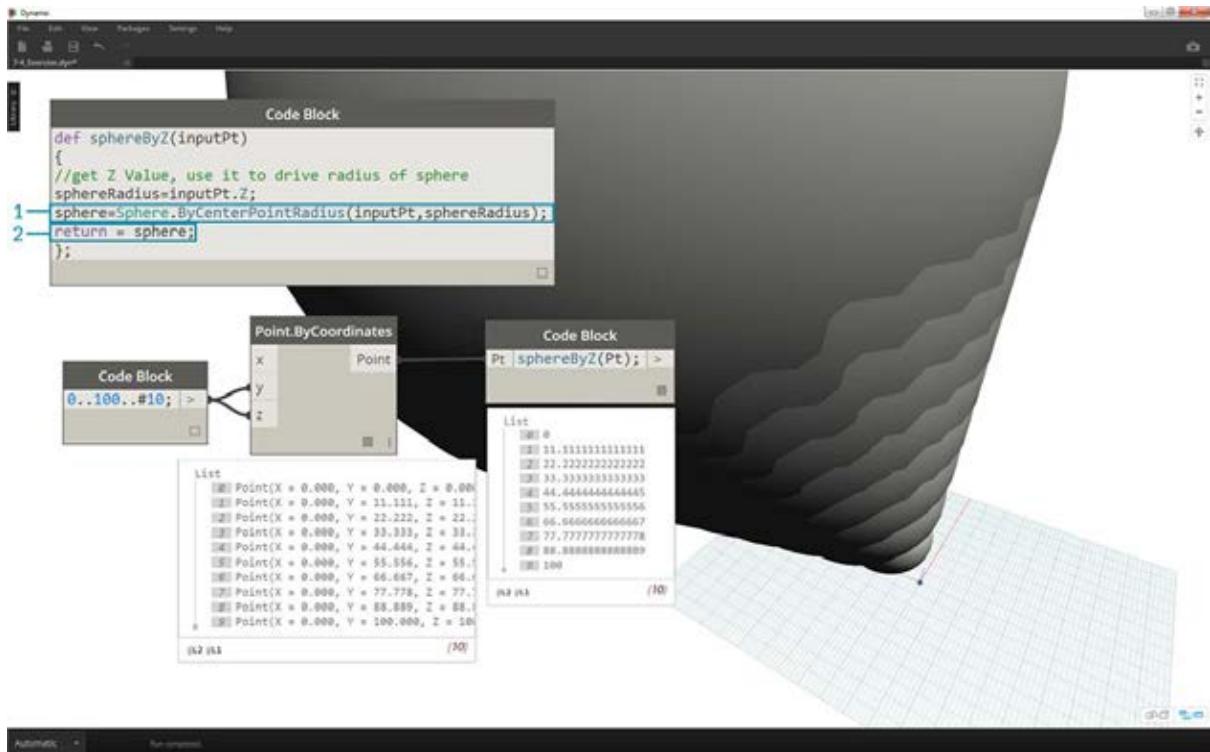
- Als Nächstes erstellen Sie eine Variable mit dem Namen *Pt*, um die Verbindung zu den Punkten herzustellen, die Sie in den vorherigen Schritten erstellt haben:

`sphereByZ(Pt)`

- Die Ausgabe zeigt lediglich Nullwerte. Dies geschieht aus dem folgenden Grund: Beim Definieren der Funktion wurde festgelegt, dass die Variable *sphereRadius* berechnet werden soll, Sie haben jedoch noch nicht definiert, was die Funktion als *Ausgabe* zurückgeben soll. Dies beheben Sie im nächsten Schritt.

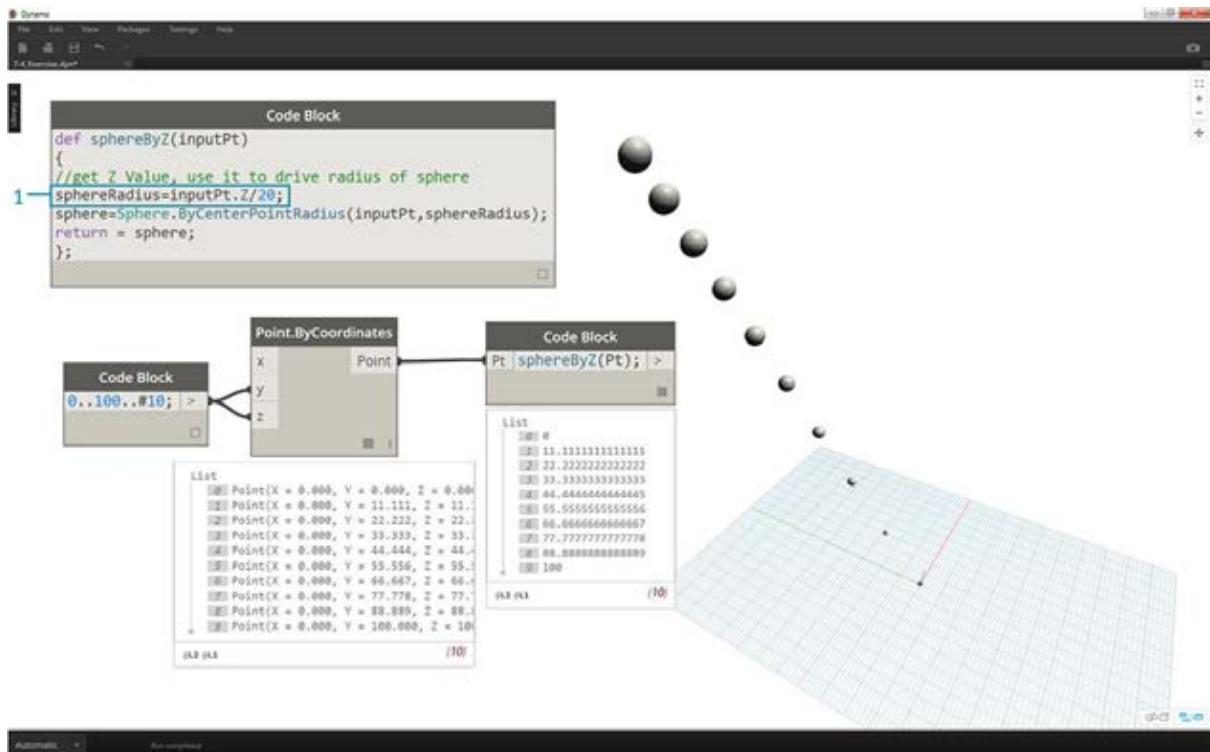


- In diesem wichtigen Schritt müssen Sie die Ausgabe der Funktion definieren. Fügen Sie die Zeile `return = sphereRadius;` in die *sphereByZ*-Funktion ein.
- Jetzt zeigt die Ausgabe im *Code Block* die z-Koordinaten der einzelnen Punkte an.

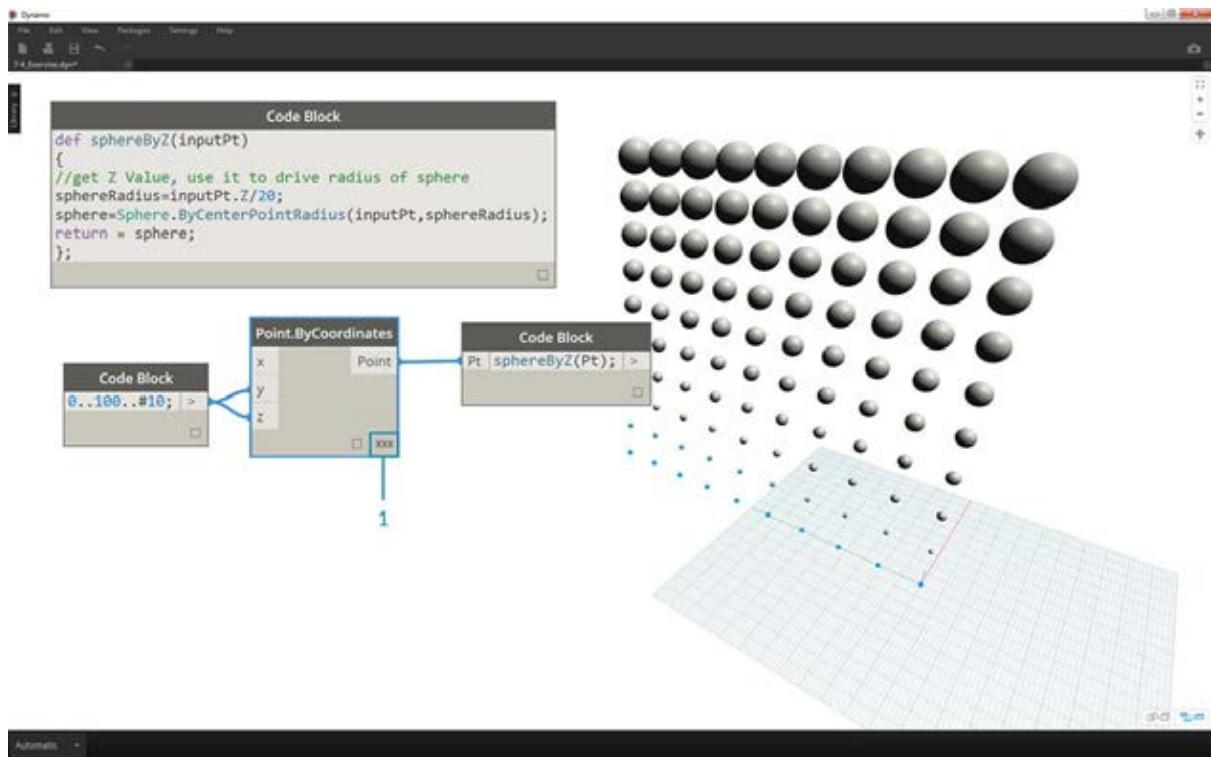


Als Nächstes erstellen Sie die Kugeln, indem Sie die *übergeordnete* Funktion bearbeiten.

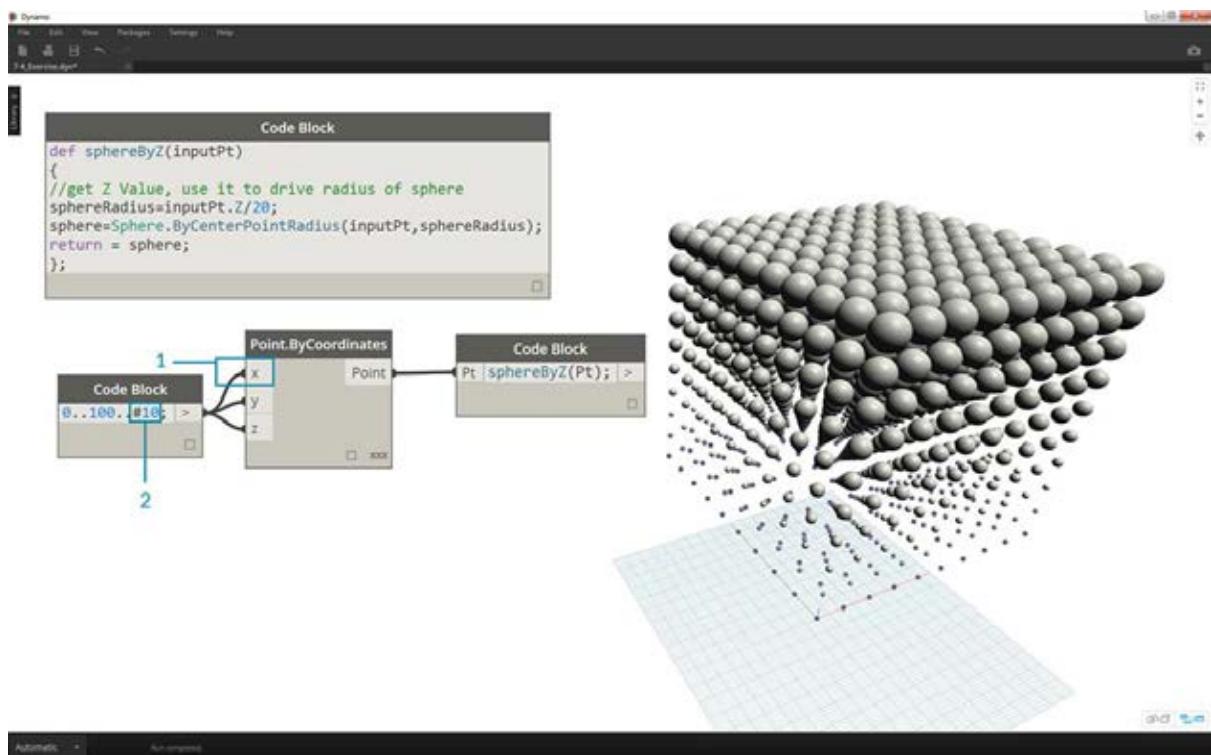
1. Definieren Sie zunächst eine Kugel mit der folgenden Codezeile:  
`sphere=Sphere.ByCenterPointRadius(inputPt,sphereRadius);`
2. Ändern Sie dann den return-Wert in *sphere* anstelle von *sphereRadius*: `return = sphere;`. Als Ergebnis erhalten Sie einige sehr große Kugeln in der Dynamo-Vorschau.



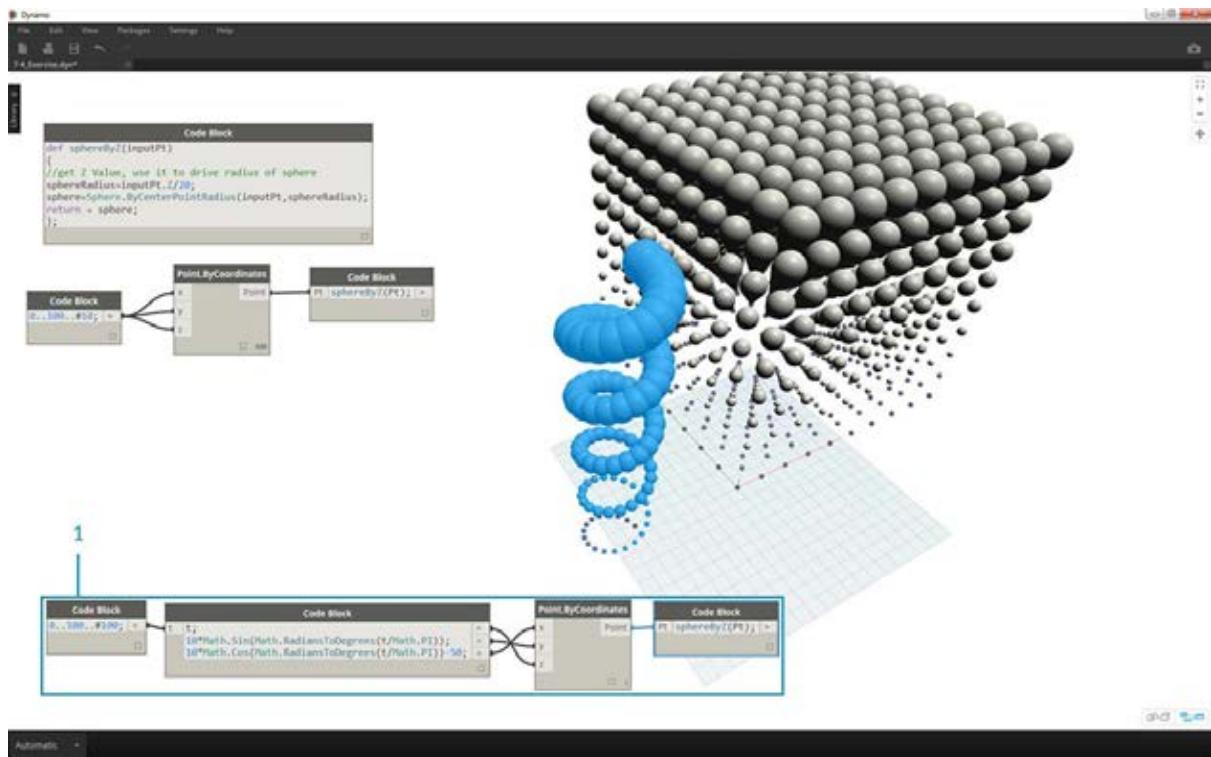
1. Um Kugeln von angemessenerer Größe zu erhalten, aktualisieren Sie den Wert von *sphereRadius*, indem Sie ihn dividieren: `sphereRadius = inputPt.Z/20;`. Jetzt sind die einzelnen Kugeln getrennt sichtbar und die Beziehung zwischen dem Radius und dem z-Wert wird erkennbar.



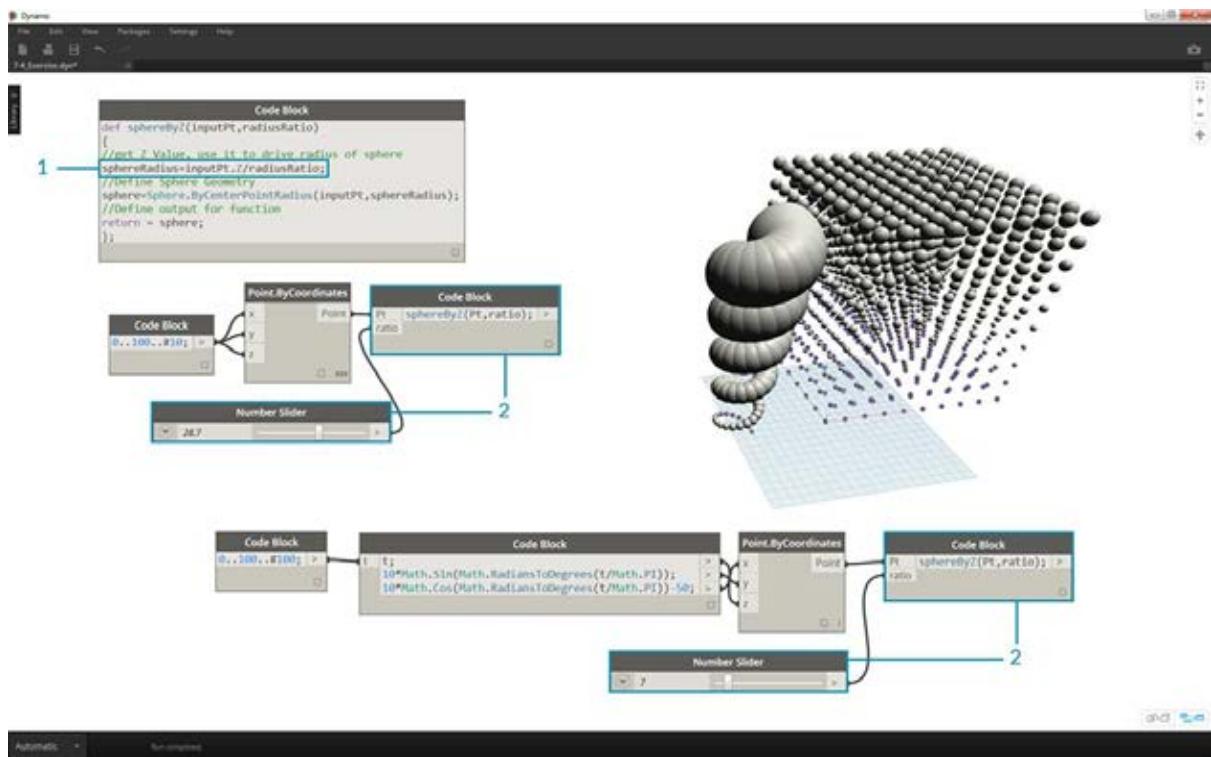
1. Indem Sie im *Point.ByCoordinates*-Block die Vergitterung von Kürzeste in Kreuzprodukt ändern, erstellen Sie ein Raster aus Punkten. Die *sphereByZ*-Funktion ist weiterhin uneingeschränkt wirksam, d. h., für alle Punkte werden Kugeln mit von ihren z-Werten abhängigen Radien erstellt.



1. Verbinden Sie jetzt als weiteren Test die ursprüngliche Zahlenliste mit der x-Eingabe von *Point.ByCoordinates*. Dadurch entsteht ein Würfel aus Kugeln.  
2. Anmerkung: Wenn diese Berechnung auf Ihrem Computer viel Zeit in Anspruch nimmt, ändern Sie den Wert #10 in einen kleineren Wert, z. B. #5.



1. Beachten Sie, dass die von Ihnen erstellte Funktion *sphereByZ* eine allgemeine Funktion ist. Sie können daher die Helix aus einer der vorigen Lektionen erneut öffnen und die Funktion darauf anwenden.



In einem letzten Schritt steuern Sie das Radienverhältnis über einen benutzerdefinierten Parameter. Zu diesem Zweck müssen Sie eine neue Eingabe für die Funktion erstellen und den Teiler 20 durch einen Parameter ersetzen.

1. Aktualisieren Sie die Definition von *sphereByZ* wie folgt:

```

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. Aktualisieren Sie die untergeordneten Codeblöcke, indem Sie der Eingabe die Variable *ratio* hinzufügen:  
`sphereByZ(Pt,ratio);`. Verbinden Sie einen Schieberegler mit der neu erstellten Codeblock-Eingabe und ändern Sie die Größe der Radien anhand des Radienverhältnisses.

## **Dynamo for Revit**

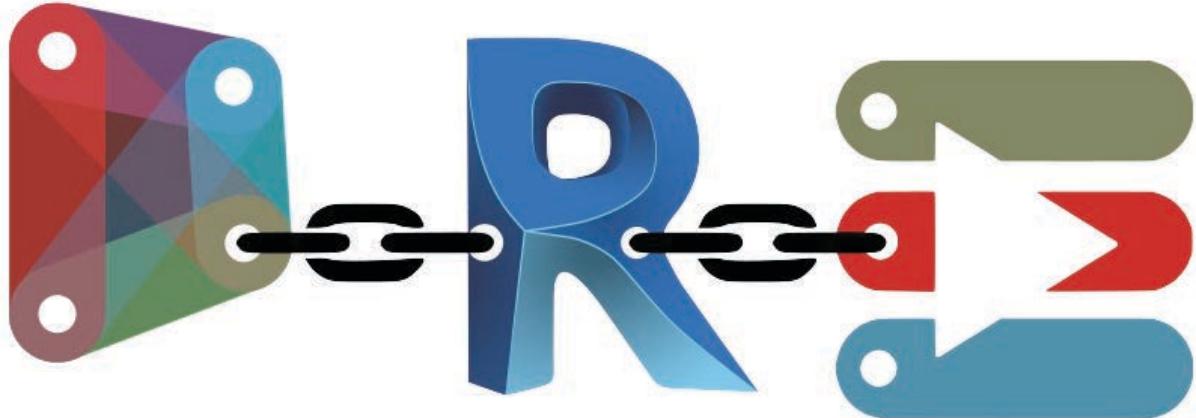
### **Dynamo for Revit**

Dynamo ist eine flexible Umgebung und für die Verwendung mit vielen verschiedenen Programmen vorgesehen. Ursprünglich wurde Dynamo allerdings für Revit entwickelt. Ein visuelles Programm eröffnet ein breites Spektrum an Möglichkeiten für Gebäudemodelle (Building Information Models, BIM). In Dynamo stehen eine ganze Suite mit eigens für Revit entwickelten Blöcken sowie Bibliotheken anderer Anbieter aus der sehr erfolgreichen AEC-Community zur Verfügung. Dieses Kapitel behandelt die Grundlagen der Verwendung von Dynamo in Revit.



# Verbindung zu Revit

## Verbindung zu Revit



Dynamo for Revit ergänzt die Gebäudemodellierung (Building Information Modelling) um die Daten- und Logikumgebung eines grafischen Algorithmeneditors. Die Flexibilität dieser Umgebung eröffnet zusammen mit einer robusten Revit-Datenbank neue Perspektiven für BIM.

Dieses Kapitel behandelt die Dynamo-Arbeitsabläufe für BIM. Die einzelnen Abschnitte enthalten im Wesentlichen Übungen, da sich ein grafischer Algorithmeneditor für BIM am besten anhand konkreter Projekte vorstellen lässt. Zunächst erhalten Sie hier jedoch eine Einführung in die Ursprünge dieses Programms.

#

### Kompatibilität mit Revit-Versionen

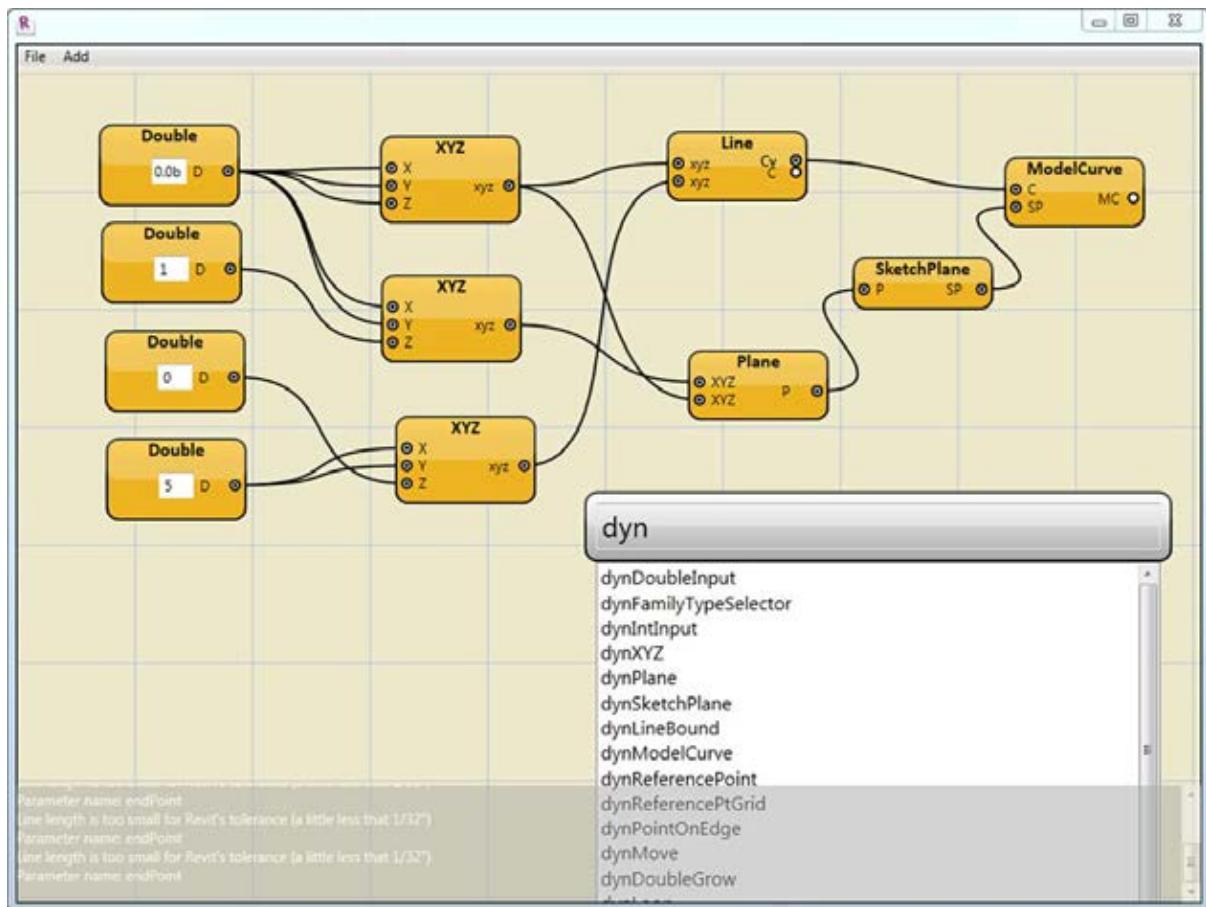
Da sich Revit und Dynamo beide immer weiterentwickeln, bemerken Sie möglicherweise, dass Ihre Revit-Version nicht mit der auf Ihrem Computer installierten Version von Dynamo for Revit kompatibel ist. Weiter unten wird erläutert, welche Versionen von Dynamo for Revit mit Revit kompatibel sind.

#### Revit-Version Erste stabile Version von Dynamo Letzte unterstützte Version von Dynamo for Revit

2013	<a href="#">0.6.1</a>	<a href="#">0.6.3</a>
2014	<a href="#">0.6.1</a>	<a href="#">0.8.2</a>
2015	<a href="#">0.7.1</a>	<a href="#">1.2.1</a>
2016	<a href="#">0.7.2</a>	<a href="#">1.3.2</a>
2017	<a href="#">0.9.0</a>	<a href="#">Letzter Daily Build</a>
2018	<a href="#">1.3.0</a>	<a href="#">Letzter Daily Build</a>
2019	<a href="#">1.3.3</a>	<a href="#">Letzter Daily Build</a>

#

### Entwicklung von Dynamo

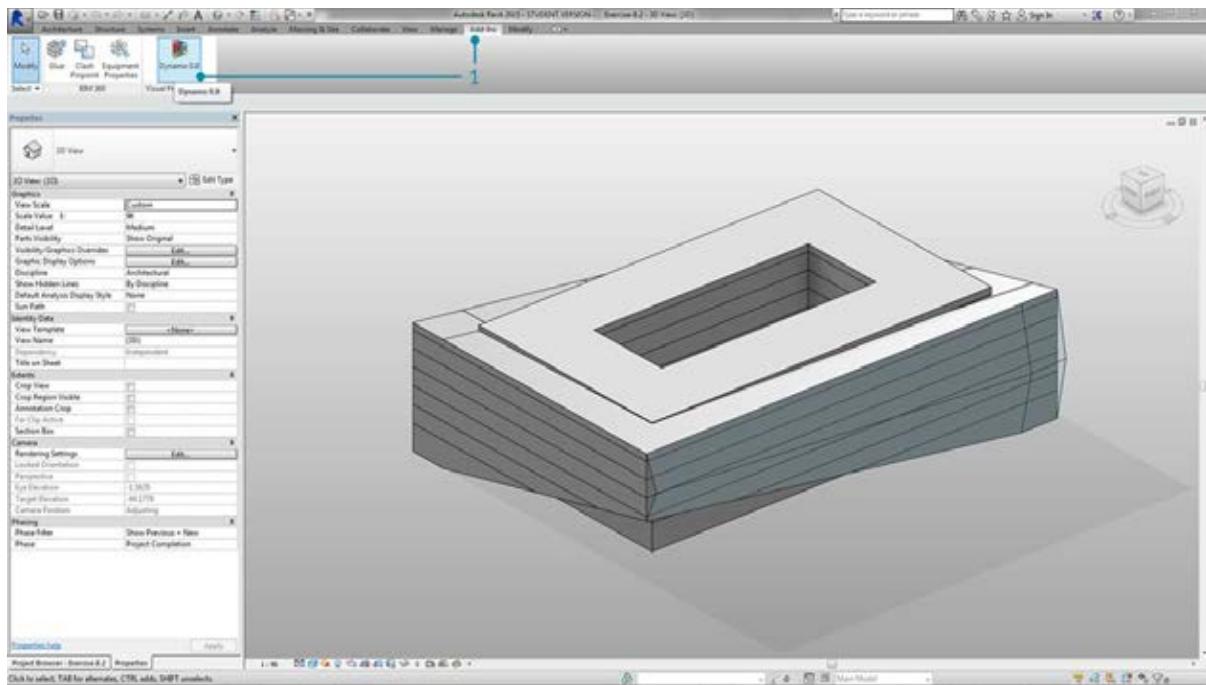


Ein spezialisiertes Entwicklerteam und eine engagierte Community haben dafür gesorgt, dass das Projekt seit seinen Anfängen kaum wiederzuerkennen ist.

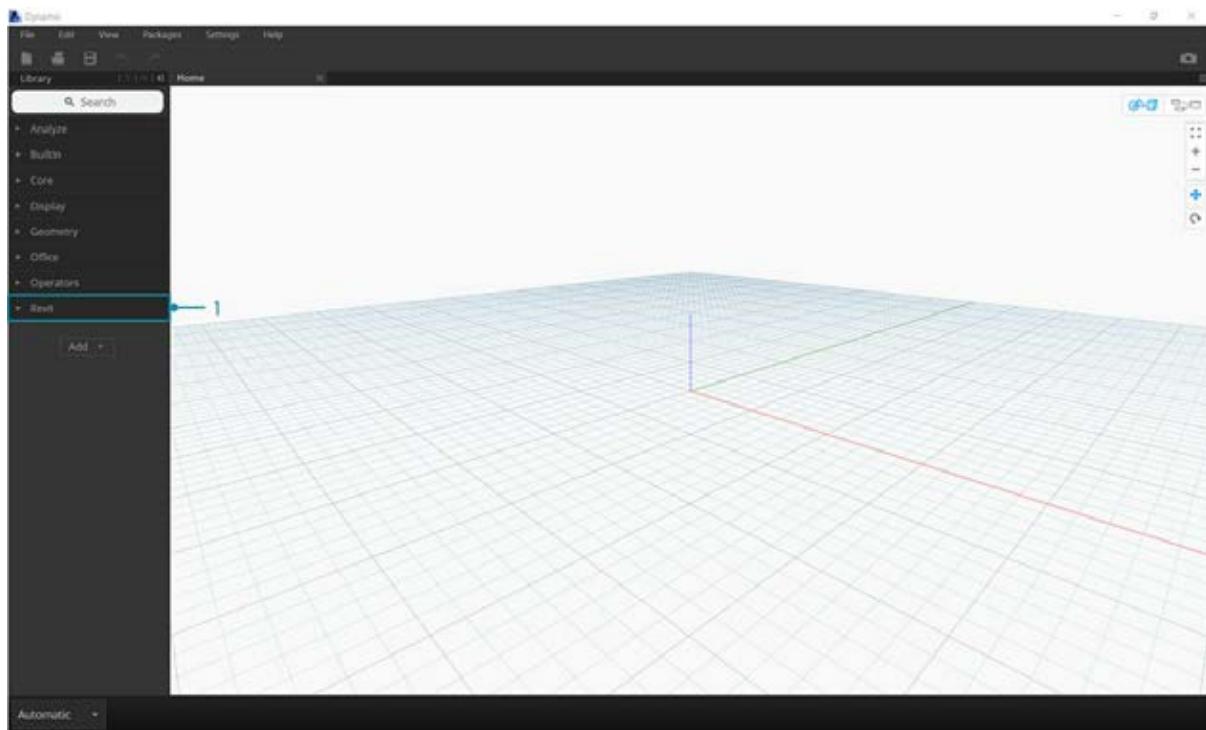
Dynamo wurde ursprünglich zur Beschleunigung von AEC-Arbeitsabläufen in Revit entwickelt. In Revit wird für jedes Projekt eine robuste Datenbank erstellt. Für den durchschnittlichen Benutzer kann jedoch der Zugriff auf diese Informationen außerhalb der Benutzeroberfläche problematisch sein. In Revit steht eine umfassende API (Application Program Interface) zur Verfügung, über die externe Entwickler eigene Werkzeuge erstellen können. Programmierer nutzen diese API schon seit Jahren. Textbasierte Skripterstellung ist jedoch nicht für jeden Benutzer möglich. Dynamo soll die Arbeit mit Revit-Daten gewissermaßen demokratisieren, indem ein leichter verständlicher grafischer Algorithmeneditor bereitgestellt wird.

Die Core-Dynamo-Blöcke ermöglichen es dem Benutzer in Verbindung mit benutzerdefinierten Revit-Blöcken, parametrische Arbeitsabläufe für Interoperabilität, Dokumentation, Analyse und Erstellung erheblich zu erweitern. Mit Dynamo können Sie lästige Arbeitsabläufe automatisieren und erfolgreich mit Entwürfen experimentieren.

## Ausführen von Dynamo in Revit



1. Navigieren Sie in einem Revit-Projekt oder im Familieneditor zu Zusatzmodule und klicken Sie auf *Dynamo*. Beachten Sie: *Dynamo* wird nur in der Datei ausgeführt, in der es geöffnet wurde.



1. Wenn Sie *Dynamo* in *Revit* öffnen, wird eine neue Kategorie namens "Revit" angezeigt. In dieser umfassenden Erweiterung der Benutzeroberfläche stehen spezielle Blöcke für *Revit*-Arbeitsabläufe zur Verfügung.\*

\*Anmerkung: Da die für *Revit* spezifische Familie von Blöcken verwendet wird, funktioniert das *Dynamo*-Diagramm nur, wenn Sie es in *Dynamo for Revit* öffnen. Wenn Sie ein Diagramm aus *Dynamo for Revit* beispielsweise in *Dynamo Sandbox* öffnen, fehlen die *Revit*-Blöcke.

## Anhalten von Blöcken

*Revit* ist eine Plattform mit robusten Funktionen zum Projektmanagement. Aus diesem Grund sind parametrische Operationen in *Dynamo* eventuell komplex und ihre Berechnung kann viel Zeit beanspruchen. Falls die Berechnung von

Blöcken in Dynamo sehr lange dauert, können Sie die Blockfunktionen anhalten ("einfrieren") und damit die Ausführung von Revit-Vorgängen unterbrechen, während Sie Ihr Diagramm entwickeln. Weitere Informationen zum Anhalten von Blöcken finden Sie im entsprechenden Abschnitt im Kapitel [Körper](#).

## Community

Da Dynamo ursprünglich für AEC erstellt wurde, bietet seine große und weiter wachsende Community ausgezeichnete Ressourcen, die es ermöglichen, von Experten der Branche zu lernen und sich mit ihnen auszutauschen. Die Dynamo-Community setzt sich zusammen aus Architekten, Ingenieuren, Programmierern und Designern, denen die Leidenschaft für Austausch und Entwicklung gemeinsam ist.

Dynamo ist ein Open Source-Projekt und entwickelt sich daher ständig weiter. Ein großer Teil dieser Entwicklungen steht in Verbindung mit Revit. Wenn Sie neu hinzugekommen sind, können Sie damit beginnen, im Diskussionsforum [Fragen zu stellen](#). Programmierer, die sich an der Entwicklung von Dynamo beteiligen möchten, finden auf der [GitHub-Seite](#) das Nötige. Eine hervorragende Ressource für Bibliotheken externe Bibliotheken ist auch der [Dynamo Package Manager](#). Viele dieser Pakete werden für die Verwendung mit AEC entwickelt. In diesem Kapitel werden solche extern entwickelten Pakete für die Unterteilung von Oberflächen in Felder vorgestellt.



The screenshot shows a web browser displaying the [Dynamo blog](https://dynamobim.org/blog/). The main header features the word "Dynamo" with a gear icon. Below the header is a large, dark image of a tire tread. A search bar labeled "Search blog posts..." is positioned above the main content area. The title of the post is "Dynamo and IronPython". The post content discusses Python functionality in Windows and includes a link to the "Dynamo FAQ". On the right side of the page, there is a sidebar titled "Archive" with a list of months from March 2018 down to October 2016.

Dynamo and IronPython

The Dynamo team recently became aware of some reports that python in Windows was not functioning on some machines with Dynamo installed. We've added some information to the [Dynamo FAQ] will attempt, and would like to share it here as well in case you run into a similar problem. Q: How do I get [Dynamo] Python functionality to work again, and use it?

Archive

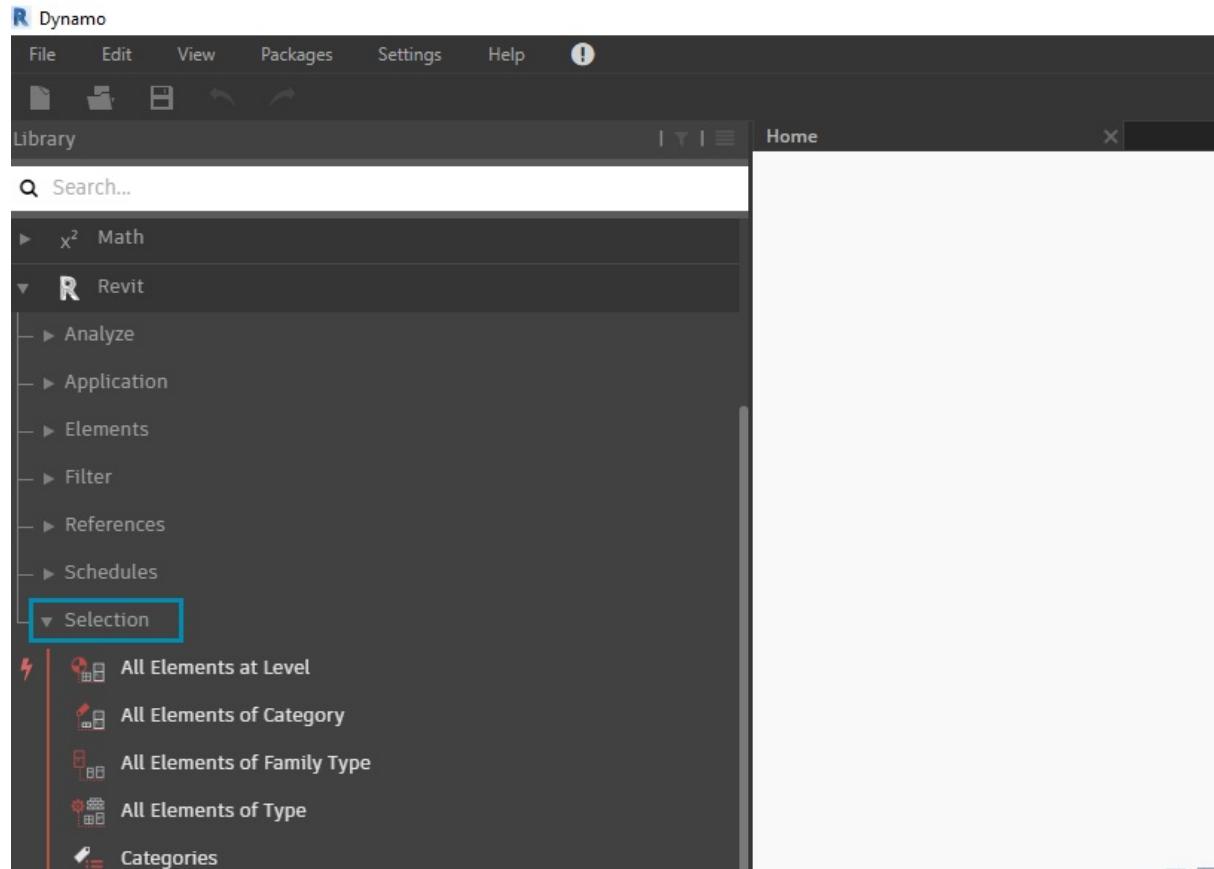
- MARCH 2018
- JANUARY 2018
- DECEMBER 2017
- NOVEMBER 2017
- OCTOBER 2017
- SEPTEMBER 2017
- AUGUST 2017
- JULY 2017
- APRIL 2017
- MARCH 2017
- JANUARY 2017
- DECEMBER 2016
- NOVEMBER 2016
- OCTOBER 2016

Für Dynamo steht auch ein aktives [Blog](#) zur Verfügung. Lesen Sie die neuesten Posts, um sich über neue Entwicklungen auf dem Laufenden zu halten.

# Auswählen

## Auswählen

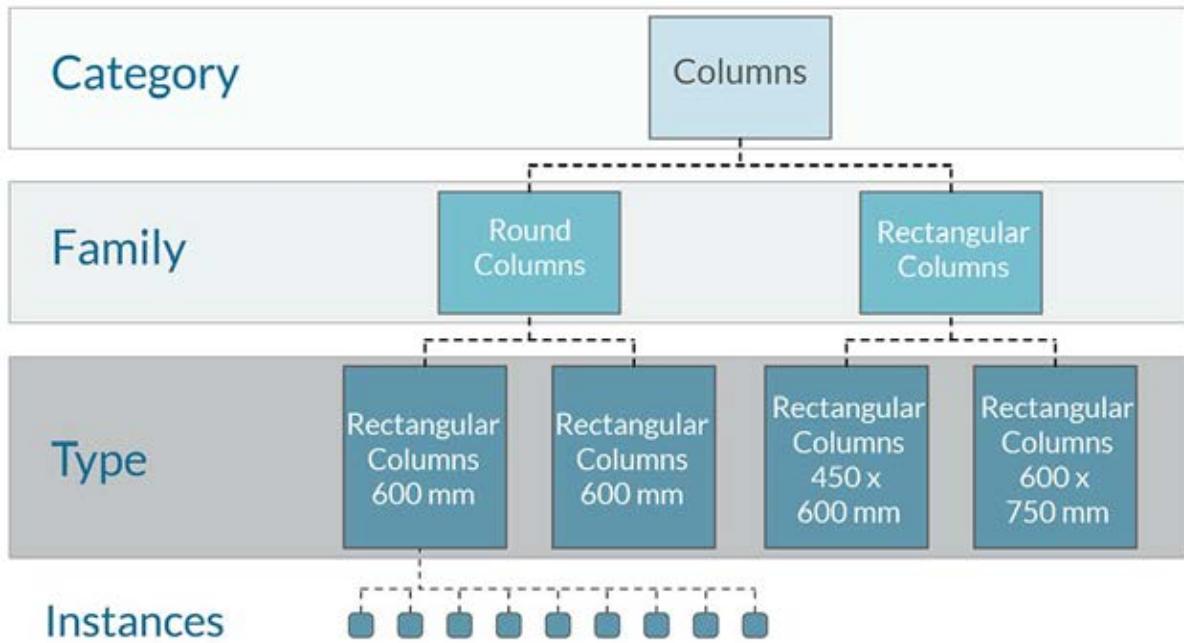
Revit ist eine datenintensive Umgebung. Dadurch steht ein ganzes Spektrum von Auswahlfunktionen über das gewohnte "Zeigen und Klicken" hinaus zur Verfügung. Es ist möglich, während der Durchführung parametrischer Vorgänge die Revit-Datenbank abzufragen und Revit-Elemente dynamisch mit Dynamo-Geometrie zu verknüpfen.



Die in der Benutzeroberfläche enthaltene Revit-Bibliothek enthält die Kategorie "Selection" mit mehreren Möglichkeiten zur Auswahl von Geometrie.

Damit Sie Revit-Elemente Ihren Zwecken entsprechend auswählen können, müssen Sie die Elementhierarchie in Revit kennen. Sollen alle Wände in einem Projekt ausgewählt werden? Wählen Sie sie nach Kategorie aus. Möchten Sie alle Eames-Stühle in Ihrer Diele im Stil der Jahrhundertmitte auswählen? In diesem Fall wählen Sie sie nach Familie aus. Machen Sie sich kurz mit der Revit-Hierarchie vertraut, bevor Sie mit einer Übung beginnen.

## Revit-Hierarchie

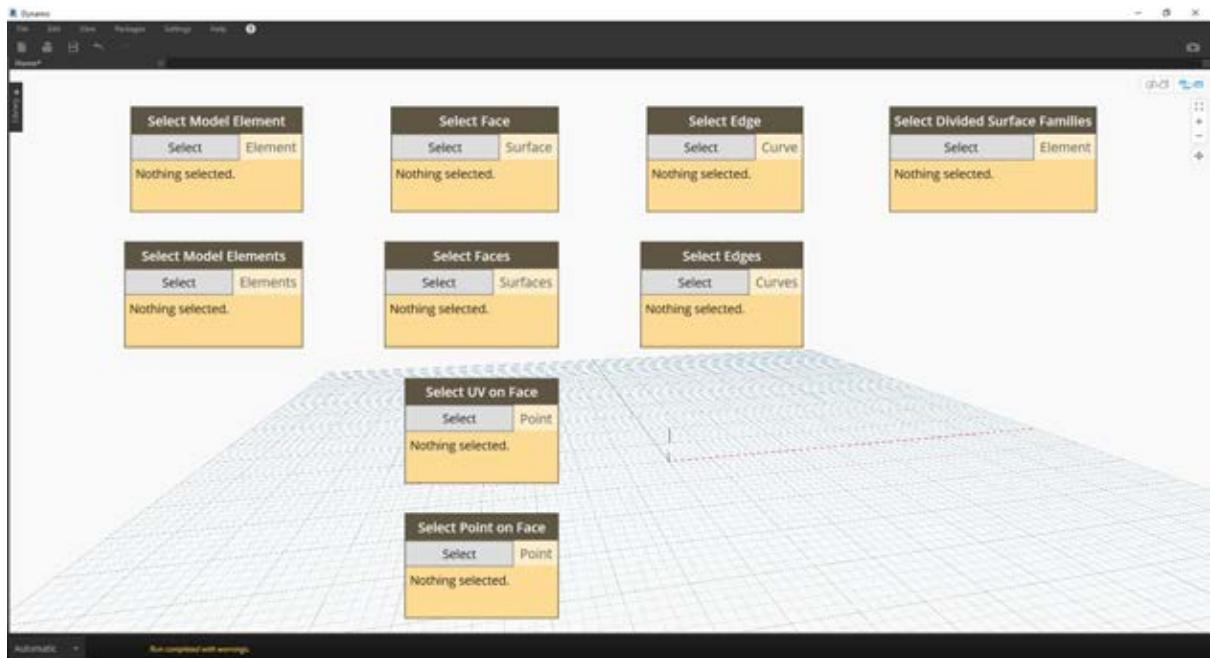


Dies lässt sich mit der Taxonomie in der Biologie vergleichen: Reich, Stamm, Klasse, Ordnung, Familie, Gattung, Art. Die Elemente in Revit sind auf ähnliche Weise geordnet. Die Revit-Hierarchie gliedert sich grundsätzlich in Kategorien, Familien, Typen\* und Exemplare. Ein Exemplar ist ein einzelnes Modellement (mit eindeutiger ID), während eine Kategorie eine allgemeine Gruppe definiert (z. B. Wände oder Geschossdecken). Diese Struktur der Revit-Datenbank ermöglicht es, ein Element und anhand der angegebenen Ebene in der Hierarchie auch alle ihm ähnlichen Elemente auszuwählen.

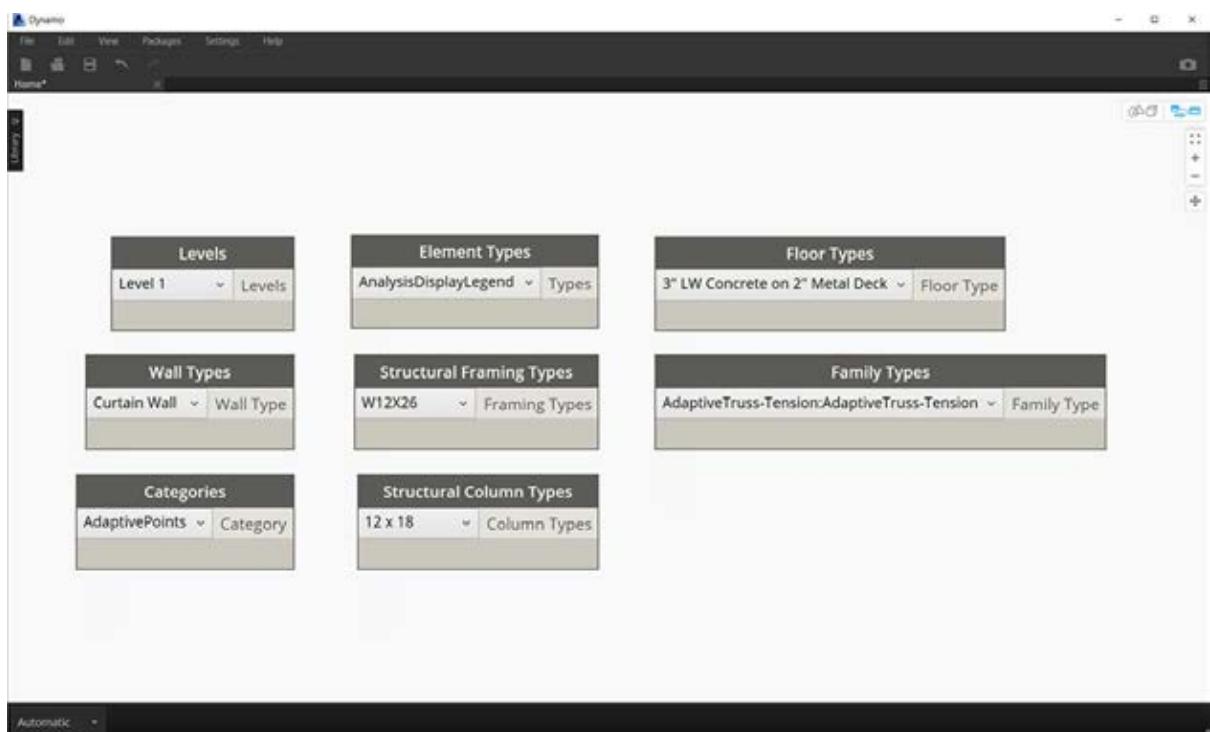
*\*Hinweis: Typen sind in Revit auf andere Weise definiert als in der Programmierung. In Revit ist mit einem Typ ein Zweig in der Hierarchie, nicht ein "Datentyp" gemeint.*

#### Datenbanknavigation mit Dynamo-Blöcken

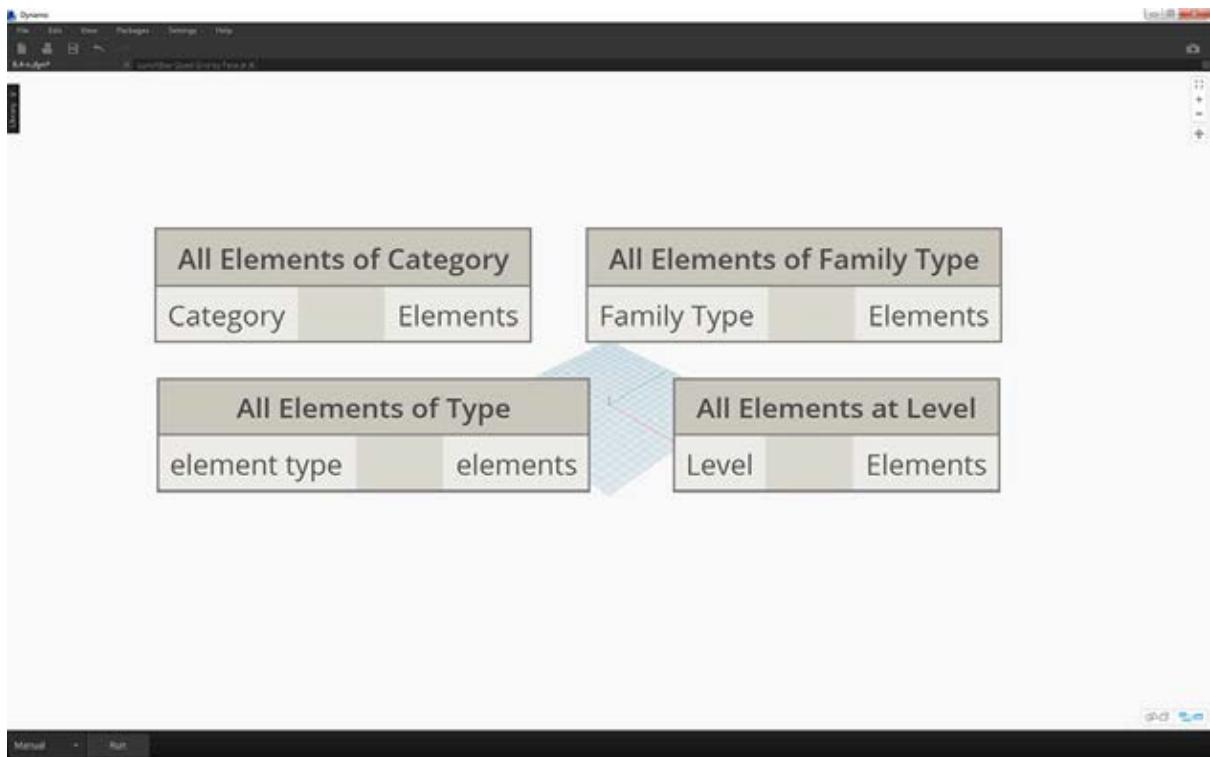
Die drei Abbildungen unten zeigen die wichtigsten Kategorien für die Auswahl von Revit-Elementen in Dynamo im Detail. Sie lassen sich hervorragend miteinander kombinieren, was in den weiter unten folgenden Übungen genauer beschrieben wird.



**Zeigen und Klicken:** Dies ist das einfachste Verfahren zur direkten Auswahl von Revit-Elementen. Sie können dabei ein vollständiges Modellelement oder nur Teile seiner Topologie (z. B. eine Fläche oder Kante) auswählen. Diese bleiben dynamisch mit dem Revit-Objekt verknüpft, d. h., wenn in der Revit-Datei die Position oder Parameter geändert werden, wird das referenzierte Dynamo-Element im Diagramm aktualisiert.



**Dropdown-Menüs:** In diesen Menüs wird eine Liste aller Elemente im Revit-Projekt erstellt, auf die Sie zugreifen können. Über diese können Sie Revit-Elemente referenzieren, die eventuell in der jeweiligen Ansicht nicht sichtbar sind. Dieses Werkzeug eignet sich hervorragend zum Abfragen bestehender und zum Erstellen neuer Elemente in einem Revit-Projekt oder im Familieneditor.



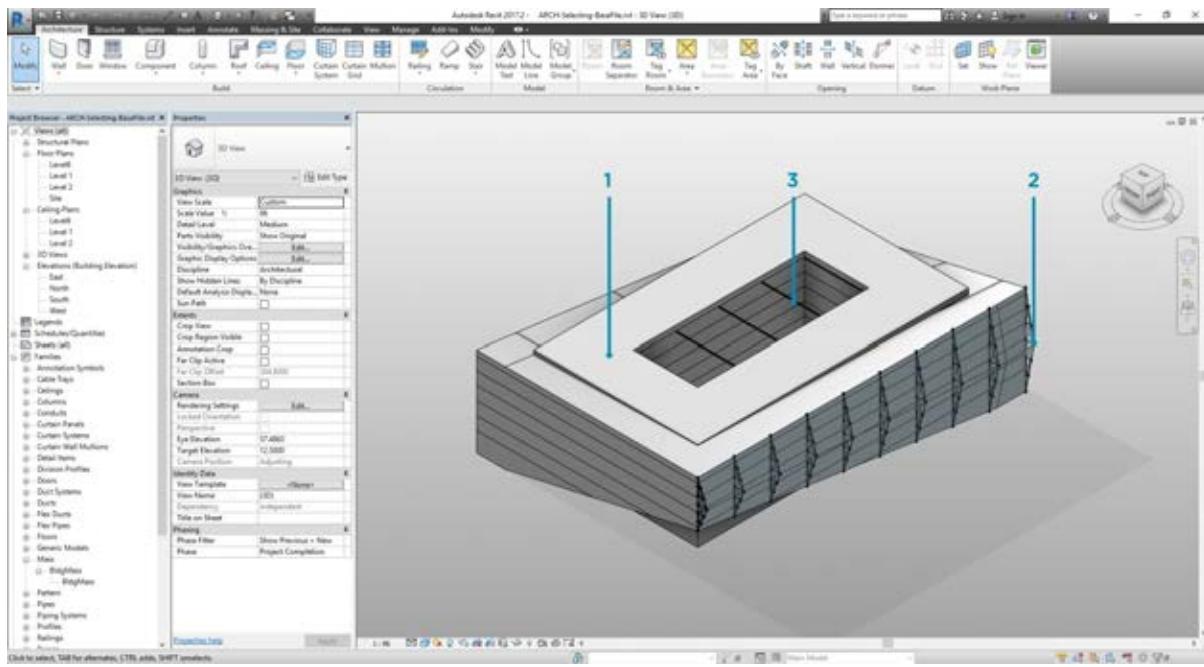
Sie können darüber hinaus Revit-Elemente anhand ihrer Ebene in der *Revit-Hierarchie* auswählen. Diese Option erweist sich als hocheffizient, wenn große Datenmengen zur Vorbereitung der Dokumentation oder für die generative Instanziierung und Anpassung verarbeitet werden müssen.

Die drei oben stehenden Abbildungen bilden die Grundlage für die folgende Übung, in der Sie Elemente in einem einfachen Revit-Projekt auswählen. Dies dient als Vorbereitung für die parametrischen Anwendungen, die Sie in den übrigen Abschnitten dieses Kapitels erstellen.

## Übung

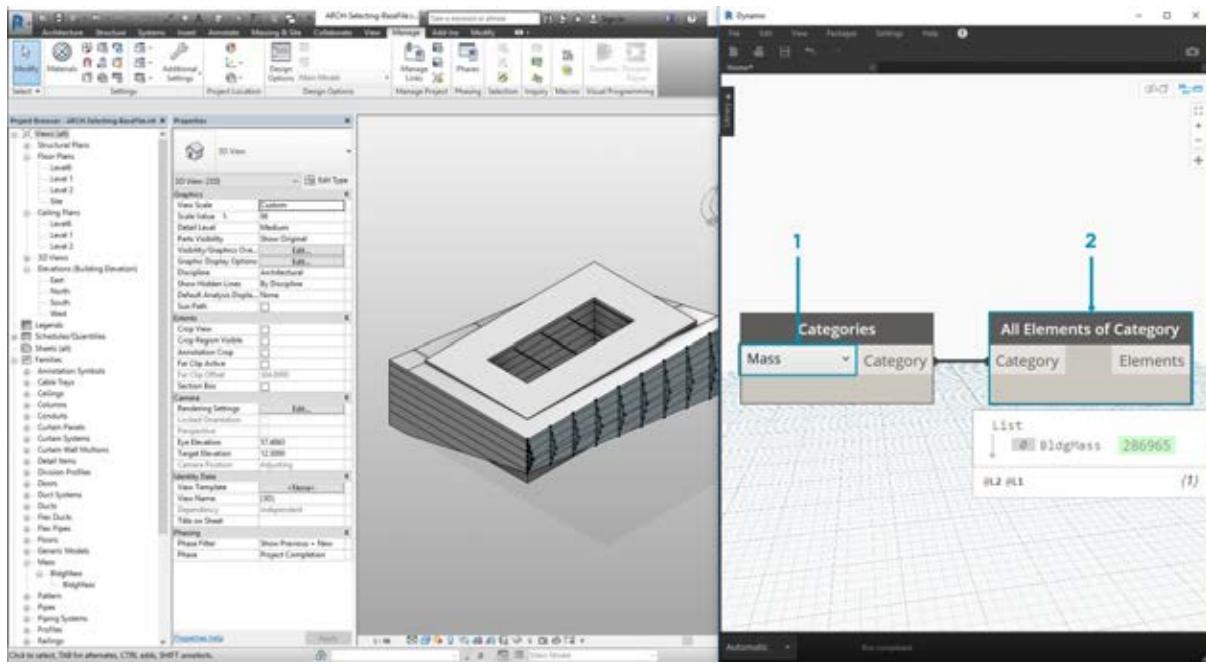
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

1. [Selecting.dyn](#)
2. [ARCH-Selecting-BaseFile.rvt](#)



Diese Revit-Beispieldatei enthält drei Elementtypen eines einfachen Gebäudes. Diese dient als Beispiel für die Auswahl von Revit-Elementen innerhalb der Revit-Hierarchie.

1. Gebäudekörper
  2. Fachwerk (adaptive Bauteile)
  3. Träger (Tragwerkselemente)

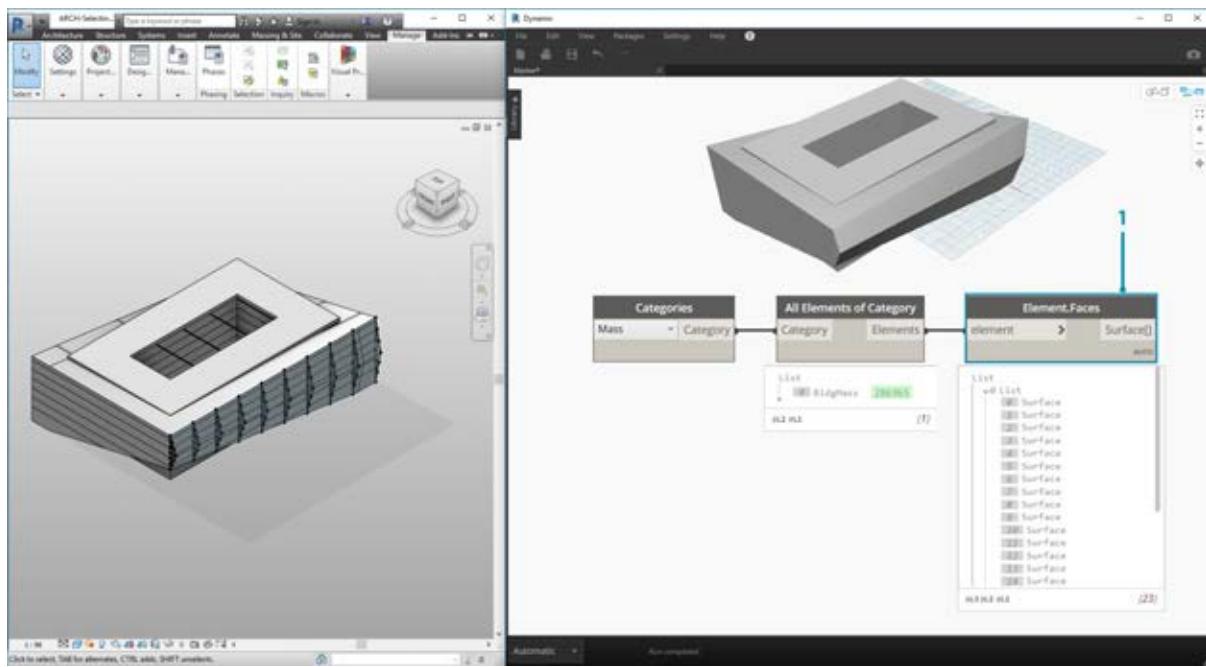


Welche Schlussfolgerungen erlauben die momentan in der Revit-Projektansicht angezeigten Elemente? Welche Ebene in der Hierarchie wird zur Auswahl der gewünschten Elemente benötigt? Bei der Arbeit in umfangreichen Projekten gestalten sich diese Fragen selbstverständlich komplexer. Es stehen zahlreiche Optionen zur Verfügung: Sie können Elemente nach Kategorien, Ebenen, Familien, Exemplaren usw. auswählen.

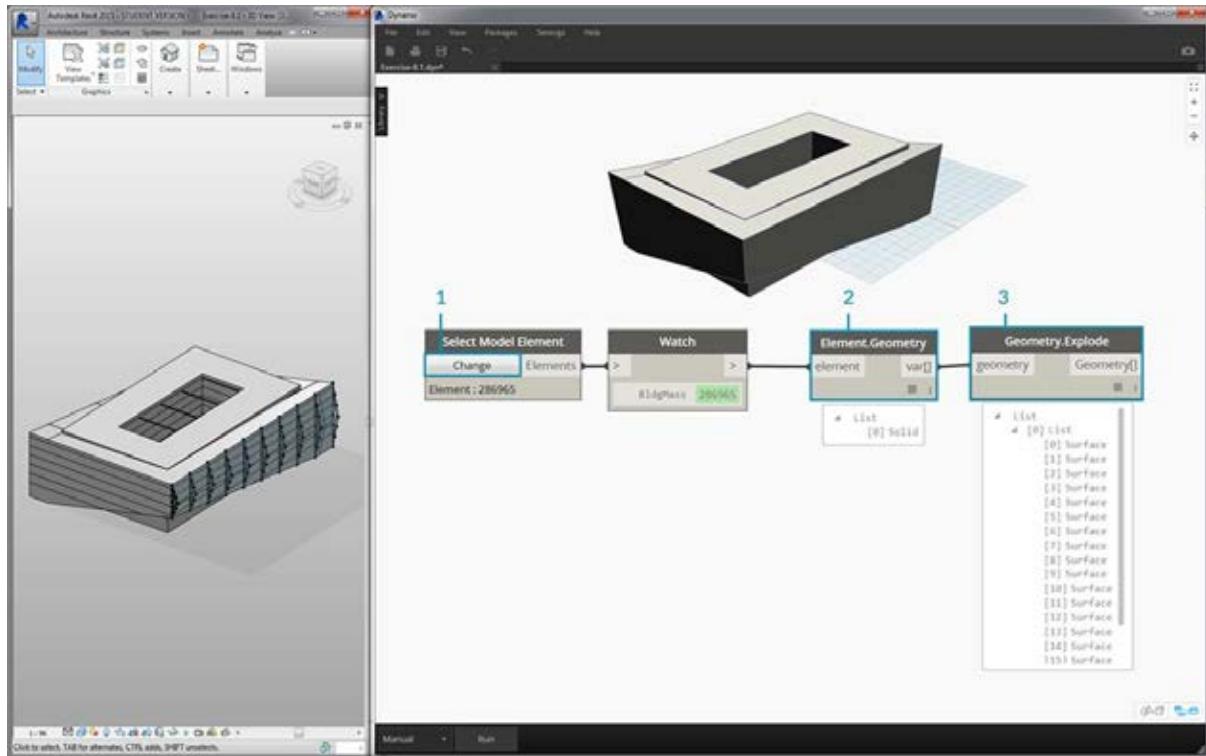
1. In diesem Fall liegt eine einfache Struktur vor. Wählen Sie daher den Gebäudekörper, indem Sie im Menüblock **Categories** die Option **Mass** wählen. Sie finden dies über Revit > Selection.
2. Die Ausgabe der Kategorie Mass ist die Kategorie selbst. Es müssen jedoch Elemente auswählen. Verwenden Sie hierfür den Block **All Elements of Category**.

Bis jetzt wird noch keine Geometrie in Dynamo angezeigt. Sie haben ein Revit-Element ausgewählt, dieses jedoch noch nicht in Dynamo-Geometrie konvertiert. Diese Unterscheidung ist wichtig. Angenommen, Sie wählen eine große Zahl von Elementen aus. In diesem Fall wäre es nicht sinnvoll, alle diese Elemente in der Vorschau in Dynamo anzusehen, da dies die Leistung beeinträchtigen würde. Dynamo ist ein Werkzeug zur Verwaltung von Revit-Projekten, ohne dass notwendigerweise Geometrie verarbeitet wird. Dies wird im nächsten Abschnitt dieses Kapitels näher betrachtet.

In diesem Fall arbeiten Sie jedoch mit einfacher Geometrie, die in der Dynamo-Vorschau angezeigt werden kann. Neben dem Eintrag "BldgMass" im oben gezeigten Watch-Block steht eine grün unterlegte Nummer\*. Dies ist die ID des Elements. Sie weist außerdem darauf hin, dass wir es mit einem Revit-Element zu tun haben, und nicht mit Dynamo-Geometrie. Im nächsten Schritt konvertieren Sie dieses Revit-Element in Dynamo-Geometrie.

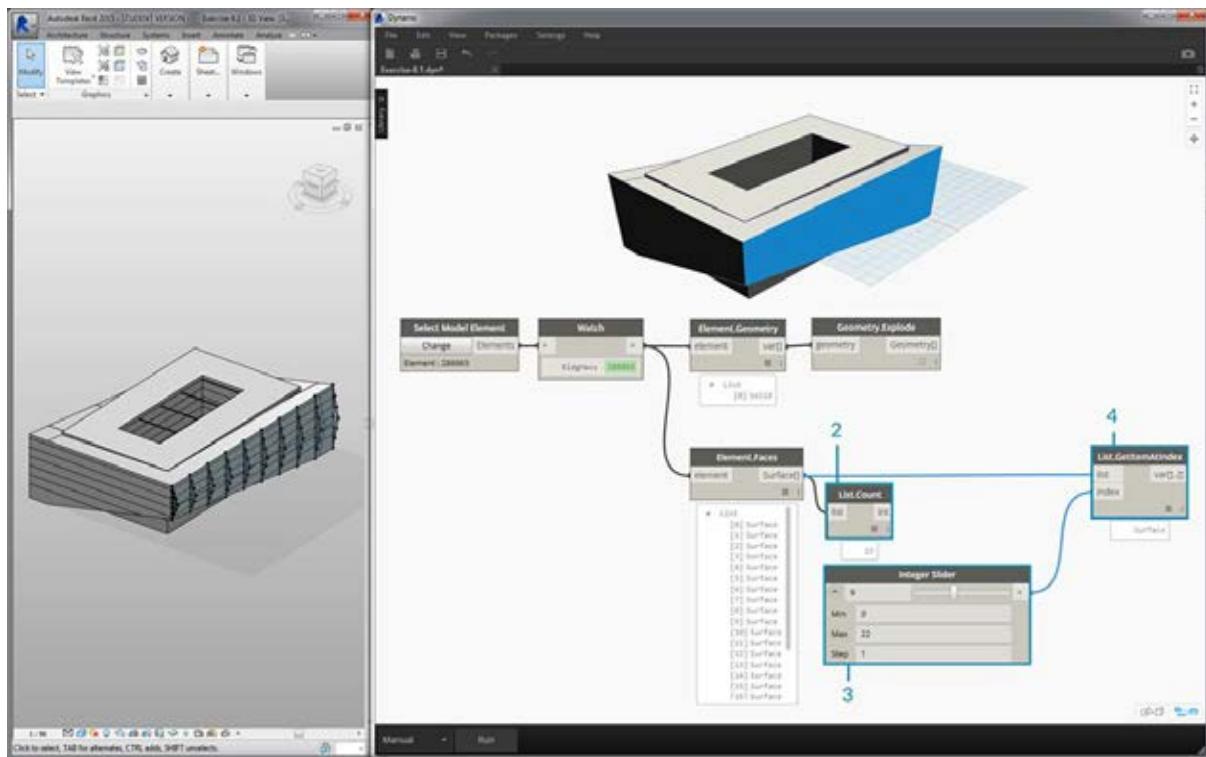


1. Durch Verwenden des Blocks *Element.Faces* erhalten Sie eine Liste von Oberflächen, die die einzelnen Flächen des Revit-Körpers repräsentieren. Die Geometrie wird jetzt im Dynamo-Ansichtsfenster angezeigt und Sie können damit beginnen, Flächen für parametrische Operationen zu referenzieren.

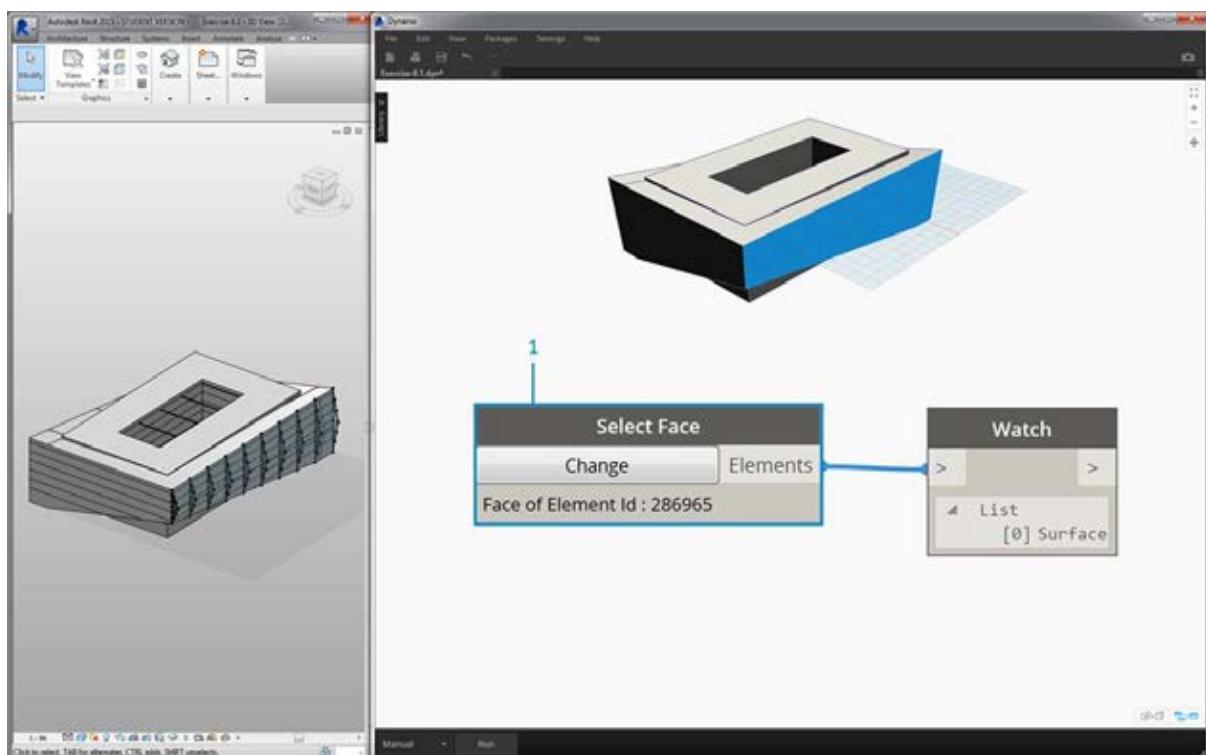


Eine Alternative dazu wird im Folgenden beschrieben. In diesem Fall nehmen Sie die Auswahl nicht über die Revit-Hierarchie (*All Elements of Category*) vor, sondern wählen Geometrie explizit in Revit aus.

1. Klicken Sie im Block *Select Model Element* auf die Schaltfläche *select* (bzw. *change*). Wählen Sie im Revit-Ansichtsfenster das gewünschte Element aus. In diesem Fall ist dies der Gebäudekörper.
2. Anstelle von *Element.Faces* können Sie mithilfe von *Element.Geometry* den gesamten Körper als Volumengeometrie auswählen. Dadurch wird die gesamte in diesem Körper enthaltene Geometrie ausgewählt.
3. Mit *Geometry.Explode* erhalten Sie ebenfalls die Liste der Oberflächen. Diese beiden Blöcke haben dieselbe Wirkung wie *Element.Faces*, aber sie bieten andere Optionen zum Ansteuern der Geometrie eines Revit-Elements.

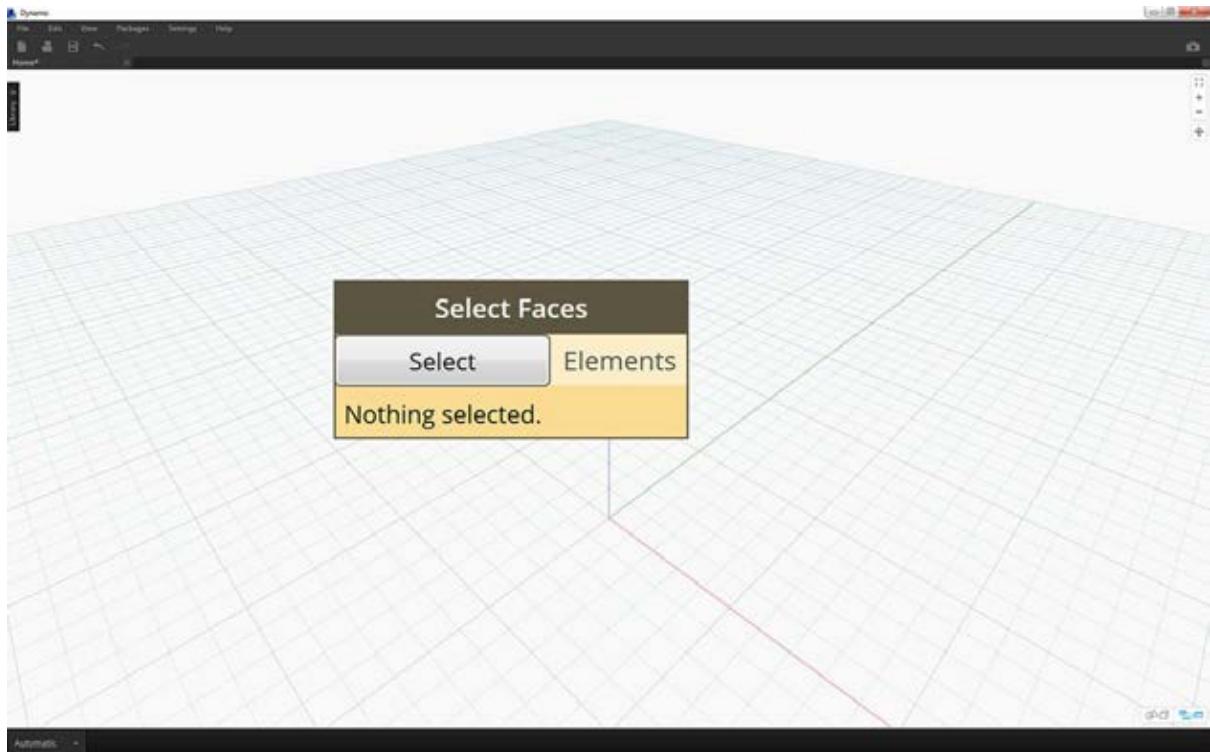


- Mithilfe einiger einfacher Listenoperationen können Sie eine bestimmte Fläche abrufen, die für Ihre Zwecke von Interesse ist.
- Der *List.Count*-Block zeigt zunächst, dass 23 Oberflächen im Körper enthalten sind.
- Ändern Sie entsprechend dieser Zahl den Höchstwert im *Integer Slider* zu 22.
- Geben Sie in *List.GetItemAtIndex* die Listen und die Werte aus dem *Integer Slider* für *index* ein. Wählen Sie durch Ziehen des Schiebereglers die Werte nacheinander aus und halten Sie bei *index* 9 an: Damit haben Sie die Hauptfassade ausgewählt, an der sich die Fachwerkelemente befinden.

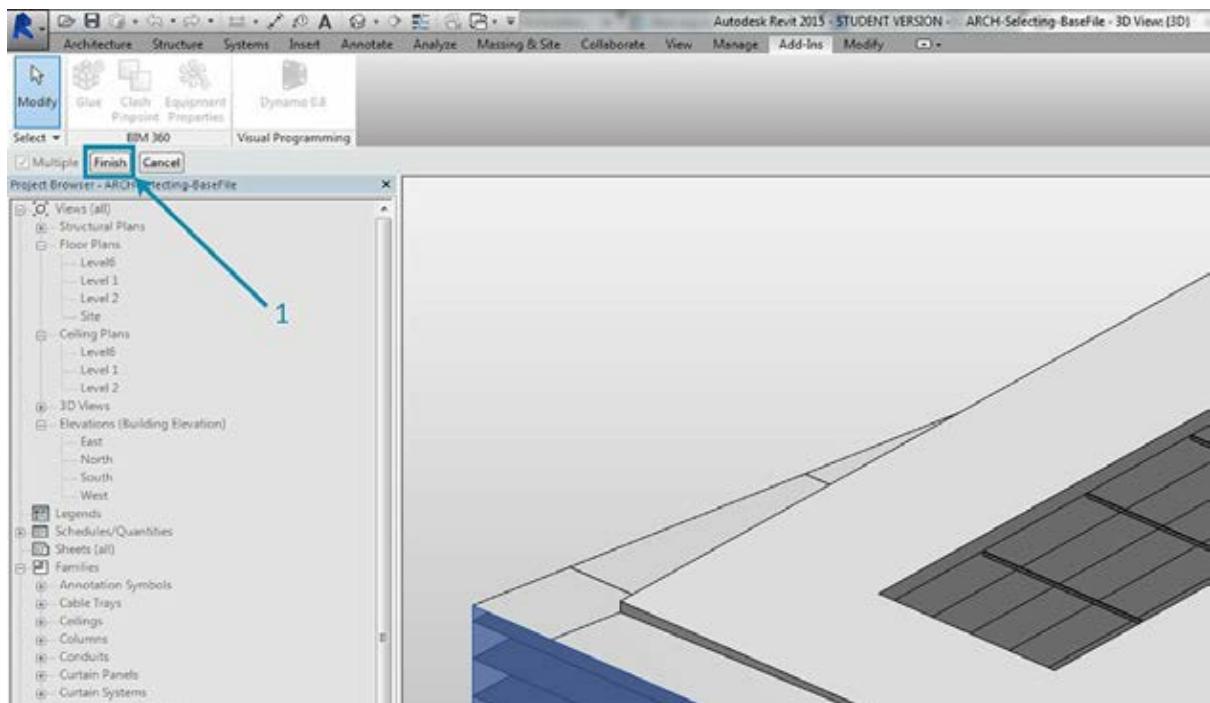


- Der letzte Schritt war etwas umständlich. Dasselbe Ergebnis erzielen Sie schneller mit dem *Select Face*-Block. Damit können Sie Flächen auswählen, die keine eigenständigen Elemente im Revit-Projekt sind. Dabei verwenden Sie dieselbe Interaktion wie bei *Select Model Element*, wobei Sie allerdings nicht das ganze

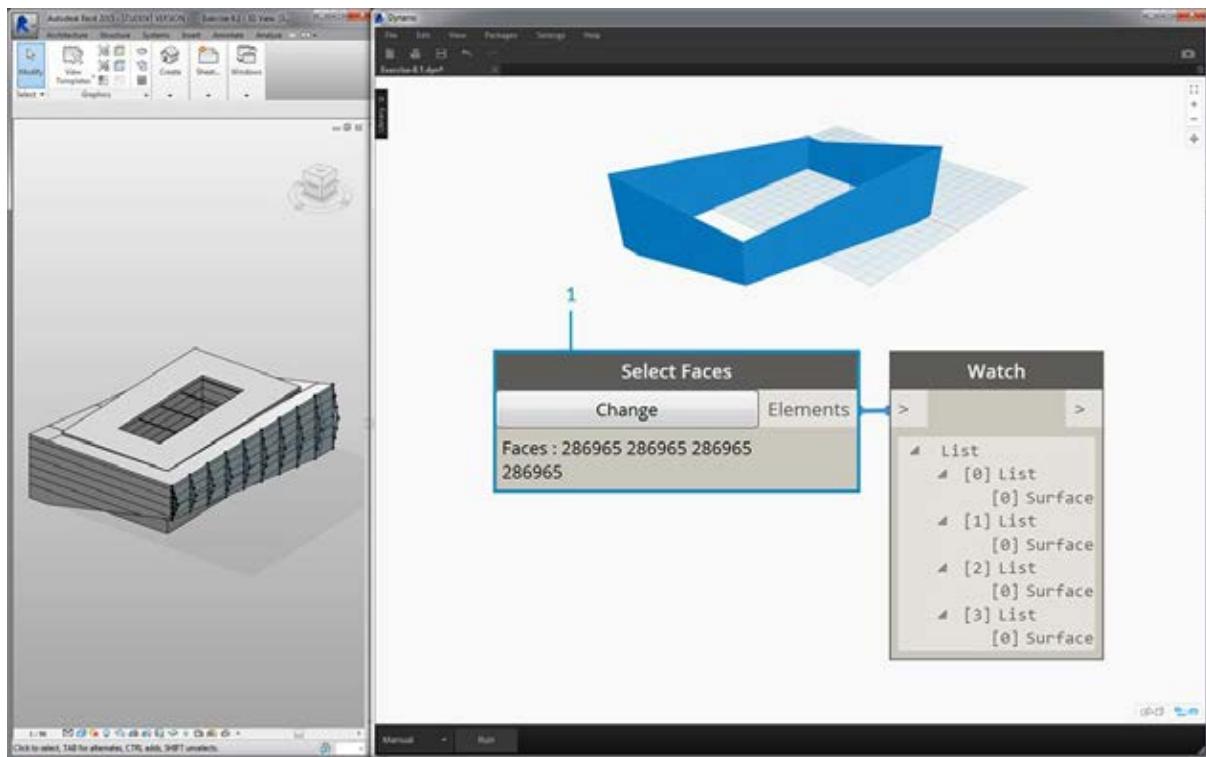
Element, sondern nur die Oberfläche auswählen.



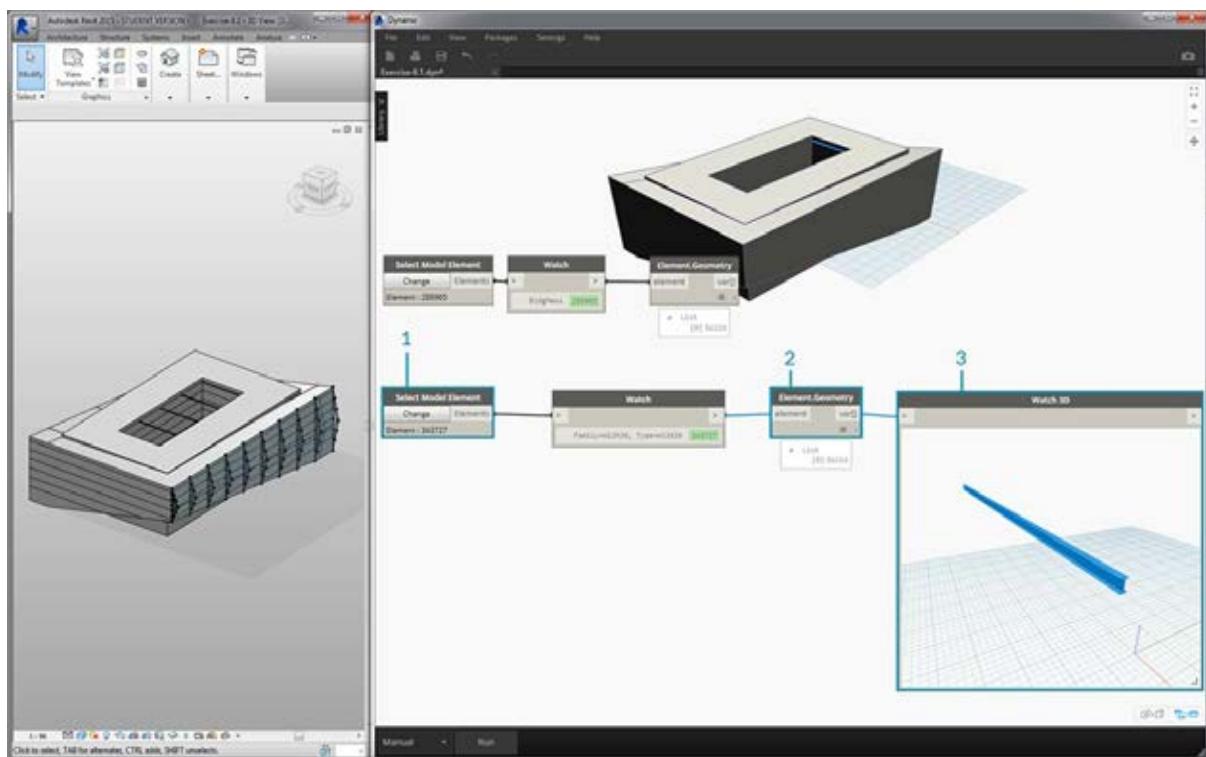
Angenommen, Sie möchten nur die Hauptfassaden des Gebäudes auswählen. Dies ist mithilfe des *Select Faces*-Blocks möglich. Klicken Sie auf die Schaltfläche "Select" und wählen Sie dann die vier Hauptfassaden in Revit aus.



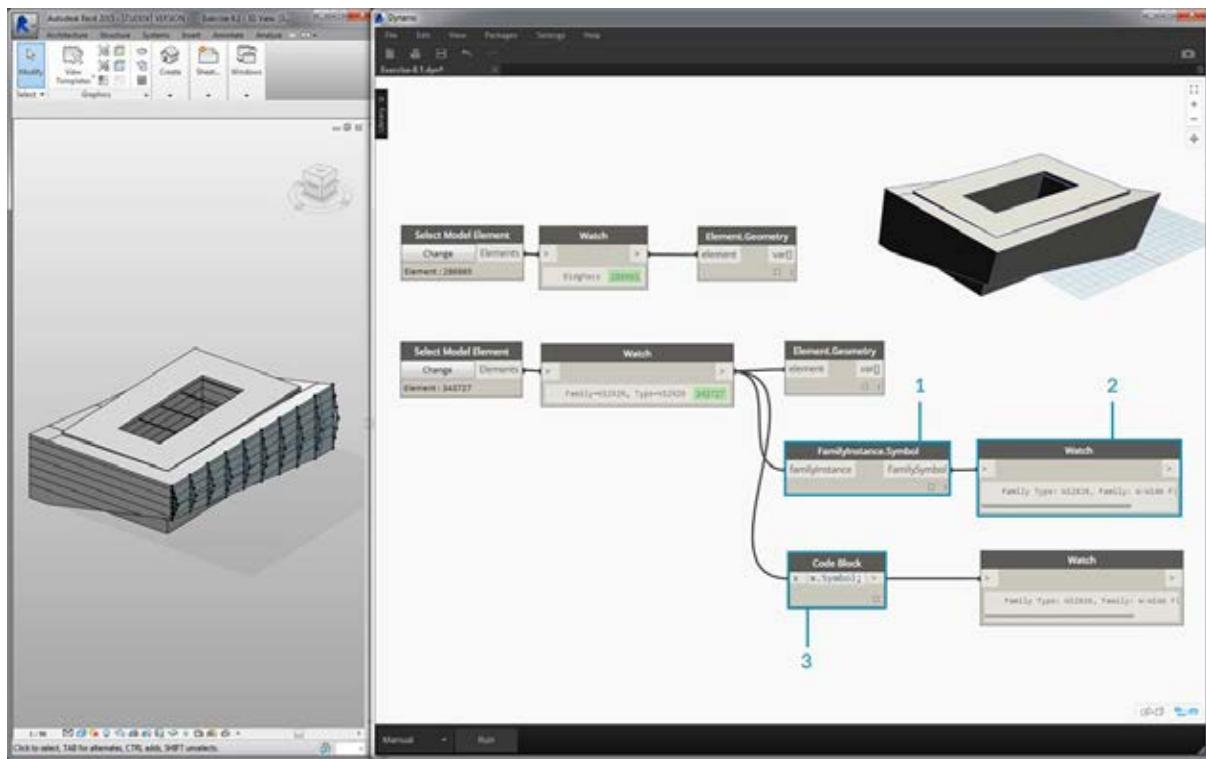
1. Achten Sie darauf, nach der Auswahl der vier Wände in Revit auf *Fertig stellen* zu klicken.



1. Damit werden die Flächen als Oberflächen in Dynamo importiert.



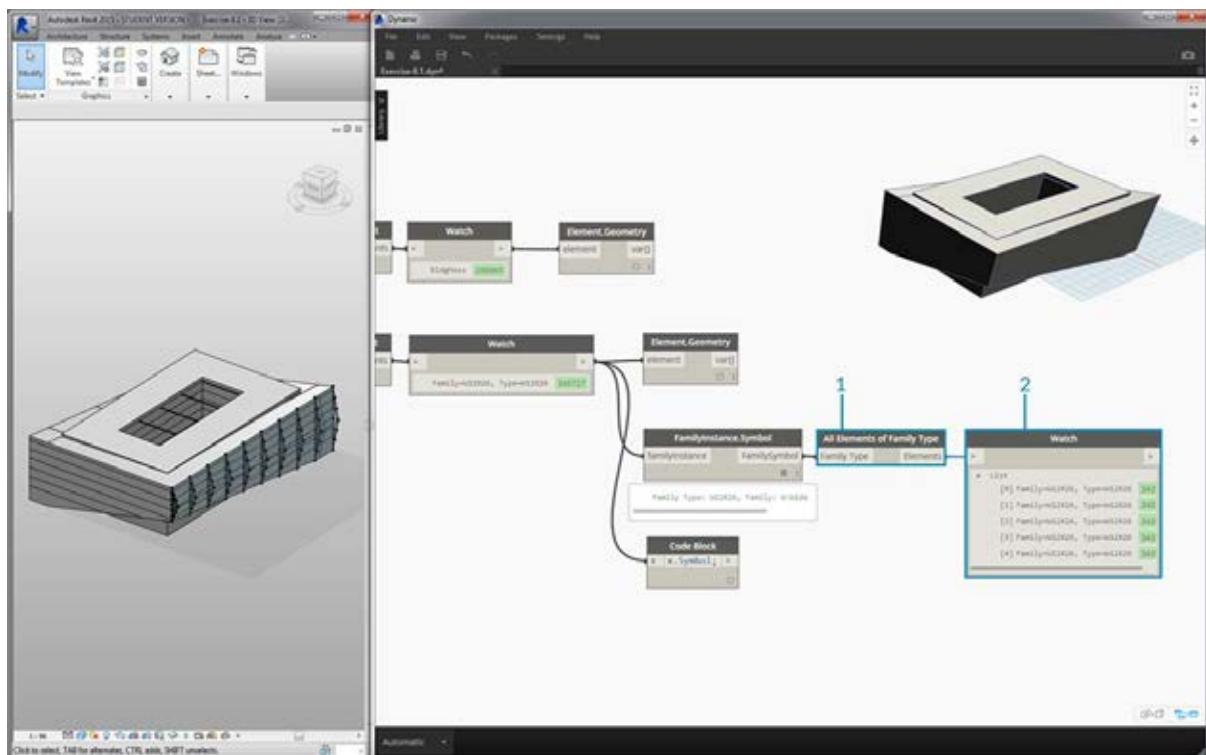
1. Als nächstes arbeiten Sie mit den Trägern über dem Foyer. Wählen Sie mithilfe des *Select Model Element*-Blocks einen der Träger aus.
2. Verbinden Sie das Trägerelement mit dem *Element.Geometry*-Block. Damit wird der Träger ins Dynamo-Ansichtsfenster übernommen.
3. In einem *Watch3D*-Block können Sie die Geometrie vergrößert anzeigen. (Falls der Träger dort nicht angezeigt wird, klicken Sie mit der rechten Maustaste und wählen Sie "Zoom anpassen".)



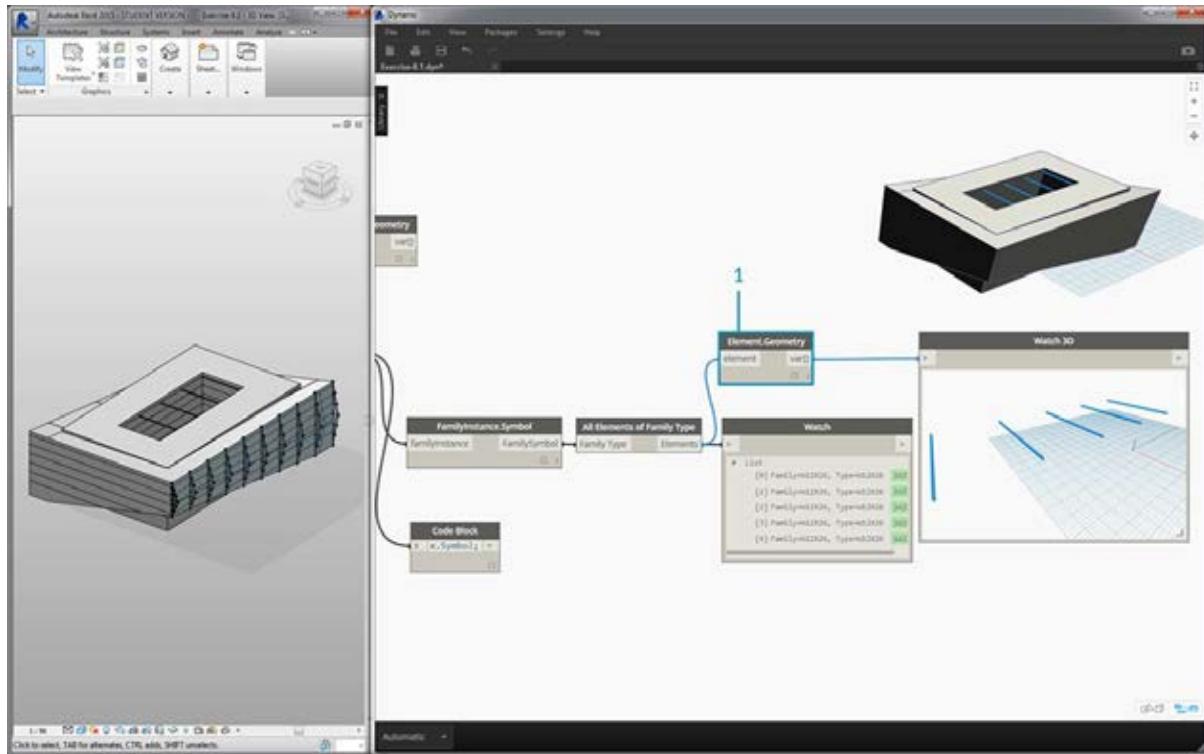
In Arbeitsabläufen mit Revit/Dynamo stellt sich häufig die Frage: Wie kann ich ein Element auswählen und alle weiteren ähnlichen Elemente abrufen? Da im ausgewählten Revit-Element sämtliche Informationen seiner Hierarchie enthalten sind, können Sie seinen Familientyp abrufen und dadurch alle Elemente dieses Typs auswählen.

1. Verbinden Sie das Trägerelement mit einem *FamilyInstance.Symbol*\*-Block.
2. Im Watch-Block ist zu erkennen, dass anstelle eines Revit-Elements jetzt ein Familiensymbol ausgegeben wird.
3. *FamilyInstance.Symbol* ist eine einfache Abfrage, d. h., Sie erzielen mithilfe eines Codeblocks mit der Eingabe `x.Symbol`; ebenso leicht dasselbe Ergebnis.

*\*Hinweis: Familiensymbol ist die in der Revit-API verwendete Bezeichnung für einen Familientyp. Um eventuelle Unklarheiten zu beseitigen, ist vorgesehen, dies in zukünftigen Versionen zu aktualisieren.*



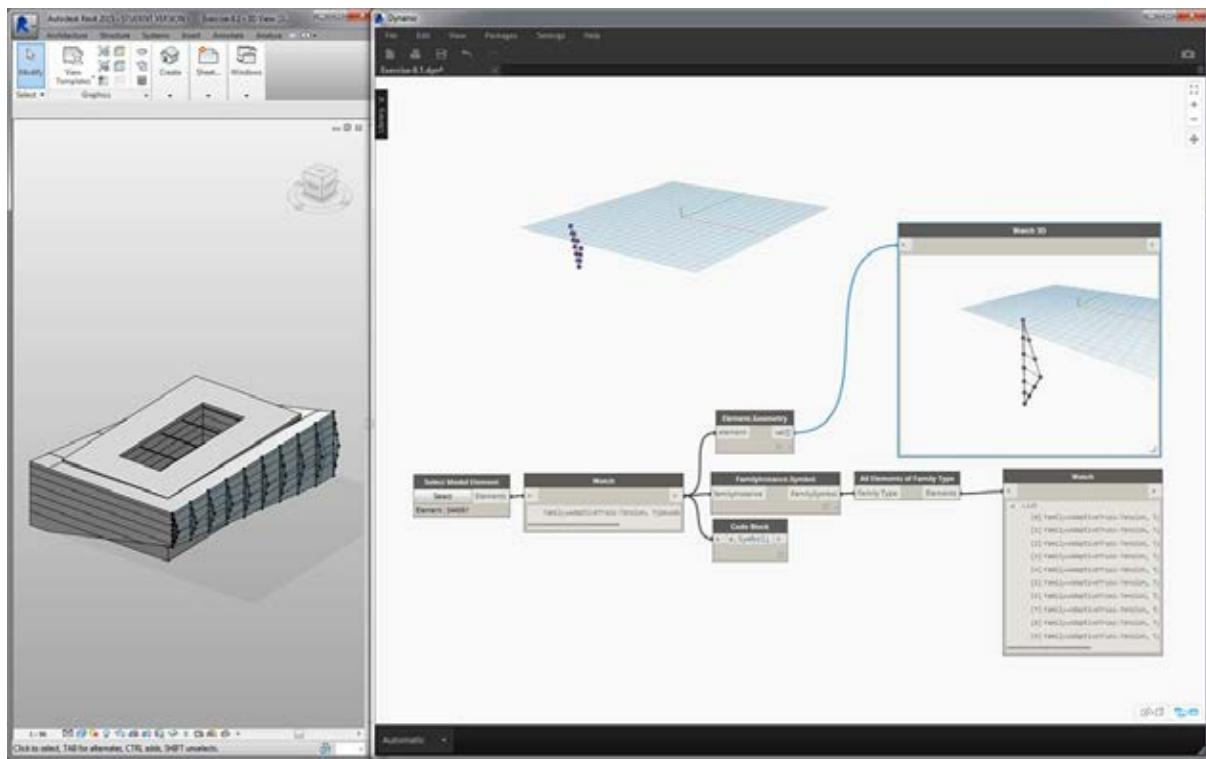
1. Verwenden Sie zum Auswählen der übrigen Träger den *All Elements of Family Type*-Block.
2. Im Watch-Block wird angezeigt, dass fünf Revit-Elemente ausgewählt sind.



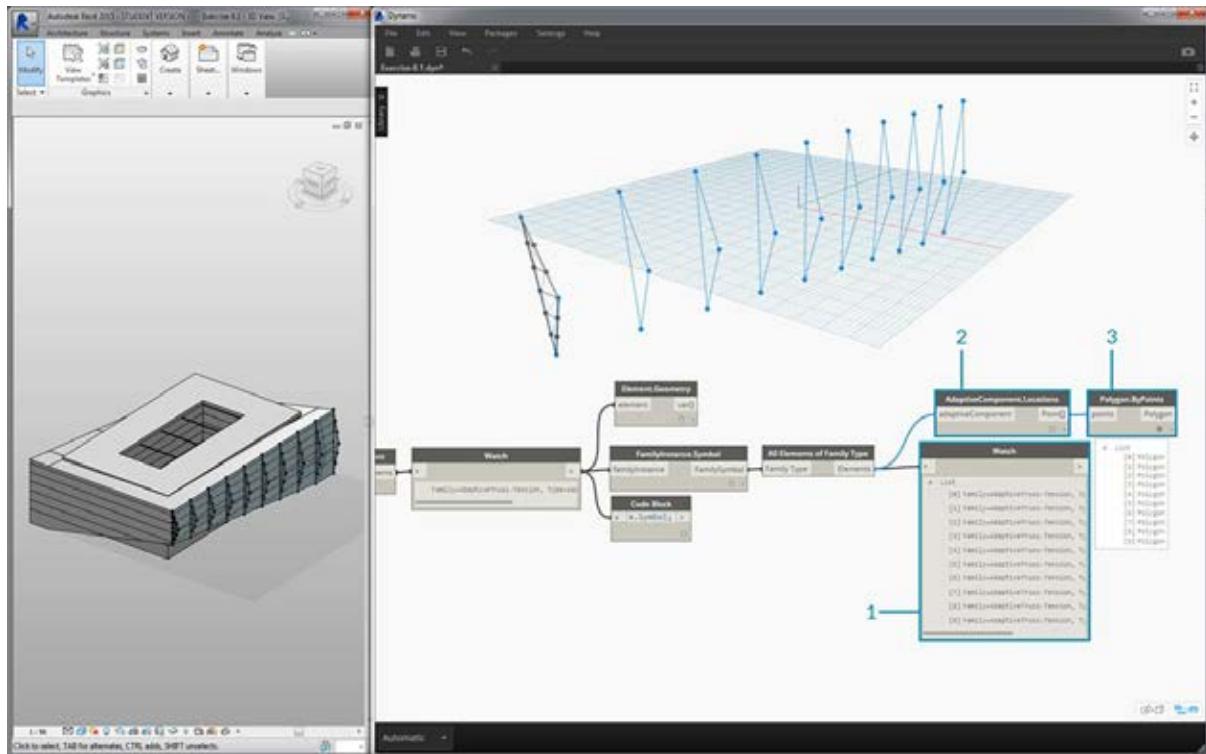
1. Alle diese fünf Elemente können ebenfalls in Dynamo-Geometrie umgewandelt werden.

Was würde geschehen, wenn 500 Träger vorhanden wären? Alle diese Elemente in Dynamo-Geometrie zu konvertieren, würde sehr viel Zeit in Anspruch nehmen. Falls die Berechnung von Blöcken in Dynamo sehr lange dauert, können Sie die Blockfunktionen anhalten ("einfrieren") und damit die Ausführung von Revit-Vorgängen unterbrechen, während Sie Ihr Diagramm entwickeln. Weitere Informationen zum Anhalten von Blöcken finden Sie im entsprechenden Abschnitt im Kapitel [Körper](#).

Angenommen, Sie möchten 500 Träger importieren: Benötigen Sie in diesem Fall sämtliche Oberflächen, um die beabsichtigte parametrische Operation durchzuführen? Oder können Sie grundlegende Informationen aus den Trägern extrahieren und generative Aufgaben mit Basisgeometrie durchführen? Diese Frage wird im weiteren Verlauf dieses Kapitels behandelt. Als Beispiel dient etwa das Fachwerksystem:



Wählen Sie mithilfe desselben Diagramms aus Blöcken nicht das Träger-, sondern das Fachwerkelement aus. Löschen Sie jedoch zuvor den Element.Geometry-Block aus dem vorigen Schritt.



1. Im *Watch*-Block wird eine Liste der in Revit ausgewählten adaptiven Bauteile angezeigt. Da grundlegende Informationen extrahiert werden sollen, beginnen Sie mit den adaptiven Punkten.
2. Verbinden Sie den *All Elements of Family Type*-Block mit dem *AdaptiveComponent.Location*-Block. Dadurch erhalten Sie eine Liste von Listen mit je drei Punkten für die Positionen der adaptiven Punkte.
3. Indem Sie einen *Polygon.ByPoints*-Block verbinden, erhalten Sie eine Polykurve. Dies ist im *Dynamo*-Ansichtsfenster zu erkennen. Mithilfe dieses Verfahrens haben Sie die Geometrie eines Elements visualisiert und die Geometrie der verbleibenden Reihen von Elementen abstrahiert (wobei mehr Elemente als in diesem Beispiel vorhanden sein könnten).

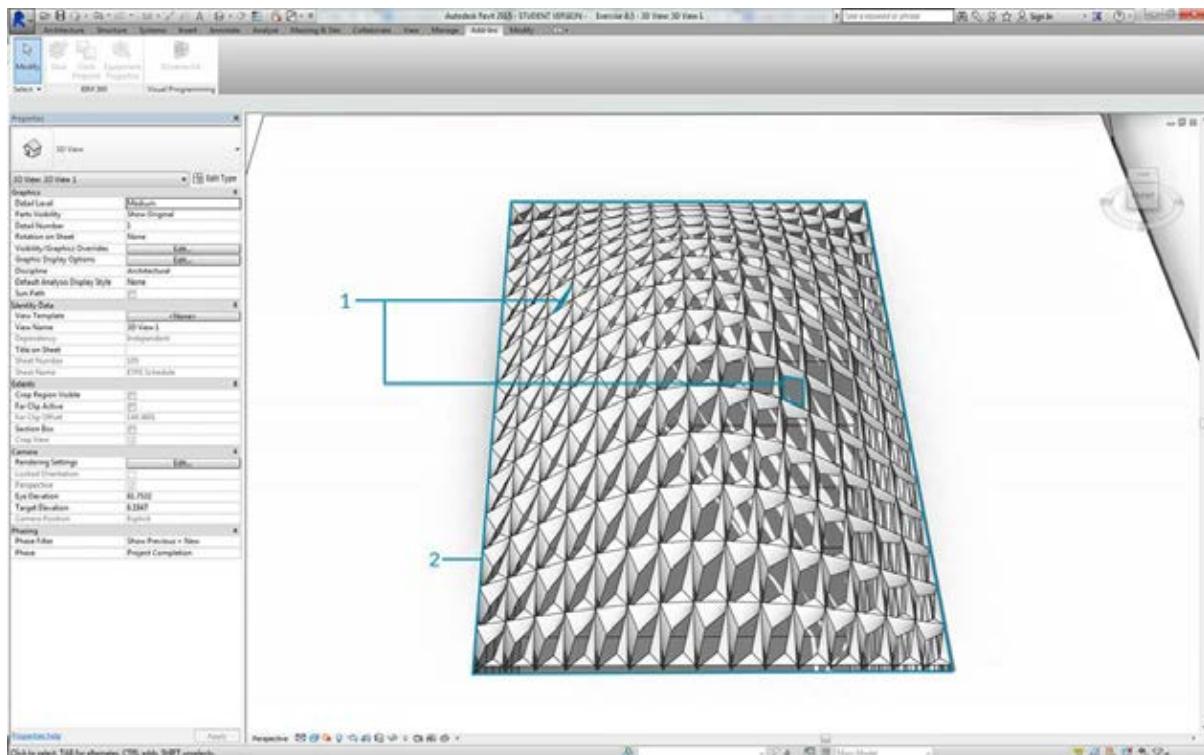
*\*Tipp: Wenn Sie in Dynamo auf die grün unterlegte Nummer eines Revit-Elements klicken, wird im Revit-Ansichtsfenster auf dieses Element eingezoomt.*

# Bearbeiten

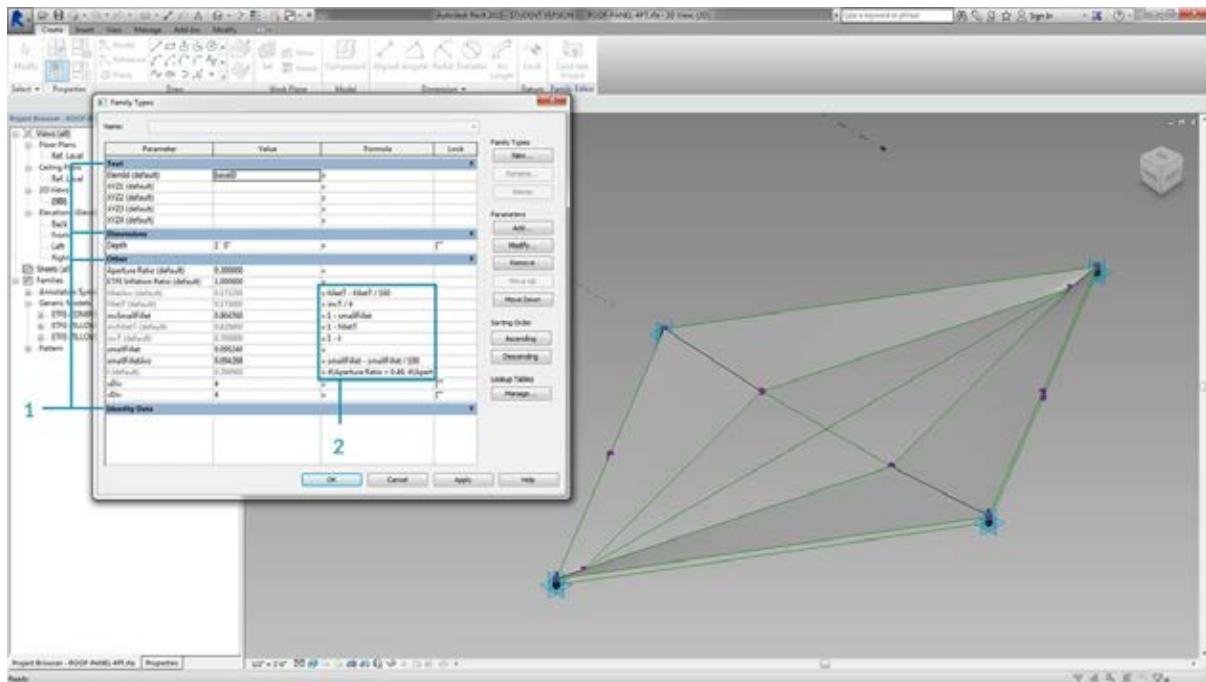
## Bearbeiten

Zu den leistungsstarken Funktionen von Dynamo gehört die Möglichkeit zum Bearbeiten von Parametern auf parametrischer Ebene. Sie können beispielsweise die Parameter eines Arrays aus Elementen mithilfe eines generativen Algorithmus oder der Ergebnisse einer Simulation steuern. Auf diese Weise können Sie einer Gruppe von Exemplaren aus derselben Familie benutzerdefinierte Eigenschaften im Revit-Projekt zuweisen.

### Typen- und Exemplarparameter



1. Exemplarparameter definieren die Öffnung der Elemente in der Dachoberfläche mit einem Öffnungsanteil zwischen 0.1 und 0.4.
2. Typenparameter werden auf sämtliche Elemente der Oberfläche angewendet, da diese zum selben Familientyp gehören. So kann beispielsweise das Material der einzelnen Elemente durch einen Typenparameter gesteuert werden.



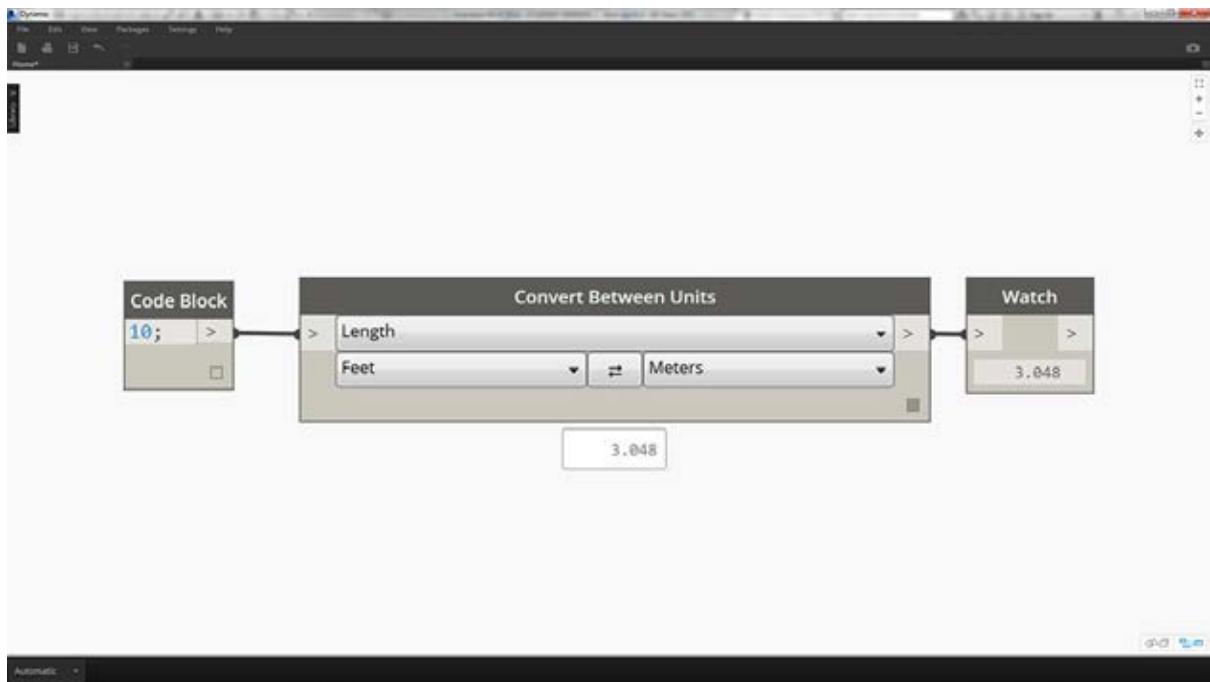
1. Wenn Sie zuvor schon Revit-Familien eingerichtet haben, beachten Sie, dass Sie einen Parametertyp (Zeichenfolge, Zahl, Bemaßung usw.) zuweisen müssen. Achten Sie darauf, beim Zuweisen von Parametern in Dynamo den richtigen Datentyp zu verwenden.
2. Sie können Dynamo auch zusammen mit parametrischen Abhängigkeiten verwenden, die in den Eigenschaften einer Revit-Familie festgelegt wurden.

Rufen Sie sich zunächst ins Gedächtnis zurück, dass in Revit Typen- und Exemplarparameter verwendet werden. Beide können in Dynamo bearbeitet werden. In der folgenden Übung verwenden Sie jedoch Exemplarparameter.

Anmerkung: Während Sie das breite Spektrum der Verwendungsmöglichkeiten für die Parameterbearbeitung kennenlernen, müssen Sie in manchen Fällen eventuell sehr zahlreiche Revit-Elemente mit Dynamo bearbeiten. Solche Vorgänge können sehr *rechenintensiv* sein und laufen deshalb eventuell nur langsam ab. Bei der Bearbeitung zahlreicher Elemente kann es daher sinnvoll sein, die Ausführung von Revit-Vorgängen mithilfe der Funktion Anhalten vorübergehend zu unterbrechen, während Sie das Diagramm entwickeln. Weitere Informationen zum Anhalten von Blöcken finden Sie im entsprechenden Abschnitt im Kapitel [Körper](#).

## Einheiten

In Dynamo werden ab Version 0.8 grundsätzlich keine Einheiten verwendet. Dadurch bleibt Dynamo als abstrakte visuelle Programmierumgebung erhalten. Dynamo-Blöcke, die mit Revit-Bemaßungen interagieren, referenzieren die Einheiten aus dem Revit-Projekt. Wenn Sie beispielsweise einen in Revit einen Längenparameter aus Dynamo festlegen, entspricht dessen Zahlenwert in Dynamo den Vorgabeeinheiten im Revit-Projekt. Für die unten stehende Übung werden Meter verwendet.



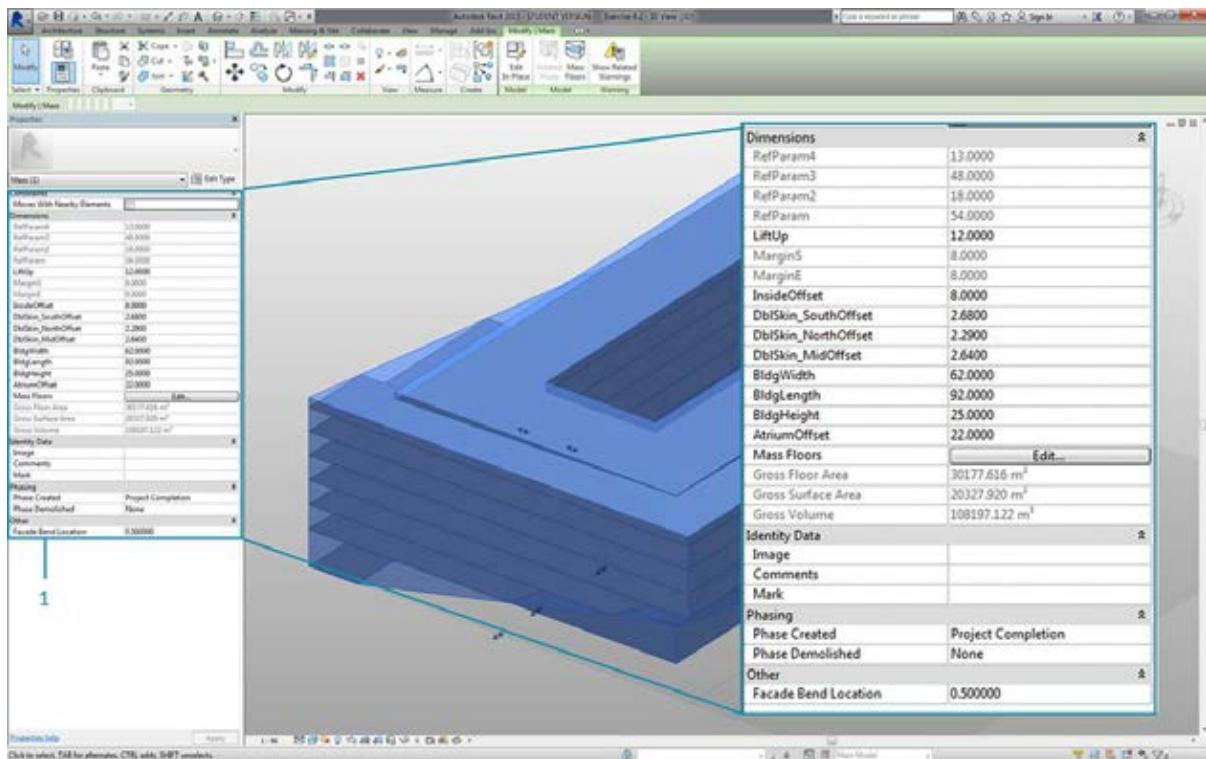
Verwenden Sie zur schnellen Konvertierung von Einheiten den Block *Convert Between Units*. Dies ist ein sehr hilfreiches Werkzeug zum Konvertieren von Längen-, Flächen- und Volumeneinheiten nach Bedarf.

## Übungslektion

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

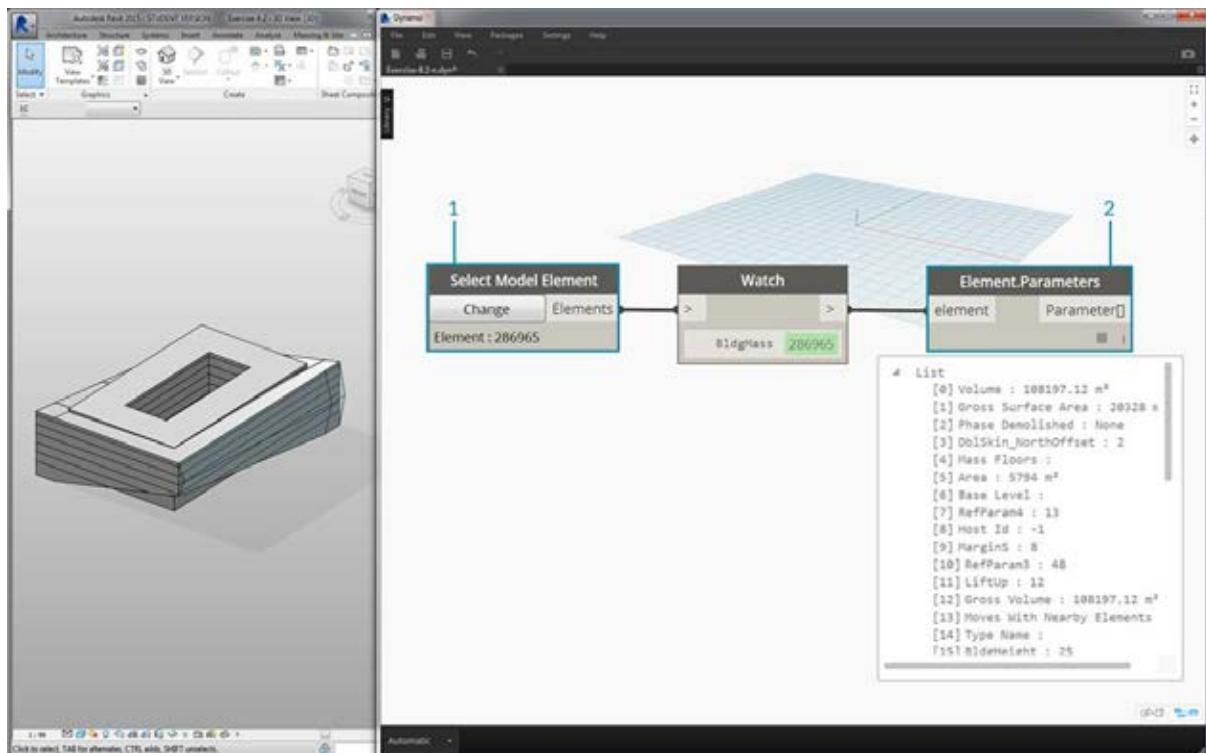
1. [Editing.dyn](#)
2. [ARCH-Editing-BaseFile.rvt](#)

In dieser Übung bearbeiten Sie Revit-Elemente, ohne geometrische Operationen in Dynamo auszuführen. In diesem Fall importieren Sie keine Dynamo-Geometrie, sondern bearbeiten lediglich Parameter in einem Revit-Projekt. Dies ist eine Übung zu Grundlagen. Benutzer mit fortgeschrittenen Revit-Kenntnissen sollten beachten, dass hier zwar die Exemplarparameter eines Körpers behandelt werden, mithilfe derselben Logik jedoch auch Arrays von Elementen umfassend angepasst werden können. Für diesen Vorgang wird der Element.SetParameterByName-Block verwendet.

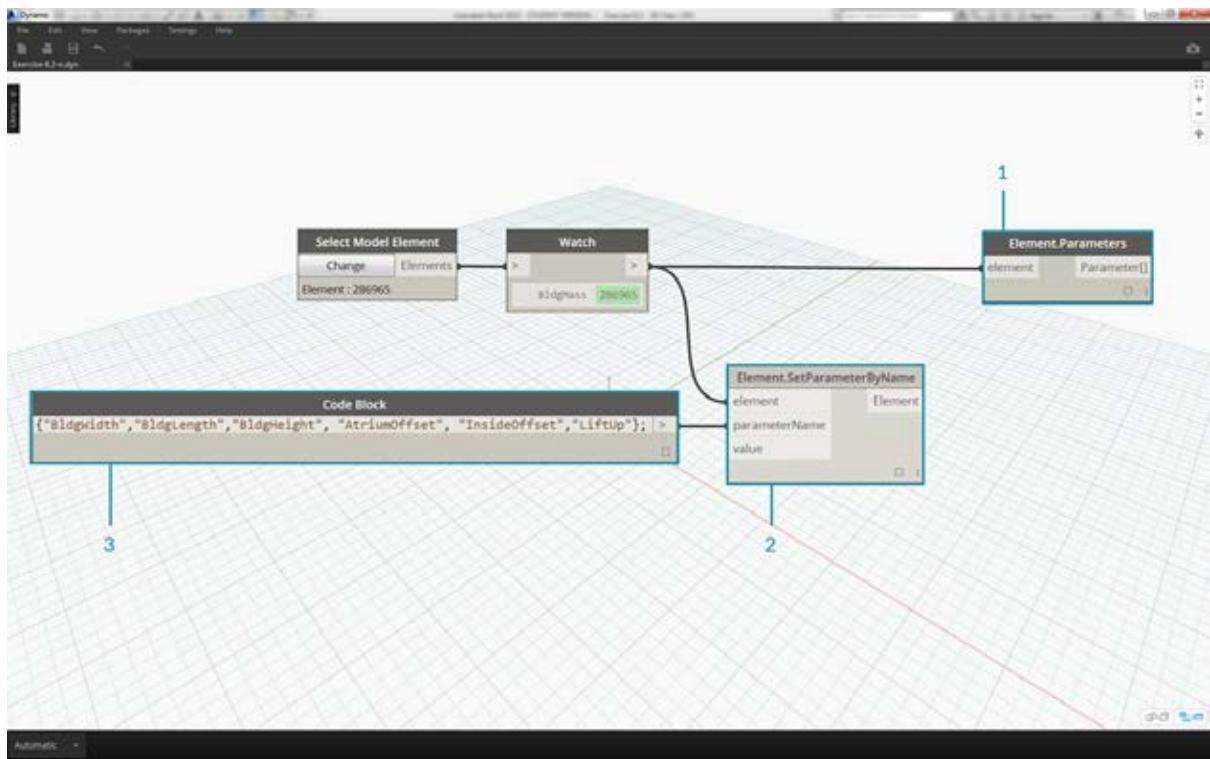


Beginnen Sie mit der Revit-Beispieldatei für diesen Abschnitt. Die Trägerelemente und adaptiven Fachwerkbinden aus dem vorigen Abschnitt wurden entfernt. Thema dieser Übung ist ein parametrisches Gerüst in Revit und seine Bearbeitung in Dynamo.

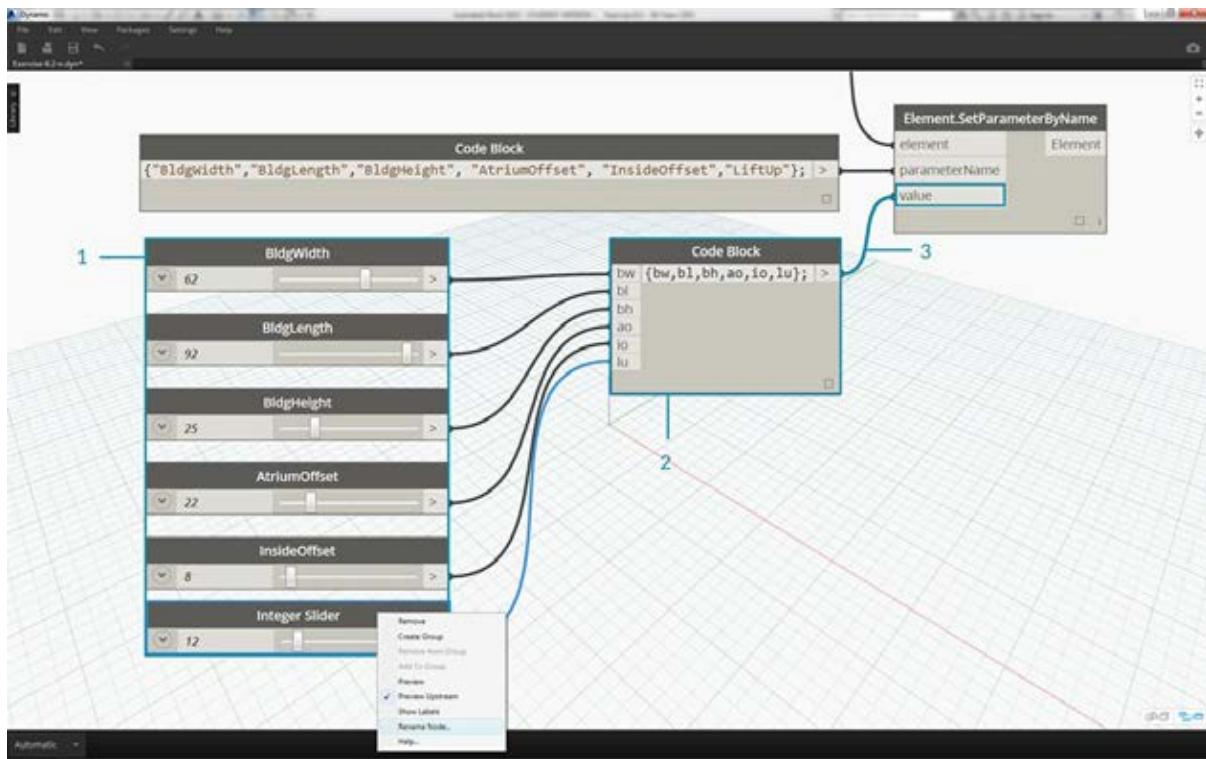
1. Wenn Sie das Gebäude in Revit unter Körper auswählen, wird in der Eigenschaftenpalette eine Reihe von Exemplarparametern angezeigt.



1. Wählen Sie den Gebäudekörper mithilfe des *Select Model Element*-Blocks aus.
2. Sie können sämtliche Parameter dieses Körpers mithilfe des *Element.Parameters*-Blocks abrufen. Dazu gehören Typen- und Exemplarparameter.

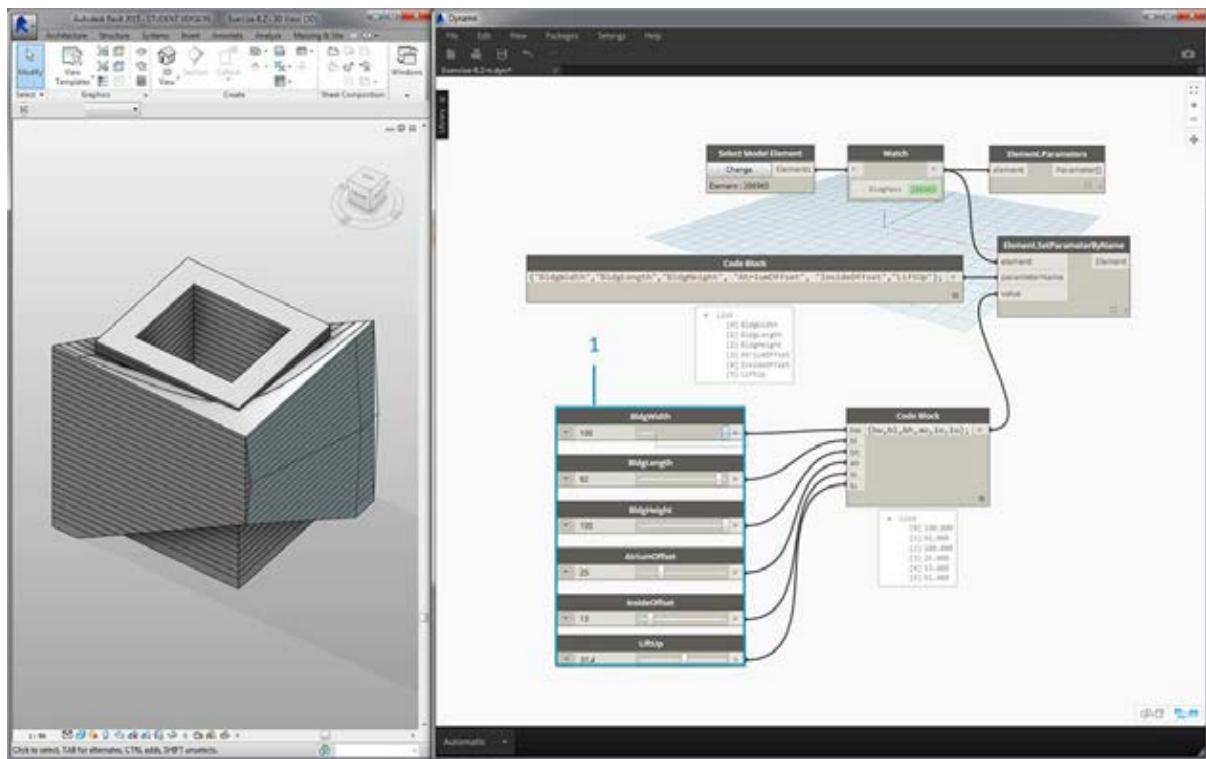


1. Im *Element.Parameters*-Block können Sie die Zielparameter suchen. Sie können stattdessen auch die Eigenschaftenpalette aus dem vorigen Schritt anzeigen, um die Namen der zu bearbeitenden Parameter zu wählen. In diesem Fall suchen Sie nach den Parametern, die sich auf die großräumigen geometrischen Veränderungen des Gebäudekörpers auswirken.
2. Sie nehmen mithilfe des *Element.SetParameterByName*-Blocks Änderungen am Revit-Element vor.
3. Definieren Sie in einem *Code Block* eine Liste der benötigten Parameter, jeweils in Anführungszeichen eingeschlossen, um sie als Zeichenfolgen zu kennzeichnen. Dies ist auch möglich, indem Sie mehrere Eingaben eines *List.Create*-Blocks jeweils mit *String*-Blöcken verbinden. Mithilfe des Codeblocks erreichen Sie dies jedoch schneller und einfacher. Achten Sie darauf, dass jede Zeichenfolge genau dem Namen in Revit entspricht, wobei die Groß- und Kleinschreibung beachtet werden muss:  
`{"BldgWidth", "BldgLength", "BldgHeight", "AtriumOffset", "InsideOffset", "LiftUp"};`



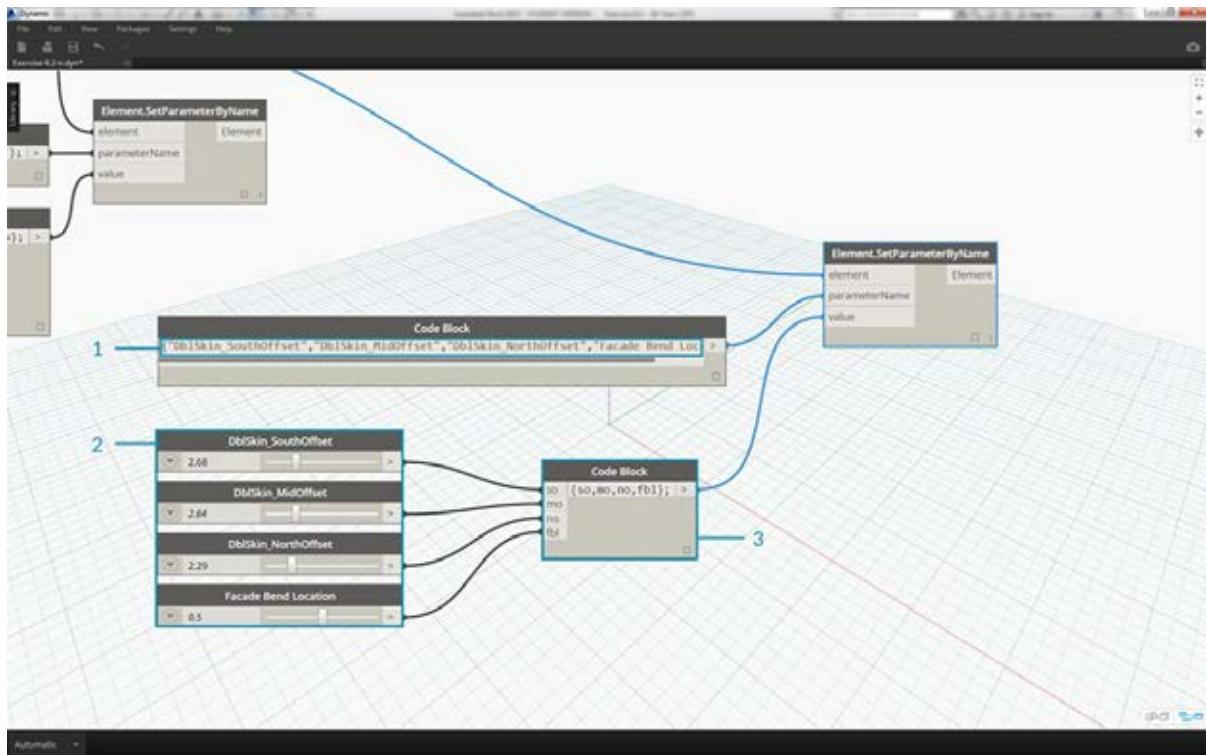
1. Darüber hinaus müssen Sie Werte für die einzelnen Parameter festlegen. Fügen Sie sechs *Integer Slider*-Blöcke in den Ansichtsbereich ein und weisen Sie ihnen die Namen der entsprechenden Parameter in der Liste zu. Legen Sie außerdem in den einzelnen Schiebereglern die in der Abbildung oben gezeigten Werte fest. Dies sind die folgenden Werte (von oben nach unten: 62, 92, 25, 22, 8, 12).
2. Definieren Sie einen weiteren *Code Block* mit einer Liste von derselben Länge wie die Liste der Parameternamen. In diesem Fall geben Sie dabei Variablennamen (ohne Anführungszeichen) an und erhalten dadurch Eingaben für den *Code Block*. Verbinden Sie die *Schiebereglern* mit den dazugehörigen Eingaben: {bw, bl, bh, ao, io, lu} ;
3. Verbinden Sie den *Code Block* mit dem *Element.SetParameterByName*-Block. Ist die Option Automatisch ausführen aktiviert, werden die Ergebnisse sofort angezeigt.

*\*Anmerkung: Diese Demonstration kann nur mit Exemplarparametern, nicht jedoch mit Typenparametern durchgeführt werden.*



Viele dieser Parameter sind genau wie in Revit voneinander abhängig. Dabei können manche Kombinationen selbstverständlich zu ungültiger Geometrie führen. Dieses Problem können Sie mithilfe definierter Formeln in den Parametereigenschaften beheben. Sie können stattdessen auch eine ähnliche Logik mit mathematischen Operationen in Dynamo einrichten. (Sie könnten dies als Zusatzübung ausprobieren.)

1. Mit der folgenden Kombination erhalten Sie ein recht originelles Design für den Gebäudekörper: 100, 92, 100, 25, 13, 51.4.

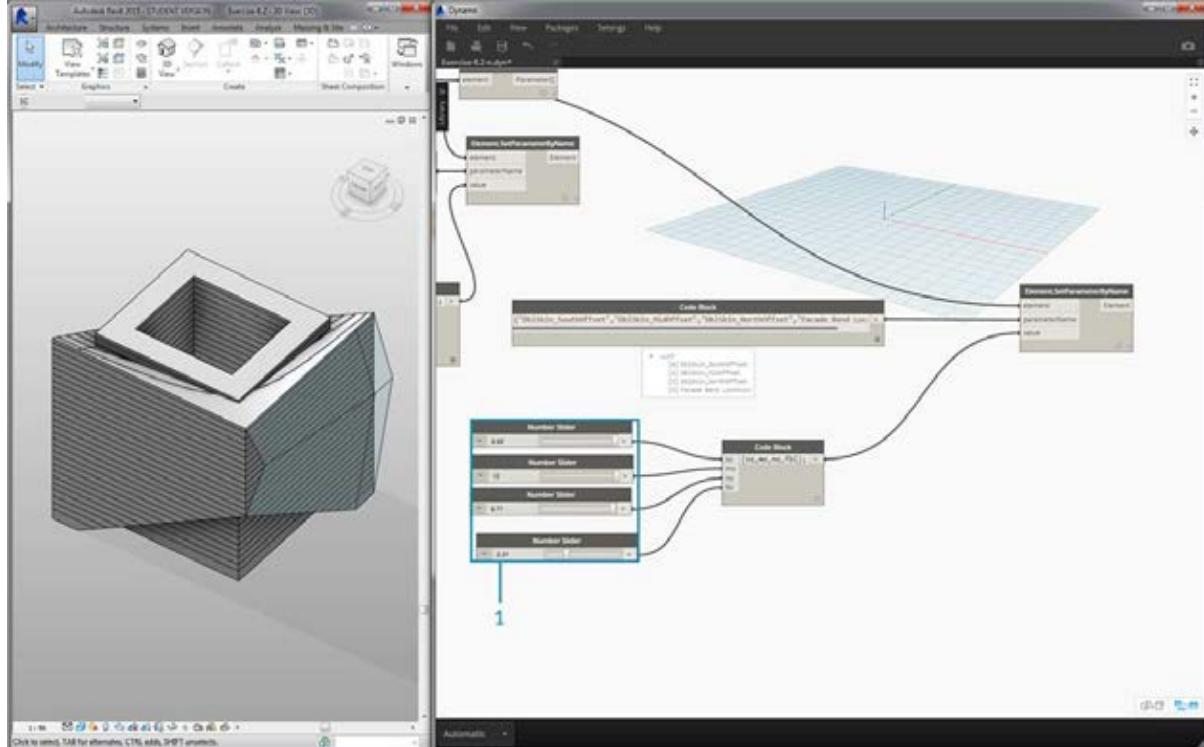


1. Kopieren Sie für diese Aufgabe das Diagramm, um mit der Fassadenverglasung zu arbeiten, an der das Fachwerksystem angebracht werden soll. In diesem Fall isolieren Sie vier Parameter: {"DblSkin\_SouthOffset", "DblSkin\_MidOffset", "DblSkin\_NorthOffset", "Facade

```
Bend Location"};.
```

2. Erstellen Sie darüber hinaus *Number Slider*-Blöcke und ändern Sie ihre Namen in die der entsprechenden Parameter. Weisen Sie den ersten drei Schiebereglern (von oben nach unten) die Domäne [0,10], dem letzten Schieberegler *Facade Bend Location* hingegen die Domäne [0,1] zu. Diese Werte sollten von oben nach unten, mit den folgenden Angaben beginnen (wobei diese hier beliebig gewählt wurden): 2.68, 2.64, 2.29, 0.5.

3. Definieren Sie einen neuen *Code Block* und verbinden Sie die Schieberegler: {so, mo, no, fbl};

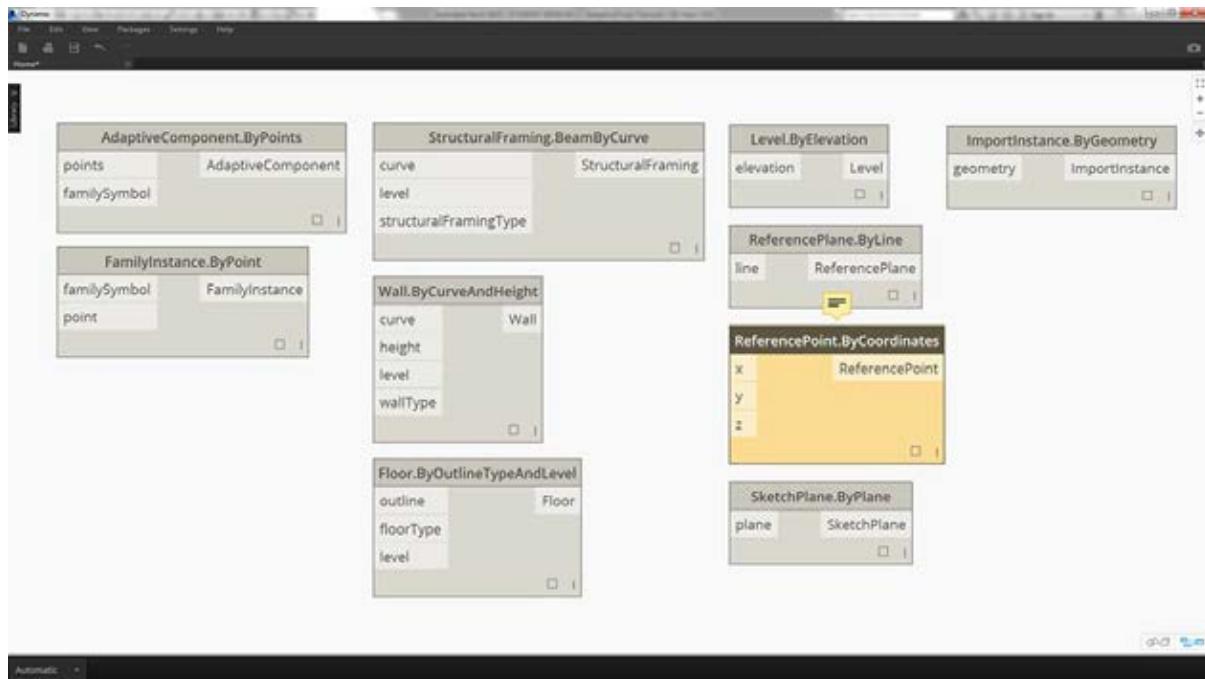


1. Indem Sie die Werte der *Schiebereglер* für diesen Teil des Diagramms ändern, können Sie die Fassadenverglasung erheblich verstärken: 9.98, 10.0, 9.71, 0.31.

# Erstellen

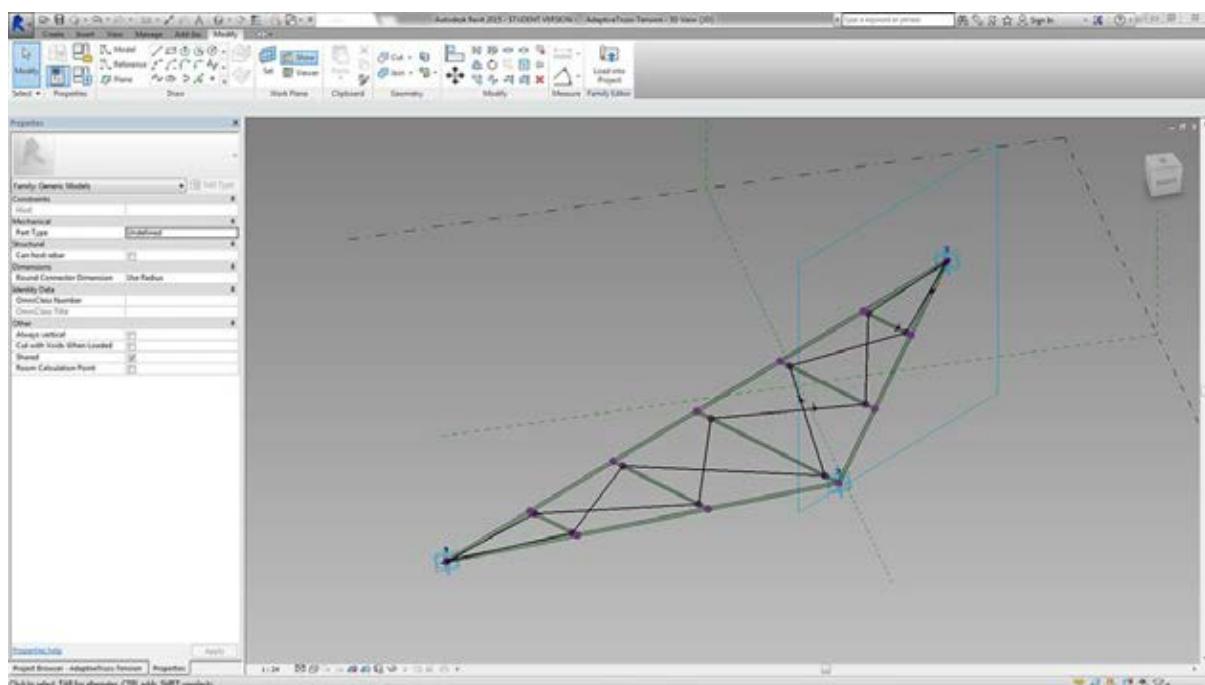
## Erstellen

Sie können in Dynamo Arrays von Revit-Elementen mit uneingeschränkter parametrischer Steuerung erstellen. Die Revit-Blöcke in Dynamo ermöglichen das Importieren von Elementen aus allgemeiner Geometrie in spezifische Kategorietypen (z. B. Wände und Geschossdecken). In diesem Abschnitt importieren Sie Elemente mit flexiblen Parametern für adaptive Bauteile.



## Adaptive Bauteile

Ein adaptives Bauteil ist eine flexible Familienkategorie, die sich für generative Anwendungen eignet. Bei der Instanziierung können Sie ein komplexes geometrisches Element erstellen, das durch die Positionen adaptiver Punkte gesteuert wird.



Dieses Beispiel zeigt ein adaptives Bauteil mit drei Punkten im Familieneditor. Es dient zum Erstellen eines durch die Positionen der einzelnen adaptiven Punkte gesteuerten Fachwerkbinders. In der folgenden Übung erstellen Sie mithilfe dieses Bauteils eine Reihe von Fachwerkbindern an einer Fassade.

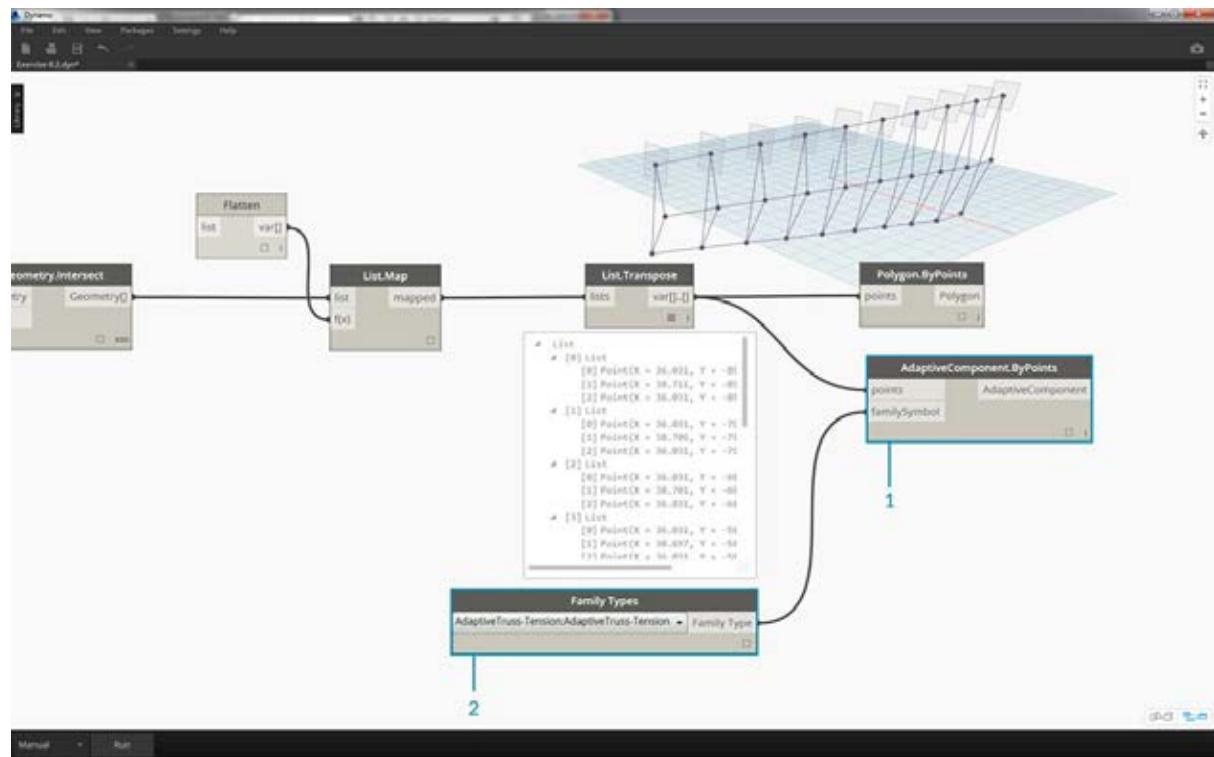
### Prinzipien der Interoperabilität

Adaptive Bauteile bieten ein gutes Beispiel für optimale Verfahren zur Interoperabilität. Sie können eine Reihe adaptiver Bauteile erstellen, indem Sie die zugrunde liegenden adaptiven Punkte definieren. Wenn Sie diese Daten in andere Programme übertragen, haben Sie die Möglichkeit, die Geometrie auf einfache Daten zu reduzieren. Dabei liegt eine ähnliche Logik zugrunde wie beim Importieren und Exportieren in Programmen wie Excel.

Angenommen, ein Berater für die Fassaden benötigt Angaben zur Position der Fachwerkelemente, möchte jedoch nicht die vollständig ausgearbeitete Geometrie analysieren. Zur Vorbereitung der Fertigung kann der Berater die Position der adaptiven Punkte ermitteln und daraus Geometrie für ein Programm wie Inventor neu generieren.

In der folgenden Übung richten Sie einen Arbeitsablauf ein, der Ihnen Zugriff auf alle diese Daten gibt und zugleich die Definition zum Erstellen von Revit-Elementen festlegt. In diesem Prozess führen Sie Konzeption, Dokumentation und Fertigung zu einem nahtlosen Arbeitsablauf zusammen. Damit erhalten Sie einen intelligenteren und effizienteren Prozess zur Interoperabilität.

### Mehrere Elemente und Listen

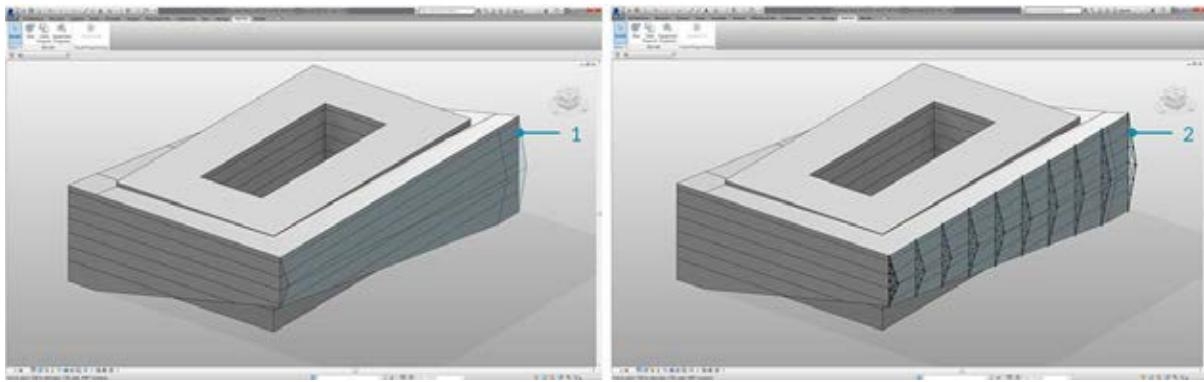


In der unten stehenden Übung wird gezeigt, wie Dynamo Daten für die Erstellung von Revit-Elementen referenziert. Um mehrere adaptive Bauteile zu generieren, definieren Sie eine Liste von Listen, wobei jede Liste drei Punkte enthält, die für die Punkte im adaptiven Bauteil stehen. Beachten Sie dies, während die Datenstrukturen in Dynamo verwaltet werden.

### Übungslektion

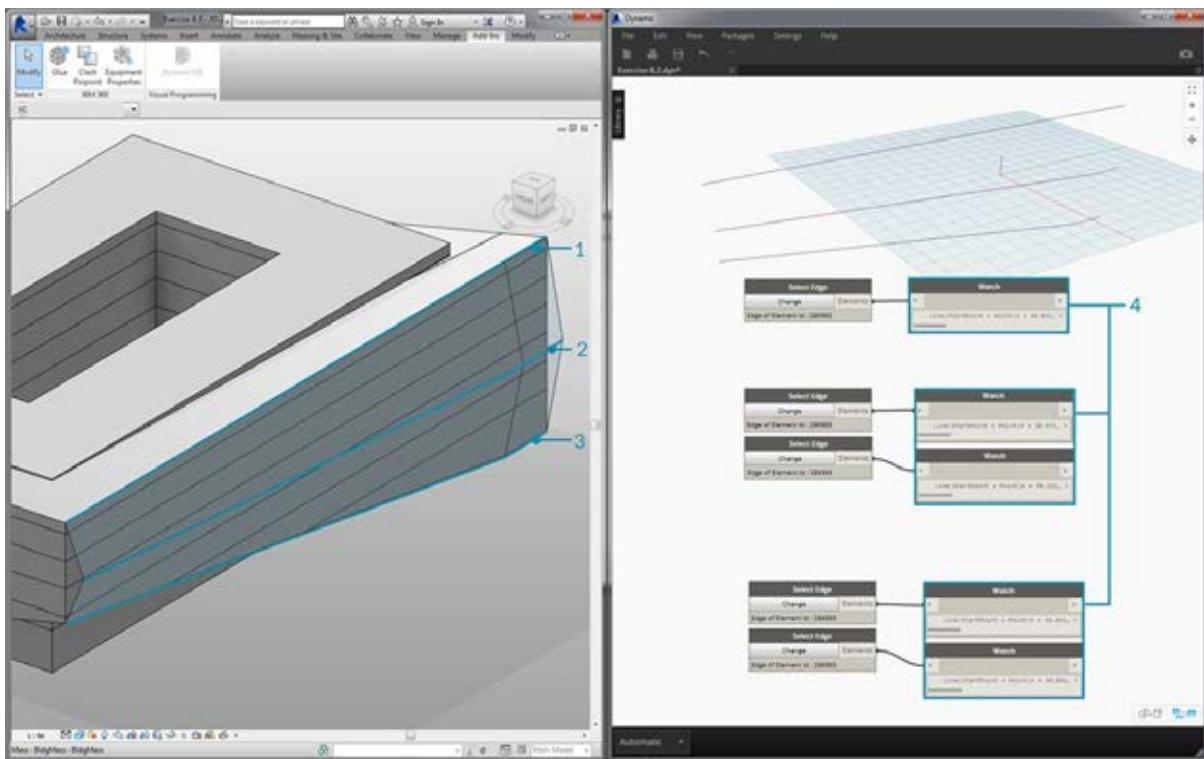
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

1. [Creating.dyn](#)
2. [ARCH-Creating-BaseFile.rvt](#)



Beginnen Sie mit der Beispieldatei für diesen Abschnitt (oder verwenden Sie weiterhin die Revit-Datei aus dem vorigen Abschnitt). Derselbe Revit-Körper wird angezeigt.

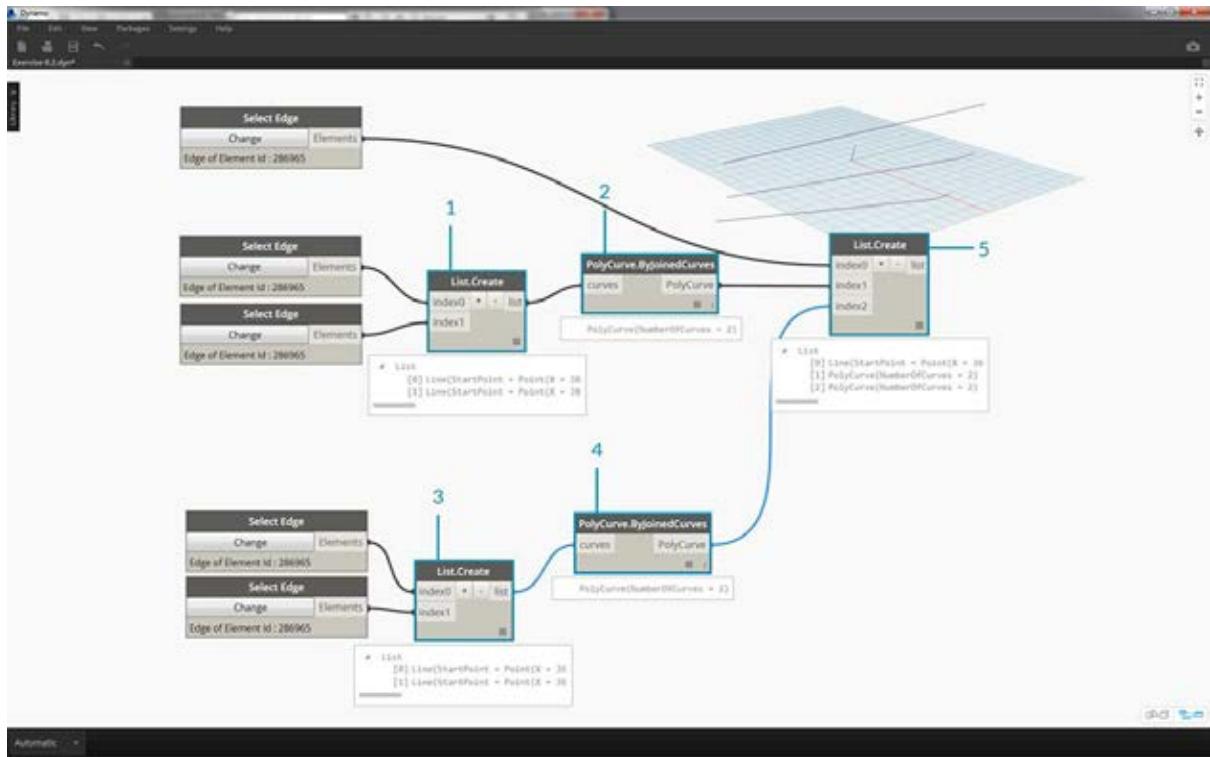
1. Dies ist der Zustand beim Öffnen der Datei.
2. Dies ist das Fachwerksystem, das Sie mit Dynamo erstellt haben, wobei eine intelligente Verknüpfung mit dem Revit-Körper genutzt wird.



Sie haben bereits mit den Blöcken *Select Model Element* und *Select Face* gearbeitet. Hier verwenden Sie *Select Edge* und arbeiten damit eine Ebene tiefer in der Geometriehierarchie. Stellen Sie für die Ausführung des Dynamo-Solver die Option *Automatisch* ein, damit das Diagramm laufend gemäß den Änderungen in der Revit-Datei aktualisiert wird. Die Kante, die Sie auswählen, ist dynamisch mit der Topologie des Revit-Elements verbunden. Solange die Topologie\* unverändert bleibt, bleibt die Verknüpfung zwischen Revit und Dynamo erhalten.

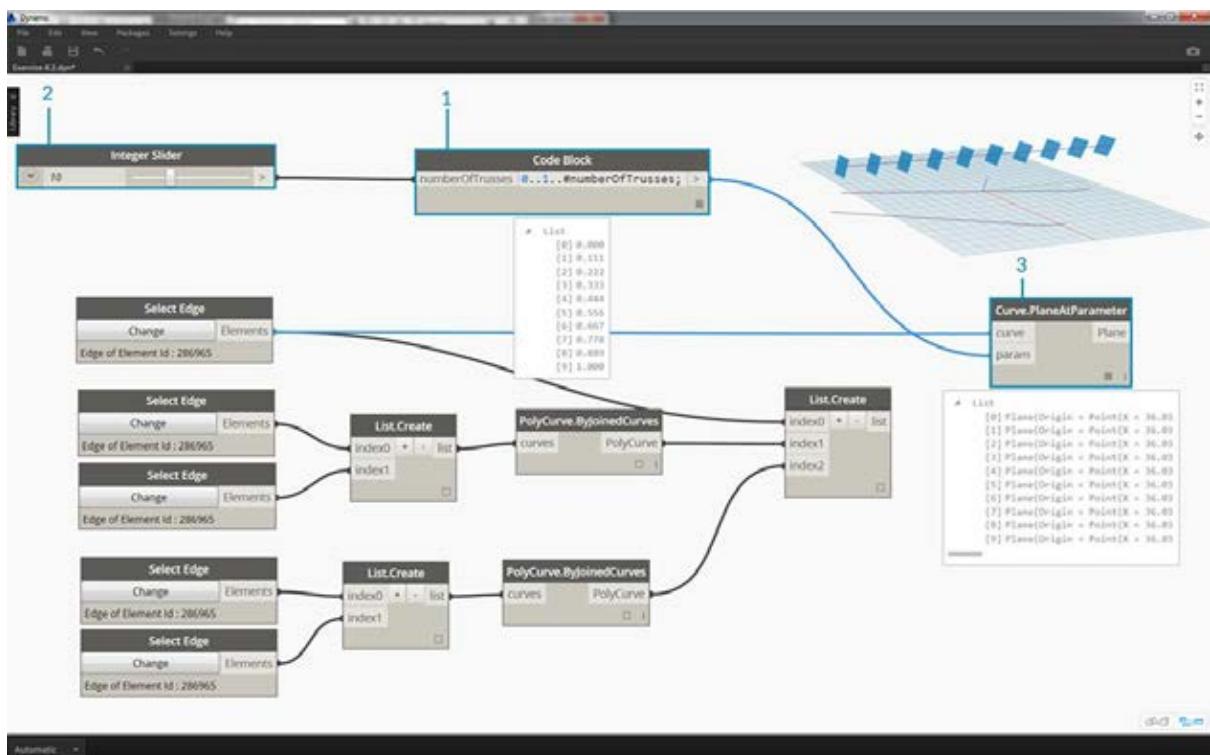
1. Wählen Sie die oberste Kurve der verglasten Fassade aus. Diese erstreckt sich über die gesamte Länge des Gebäudes. Falls Sie bei der Auswahl der Kante Schwierigkeiten haben, können Sie in Revit den Mauszeiger auf die Kante setzen und die *Tabulatortaste* drücken, bis die gewünschte Kante hervorgehoben wird.
2. Wählen Sie über zwei *Select Edge*-Blöcke die beiden Kanten für den Vorsprung in der Mitte der Fassade aus.
3. Wiederholen Sie dies für die Kanten am unteren Ende der Fassade in Revit.
4. In den *Watch*-Blöcken wird angezeigt, dass in Dynamo jetzt Linien vorhanden sind. Diese werden automatisch in Dynamo-Geometrie konvertiert, da die Kanten selbst keine Revit-Elemente sind. Diese Kurven sind die Referenzen, die zum Instanziieren der adaptiven Fachwerkbinder entlang der Fassade verwendet werden.

\*Anmerkung: Um eine konsistente Topologie zu erhalten, wird hier ein Modell verwendet, dem keine zusätzlichen Flächen oder Kanten hinzugefügt wurden. Sie können zwar mithilfe von Parametern seine Form ändern, seine Struktur bleibt jedoch einheitlich.



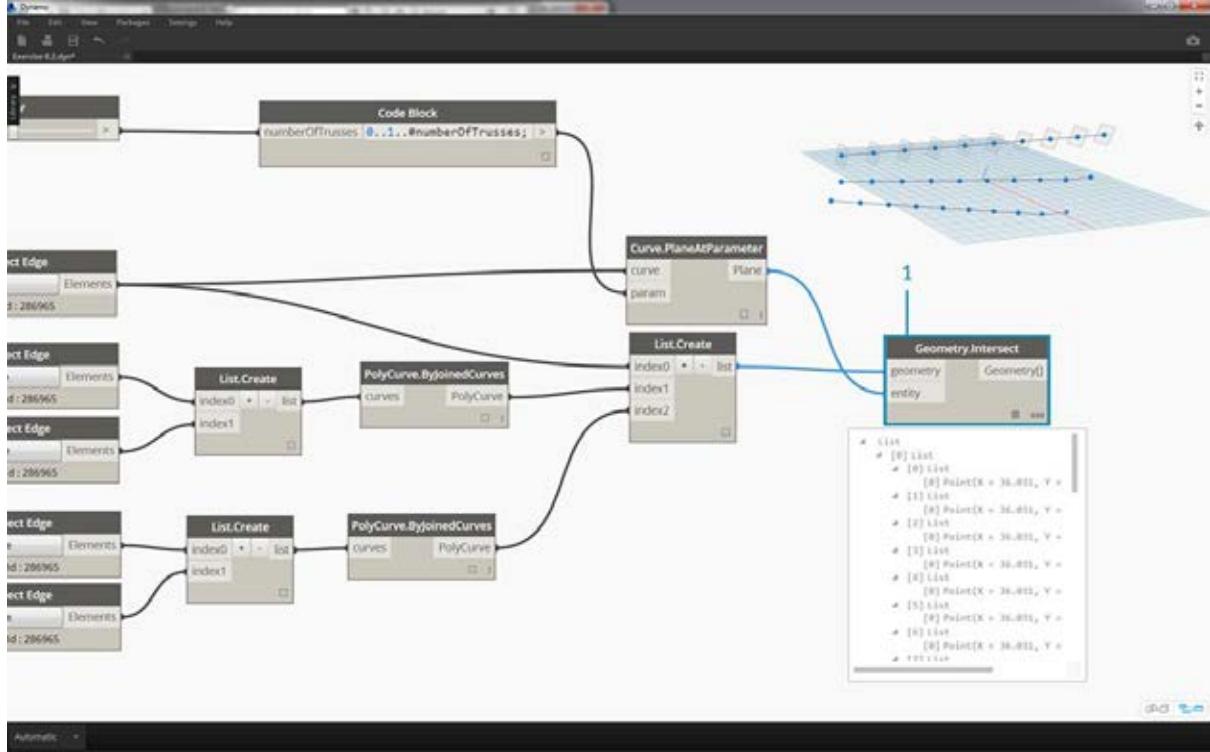
Als Erstes müssen Sie die Kurven verbinden und zu einer Liste zusammenführen. Dadurch *gruppieren* Sie die Kurven, um geometrische Operationen durchführen zu können.

1. Erstellen Sie eine Liste für die beiden Kurven in der Mitte der Fassade.
2. Verbinden Sie die beiden Kurven zu einer Polykurve, indem Sie die *List.Create*-Komponente mit einem *Polycurve.ByJoinedCurves*-Block verbinden.
3. Erstellen Sie eine Liste für die beiden Kurven am unteren Rand der Fassade.
4. Verbinden Sie die beiden Kurven zu einer Polykurve, indem Sie die *List.Create*-Komponente mit einem *Polycurve.ByJoinedCurves*-Block verbinden.
5. Führen Sie schließlich die drei Hauptkurven (eine Linie und zwei Polykurven) zu einer Liste zusammen.



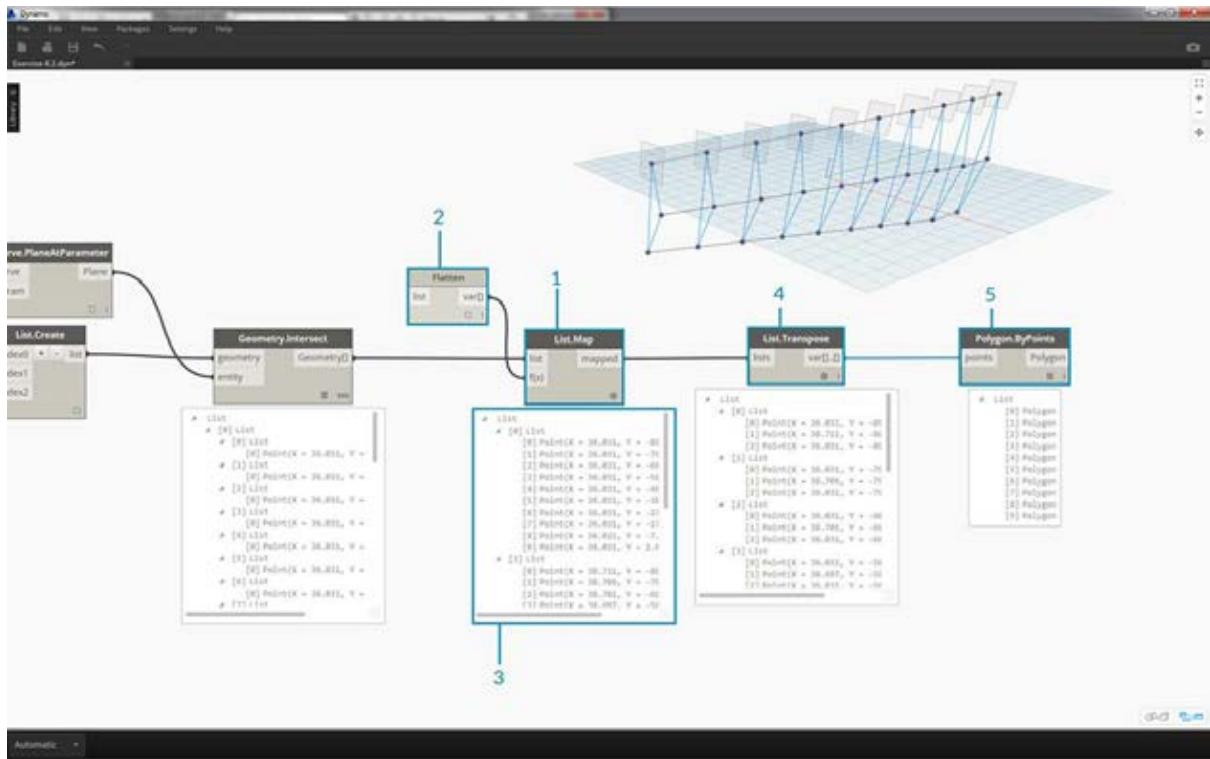
Danach verwenden Sie die oberste der Kurven, die eine über die gesamte Länge der Fassade verlaufende Linie ist. Sie erstellen entlang dieser Linie Ebenen, die die in der Liste zusammengefassten Kurven schneiden.

1. Definieren Sie in einem *Code Block* einen Bereich mit der Syntax `0..1..#numberOfTrusses;`.
2. Verbinden Sie einen *Integer Slider* mit der Eingabe des Codeblocks. Dessen Werte geben naheliegenderweise die Anzahl der Fachwerkbinder an. Beachten Sie, dass der Schieberegler die Anzahl der Einträge in einem von `0` bis `1` definierten Bereich steuert.
3. Verbinden Sie den *Code Block* mit der *param*-Eingabe eines *Curve.PlaneAtParameter*-Blocks und verbinden Sie die obere Kante mit der *curve*-Eingabe. Damit erhalten Sie zehn Ebenen, die gleichmäßig über die Länge der Fassade verteilt sind.



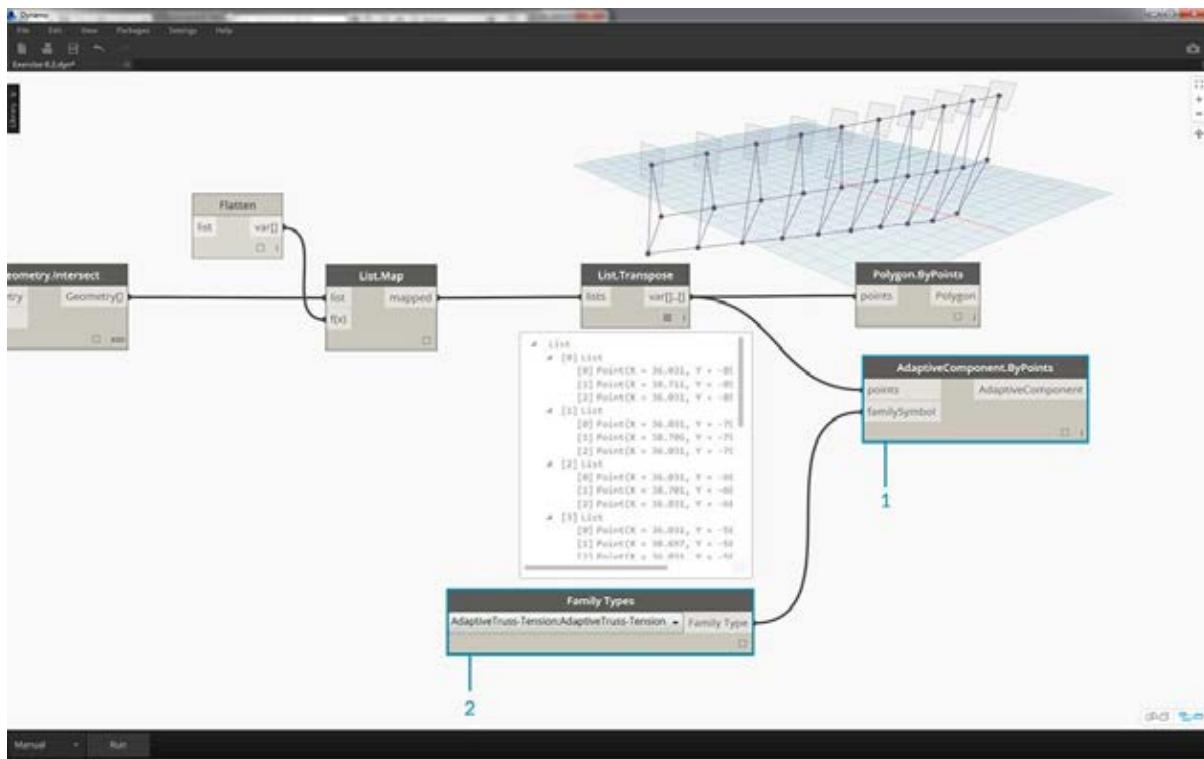
Eine Ebene ist ein abstraktes Geometrieelement, das für einen unendlichen zweidimensionalen Raum steht. Ebenen eignen sich ausgezeichnet zum Erstellen von Konturen und Schnitten wie in diesem Schritt gezeigt.

1. Verwenden Sie als Nächstes den *Geometry.Intersect*-Block: Verbinden Sie *Curve.PlaneAtParameter* mit der *entity*-Eingabe von *Geometry.Intersect*. Verbinden Sie den *List.Create*-Block mit der *geometry*-Eingabe. Im Dynamo-Ansichtsfenster werden daraufhin die Schnittpunkte der einzelnen Kurven mit den definierten Ebenen angezeigt.



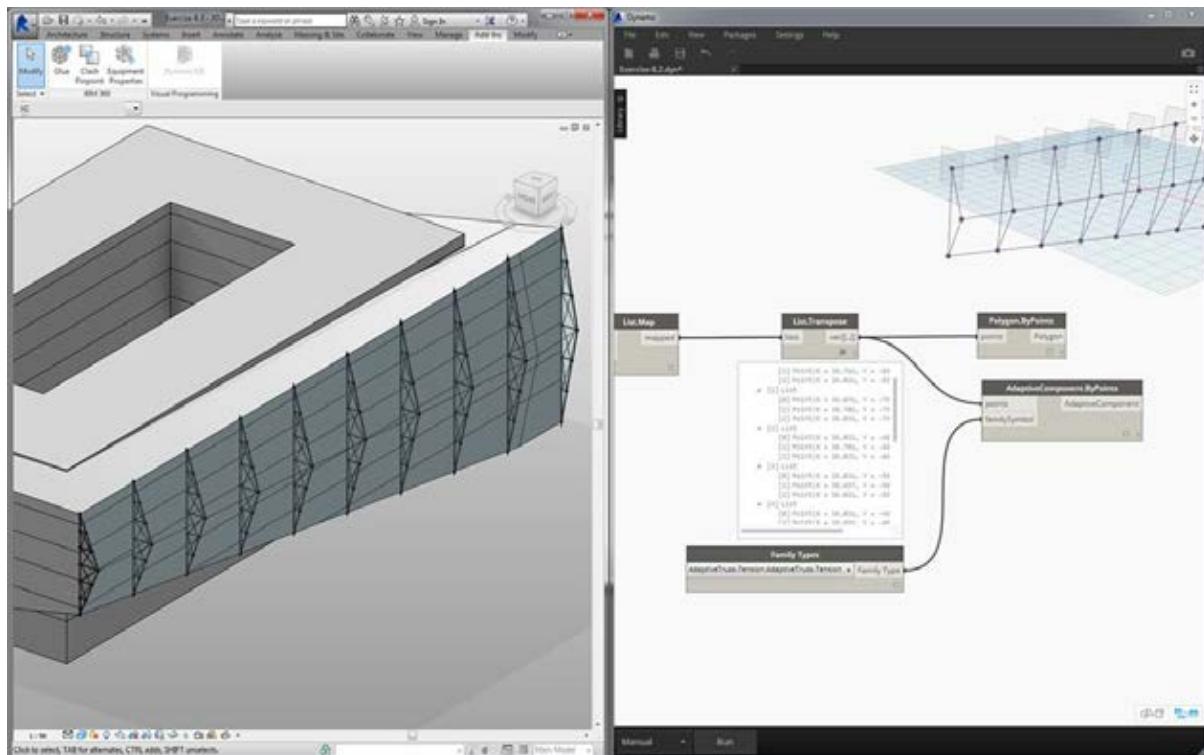
Die Ausgabe ist eine Liste aus Listen von Listen. Sie enthält zu viele Listen für diesen Verwendungszweck. Sie müssen die Liste teilweise vereinfachen. Dazu müssen Sie die zweithöchste Ebene der Liste ansteuern und das Ergebnis vereinfachen. Verwenden Sie dazu die *List.Map*-Operation, die im Kapitel zu Listen in diesem Handbuch beschrieben wird.

1. Verbinden Sie den *Geometry.Intersect*-Block mit der list-Eingabe von *List.Map*.
2. Verbinden Sie einen *Flatten*-Block mit der f(x)-Eingabe von *List.Map*. Als Ergebnis erhalten Sie drei Listen, jeweils mit so vielen Einträgen, wie Fachwerkbinder erstellt werden sollen.
3. Sie müssen diese Daten ändern. Für die Instanziierung des Fachwerkbinders benötigen Sie dieselbe Anzahl adaptiver Punkte wie in der Familie definiert. Dieses adaptive Bauteil weist drei Punkte auf. Sie benötigen also anstelle von drei Listen mit je zehn Einträgen (numberOfTrusses) zehn Listen mit je drei Einträgen. Auf diese Weise können Sie 10 adaptive Bauteile erstellen.
4. Verbinden Sie den *List.Map*-Block mit einem *List.Transpose*-Block. Damit werden die gewünschten Daten ausgegeben.
5. Um sich zu vergewissern, dass Sie die richtigen Daten erhalten haben, fügen Sie im Ansichtsbereich einen *Polygon.ByPoints*-Block hinzu und überprüfen Sie das Ergebnis in der Dynamo-Vorschau.

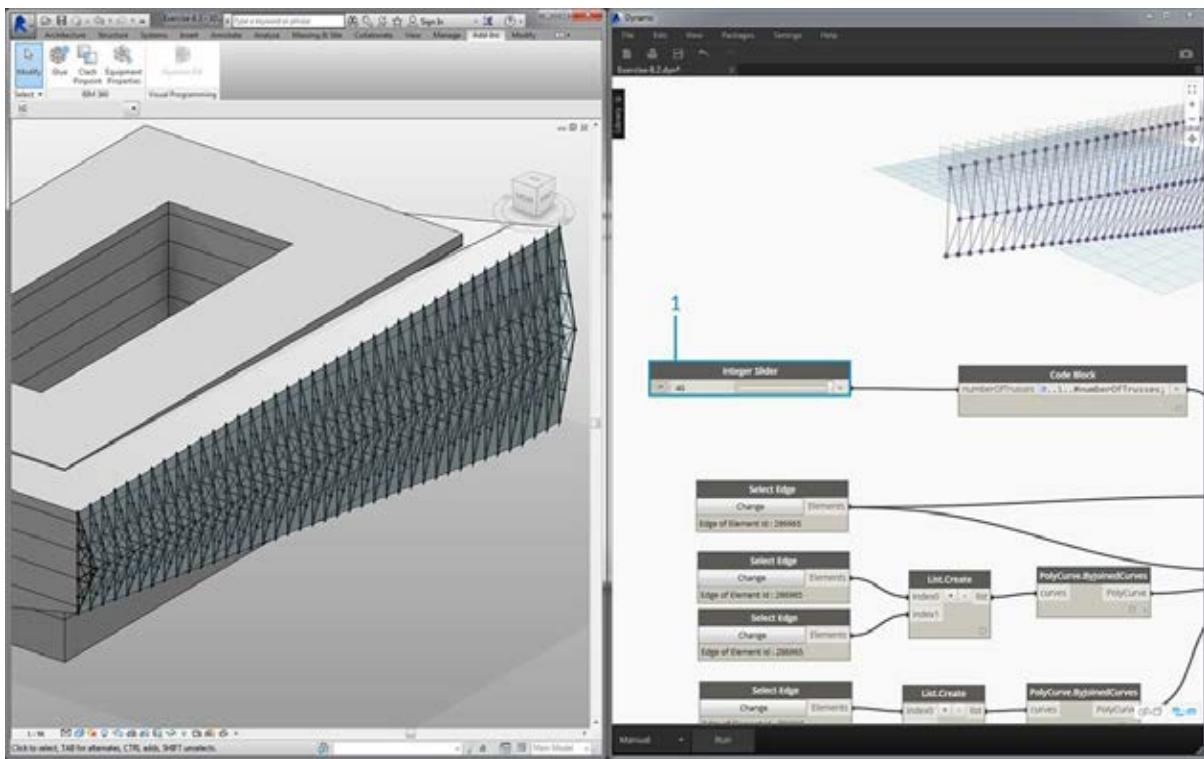


Ordnen Sie jetzt die adaptiven Bauteile in einem Array an, wobei Sie dasselbe Verfahren verwenden wie beim Erstellen der Polygone.

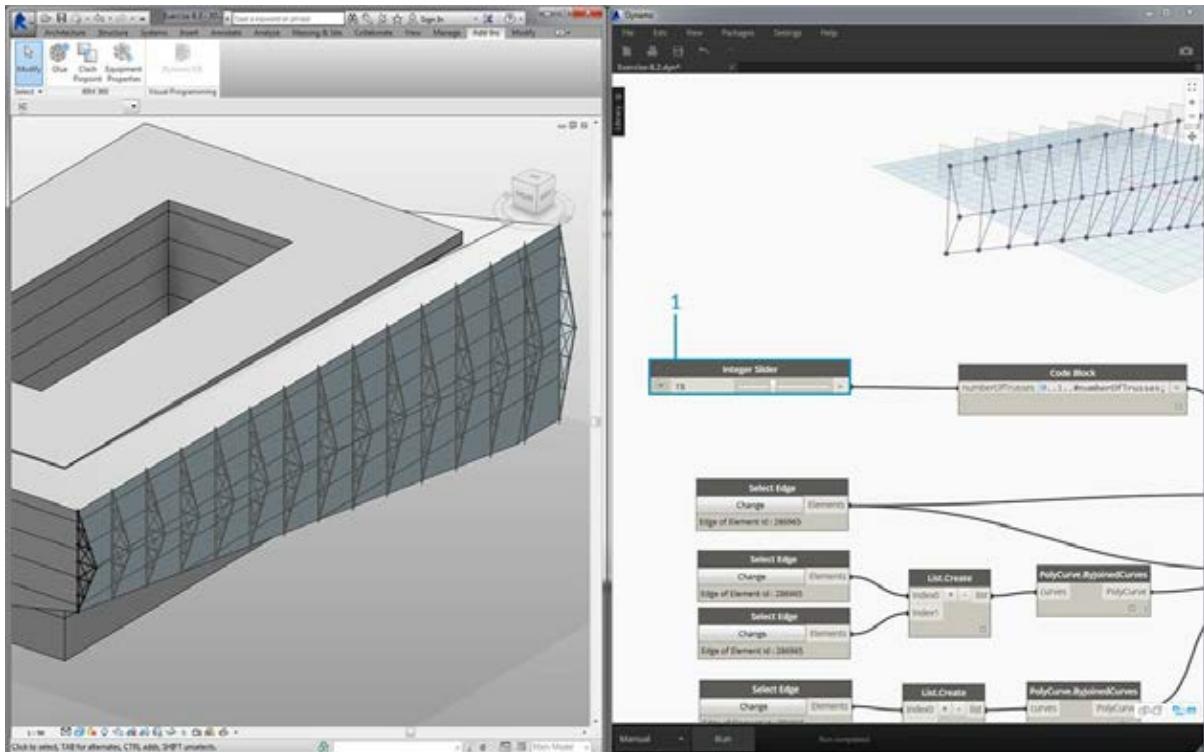
1. Fügen Sie im Ansichtsbereich einen *AdaptiveComponent.ByPoints*-Block hinzu und verbinden Sie den *List.Transpose*-Block mit der *points*-Eingabe.
  2. Wählen Sie in einem *Family Types*-Block die *AdaptiveTruss*-Familie und verbinden Sie den Block mit der *familySymbol*-Eingabe des *AdaptiveComponent.ByPoints*-Blocks.



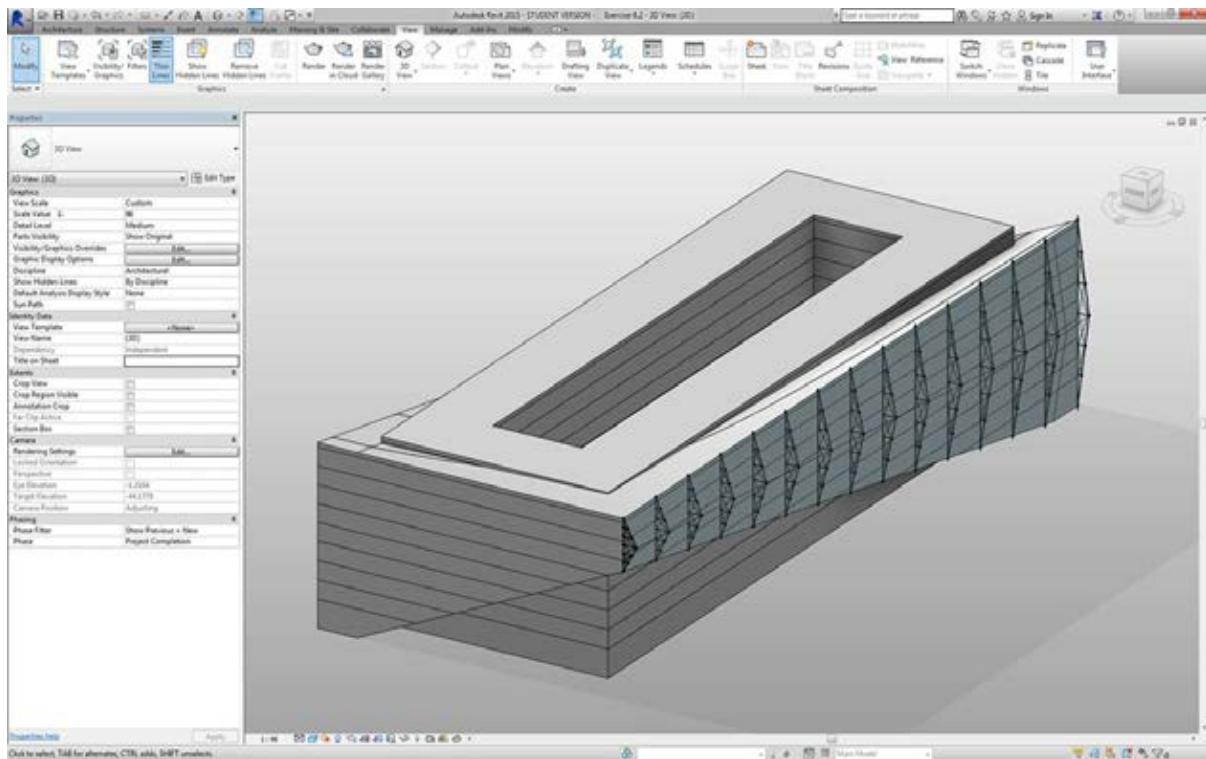
In Revit sind jetzt die zehn Fachwerkbinder gleichmäßig über die Länge der Fassade verteilt zu sehen.



1. Testen Sie das Diagramm: Erhöhen Sie den Wert für *numberOfTrusses* auf 40, indem Sie den Wert im *Integer Slider* ändern. Sie erhalten zahlreiche Fachwerkbinden: Dies mag nicht sonderlich realistisch wirken, es zeigt jedoch, dass die parametrische Verknüpfung funktionstüchtig ist.



1. Wählen Sie einen angemessenen mittleren Wert für das Fachwerksystem, indem Sie 15 für *numberOfTrusses* festlegen.



Führen Sie abschließend einen weiteren Test durch: Indem Sie in Revit den Körper auswählen und seine Exemplarparameter bearbeiten, können Sie die Form des Gebäudes ändern und beobachten, wie die Fachwerkbinden angepasst werden. Denken Sie daran, dass dieses Dynamo-Diagramm geöffnet sein muss, damit die Aktualisierung sichtbar ist. Sobald das Diagramm geschlossen wird, geht die Verknüpfung verloren.

### DirectShape-Elemente

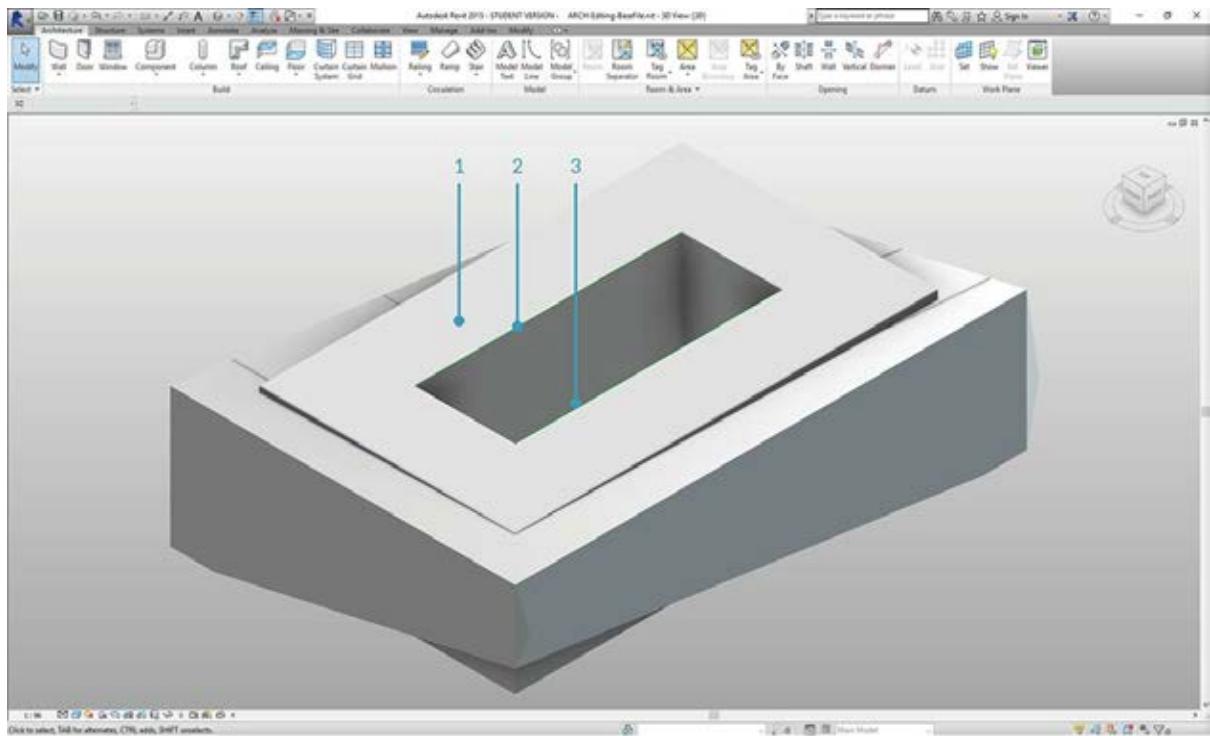
Mit DirectShape steht ein weiteres Verfahren zum Importieren parametrischer Dynamo-Geometrie in Revit zur Verfügung. Kurz zusammengefasst: Das DirectShape-Element und die dazugehörigen Klassen ermöglichen es, extern erstellte geometrische Formen in einem Revit-Dokument zu speichern. Zu dieser Geometrie können geschlossene Körper oder Netze gehören. DirectShape ist in erster Linie für den Import von Formen aus anderen Formaten wie z. B. IFC oder STEP vorgesehen, die nicht genügend Informationen zum Erstellen eines "echten" Revit-Elements zur Verfügung stehen. Die DirectShape-Funktionen eignen sich genau wie bei IFC- und STEP-Arbeitsabläufen auch zum Importieren mit Dynamo erstellter Geometrie als echte Elemente in Revit-Projekte.

Die folgende Übung zeigt den Ablauf für den Import von Dynamo-Geometrie als DirectShape in ein Revit-Projekt. Mithilfe dieses Verfahrens können Sie die Kategorie, das Material und den Namen der importierten Geometrie zuweisen, wobei die parametrische Verknüpfung mit dem Dynamo-Diagramm erhalten bleibt.

### Übungslektion

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

1. [DirectShape.dyn](#)
2. [ARCH-DirectShape-BaseFile.rvt](#)



Öffnen Sie als Erstes die Beispieldatei für diese Lektion: ARCH-DirectShape-BaseFile.rvt.

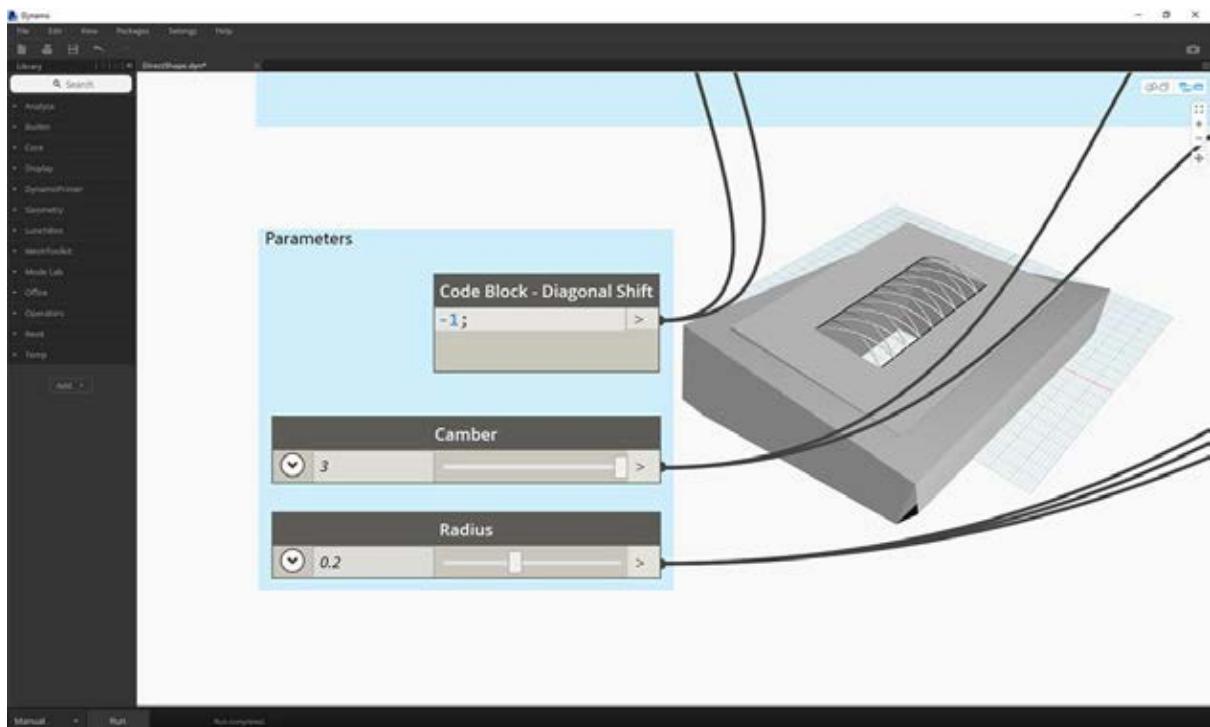
1. Die 3D-Ansicht zeigt den Gebäudekörper aus der letzten Lektion.
2. Entlang der Kante des Foyers verläuft eine Referenzkurve, die Sie in Dynamo referenzieren werden.
3. Entlang der gegenüberliegenden Kante des Foyers verläuft eine zweite Referenzkurve, die ebenfalls in Dynamo referenziert werden soll.



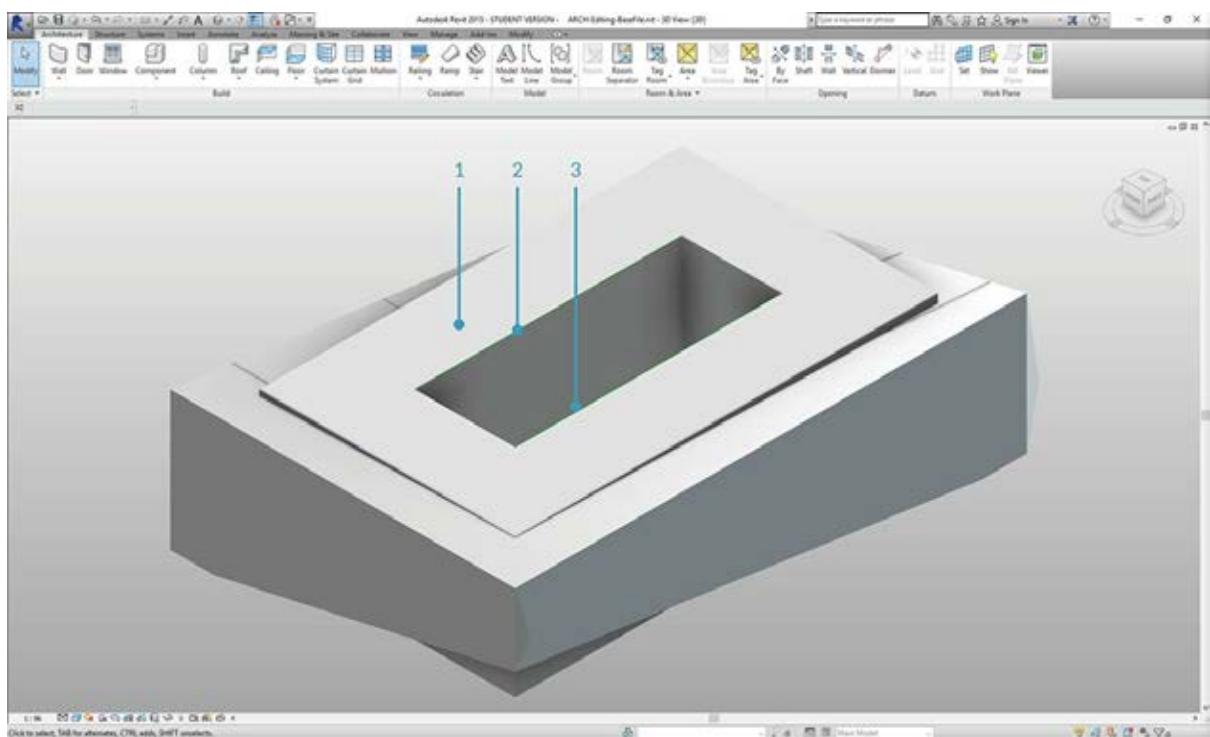
1. Zum Referenzieren der Geometrie in Dynamo verwenden Sie *Select Model Element* für die einzelnen Elemente in Revit. Wählen Sie den Körper in Revit aus und importieren Sie die Geometrie mithilfe von *Element.Faces* in Dynamo. Dadurch wird der Körper in der Dynamo-Vorschau angezeigt.
2. Importieren Sie eine der Referenzkurven mithilfe von *Select Model Element* und *CurveElement.Curve* in Dynamo.
3. Importieren Sie die andere Referenzkurve mithilfe von *Select Model Element* und *CurveElement.Curve* in Dynamo.



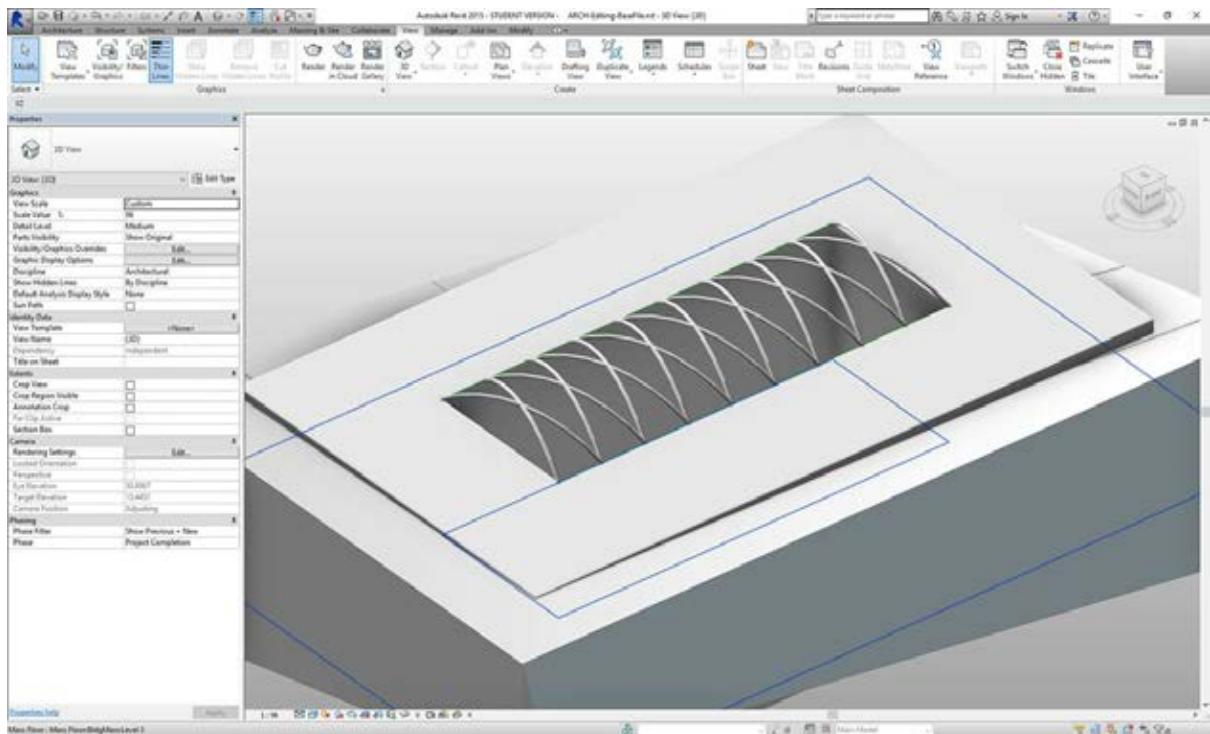
1. Wenn Sie das Beispieldiagramm verkleinern und nach rechts schwenken, sehen Sie eine große Gruppe von Blöcken. Diese stehen für die geometrischen Operationen, mit denen die in der Dynamo-Vorschau gezeigte Gitterkonstruktion für das Dach generiert wird. Diese Blöcke wurden mithilfe der Funktion *Block zu Code* erstellt, die im Abschnitt zu [Codeblöcken](#) dieses Handbuchs beschrieben wird.
2. Diese Konstruktion wird im Wesentlichen durch drei Parameter gesteuert: Diagonal Shift, Camber und Radius.



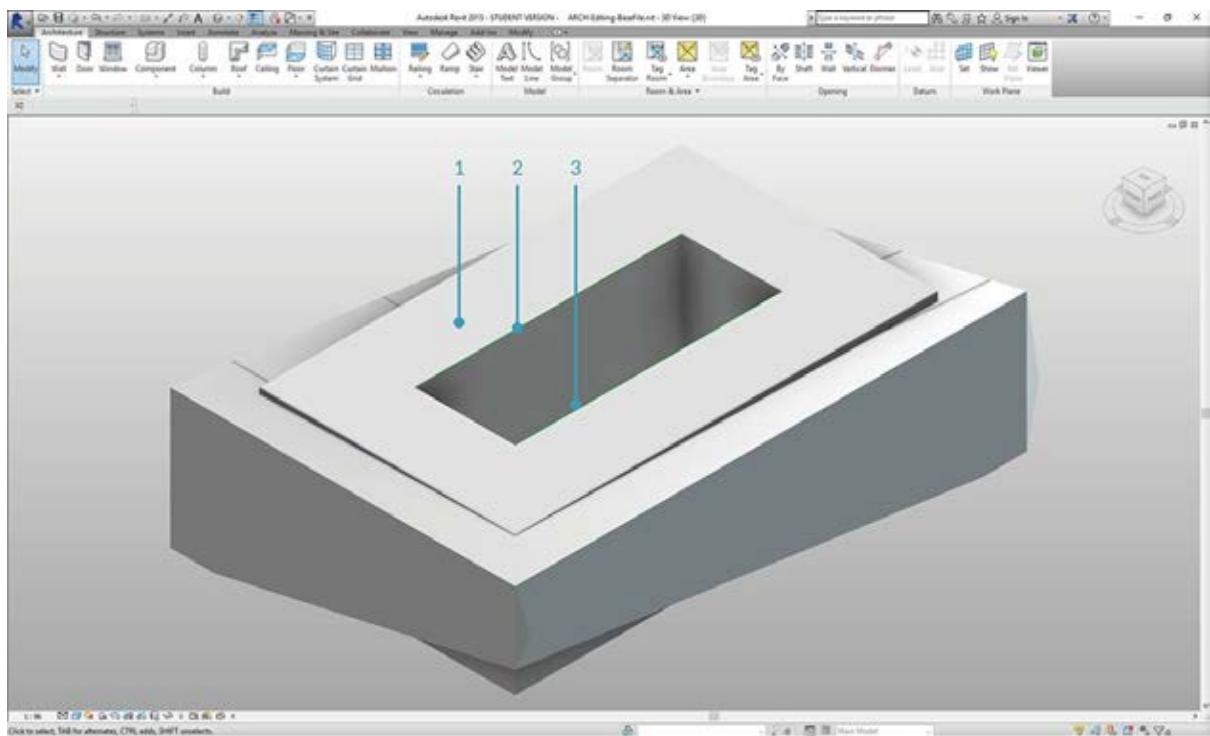
Vergrößern Sie die Darstellung der Parameter für dieses Diagramm. Indem Sie mit diesen experimentieren, erhalten Sie unterschiedliche Geometrie.



1. Fügen Sie im Ansichtsbereich einen *DirectShape.ByGeometry*-Block hinzu. Dieser Block weist die vier Eingaben **geometry**, **category**, **material** und **name** auf.
2. Die Geometrie ist der Körper, der mithilfe des Diagrammteils für die Geometrieerstellung erstellt wird.
3. Die category-Eingabe wird mithilfe der Dropdown-Liste im *Categories*-Block gewählt. Verwenden Sie hier "Tragwerk".
4. Die material-Eingabe wird über das Array von Blöcken darüber ausgewählt. In diesem Fall kann allerdings auch einfach die Vorgabe definiert werden.



Nachdem Sie Dynamo ausgeführt haben, befindet sich die importierte Geometrie im Revit-Projekt auf dem Dach. Dabei handelt es sich nicht um ein allgemeines Modell, sondern ein Tragwerkselement. Die parametrische Verknüpfung mit Dynamo bleibt erhalten.



1. Experimentieren Sie mit dem Dynamo-Diagramm, indem Sie für den Parameter Diagonal Shift den Wert -2 festlegen. Wenn Sie anschließend Dynamo erneut ausführen, wird eine neue DirectShape importiert.

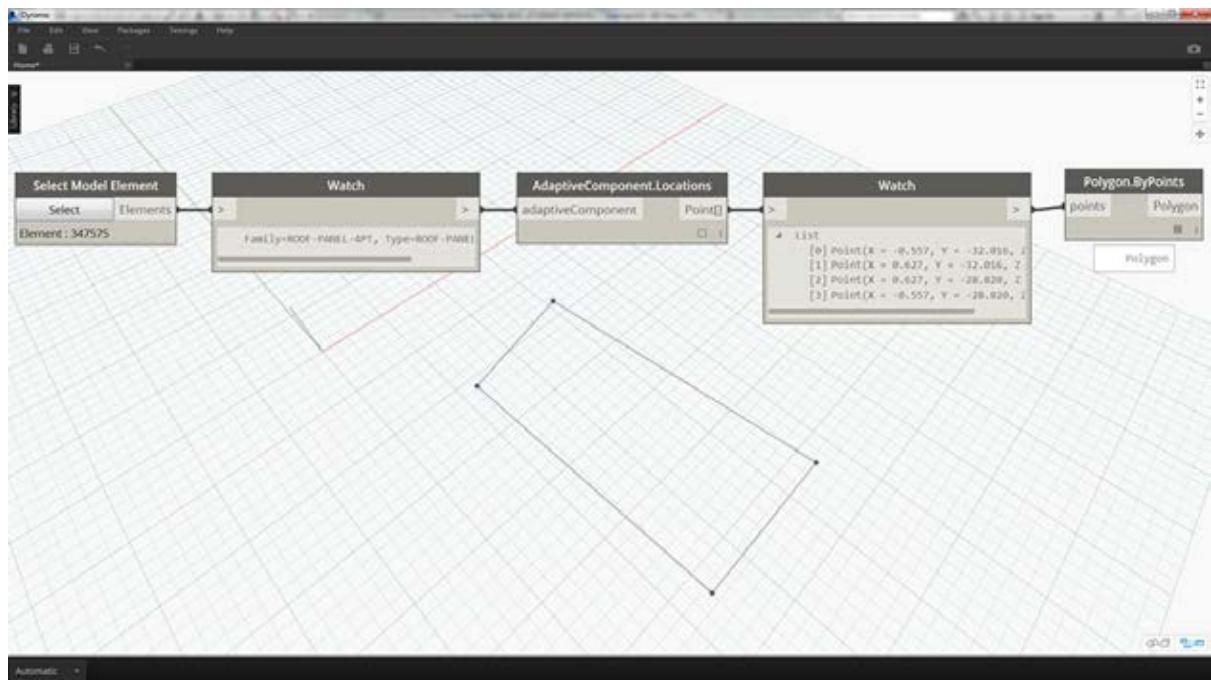
# Anpassen

## Anpassen

Bis hierher wurde die Bearbeitung eines einfachen Gebäudekörpers behandelt. Im Folgenden wird die Verknüpfung zwischen Revit und Dynamo am Beispiel der Bearbeitung zahlreicher Elemente in ein und demselben Vorgang ausführlicher erläutert. Anpassungen in größerem Rahmen sind etwas komplizierter, da für die Datenstrukturen komplexere Listenoperationen durchgeführt werden müssen. Deren Ausführung folgt jedoch grundsätzlich denselben Prinzipien. Im Folgenden werden einige Analysemöglichkeiten ausgehend von einer Gruppe adaptiver Bauteile beschrieben.

### Punktposition

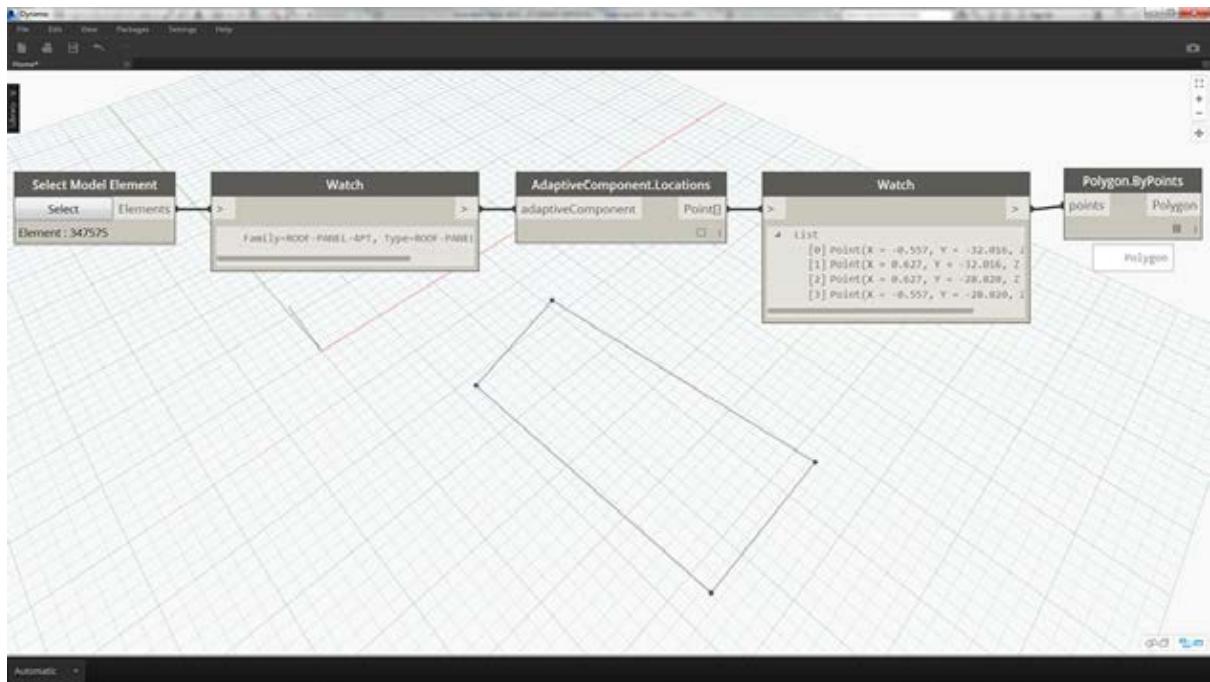
Angenommen, Sie haben eine Gruppe adaptiver Bauteile erstellt und möchten deren Parameter anhand ihrer Punktpositionen bearbeiten. Mithilfe dieser Punkte könnten Sie z. B. einen Parameter für die Dicke in Verbindung mit der Fläche des Elements steuern. Oder Sie könnten einen Parameter für die Opazität in Abhängigkeit von der Sonneneinstrahlung im Jahresverlauf steuern. Dynamo ermöglicht die Verbindung zwischen Analysen und Parametern in wenigen einfachen Schritten. Die folgende Übung zeigt eine grundlegende Version hiervon.



Fragen Sie die adaptiven Punkte eines ausgewählten adaptiven Bauteils mithilfe des *AdaptiveComponent.Locations*-Blocks ab. Dadurch können Sie eine abstrakte Version eines Revit-Elements für die Analyse nutzen.

Indem Sie die Positionen der Punkte adaptiver Bauteile extrahieren, können Sie eine Reihe von Analysen für dieses Element ausführen. Mithilfe eines adaptiven Bauteils mit vier Punkten können Sie beispielsweise die Ebenenabweichung eines gegebenen Dachelements analysieren.

### Analyse der Solarausrichtung



Ordnen Sie mithilfe der Neuzuordnungsfunktion einen Datensatz einem Parameterbereich zu. Dies ist eines der wichtigsten Werkzeuge für parametrische Modelle, wie in der unten folgenden Übung gezeigt.

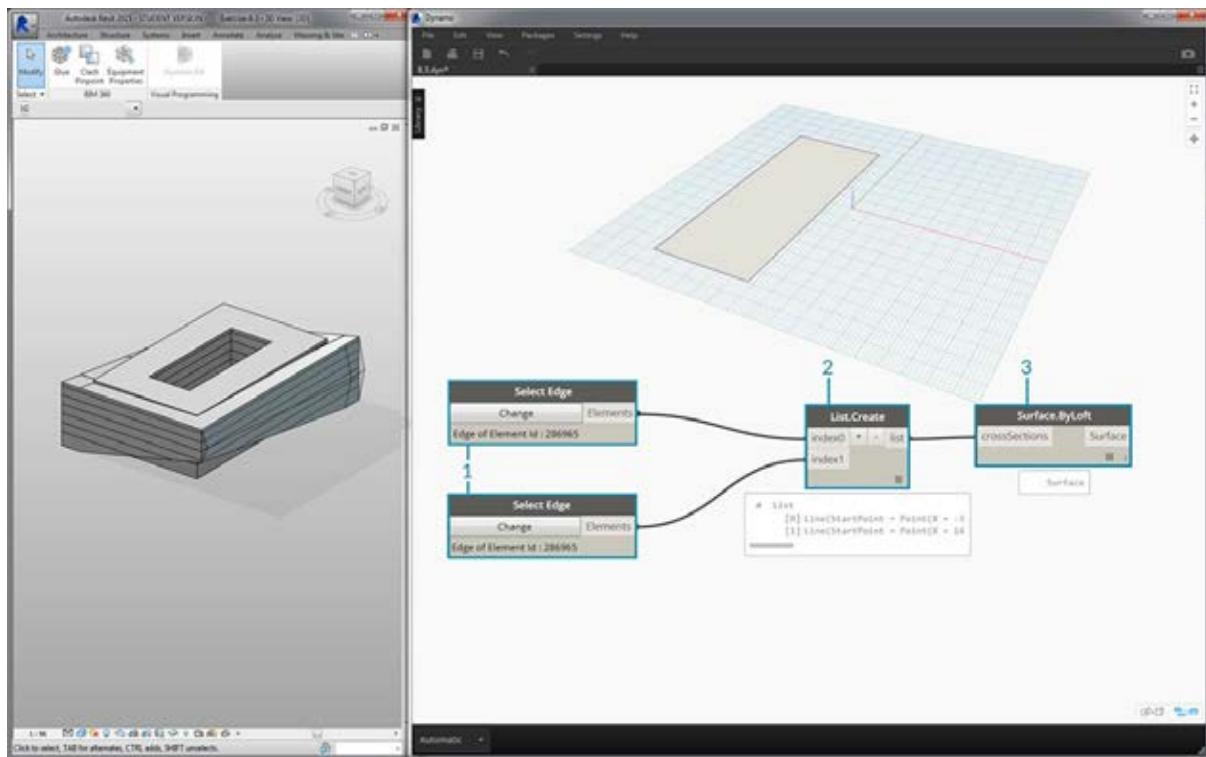
In Dynamo können Sie über die Positionen der Punkte von adaptiven Bauteilen eine optimale Ebene für jedes einzelne Element erstellen. Darüber hinaus können Sie den Sonnenstand aus der Revit-Datei abfragen und die Ausrichtung der Ebene relativ zur Sonne mit derjenigen anderer adaptiver Bauteile vergleichen. In der folgenden Übung richten Sie dies ein, indem Sie eine algorithmische Dachform erstellen.

## Übungslektion

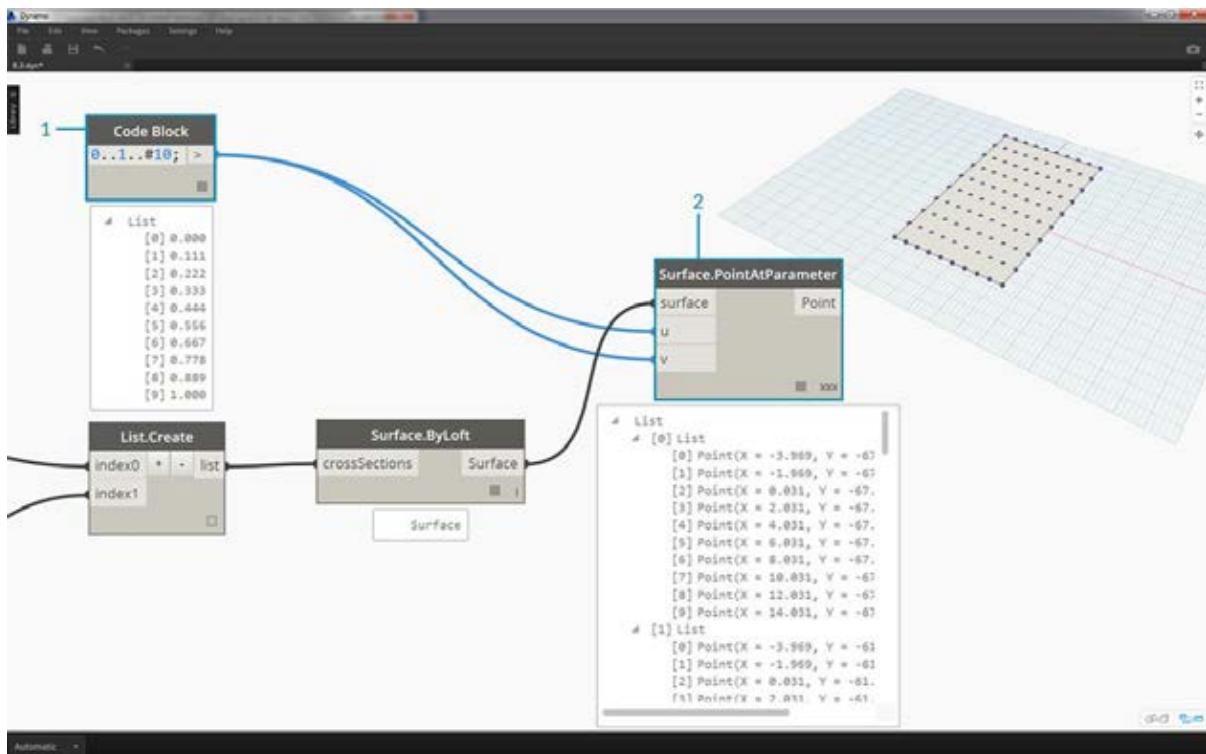
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

1. [Customizing.dyn](#)
2. [ARCH-Customizing-BaseFile.rvt](#)

In dieser Übung entwickeln Sie die im vorigen Abschnitt entwickelten Verfahren weiter. In diesem Fall definieren Sie eine parametrische Oberfläche aus Revit-Elementen, instanziiieren adaptive Bauteile mit vier Punkten und bearbeiten diese anschließend anhand ihrer Ausrichtung relativ zur Sonne.

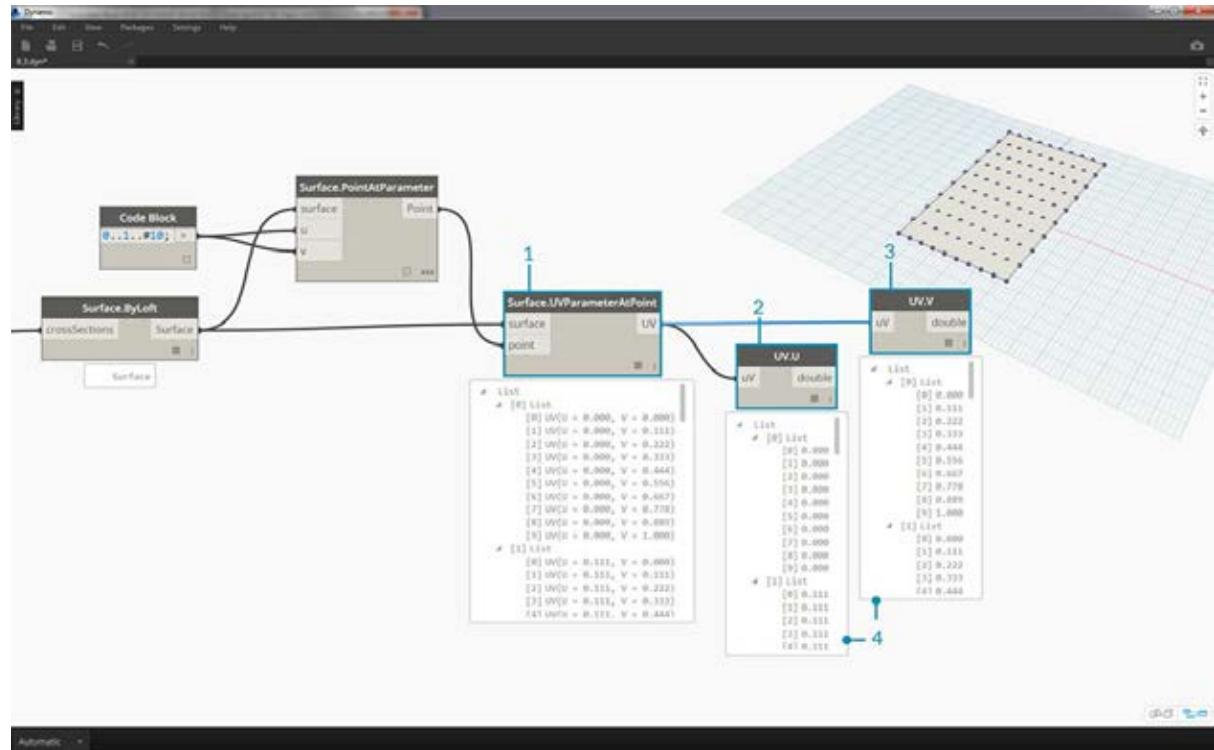


1. Beginnen Sie, indem Sie mithilfe eines *Select Edge*-Blocks zwei Kanten auswählen. Die beiden Kanten sind die Längsseiten des Foyers.
2. Fassen Sie die beiden Kanten mithilfe eines *List.Create*-Blocks in einer Liste zusammen.
3. Erstellen Sie mithilfe eines *Surface.ByLoft*-Blocks eine Oberfläche zwischen den beiden Kanten.

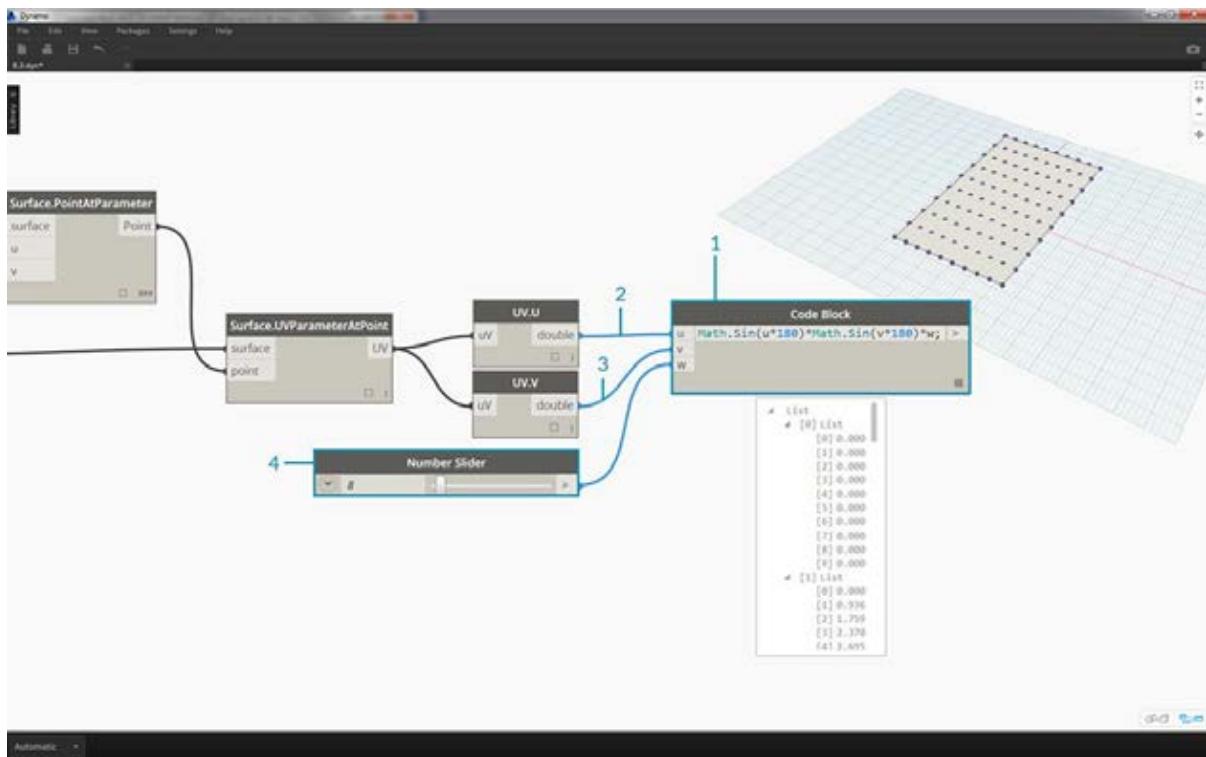


1. Definieren Sie in einem *Code Block* einen Bereich von 0 bis 1 mit 10 gleichmäßig verteilten Werten:  
0..1..#10;.
2. Verbinden Sie den *Code Block* mit den *u-* und *v-Eingaben* eines *Surface.PointAtParameter*-Blocks und verbinden Sie den *Surface.ByLoft*-Block mit der *surface*-Eingabe. Klicken Sie mit der rechten Maustaste auf den Block und ändern Sie die *Vergitterung* in *Kreuzprodukt*. Dadurch erhalten Sie ein Raster aus Punkten auf der Oberfläche.

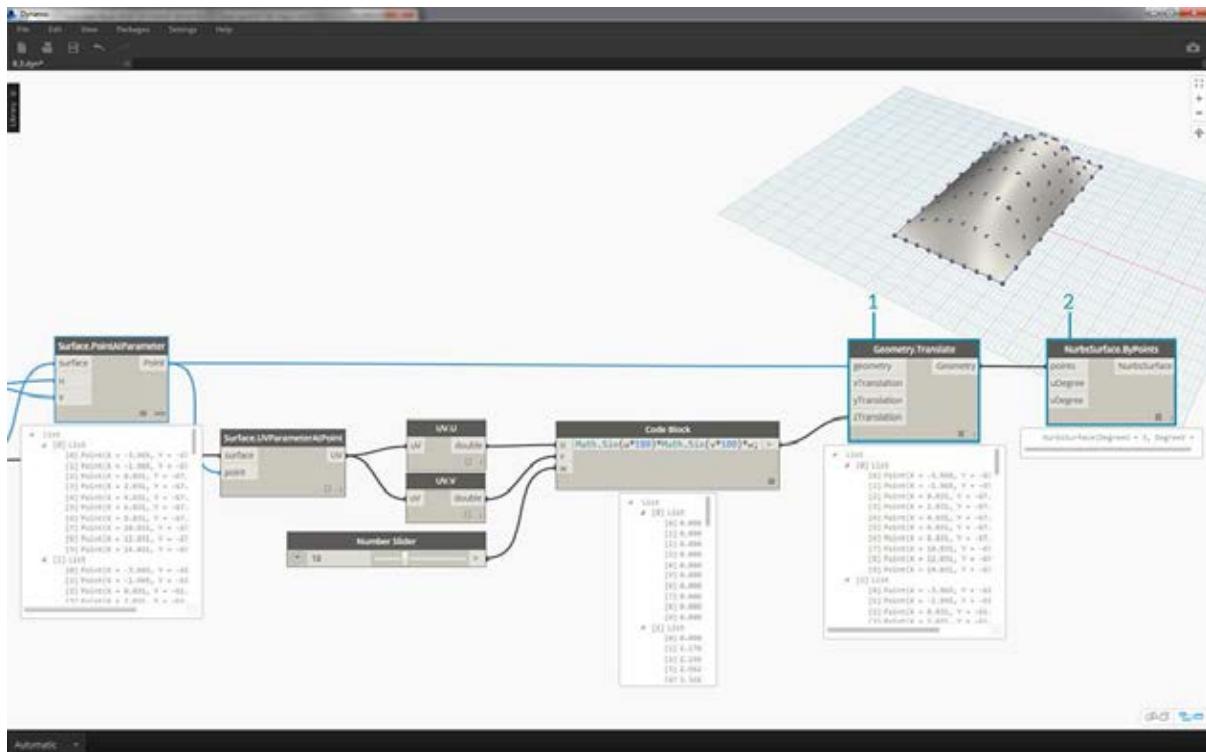
Die Punkte in diesem Raster werden als Steuerpunkte für eine parametrisch definierte Oberfläche genutzt. Als Nächstes extrahieren Sie die u- und v-Positionen dieser Punkte, damit Sie sie mit einer parametrischen Formel verbinden und dabei die bestehende Datenstruktur beibehalten können. Dies ist durch Abfragen der Parameterpositionen der eben erstellten Punkte möglich.



1. Fügen Sie einen *Surface.ParameterAtPoint*-Block in den Ansichtsbereich ein und verbinden Sie seine Eingaben wie oben gezeigt.
2. Fragen Sie die *u*-Werte dieser Parameter mithilfe eines *UV.U*-Blocks ab.
3. Fragen Sie die *v*-Werte dieser Parameter mithilfe eines *UV.V*-Blocks ab.
4. Die Ausgaben zeigen die *u*- bzw. *v*-Werte der einzelnen Punkte in der Oberfläche. Dadurch erhalten Sie für beide Werte jeweils einen Bereich zwischen 0 und 1 in der benötigten Datenstruktur. Jetzt können Sie einen parametrischen Algorithmus anwenden.



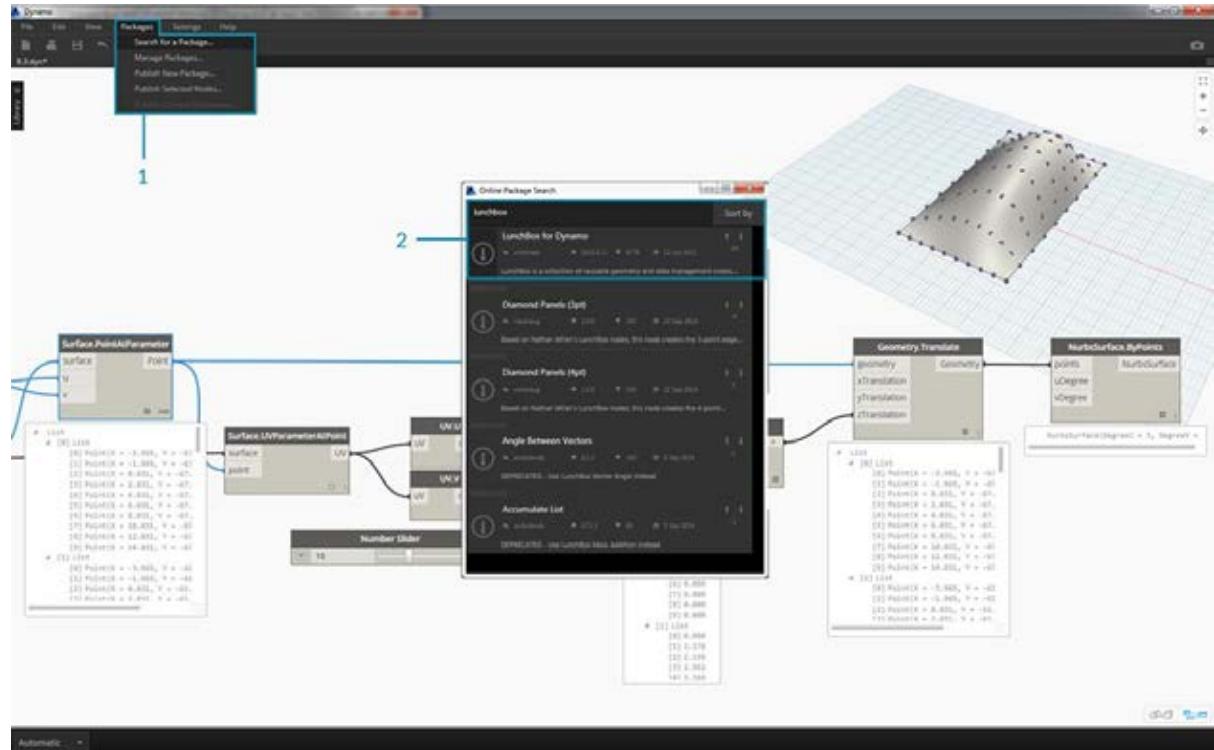
- Fügen Sie im Ansichtsbereich einen *Code Block* hinzu und geben Sie den folgenden Code ein:  
 $\text{Math.Sin}(u*180)*\text{Math.Sin}(v*180)*w;$ . Mit dieser parametrischen Funktion wird aus der flachen Oberfläche eine durch Sinusfunktionen definierte Wölbung erstellt.
- Die *u*-Eingabe wird mit *UV.U* verbunden.
- Die *v*-Eingabe wird mit *UV.V* verbunden.
- Die *w*-Eingabe steht für die *Amplitude* der Form. Verbinden Sie daher einen *Number Slider* mit ihr.



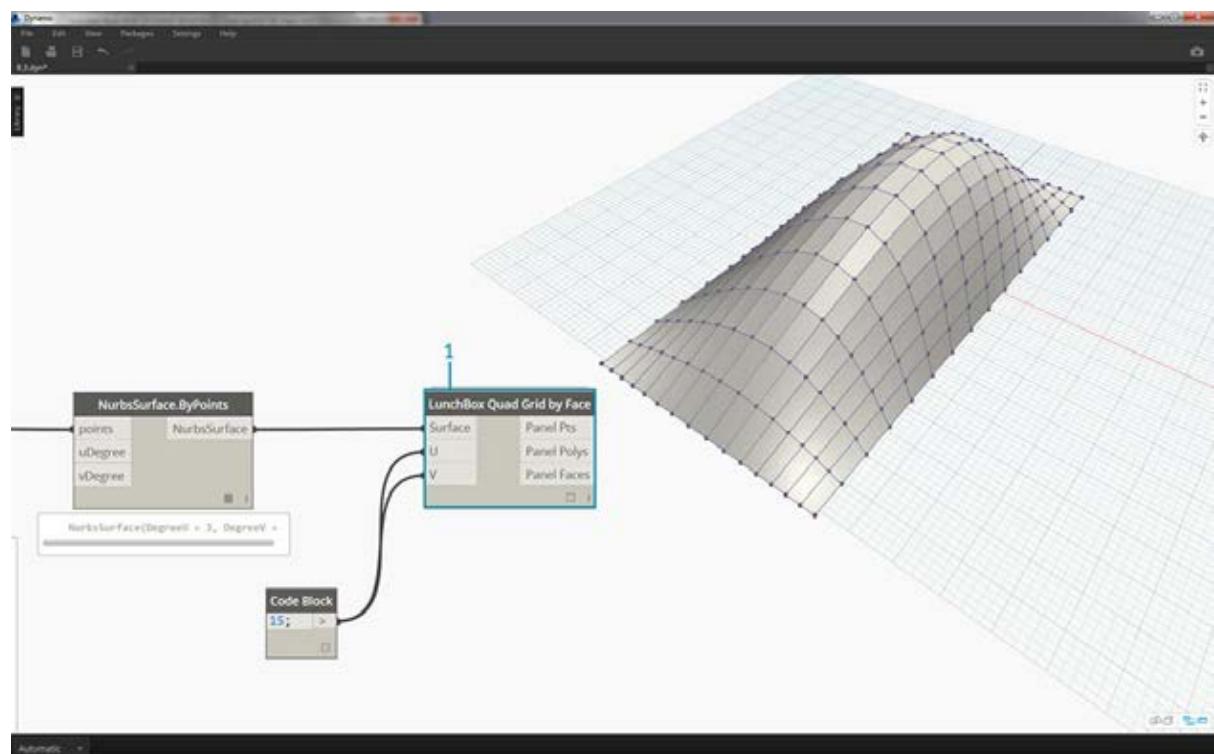
- Dadurch erhalten Sie eine Liste mit durch den Algorithmus definierten Werten. Verwenden Sie diese Werteliste, um die Punkte nach oben, d. h. in +Z-Richtung, zu verschieben. Hierfür verwenden Sie *Geometry.Translate*: Verbinden Sie den *Code Block* mit *zTranslation* und *Surface.PointAtParameter* mit der *geometry*-Eingabe. Die neuen Punkte sollten in der Dynamo-Vorschau angezeigt werden.

- Schließlich erstellen Sie eine Oberfläche, indem Sie den Block aus dem vorigen Schritt mit der points-Eingabe eines *NurbsSurface.ByPoints*-Blocks verbinden. Damit haben Sie eine parametrische Oberfläche erstellt. Wenn Sie den Schiebergler ziehen, können Sie beobachten, wie die Oberfläche sich wölbt und abflacht.

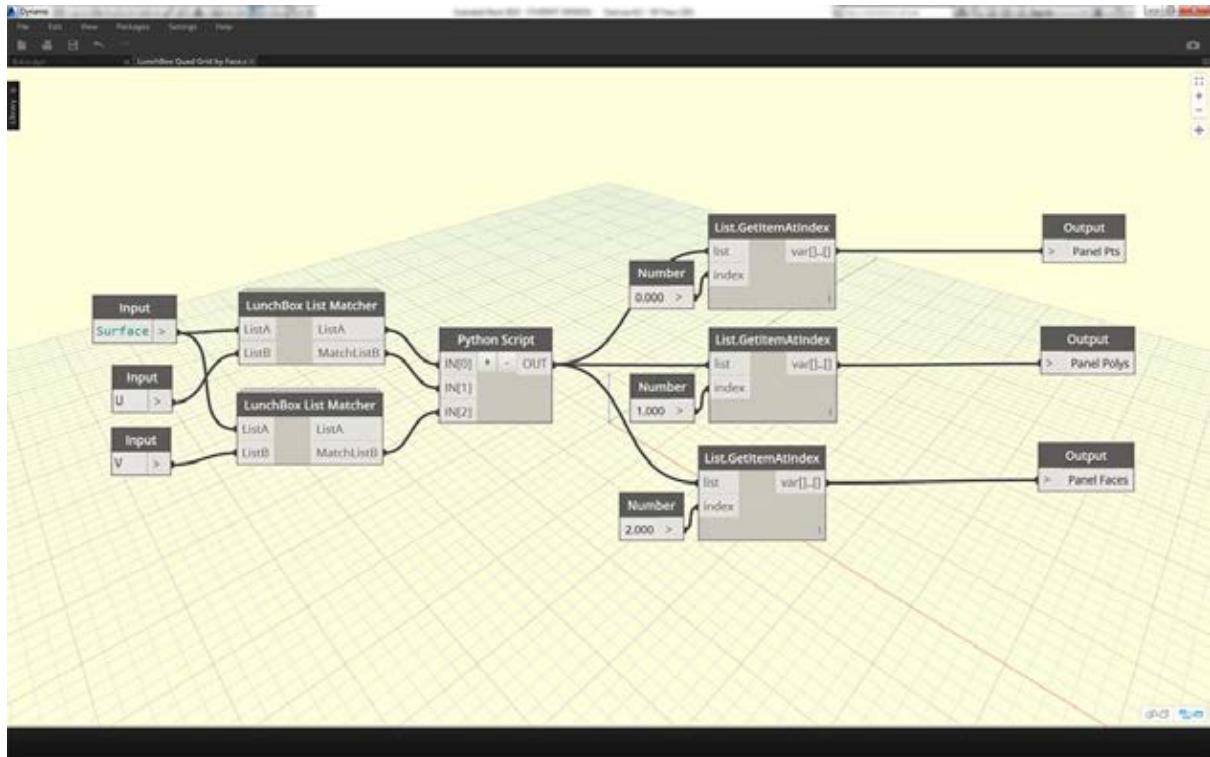
Diese parametrische Oberfläche muss jetzt in Felder unterteilt werden, damit adaptive Bauteile mit vier Punkten darauf angeordnet werden können. Dynamo verfügt nicht über integrierte Funktionen zum Unterteilen von Oberflächen. Suchen Sie daher in der Community nach geeigneten Dynamo-Paketen.



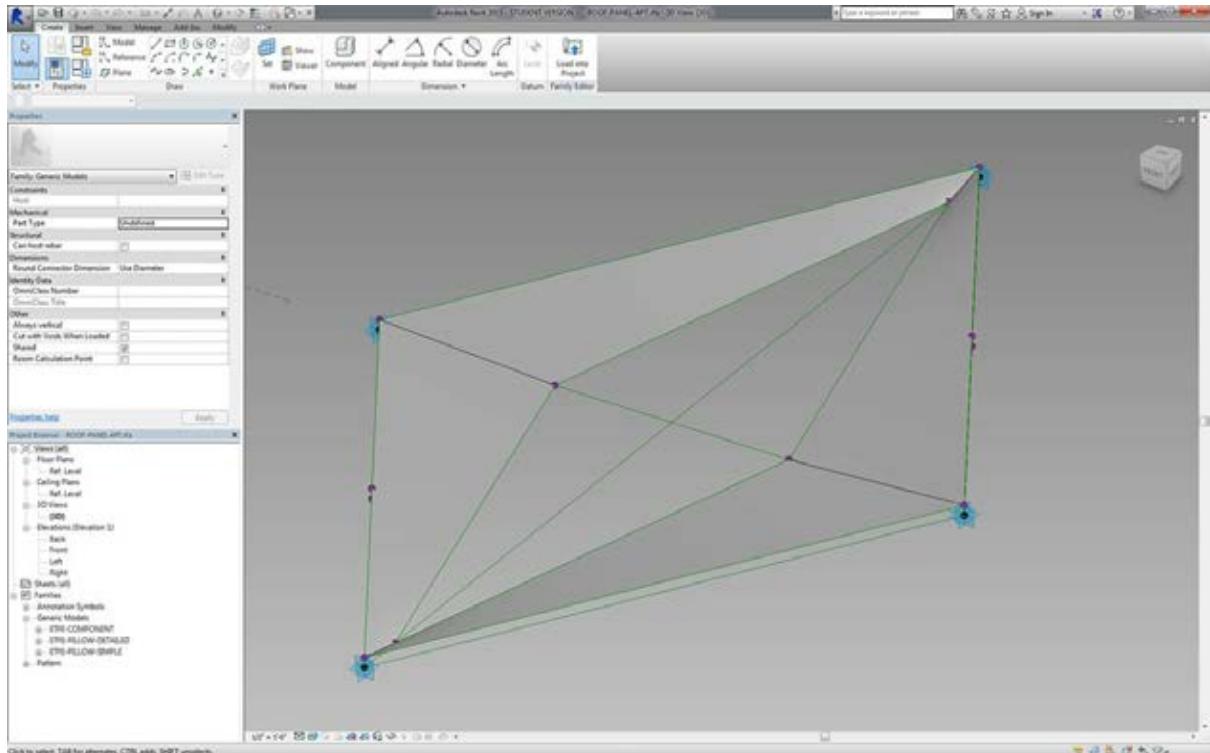
- Wechseln Sie zu *Pakete > Suchen nach Paket*.
- Suchen Sie nach *LunchBox* und laden Sie *LunchBox for Dynamo* herunter. Die hier enthaltenen Werkzeuge sind äußerst hilfreich bei geometrischen Operationen wie dieser.



- Nach dem Herunterladen haben Sie vollen Zugriff auf die LunchBox-Suite. Suchen Sie nach *Quad Grid* und wählen Sie *LunchBox Quad Grid By Face*. Verbinden Sie die parametrische Oberfläche mit der *surface*-Eingabe und legen Sie als Unterteilungen für *U* und *V* jeweils 15 fest. In der Dynamo-Vorschau sollte jetzt eine viereckige Elemente unterteilte Oberfläche zu sehen sein.

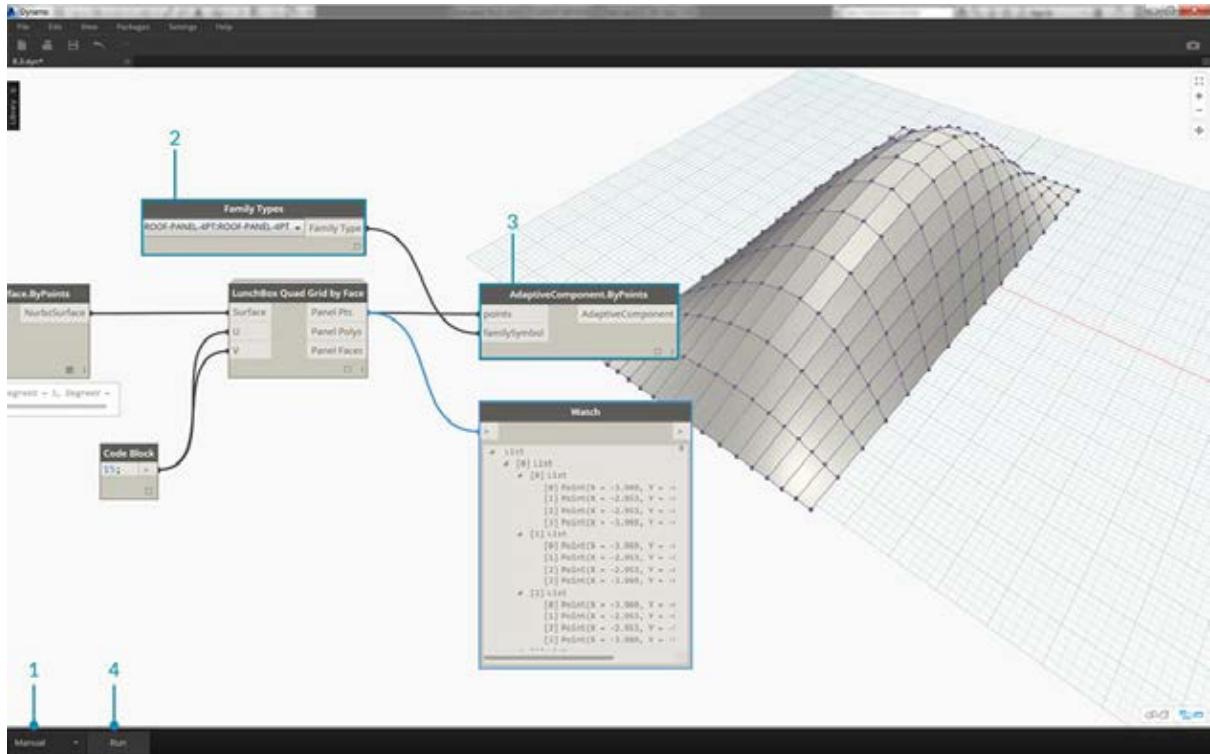


Wenn Sie an der Programmstruktur des *LunchBox*-Blocks interessiert sind, können Sie sie durch Doppelklicken auf den Block anzeigen.



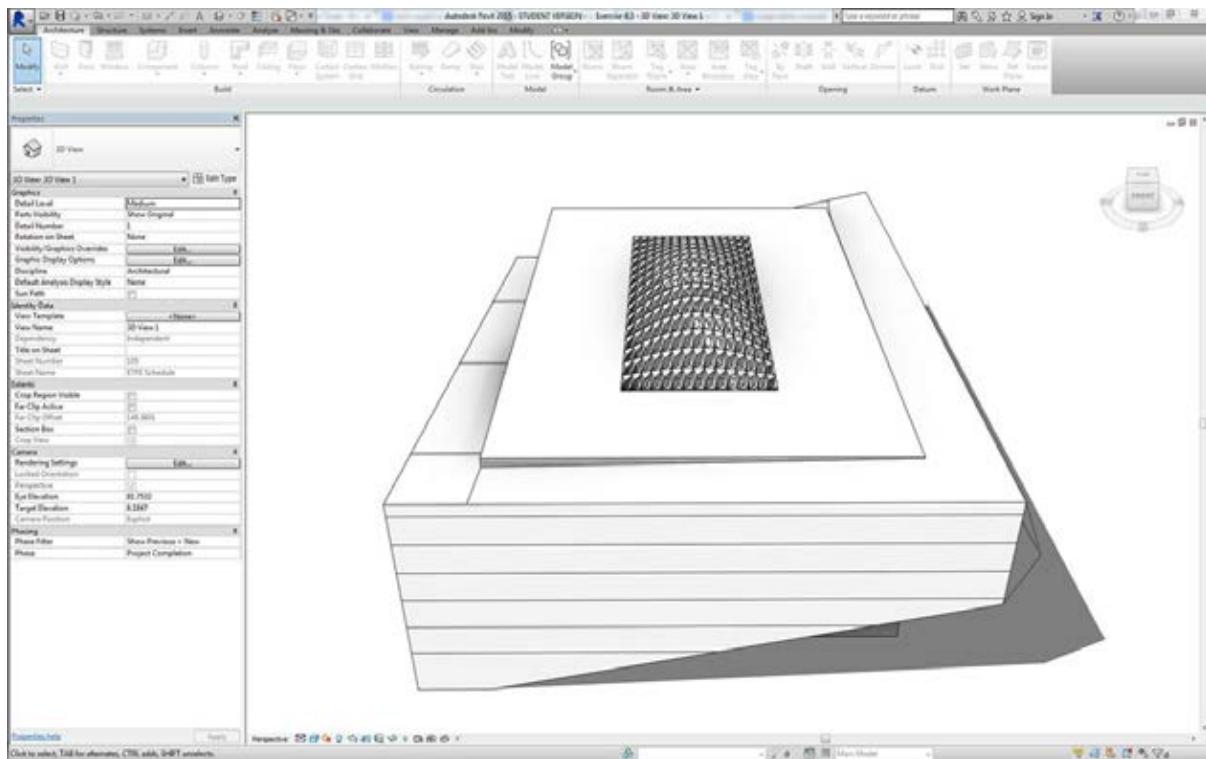
Kehren Sie zu Revit und zum hier verwendeten adaptiven Bauteil zurück. Ohne allzu sehr ins Einzelne zu gehen: Dies ist das zu instanzierende Dachelement. Dieses adaptive Bauteil mit vier Punkten ist eine grobe Darstellung eines EFTE-Systems. Die Öffnung des Abzugskörpers in der Mitte wird durch einen Parameter namens *ApertureRatio*

gesteuert.

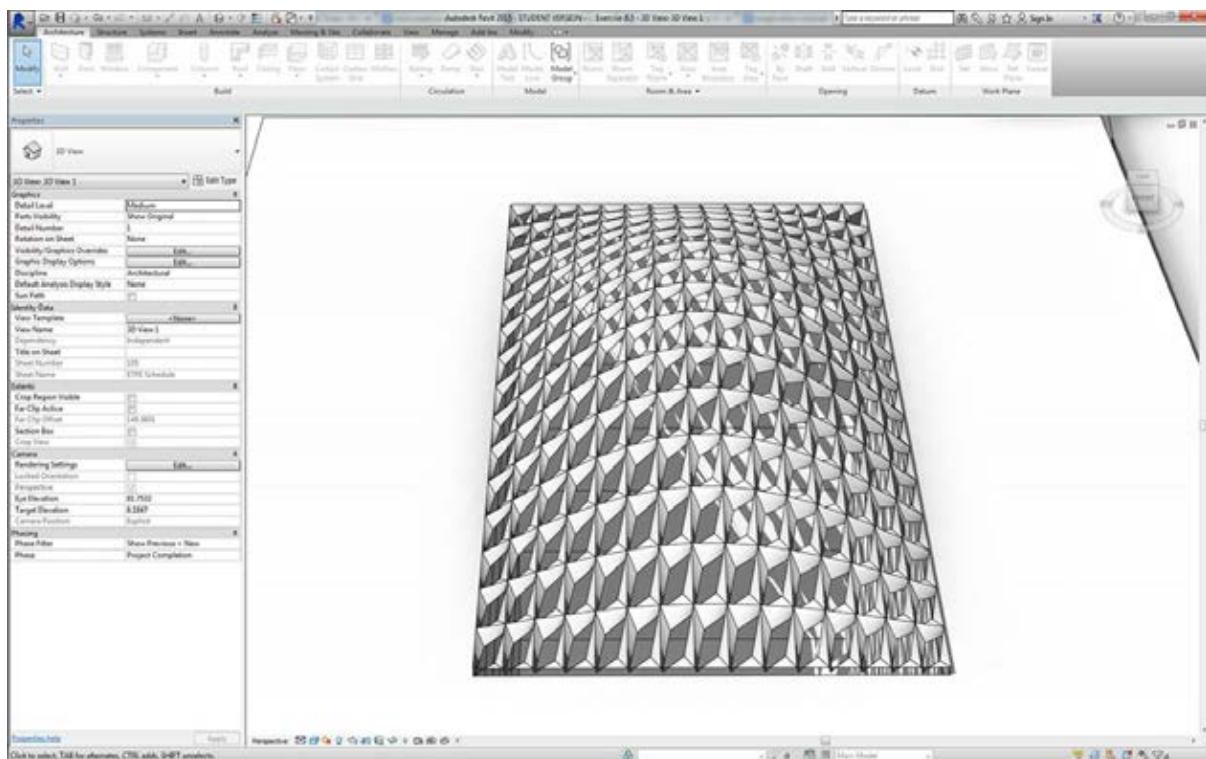


1. Da hier große Mengen von Geometrie in Revit instanziert werden, achten Sie darauf, den Dynamo-Solver auf *Manuell* einzustellen.
2. Fügen Sie im Ansichtsbereich einen *Family Types*-Block ein und wählen Sie "ROOF-PANEL-4PT".
3. Fügen Sie im Ansichtsbereich einen *AdaptiveComponent.ByPoints*-Block ein und verbinden Sie die *Panel Pts*-Ausgabe des *LunchBox Quad Grid by Face*-Blocks mit der *points*-Eingabe. Verbinden Sie den *Family Types*-Block mit der *familySymbol*-Eingabe.
4. Klicken Sie auf *Ausführen*. Revit benötigt etwas Zeit zum Erstellen der Geometrie. Wenn dies zu lange dauert, reduzieren Sie den *Codeblock*-Wert 15 auf eine kleinere Zahl. Dadurch erhalten Sie weniger Elemente auf dem Dach.

Anmerkung: Falls die Berechnung von Blöcken in Dynamo sehr lange dauert, können Sie die Blockfunktionen anhalten ("einfrieren") und damit die Ausführung von Revit-Vorgängen unterbrechen, während Sie Ihr Diagramm entwickeln. Weitere Informationen zum Anhalten von Blöcken finden Sie im entsprechenden Abschnitt im Kapitel [Körper](#).

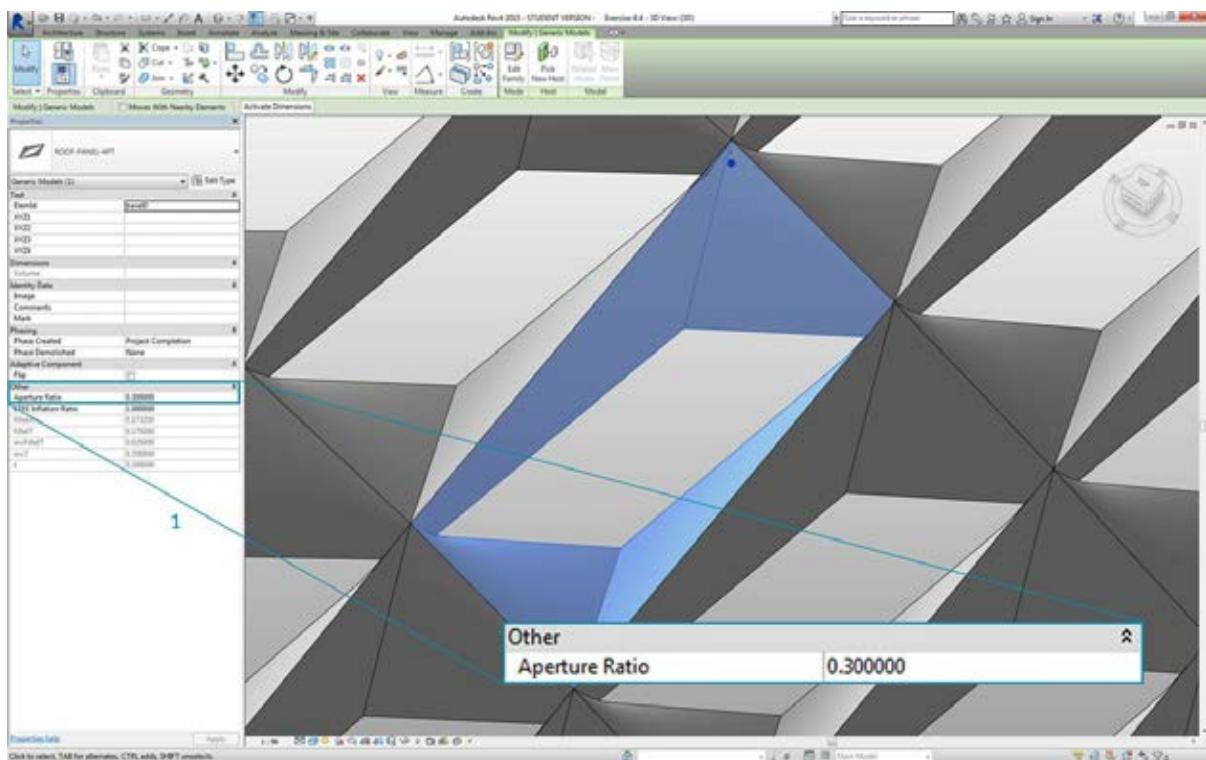


In Revit sind die Elemente jetzt auf dem Dach angeordnet.

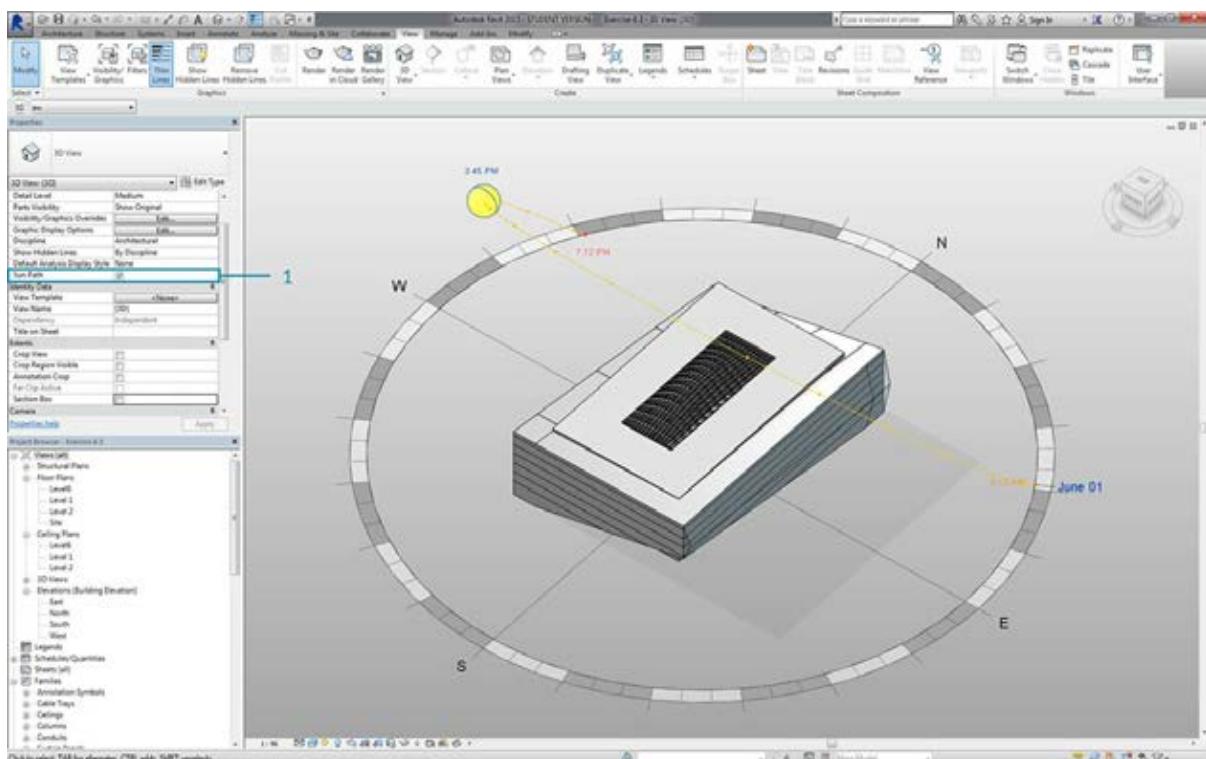


Wenn Sie die Ansicht vergrößern, erhalten Sie einen genaueren Eindruck der Eigenschaften der Oberfläche.

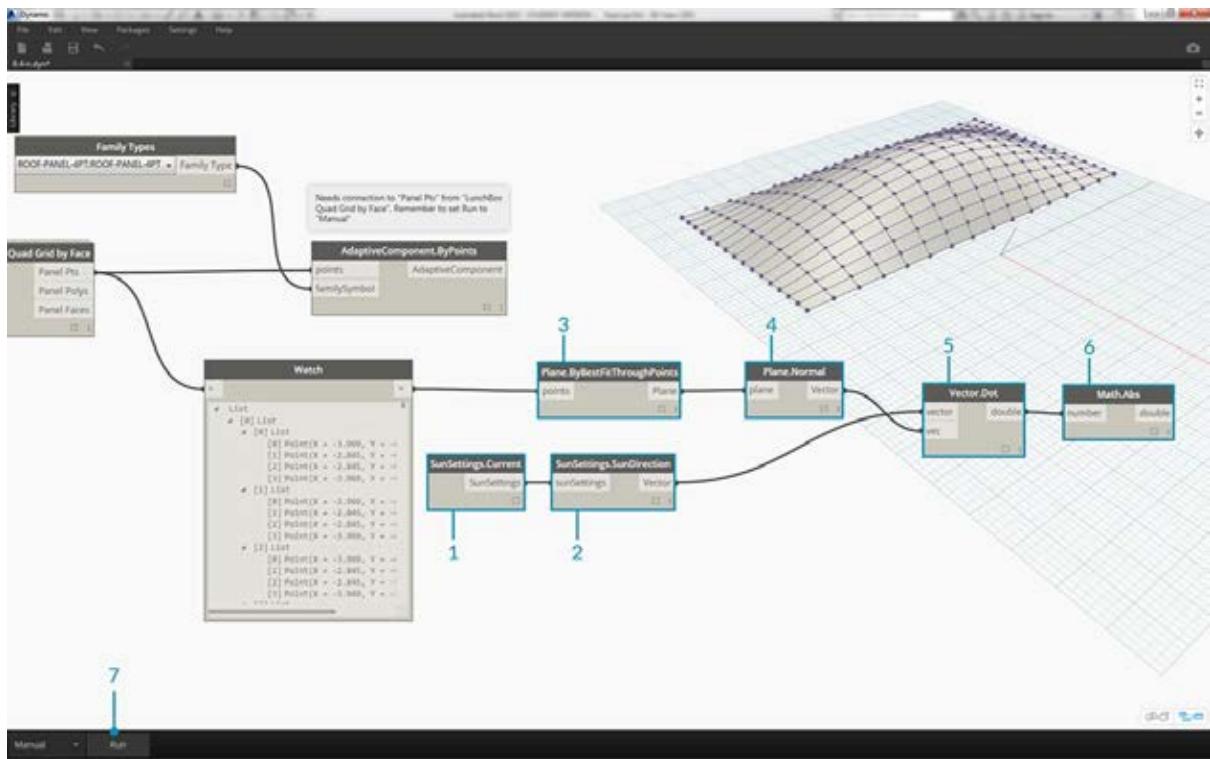
## Analyse



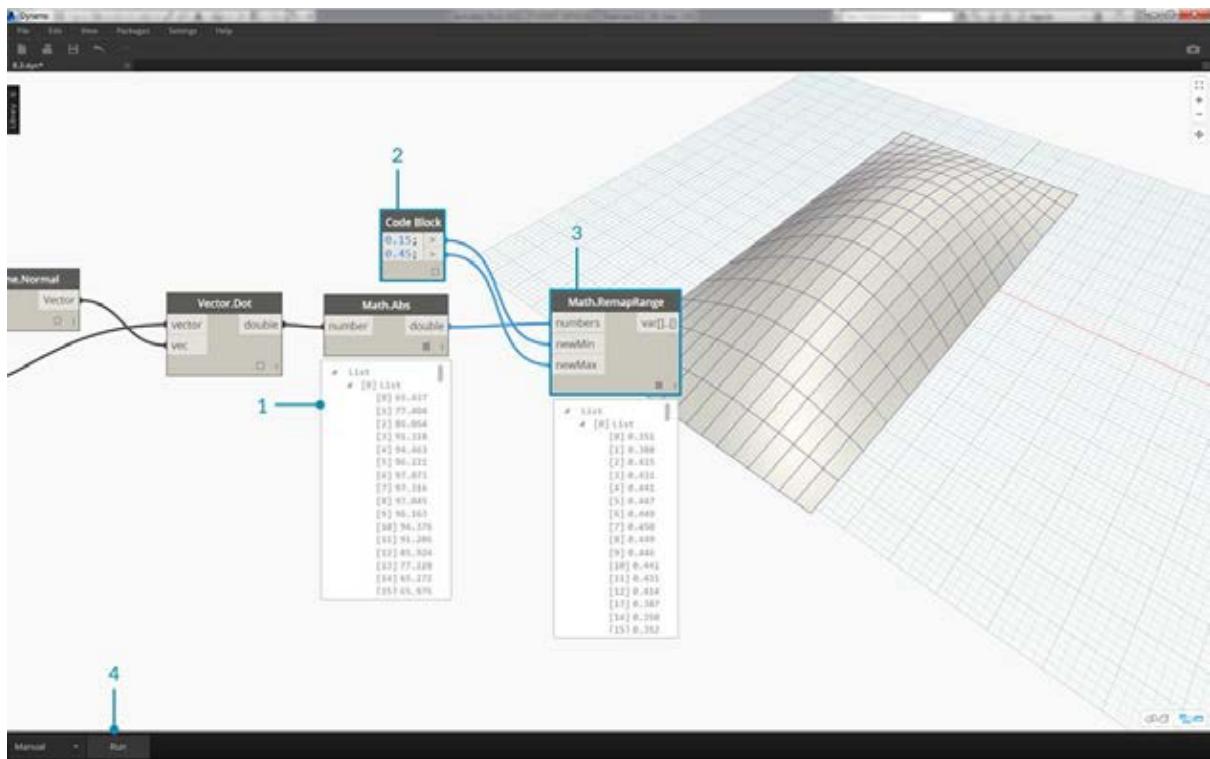
- Als Nächstes entwickeln Sie dies weiter, um die Öffnung der einzelnen Elemente in Abhängigkeit von der Sonneneinstrahlung an seiner Position zu steuern. Wenn Sie die Darstellung in Revit vergrößern und eines der Elemente auswählen, wird in der Eigenschaftenleiste der Parameter *Aperture Ratio* angezeigt. Die Familie wurde so eingerichtet, dass Öffnungsgrade zwischen ungefähr 0.05 und 0.45 möglich sind.



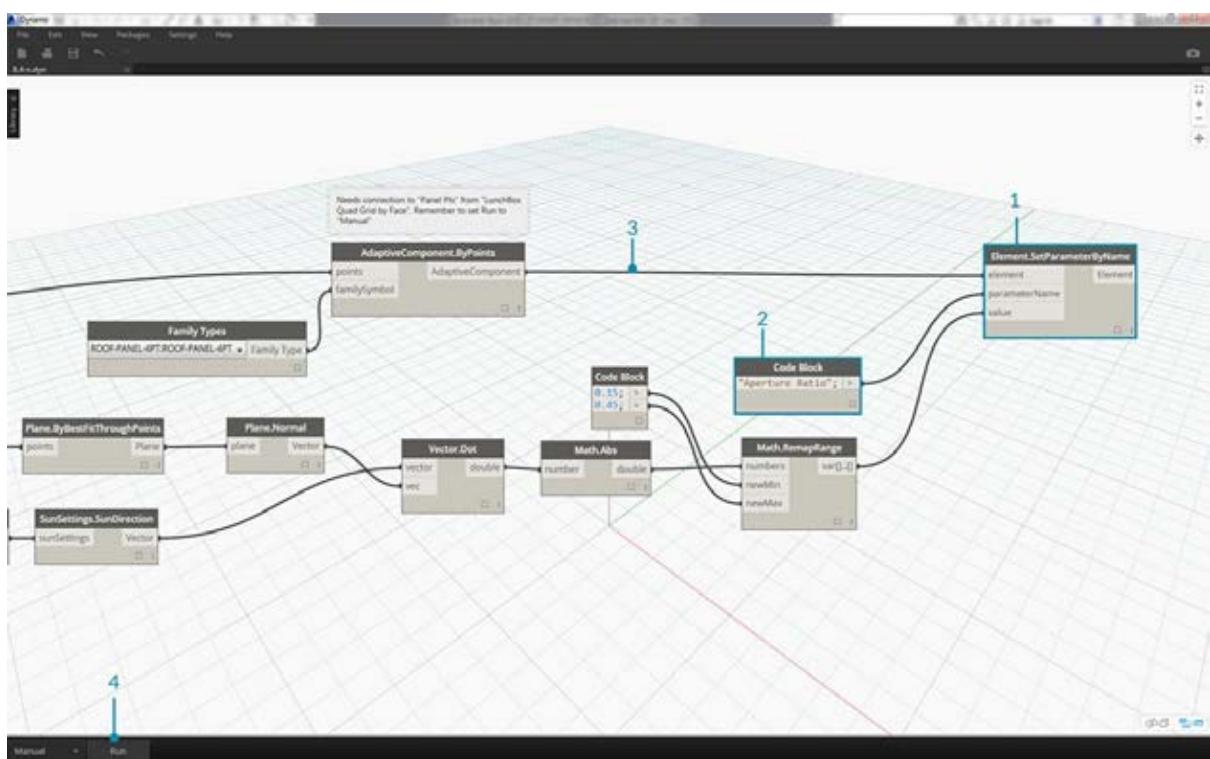
- Wenn Sie die Sonnenbahn aktivieren, wird der aktuelle Sonnenstand in Revit angezeigt.



1. Diesen Sonnenstand können Sie mithilfe des *SunSettings.Current*-Blocks referenzieren.
2. Verbinden Sie die Sonneneinstellungen mit *Sunsetting.SunDirection*, um den Solarvektor zu erhalten.
3. Erstellen Sie ausgehend von den *Panel Pts*, aus denen Sie die adaptiven Bauteile erstellt haben, mithilfe von *Plane.ByBestFitThroughPoints* annähernde Ebenen für die Bauteile.
4. Rufen Sie die *normal* für diese Ebene ab.
5. Berechnen Sie mithilfe des *Skalarproduks* die Sonnenrichtung. Das Skalarprodukt ist eine Formel zur Bestimmung der Parallelität oder Antiparallelität zweier Vektoren. Sie vergleichen also die Ebenennormale jedes einzelnen adaptiven Bauteils mit dem Solarvektor, um eine ungefähre Simulation des Winkels gegenüber der Sonnenstrahlung zu erhalten.
6. Ermitteln Sie *absoluten* Wert des Ergebnisses. Dadurch wird die Richtigkeit des Skalarprodukts für den Fall sichergestellt, dass die Ebenennormale in die entgegengesetzte Richtung zeigt.
7. Klicken Sie auf *Ausführen*.

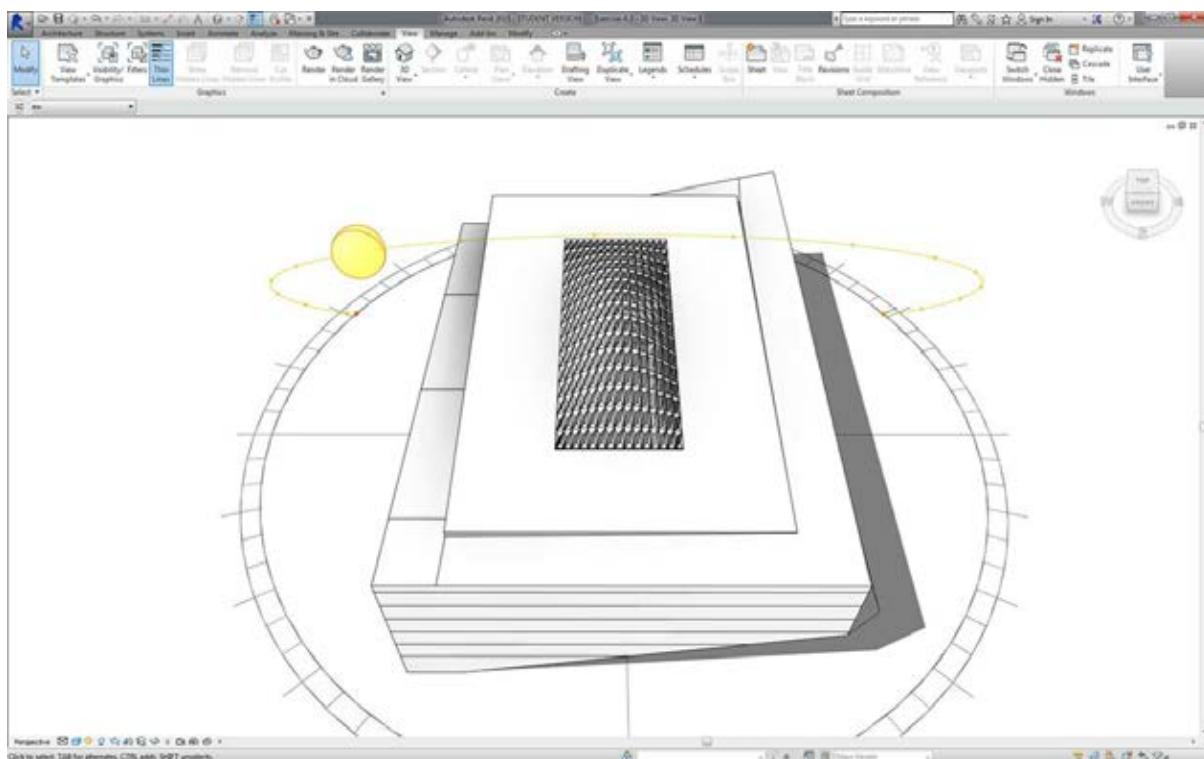


1. Für das Skalarprodukt wird ein großer Zahlenbereich ausgegeben. Die relative Verteilung dieser Zahlen soll verwendet werden, ihr Bereich muss jedoch auf den geeigneten Bereich für den Parameter *Aperture Ratio*, der bearbeitet werden soll, verdichtet werden.
2. Hierfür ist *Math.RemapRange* hervorragend geeignet. Diese Funktion übernimmt eine eingegebene Liste und ordnet ihre Grenzwerte zwei Zielwerten zu.
3. Definieren Sie die Zielwerte 0.15 und 0.45 in einem *Code Block*.
4. Klicken Sie auf *Ausführen*.

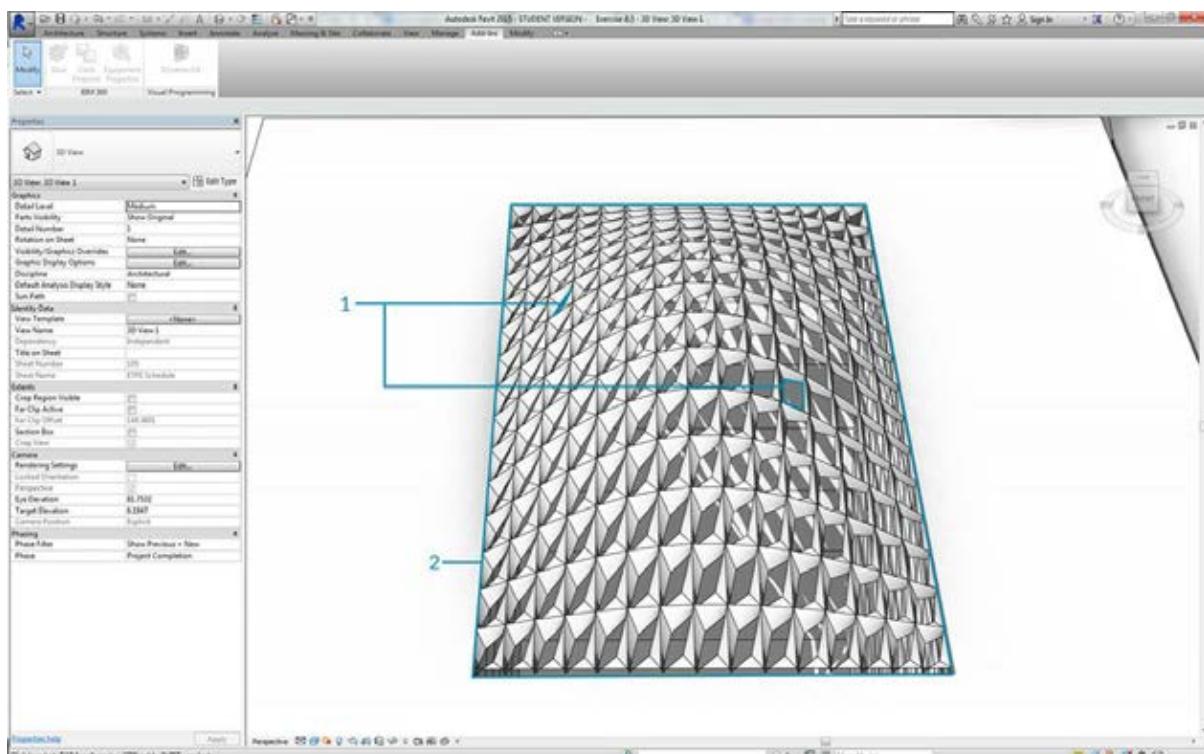


1. Verbinden Sie die neu zugeordneten Werte mit einem *Element.SetParameterByName*-Block.
2. Verbinden Sie die Zeichenfolge "Aperture Ratio" mit der *parameterName*-Eingabe.
3. Verbinden Sie die adaptiven Bauteile mit der *element*-Eingabe.

4. Klicken Sie auf Ausführen.



In Revit ist die Auswirkung des Sonnenwinkels auf die Öffnung der EFTE-Elemente auch aus größerer Entfernung zu erkennen.



Wenn Sie die Ansicht vergrößern, ist zu sehen, dass die der Sonne zugewandten EFTE-Elemente stärker geschlossen sind. In diesem Fall soll eine Überhitzung durch Sonneneinstrahlung vermieden werden. Wenn Sie den Lichteinfall in Abhängigkeit von der Sonneneinstrahlung steuern wollten, könnten Sie dafür einfach die Domäne im *Math.RemapRange*-Block ändern.

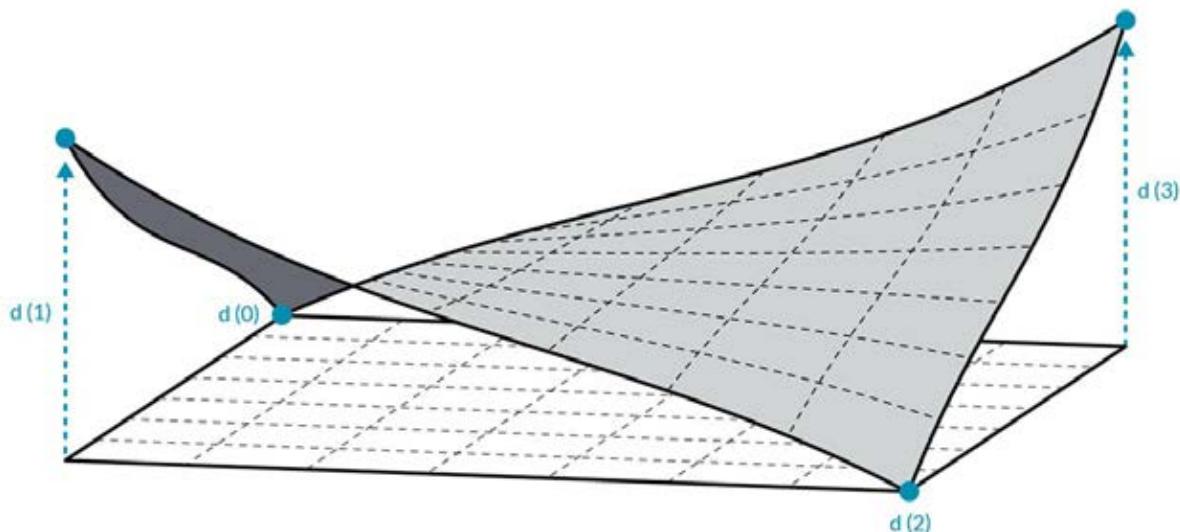
# Dokumentation

## Dokumentation

Die Bearbeitung von Parametern für die Dokumentation schließt sich an die in den vorigen Abschnitten behandelten Aufgaben an. In diesem Abschnitt bearbeiten Sie Parameter, mit deren Hilfe Sie nicht die geometrischen Eigenschaften von Elementen steuern, sondern eine Revit-Datei für die Konstruktionsunterlagen erstellen können.

### Abweichung

In der folgenden Übung verwenden Sie einen einfachen Block für die Abweichung von der Ebene, um einen Revit-Plan für die Dokumentation zu erstellen. Die einzelnen Elemente der parametrisch definierten Dachkonstruktion weisen unterschiedliche Abweichungswerte auf. Die Werte innerhalb dieses Bereichs sollen mithilfe von Farben gekennzeichnet und die adaptiven Punkte in einer Bauteilliste ausgegeben werden, sodass die Daten an einen Fassadenspezialisten, Bauingenieur oder Subunternehmer weitergegeben werden können.



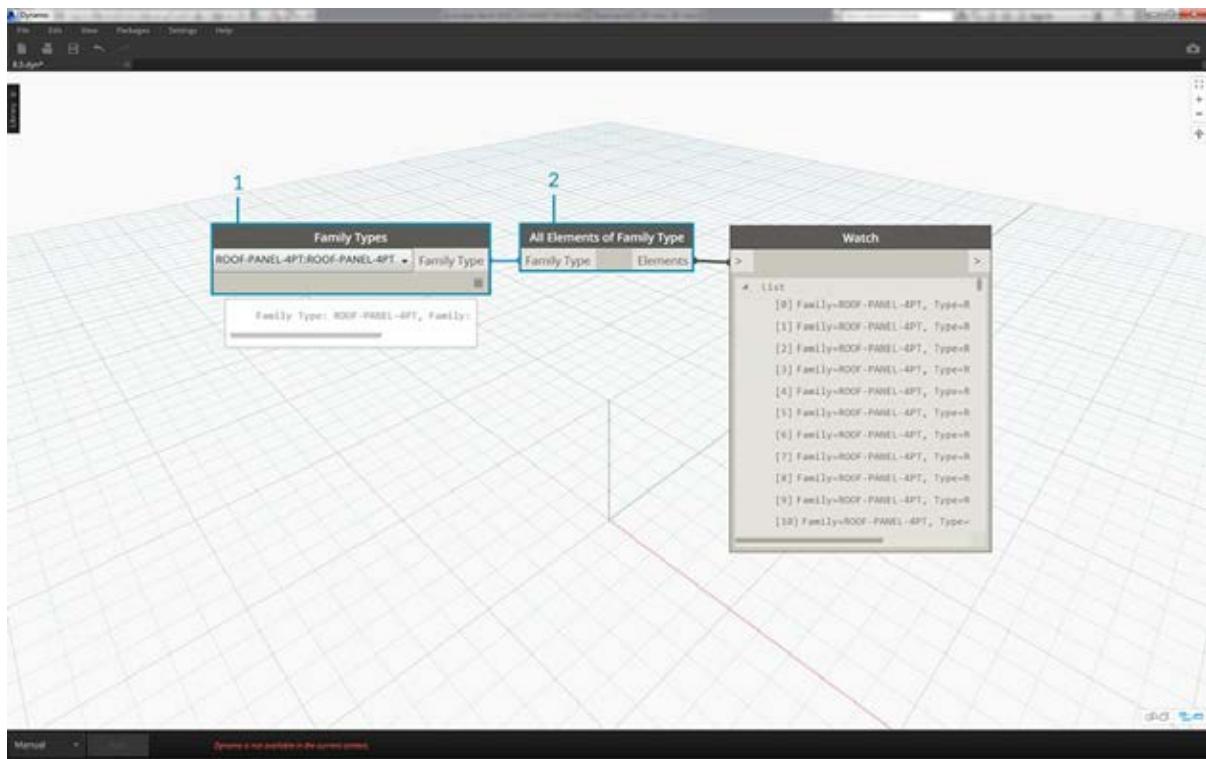
Der Block für die Abweichung von der Ebene berechnet, in welchem Grad die Gruppe aus vier Punkten von der optimalen Ebene zwischen ihnen abweicht. Dies ist eine schnelle und einfache Möglichkeit zur Untersuchung der Realisierbarkeit.

### Übungslektion

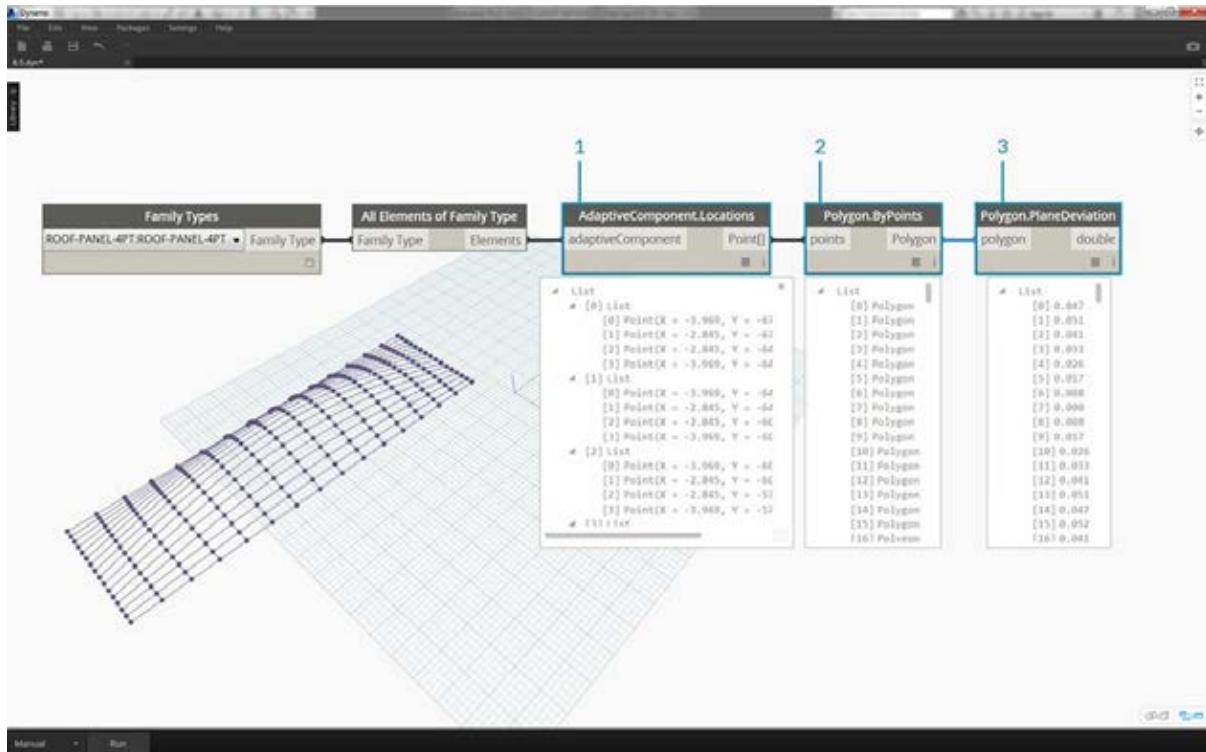
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.

1. [Documenting.dyn](#)
2. [ARCH-Documenting-BaseFile.rvt](#)

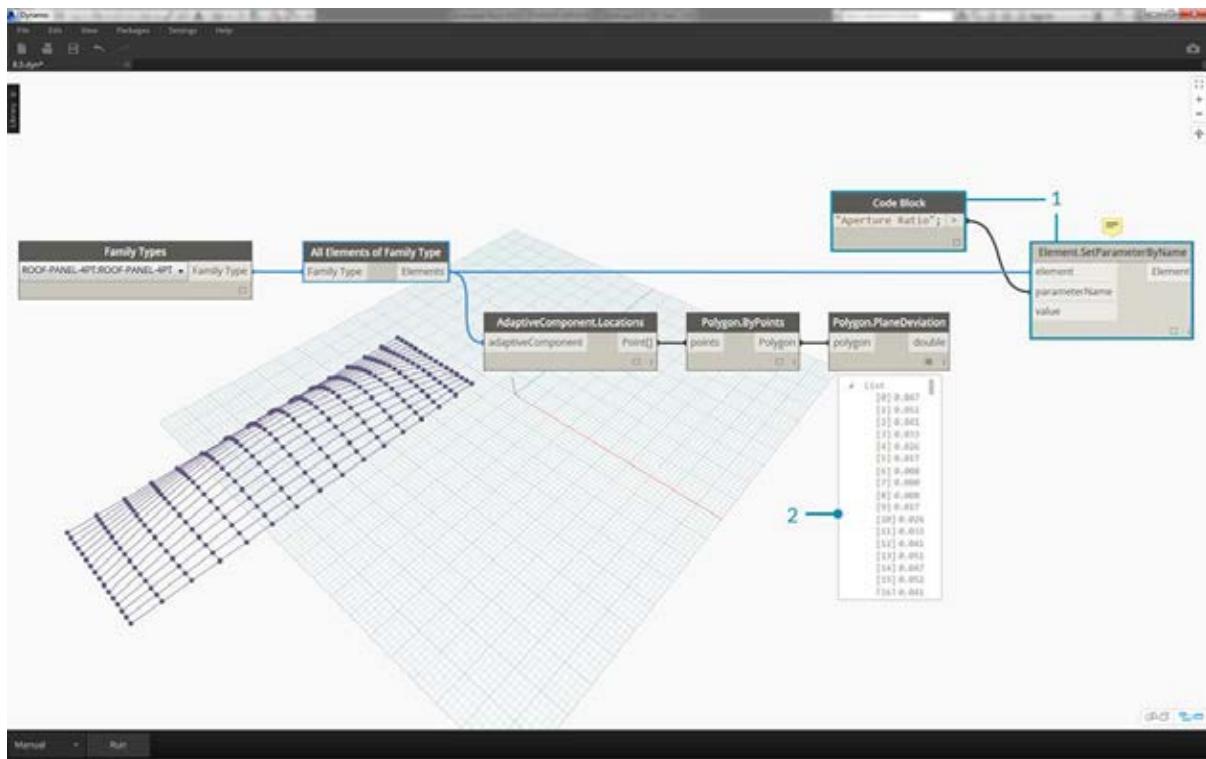
Beginnen Sie mit der Revit-Datei für diesen Abschnitt (oder verwenden Sie weiterhin die Datei aus dem vorigen Abschnitt). Diese Datei zeigt eine Gruppe von EFTE-Elementen auf dem Dach. Diese Elemente werden in dieser Übung referenziert.



1. Fügen Sie einen *Family Types*-Block in den Ansichtsbereich ein und wählen Sie "ROOF-PANEL-4PT".
2. Verbinden Sie diesen Block mit einem *All Elements of Family Type*-Block, um alle diese Elemente auszuwählen und aus Revit in Dynamo zu übernehmen.

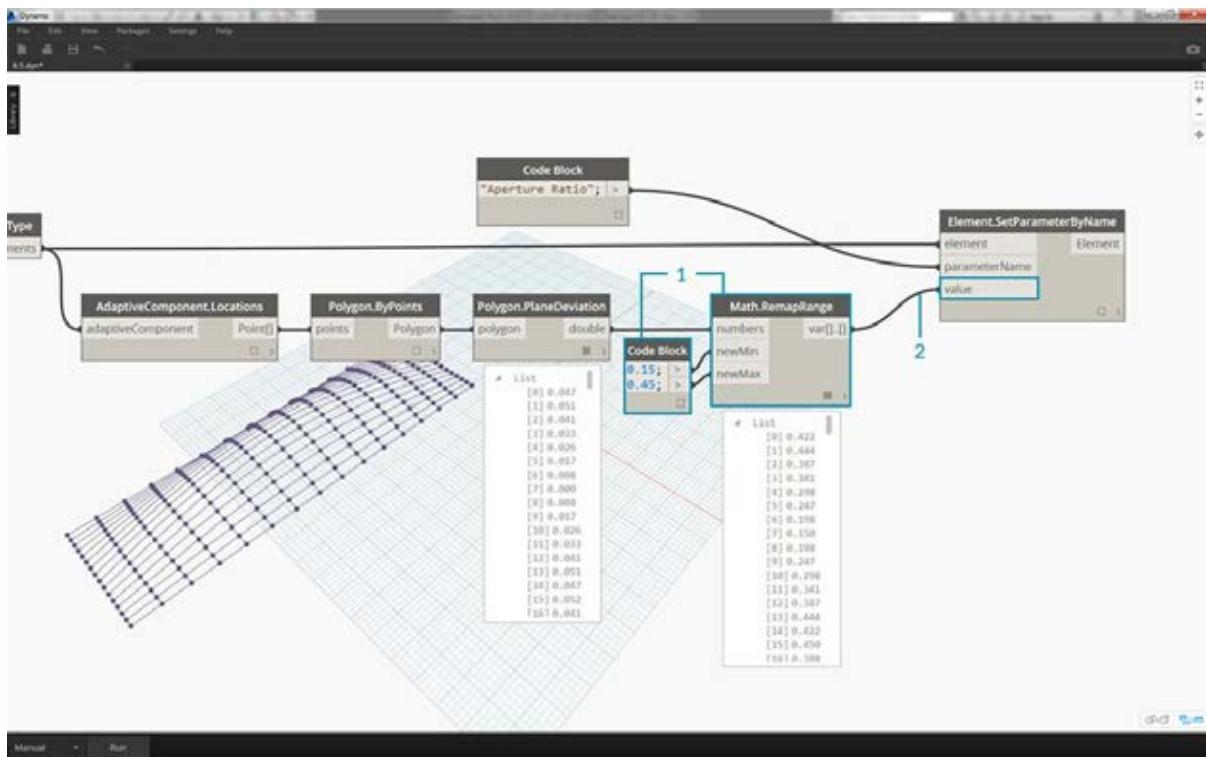


1. Rufen Sie die Positionen der adaptiven Punkte für die einzelnen Elemente mithilfe eines *AdaptiveComponent.Locations*-Blocks ab.
2. Erstellen Sie mithilfe eines *Polygon.ByPoints*-Blocks Polygone jeweils aus den vier Eckpunkten. Dadurch erhalten Sie eine abstrakte Version des aus Einzelementen bestehenden Systems in Dynamo, ohne dass die vollständige Geometrie des Revit-Elements importiert werden muss.
3. Berechnen Sie die planare Abweichung mithilfe des *Polygon.PlanarDeviation*-Blocks.

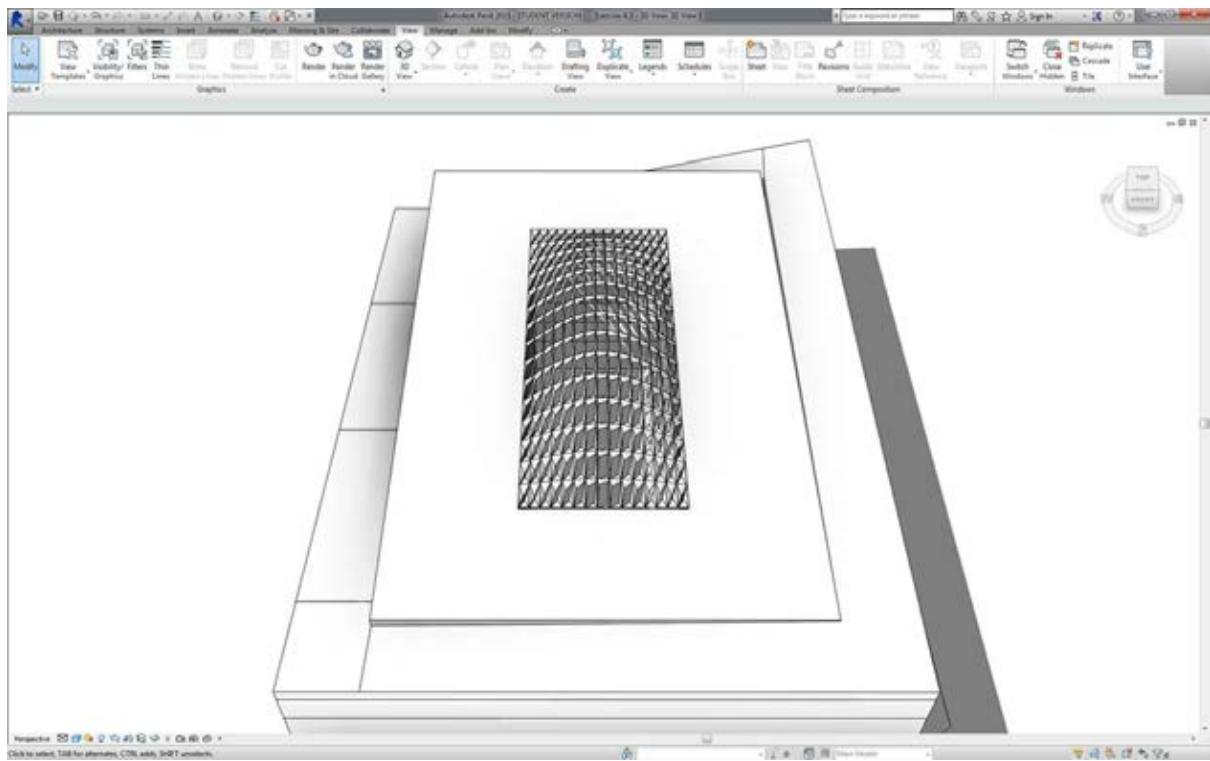


In der folgenden kurzen Zusatzübung legen Sie den *Öffnungsgrad* der einzelnen Elemente anhand ihrer planaren Abweichung fest.

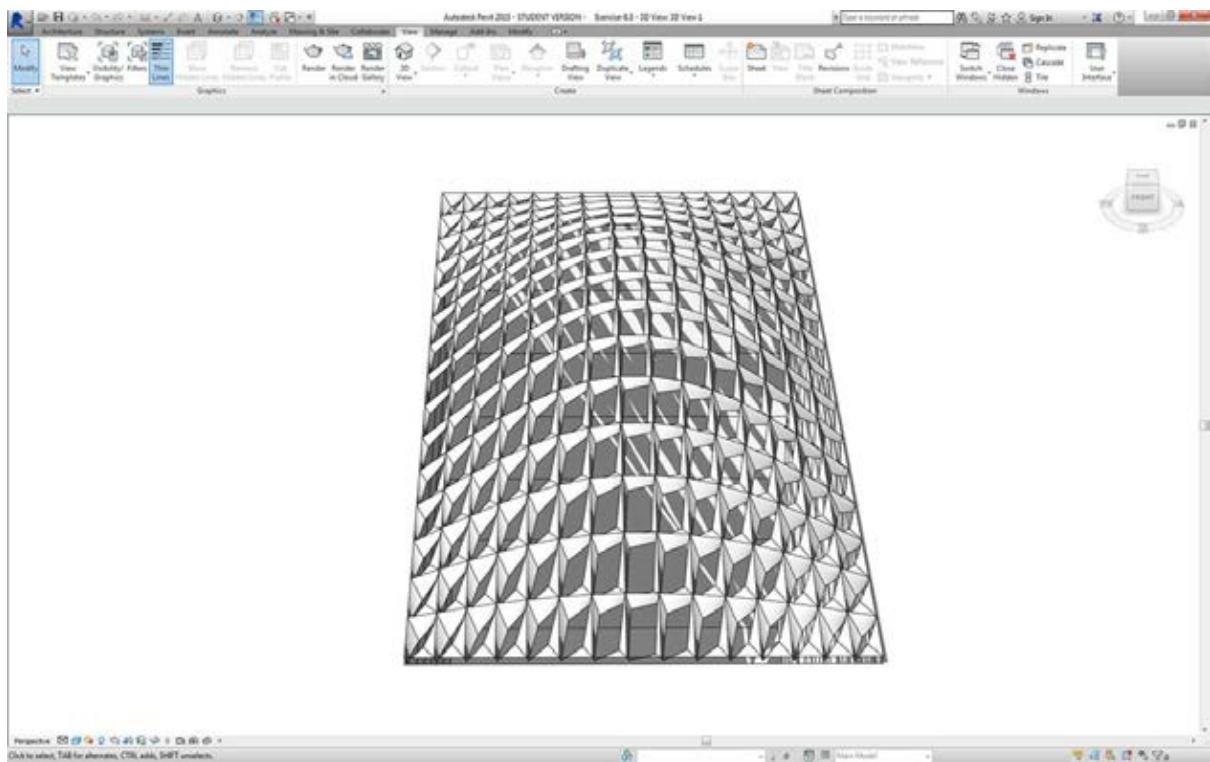
1. Fügen Sie im Ansichtsbereich einen *Element.SetParameterByName*-Block hinzu und verbinden Sie die adaptiven Bauteile mit der *element*-Eingabe. Verbinden Sie einen *Code Block* mit der Angabe "Aperture Ratio" mit der *parameterName*-Eingabe.
2. Die Ergebnisse der Abweichungsberechnung können nicht direkt mit der *value*-Eingabe verbunden werden, da die Werte zunächst dem Parameterbereich neu zugeordnet werden müssen.



1. Ordnen Sie mithilfe von *Math.RemapRange* die Abweichungswerte einer Domäne zwischen .15 und .45 zu.
2. Verbinden Sie diese Ergebnisse mit der *value*-Eingabe von *Element.SetParameterByName*.



In Revit ist die Veränderung der Öffnungen in der Oberfläche *ungefähr* zu erkennen.

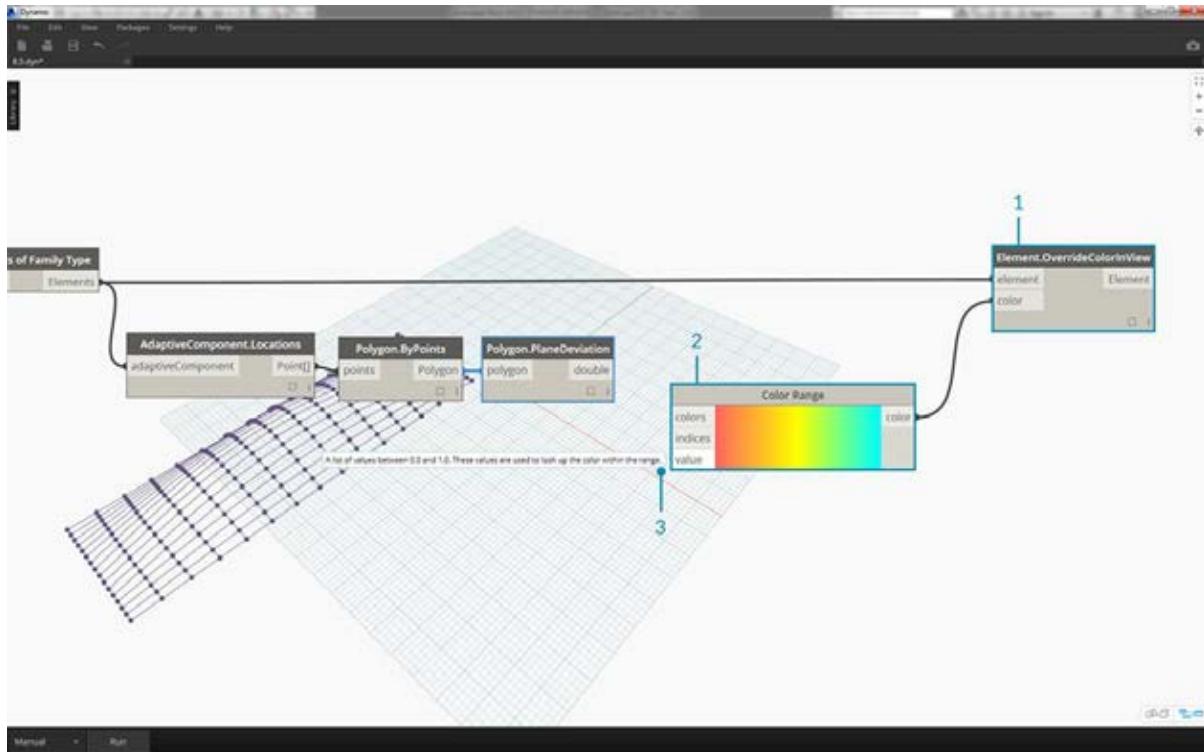


Die vergrößerte Darstellung zeigt deutlicher, dass die weniger weit geöffneten Elemente sich näher an den Ecken befinden. Die weit geöffneten Elemente befinden sich hingegen ganz oben. Die Eckbereiche weisen stärkere Abweichungen, der Scheitel der Wölbung dagegen die geringste Krümmung auf. Das Ergebnis ist daher überzeugend.

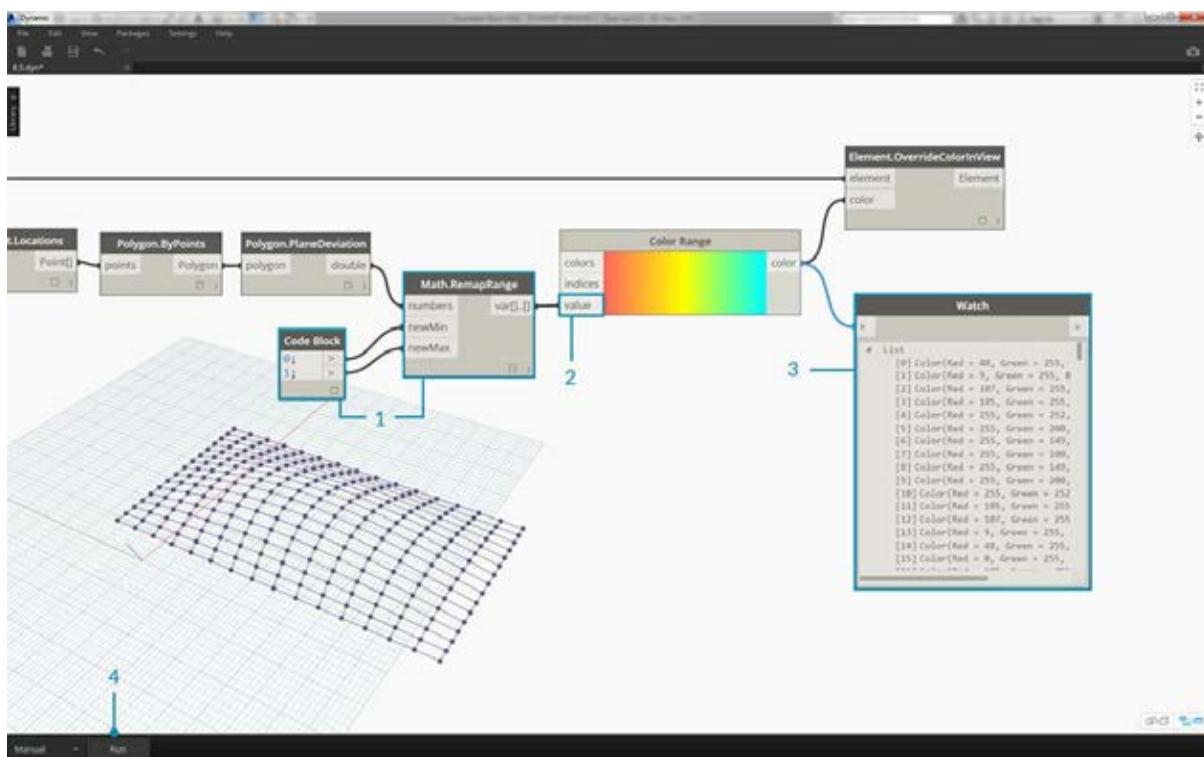
## **Farben und Dokumentation**

Durch Festlegen des Öffnungsgrads wird die Abweichung der Elemente des Dachs nicht präzise dargestellt. Zudem kommt es zu Änderungen an der Geometrie des Elements selbst. Angenommen, Sie möchten lediglich die Abweichung unter dem Gesichtspunkt der Realisierbarkeit analysieren. In diesem Fall wäre es hilfreich, für die Dokumentation die Elemente in

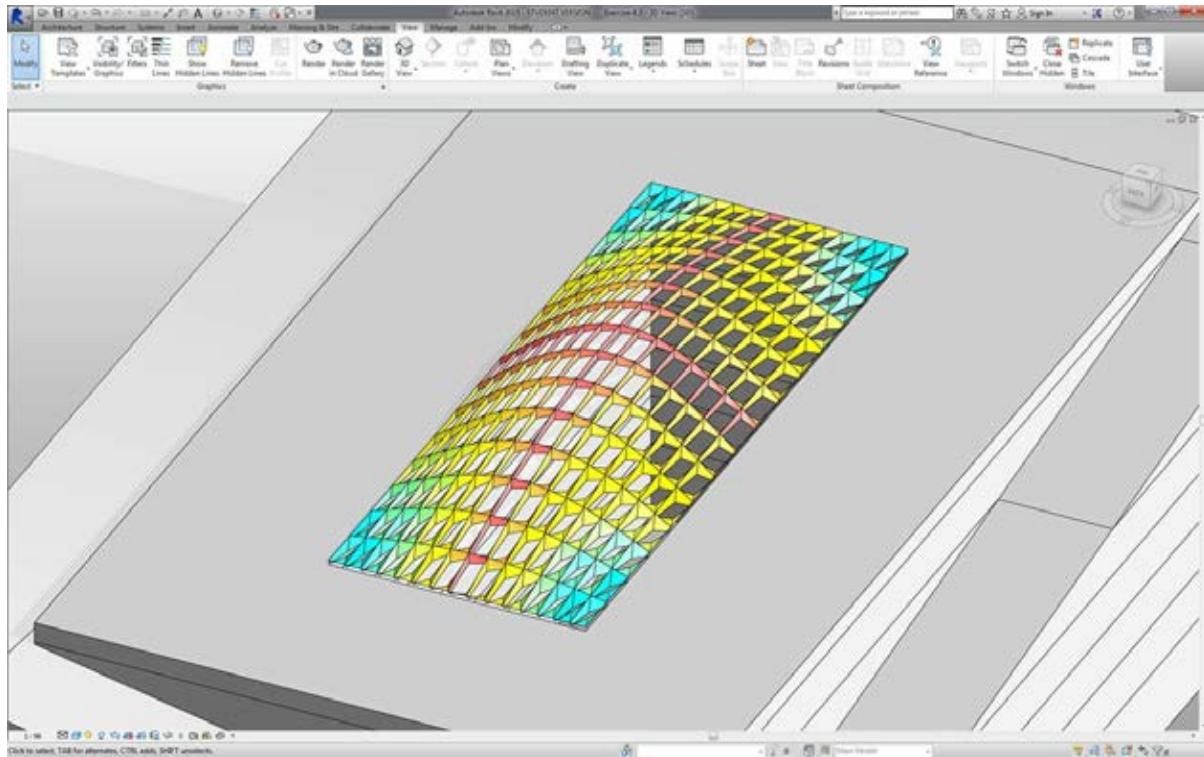
Abhängigkeit von Abweichungsbereichen farbig zu kennzeichnen. Dies ist mithilfe der folgenden Schritte möglich, wobei der Vorgang dem oben beschriebenen sehr ähnlich ist.



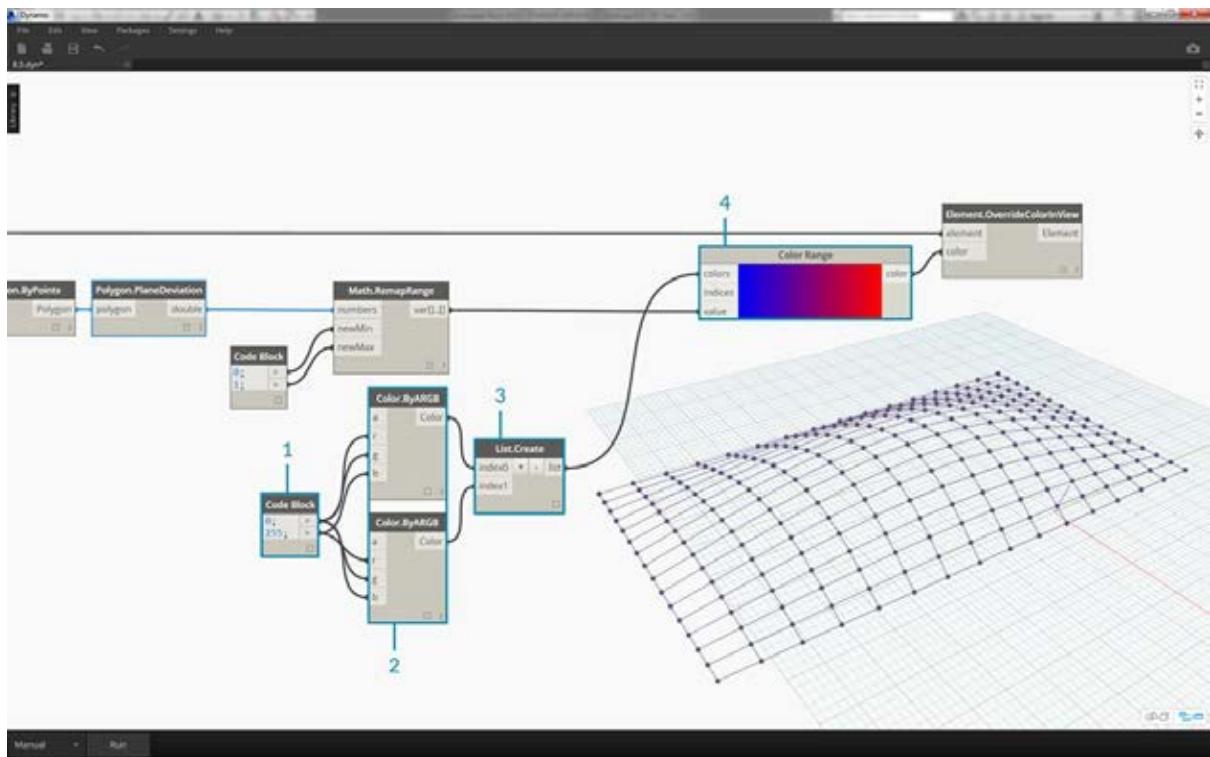
1. Entfernen Sie `Element.SetParameterByName` und die dazugehörigen Blöcke und fügen Sie `Element.OverrideColorInView` hinzu.
2. Fügen Sie im Ansichtsbereich einen `Color Range`-Block hinzu und verbinden Sie ihn mit der `color`-Eingabe von `Element.OverrideColorInView`. Um den Farbverlauf zu erstellen, müssen Sie noch die Abweichungswerte mit dem Farbbereich verbinden.
3. Wenn Sie den Mauszeiger auf die `value`-Eingabe setzen, wird angezeigt, dass Werte zwischen 0 und 1 für die Zuordnung von Farben zu Werten erforderlich sind. Sie müssen daher die Abweichungswerte für diesen Bereich neu zuordnen.



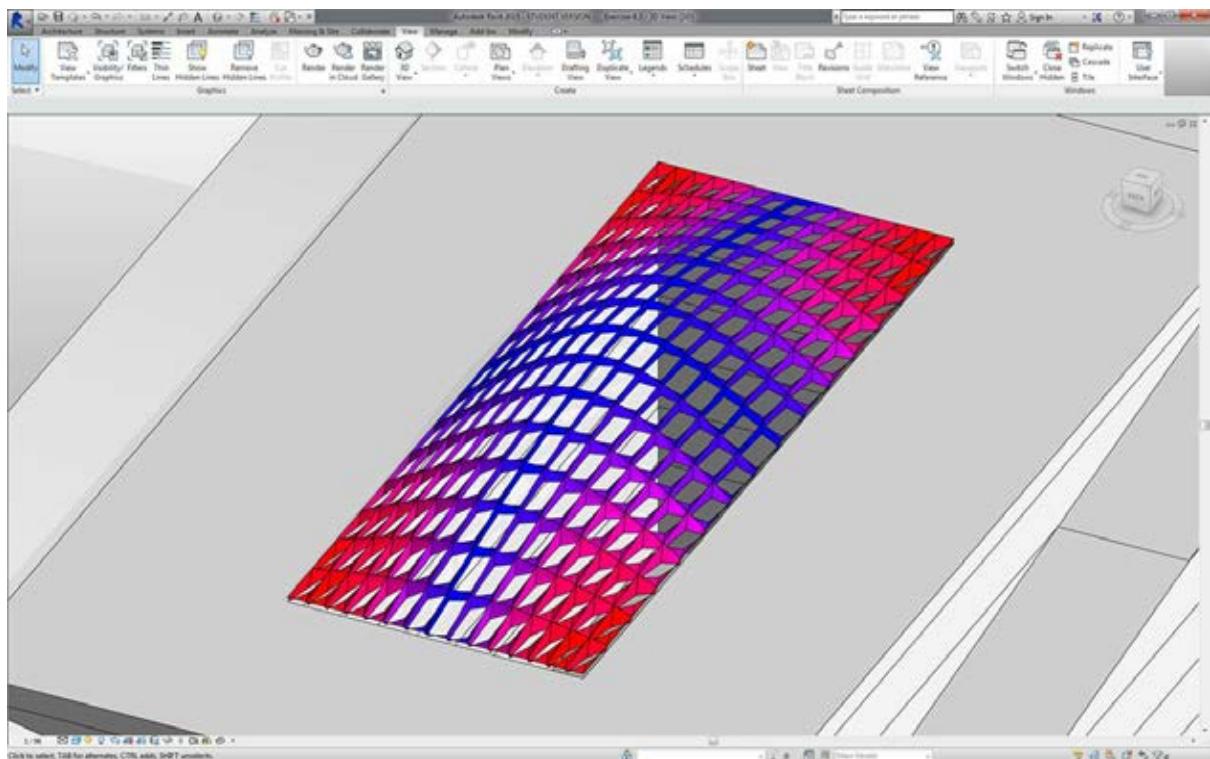
1. Ordnen Sie mithilfe von *Math.RemapRange* die Werte für die planare Abweichung dem Bereich zwischen 0 und 1 zu. (Anmerkung: Sie können die Quelldomäne auch mithilfe eines *MapTo*-Blocks definieren.)
2. Verbinden Sie die Ergebnisse mit einem *Color Range*-Block.
3. Dadurch erhalten Sie als Ausgabe einen Bereich von Farben anstelle eines Bereichs von Zahlen.
4. Falls Manuell eingestellt ist, klicken Sie auf *Ausführen*. Für den Rest dieses Vorgangs können Sie die Einstellung Automatisch verwenden.



In Revit ist jetzt eine wesentlich übersichtlichere Darstellung mit einem Farbverlauf zu sehen, der die planare Abweichung unter Verwendung des Farbbereichs zeigt. Die Farben müssen jedoch möglicherweise angepasst werden. Momentan werden die Werte für die kleinste Abweichung in Rot angezeigt: Dies scheint das Gegenteil des erwarteten Resultats zu sein. Stattdessen soll der Höchstwert der Abweichung in Rot dargestellt werden, während für die kleinste Abweichung eine ruhigere Farbe verwendet wird. Kehren Sie zu Dynamo zurück, um dies zu korrigieren.

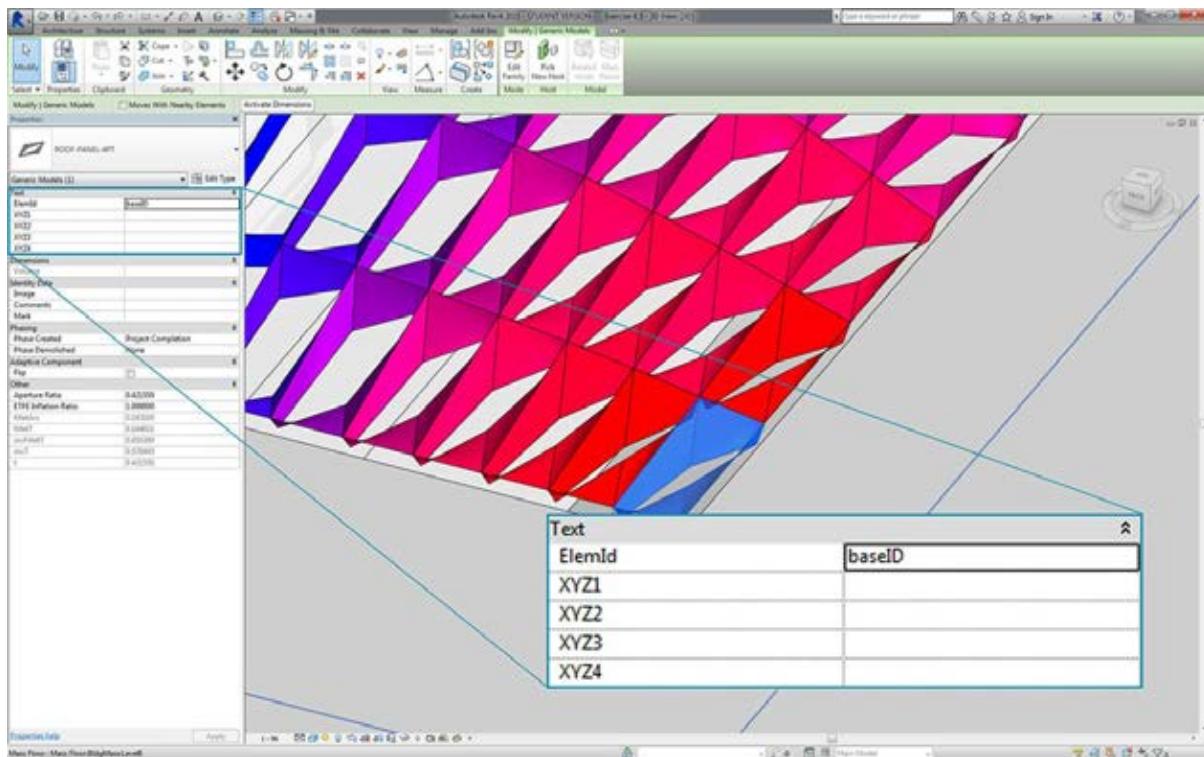


1. Geben Sie in einem *Code Block* zwei Zahlen in zwei getrennten Zeilen ein: 0 ; und 255 ;.
2. Erstellen Sie die Farben Rot und Blau, indem Sie die entsprechenden Werte mit zwei *Color.ByARGB*-Blöcken verbinden.
3. Erstellen Sie eine Liste aus diesen beiden Farben.
4. Verbinden Sie diese Liste mit der *colors*-Eingabe des *Color Range* und beobachten Sie die Aktualisierung dieses benutzerdefinierten Farbbereichs.

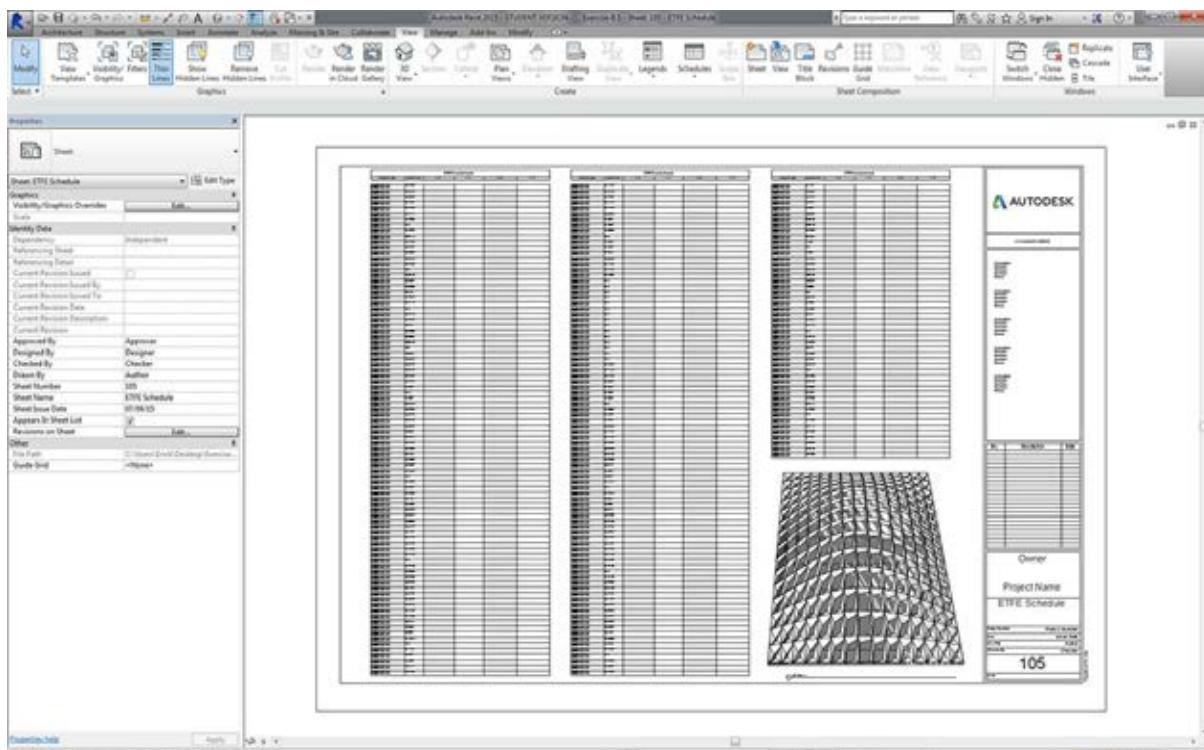


In Revit ist die Darstellung der Bereiche mit maximaler Abweichung in den Ecken jetzt besser verständlich. Mithilfe dieses Blocks werden Farben in einer Ansicht überschrieben. Aus diesem Grund kann es sehr hilfreich sein, wenn unter den Zeichnungen ein bestimmter Plan für einen bestimmten Analysetyp vorhanden ist.

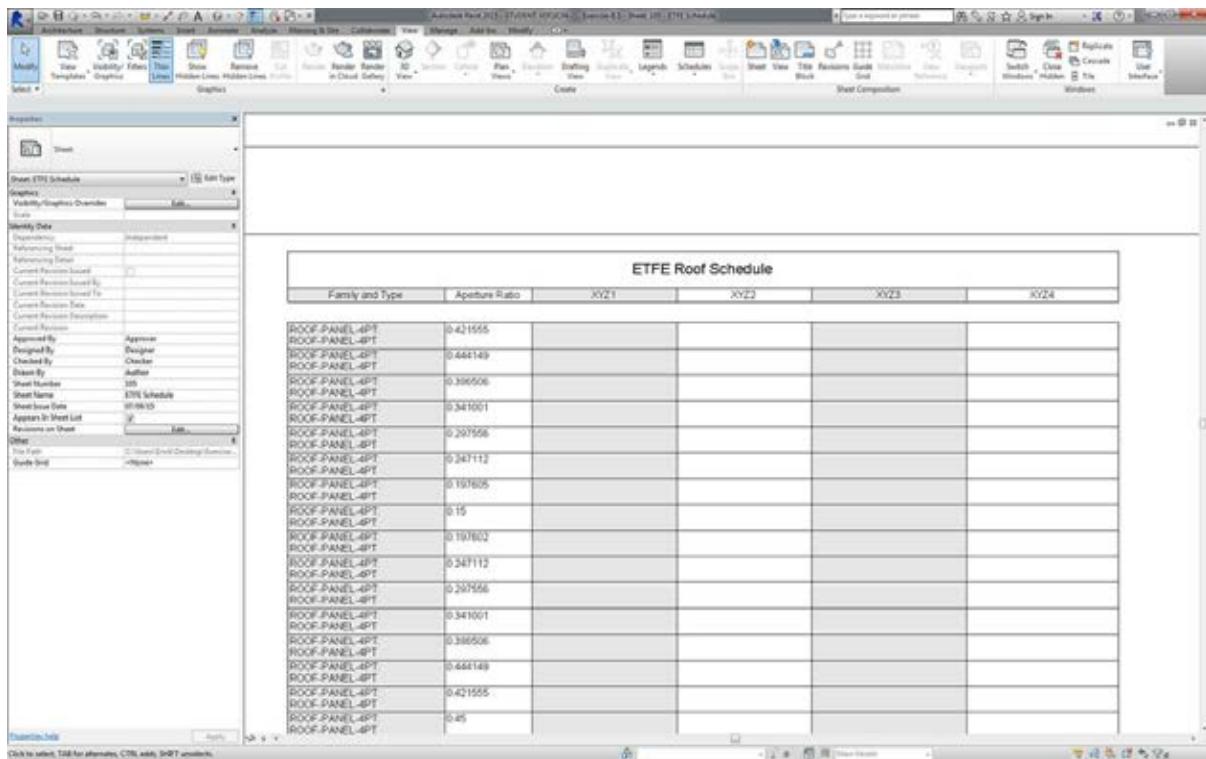
## Bauteillisten



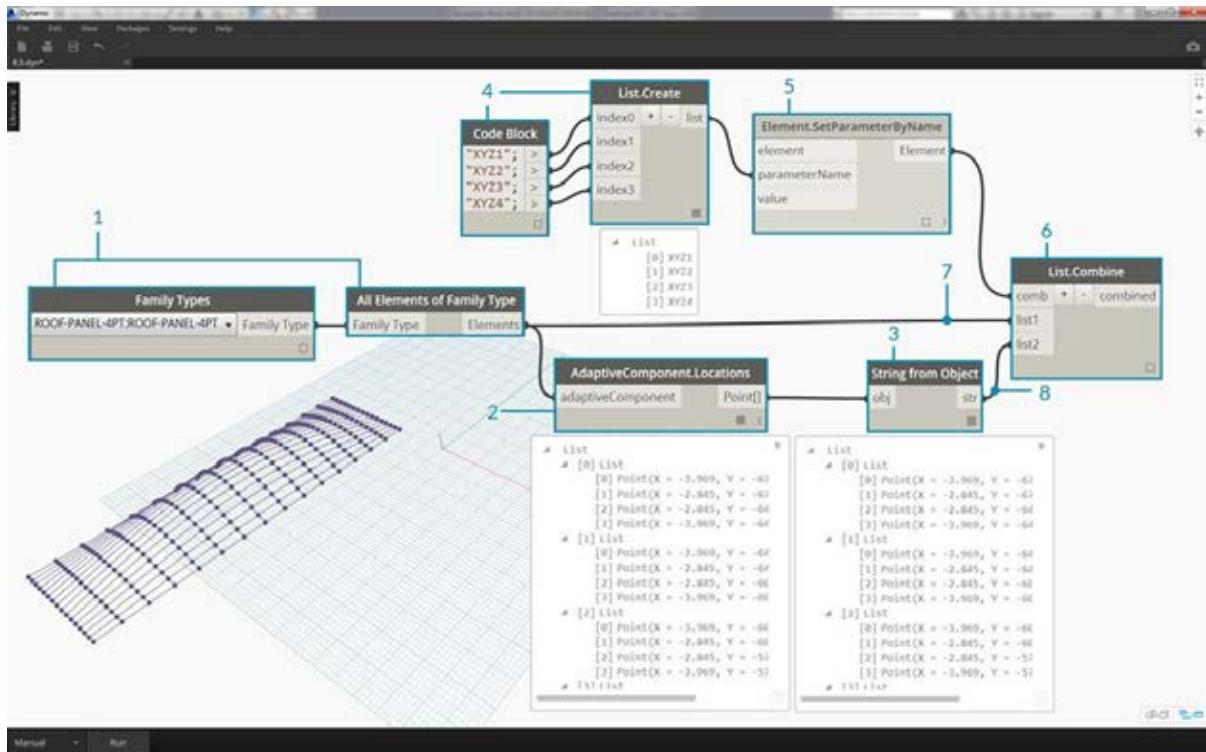
1. Wenn Sie eines der ETFE-Elemente in Revit auswählen, werden vier Exemplarparameter angezeigt: *XYZ1*, *XYZ2*, *XYZ3* und *XYZ4*. Diese sind nach korrekter Erstellung leer. Diese Parameter sind textbasiert und benötigen Werte. Sie schreiben mithilfe von Dynamo die Positionen der adaptiven Punkte in die einzelnen Parameter. Dies verbessert die Interoperabilität, falls die Geometrie an einen Bauingenieur oder Fassadenspezialisten weitergegeben werden soll.



Der Beispielplan zeigt eine große, leere Bauteilliste. Die XYZ-Parameter sind gemeinsam genutzte Parameter in der Revit-Datei und können daher in die Bauteilliste aufgenommen werden.



Die vergrößerte Darstellung zeigt, dass die XYZ-Parameter noch ausgefüllt werden müssen. Die ersten beiden Parameter werden von Revit vorgegeben.

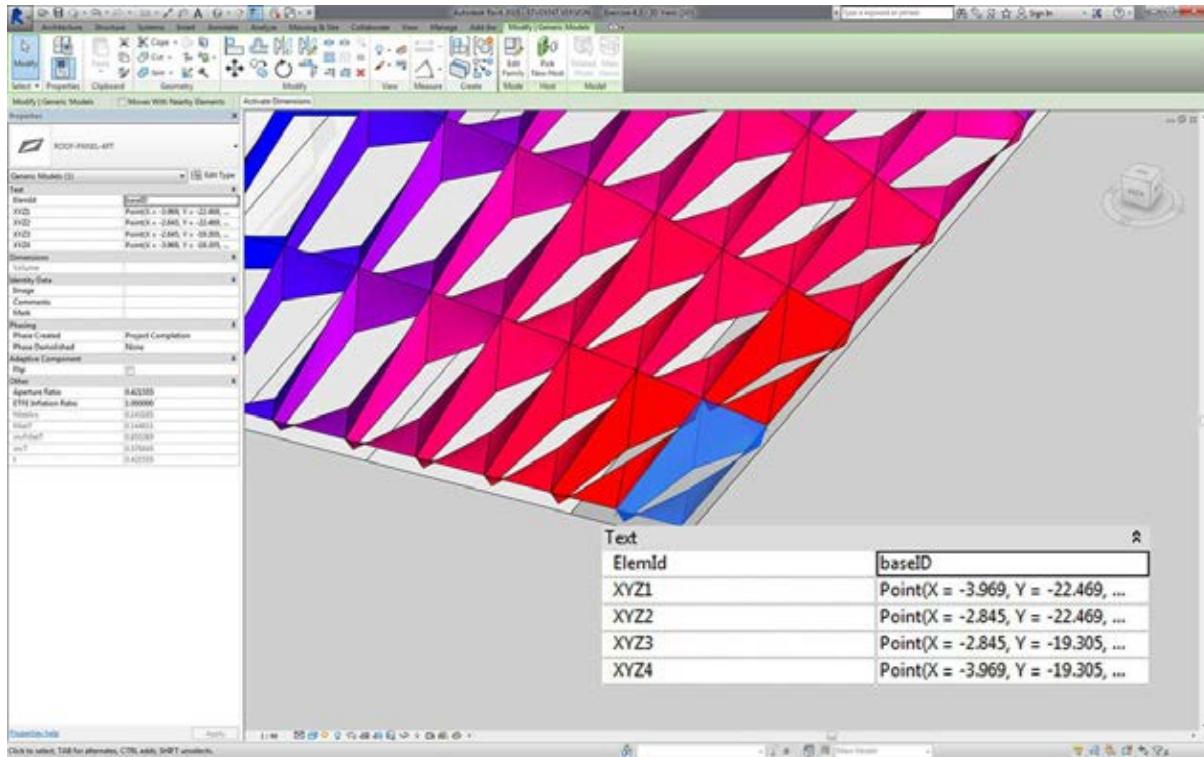


Um diese Werte zu schreiben, ist eine komplexe Listenoperation erforderlich. Das Diagramm selbst ist recht einfach, nutzt jedoch in großem Umfang die Listenzuordnung wie im Kapitel zu Listen beschrieben.

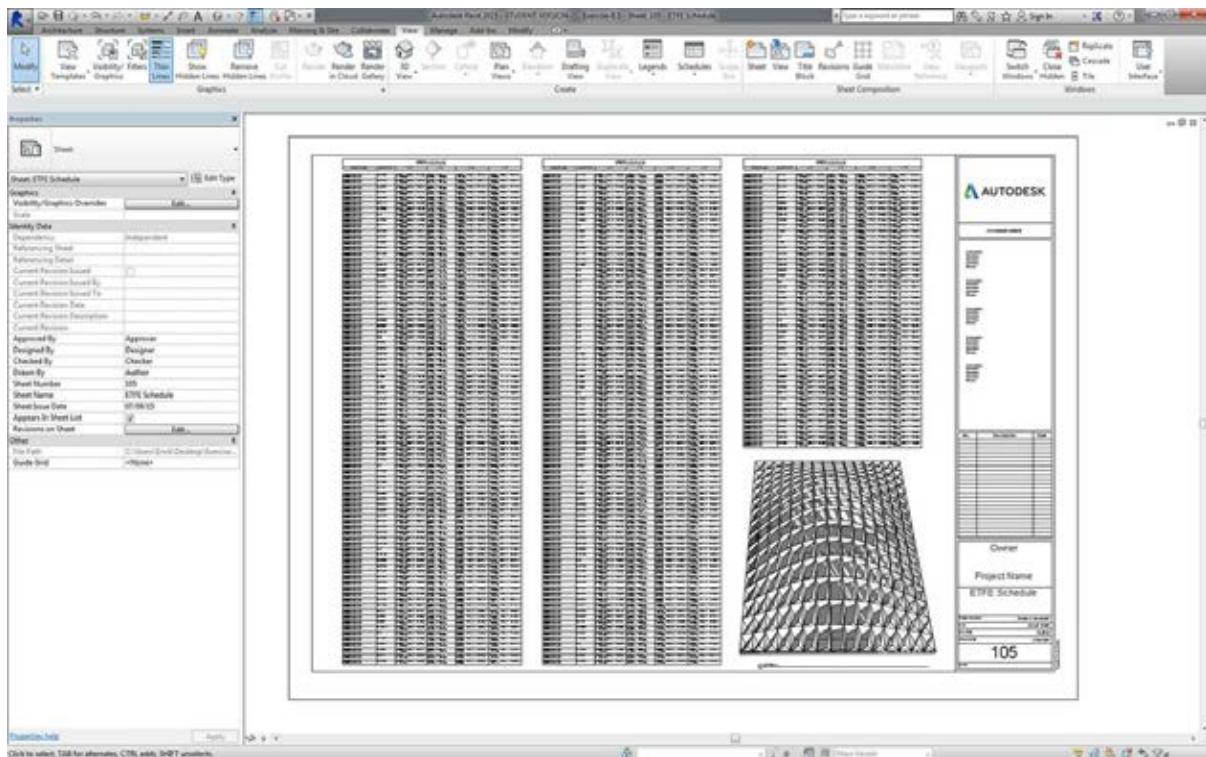
1. Wählen Sie mithilfe der oben gezeigten Blöcke alle adaptiven Bauteile aus.
2. Extrahieren Sie die Positionen der einzelnen Punkte mithilfe von `AdaptiveComponent.Locations`.
3. Konvertieren Sie diese Punkte in Zeichenfolgen. Beachten Sie, dass Sie hier textbasierte Parameter verwenden, und achten Sie darauf, den richtigen Datentyp einzugeben.
4. Erstellen Sie eine Liste mit den vier Zeichenfolgen, die die zu ändernden Parameter definieren: `XYZ1`, `XYZ2`,

*XYZ3* und *XYZ4*.

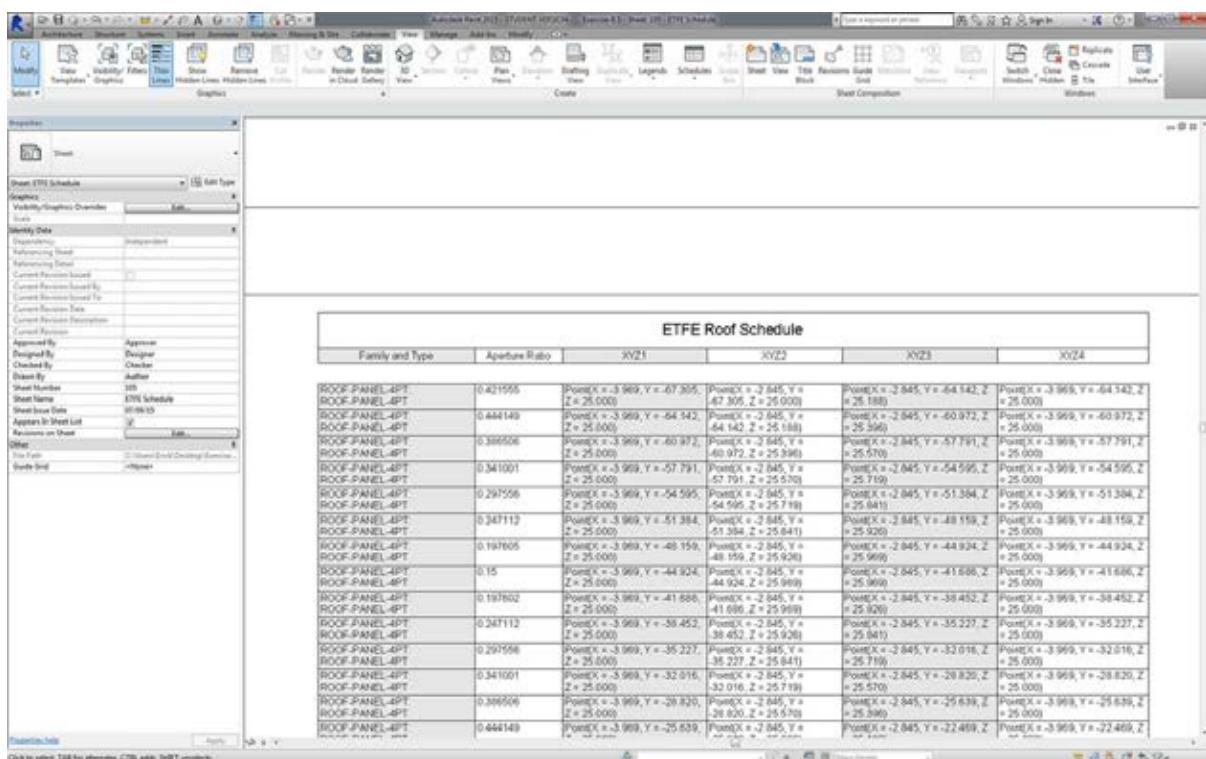
5. Verbinden Sie diese Liste mit der *parameterName*-Eingabe von *Element.SetParameterByName*.
6. Verbinden Sie *Element.SetParameterByName* mit der *combinator*-Eingabe von *List.Combine*.
7. Verbinden Sie die *adaptiven Bauteile* mit *list1*.
8. Verbinden Sie *String from Object* mit *list2*.
9. In diesem Vorgang werden Listen zugeordnet: Für jedes Element werden vier Werte geschrieben, wodurch eine komplexe Datenstruktur entsteht. Der *List.Combine*-Block definiert eine Operation, die eine Stufe tiefer in der Datenhierarchie abläuft. Aus diesem Grund wurden die Eingaben für Element und Wert leer gelassen.  
*List.Combine* verbindet die Unterlisten aus seinen Eingaben mit den leeren Eingaben von  
*List.SetParameterByName* in der Reihenfolge, in der sie verbunden wurden.



Wenn Sie jetzt ein Element in Revit auswählen, werden die Werte der einzelnen Parameter als Zeichenfolgen angezeigt. In der Praxis würden Sie ein einfacheres Format zum Schreiben von Punkten (x, y, z) erstellen. Sie könnten dies mithilfe von Zeichenfolgenoperationen in Dynamo erreichen. Da dies jedoch den Rahmen dieses Kapitels sprengen würde, wird dieses Verfahren hier nicht behandelt.



Ansicht der Bauteilliste mit ausgefüllten Parametern



Für jedes EFTE-Element sind jetzt die xyz-Koordinaten der adaptiven Punkte an seinen Ecken angegeben und können für die Fertigung der Elemente verwendet werden.

## Wörterbücher in Dynamo

### Wörterbücher in Dynamo

Wörterbücher stehen für eine Sammlung von Daten, die in Bezug zu einem anderen Datenelement stehen, das als Schlüssel bezeichnet wird. Wörterbücher bieten die Möglichkeit, Daten zu suchen, zu löschen und in eine Sammlung einzufügen.

Im Prinzip ist ein Wörterbuch eine wirklich clevere Methode, etwas nachzuschlagen.

*Die Wörterbuch-Funktionalität ist schon seit einiger Zeit in Dynamo verfügbar, aber in Dynamo 2.0 haben Sie neue Möglichkeiten, diesen Datentyp zu verwalten.*

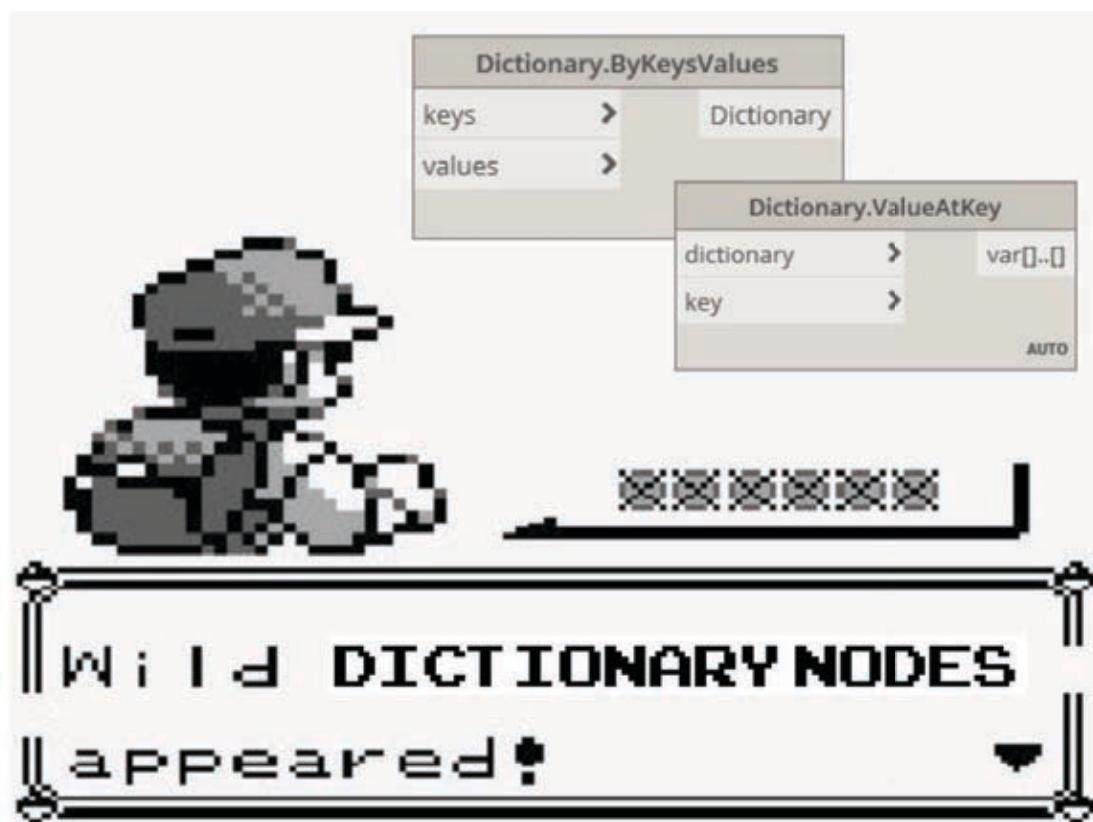


Abbildung mit freundlicher Genehmigung von [sixtysecondrevit.com](http://sixtysecondrevit.com)

# Wörterbücher

## Wörterbücher

Dynamo 2.0 führt das Konzept der Trennung des Datentyps Wörterbuch und des Datentyps Liste ein. Diese Neuerung bewirkt einige wichtige Änderungen hinsichtlich der Art und Weise, wie Sie Daten erstellen und in Ihren Arbeitsabläufen verwenden. Vor Version 2.0 waren Wörterbücher und Listen ein kombinierter Datentyp. Kurz gesagt: Listen waren eigentlich Wörterbücher mit ganzzahligen Schlüsseln.

- **Was ist ein Wörterbuch?**

Bei einem Wörterbuch handelt es sich um einen Datentyp, der aus einer Sammlung von Schlüssel-Wert-Paaren besteht. Jeder Schlüssel in einer Sammlung ist eindeutig. In einem Wörterbuch gibt es keine Reihenfolge. Im Prinzip "schlagen Sie Dinge nach", indem Sie einen Schlüssel anstelle eines Indexwerts in einer Liste verwenden. *In Dynamo 2.0 können Schlüssel nur Zeichenfolgen sein.*

- **Was ist eine Liste?**

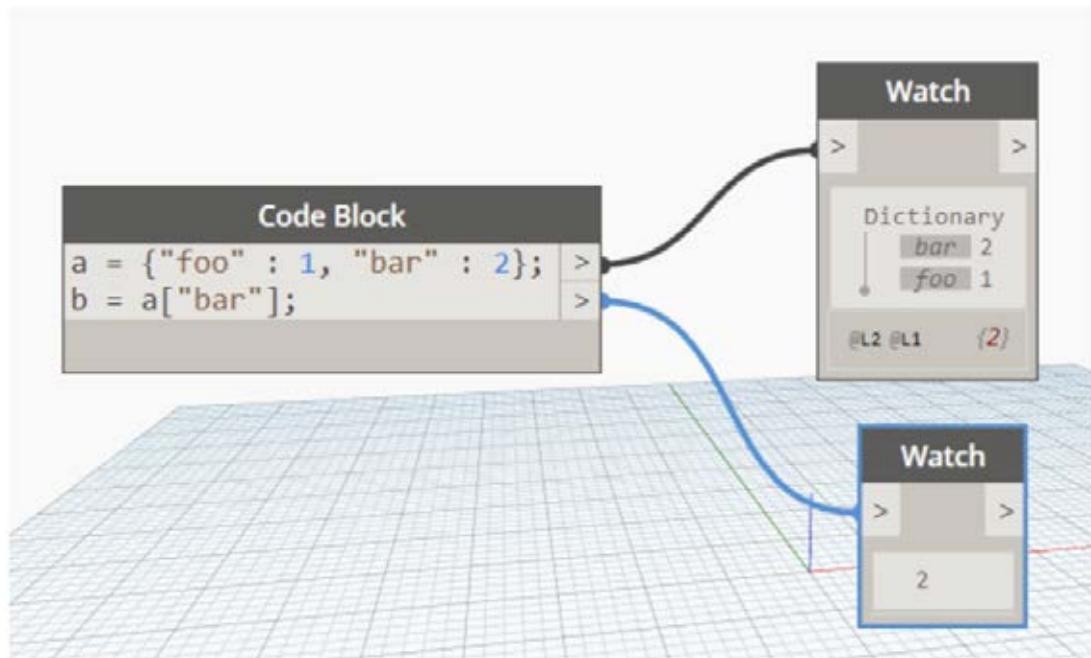
Bei einer Liste handelt es sich um einen Datentyp, der aus einer Sammlung von sortierten Werten besteht. In Dynamo verwenden Listen Ganzzahlen als Indexwerte.

- **Warum wurde diese Änderung vorgenommen und warum ist sie wichtig für mich?**

Durch die Trennung von Wörterbüchern und Listen werden Wörterbücher zu wichtigen Datentypen, mit denen Sie schnell und einfach Werte speichern und nachschlagen können, ohne die Indexwerte kennen oder eine exakte Listenstruktur durch Ihre Arbeitsabläufe beibehalten zu müssen. Während der Benutzertests sahen wir eine erhebliche Reduzierung der Diagrammgröße, wenn Wörterbüchern anstatt mehrerer `GetItemAtIndex`-Blöcke verwendet wurden.

- **Welche Änderungen wurden vorgenommen?**

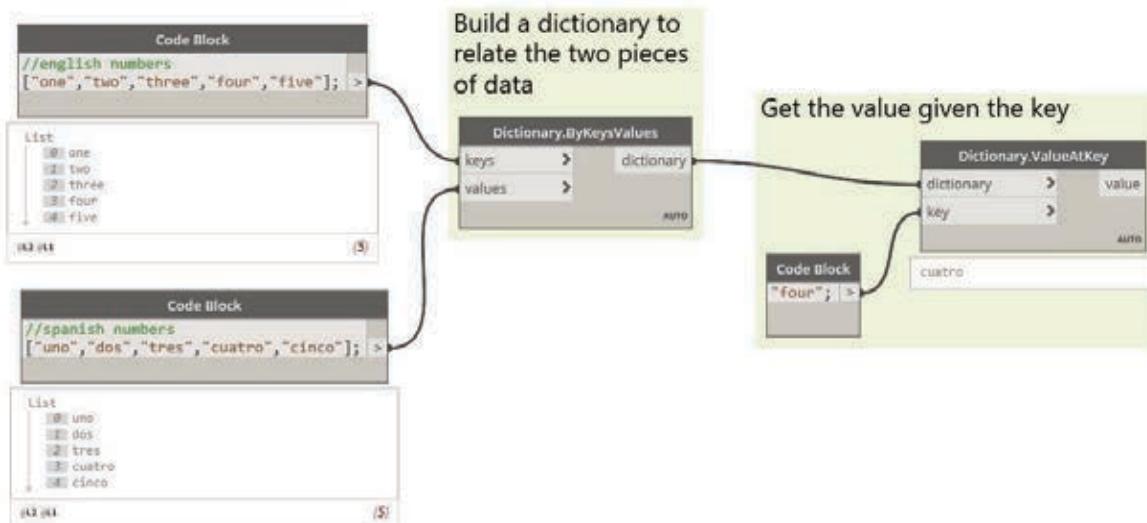
- Es wurden Änderungen an der *Syntax* vorgenommen. Die Initialisierung und Verwendung von Wörterbüchern und Listen in Codeblöcken funktioniert jetzt anders.
- Wörterbücher verwenden die folgende Syntax: `{schlüssel:wert}`
- Listen verwenden die folgende Syntax: `[wert,wert,wert]`
- Es wurden *neue Blöcke* in der Bibliothek eingeführt, mit denen Sie einfacher Wörterbücher erstellen, ändern und abfragen können.
- Listen, die in 1.x-Codeblöcken erstellt wurden, werden automatisch beim Laden des Skripts zur neuen Syntax migriert, die eckige `[ ]` anstelle von geschweiften Klammern `{ }` verwendet.



- Warum ist dies wichtig für mich? Wie würde ich diese Dateitypen verwenden?

In der Computerwissenschaft handelt es sich bei Wörterbüchern – wie bei Listen – um Sammlungen von Objekten. Während Listen jedoch in einer bestimmten Reihenfolge erstellt werden, sind die Sammlungen in Wörterbücher *nicht sortiert*. Es sind keine Nummernsequenzen (Indizes) erforderlich. Stattdessen werden *Schlüssel* verwendet.

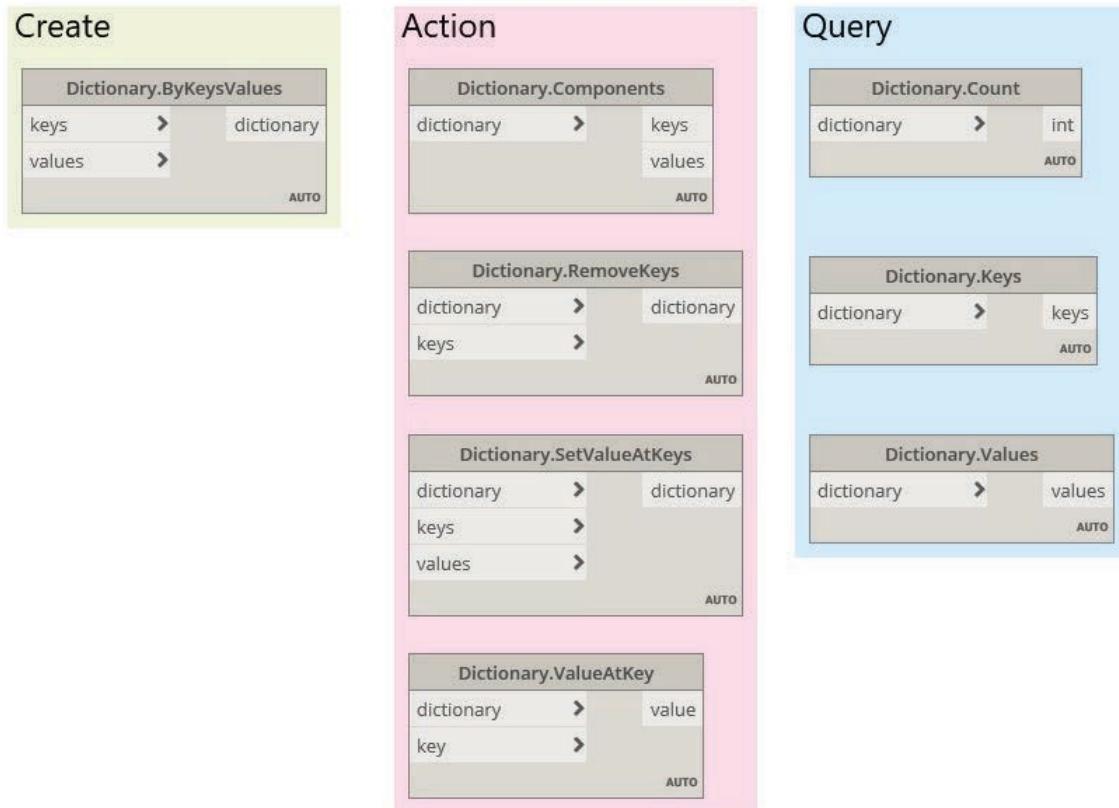
In der folgenden Abbildung sehen Sie eine mögliche Verwendung eines Wörterbuchs. Wörterbücher werden häufig genutzt, um zwei Teile von Daten zu verbinden, die vielleicht keine direkte Beziehung zueinander aufweisen. In unserem Fall verbinden wir die spanische Version eines Worts mit der englischen Version, so dass wir das Wort später nachschlagen können.



# Wörterbuch-Blöcke

## Wörterbuch-Blöcke

Dynamo 2.0 stellt eine Reihe von Wörterbuch-Blöcken für die Verwendung bereit. Dies umfasst die Blöcke *create*, *action* und *query*.



- **Dictionary.ByKeysValues** erstellt ein Wörterbuch mit den bereitgestellten Werten und Schlüsseln. (*Die Anzahl der Einträge entspricht der Länge der kürzesten Liste.*)
- **Dictionary.Components** erstellt die Komponenten des Eingabe-Wörterbuchs. (*Dieser Vorgang ist die Umkehrung der Block-Erstellung.*)
- **Dictionary.RemoveKeys** erzeugt ein neues Wörterbuch-Objekt und entfernt die Eingabe-Schlüssel.
- **Dictionary.SetValueAtKeys** erzeugt ein neues Wörterbuch anhand der eingegebenen Schlüssel und Werte und ersetzt den aktuellen Wert in den entsprechenden Schlüsseln.
- **Dictionary.ValueAtKey** gibt den Wert am Eingabeschlüssel zurück.
- **Dictionary.Count** gibt an, wie viele Schlüssel-Wert-Paare sich im Wörterbuch befinden.
- **Dictionary.Keys** gibt auch zurück, welche Schlüssel derzeit im Wörterbuch gespeichert sind.
- **Dictionary.Values** gibt zurück, welche Werte derzeit im Wörterbuch gespeichert sind.

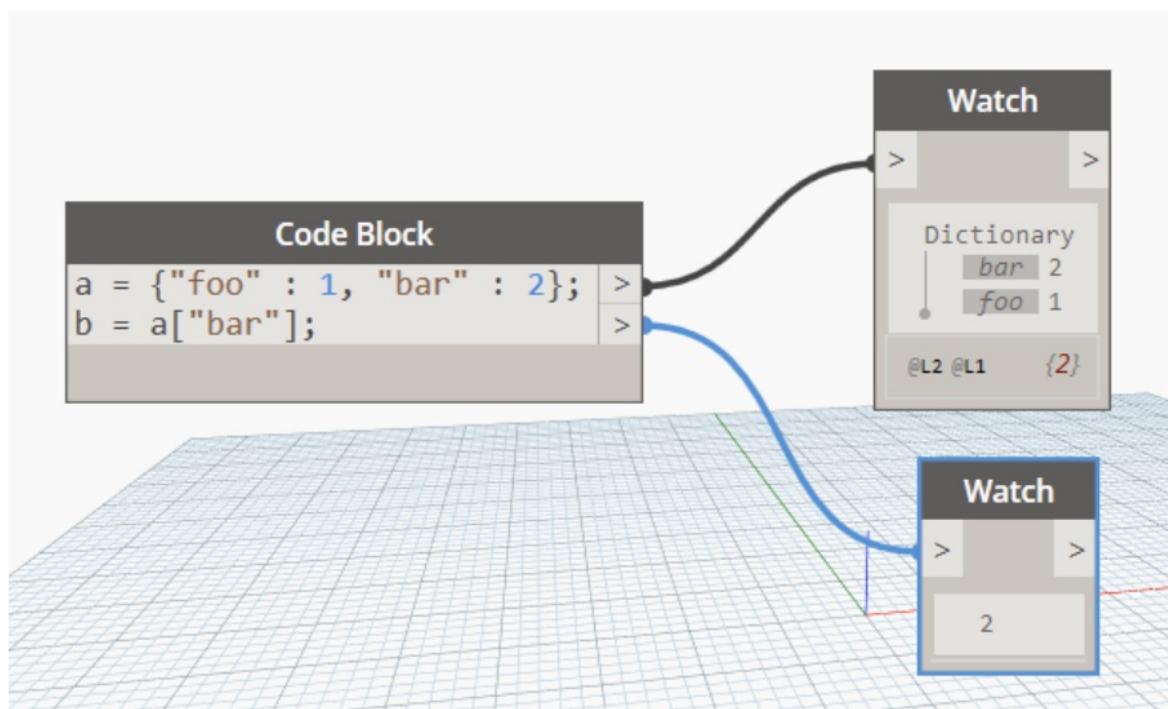
Die Möglichkeit, Daten allgemein mit Wörterbüchern in Beziehung zu setzen, ist eine großartige Alternative zur vorherigen Verwendung von Indizes und Listen.

# Wörterbücher in Codeblöcken

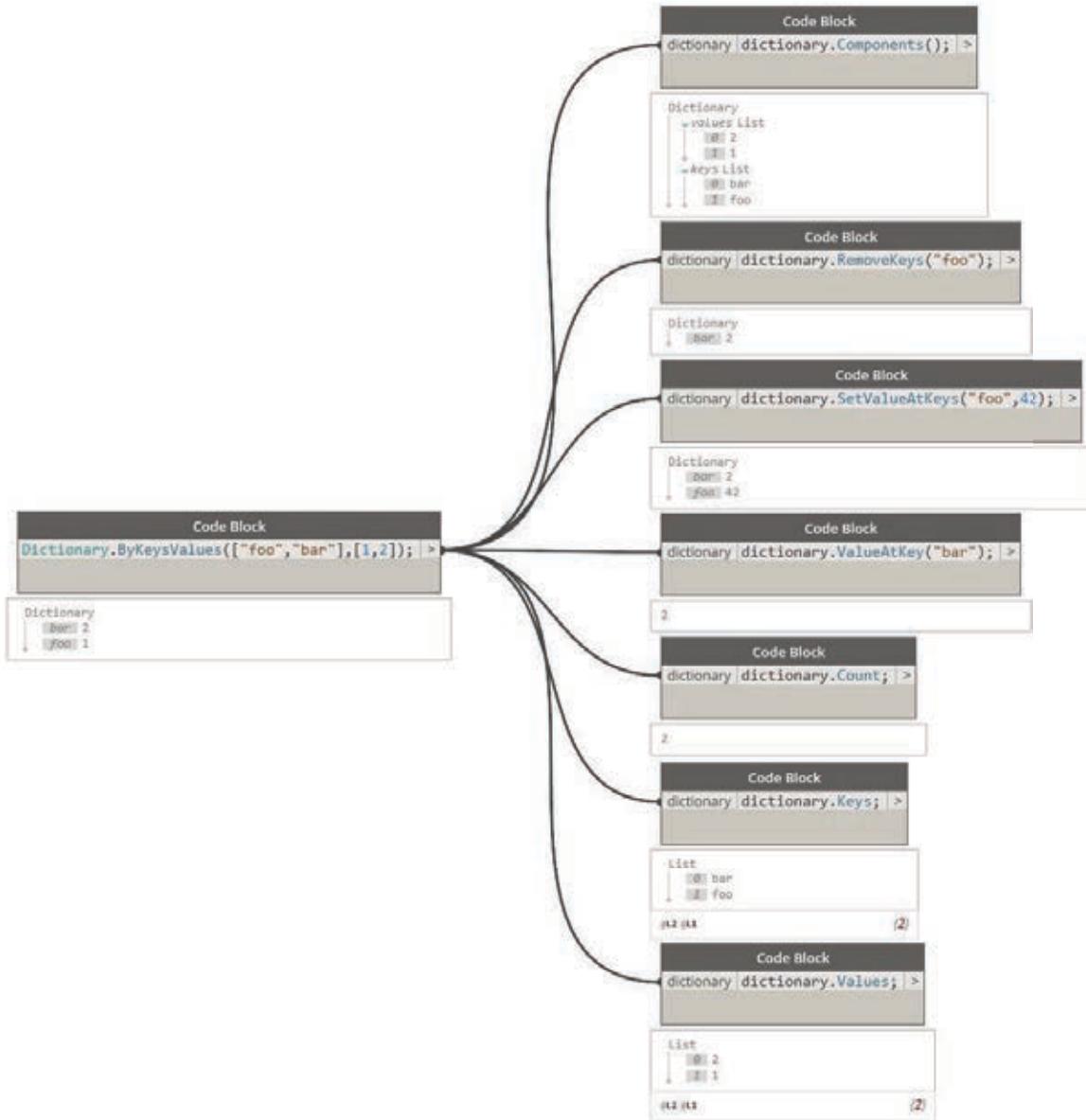
## Wörterbücher in Codeblöcken

Dynamo 2.0 stellt nicht nur die oben erwähnten Blöcke für Wörterbücher zur Verfügung, sondern gibt Codeblöcken auch neue Funktionen.

Sie können die im Folgenden beschriebene Syntax für Blöcke verwenden, oder sie auf DesignScript-Basis darstellen.



Da ein Wörterbuch ist in Dynamo ein Objekttyp ist, können wir die folgenden Aktionen damit ausführen.



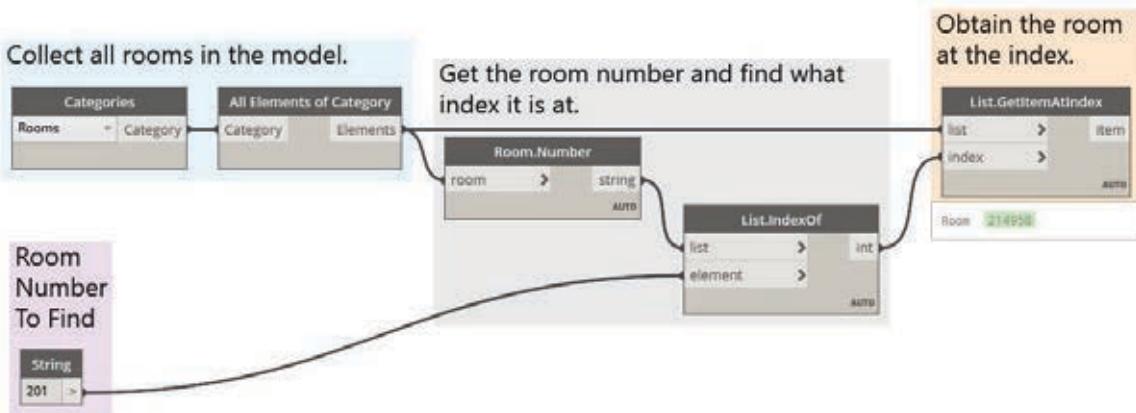
Diese Art der Interaktion ist besonders nützlich, um Revit-Daten mit Zeichenfolgen in Bezug zu setzen. Als Nächstes sehen wir uns einige Anwendungsfälle in Revit an.

# Wörterbücher – Revit-Anwendungsfälle

## Wörterbücher – Revit-Anwendungsfälle

Wollten Sie jemals etwas in Revit anhand eines Datenelements nachschlagen?

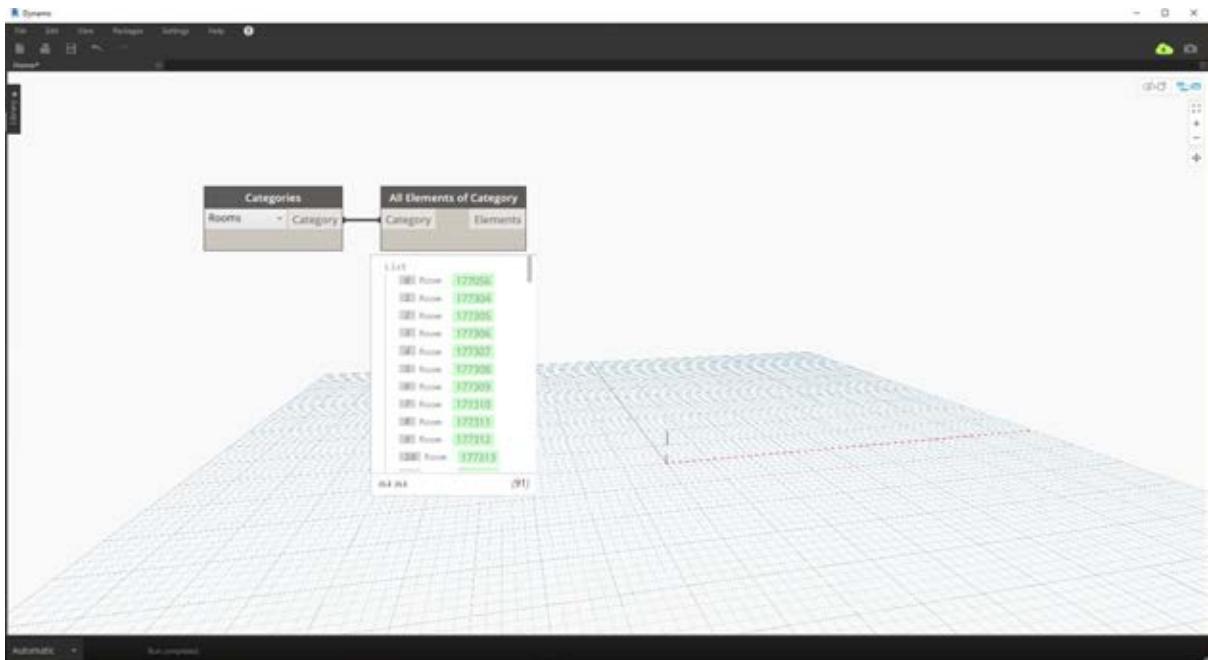
Wahrscheinlich haben Sie dazu etwa die folgenden Schritte ausgeführt:



In der Abbildung oben gezeigt, sammeln wir aller Räume im Revit-Modell, rufen den Index des gewünschten Raum (nach Raumnummer) ab, und erhalten schließlich die Indexnummer des Raums.

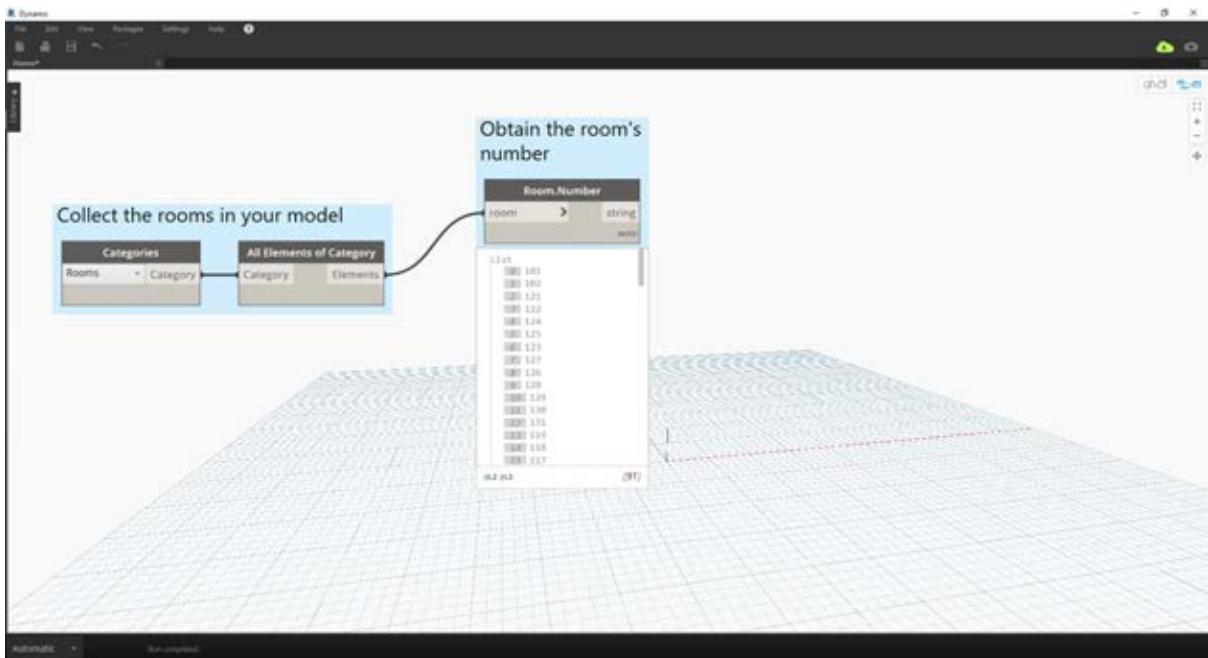
**Lassen Sie uns das nun mit Wörterbüchern probieren.**

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As"): [RoomDictionary.dyn](#). Eine vollständige Liste der Beispieldateien finden Sie im Anhang.



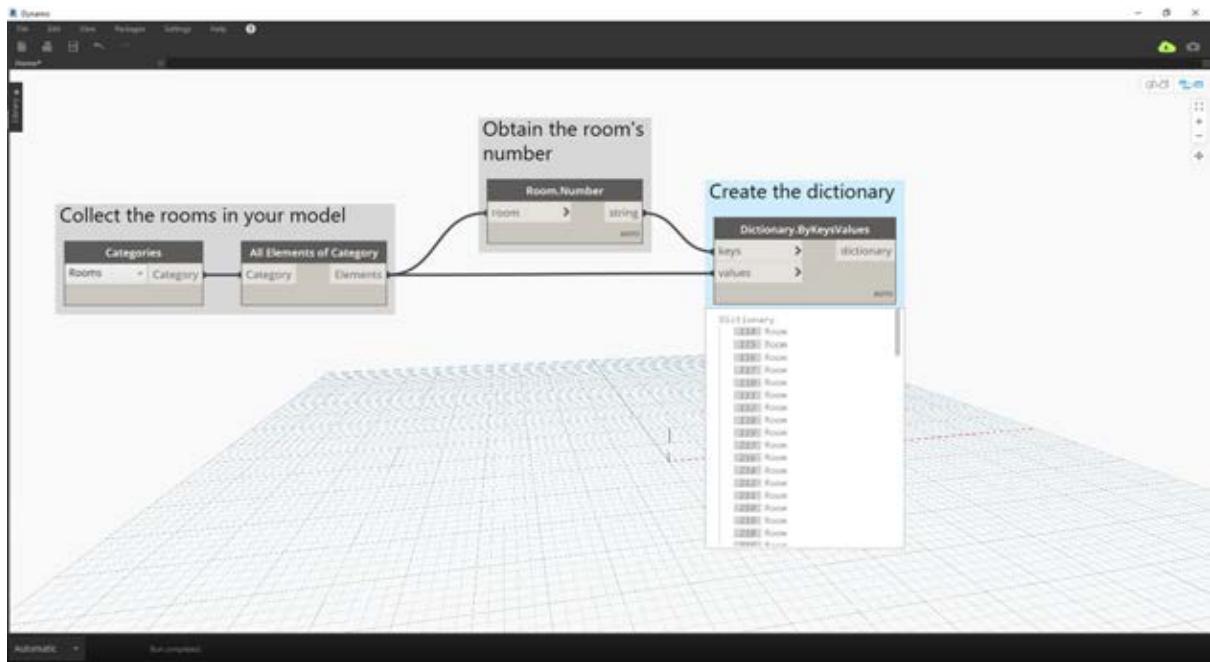
Zuerst müssen wir alle Räume in das Revit-Modell sammeln.

- Wir wählen wir die Revit-Kategorie, die wir verwenden möchten (in diesem Fall Räume).
- Wir weisen Dynamo an, alle diese Elemente zu sammeln.



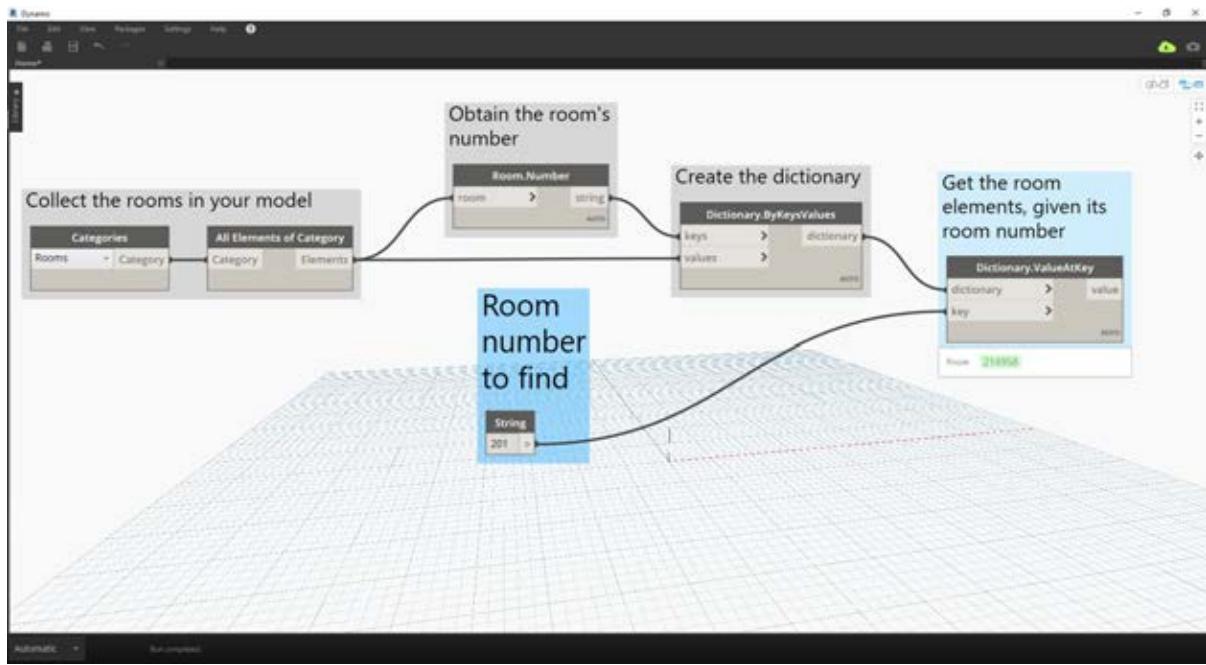
als Nächstes müssen wir entscheiden, welche Schlüssel wir verwenden, um diese Daten zu suchen. (Informationen zu Schlüsseln finden Sie im Abschnitt [9-1 Was ist ein Wörterbuch?](#)).

- Die Daten, die wir verenden, ist die Raumnummer.



Jetzt erstellen wir das Wörterbuch mit den angegebenen Schlüsseln und Elementen.

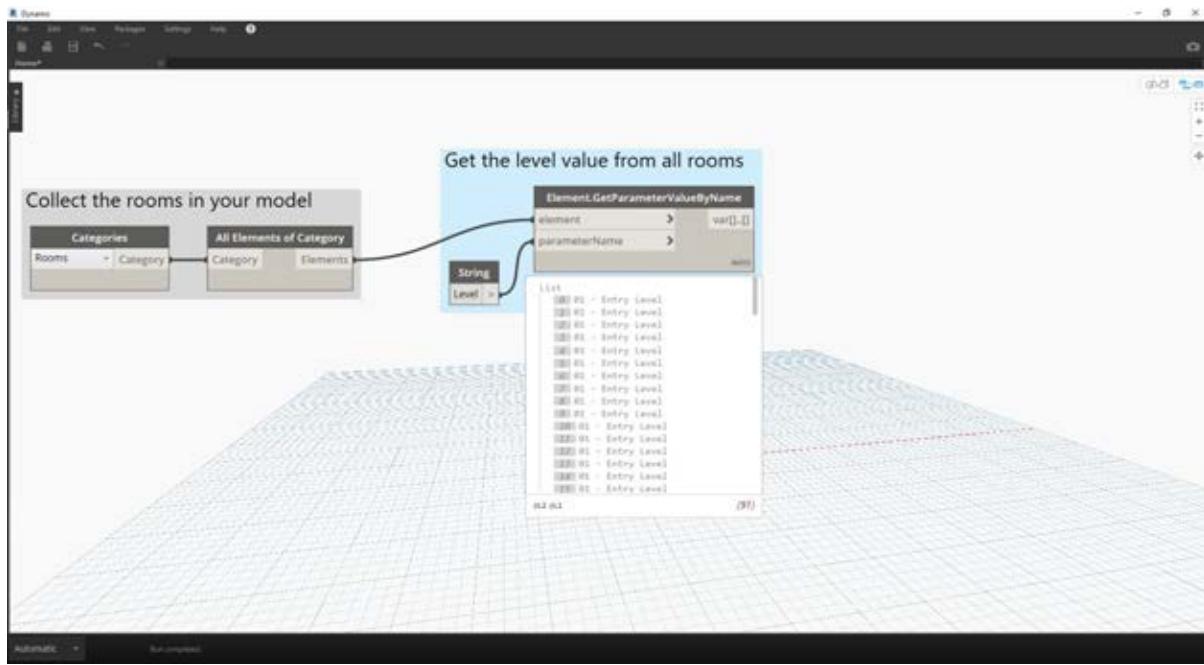
- Der Block **Dictionary.ByKeysValues** erstellt ein Wörterbuch anhand der entsprechenden Eingaben.
- Bei **Schlüsseln** muss es sich um eine Zeichenfolge handeln, aber **Werte** können verschiedene Objekttypen sein.



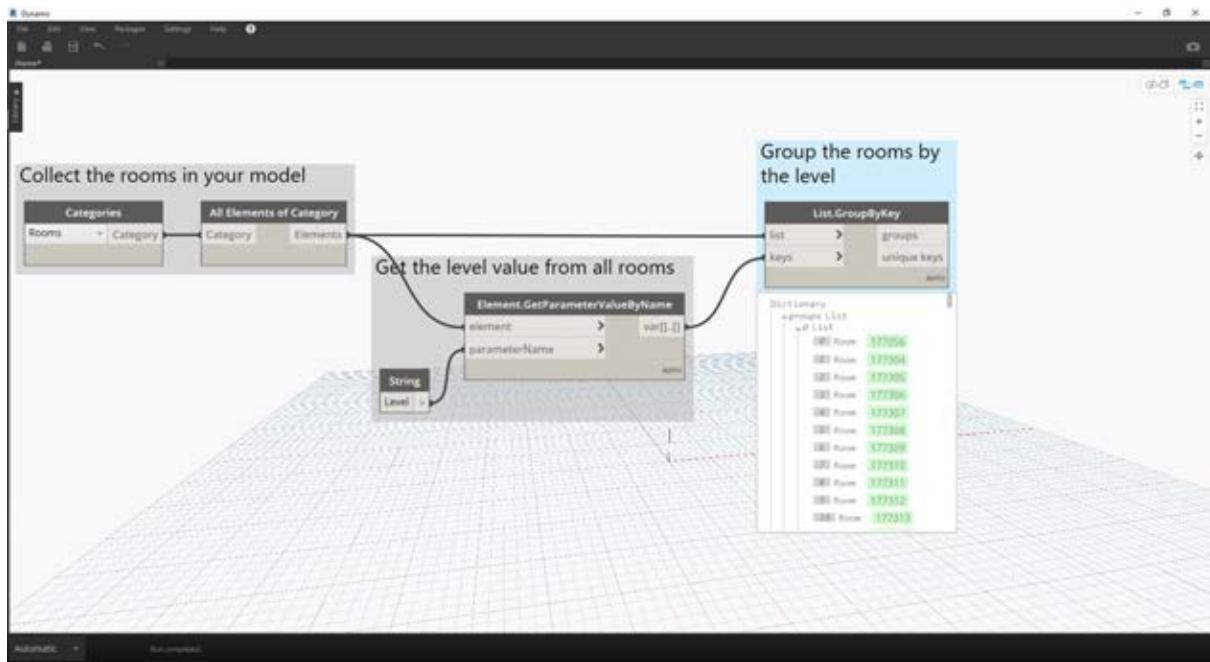
Jetzt können wir einen Raum aus dem Wörterbuch über seine Raumnummer abrufen.

- Zeichenfolge ist der Schlüssel, den wir verwenden, um ein Objekt im Wörterbuch nachzusuchen.
- `Dictionary.ValueAtKey` ruft das Objekt aus dem Wörterbuch ab.

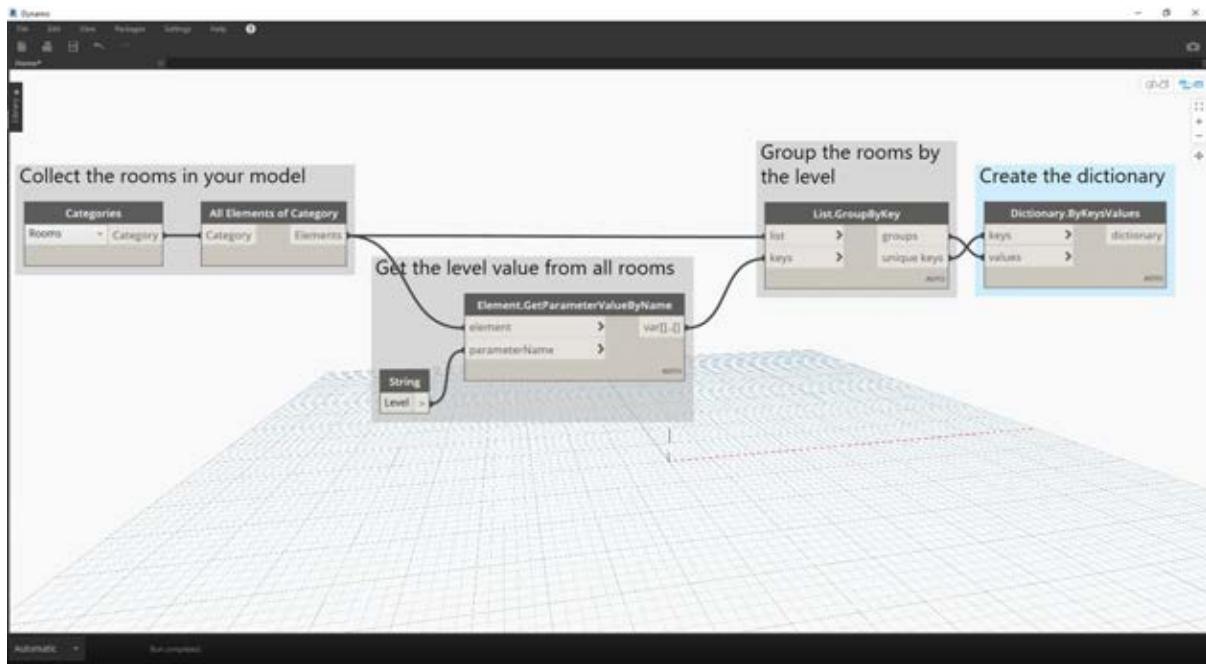
**Mit derselben Wörterbuch-Logik können wir auch Wörterbücher mit gruppierten Objekten erstellen. Wenn wir zum Beispiel alle Räume auf einer bestimmten Ebene nachschlagen möchten, können das obige Diagramm folgendermaßen verändern.**



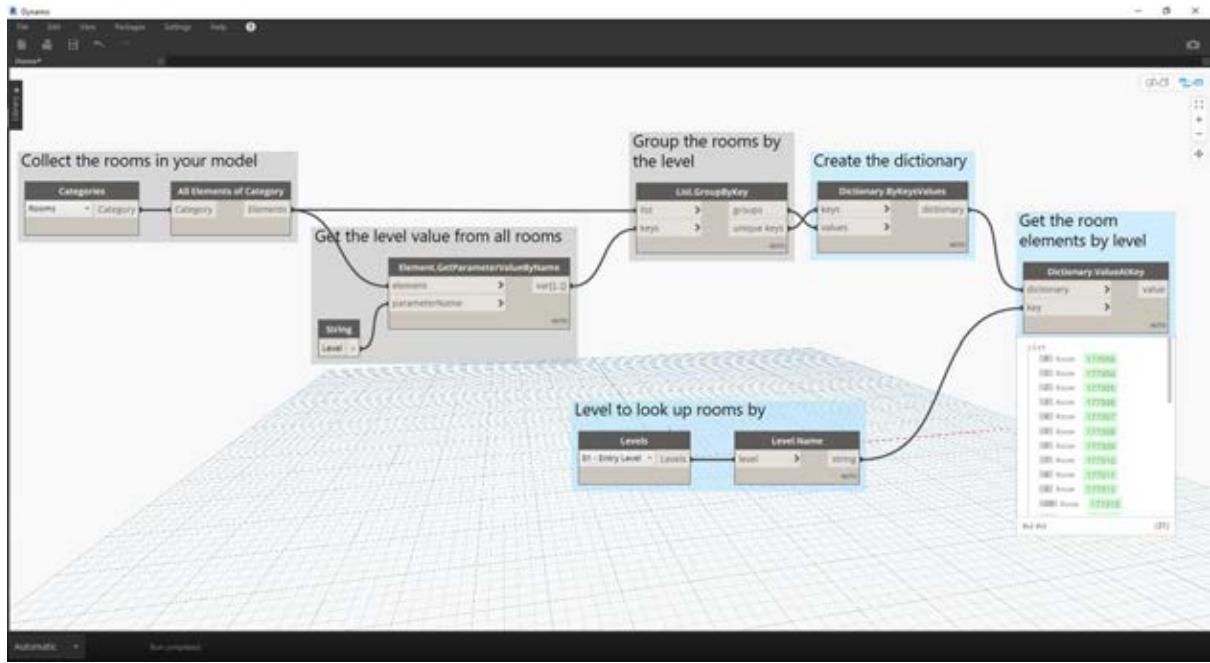
- Anstatt die Zimmernummer als Schlüssel zu nutzen, können wir nun einen Parameterwert verwenden (in diesem Fall Ebene).



- Jetzt können wir die Räume nach der Ebene gruppieren, auf der sie sich befinden.



- Wir haben die Elemente nach Ebene gruppiert. Jetzt können wir die gemeinsam verwendeten Schlüssel (eindeutige Schlüssel) als unsere eindeutigen Schlüssel für das Wörterbuch und die Listen der Räume als die Elemente verwenden.



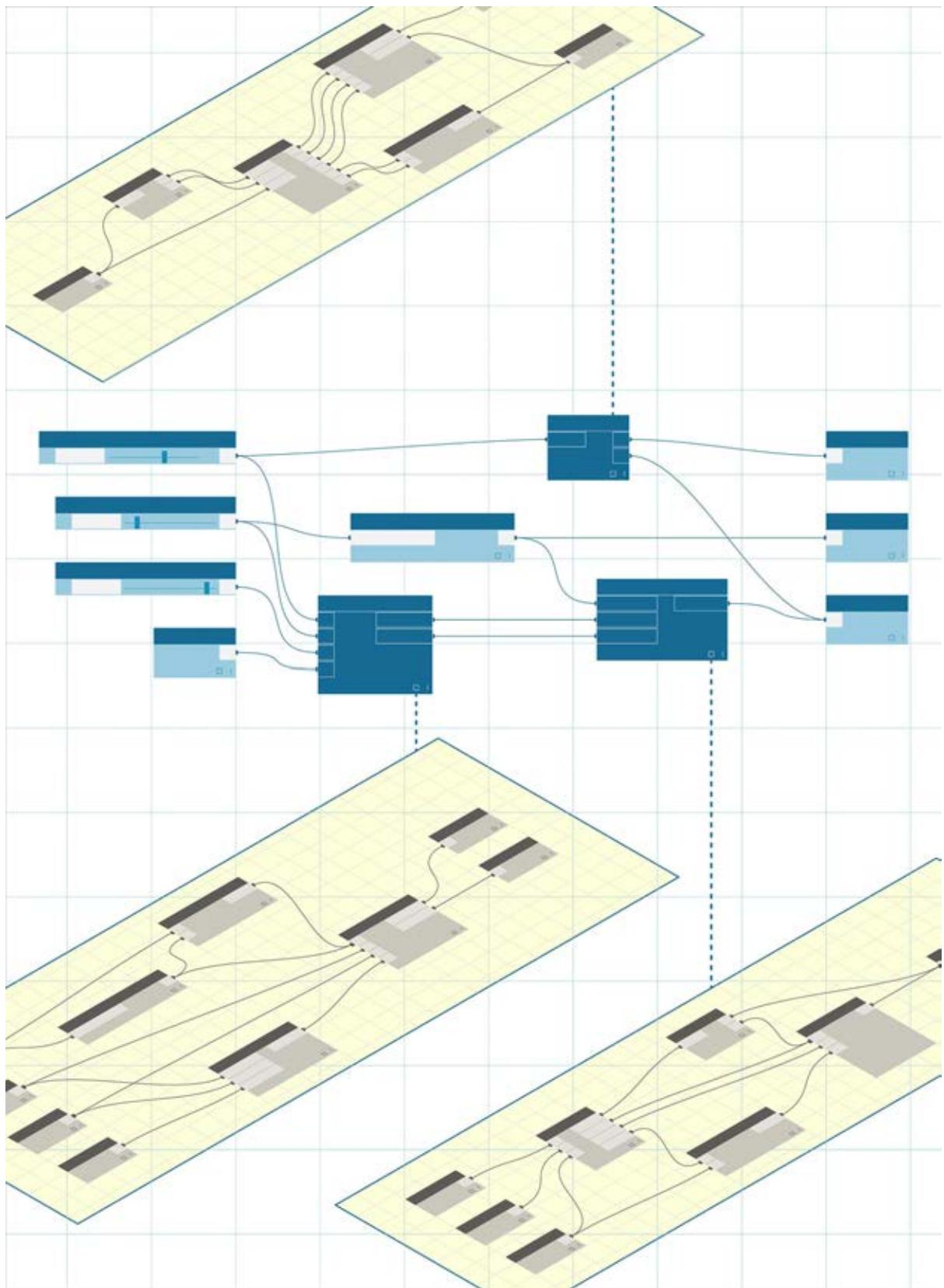
- Letztendlich können wir wie Ebenen im Revit-Modell nutzen, um zu ermitteln, welche Räume sich auf dieser Ebene im Wörterbuch befinden. `Dictionary.ValueAtKey` nimmt den Ebenennamen und gibt die Raumobjekte auf dieser Ebene wieder.

Mit dem Wörterbuch stehen uns praktisch unbeschränkte Möglichkeiten zur Verfügung. Die Möglichkeit, eine Beziehung zwischen Ihren BIM-Daten in Revit und dem eigentlichen Element herzustellen, ermöglicht zahlreiche Anwendungsfälle.

## **Benutzerdefinierte Blöcke**

### **Benutzerdefinierte Blöcke**

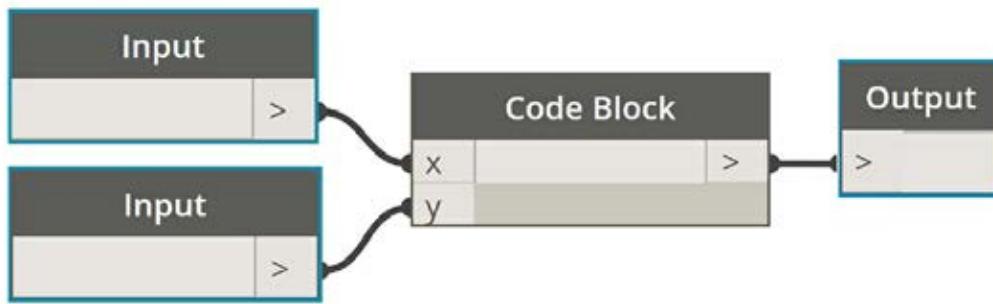
In der Blockbibliothek von Dynamo steht eine Vielfalt vorgegebener Funktionen zur Verfügung. Wenn Sie bestimmte Routinen häufig einsetzen oder spezielle Diagramme an die Community weitergeben möchten, bietet Ihnen Dynamo noch weiterreichende Möglichkeiten.



# Benutzerdefinierte Blöcke

## Benutzerdefinierte Blöcke

Dynamo bietet zahlreiche Core-Blöcke für eine Vielfalt von Aufgaben in der visuellen Programmierung. In manchen Fällen erhalten Sie jedoch schnellere, elegantere oder leichter weiterzugebende Lösungen, indem Sie Ihre eigenen Blöcke konstruieren. Diese können in verschiedenen Projekten wiederverwendet werden, tragen dazu bei, das Diagramm klarer und übersichtlicher zu gestalten, und können an den Package Manager übergeben und mit der globalen Dynamo-Community geteilt werden.



## Bereinigen des Diagramms

Benutzerdefinierte Blöcke werden durch Verschachteln anderer Blöcke und benutzerdefinierter Blöcke in einem benutzerdefinierten Dynamo-Block konstruiert. Dieser ist im Prinzip ein Container. Bei der Ausführung dieses Container-Blocks im Diagramm werden sämtliche darin enthaltenen Funktionen ausgeführt. Dies ermöglicht die Wiederverwendung und Weitergabe nützlicher Kombinationen von Blöcken.

## Übernehmen von Änderungen

Wenn mehrere Kopien eines benutzerdefinierten Blocks im Diagramm vorhanden sind, können Sie alle diese Kopien aktualisieren, indem Sie den zugrunde liegenden benutzerdefinierten Block aktualisieren. Dadurch können Sie das Diagramm nahtlos entsprechend eventuellen Änderungen in Ihrem Arbeitsablauf oder Entwurf aktualisieren.

## Arbeitsteilung

Zu den größten Vorteilen benutzerdefinierter Blöcke gehören wohl ihre Möglichkeiten zur Arbeitsteilung. Wenn ein "Power-User" ein komplexes Dynamo-Diagramm erstellt und dies an einen Designer weitergibt, der noch nicht mit Dynamo gearbeitet hat, kann er das Diagramm auf die wesentlichen, für die Interaktion erforderlichen Angaben komprimieren. Der benutzerdefinierte Block kann geöffnet werden, um das Diagramm in seinem Inneren zu bearbeiten, wobei der Container jedoch einfach bleibt. Benutzerdefinierte Blöcke ermöglichen auf diese Weise den Dynamo-Benutzern die Entwicklung klarer und intuitiver Diagramme.

# PointsToSurface

points

Points

sourceSurface

targetSurface

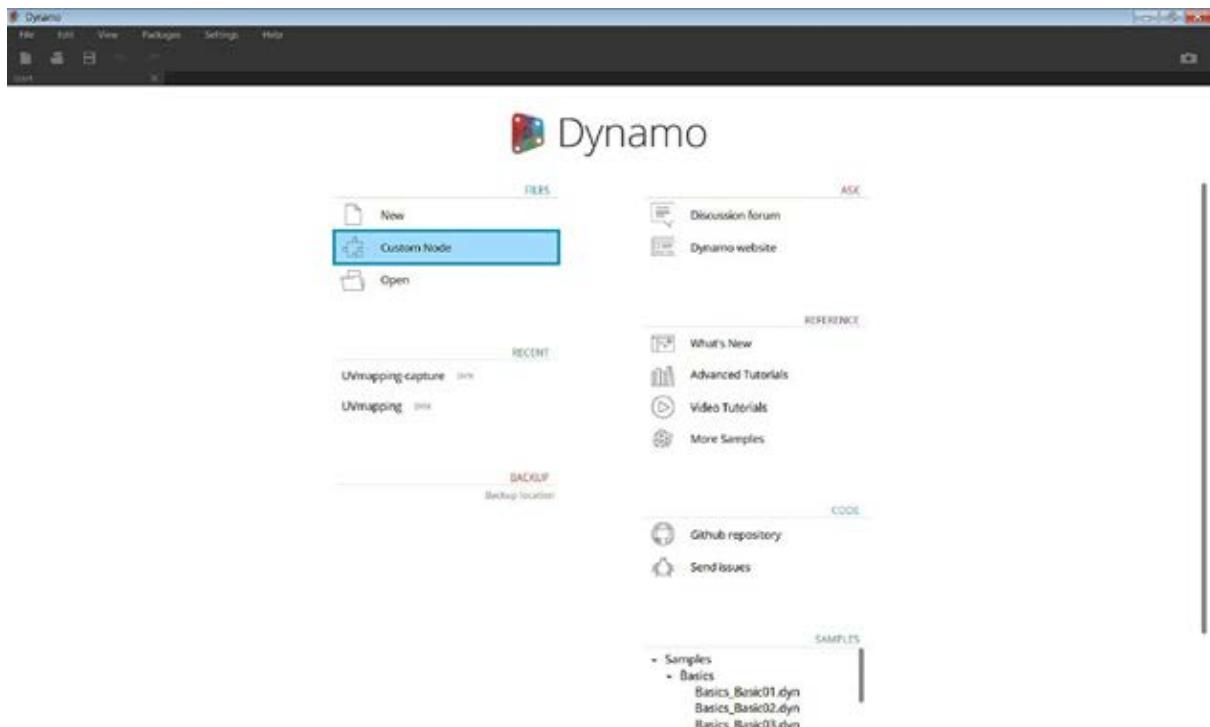


## Zahlreiche Methoden zur Blockentwicklung

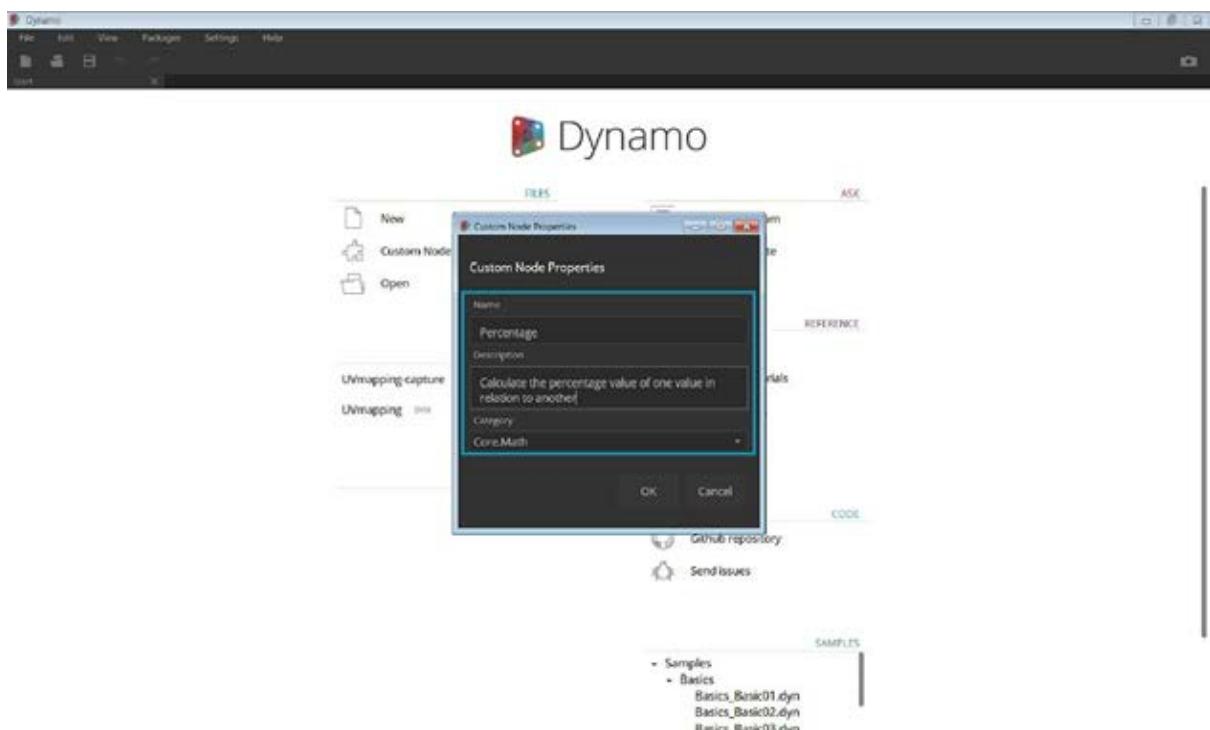
Zur Entwicklung benutzerdefinierter Blöcke in Dynamo steht eine große Vielfalt an Methoden zur Verfügung. In den Beispielen in diesem Kapitel erstellen Sie benutzerdefinierte Blöcke direkt in der Dynamo-Benutzeroberfläche. Wenn Sie Programmierer sind und Interesse an C# oder Zero-Touch-Formatierung haben, finden Sie auf [dieser Seite](#) im Dynamo-Wiki eine genauere Erklärung.

## Umgebung für benutzerdefinierte Blöcke

Im Folgenden erstellen Sie in der Umgebung für benutzerdefinierte Blöcke einen einfachen Block zur Berechnung eines Prozentwerts. Die Umgebung für benutzerdefinierte Blöcke unterscheidet sich zwar von der Umgebung für Diagramme in Dynamo, die Interaktion läuft jedoch im Wesentlichen auf dieselbe Weise ab. Mit diesen Informationen können Sie damit beginnen, Ihren ersten benutzerdefinierten Block zu erstellen.

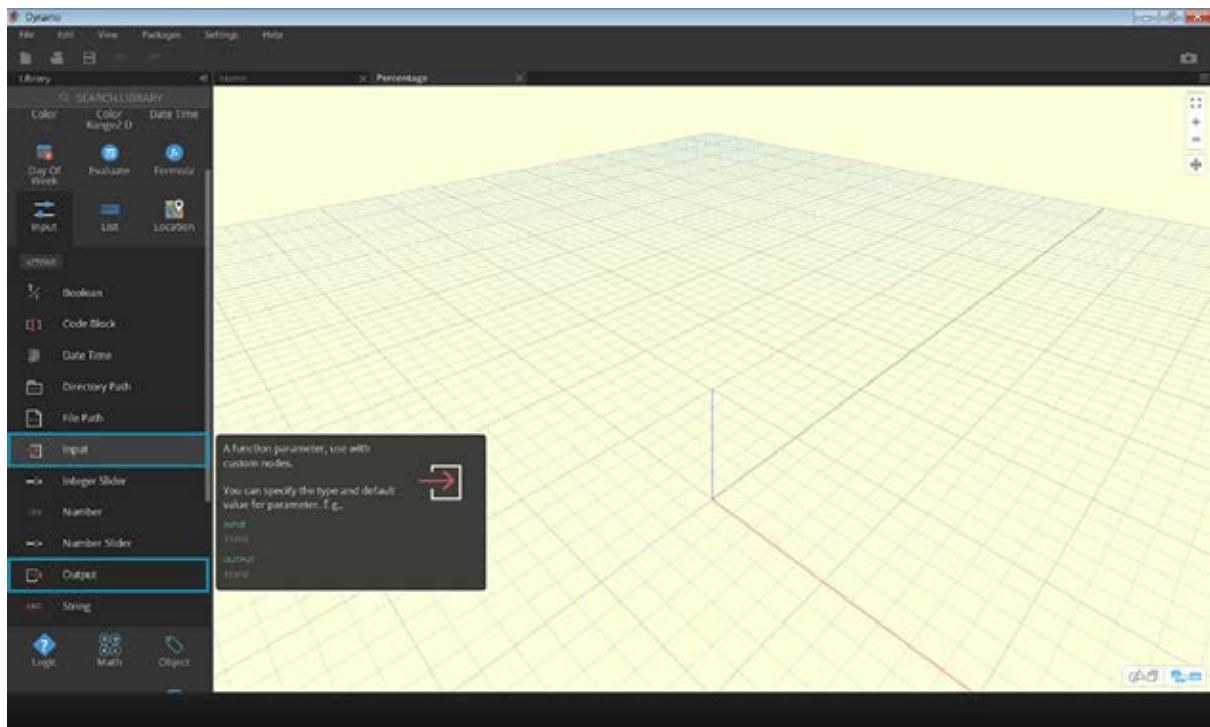


Um einen neuen benutzerdefinierten Block zu erstellen, starten Sie Dynamo und wählen Sie Benutzerdefinierter Block oder geben Sie Strg + Umschalt + N im Ansichtsbereich ein.

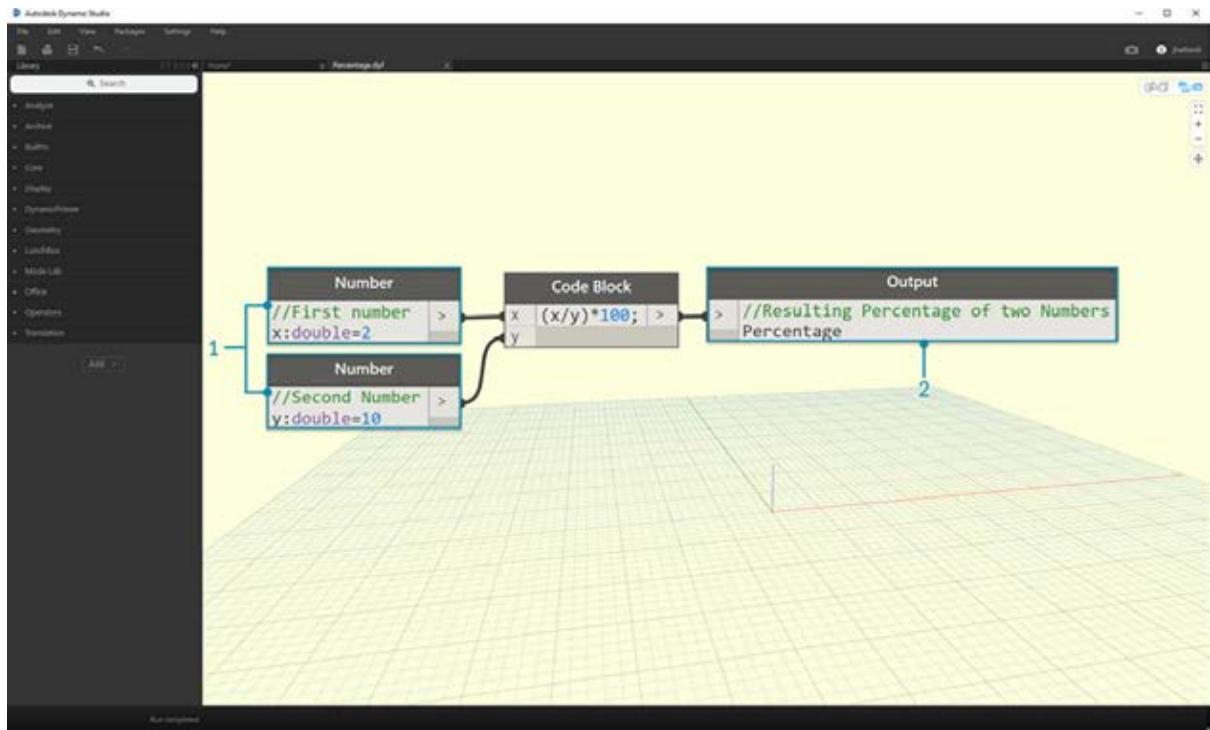


Weisen Sie im Dialogfeld Eigenschaften für benutzerdefinierten Block einen Namen, eine Beschreibung und eine Kategorie zu.

1. **Name:** Prozentsatz
2. **Beschreibung:** Berechnung des Prozentsatzes eines Werts relativ zu einem anderen.
3. **Kategorie:** Core.Math

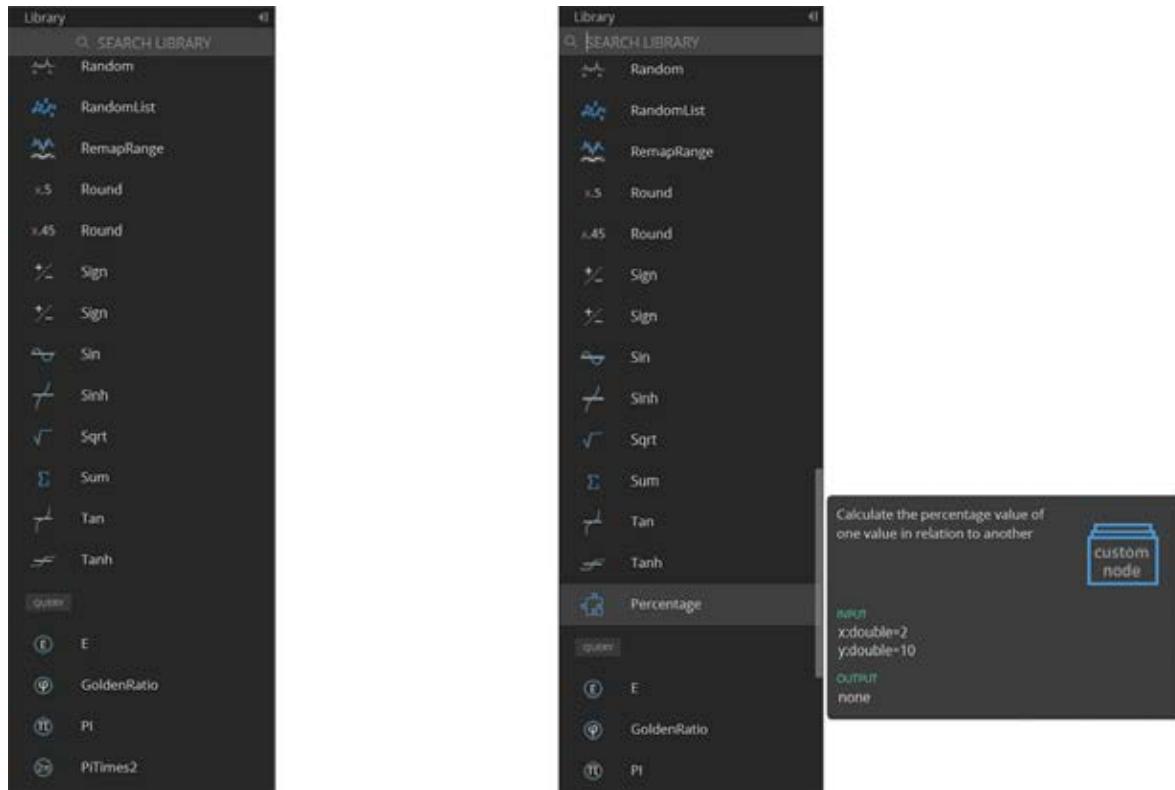


Dadurch wird ein Ansichtsbereich mit gelbem Hintergrund geöffnet, der darauf hinweist, dass Sie in einem benutzerdefinierten Block arbeiten. In diesem Ansichtsbereich haben Sie Zugriff auf alle Core-Blöcke von Dynamo sowie die Blöcke **Input** und **Output**, die zum Benennen der in den benutzerdefinierten Block und aus ihm herausfließenden Daten dienen. Sie finden diese unter *Core > Input*.



- Eingaben:** Input-Blöcke erstellen die Eingaben des benutzerdefinierten Blocks. Verwenden Sie für einen Input-Block die folgende Syntax: `input_name : datatype = default_value(optional)`
- Ausgaben:** Diese Blöcke funktionieren ähnlich wie Input-Blöcke, dienen jedoch zum Erstellen und Benennen der Ausgaben des benutzerdefinierten Blocks. Es ist sinnvoll, den Ein- und Ausgaben **benutzerdefinierte Kommentare** hinzuzufügen, um den Typ der Ein- bzw. Ausgabe zu verdeutlichen. Dies wird im Abschnitt [Erstellen eines benutzerdefinierten Blocks](#) genauer beschrieben.

Sie können diesen benutzerdefinierten Block als DYF-Datei (im Gegensatz den Standard-DYN-Dateien) speichern. Er wird dann automatisch der laufenden und zukünftigen Sitzungen hinzugefügt. Der benutzerdefinierte Block befindet sich in der Bibliothek in der Kategorie, die Sie in seinen Eigenschaften angegeben haben.



Links: Kategorie Core > Math der vorgegebenen Bibliothek. Rechts: Core > Math mit dem neuen benutzerdefinierten Block

## Weitere Schritte

Damit haben Sie Ihren ersten benutzerdefinierten Block erstellt. In den nächsten Abschnitten werden die Funktionen benutzerdefinierter Blöcke und die Veröffentlichung allgemeiner Arbeitsabläufe genauer betrachtet. Thema des folgenden Abschnitts ist die Entwicklung eines benutzerdefinierten Blocks, der Geometrie von einer Oberfläche auf eine andere überträgt.

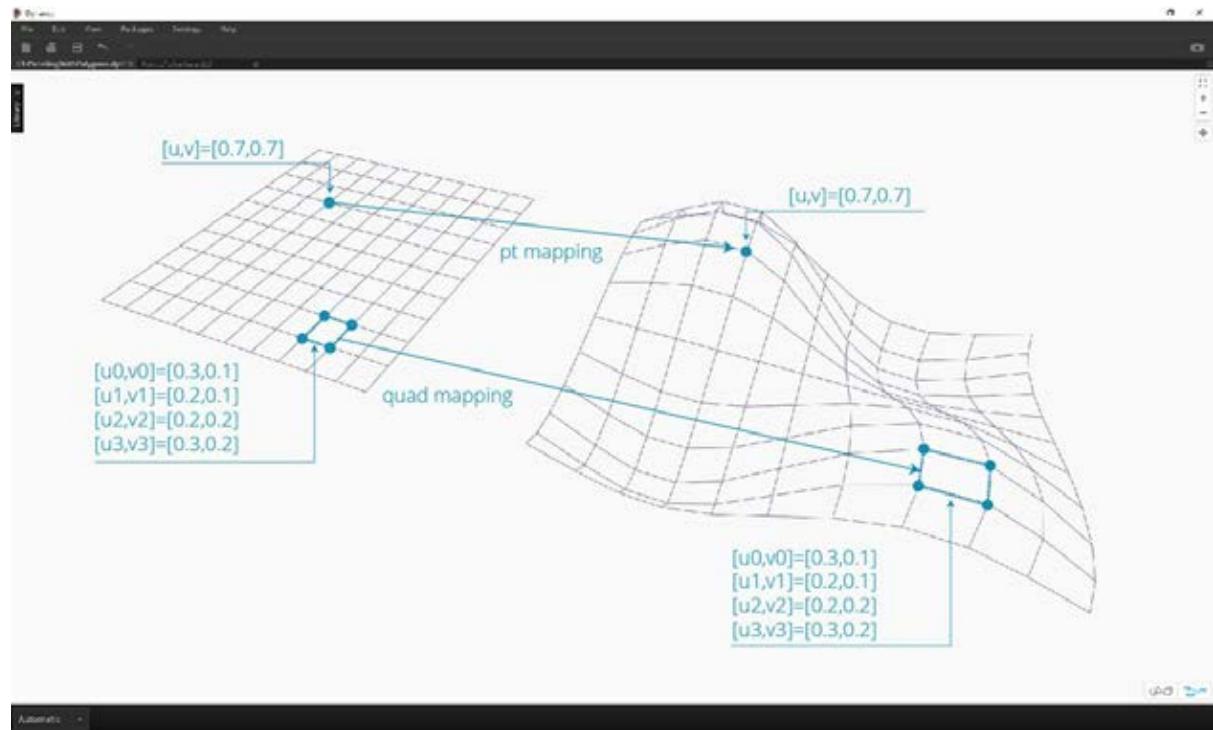
# Benutzerdefinierte Blöcke erstellen

## Benutzerdefinierte Blöcke erstellen

Dynamo bietet mehrere Methoden zum Erstellen benutzerdefinierter Blöcke. Sie können benutzerdefinierte Blöcke neu, aus bestehenden Diagrammen oder explizit in C# erstellen. In diesem Abschnitt wird die Erstellung eines benutzerdefinierten Blocks in der Benutzeroberfläche von Dynamo aus einem bestehenden Diagramm beschrieben. Dieses Verfahren eignet sich ausgezeichnet dazu, den Arbeitsbereich übersichtlicher zu gestalten und Gruppen von Blöcken zur Wiederverwendung zusammenzufassen.

### Benutzerdefinierte Blöcke für die UV-Zuordnung

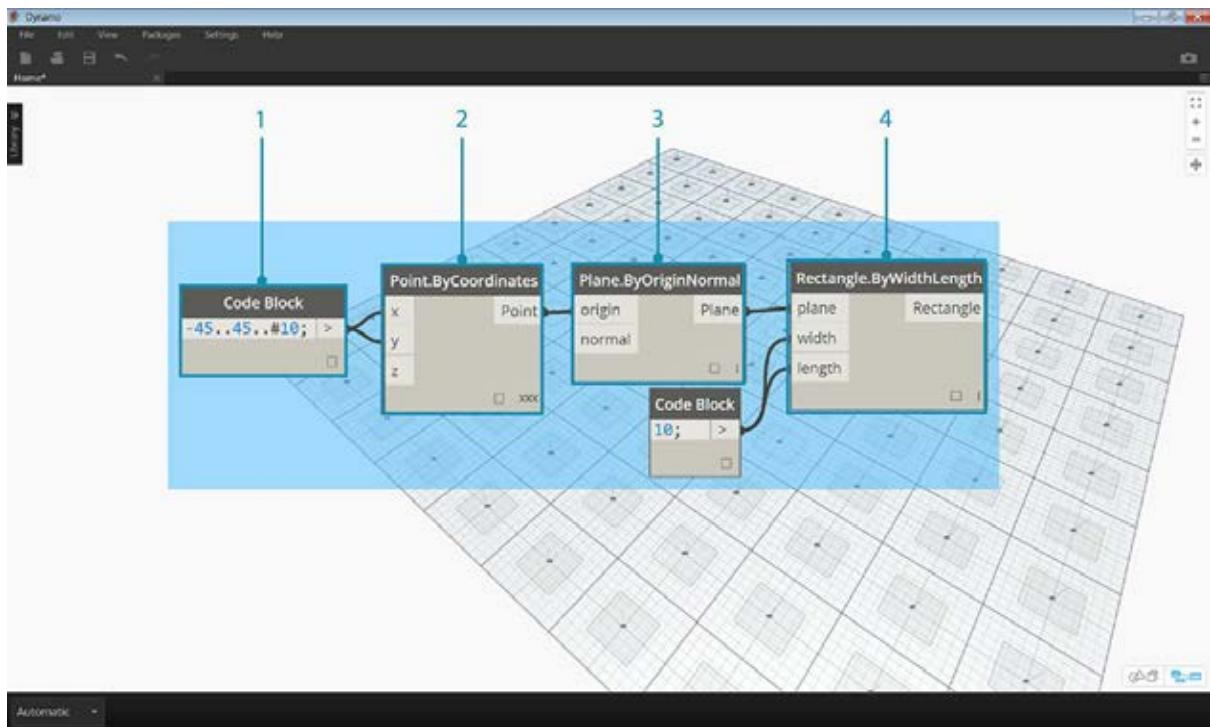
In der unten gezeigten Abbildung wird ein Punkt aus einer Oberfläche mithilfe von UV-Koordinaten einer anderen zugeordnet. Nach diesem Prinzip erstellen Sie eine in Elemente aufgeteilte Oberfläche, die Kurven in der xy-Ebene referenziert. In diesem Fall erstellen Sie viereckige Elemente für die Unterteilung. Nach derselben Logik können Sie jedoch mithilfe der UV-Zuordnung eine große Vielfalt von Elementen erstellen. Es bietet sich an, hier einen benutzerdefinierten Block zu entwickeln, da Sie auf diese Weise ähnliche Vorgänge in diesem Diagramm oder in anderen Dynamo-Arbeitsabläufen leichter wiederholen können.



### Einen benutzerdefinierten Block aus einem bestehenden Diagramm erstellen

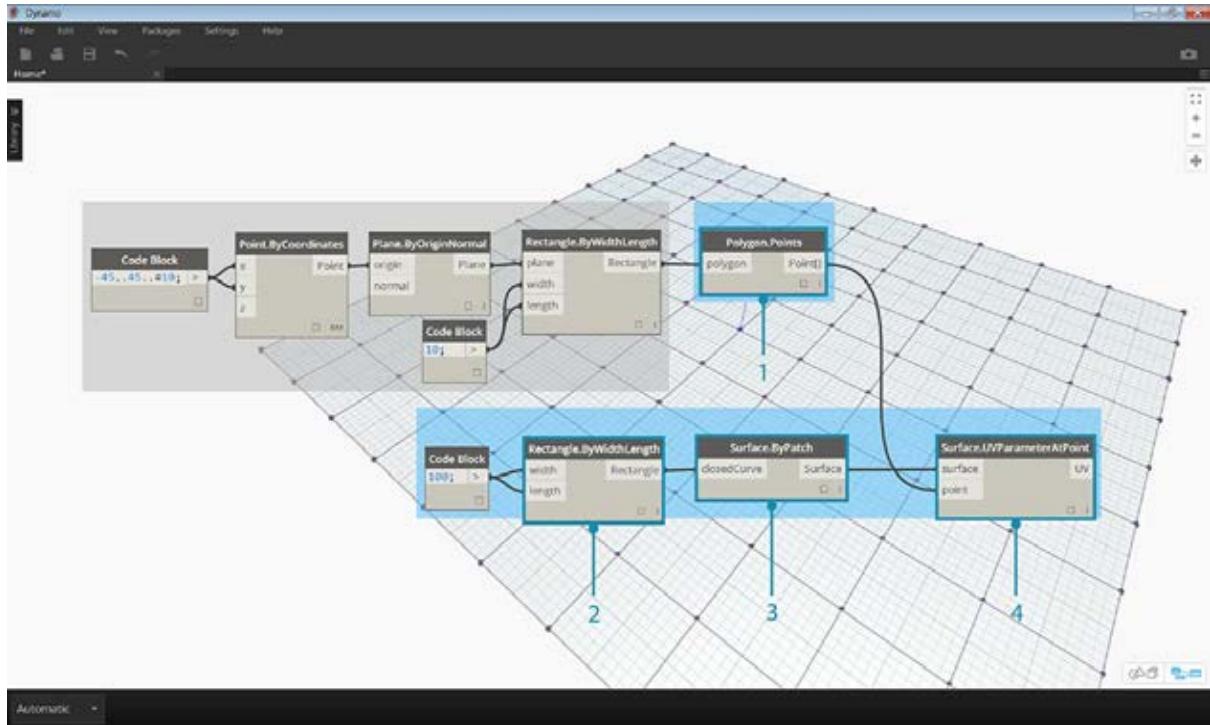
Laden Sie die Beispieldateien für diese Übungslektion herunter (durch Rechtsklicken und Wahl von "Save Link As...") und extrahieren Sie sie. Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [UV-CustomNode.zip](#)

Sie beginnen mit einem Diagramm, das in einem benutzerdefinierten Block verschachtelt werden soll. In diesem Beispiel erstellen Sie ein Diagramm, mit dem Polygone aus einer Basisoberfläche mithilfe von UV-Koordinaten einer Zieloberfläche zugeordnet werden. Diese UV-Zuordnung wird häufig verwendet. Sie bietet sich daher für einen benutzerdefinierten Block an. Weitere Informationen zu Oberflächen und zum UV-Raum finden Sie in Abschnitt 5.5. Das vollständige Diagramm ist *UVmapping\_Custom-Node.dyn* aus der ZIP-Datei, die Sie heruntergeladen haben.



- Code Block:** Erstellen Sie in einem Codeblock einen Bereich mit 10 Zahlen zwischen 45 und -45.
- Point.ByCoordinates:** Verbinden Sie die Ausgaben des Codeblocks mit den x- und y-Eingaben und legen Sie Kreuzprodukt als Vergitterung fest. Sie haben nun ein Raster von Punkten.
- Plane.ByOriginNormal:** Verbinden Sie die Point-Ausgabe mit der origin-Eingabe, um an jeder der Punktpositionen eine Ebene zu erstellen. Dabei wird der vorgegebene Normalenvektor (0,0,1) verwendet.
- Rectangle.ByWidthLength:** Verbinden Sie die Ebenen aus dem vorigen Schritt mit der plane-Eingabe und legen Sie mithilfe eines Codeblocks jeweils 10 als Breite und Länge fest.

Daraufhin müsste ein Raster aus Rechtecken angezeigt werden. Diese Rechtecke ordnen Sie mithilfe von UV-Koordinaten einer Zieloberfläche zu.

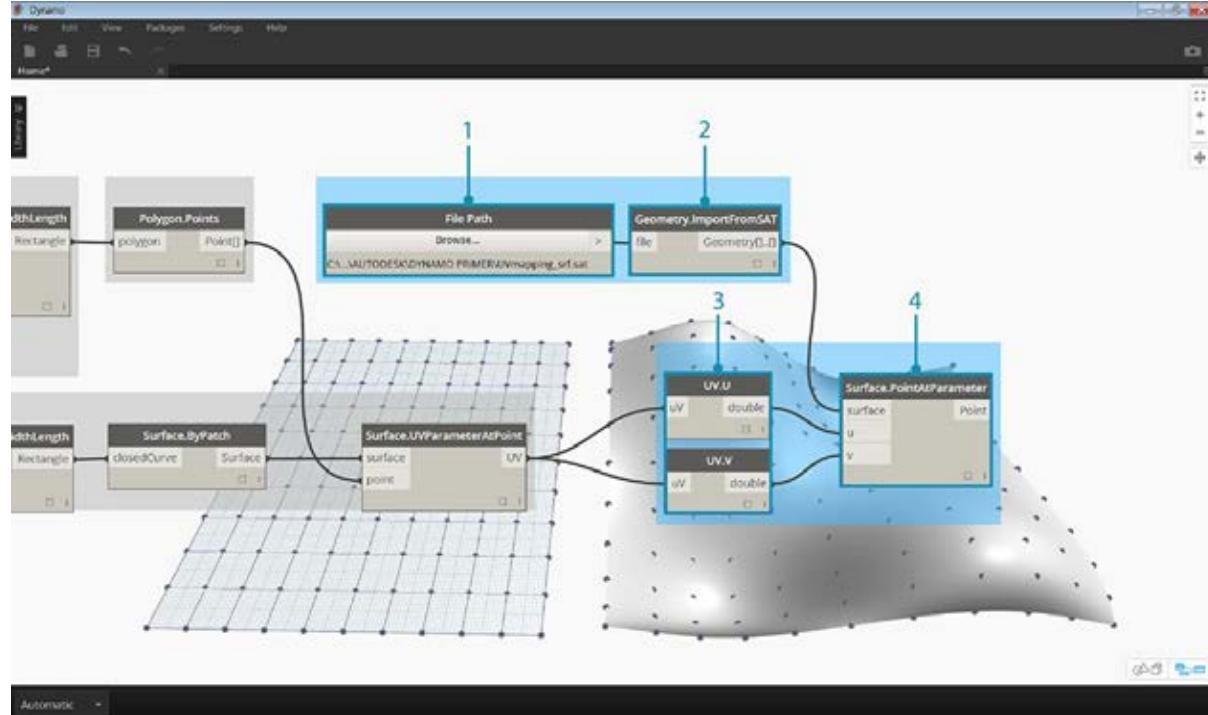


- Polygon.Points:** Verbinden Sie die rectangle-Ausgabe aus dem vorigen Schritt mit der polygon-Eingabe, um die Eckpunkte der einzelnen Rechtecke zu extrahieren. Diese Punkte werden dann der Zieloberfläche

zuordnen.

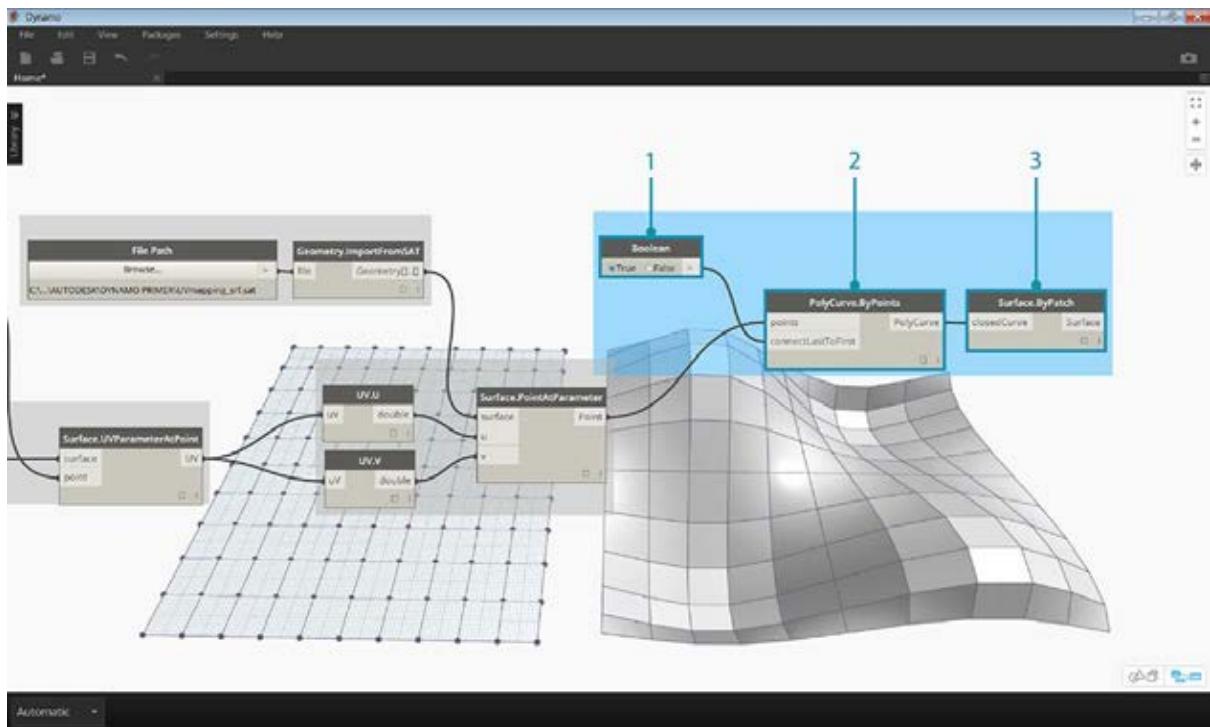
2. **Rectangle.ByWidthLength**: Legen Sie mithilfe eines Codeblocks mit dem Wert 100 die Breite und Länge eines Rechtecks fest. Dies definiert die Begrenzung der Basisfläche.
3. **Surface.ByPatch**: Verbinden Sie das Rechteck aus dem vorigen Schritt mit der *closedCurve*-Eingabe, um eine Basisoberfläche zu erstellen.
4. **Surface.UVParameterAtPoint**: Verbinden Sie die *Point*-Ausgabe des *Polygon.Points*-Blocks und die *Surface*-Ausgabe des *Surface.ByPatch*-Blocks, um die UV-Parameter an den einzelnen Punkten zu erhalten.

Damit haben Sie eine Basisoberfläche und einen Satz UV-Koordinaten erstellt. Jetzt können Sie eine Zielloberfläche importieren und die Punkte auf den Oberflächen zuordnen.



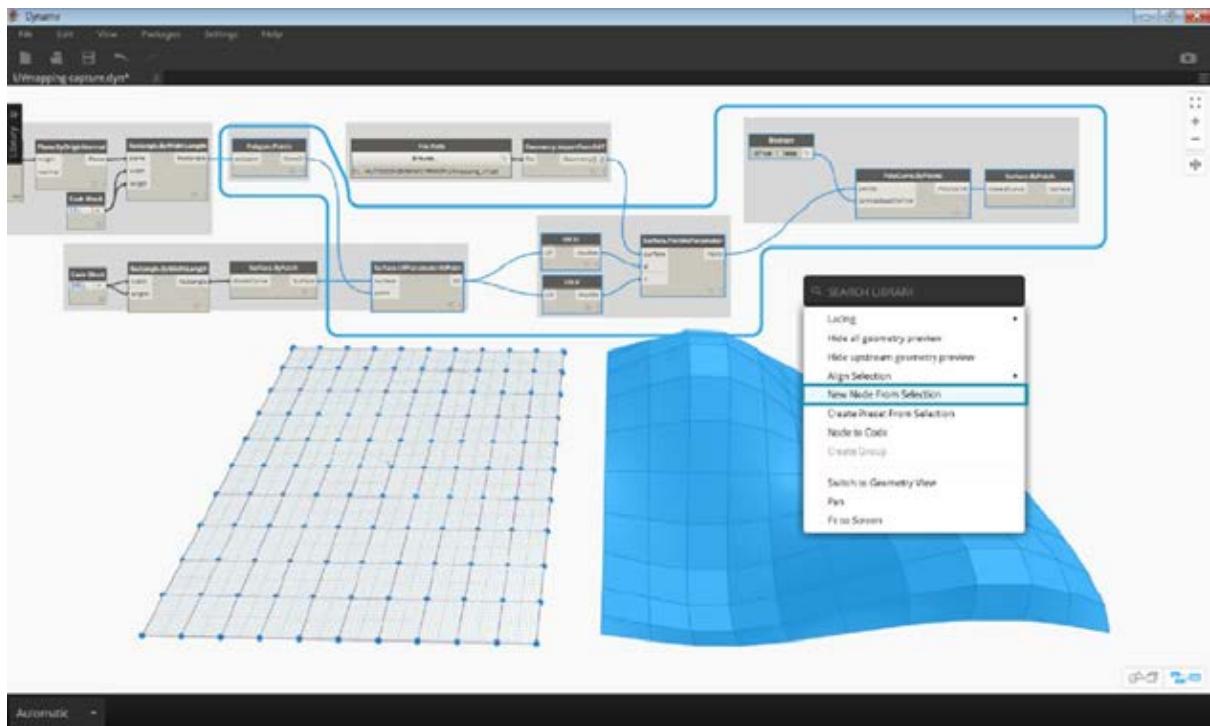
1. **File Path**: Wählen Sie den Dateipfad der Oberfläche aus, den Sie importieren möchten. Die Datei muss eine SAT-Datei sein. Klicken Sie auf die Schaltfläche *Durchsuchen* und navigieren Sie zur Datei *UVmapping\_srf.sat* aus der im oben beschriebenen Schritt heruntergeladenen ZIP-Datei.
2. **Geometry.ImportFromSAT**: Verbinden Sie den Dateipfad, um die Oberfläche zu importieren. Die importierte Oberfläche sollte in der Geometrievorschau angezeigt werden.
3. **UV**: Verbinden Sie die Ausgabe der UV-Parameter mit einem **UV.U**- und einem **UV.V**-Block.
4. **Surface.PointAtParameter**: Verbinden Sie die importierte Oberfläche sowie die U- und V-Koordinaten. Damit sollte ein Raster von 3D-Punkten auf der Zielloberfläche angezeigt werden.

Der letzte Schritt besteht darin, mithilfe der 3D-Punkte rechteckige Oberflächenelemente zu erstellen.



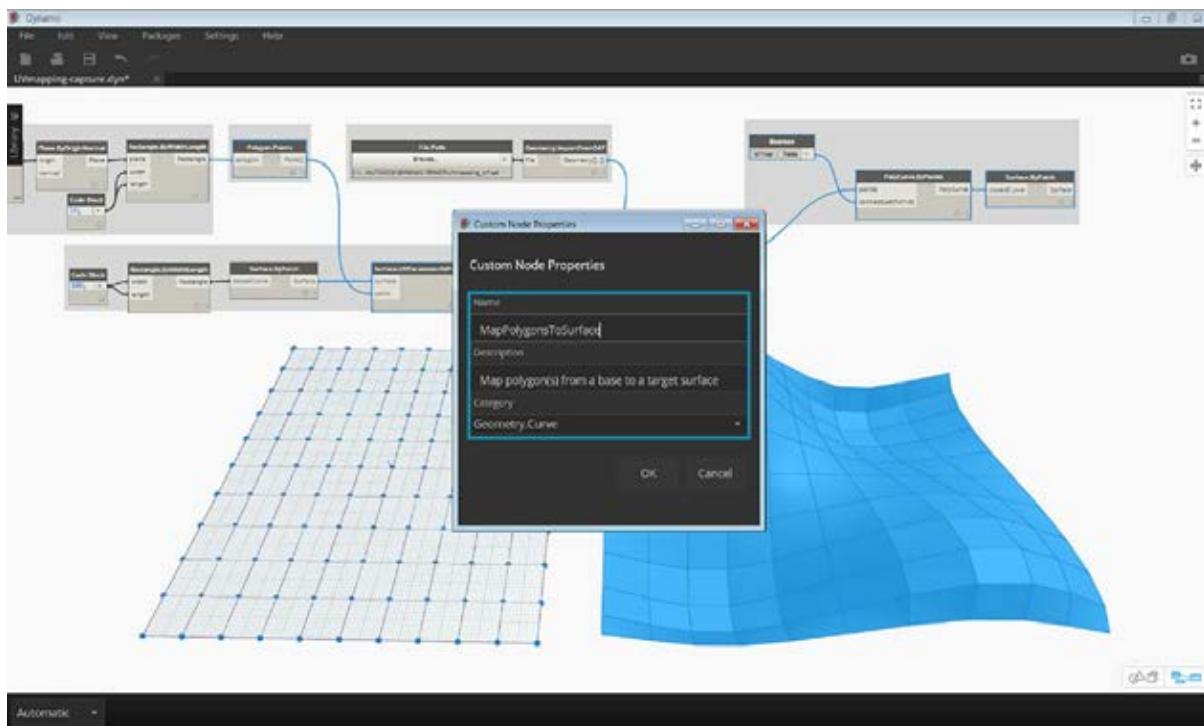
1. **PolyCurve.ByPoints:** Verbinden Sie die Punkte auf der Oberfläche, um eine durch die Punkte verlaufende Polykurve zu konstruieren.
2. **Boolean:** Fügen Sie im Ansichtsbereich einen Boolean-Block hinzu, verbinden Sie ihn mit der *connectLastToFirst*-Eingabe und legen Sie True fest, um die Polykurven zu schließen. Die Oberfläche sollte jetzt in rechteckige Felder unterteilt sein.
3. **Surface.ByPatch:** Verbinden Sie die Polykurven mit der *closedCurve*-Eingabe, um die Oberflächenfelder zu erstellen.

Als Nächstes wählen Sie die Blöcke aus, die in einem benutzerdefinierten Block verschachtelt werden sollen, wobei Sie berücksichtigen, welche Ein- und Ausgaben Sie für Ihren Block benötigen. Der benutzerdefinierte Block soll so flexibel wie möglich sein, d. h., es sollten nicht nur Rechtecke, sondern beliebige Polygone zugeordnet werden können.

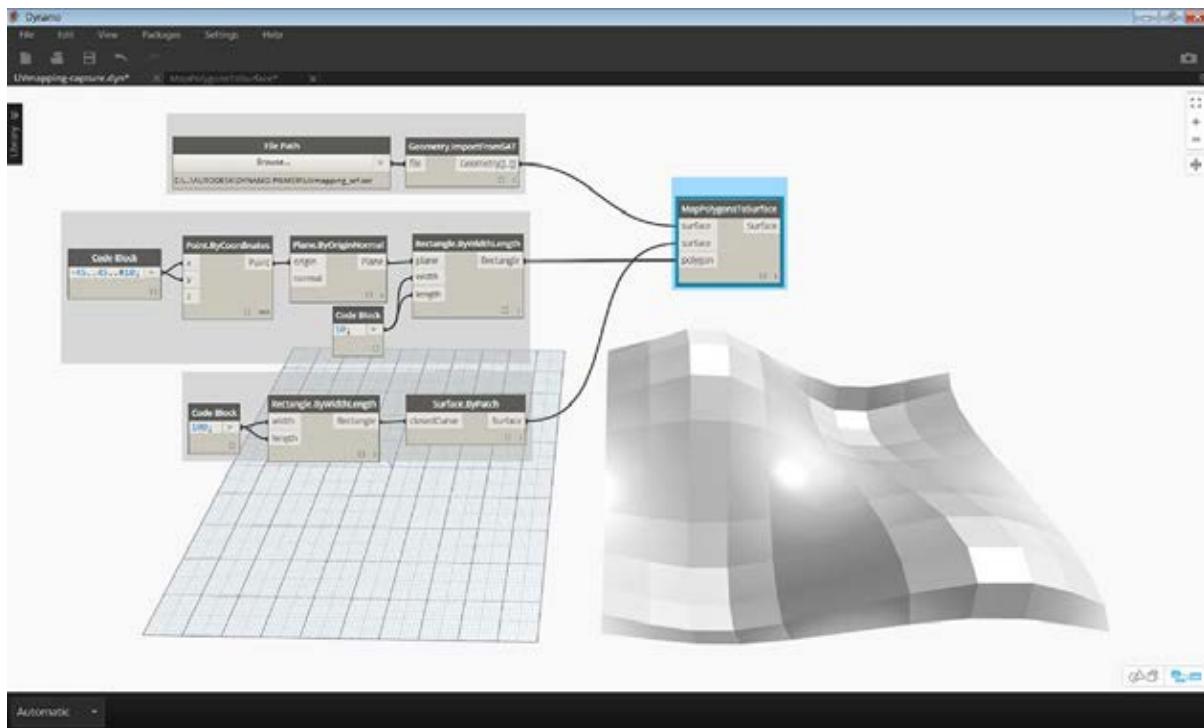


Wählen Sie (beginnend mit *Polygon.Points*) die oben gezeigten Blöcke aus, klicken Sie mit der rechten Maustaste in

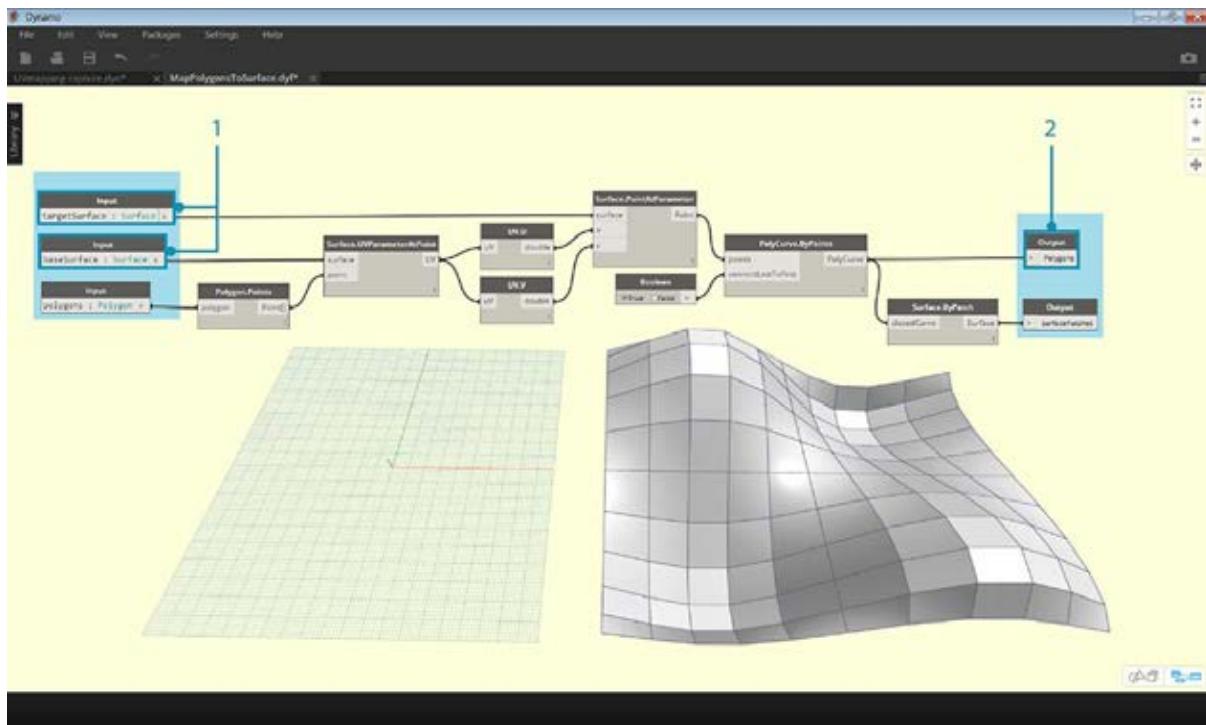
den Ansichtsbereich und wählen Sie *Block aus Auswahl erstellen*.



Weisen Sie im Dialogfeld Eigenschaften für den benutzerdefinierten Block einen Namen, eine Beschreibung und eine Kategorie zu.

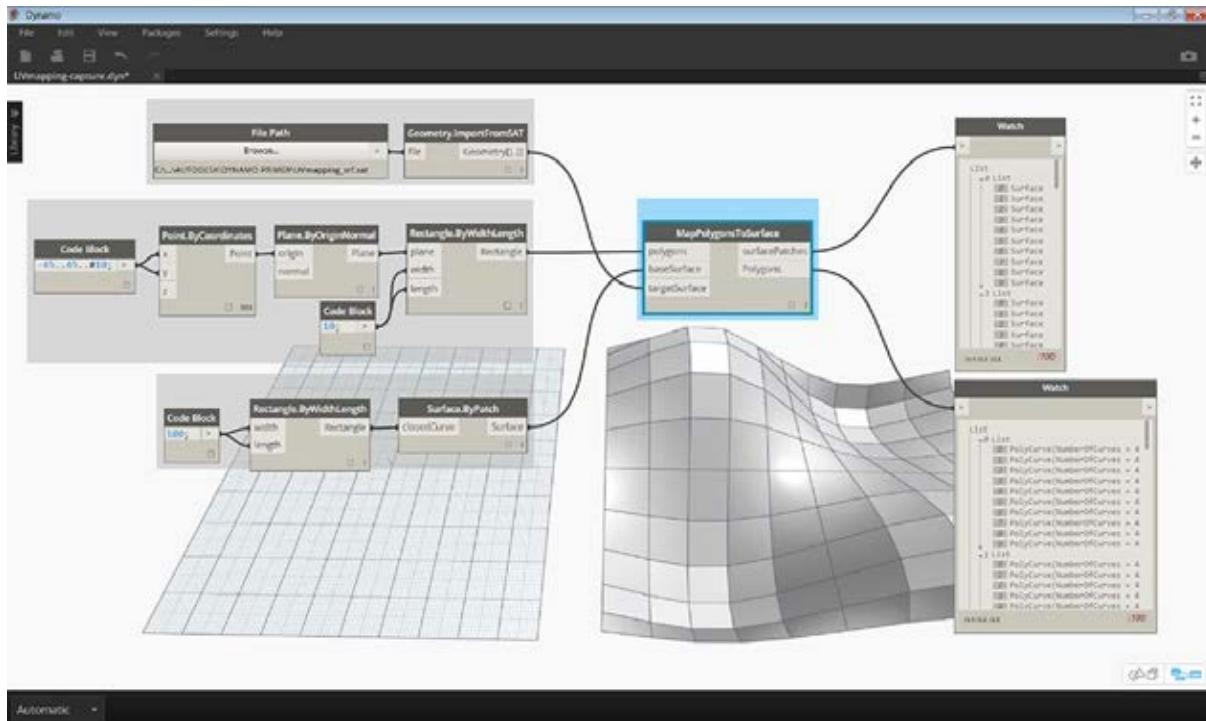


Der Ansichtsbereich ist mit dem benutzerdefinierten Block wesentlich übersichtlicher. Den Namen der Ein- und Ausgaben wurden die entsprechenden Angaben aus den Originalblöcken zugrunde gelegt. Bearbeiten Sie den benutzerdefinierten Block, um aussagekräftigere Namen anzugeben.



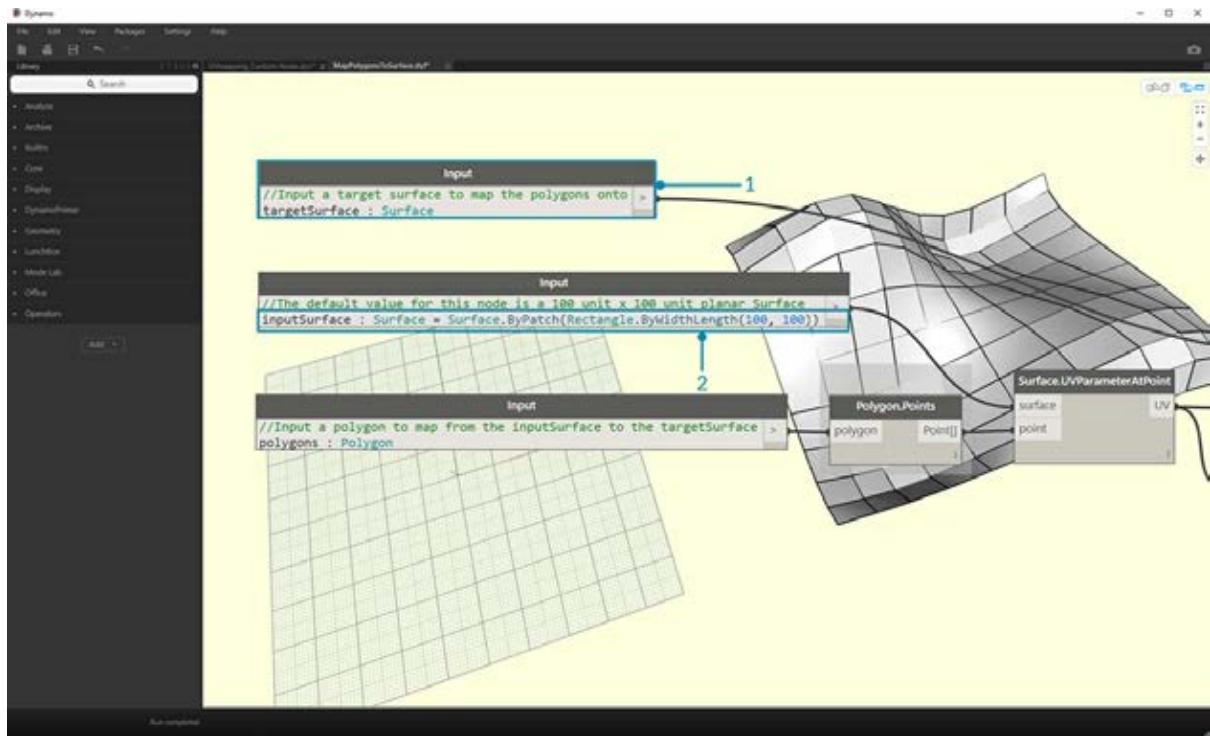
Doppelklicken Sie auf den benutzerdefinierten Block, um ihn zu bearbeiten. Dadurch öffnen Sie einen Arbeitsbereich mit gelbem Hintergrund, der darauf hinweist, dass Sie im Inneren eines Blocks arbeiten.

1. **Eingaben:** Ändern Sie die Namen der Eingaben zu *baseSurface* und *targetSurface*.
2. **Ausgaben:** Fügen Sie eine zusätzliche Ausgabe für die zugeordneten Polygone hinzu. Speichern Sie den benutzerdefinierten Block, und kehren Sie zur Ausgangsansicht zurück.



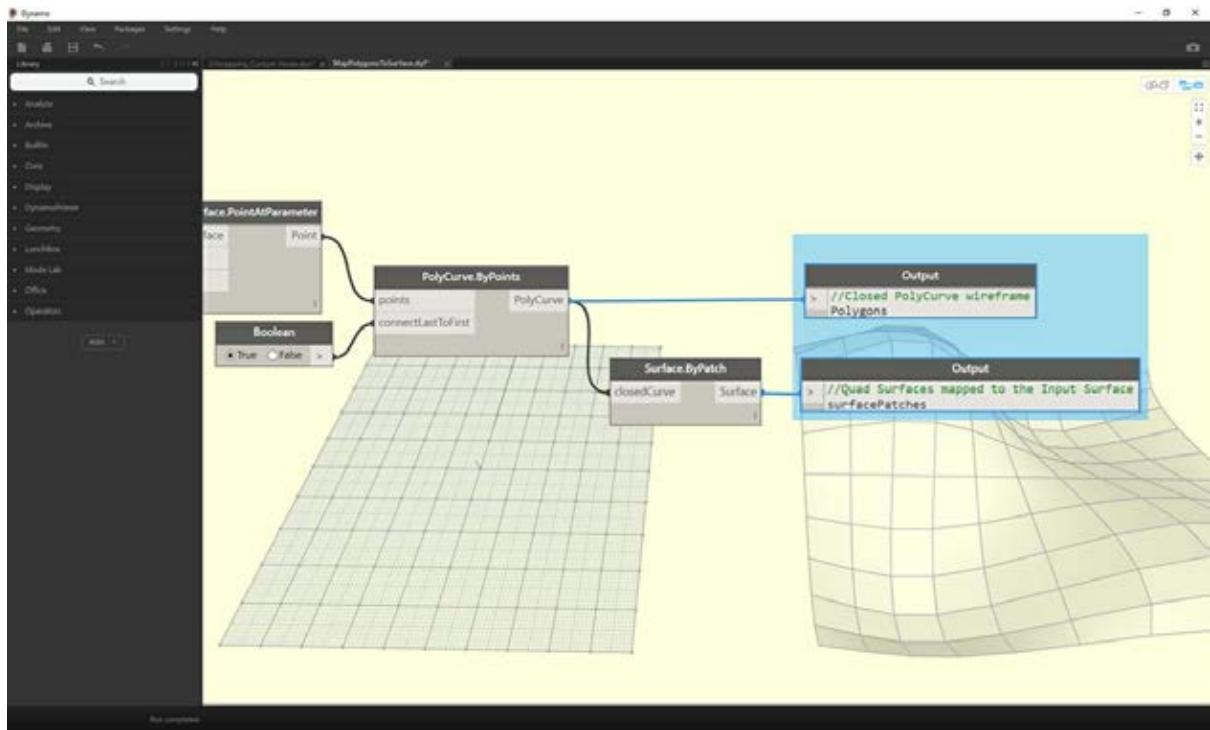
Im **MapPolygonsToSurface**-Block wurden die eben vorgenommenen Änderungen übernommen.

Um den benutzerdefinierten Block noch zuverlässiger zu gestalten, können Sie außerdem **benutzerdefinierte Kommentare** hinzufügen. Kommentare können Aufschluss über den Typ der Ein- und Ausgaben geben oder Erläuterungen zur Funktionsweise des Blocks enthalten. Kommentare werden angezeigt, wenn der Benutzer den Cursor auf eine Eingabe oder Ausgabe eines benutzerdefinierten Blocks setzt.

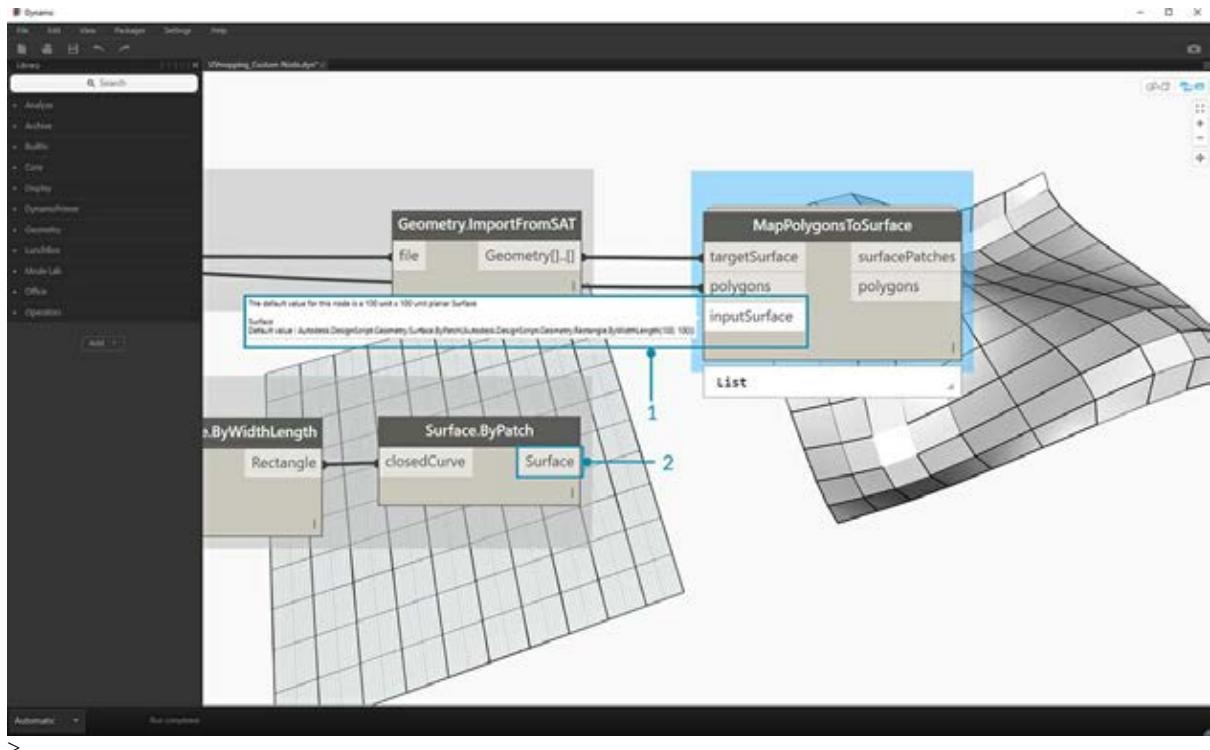


Doppelklicken Sie auf den benutzerdefinierten Block, um ihn zu bearbeiten. Dadurch wird erneut der Arbeitsbereich mit dem gelben Hintergrund geöffnet.

1. Beginnen Sie mit der Bearbeitung des Eingabe-Codeblocks. Um mit einem Kommentar zu beginnen, geben Sie `/**` und anschließend den Kommentartext ein. Geben Sie Informationen ein, die das Verständnis des Blocks erleichtern können. In diesem Fall wird `targetSurface` beschrieben.
2. Legen Sie außerdem den Vorgabewert für `inputSurface` fest, indem Sie als Eingabetyp einen Wert vorgeben. In diesem Fall wird als Vorgabewert das ursprüngliche `Surface.ByPatch` angegeben.



Kommentare können auch auf Ausgaben angewendet werden. Beginnen Sie mit der Bearbeitung des Texts im Ausgabe-Codeblock. Um mit einem Kommentar zu beginnen, geben Sie `/**` und anschließend den Kommentartext ein. In diesem Fall werden die Ausgaben `Polygons` und `surfacePatches` mit ausführlicheren Beschreibungen erläutert.



1. Setzen Sie den Cursor auf die Eingaben des benutzerdefinierten Blocks, um die Kommentare anzuzeigen.
2. Da für *inputSurface* ein Vorgabewert festgelegt ist, können Sie die Definition auch ohne Eingabewert für die Oberfläche ausführen.

# Hinzufügen zu Ihrer Bibliothek

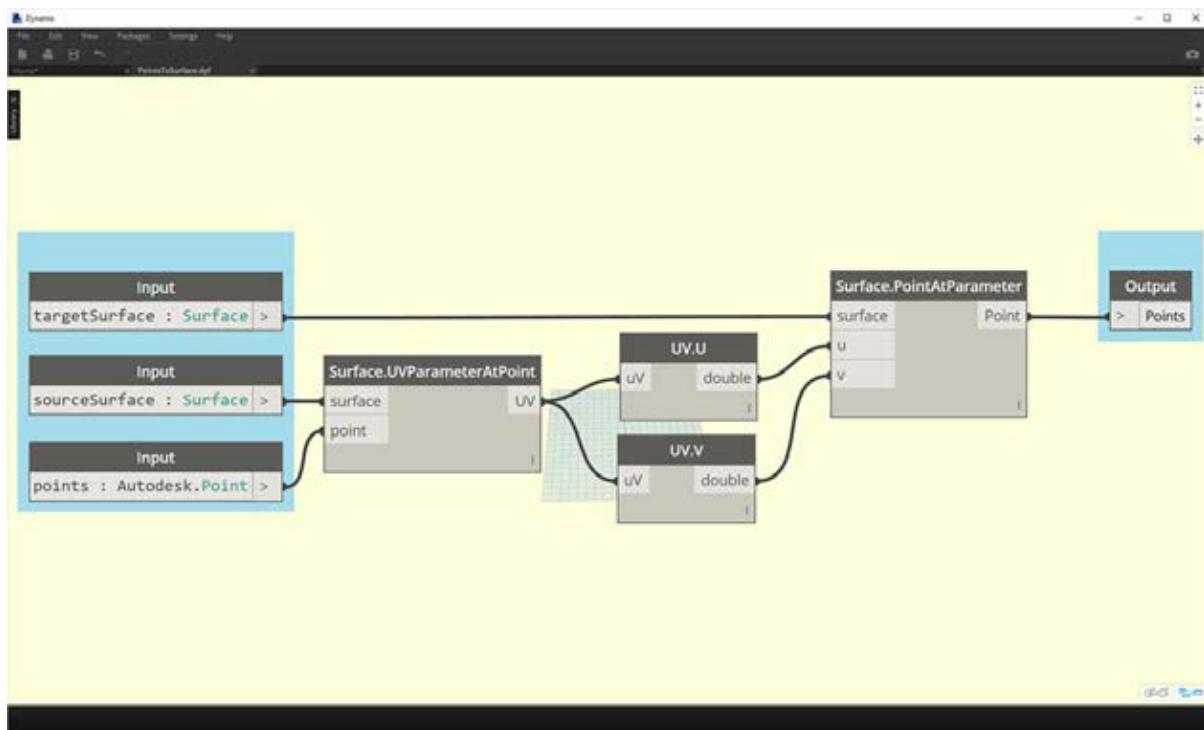
## Hinzufügen zu Ihrer Bibliothek

Sie haben einen benutzerdefinierten Block erstellt und ihn auf einen bestimmten Prozess im Dynamo-Diagramm angewendet. Da dieser Block sehr nützlich ist, möchten Sie ihn in die Dynamo-Bibliothek aufnehmen, damit er in anderen Diagrammen referenziert werden kann. Dazu müssen Sie den Block lokal veröffentlichen. Sie gehen dabei auf ähnliche Weise vor wie beim Veröffentlichen von Paketen, das im nächsten Kapitel im Detail behandelt wird.

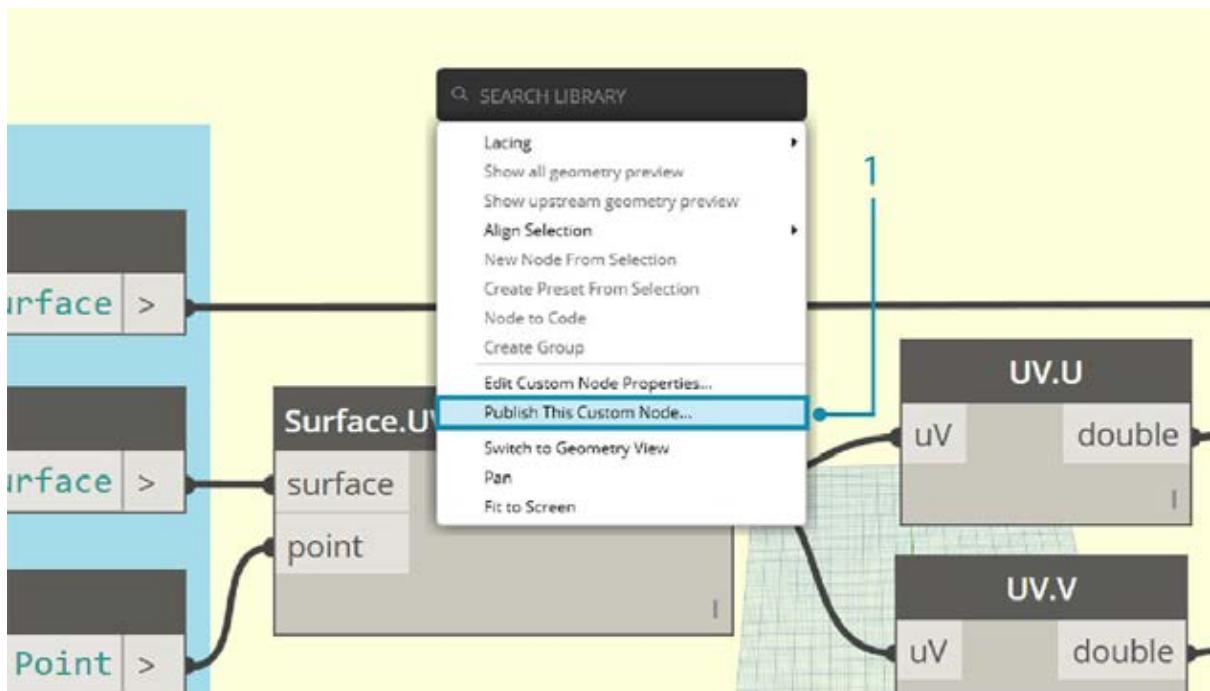
### Lokale Veröffentlichung eines benutzerdefinierten Blocks

Verwenden Sie weiterhin den benutzerdefinierten Block, den Sie im vorigen Abschnitt erstellt haben. Indem Sie den Block lokal veröffentlichen, stellen Sie ihn in Ihrer Dynamo-Bibliothek bereit und können darauf zugreifen, wenn Sie eine neue Sitzung öffnen. Wenn ein Block nicht veröffentlicht wird, muss er für ein Dynamo-Diagramm, das diesen Block referenziert, in dessen Ordner enthalten sein (oder über *Datei > Bibliothek importieren* in Dynamo importiert werden).

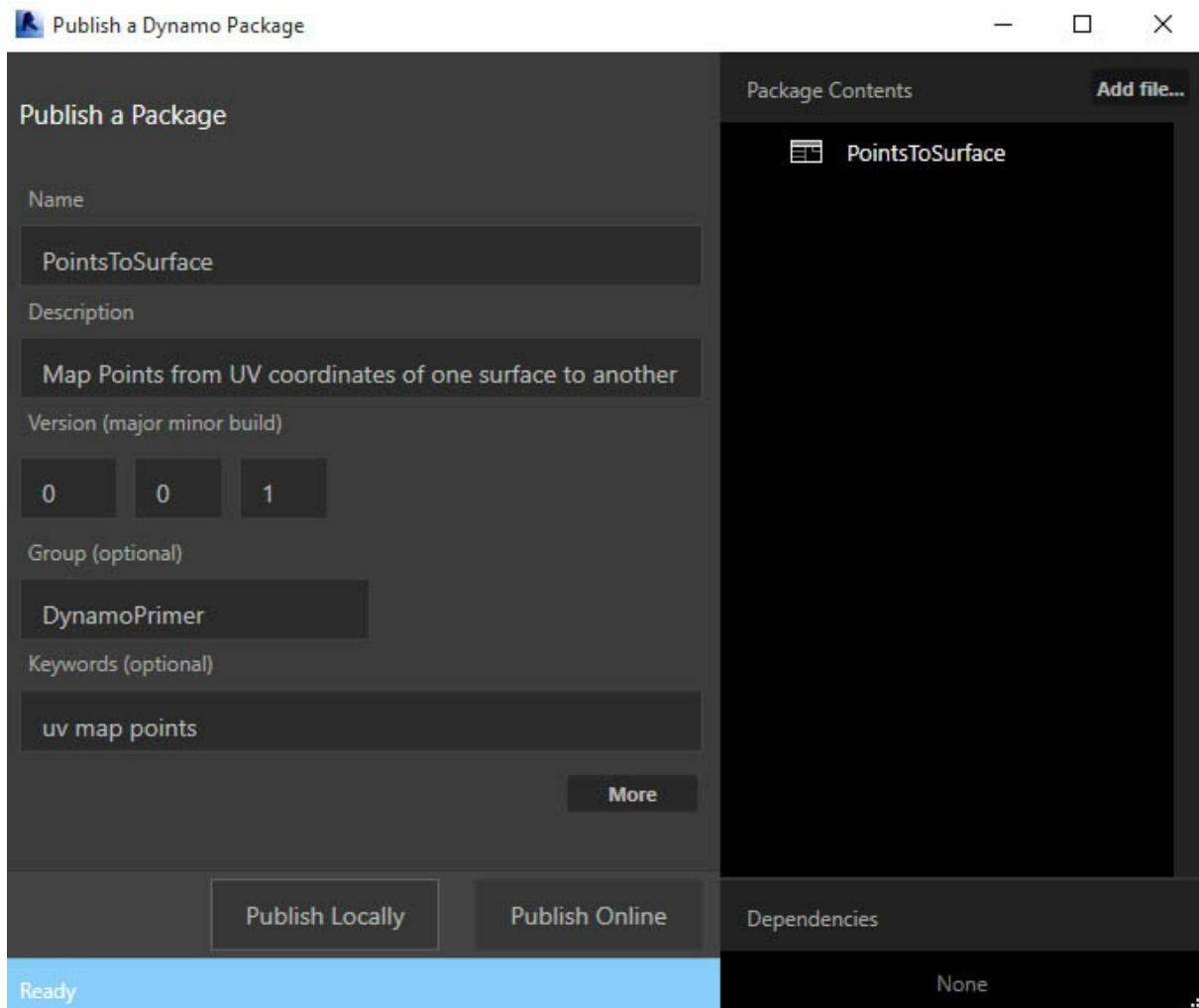
Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [PointsToSurface.dyn](#)



Nachdem Sie den benutzerdefinierten PointsToSurface-Block geöffnet haben, wird das oben gezeigte Diagramm im Editor für benutzerdefinierte Blöcke von Dynamo angezeigt. Sie können einen benutzerdefinierten Block auch durch Doppelklicken im Diagrammeditor von Dynamo öffnen.

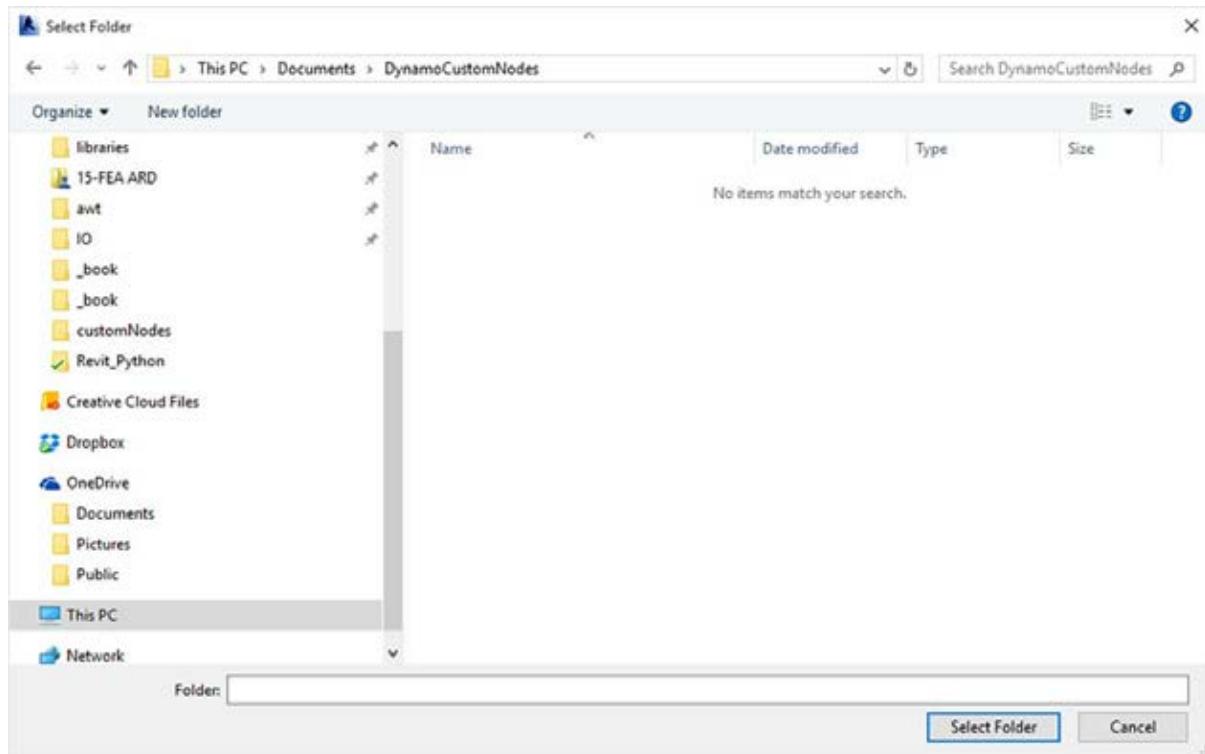


- Um einen benutzerdefinierten Block lokal zu veröffentlichen, klicken Sie mit der rechten Maustaste in den Ansichtsbereich und wählen Sie *Diesen benutzerdefinierten Block veröffentlichen*.

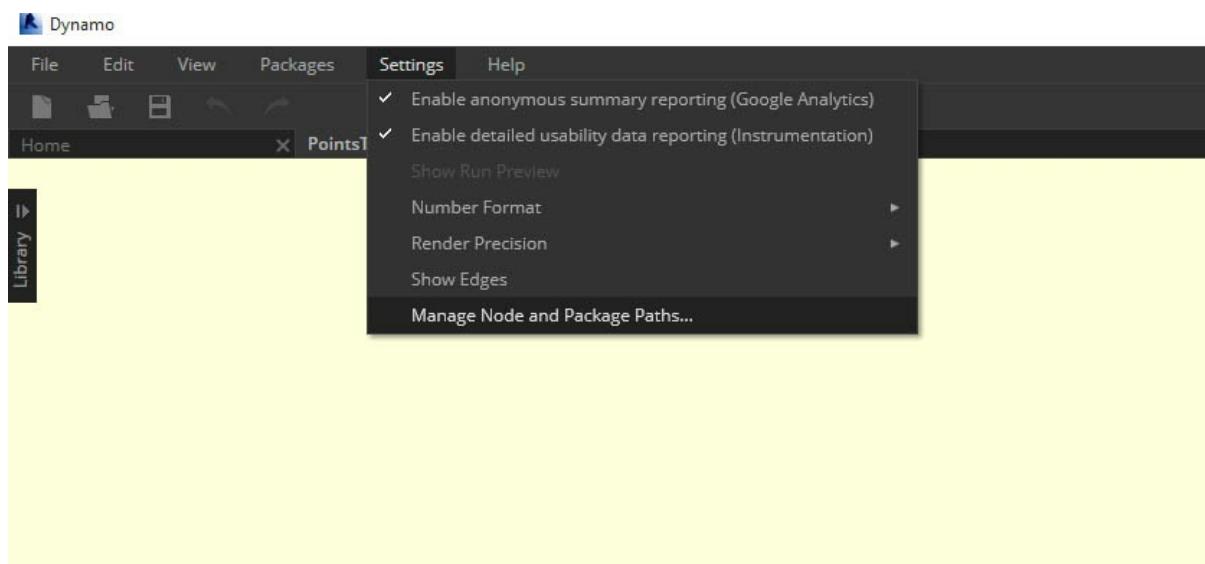


Geben Sie wie in der Abbildung oben gezeigt die nötigen Informationen ein und wählen Sie *Lokal publizieren*.

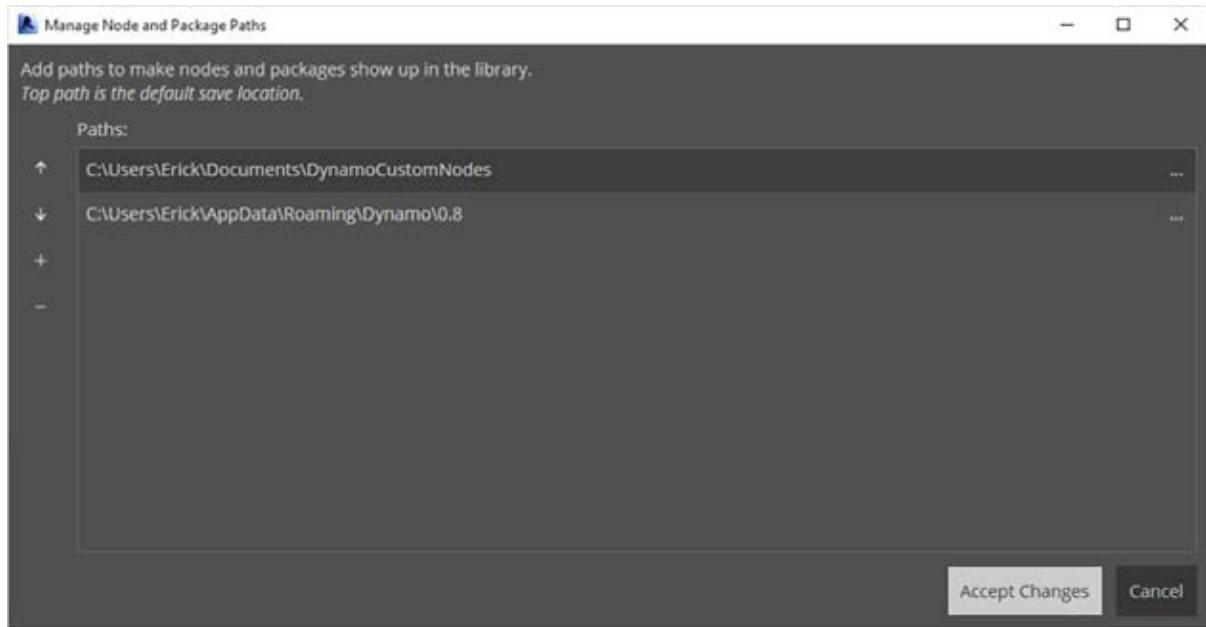
Beachten Sie, dass das Feld Gruppe den Haupteintrag angibt, der über das Dynamo-Menü aufgerufen wird.



Wählen Sie einen Ordner, in dem alle benutzerdefinierten Blöcke gespeichert werden sollen, die Sie lokal veröffentlichen werden. Dynamo prüft diesen Ordner jedes Mal beim Laden der Anwendung. Achten Sie daher darauf, dass der Ordner sich an einem dauerhaften Speicherort befindet. Navigieren Sie zu diesem Ordner und wählen Sie *Ordner auswählen*. Damit haben Sie den Dynamo-Block lokal veröffentlicht. Er steht jetzt jedes Mal, wenn Sie das Programm laden, im Dynamo-Werkzeugkasten zur Verfügung.

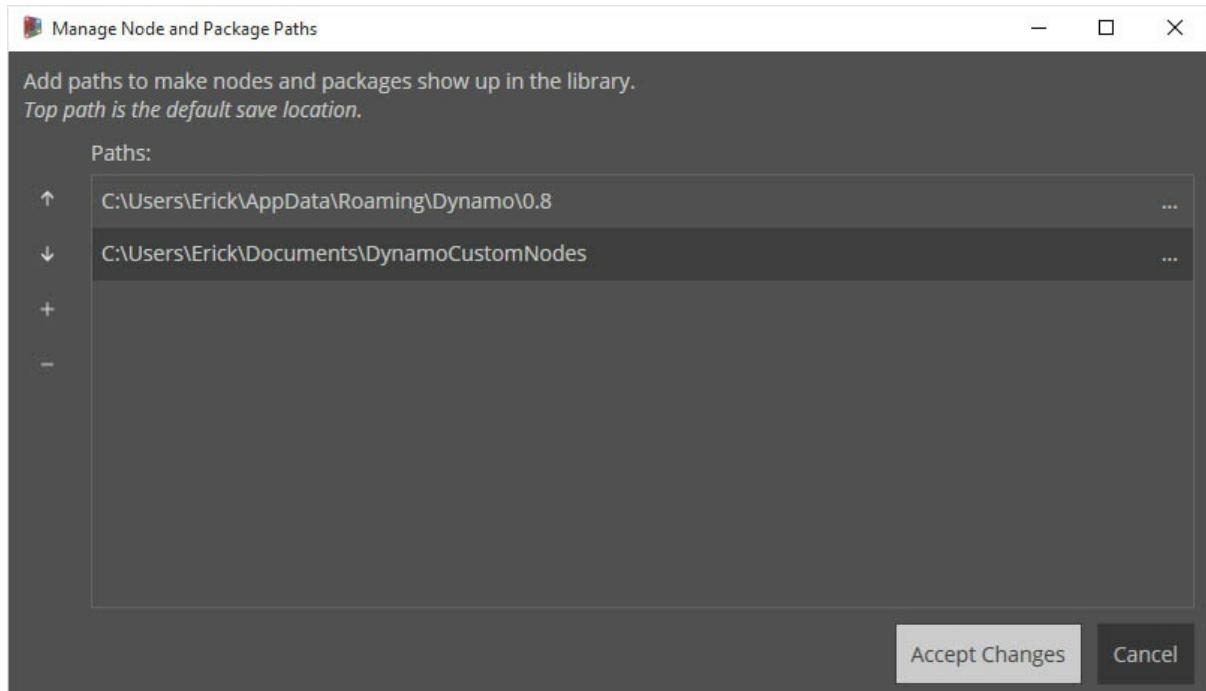


1. Um den Speicherort des benutzerdefinierten Blocks zu überprüfen, wechseln Sie zu *Einstellungen > Pfade für Blöcke und Pakete verwalten*.

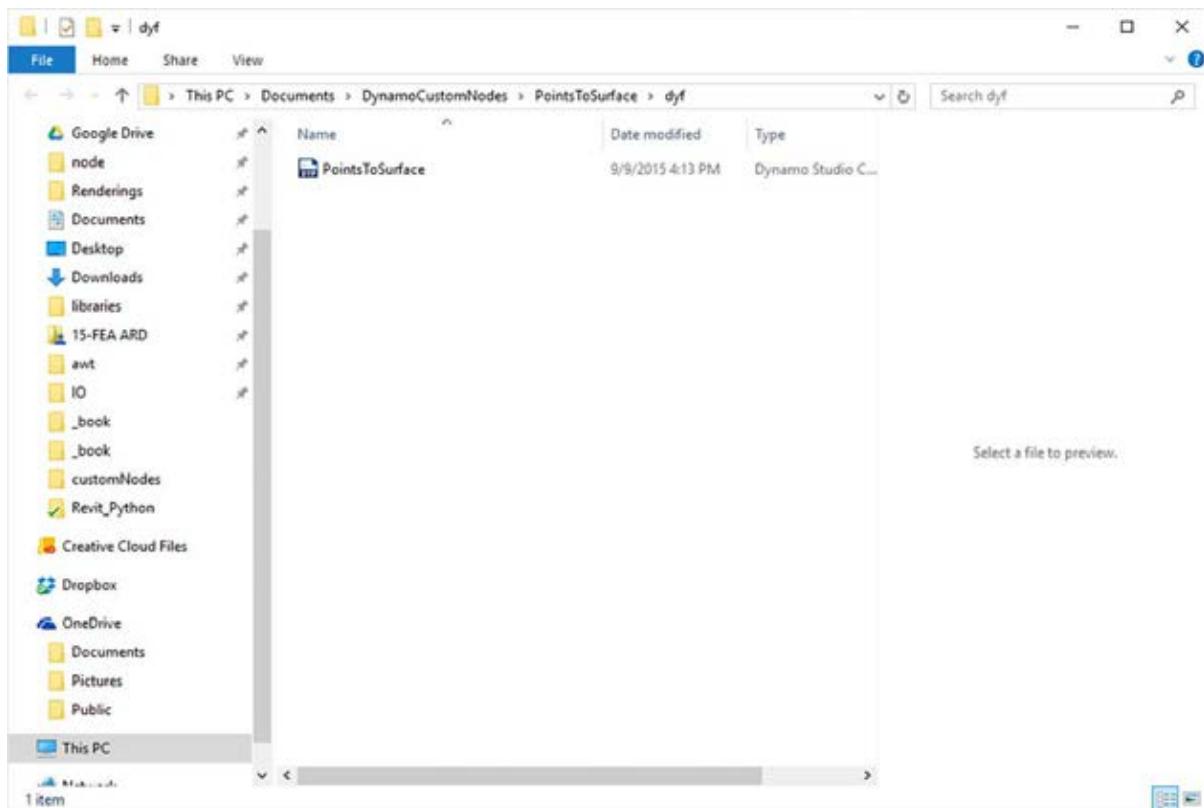


In diesem Fenster werden zwei Pfade angezeigt: `AppData\Roaming\Dynamic...` ist der vorgegebene Speicherort der online installierten Dynamo-Pakete. `Documents\DynamicCustomNodes...` gibt den Speicherort der von Ihnen veröffentlichten benutzerdefinierten Blöcke an.\*

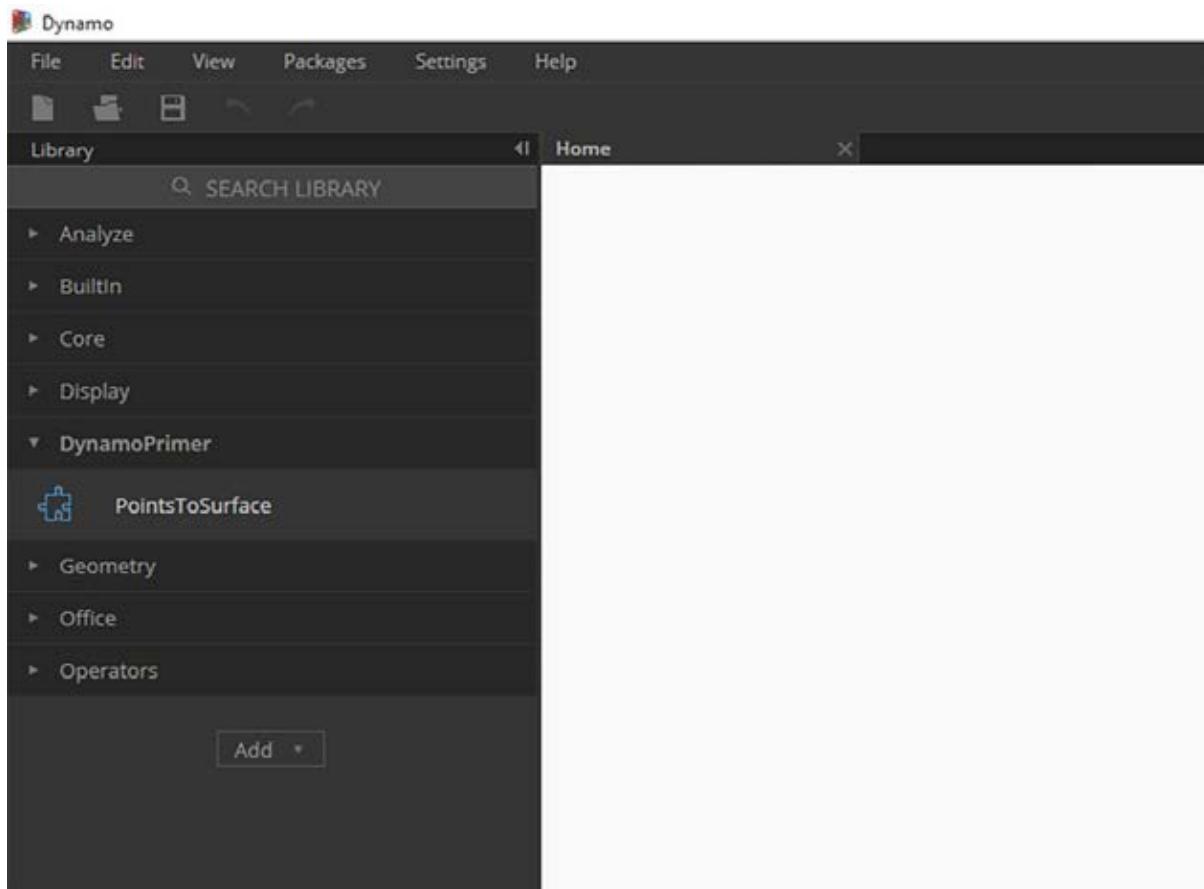
1. Es ist sinnvoll, den Pfad des lokalen Ordners in der oben gezeigten Liste nach unten zu verschieben (indem Sie den Ordnerpfad auswählen und auf den nach unten zeigenden Pfeil links neben den Pfadnamen klicken). Der zuoberst stehende Ordner ist die Vorgabe für die Installation von Paketen. Indem Sie den vorgegebenen Installationspfad für Dynamo-Pakete als Vorgabe beibehalten, stellen Sie daher sicher, dass Online-Pakete und Ihre lokal veröffentlichten Blöcke separat abgelegt werden.\*



Hier wurde die Reihenfolge der Pfadnamen vertauscht, damit Pakete unter dem Vorgabepfad von Dynamo installiert werden.



Wenn Sie zu diesem lokalen Ordner navigieren, finden Sie den ursprünglichen benutzerdefinierten Block im Ordner **dyf**. Dies ist die Erweiterung von Dateien für benutzerdefinierte Dynamo-Blöcke. Sie können die Datei in diesem Ordner bearbeiten. Der Block wird dann in der Benutzeroberfläche aktualisiert. Sie können auch weitere Blöcke im Ordner **DynamoCustomNode** hinzufügen. Diese werden beim Neustart von Dynamo Ihrer Bibliothek hinzugefügt.



Wenn Sie Dynamo jetzt laden, wird der Block *PointsToSurface* jedes Mal in der Gruppe *DynamoPrimer* Ihrer Dynamo-Bibliothek angezeigt.

# Python

# Python

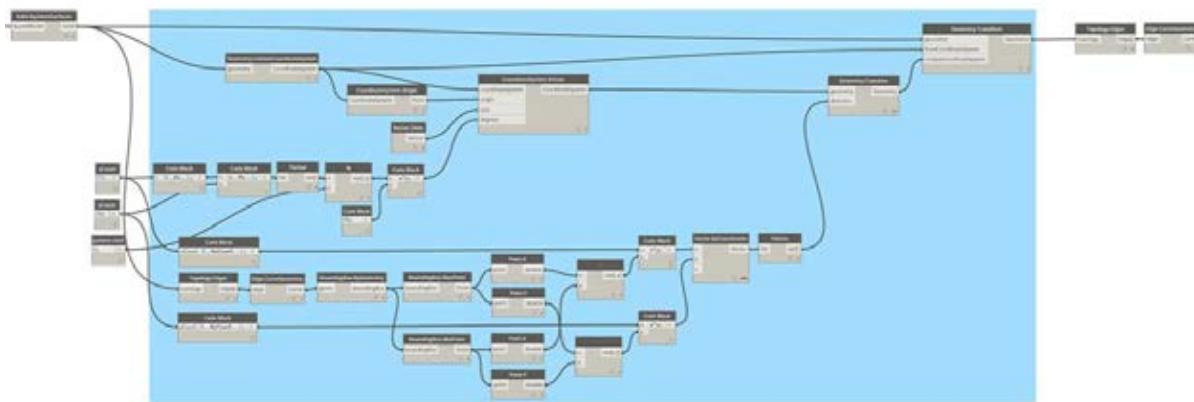


Python ist eine häufig verwendete Programmiersprache, die sich aufgrund ihrer Syntax großer Beliebtheit erfreut. Sie ist leicht zu lesen und damit leichter zu erlernen als viele andere Sprachen. Python unterstützt Module und Pakete und kann in bestehende Anwendungen eingebettet werden. Für die Beispiele in diesem Abschnitt werden Python-Grundkenntnisse vorausgesetzt. Eine geeignete Ressource für den Einstieg in Python finden Sie auf der Seite [Getting Started](#) auf [Python.org](#).

# Visuelle und Textprogrammierung im Vergleich

Wozu dient die Textprogrammierung in der visuellen Programmierungsumgebung von Dynamo? Die visuelle Programmierung bietet, wie in Kapitel 1.1 beschrieben, zahlreiche Vorteile. Sie ermöglicht es, in einer intuitiven visuellen Oberfläche Programme zu entwickeln, ohne eine spezielle Syntax zu erlernen. Visuelle Programme können jedoch recht unübersichtlich werden und enthalten zuweilen nicht genügend Funktionalität. So stehen in Python beispielsweise wesentlich praktischere Methoden zum Schreiben von Bedingungsanweisungen (if/then) und für Schleifen zur Verfügung. Python ist ein leistungsstarkes Werkzeug, das das Funktionsspektrum von Dynamo erweitern und Ihnen die Möglichkeit geben kann, eine große Gruppe von Blöcken durch einige wenige präzise Codezeilen zu ersetzen.

## Visuelles Programm:



### **Textprogramm:**

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

solid = IN[0]
seed = IN[1]
xCount = IN[2]
yCount = IN[3]

solids = []
```

```

yDist = solid.BoundingBox.MaxPoint.Y-solid.BoundingBox.MinPoint.Y
xDist = solid.BoundingBox.MaxPoint.X-solid.BoundingBox.MinPoint.X

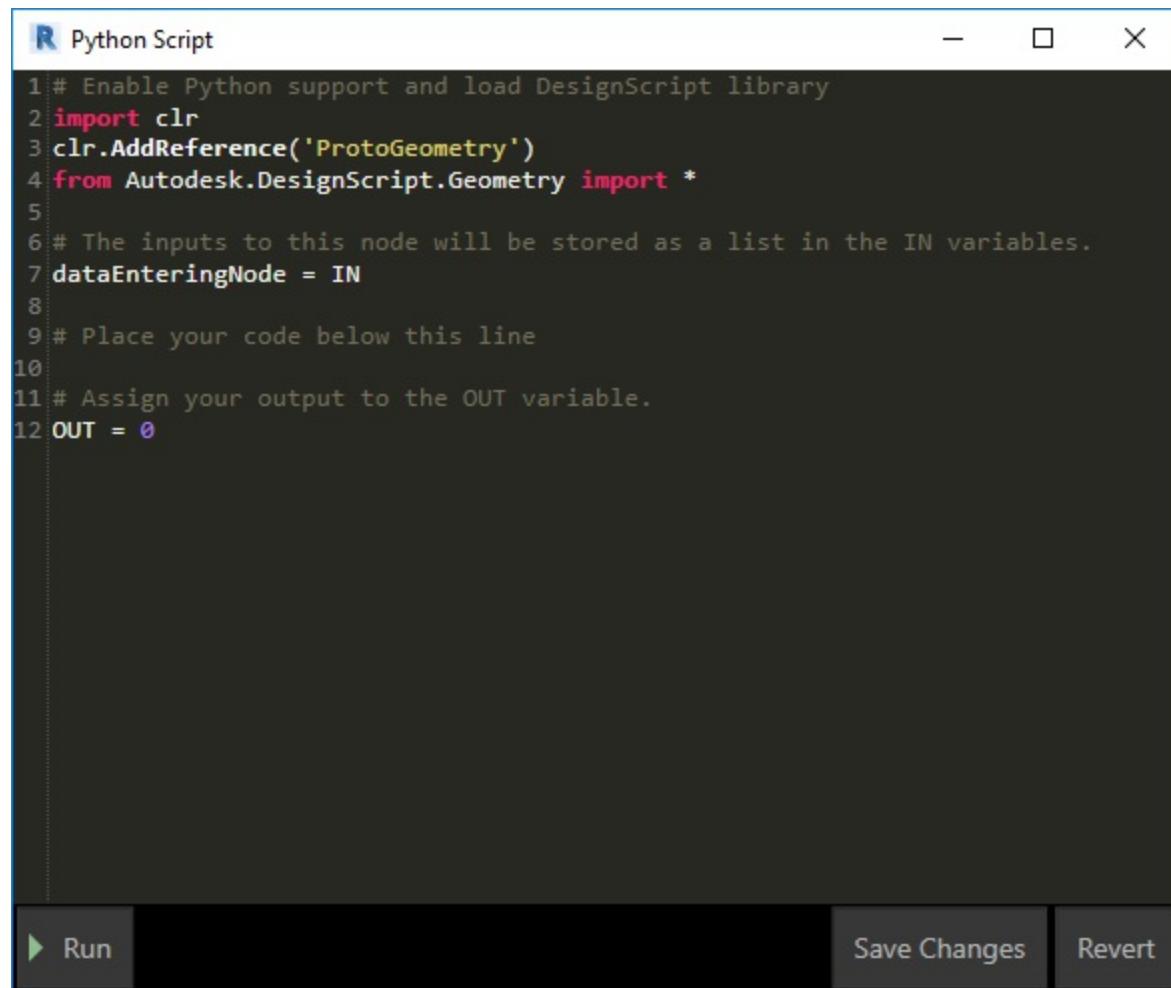
for i in xRange:
    for j in yRange:
        fromCoord = solid.ContextCoordinateSystem
        toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin, Vector.ByCoordinates
        vec = Vector.ByCoordinates((xDist*i), (yDist*j), 0)
        toCoord = toCoord.Translate(vec)
        solids.append(solid.Transform(fromCoord, toCoord))

OUT = solids

```

## Der Python-Block

Python-Blöcke sind genau wie Codeblöcke eine Scripting-Oberfläche innerhalb einer Umgebung für die visuelle Programmierung. Der Python-Block befindet sich unter *Core > Scripting* in der Bibliothek. Durch Doppelklicken auf den Block wird der Python-Skript-Editor geöffnet. (Sie können auch mit der rechten Maustaste auf den Block klicken und *Bearbeiten ... auswählen*.)



The screenshot shows a Python script editor window titled "Python Script". The code area contains the following Python script:

```

1 # Enable Python support and load DesignScript library
2 import clr
3 clr.AddReference('ProtoGeometry')
4 from Autodesk.DesignScript.Geometry import *
5
6 # The inputs to this node will be stored as a list in the IN variables.
7 dataEnteringNode = IN
8
9 # Place your code below this line
10
11 # Assign your output to the OUT variable.
12 OUT = 0

```

At the bottom of the window, there are three buttons: "Run" (with a play icon), "Save Changes", and "Revert".

Oben auf dem Bildschirm befindet sich vorgegebener Text, der es Ihnen erleichtern soll, die benötigten Bibliotheken zu referenzieren. Eingaben werden in der IN-Reihe gespeichert. Werte werden durch Zuweisung zur OUT-Variablen an Dynamo zurückgegeben.

In der Autodesk.DesignScript.Geometry-Bibliothek können Sie Punktnotation ähnlich wie in Codeblöcken verwenden. Weitere Informationen zur Dynamo-Syntax finden Sie in Kapitel 7.2 sowie im [DesignScript-Handbuch](#). Wenn Sie einen Geometrietyp, z. B. 'Point.' eingeben, wird eine Liste mit den Methoden zum Erstellen und Abfragen von Punkten angezeigt.

The screenshot shows the Python Script editor window in Dynamo. The script code is as follows:

```
1 # Enable Python support and load DesignScript library
2 import clr
3 clr.AddReference('ProtoGeometry')
4 from Autodesk.DesignScript.Geometry import *
5
6 # The inputs to this node will be stored as a list in the IN variables.
7 dataEnteringNode = IN
8
9 # Place your code below this line
10 Point.
11
12 # Assign values to OUT variable.
13 OUT = 0
14
```

A code completion dropdown is open at the end of the line "10 Point.". The list includes methods like Add, Approximate, AsVector, BoundingBox, ByCartesianCoordinate, ByCoordinates, ByCylindricalCoordinate, BySphericalCoordinate, ClosestPointTo, ContextCoordinateSys, Dispose, DistanceTo, and DoesIntersect.

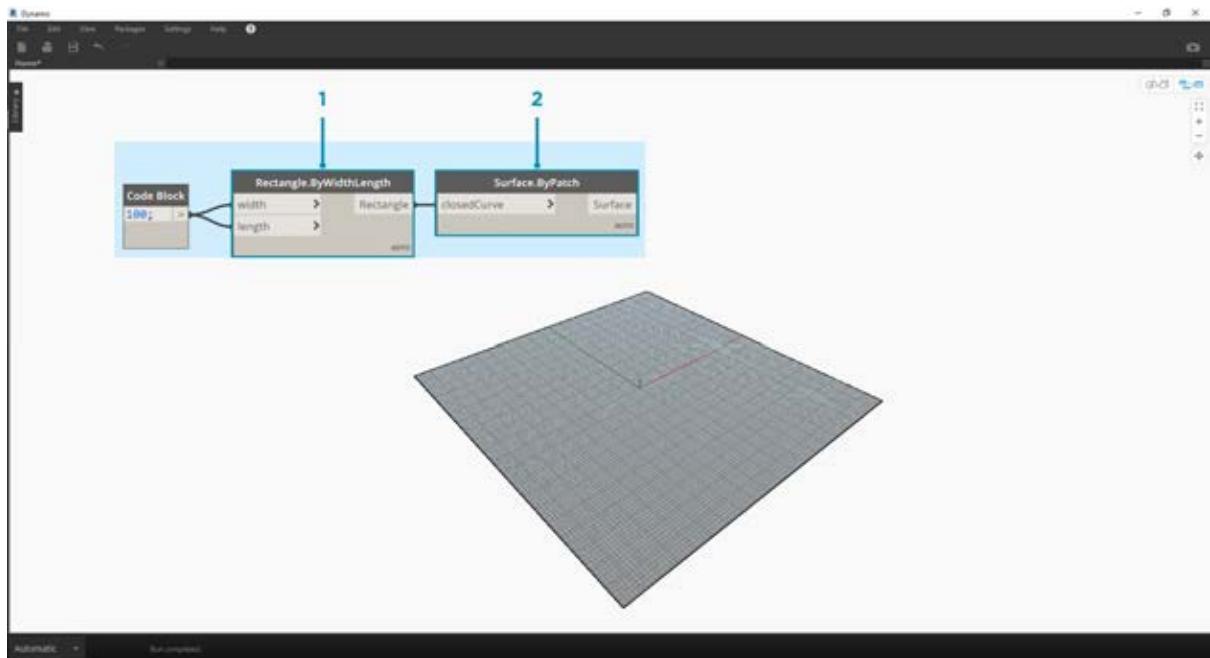
At the bottom of the editor are buttons for Run, Save Changes, and Revert.

Zu den Methoden gehören Konstruktoren, wie *ByCoordinates*, Aktionen wie *Add* und Abfragen wie X-, Y- und Z-Koordinaten.

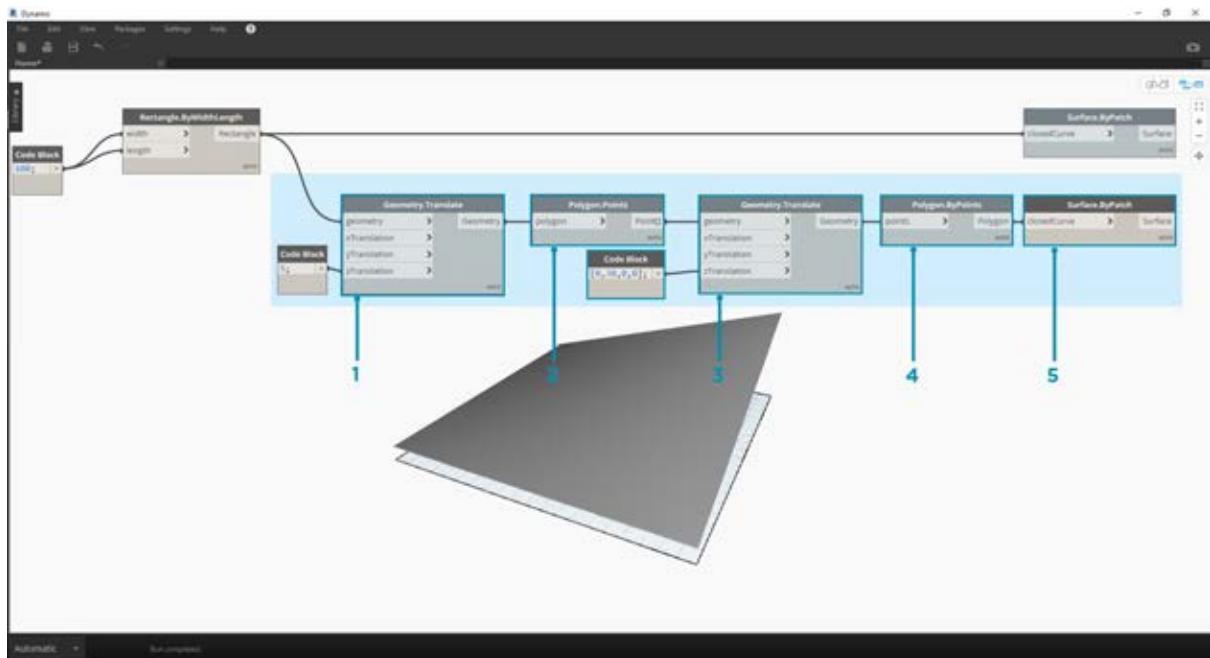
## Übung

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Python\\_Custom-Node.dyn](#)

In diesem Beispiel schreiben Sie ein Python-Skript zum Erstellen von Mustern aus einem Körpermodul und wandeln das Skript in einen benutzerdefinierten Block um. Zuerst erstellen Sie das Körpermodul mithilfe von Dynamo-Blöcken.

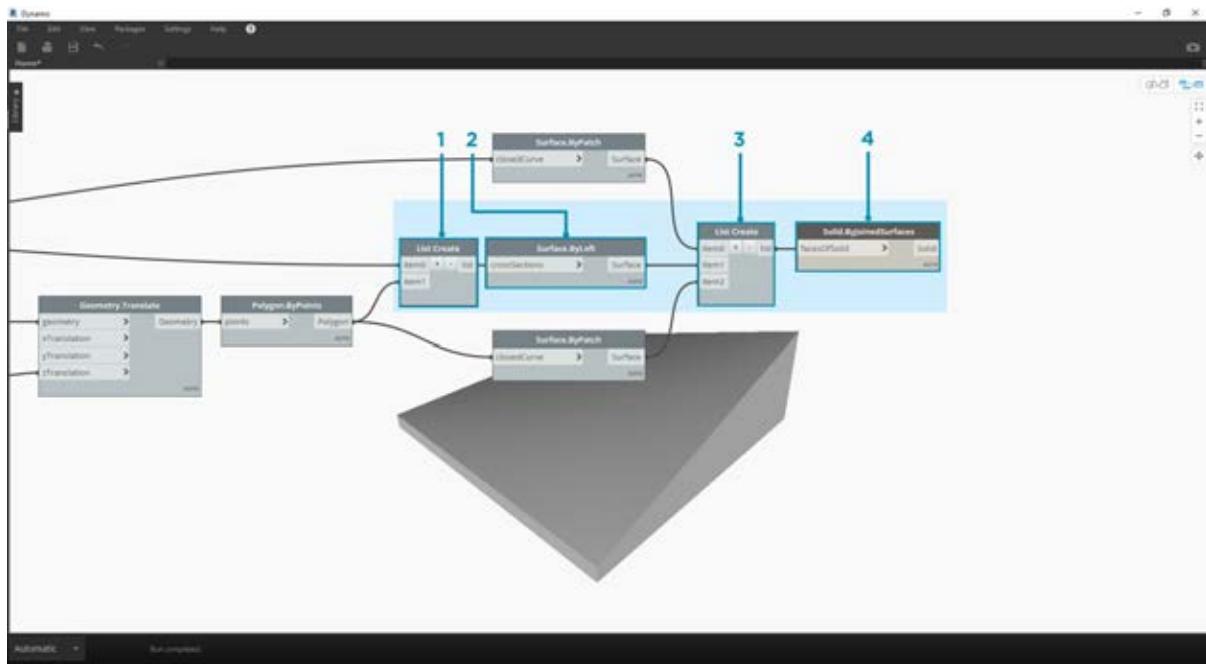


1. **Rectangle.ByWidthLength:** Erstellen Sie ein Rechteck, das als Basis für den Körper verwendet wird.
2. **Surface.ByPatch:** Verbinden Sie das Rechteck mit der *closedCurve*-Eingabe, um die untere Oberfläche zu erstellen.



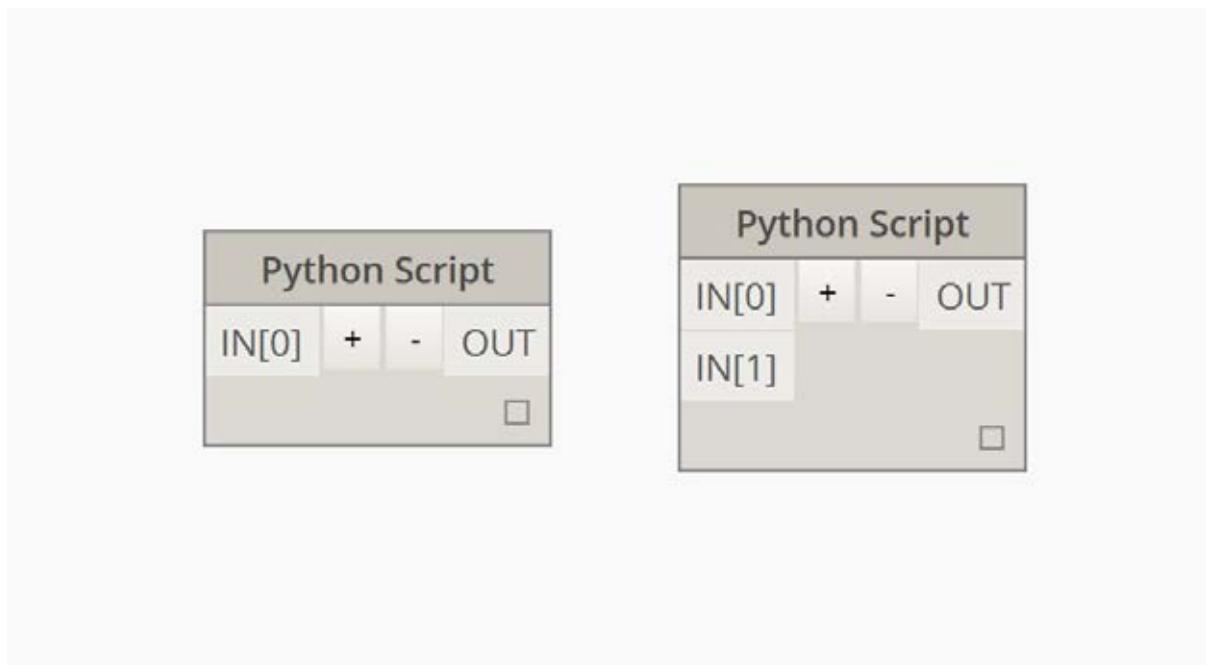
1. **Geometry.Translate:** Verbinden Sie das Rechteck mit der *geometry*-Eingabe, um es nach oben zu verschieben, wobei Sie mithilfe eines Codeblocks die allgemeine Dicke des Körpers festlegen.
2. **Polygon.Points:** Fragen Sie die Eckpunkte des verschobenen Rechtecks ab.
3. **Geometry.Translate:** Erstellen Sie mithilfe eines Codeblocks eine Liste mit vier Werten für die vier Punkte, wobei Sie eine Ecke des Körpers nach oben verschieben.
4. **Polygon.ByPoints:** Erstellen Sie das obere Polygon aus den verschobenen Punkten erneut.
5. **Surface.ByPatch:** Schließen Sie das Polygon, um die obere Oberfläche zu erstellen.

Damit haben Sie die obere und untere Oberfläche erstellt. Erstellen Sie jetzt durch eine Erhebung zwischen den beiden Profilen die Seiten des Körpers.



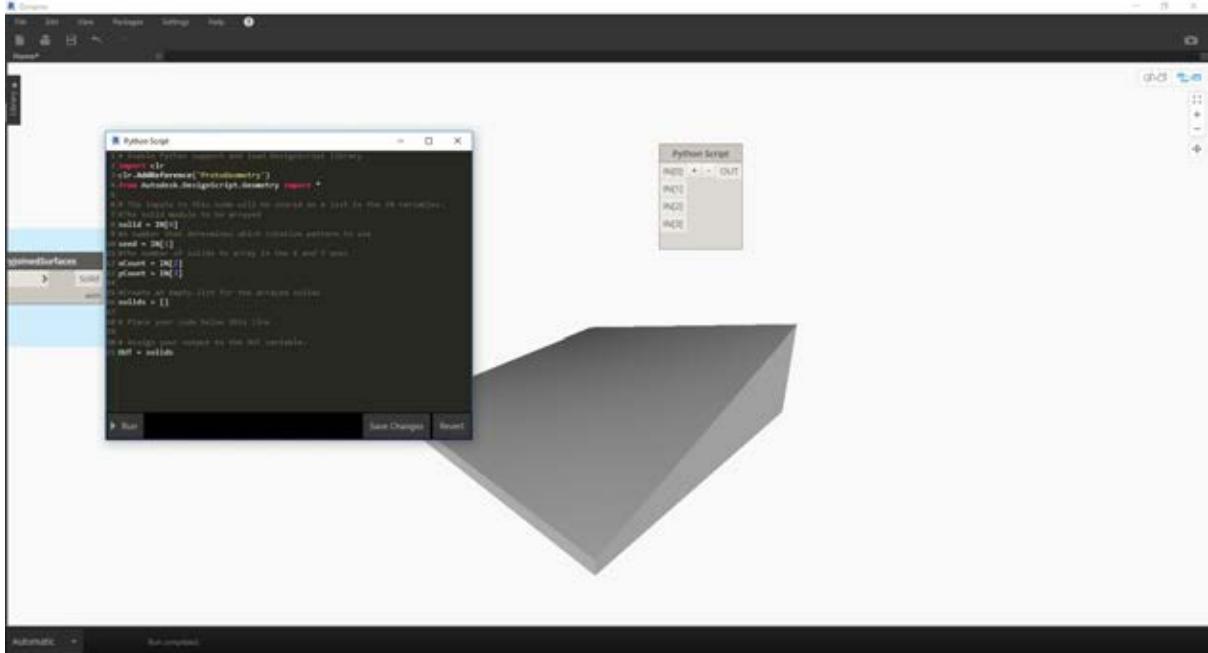
1. **List.Create:** Verbinden Sie das Rechteck unten und das Polygon oben mit den index-Eingaben.
2. **Surface.ByLoft:** Erstellen Sie über eine Erhebung die Seiten des Körpers.
3. **List.Create:** Verbinden Sie die obere und untere sowie die seitlichen Oberflächen mit den index-Eingaben, um eine Liste der Oberflächen zu erhalten.
4. **Solid.ByJoinedSurfaces:** Verbinden Sie die Flächen, um das Körpermodul zu erstellen.

Damit haben Sie den Körper erstellt. Fügen Sie jetzt einen Block für das Python-Skript in den Arbeitsbereich ein.



Um dem Block weitere Eingaben hinzuzufügen, schließen Sie den Editor und klicken Sie auf das +-Symbol im Block. Die Eingaben erhalten die Namen IN[0], IN[1] usw., um anzusehen, dass sie Einträge in einer Liste darstellen.

Sie beginnen, indem Sie die Ein- und Ausgaben definieren. Doppelklicken Sie auf den Block, um den Python-Editor zu öffnen.



```

# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]
#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#create an empty list for the arrayed solids
solids = []

# Place your code below this line

# Assign your output to the OUT variable.
OUT = solids

```

Dieser Code wird im weiteren Verlauf der Übung leichter verständlich. Als Nächstes müssen Sie überlegen, welche Informationen Sie zum Erstellen einer Reihe aus dem Körpermodul benötigen. Als Erstes müssen Sie die Maße des Körpers kennen, um die Entfernung für die Verschiebung zu ermitteln. Wegen eines Fehlers bei Begrenzungsrahmen müssen Sie diesen anhand der Kurvengeometrie der Kanten erstellen.

```
1 # Enable Python support and load DesignScript library
2 import clr
3 clr.AddReference('ProtoGeometry')
4 from Autodesk.DesignScript.Geometry import *
5
6 # The inputs to this node will be stored as a list in the IN variables.
7 #The solid module to be arrayed
8 solid = IN[0]
9 #A number that determines which rotation pattern to use
10 seed = IN[1]
11 #The number of solids to array in the X and Y axes
12 xCount = IN[2]
13 yCount = IN[3]
14
15 #Create an empty list for the arrayed solids
16 solids = []
17 # Create an empty list for the edge curves
18 crvs = []
19
20 # Place your code below this line
21 #Loop through edges and append corresponding curve geometry to the list
22 for edge in solid.Edges:
23     crvs.append(edge.CurveGeometry)
24 #Get the bounding box of the curves
25 bbox = BoundingBox.ByGeometry(crvs)
26
27 #Get the X and Y translation distance based on the bounding box
28 yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
29 xDist = bbox.MaxPoint.X-bbox.MinPoint.X
30
31 # Assign your output to the OUT variable.
32 OUT = solids
```

Run Save Changes Revert

Ein Blick auf den Python-Block in Dynamo. Dieselbe Syntax wie in den Titeln der Blöcke in Dynamo wird auch hier verwendet. Im Folgenden sehen Sie den kommentierten Code.

```
# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]
#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#Create an empty list for the arrayed solids
solids = []
# Create an empty list for the edge curves
crvs = []

# Place your code below this line
#Loop through edges and append corresponding curve geometry to the list
```

```

for edge in solid.Edges:
    crvs.append(edge.CurveGeometry)
#Get the bounding box of the curves
bbox = BoundingBox.ByGeometry(crvs)

#Get the X and Y translation distance based on the bounding box
yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
xDist = bbox.MaxPoint.X-bbox.MinPoint.X

# Assign your output to the OUT variable.
OUT = solids

```

Da die Körpermodule sowohl verschoben als auch gedreht werden sollen, verwenden Sie hier die Geometry.Transform-Operation. Wenn Sie den Geometry.Transform-Block genauer betrachten, sehen Sie, dass für die Transformation des Körpers ein Quell- und ein Zielkoordinatensystem benötigt werden. Die Quelle ist das Koordinatensystem, das den Kontext für den Ausgangskörper bildet, während als Ziel ein eigenes Koordinatensystem für jedes Modul in der Reihe verwendet wird. Das bedeutet, dass Sie die x- und y-Werte in einer Schleife verarbeiten müssen, um das Koordinatensystem jedes Mal auf andere Weise zu transformieren.

```

R Python Script
1 # Enable Python support and load DesignScript library
2 import clr
3 clr.AddReference('ProtoGeometry')
4 from Autodesk.DesignScript.Geometry import *
5
6 # The inputs to this node will be stored as a list in the IN variables.
7 #The solid module to be arrayed
8 solid = IN[0]
9 #A number that determines which rotation pattern to use
10 seed = IN[1]
11 #The number of solids to array in the X and Y axes
12 xCount = IN[2]
13 yCount = IN[3]
14
15 #Create an empty list for the arrayed solids
16 solids = []
17 # Create an empty list for the edge curves
18 crvs = []
19
20 #Place your code below this line
21 #Loop through edges and append corresponding curve geometry to the list
22 for edge in solid.Edges:
23     crvs.append(edge.CurveGeometry)
24 #Get the bounding box of the curves
25 bbox = BoundingBox.ByGeometry(crvs)
26
27 #Get the X and Y translation distance based on the bounding box
28 yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
29 xDist = bbox.MaxPoint.X-bbox.MinPoint.X
30 #Get the source coordinate system
31 fromCoord = solid.ContextCoordinateSystem
32
33 #Loop through X and Y
34 for i in range(xCount):
35     for j in range(yCount):
36         #Rotate and translate the coordinate system
37         toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates(0,0,1),(90*(i+j%seed)))
38         vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
39         toCoord = toCoord.Translate(vec)
40         #Transform the solid from the source coord system to the target coord system and append to the list
41         solids.append(solid.Transform(fromCoord,toCoord))
42
43 # Assign your output to the OUT variable.
44 OUT = solids

```

Ein Blick auf den Python-Block in Dynamo. Im Folgenden sehen Sie den kommentierten Code.

```

# Enable Python support and load DesignScript library
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# The inputs to this node will be stored as a list in the IN variables.
#The solid module to be arrayed
solid = IN[0]
#A number that determines which rotation pattern to use
seed = IN[1]

```

```

#The number of solids to array in the X and Y axes
xCount = IN[2]
yCount = IN[3]

#Create an empty list for the arrayed solids
solids = []
# Create an empty list for the edge curves
crvs = []

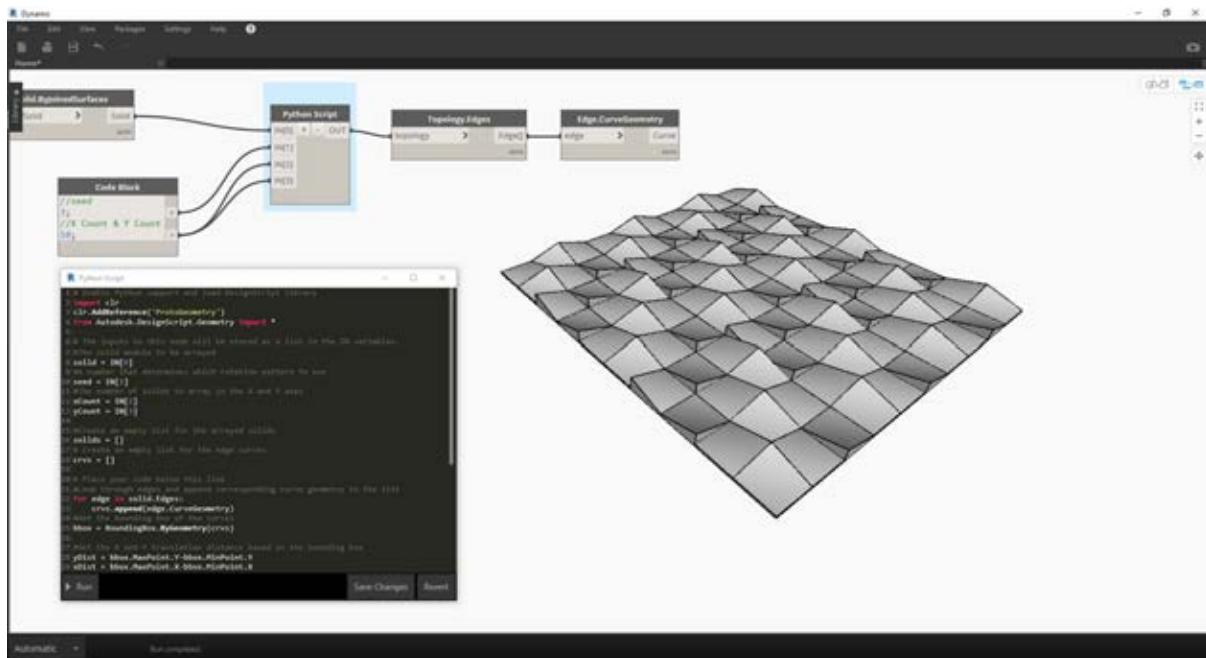
# Place your code below this line
#Loop through edges and append corresponding curve geometry to the list
for edge in solid.Edges:
    crvs.append(edge.CurveGeometry)
#Get the bounding box of the curves
bbox = BoundingBox.ByGeometry(crvs)

#Get the X and Y translation distance based on the bounding box
yDist = bbox.MaxPoint.Y-bbox.MinPoint.Y
xDist = bbox.MaxPoint.X-bbox.MinPoint.X
#get the source coordinate system
fromCoord = solid.ContextCoordinateSystem

#Loop through X and Y
for i in range(xCount):
    for j in range(yCount):
        #Rotate and translate the coordinate system
        toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates
        vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
        toCoord = toCoord.Translate(vec)
        #Transform the solid from the source coord system to the target coord system and app
        solids.append(solid.Transform(fromCoord,toCoord))

# Assign your output to the OUT variable.
OUT = solids

```

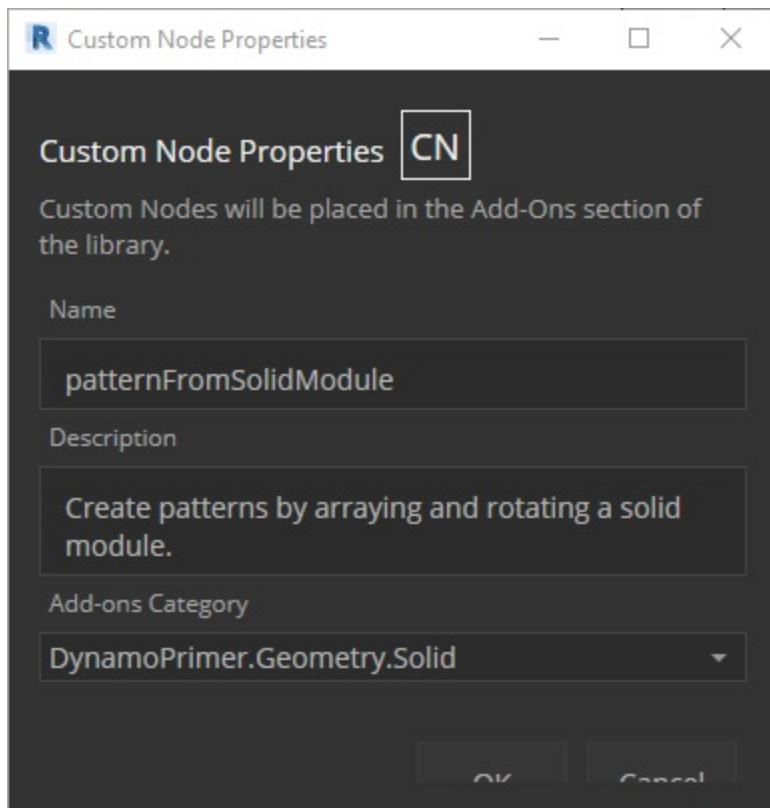


Durch Klicken auf den Python-Block können wir unseren Code ausführen.



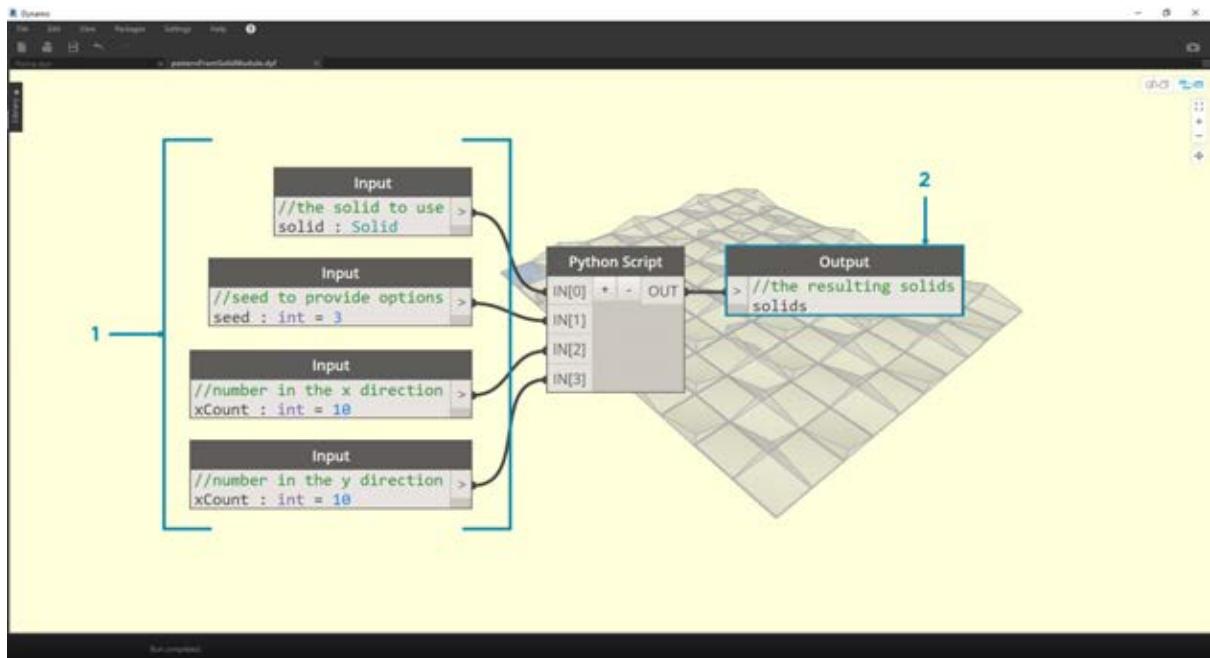
Ändern Sie den Wert für die Ausgangszahl, um verschiedene Muster zu erstellen. Indem Sie die Parameter des Körpermoduls selbst ändern, erzielen Sie ebenfalls unterschiedliche Wirkungen. Sie können Ausgangswerte in Dynamo 2.0 einfach ändern und auf Ausführen klicken, ohne das Python-Fenster zu schließen.

Damit haben Sie ein nützliches Python-Skript erstellt. Speichern Sie dieses jetzt als benutzerdefinierten Block. Wählen Sie den Python-Skript-Block aus, klicken Sie mit der rechten Maustaste darauf und wählen Sie Block aus Auswahl erstellen.

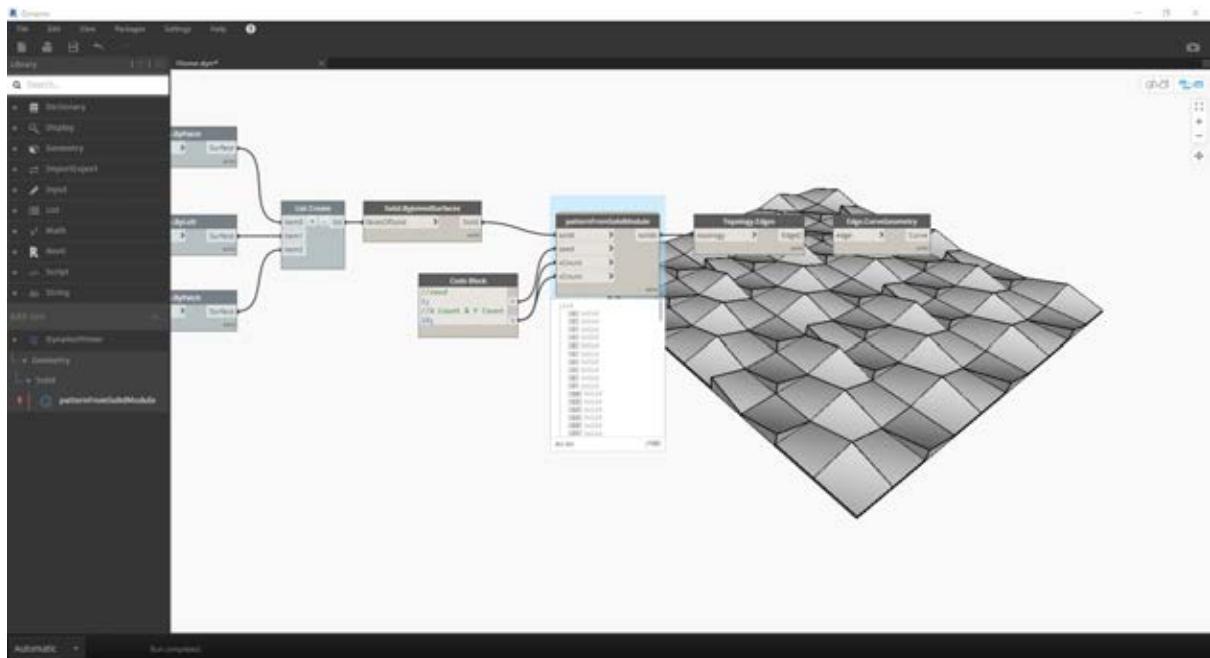


Weisen Sie einen Namen, eine Beschreibung und eine Kategorie zu.

Dadurch wird ein neuer Arbeitsbereich geöffnet, in dem Sie den benutzerdefinierten Block bearbeiten können.



1. **Eingabe-Blöcke:** Geben Sie den Eingaben aussagekräftigere Namen und fügen Sie Datentypen und Vorgabewerte hinzu.
2. **Ausgabe-Blöcke:** Ändern Sie den Namen der Ausgabe und speichern Sie den Block als .dyf-Datei.



Die vorgenommenen Änderungen werden in den benutzerdefinierten Block übernommen.

# Python und Revit

## Python und Revit

Nachdem im vorigen Abschnitt die Verwendung von Python-Skripts in Dynamo gezeigt wurde, erhalten Sie hier eine Einführung zur Einbindung von Revit-Bibliotheken in die Skriptumgebung. Rufen Sie sich kurz ins Gedächtnis zurück, dass Sie die Dynamo-Core-Blöcke mithilfe der ersten drei Zeilen im folgenden Codeabschnitt importiert haben. Um die Blöcke, Elemente und die Dokumentenverwaltung von Revit zu importieren, sind lediglich einige weitere Zeilen erforderlich:

```
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

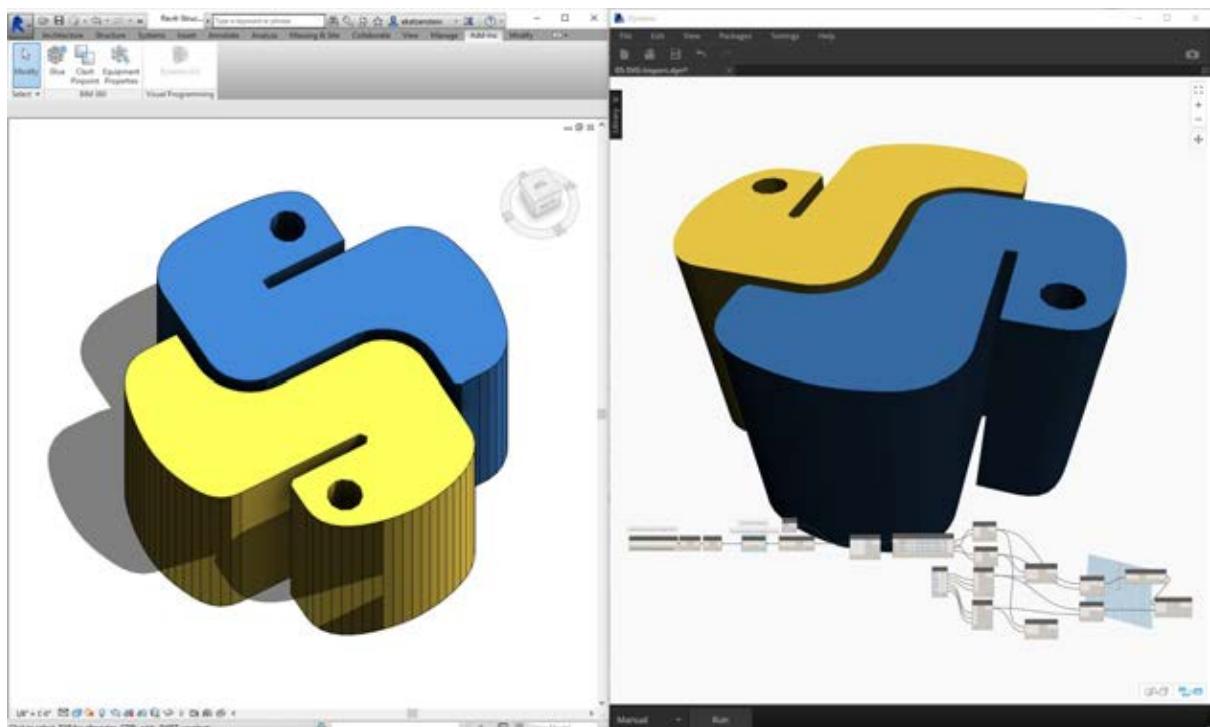
# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit

# Import Revit elements
from Revit.Elements import *

# Import DocumentManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager

import System
```

Dadurch erhalten Sie Zugriff auf die Revit-API und können benutzerdefinierte Skripte für beliebige Revit-Aufgaben erstellen. Die Kombination der visuellen Programmierung mit der Skripterstellung in der Revit-API bringt erhebliche Verbesserungen für die Zusammenarbeit und die Entwicklung von Werkzeugen mit sich. So könnten beispielsweise ein BIM-Manager und ein Schemaplanentwickler gemeinsam am selben Diagramm arbeiten. Bei dieser Zusammenarbeit können sie sowohl den Entwurf als auch die Realisierung des Modells verbessern.



## Plattformspezifische APIs

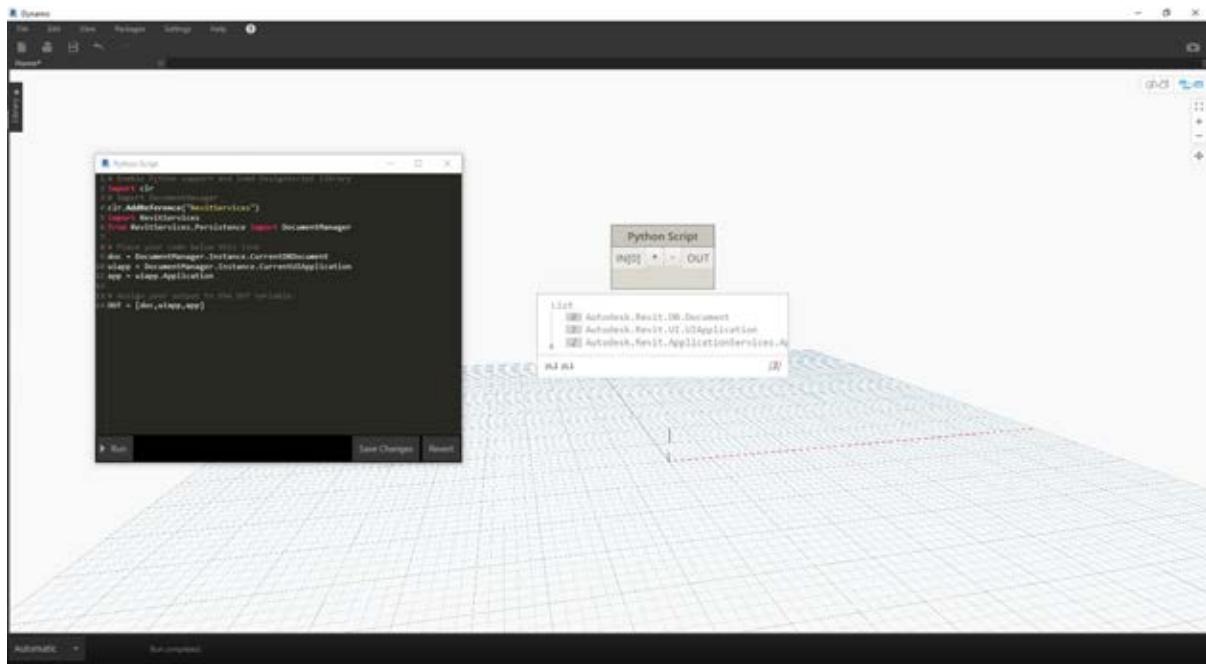
Mit dem Dynamo-Projekt ist beabsichtigt, die Möglichkeiten der Plattformimplementierung zu erweitern. Da Dynamo nach und nach weitere Programme in seine Palette aufnimmt, erhalten Benutzer Zugriff auf plattformspezifische APIs aus der Python-Skriptumgebung. In diesem Abschnitt wird ein Fallbeispiel für Revit behandelt. Zukünftig sollen jedoch weitere Kapitel mit umfassenden Lernprogrammen zur Skripterstellung für andere Plattformen bereitgestellt werden. Darüber hinaus stehen jetzt zahlreiche [IronPython](#)-Bibliotheken zur Verfügung, die Sie in Dynamo importieren können.

Die folgenden Beispiele zeigen Möglichkeiten zur Implementierung Revit-spezifischer Vorgänge aus Dynamo mit Python. Eine genauere Beschreibung der Beziehung zwischen Python einerseits und Dynamo und Revit andererseits finden Sie auf der [Wiki-Seite zu Dynamo](#). Eine andere nützliche Ressource für Python und Revit ist das [Revit Python Shell](#)-Projekt.

## Übungslektion 01

Erstellen Sie ein neues Revit-Projekt. Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Revit-Doc.dyn](#)

In diesen Übungen lernen Sie die grundlegenden Python-Skripts in Dynamo für Revit kennen. Dabei liegt das Hauptaugenmerk auf der Verarbeitung von Revit-Dateien und -Elementen sowie der Kommunikation zwischen Revit und Dynamo.



Dies ist eine einfache Methode zum Abrufen von `doc`, `uiapp` und `app` für die mit der Dynamo-Sitzung verknüpfte Revit-Datei. Programmierern, die zuvor bereits in der Revit-API gearbeitet haben, fallen eventuell die Einträge in der Liste des Watch-Blocks auf. Falls diese Einträge ungewohnt wirken, besteht kein Grund zur Beunruhigung. In den weiteren Übungen werden andere Beispiele verwendet.

Der folgende Code zeigt, wie Sie die Revit-Dienste importieren und die Dokumentdaten in Dynamo abrufen können:

A screenshot of the Dynamo software interface. In the center, there is a large workspace containing a 3D wireframe grid. Overlaid on the workspace is a Python Script component. The script window shows the following code:

```
1# Enable Python support and load DesignScript library
2import clr
3# Import DocumentManager
4clr.AddReference("RevitServices")
5import RevitServices
6from RevitServices.Persistence import DocumentManager
7
8# Place your code below this line
9doc = DocumentManager.Instance.CurrentDBDocument
10uiapp = DocumentManager.Instance.CurrentUIApplication
11app = uiapp.Application
12
13# Assign your output to the OUT variable.
14OUT = [doc,uiapp,app]
```

The results pane to the right of the script component displays a list titled "List" with three items:

- Autodesk.Revit.DB.Document
- Autodesk.Revit.UI.IUIApplication
- Autodesk.Revit.ApplicationServices.Application

At the bottom of the workspace, there are three buttons: "Run", "Save Changes", and "Revert".

Ein Blick auf den Python-Block in Dynamo. Im Folgenden sehen Sie den kommentierten Code.

```
# Enable Python support and load DesignScript library
import clr
# Import DocumentManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager

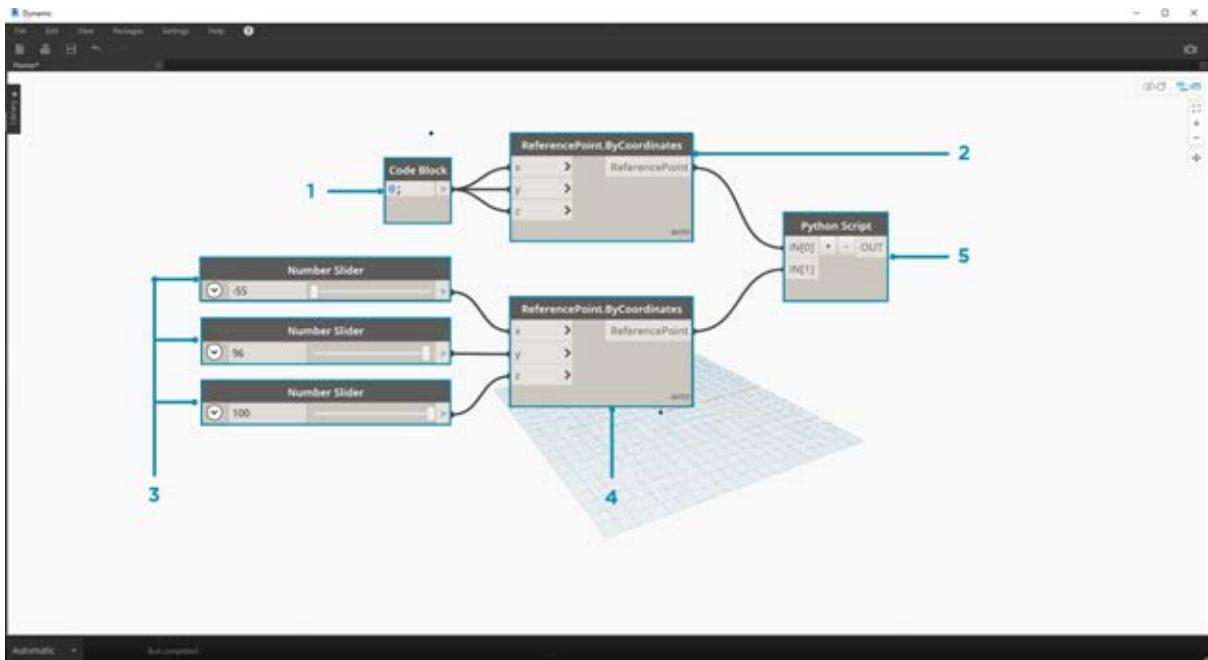
# Place your code below this line
doc = DocumentManager.Instance.CurrentDBDocument
uiapp = DocumentManager.Instance.CurrentUIApplication
app = uiapp.Application

# Assign your output to the OUT variable.
OUT = [doc,uiapp,app]
```

## Übungslektion 02

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As"). Eine vollständige Liste der Beispieldateien finden Sie im Anhang [Revit-ReferenceCurve.dyn](#)

In dieser Übung erstellen Sie mithilfe des Python-Blocks von Dynamo eine einfache Modellkurve in Revit.



Beginnen Sie mit der in der Abbildung oben gezeigten Gruppe von Blöcken. Sie erstellen zunächst mithilfe von Dynamo-Blöcken zwei Referenzpunkte in Revit.

Beginnen Sie, indem Sie in Revit eine neue Entwurfskörperfamilie erstellen. Starten Sie Dynamo und erstellen Sie die Gruppe von Blöcken in der Abbildung oben. Sie erstellen zunächst mithilfe von Dynamo-Blöcken zwei Referenzpunkte in Revit.

1. Erstellen Sie einen Codeblock weisen Sie ihm den Wert "0" zu.
2. Verbinden Sie diesen Wert mit den x-, y- und z-Eingaben eines ReferencePoint.ByCoordinates-Blocks.
3. Erstellen Sie drei Schieberegler mit dem Bereich zwischen -100 und 100 und der Schrittgröße 1.
4. Verbinden Sie die Schieberegler mit einem ReferencePoint.ByCoordinates-Block.
5. Fügen Sie einen Python-Block im Arbeitsbereich hinzu, klicken Sie auf die Schaltfläche +, um eine weitere Eingabe hinzuzufügen, und verbinden Sie die beiden Referenzpunkte mit den Eingaben. Öffnen Sie den Python-Block.

```
1 # Enable Python support and load DesignScript library
2 import clr
3 # Import RevitNodes
4 clr.AddReference("RevitNodes")
5 import Revit
6 # Import Revit elements
7 from Revit.Elements import *
8 import System
9
10 # The inputs to this node will be stored as a list in the IN variables.
11 startRefPt = IN[0]
12 endRefPt = IN[1]
13
14 # Place your code below this line
15 #define system array to match with required inputs
16 refPtArray = System.Array[ReferencePoint]([startRefPt, endRefPt])
17
18 # Assign your output to the OUT variable.
19 OUT = CurveByPoints.ByReferencePoints(refPtArray)
```

Run Save Changes Revert

Ein Blick auf den Python-Block in Dynamo. Im Folgenden sehen Sie den kommentierten Code.

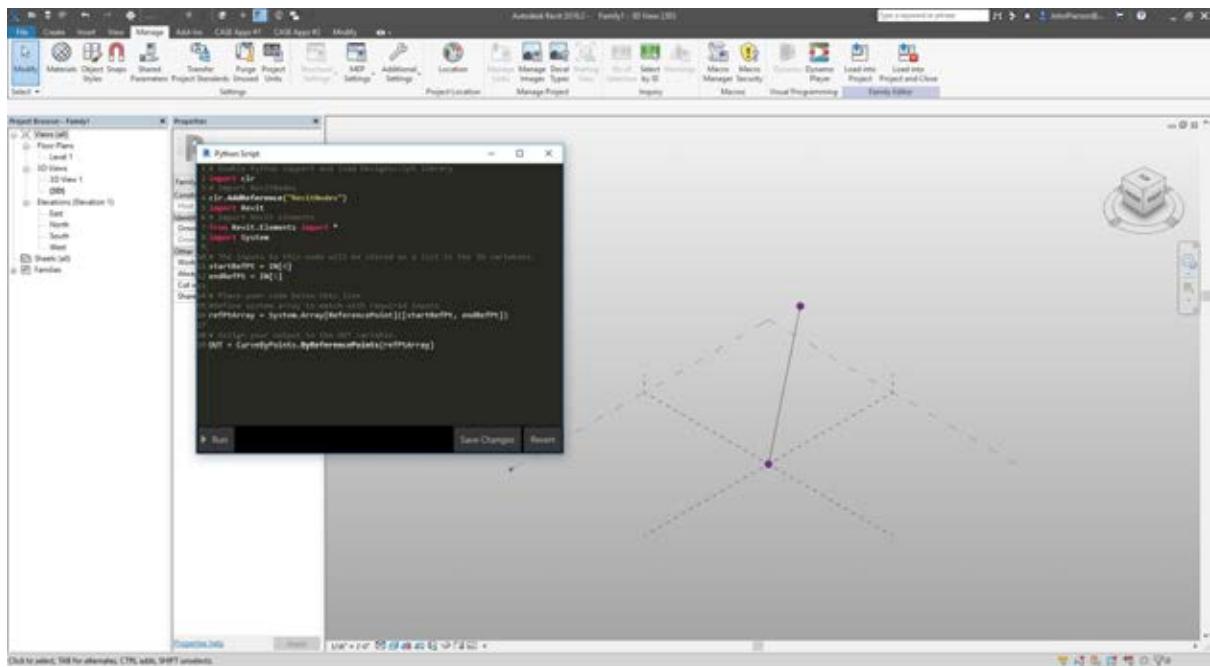
1. **System.Array:** Für Revit wird eine Systemreihe (anstelle einer Python-Liste) benötigt. Hierfür genügt eine weitere Codezeile. Indem Sie sorgfältig auf die Argumenttypen achten, erleichtern Sie jedoch die Python-Programmierung in Revit.

```
import clr

# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit
# Import Revit elements
from Revit.Elements import *
import System

#define inputs
startRefPt = IN[0]
endRefPt = IN[1]

#define system array to match with required inputs
refPtArray = System.Array[ReferencePoint]([startRefPt, endRefPt])
#create curve by reference points in Revit
OUT = CurveByPoints.ByReferencePoints(refPtArray)
```

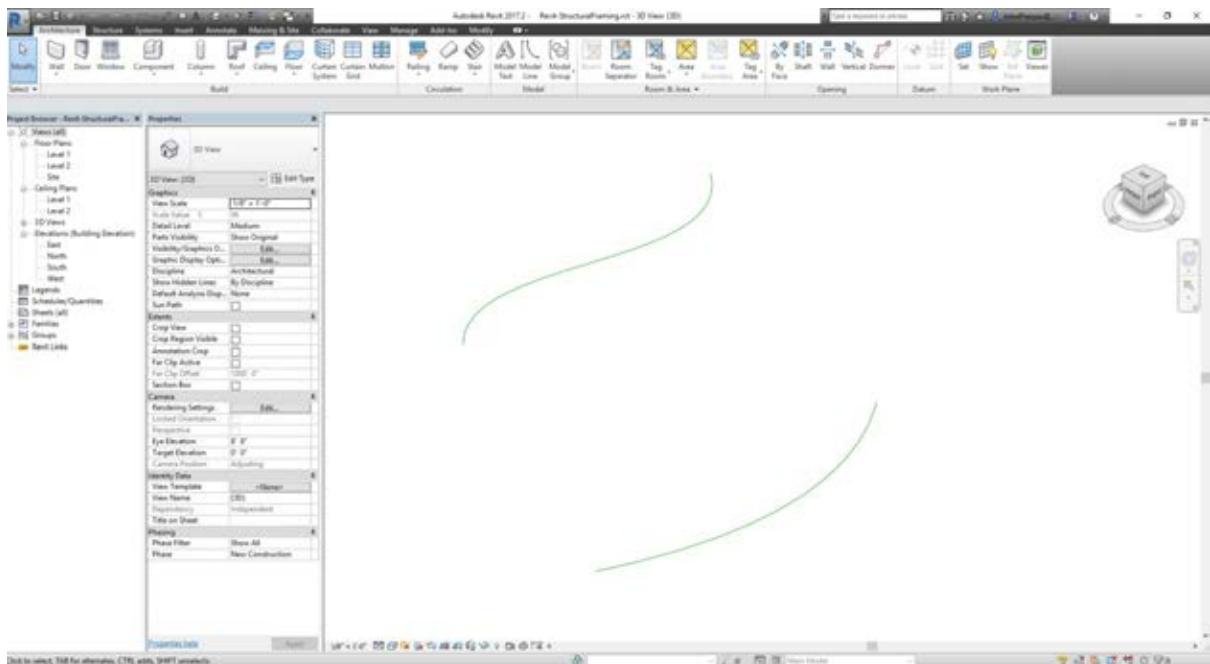


Sie haben in Dynamo zwei Referenzpunkte erstellt und diese in Python mit einer Linie verbunden. In der nächsten Übung führen Sie dies weiter.

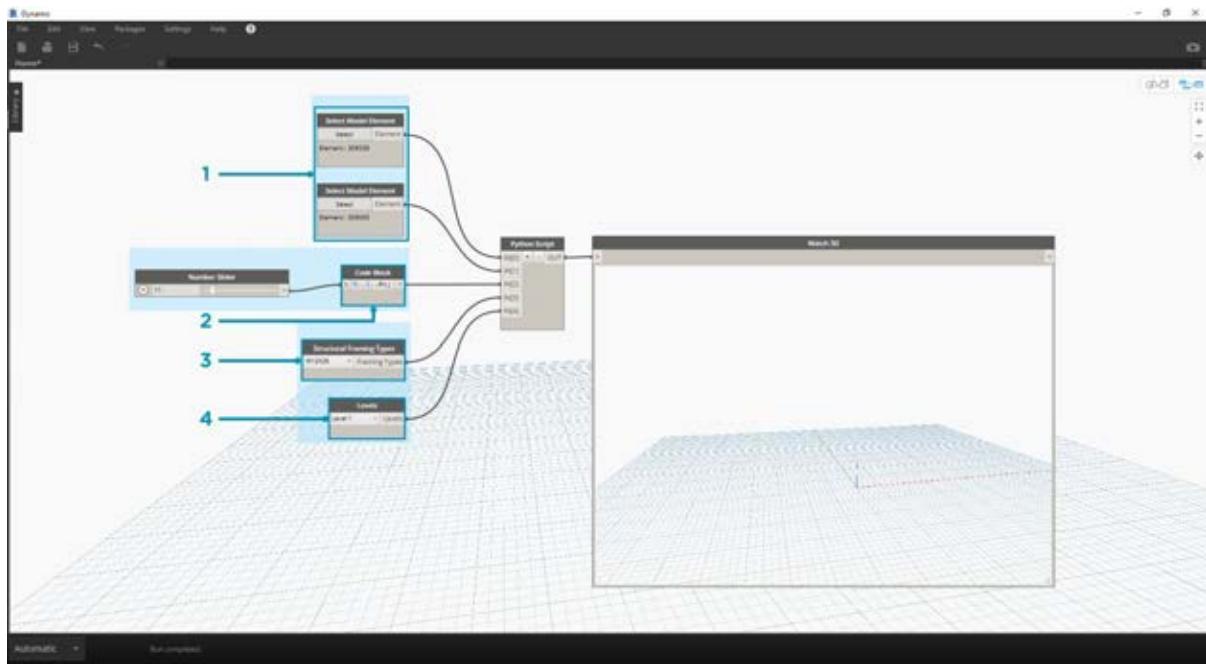
### Übungslektion 03

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Revit-StructuralFraming.zip](#)

Diese Übung ist relativ einfach, macht jedoch die Verbindung von Daten und Geometrie zwischen Revit und Dynamo – in beiden Richtungen – deutlich. Öffnen Sie zuerst Revit-StructuralFraming.rvt. Laden Sie anschließend Dynamo und öffnen Sie die Datei Revit-StructuralFraming.dyn.



Diese Datei ist denkbar einfach. Sie umfasst zwei auf Ebene 1 und Ebene 2 gezeichnete Referenzkurven. Diese Kurven sollen in Dynamo übernommen werden, wobei eine Direktverknüpfung bestehen bleibt.



Diese Datei enthält eine Gruppe von Blöcken, die mit den fünf Eingaben eines Python-Blocks verbunden sind.

1. **Select Model Element-Blöcke:** Klicken Sie jeweils auf die Schaltfläche Auswählen und wählen Sie die zugehörige Kurv in Revit aus.
2. **Code Block:** Geben Sie die Syntax "0..1..#x;" ein und verbinden Sie einen Integer Slider mit Werten zwischen 0 und 20 mit der x-Eingabe. Dadurch wird die Anzahl der Träger gesteuert, die zwischen den beiden Kurven gezeichnet werden sollen.
3. **Structural Framing Types:** Wählen Sie hier den vorgegebenen W12x26-Träger aus der Dropdown-Liste.
4. **Levels:** Wählen Sie die "Level 1".

The screenshot shows a Python script editor window titled "Python Script". The code is written in Python and performs the following steps:

- Imports necessary modules: `clr`, `Dynamo Geometry`, `ProtoGeometry`, `Autodesk.DesignScript.Geometry`, `RevitNodes`, `Revit`, and `System`.
- Queries Revit elements and converts them to Dynamo Curves.
- Defines input parameters: `framingType` (IN[3]), `designLevel` (IN[4]).
- Creates a list for output: `OUT = []`.
- Loops through values IN[2]:
  - Defines Dynamo Points on each curve: `ptA=Curve.PointAtParameter(crvA,val)` and `ptB=Curve.PointAtParameter(crvB,val)`.
  - Creates Dynamo line: `beamCrv=Line.ByStartPointEndPoint(ptA,ptB)`.
  - Creates Revit Element from Dynamo Curves: `beam = StructuralFraming.BeamByCurve(beamCrv,designLevel,framingType)`.
  - Converts Revit Element into list of Dynamo Surfaces: `OUT.append(bean.Faces)`.

At the bottom of the window are buttons for "Run", "Save Changes", and "Revert".

Dieser Code in Python ist etwas komplexer, aus den darin enthaltenen Kommentaren geht jedoch hervor, wie der Prozess abläuft.

```
import clr
#import Dynamo Geometry
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *
# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit
# Import Revit elements
from Revit.Elements import *
import System

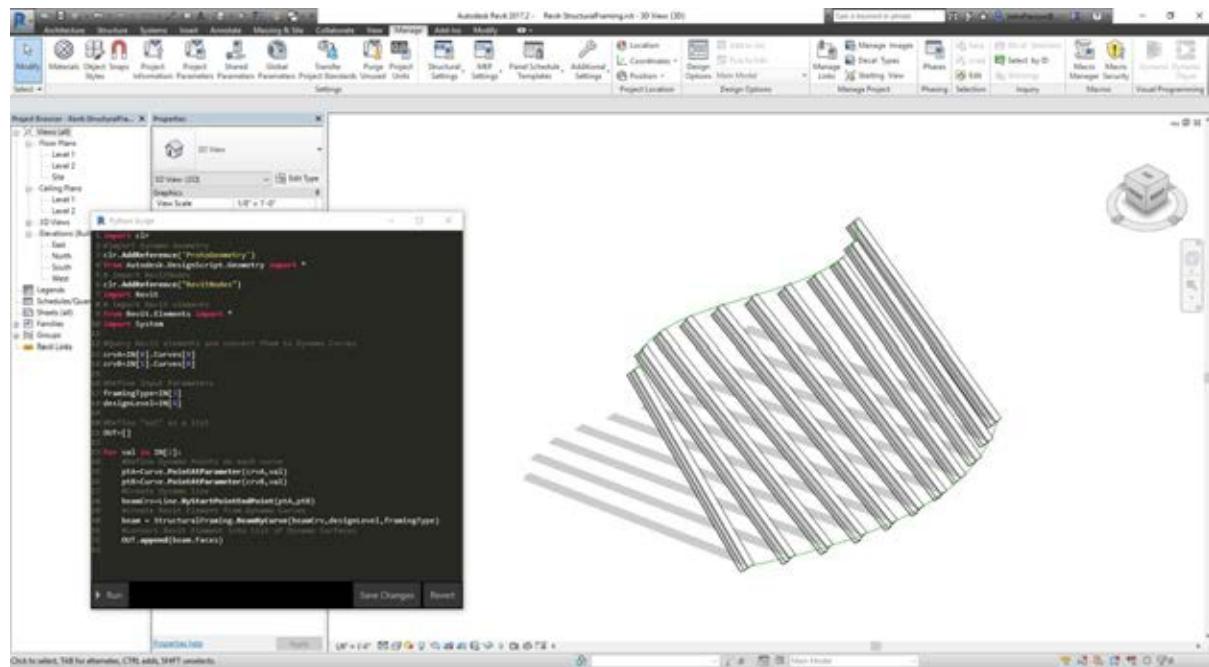
#Query Revit elements and convert them to Dynamo Curves
crvA=IN[0].Curves[0]
crvB=IN[1].Curves[0]

#define input Parameters
framingType=IN[3]
designLevel=IN[4]

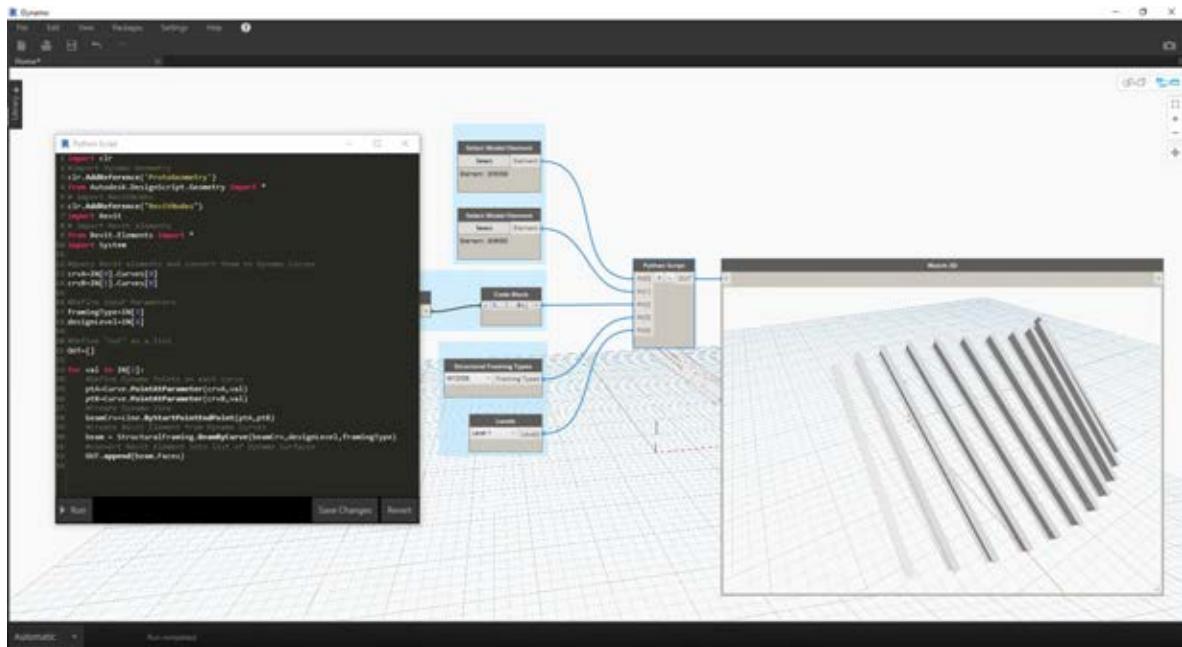
#define "out" as a list
OUT=[]

for val in IN[2]:
    #Define Dynamo Points on each curve
```

```
pta=Curve.PointAtParameter(crvA,val)
ptB=Curve.PointAtParameter(crvB,val)
#Create Dynamo line
beamCrv=Line.ByStartPointEndPoint(ptA,ptB)
#create Revit Element from Dynamo Curves
beam = StructuralFraming.BeamByCurve(beamCrv,designLevel,framingType)
#convert Revit Element into list of Dynamo Surfaces
OUT.append(beam.Faces)
```



In Revit wird eine Reihe von Trägern zwischen den beiden Kurven als Tragwerkselemente angezeigt. Anmerkung: Dies ist kein realistisches Beispiel. Die Tragwerkselemente sind nur als Beispiele für aus Dynamo erstellte native Revit-Exemplare.

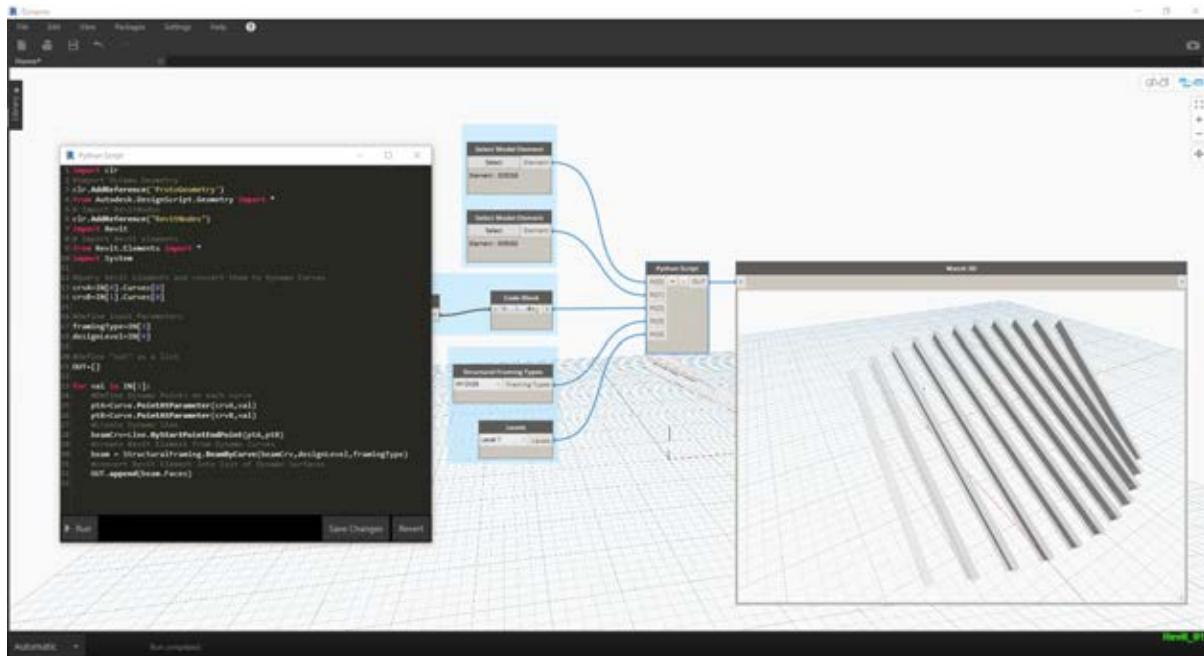


In Dynamo werden die Ergebnisse ebenfalls angezeigt. Die Träger im Watch3D-Block verweisen auf die aus den Revit-Elementen abgefragte Geometrie.

Dabei werden Daten aus der Revit-Umgebung für die Dynamo-Umgebung konvertiert. Zusammenfassung: Der Prozess läuft wie folgt ab:

1. Wählen Sie das Revit-Element aus.
2. Konvertieren Sie das Revit-Element in eine Dynamo-Kurve.
3. Unterteilen Sie die Dynamo-Kurve in eine Folge von Dynamo-Punkten.
4. Erstellen Sie Dynamo-Linien mithilfe der Dynamo-Punkte zwischen den beiden Kurven.
5. Erstellen Sie Revit-Träger durch Referenzieren der Dynamo-Linien.
6. Geben Sie Dynamo-Oberflächen durch Abfragen der Geometrie der Revit-Träger aus.

Dies mag etwas umständlich erscheinen. Das Skript erleichtert den Vorgang jedoch erheblich: Sie müssen jetzt lediglich die Kurve in Revit bearbeiten und den Solver erneut ausführen. (Es ist möglich, dass Sie dabei die bisherigen Träger löschen müssen.) *Dies liegt daran, dass wir die Träger in Python platzieren und dadurch die Zuordnung der OOTB-Blöcke auflösen.*



Werden die Referenzkurven in Revit aktualisiert, erhalten Sie eine neue Reihe von Trägern.

# Python-Vorlagen

## Python-Vorlagen

In Dynamo 2.0 haben Sie die Möglichkeit eine Standardvorlage (\*.py-Erweiterung) festzulegen, die verwendet wird, wenn Sie das Python-Fenster zum ersten Mal öffnen. Entwickler haben sich diese Funktion schon lange gewünscht, da sie die Verwendung von Python innerhalb von Dynamo beschleunigt. Die Verwendung einer Vorlage ermöglicht es uns, vorgabemäßige Imports jederzeit startbereit zu haben, wenn wir ein benutzerdefiniertes Python-Skript entwickeln möchten.

Die Vorlage befindet sich im Ordner APPDATA Ihrer Dynamo-Installation.

Diese ist in der Regel: (%appdata%/Dynamo/Core/{version}/).

Share View				
C:\Users\ USERNAME \AppData\Roaming\Dynamo\Dynamo Core\2.0				
	Name	Date modified	Type	Size
ss	definitions	4/17/2018 7:48 AM	File folder	
d	Logs	5/1/2018 9:32 AM	File folder	
d	packages	4/17/2018 9:23 AM	File folder	
r	DynamoSettings.xml	5/1/2018 9:32 AM	XML Document	2 KB
r	PythonTemplate.py	5/8/2018 9:15 AM	Python File	1 KB

## Einrichten der Vorlage

Um diese Funktion nutzen zu können, müssen wir unserer Datei `DynamoSettings.xml` die folgende Zeile hinzufügen.  
*(in Editor bearbeiten)*

```
29 <CustomPackageFolders>
30   <string>C:\Users\ USERNAME \AppData\Roaming\Dynamo\Core\2.0</string>
31 </CustomPackageFolders>
32 <PackageDirectoriesToUninstall />
33 <PythonTemplateFilePath />
34 <BackupInterval>60000</BackupInterval>
35 <BackupFilesCount>1</BackupFilesCount>
36 <PackageDownloadTouAccepted>false</PackageDownloadTouAccepted>
```

Ersetzen Sie alle Vorkommen von durch das Folgende:

```
<PythonTemplateFilePath>
C:\Users\CURRENTUSER\AppData\Roaming\Dynamo\Core\2.0\PythonTemplate.py
</PythonTemplateFilePath>
```

*Hinweis: Ersetzen Sie CURRENTUSER mit Ihrem Benutzernamen.*

Als Nächstes müssen wir eine Vorlage mit den Funktionen erstellen, die wir integrieren möchten. In diesem Fall können wir die Revit-bezogenen Importe und einige andere typische Elemente einbetten, die wir bei der Arbeit mit Revit verwenden.

Sie können mit einem leeren Editor-Dokument beginnen und den folgenden Code einfügen:

```
import clr

clr.AddReference('RevitAPI')
from Autodesk.Revit.DB import *
from Autodesk.Revit.DB.Structure import *

clr.AddReference('RevitAPIUI')
```

```

from Autodesk.Revit.UI import *
clr.AddReference('System')
from System.Collections.Generic import List

clr.AddReference('RevitNodes')
import Revit
clr.ImportExtensions(Revit.GeometryConversion)
clr.ImportExtensions(Revit.Elements)

clr.AddReference('RevitServices')
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

doc = DocumentManager.Instance.CurrentDBDocument
uidoc=DocumentManager.Instance.CurrentUIApplication.ActiveUIDocument

#Preparing input from dynamo to revit
element = UnwrapElement(IN[0])

#Do some action in a Transaction
TransactionManager.Instance.EnsureInTransaction(doc)

TransactionManager.Instance.TransactionTaskDone()

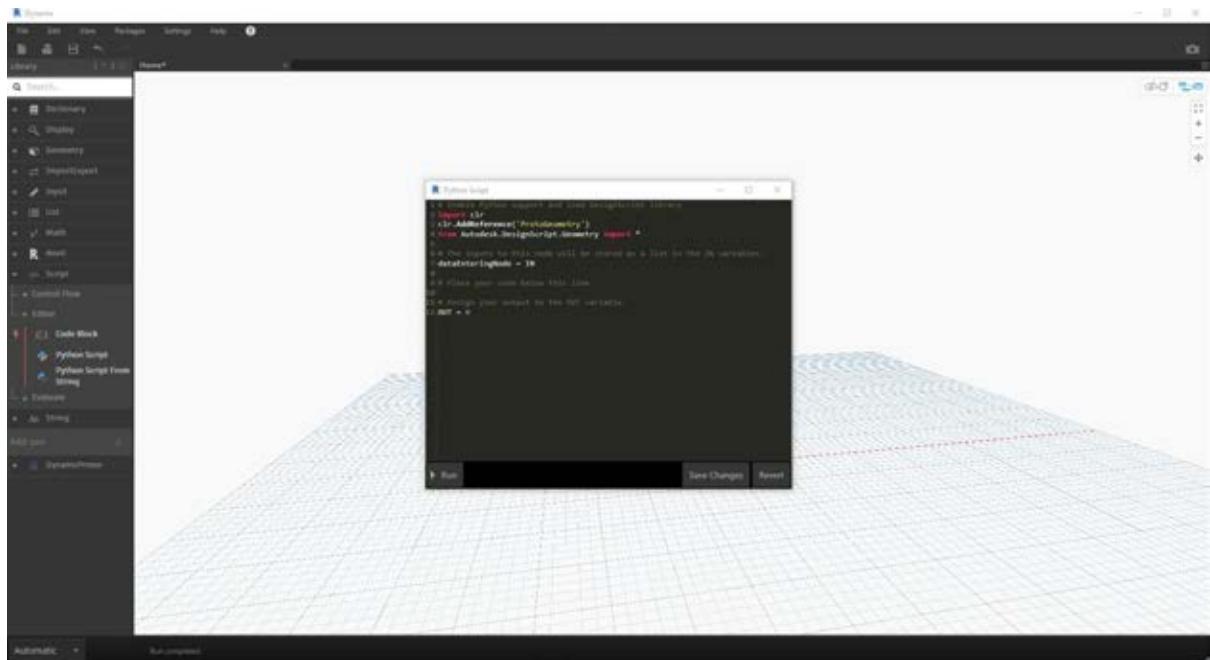
OUT = element

```

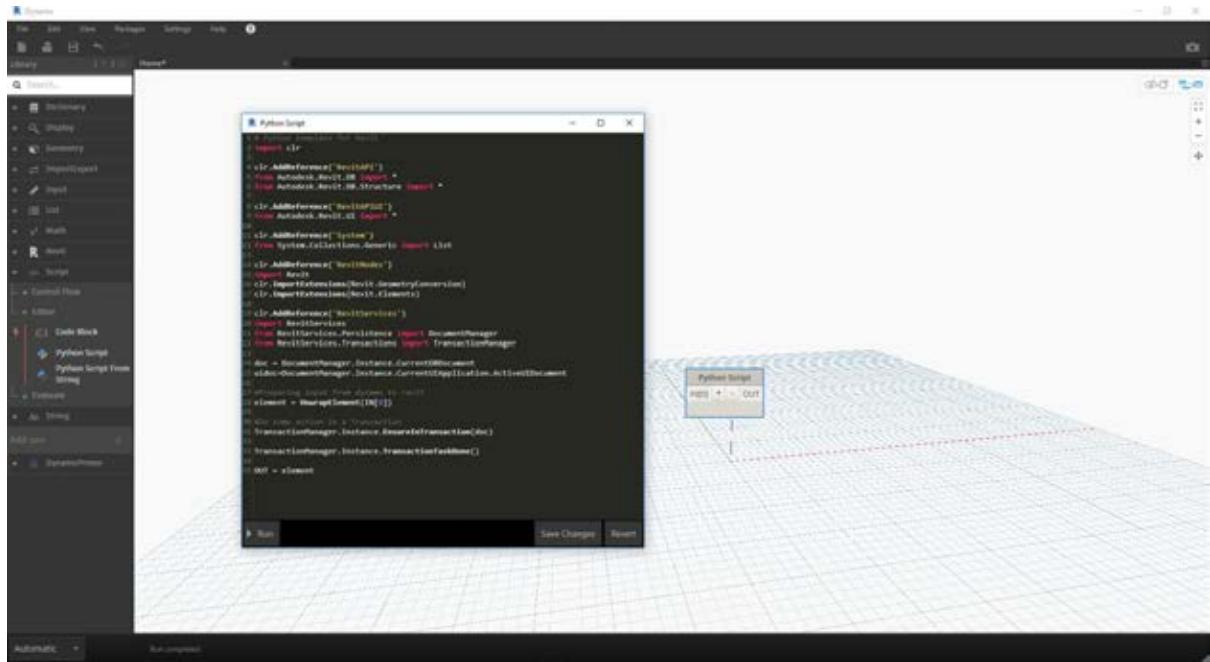
Anschließend speichern Sie diese Datei als **PythonTemplate.py** am Speicherort APPDATA.

### **Das Verhalten des Python-Skripts danach**

Nach dem Erstellen der Python-Vorlage sucht Dynamo jedes Mal danach, wenn Sie einen Python-Block einfügen. Wenn sie nicht gefunden wird, wird das vorgabemäßige Python-Fenster angezeigt.



Wenn die Python-Vorlage gefunden wird (beispielsweise für Revit), werden alle Ihre vorgegebenen Elemente angezeigt, die Sie integriert haben.

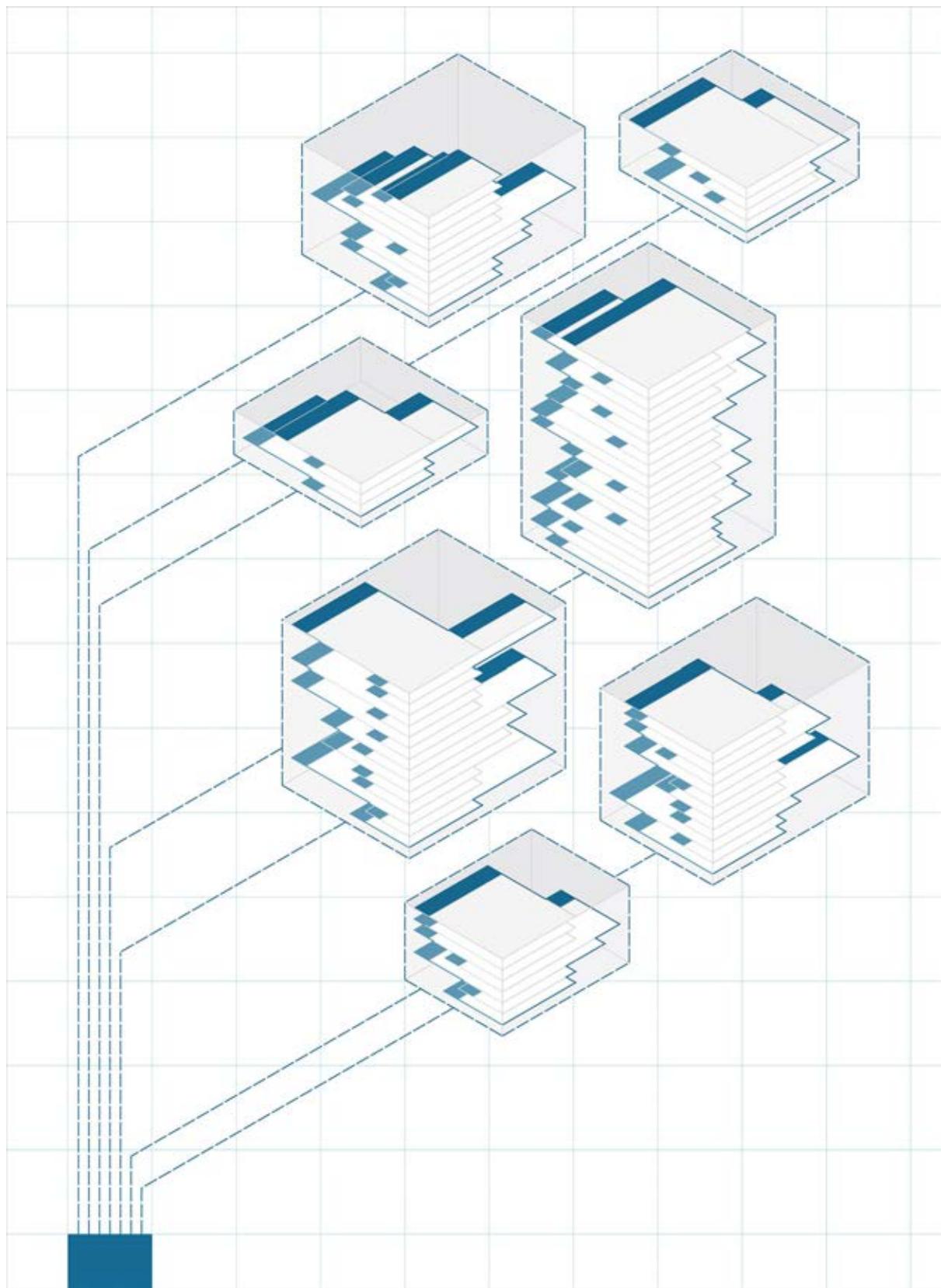


Weitere Informationen zu dieser großartigen Ergänzung (von Radu Gidei) finden Sie hier.  
<https://github.com/DynamoDS/Dynamo/pull/8122>

## **Pakete**

### **Pakete**

Nachdem Sie einige benutzerdefinierte Blöcke erstellt haben, beginnen Sie im nächsten Schritt damit, sie zu ordnen und in Form von Paketen zu veröffentlichen: eine einfache Methode, Ihre Blöcke zu speichern und in der Dynamo-Community bereitzustellen.



# Pakete

## Pakete

Ein Paket ist, kurz gesagt, eine Sammlung benutzerdefinierter Blöcke. Der Dynamo Package Manager ist ein Community-Portal, aus dem Sie beliebige Pakete herunterladen können, die online veröffentlicht wurden. Diese Toolsets werden von externen Anbietern entwickelt und stellen Erweiterungen der Hauptfunktionen von Dynamo dar. Sie stehen für alle Benutzer zur Verfügung und können durch einfaches Klicken auf eine Schaltfläche heruntergeladen werden.

The screenshot shows the homepage of the Dynamo Package Manager. At the top, there are navigation links for 'Browse' and 'Search'. Below that is the title 'Dynamo Package Manager' with the subtitle 'Share and discover workflows for Dynamo visual programming'. Underneath, there are three large numerical statistics: 106248 (ITEMS), 614 (PACKAGES), and 143 (AUTHORS). The main content area is divided into two sections: 'Packages' and 'Authors'. The 'Packages' section contains four tables: 'Newest', 'Most Recently Updated', 'Most Installed', and 'Most Depended Upon'. The 'Authors' section contains four tables: 'Most Voted For', 'Most Recently Active', 'Most Prolific', and 'Most Installed'. Each table lists several packages or authors with their names, descriptions, and metrics like votes or installs.

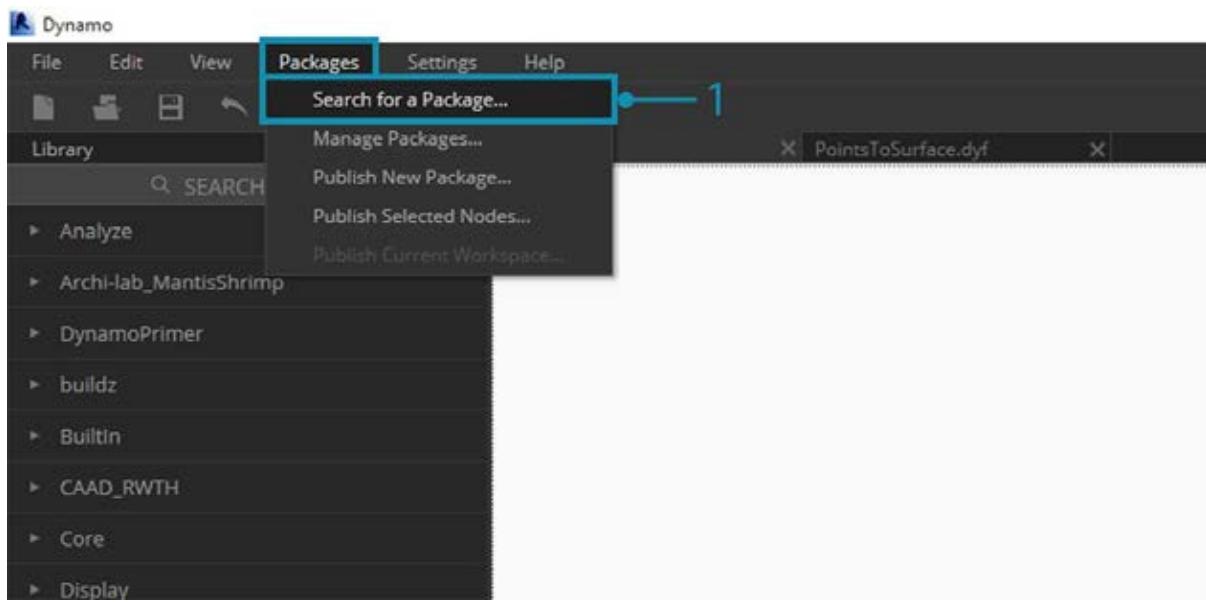
Category	Item 1	Item 2	Item 3	Item 4
Newest	LOD Greening (1 day ago)	WoollyBog (2 days ago)	WoollyBog (2 days ago)	WoollyBog (2 days ago)
Most Recently Updated	LOD Greening (2 days ago)	WoollyBog (2 days ago)	WoollyBog (2 days ago)	WoollyBog (2 days ago)
Most Installed	WoollyBog (2 days ago)	WoollyBog (2 days ago)	WoollyBog (2 days ago)	WoollyBog (2 days ago)
Most Depended Upon	(1 day ago)	(1 day ago)	(1 day ago)	(1 day ago)

Category	Author 1	Author 2	Author 3	Author 4
Most Voted For	john_dg (1 day ago)			
Most Recently Active	john_dg (2 days ago)			
Most Prolific	john_dg (2 days ago)			
Most Installed	john_dg (2 days ago)			

Community-Engagement wie dieses ist die Grundlage des Erfolgs von Open Source-Projekten wie Dynamo. Dank der Arbeit dieser hochmotivierten externen Entwickler kann Dynamo für Arbeitsabläufe in zahlreichen verschiedenen Branchen genutzt werden. Aus diesem Grund hat das Dynamo-Team sich geschlossen bemüht, die Entwicklung und Veröffentlichung von Paketen zu vereinheitlichen. (Dies wird in den folgenden Abschnitten detaillierter beschrieben.)

## Installation eines Pakets

Die einfachste Methode zum Installieren eines Pakets ist die Verwendung des Werkzeugkastens Pakete in der Dynamo-Benutzeroberfläche. Diese Methode wird im Folgenden beschrieben. In diesem Beispiel installieren Sie ein häufig verwendetes Paket zum Erstellen viereckiger Felder in einem Raster.



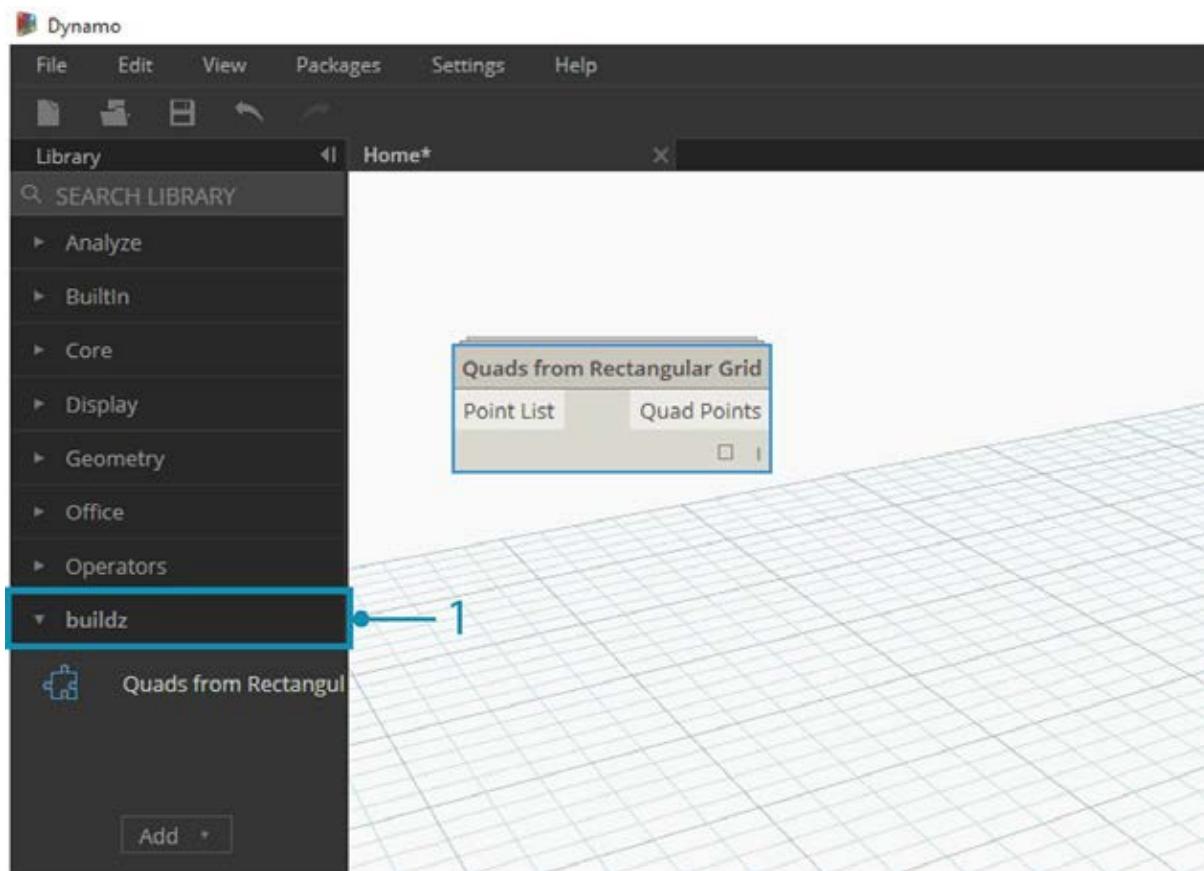
1. Wechseln Sie in Dynamo zu *Pakete > Suchen nach Paket*.

A screenshot of the 'Online Package Search' window. The search bar at the top contains the text 'quads from rectangular grid'. To the right of the search bar is a 'Sort by' button. Below the search bar, there are two search results:

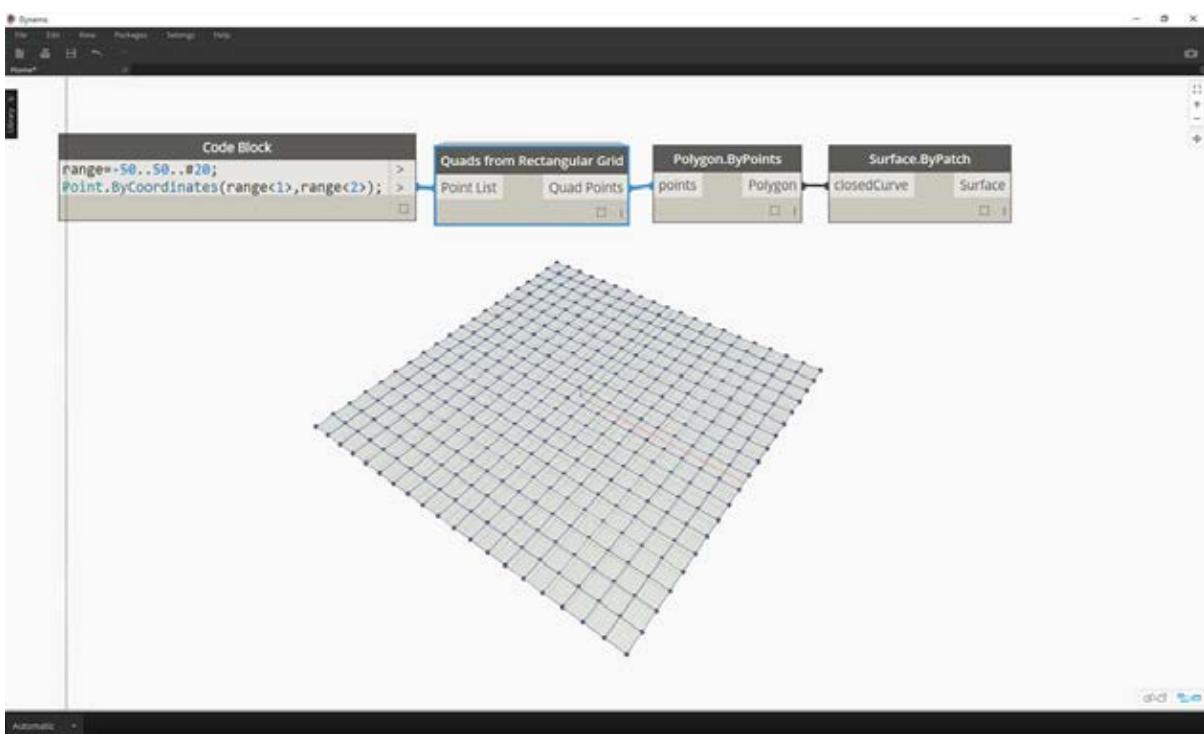
- Quads from Rectangular Grid** by kronz (version 0.1.0, 1843 downloads, 27 Dec 2014). The description states: 'Given a grid of points (xyz or uv) in lists of rows/columns, create a list of...'. This result has a download icon with a blue dot and the number '1' next to it.
- UV Quads on Surface** by colin.mccrone (version 0.0.1, 929 downloads, 18 May 2014). The description states: 'Find groups of 4 points describing cells in a rectangular grid on a surface....'

Suchen Sie mithilfe der Suchleiste nach "quads from rectangular grid". Nach kurzer Zeit sollten alle Pakete, die dieser Suchabfrage entsprechen, angezeigt werden. Sie müssen in diesem Fall das erste Paket mit passendem Namen auswählen.

1. Klicken Sie auf den Download-Pfeil links neben dem Paketnamen. Das Paket wird installiert. Fertig!



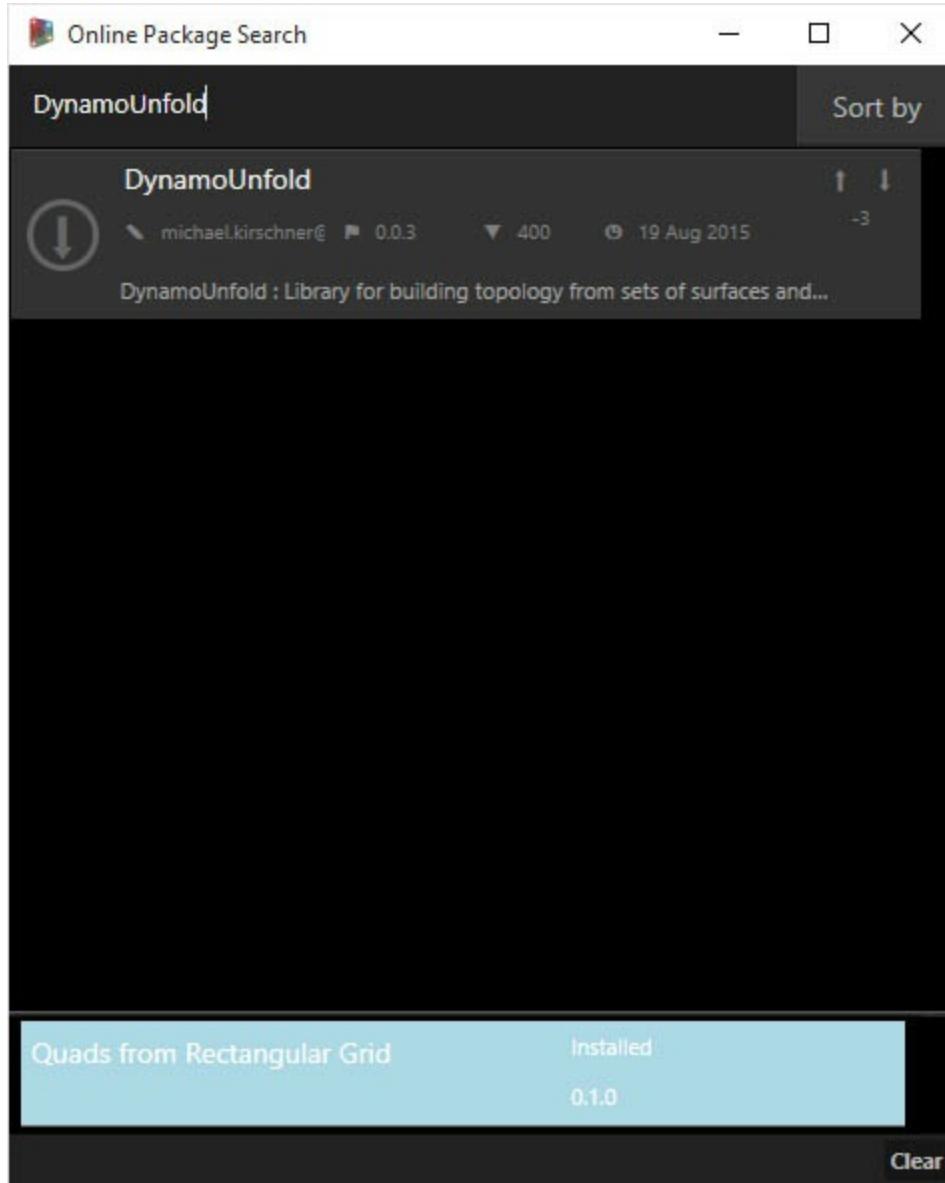
1. In der Dynamo-Bibliothek wird jetzt eine weitere Gruppe namens *buildz* angezeigt. Dieser Name bezieht sich auf den [Entwickler](#) des Pakets und der benutzerdefinierte Block wird in dieser Gruppe abgelegt. Sie können ihn sofort verwenden.



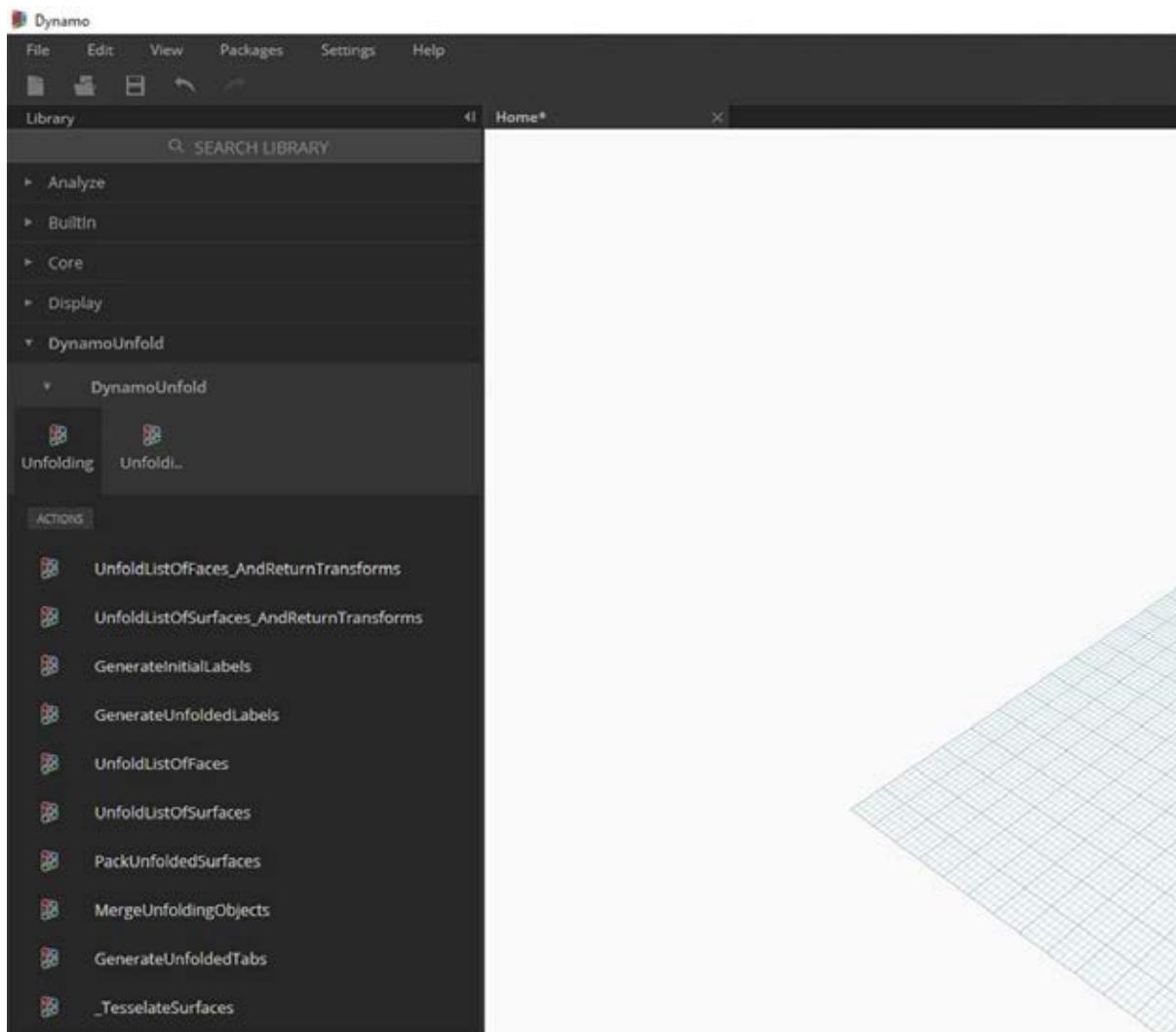
Hier wurde mithilfe einer kurzen Codeblock-Operation zum Definieren eines Rechteckrasters eine Liste rechteckiger Felder erstellt.

## Ordner in Paketen

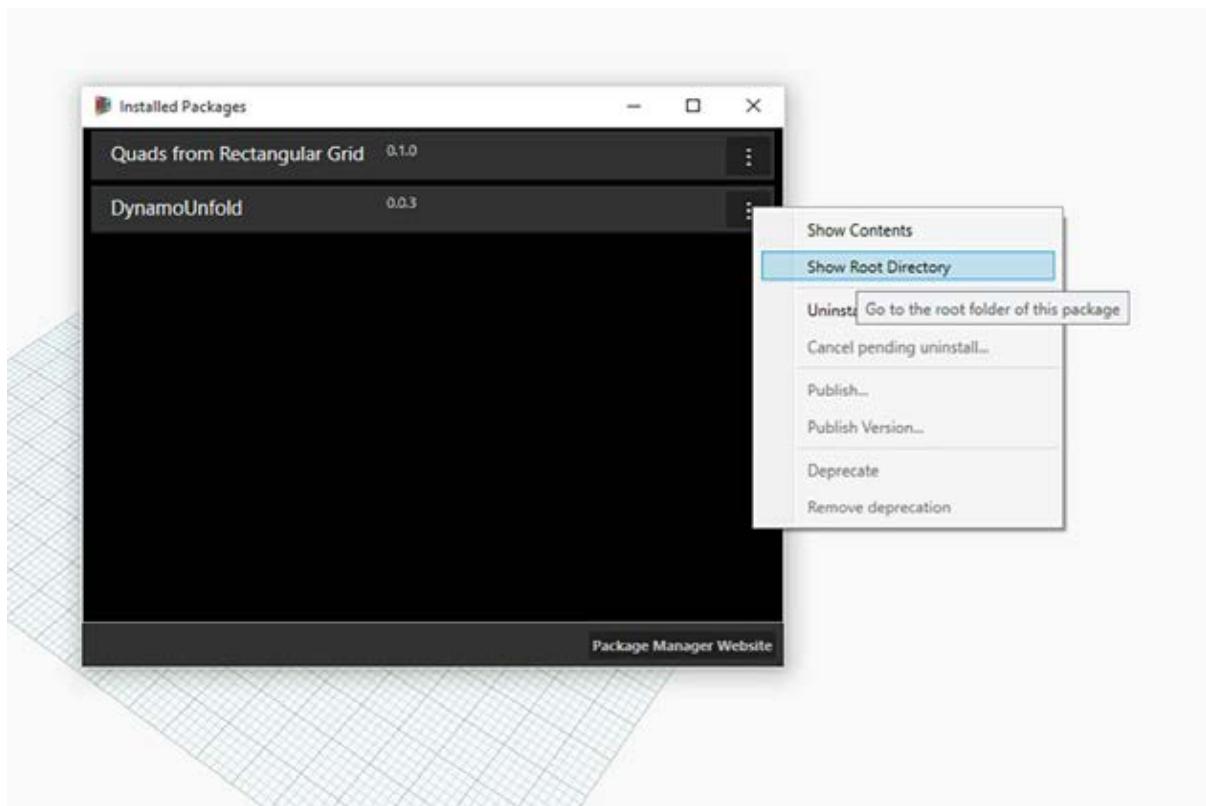
Das Paket im vorigen Beispiel enthält nur einen benutzerdefinierten Block. Pakete, die mehrere benutzerdefinierte Blöcke und die unterstützenden Datendateien enthalten, werden jedoch auf dieselbe Weise heruntergeladen. Dies wird hier an einem umfassenderen Paket demonstriert: Dynamo Unfold.



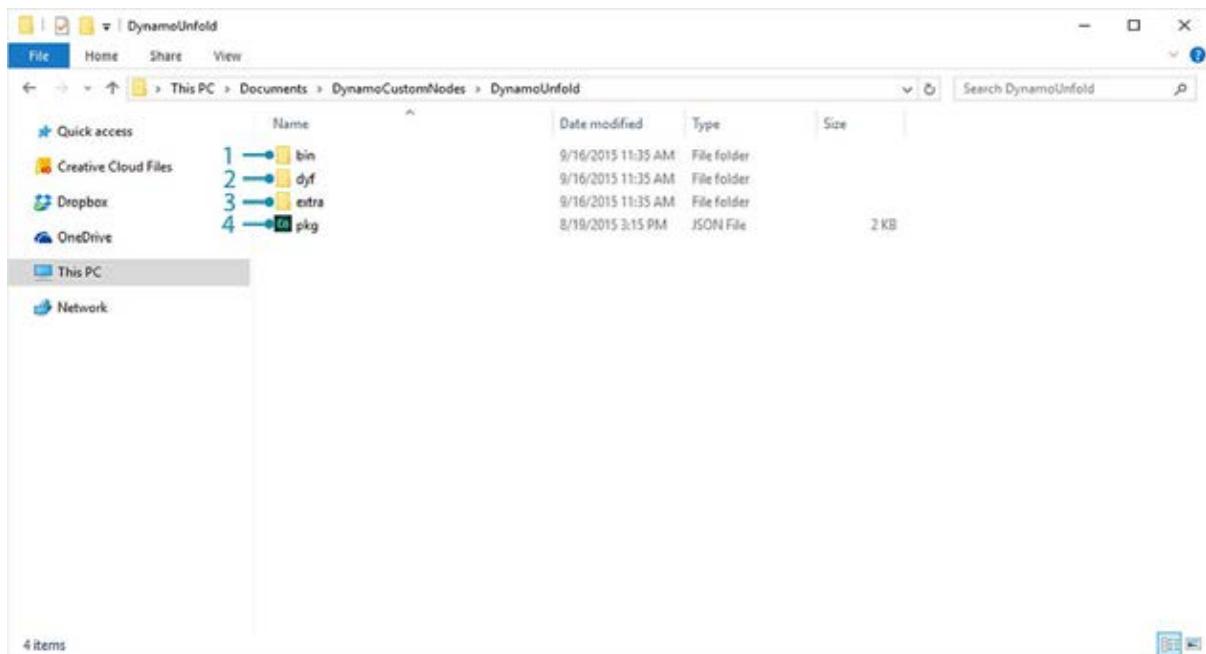
Beginnen Sie wie im Beispiel oben, indem Sie *Pakete > Suchen nach Paket* wählen. Suchen Sie in diesem Fall nach *DynamoUnfold* – in einem Wort geschrieben und unter Berücksichtigung der Groß- und Kleinschreibung. Wenn die Pakete angezeigt werden, laden Sie sie durch Klicken auf den Pfeil links neben dem Paketnamen herunter. Damit wird *Dynamo Unfold* in der *Dynamo*-Bibliothek installiert.



Die Dynamo-Bibliothek enthält jetzt die Gruppe *DynamoUnfold* mit mehreren Kategorien und benutzerdefinierten Blöcken.

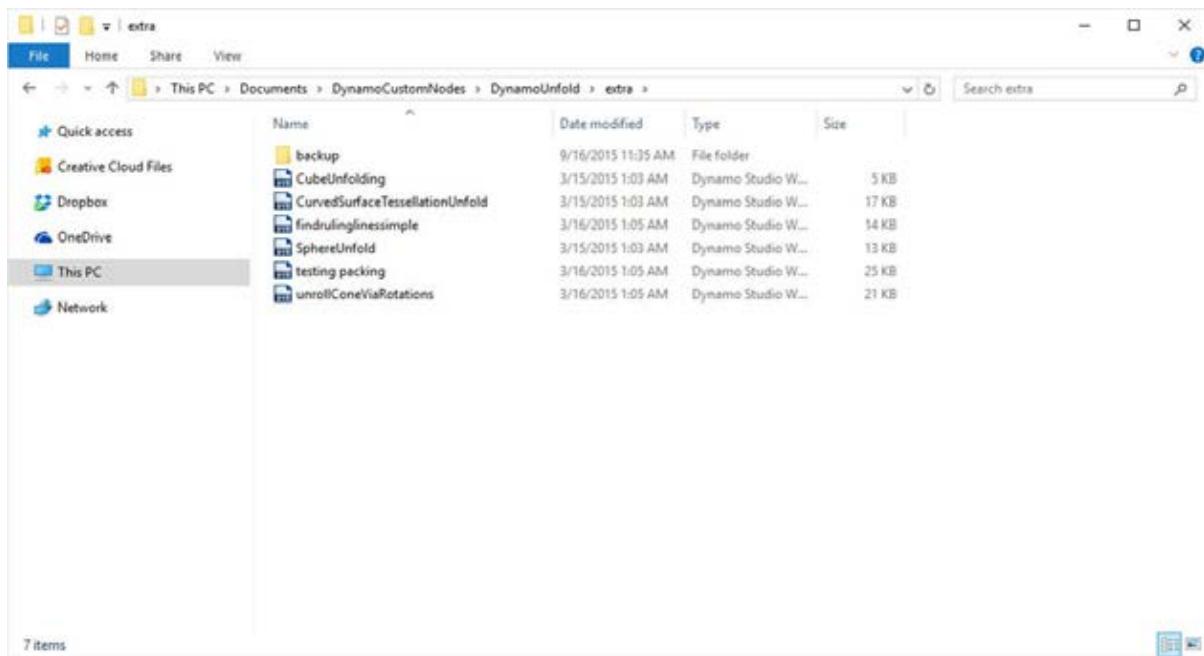


Als Nächstes betrachten Sie die Dateistruktur des Pakets genauer. Wählen Sie *Pakete > Pakete verwalten* in Dynamo. Das oben gezeigte Fenster mit den beiden eben installierten Bibliotheken wird angezeigt. Klicken Sie auf die Schaltfläche rechts neben *DynamoUnfold* und wählen Sie *Stammverzeichnis anzeigen*.

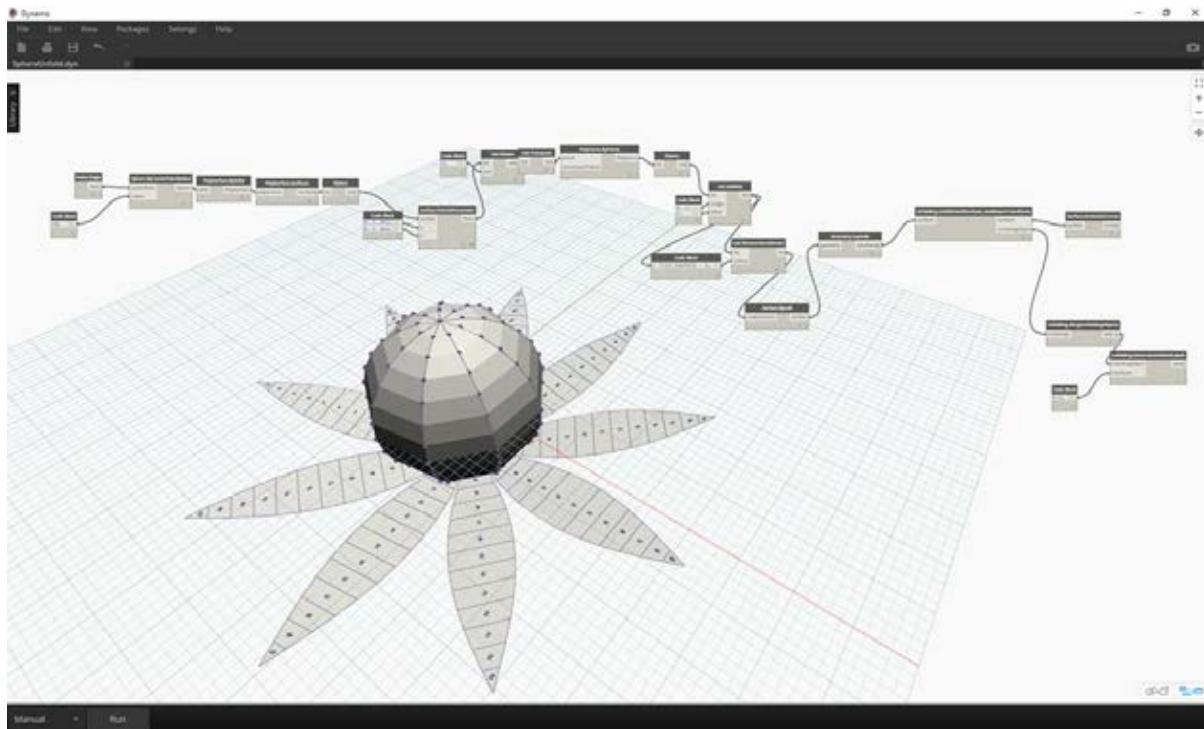


Dadurch gelangen Sie zum Stammverzeichnis des Pakets. Hier sind drei Ordner und eine Datei vorhanden.

1. Im Ordner *bin* werden DLL-Dateien gespeichert. Dieses Dynamo-Paket wurde mit Zero-Touch entwickelt, d.h., die benutzerdefinierten Blöcke wurden in diesem Ordner abgelegt.
2. Im Ordner *dyf* befinden sich die benutzerdefinierten Blöcke. Da dieses Paket nicht mithilfe benutzerdefinierter Dynamo-Blöcke entwickelt wurde, ist dieser Ordner in diesem Paket leer.
3. Der Ordner *extra* enthält alle zusätzlichen Dateien, darunter Ihre Beispieldateien.
4. Die Datei *pkg* ist eine einfache Textdatei, die die Einstellungen des Pakets definiert. Sie können diese Datei für den Augenblick ignorieren.



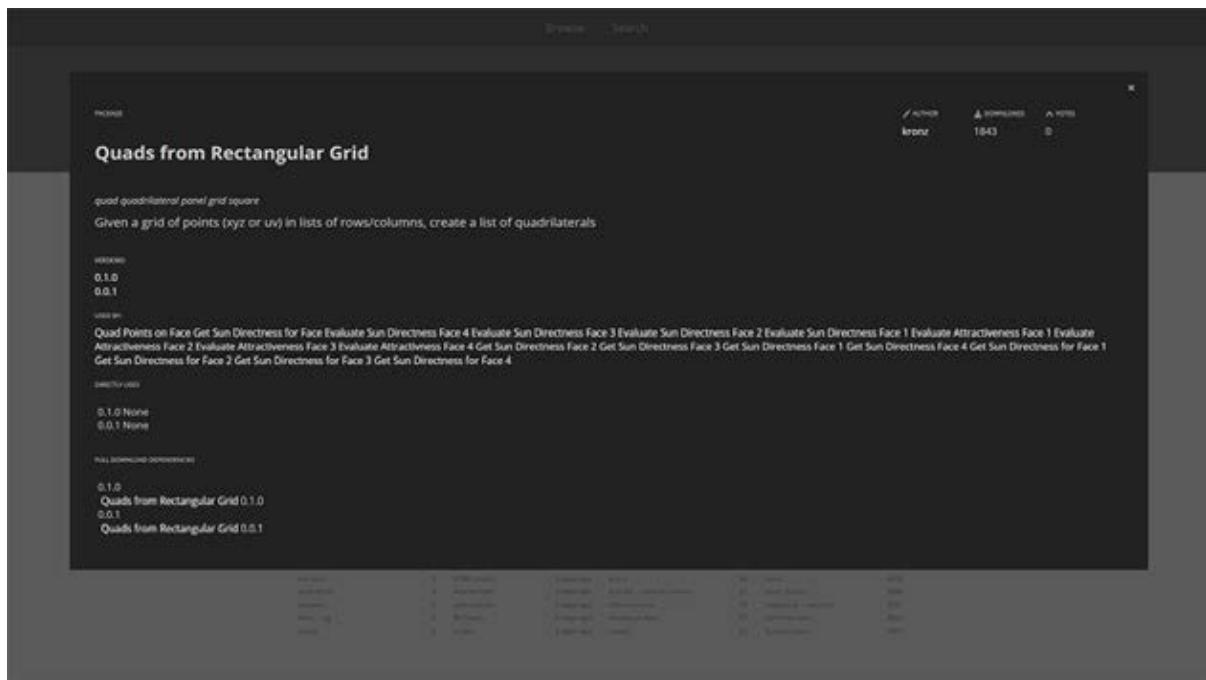
Wenn Sie den Ordner *extra* öffnen, sehen Sie eine Reihe von Beispieldateien, die mit der Installation heruntergeladen wurden. Beispieldateien stehen nicht in allen Paketen zur Verfügung. Falls sie jedoch vorhanden sind, finden Sie sie in diesem Ordner. Öffnen Sie *SphereUnfold*.



Nachdem Sie die Datei geöffnet und im Solver auf *Ausführen* geklickt haben, erhalten Sie das Netz einer Kugel! Beispieldateien wie diese erleichtern den Einstieg in die Arbeit mit einem neuen Dynamo-Paket.

## Dynamo Package Manager

Sie können auch online im [Dynamo Package Manager](#) nach Dynamo-Paketen suchen. Dies ist ein sehr effizientes Verfahren für die Suche nach Paketen, da diese im Repository nach Anzahl der Downloads und Beliebtheit sortiert werden. Sie finden auf diese Weise auch mühelos Informationen zu kürzlich erfolgten Aktualisierungen der Pakete: Manche Dynamo-Pakete unterliegen einer Versionskontrolle und es bestehen Abhängigkeiten zu Dynamo-Builds.



Durch Klicken auf *Quads from Rectangular Grid* im Dynamo Package Manager zeigen Sie die Beschreibungen, Versionen, den Entwickler sowie eventuelle Abhängigkeiten an.

Sie können die Paketdateien auch über den Dynamo Package Manager herunterladen, der direkte Download in Dynamo ist jedoch ein nahtloser Ablauf.

### Wo werden die Dateien lokal gespeichert?

Wenn Sie Dateien über Dynamo Package Manager herunterladen oder prüfen möchten, wo die Paketdateien abgelegt werden, wählen Sie *Einstellungen > Pfade für Blöcke und Pakete verwalten*. Durch Klicken auf die Schaltfläche mit den drei Punkten neben dem Ordner können Sie den Stammordner kopieren und den Inhalt des Pakets in Ihrem Explorer-Fenster im Detail anzeigen. Pakete werden per Vorgabe unter einem Speicherort ähnlich dem folgenden installiert:  
*C:/Users/[Benutzername]/AppData/Roaming/Dynamo/[Dynamo-Version]*.

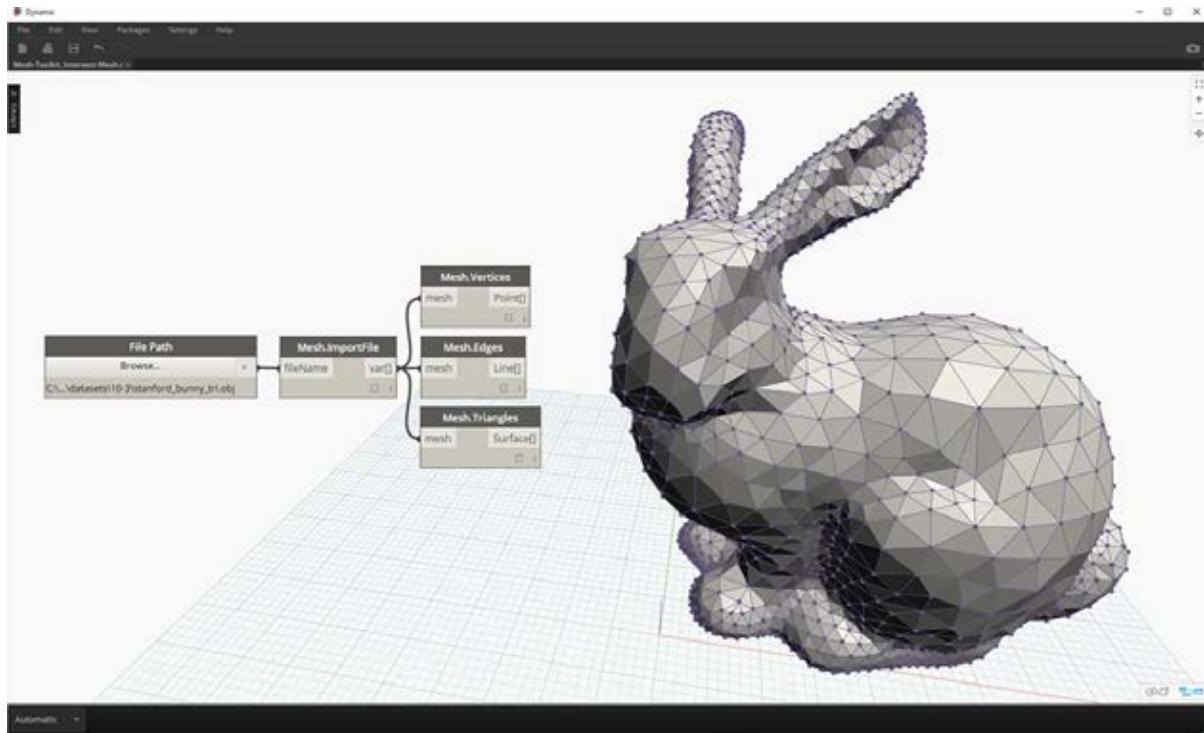
### Weitere Schritte mit Paketen

Die Dynamo-Community wächst laufend und entwickelt sich dabei weiter. Besuchen Sie Dynamo Package Manager von Zeit zu Zeit, um über neue inspirierenden Entwicklungen auf dem Laufenden zu bleiben. In den folgenden Abschnitten werden Pakete eingehender behandelt, wobei sowohl auf die Perspektive des Endbenutzers eingegangen als auch die Entwicklung eigener Dynamo-Pakete behandelt wird.

# Fallstudie zu Paketen: Mesh Toolkit

## Fallstudie zu Paketen: Mesh Toolkit

Das Dynamo Mesh Toolkit enthält Werkzeuge zum Importieren von Netzen aus externen Dateiformaten, zum Erstellen von Netzen aus Dynamo-Geometrieobjekten und zum manuellen Erstellen von Netzen aus ihren Scheitelpunkten und Indizes. In der Bibliothek stehen außerdem Werkzeuge zum Ändern oder Reparieren von Netzen sowie zum Extrahieren horizontaler Segmente zur Verwendung in der Fertigung zur Verfügung.

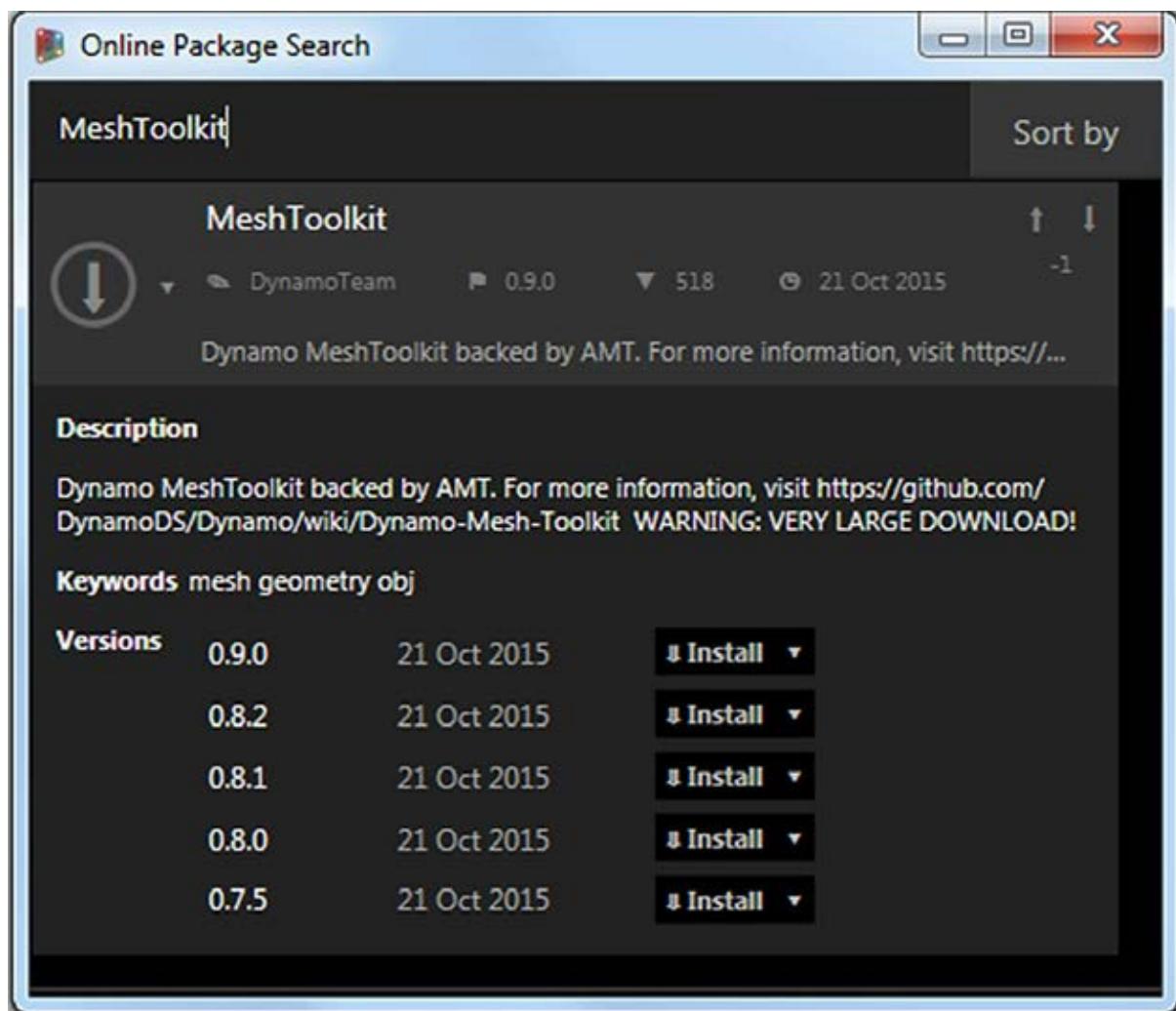


Das Dynamo Mesh Toolkit ist Bestandteil der von Autodesk durchgeführten laufenden Forschungsarbeiten zu Netzen und wird daher in den kommenden Jahren weiter ausgebaut. Sie können damit rechnen, dass häufig neue Methoden zu diesem Toolkit hinzukommen. Falls Sie Kommentare oder Vorschläge haben oder Fehler melden möchten, ist das Dynamo-Team jederzeit für Sie da.

### Netze und Volumenkörper im Vergleich

In der unten folgenden Übung werden einige grundlegende Operationen für Netze unter Verwendung von Mesh Toolkit gezeigt. In der Übung schneiden Sie ein Netz mit einer Folge von Ebenen. Wenn Sie hierfür Volumenkörper verwenden, kann dies sehr viel Rechenaufwand erfordern. Bei Netzen ist die "Auflösung" im Gegensatz zu Volumenkörpern festgelegt. Darüber hinaus werden Netze nicht mathematisch, sondern topologisch definiert und die Auflösung kann für die Zwecke der jeweiligen Aufgabe definiert werden. Weitere Informationen zum Verhältnis von Netzen und Volumenkörpern finden Sie im Kapitel [Geometrie für Computational Design](#) in diesem Leitfaden. Eine genauere Analyse von Mesh Toolkit finden Sie auf der [Dynamo-Wiki-Seite](#). Die folgende Übung zeigt die Verwendung dieses Pakets.

### Installieren von Mesh Toolkit

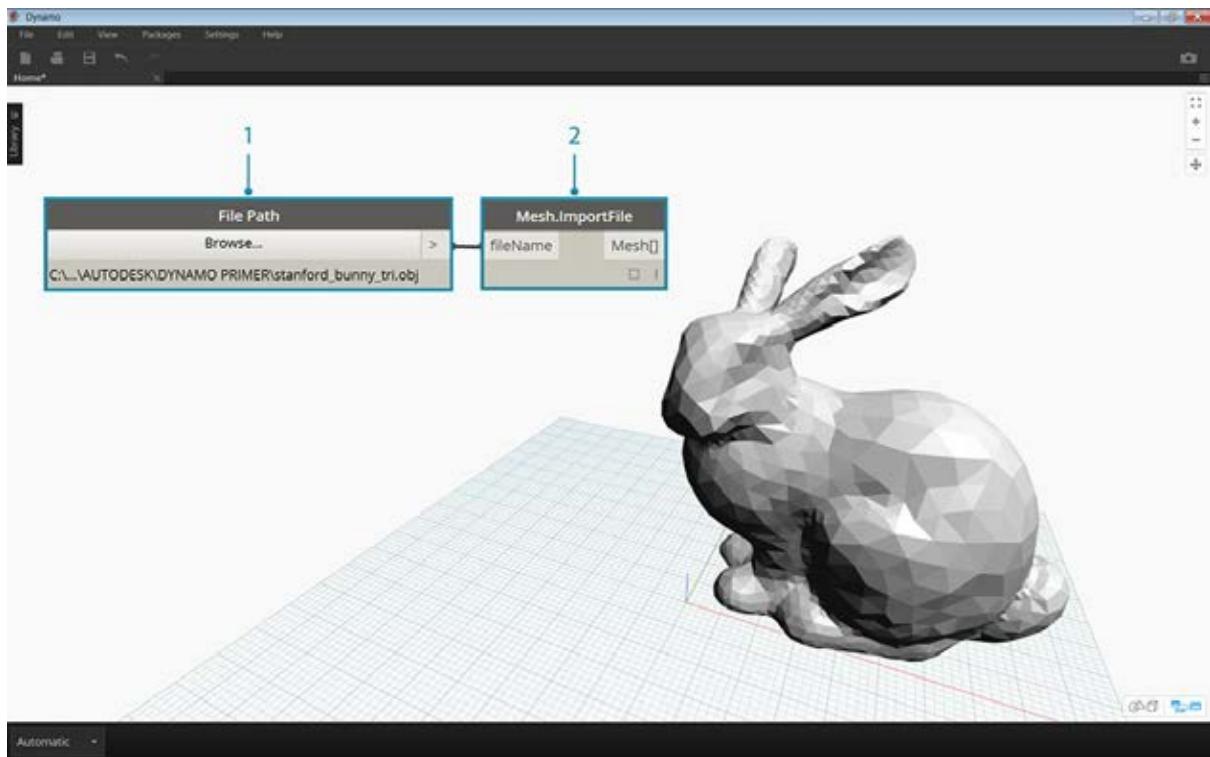


Wählen Sie in Dynamo in der Menüleiste oben *Pakete > Suchen nach Paket*. Geben Sie *MeshToolkit* in das Suchfeld ein (in einem Wort und unter Berücksichtigung der Groß- und Kleinschreibung). Klicken Sie auf den Download-Pfeil für das zu Ihrer Version von Dynamo passende Paket. Dieser einfache Schritt genügt.

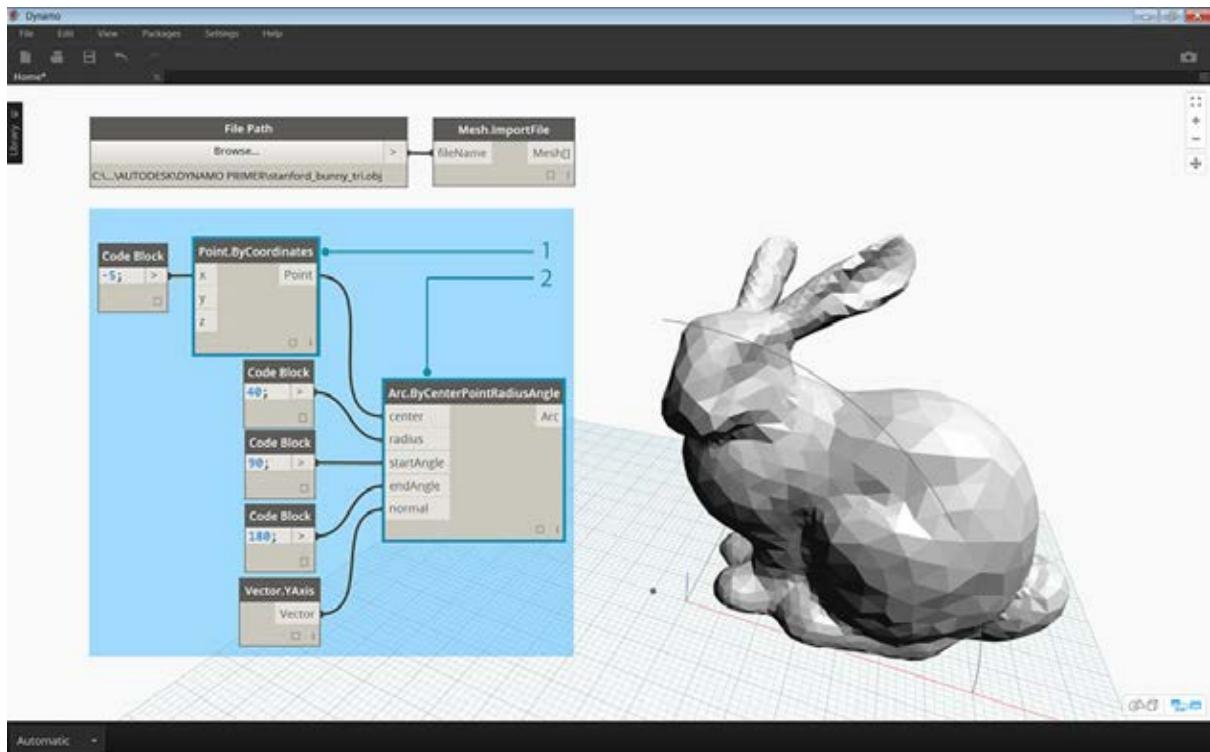
## Übungslektion

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [MeshToolkit.zip](#)

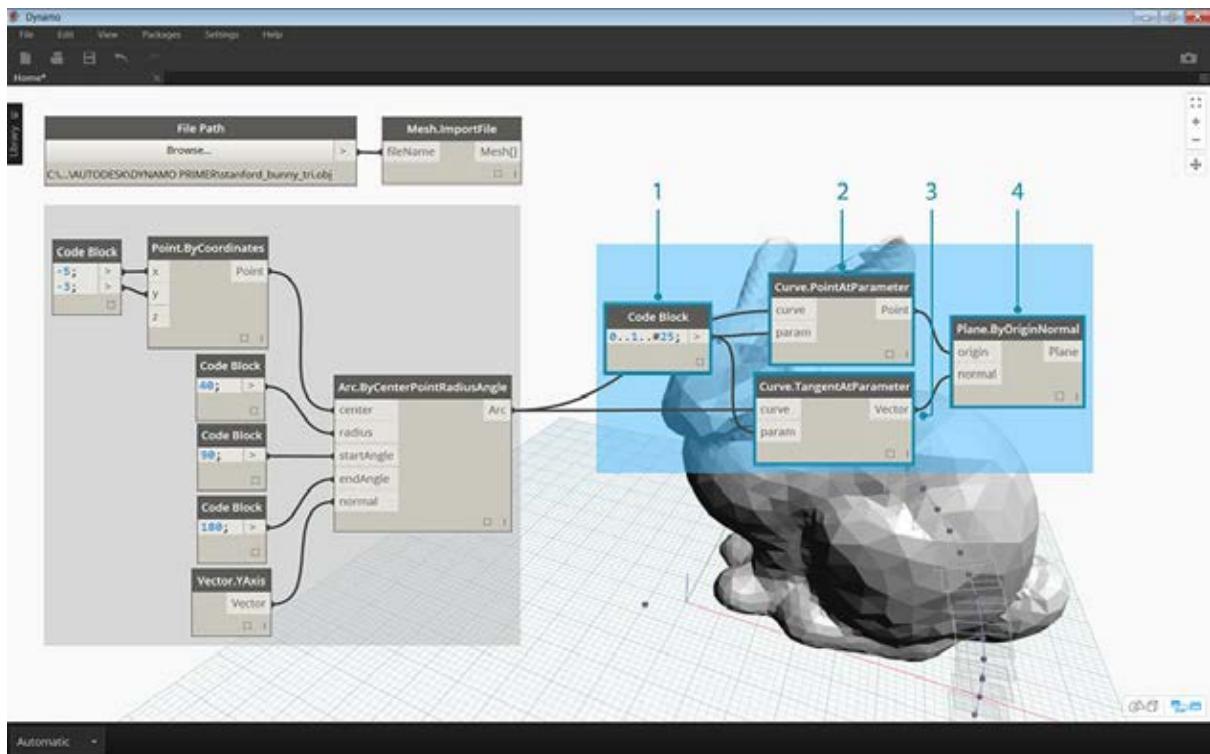
Öffnen Sie zunächst *Mesh-Toolkit\_Intersect-Mesh.dyn* in *Dynamo*. In diesem Beispiel verwenden Sie den Intersect-Block innerhalb von Mesh Toolkit. Sie importieren ein Netz und schneiden es mit einer Reihe eingegebener Ebenen, um Segmente zu erhalten. Davon ausgehend kann das Modell für die Fertigung auf einem Laser- oder Wasserstrahlwerkzeug oder einer CNC-Fräse vorbereitet werden.



1. **File Path:** Suchen Sie die zu importierende Netzdatei (*stanford\_bunny\_tri.obj*). Unterstützte Dateitypen sind .mix und .obj.
2. **Mesh.ImportFile:** Verbinden Sie den Dateipfad, um das Netz zu importieren.

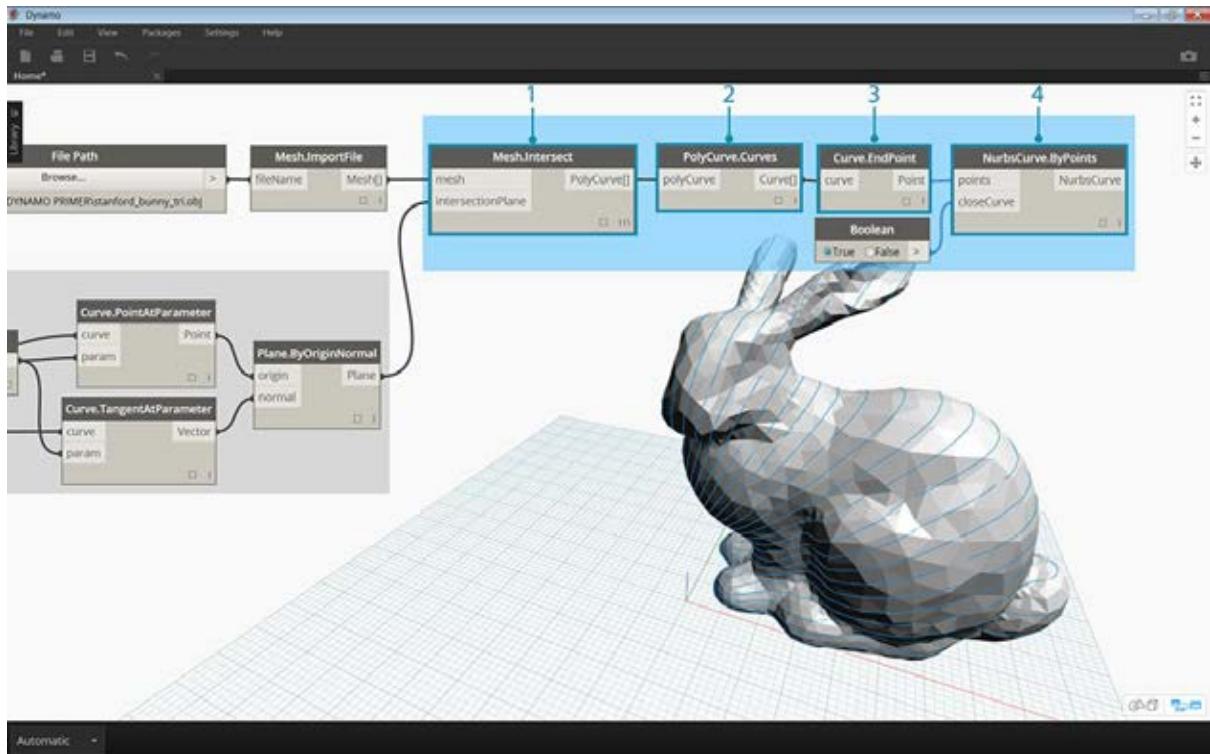


1. **Point.ByCoordinates:** Legen Sie einen Punkt fest. Dieser wird als Mittelpunkt eines Bogens verwendet.
2. **Arc.ByCenterPointRadiusAngle:** Konstruieren Sie einen Bogen um den Punkt. Mithilfe dieser Kurve wird eine Reihe von Ebenen platziert.



1. **Code Block**: Erstellen Sie eine Reihe von Zahlen zwischen Null und Eins.
2. **Curve.PointAtParameter**: Verbinden Sie den Bogen mit der *curve*-Eingabe und die Ausgabe des Codeblocks mit der *param*-Eingabe, um eine Reihe von Punkten entlang der Kurve abzurufen.
3. **Curve.TangentAtParameter**: Verbinden Sie die Eingaben auf dieselbe Weise wie beim vorigen Block.
4. **Plane.ByOriginNormal**: Verbinden Sie die Punkte mit der *origin*-Eingabe und die Vektoren mit der *normal*-Eingabe, um eine Reihe von Ebenen an den einzelnen Punkten zu erstellen.

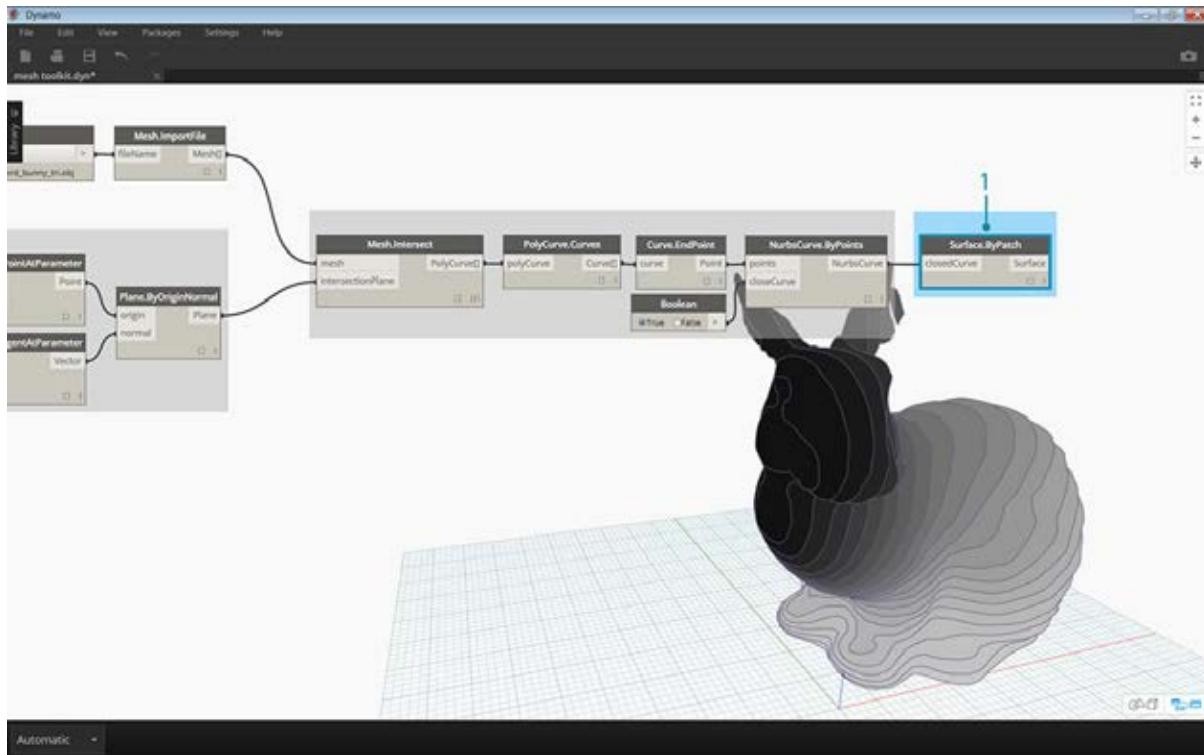
Damit sollte eine am Bogen entlang ausgerichtete Reihe von Ebenen angezeigt werden. Diese Ebenen verwenden Sie als Nächstes zum Schneiden des Netzes.



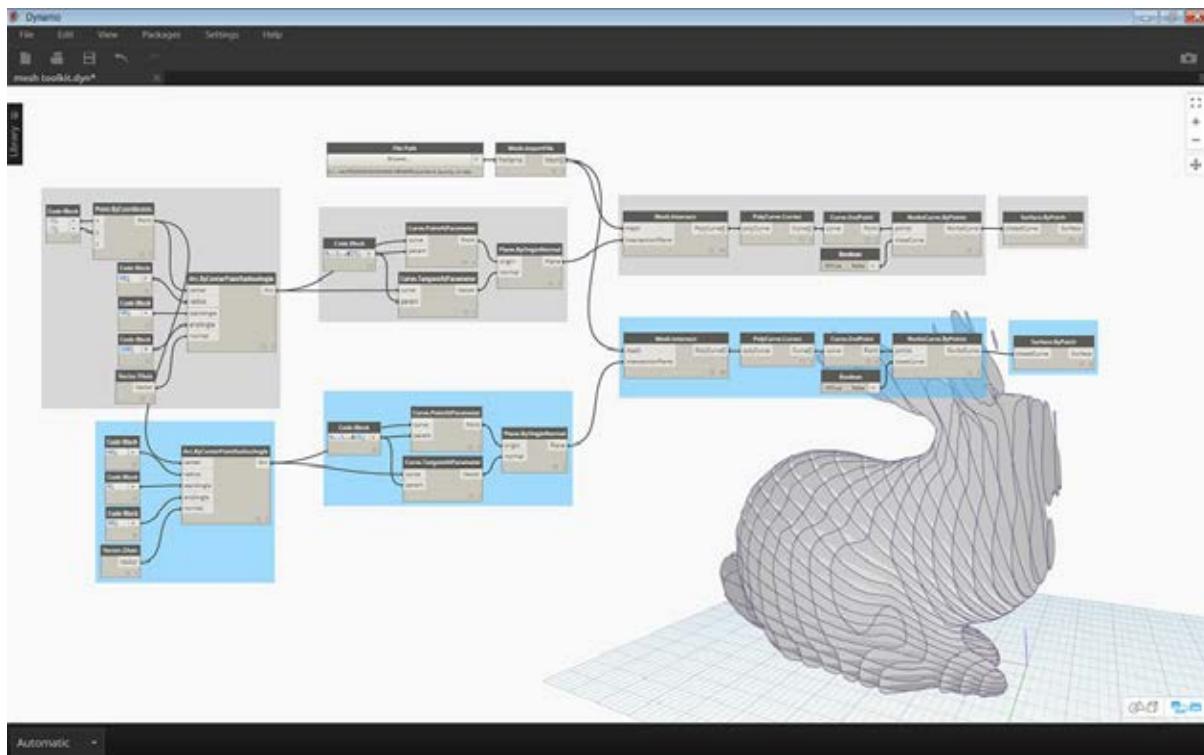
1. **Mesh.Intersect**: Schneiden Sie die Ebenen mit dem importierten Netz, sodass eine Reihe von

Polykurvenkonturen entsteht.

2. **PolyCurve.Curves**: Teilen Sie die Polykurven in ihre Kurvenfragmente auf.
3. **Curve.EndPoint**: Extrahieren Sie die Endpunkte der einzelnen Kurven.
4. **NurbsCurve.ByPoints**: Konstruieren Sie mithilfe der Punkte eine Nurbs-Kurve. Schließen Sie die Kurven mithilfe eines Boolean-Blocks mit dem Wert *True*.



1. **Surface.ByPatch**: Konstruieren Sie Oberflächenfelder für die einzelnen Konturen, um das Netz in Segmente aufzuteilen.



Fügen Sie eine zweite Segmentierung hinzu, um einen gitter- oder eierkartonähnlichen Effekt zu erzielen.

Ihnen ist vielleicht aufgefallen, dass die Schnittvorgänge für Netze schneller berechnet werden als für vergleichbare Volumenkörper. Arbeitsabläufe wie der in dieser Übung beschriebene eignen sich gut für die Arbeit mit Netzen.

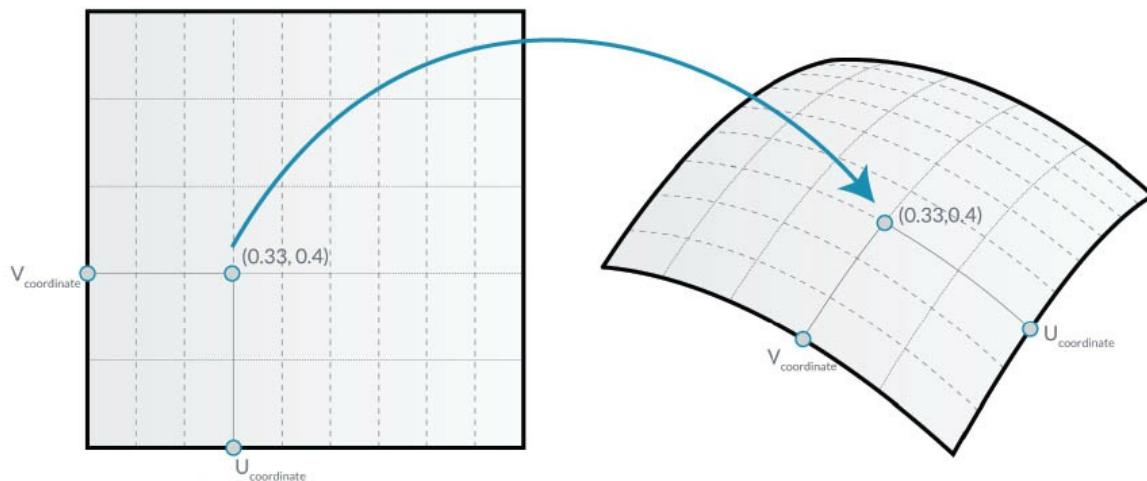
# Entwickeln von Paketen

## Entwickeln von Paketen

In Dynamo steht eine Reihe von Verfahren zum Entwickeln von Paketen zur Verfügung, die Sie für Ihre eigenen Zwecke verwenden oder für die Dynamo-Community bereitstellen können. In der folgenden Fallstudie wird ein bestehendes Paket zerlegt, um den Aufbau von Paketen zu demonstrieren. Dabei werden Lektionen aus dem vorigen Kapitel zugrunde gelegt: Eine Gruppe benutzerdefinierter Blöcke zum Zuordnen von Geometrie aus einer Dynamo-Oberfläche zu einer anderen mithilfe von UV-Koordinaten wird bereitgestellt.

### MapToSurface

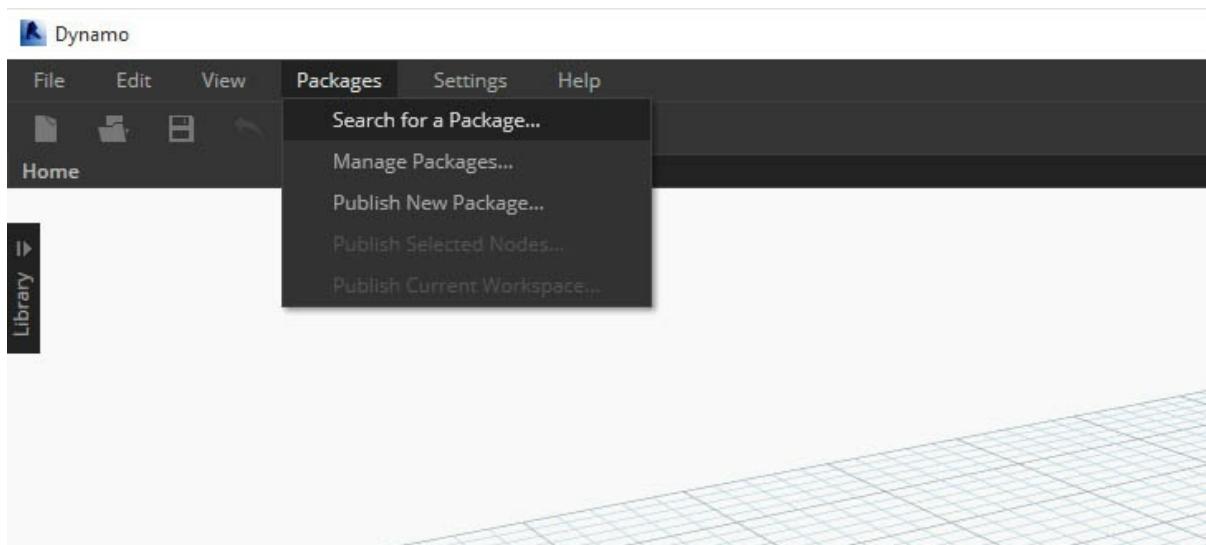
Sie arbeiten mit einem Beispieldatenpaket, das die UV-Zuordnung von Punkten aus einer Oberfläche zu einer anderen zeigt. Sie haben die Grundfunktionen dieses Werkzeugs bereits im Abschnitt [Erstellen eines benutzerdefinierten Blocks](#) in diesem Leitfaden erstellt. In den folgenden Dateien wird gezeigt, wie Sie die UV-Zuordnung weiterentwickeln und eine Gruppe von Werkzeugen für eine Bibliothek erstellen können, die anschließend veröffentlicht werden kann.



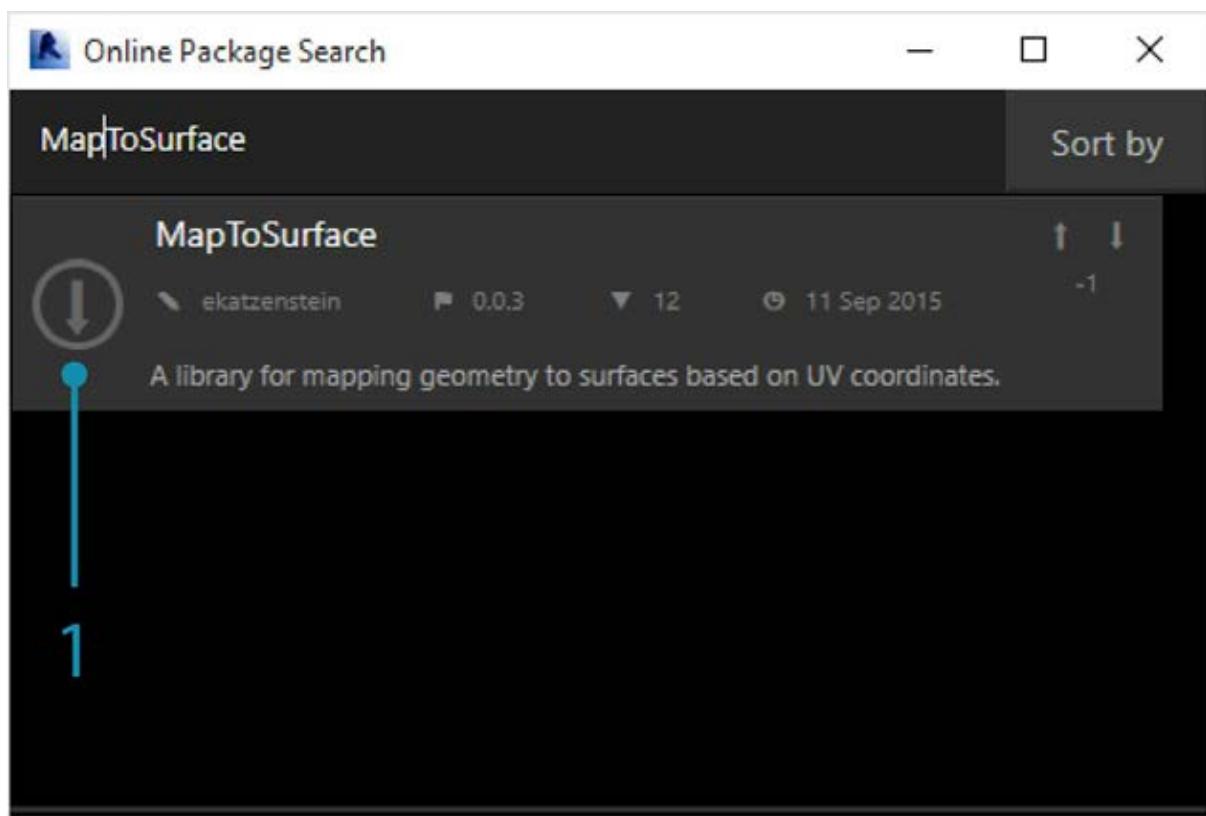
In der unten gezeigten Abbildung wird ein Punkt aus einer Oberfläche mithilfe von UV-Koordinaten einer anderen zugeordnet. Dem Paket liegt dasselbe Prinzip zugrunde, es wird jedoch komplexere Geometrie verwendet.

### Installieren des Pakets

Im vorigen Kapitel wurden in Dynamo verfügbare Methoden vorgestellt, mit denen eine Oberfläche mithilfe in der xy-Ebene definierter Kurven in Felder unterteilt werden kann. In dieser Fallstudie wird dies auf mehrdimensionale Geometrie erweitert. Dabei installieren Sie das fertige Paket und es wird gezeigt, wie dieses entwickelt wurde. Im nächsten Abschnitt wird gezeigt, wie dieses Paket veröffentlicht wurde.

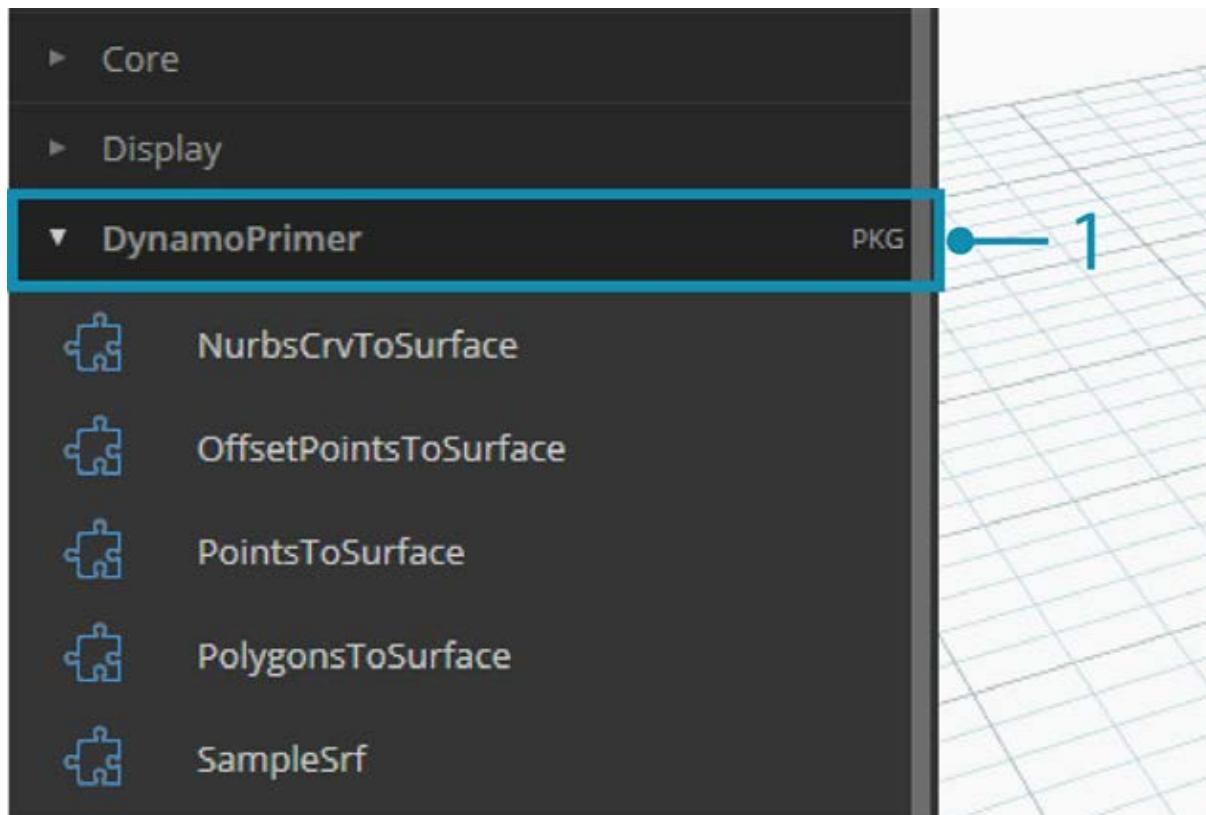


Dies ist der einfache Teil. Navigieren Sie in Dynamo zu *Pakete > Suchen nach Paket*.



Suchen Sie nach dem Paket *MapToSurface* (in einem Wort).

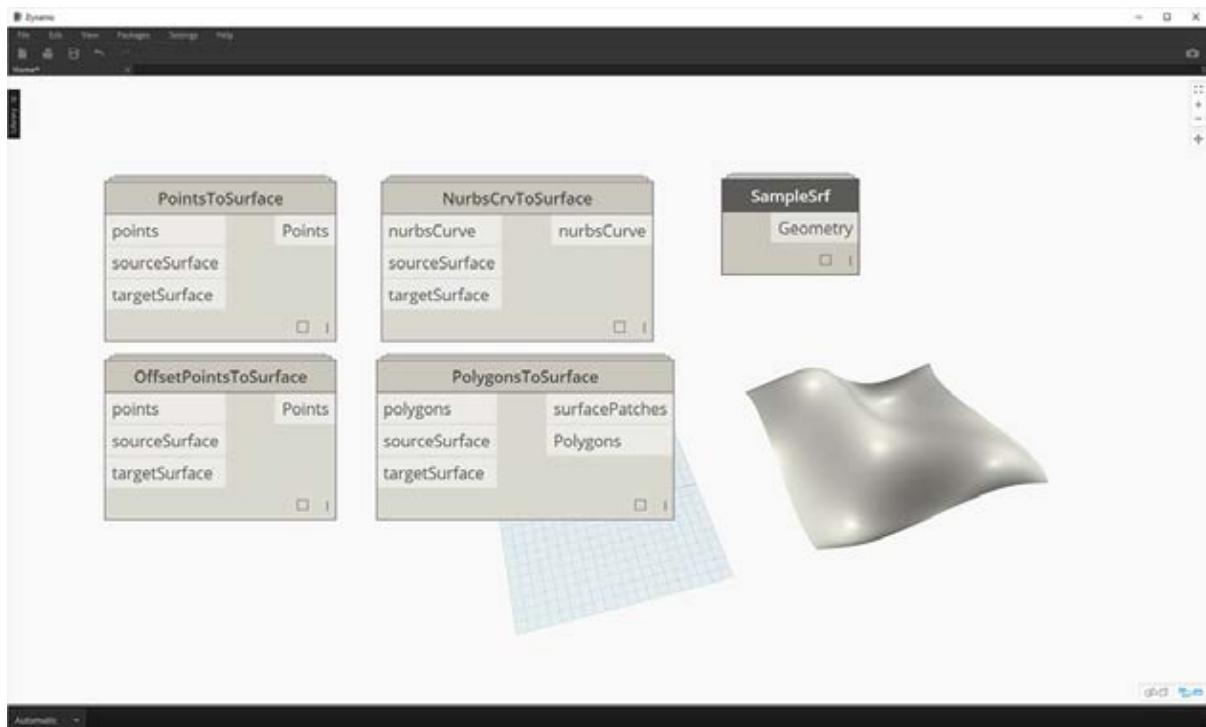
1. Wenn Sie das Paket gefunden haben, klicken Sie auf den Download-Pfeil links neben dem Paketnamen. Dadurch wird das Paket in Dynamo installiert.



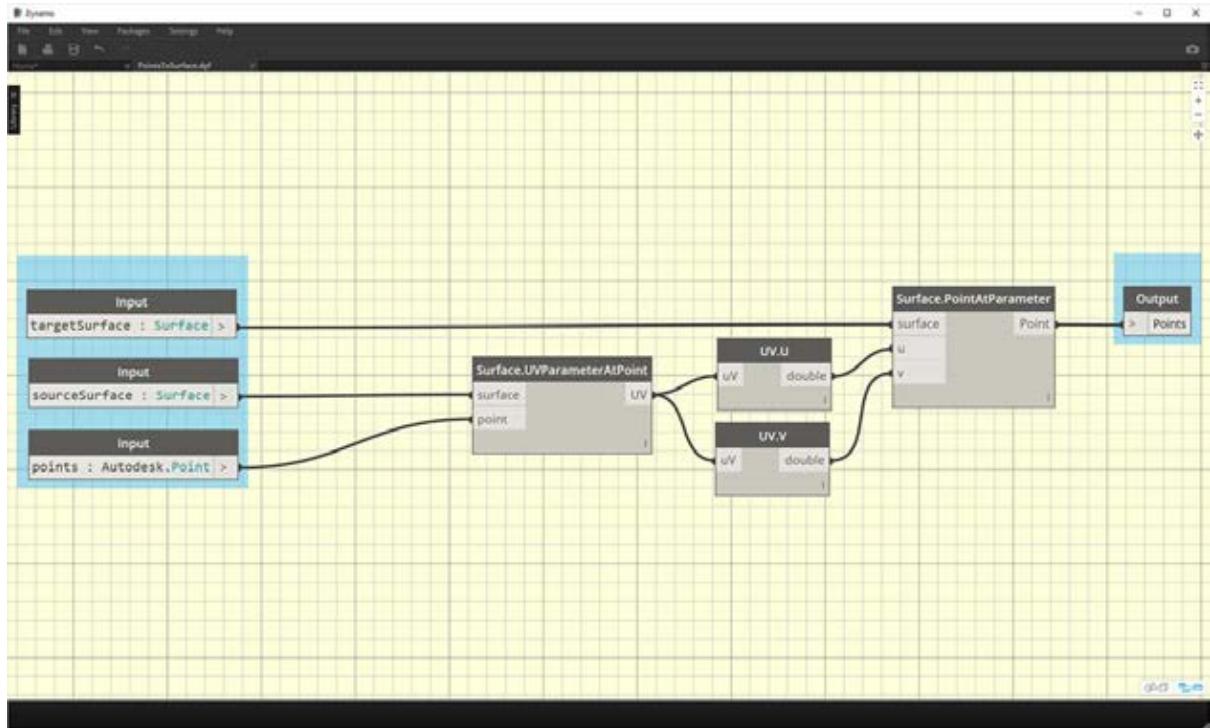
1. Nach der Installation werden die benutzerdefinierten Blöcke in der Gruppe **DynamoPrimer** oder in Ihrer **Dynamo-Bibliothek** angezeigt. Nachdem Sie das Paket installiert haben, analysieren Sie dieses jetzt.

### Benutzerdefinierte Blöcke

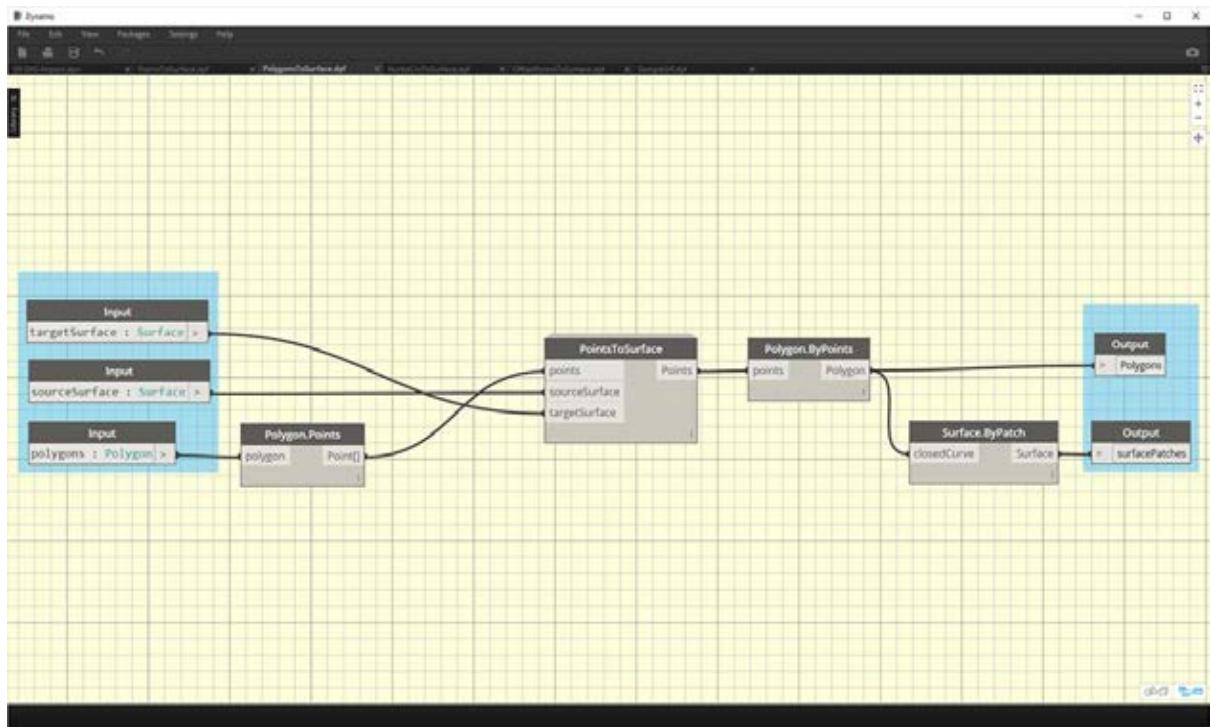
Für das Paket, das Sie hier erstellen, werden fünf benutzerdefinierte Blöcke verwendet. Diese Blöcke sind hier als Referenz bereits definiert. Im Folgenden analysieren Sie die Funktionen der einzelnen Blöcke. Einige der benutzerdefinierten Blöcke wurden ihrerseits unter Verwendung benutzerdefinierter Blöcke erstellt. Das Layout der Diagramme soll anderen Benutzern das Verständnis erleichtern.



Dieses Paket ist mit fünf benutzerdefinierten Blöcken recht einfach konstruiert. In den nachfolgenden Schritten wird der Aufbau der einzelnen Blöcke kurz erläutert.

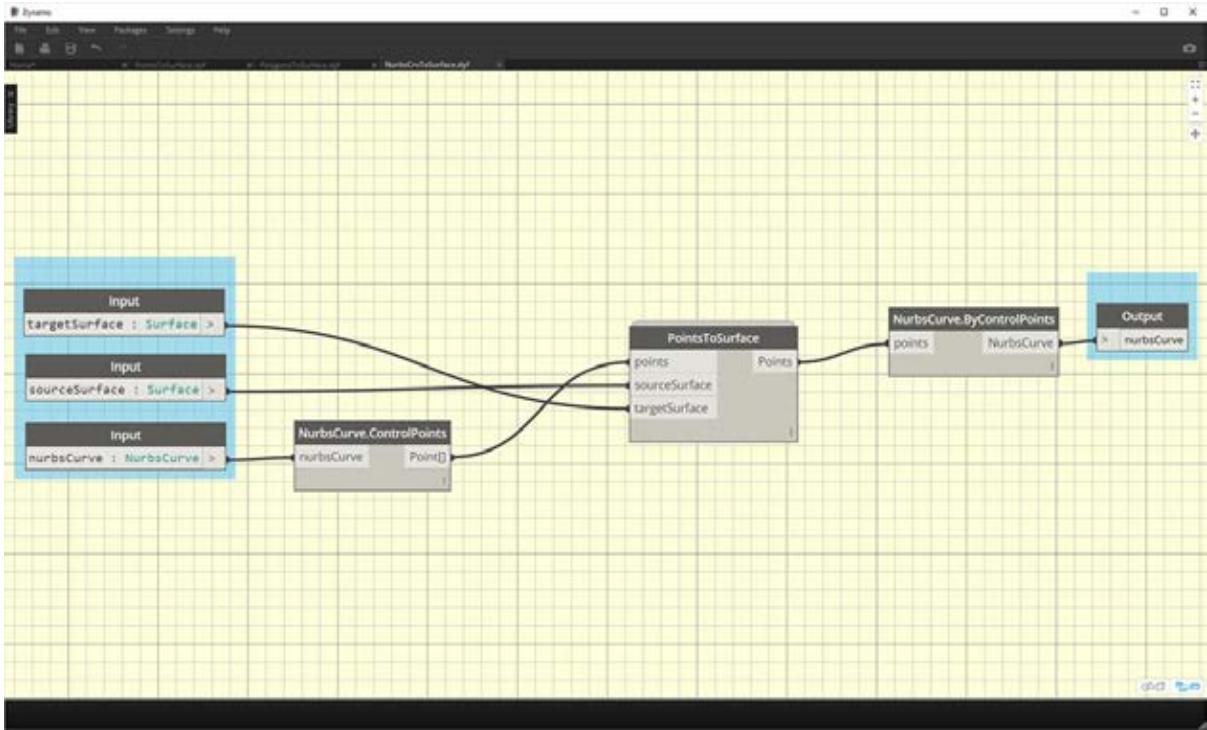


**PointsToSurface:** Dieser recht einfache benutzerdefinierte Block bildet die Grundlage für alle anderen Zuordnungsblöcke. Dieser Block ordnet, einfach ausgedrückt, einen Punkt mit UV-Koordinaten auf einer Quelloberfläche der Position mit den entsprechenden UV-Koordinaten auf der Zieloberfläche zu. Punkte sind das einfachste Geometrieelement, aus dem komplexere Geometrie erstellt wird. Aus diesem Grund können Sie mithilfe dieser Logik 2D- und sogar 3D-Geometrie von einer Oberfläche einer anderen zuweisen.

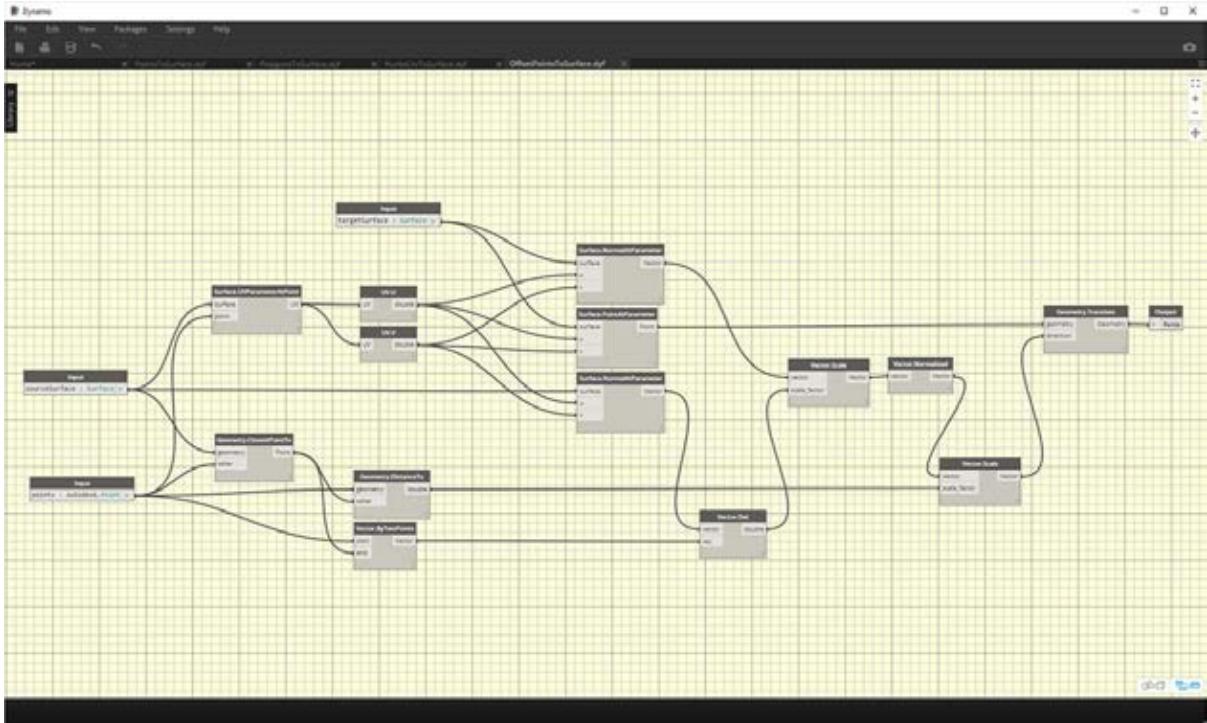


**PolygonsToSurface:** Die Logik zum Erweitern der Punktzuordnung von 1D- auf 2D-Geometrie wird hier auf einfache Weise anhand von Polygonen gezeigt. Beachten Sie, dass der *PointsToSurface*-Block in diesem Block verschachtelt ist. Dadurch können Sie Punkte jedes Polygons der Oberfläche zuordnen und dann das Polygon aus den zugeordneten Punkten rekonstruieren. Durch Verwendung der geeigneten Datenstruktur (eine Liste aus Listen von Punkten) bleiben

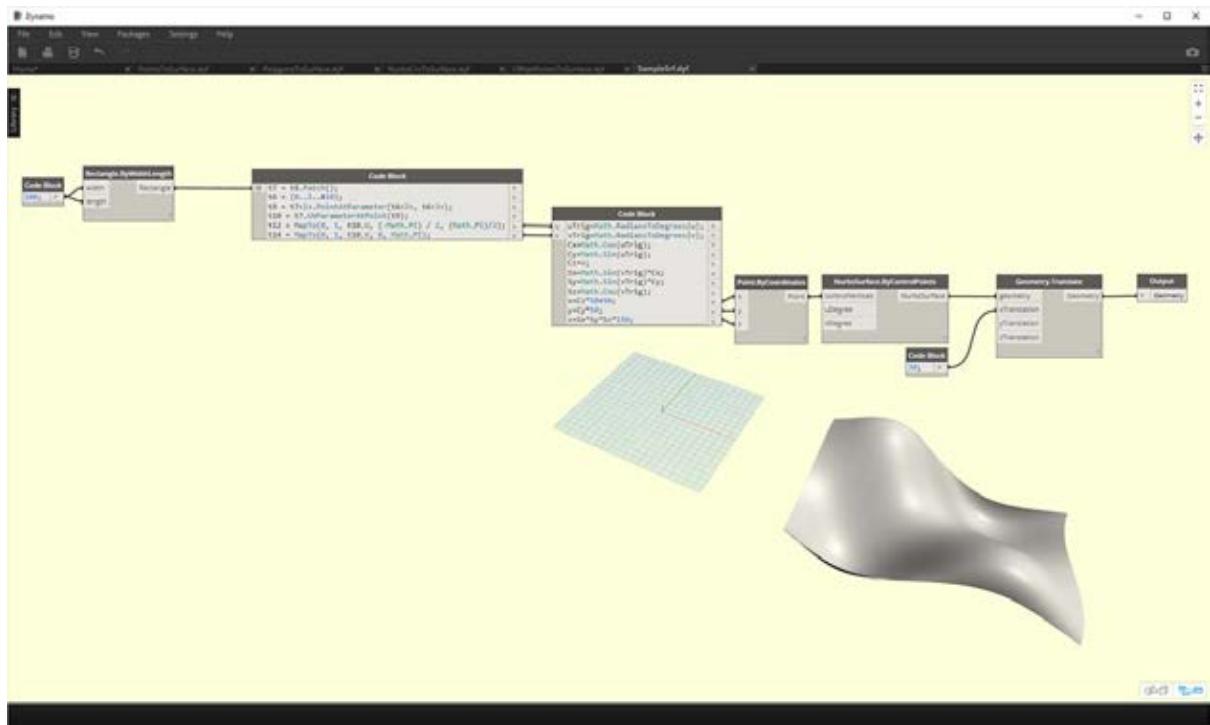
die Polygone nach ihrer Umwandlung in eine Punktsammlung als separate Polygone erhalten.



**NurbsCryptoSurface:** Hier kommt dieselbe Logik zum Einsatz wie beim *PolygonsToSurface*-Block. Dabei werden allerdings keine Punkte für Polygone, sondern Steuerpunkte für Nurbs-Kurven zugeordnet.



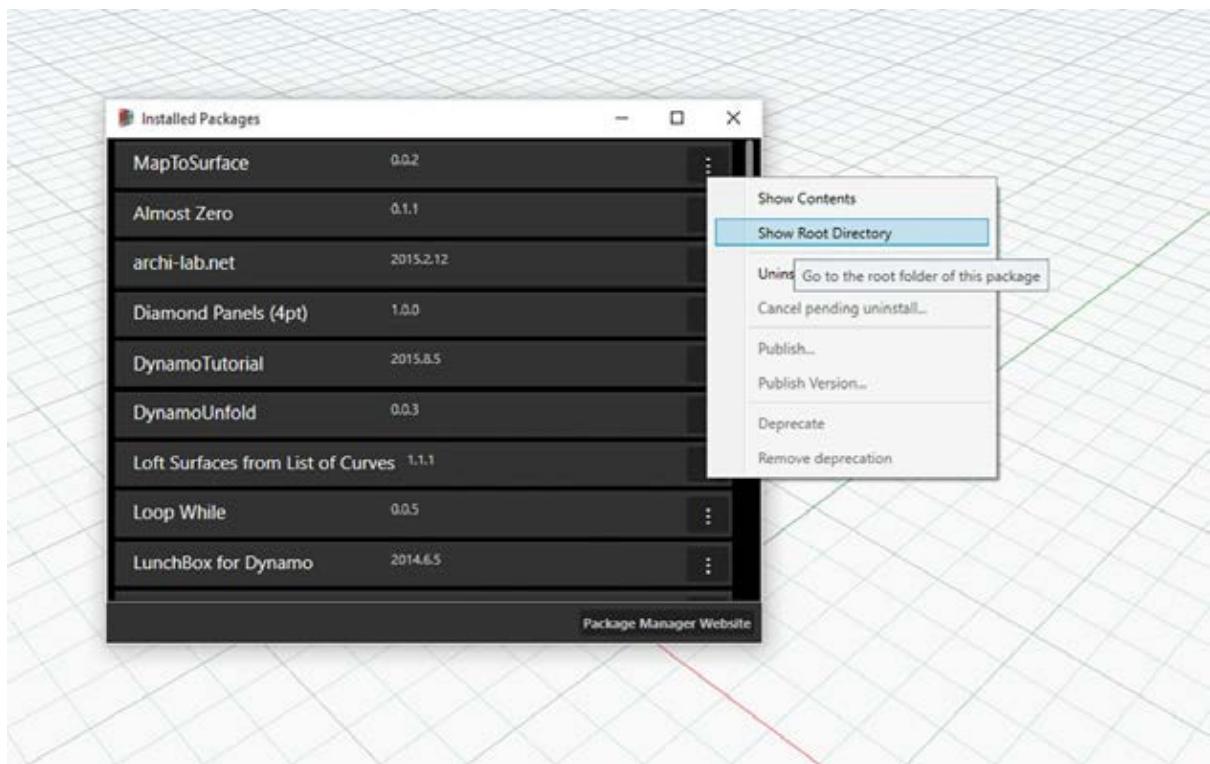
**OffsetPointsToSurface:** Dieser Block ist etwas komplexer, wobei jedoch ein einfaches Prinzip zugrunde liegt: Mit diesem Block werden genau wie beim *PointsToSurface*-Block Punkte aus einer Oberfläche einer anderen zugeordnet. Dabei werden jedoch auch Punkte berücksichtigt, die nicht auf der ursprünglichen Quelloberfläche liegen. Für diese Punkte wird ihr Abstand zum nächstgelegenen UV-Parameter abgerufen und dieser Abstand wird der Normalen an der entsprechenden UV-Koordinatenposition der Zielloberfläche zugeordnet. Bei der Bearbeitung der Beispieldateien wird dies leichter verständlich.



**SampleSrf:** Mit diesem einfachen Block wird eine parametrische Oberfläche erstellt, die aus dem Quellraster einer gewellten Oberfläche in den Beispieldateien zugeordnet werden kann.

## Beispieldateien

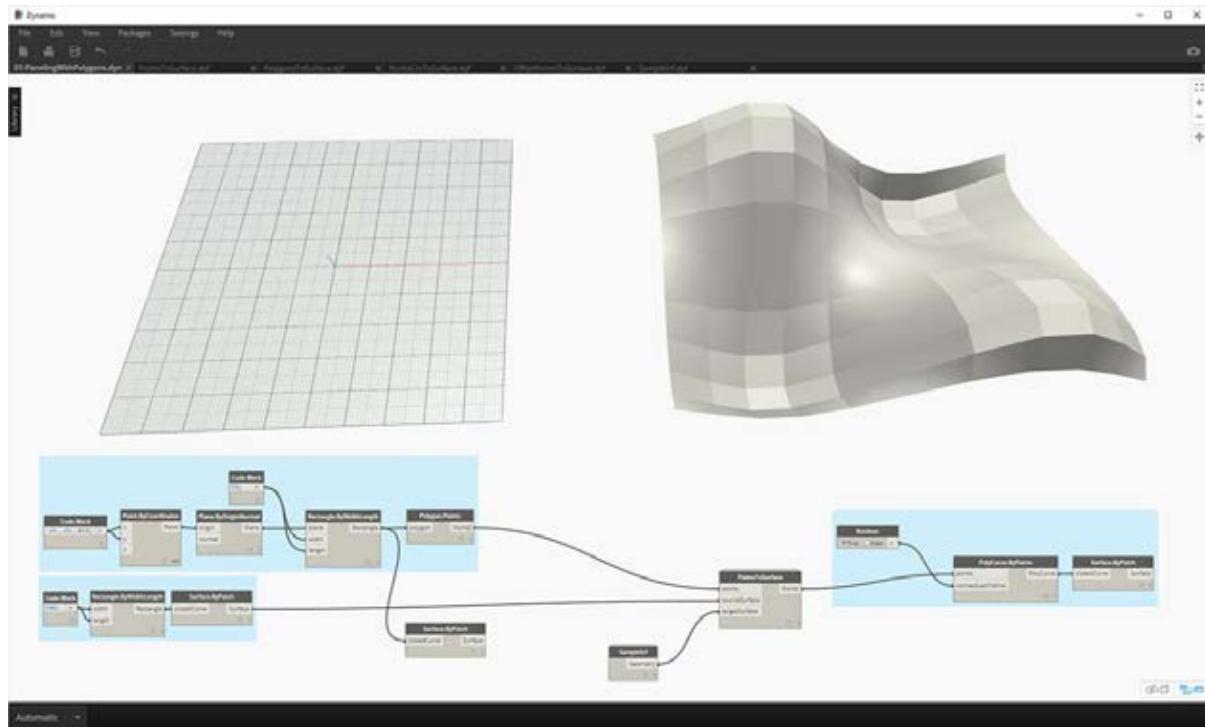
Die Beispieldateien befinden sich im Stammordner des Pakets. Navigieren Sie in Dynamo über *Pakete > Pakete verwalten* zu diesem Ordner.



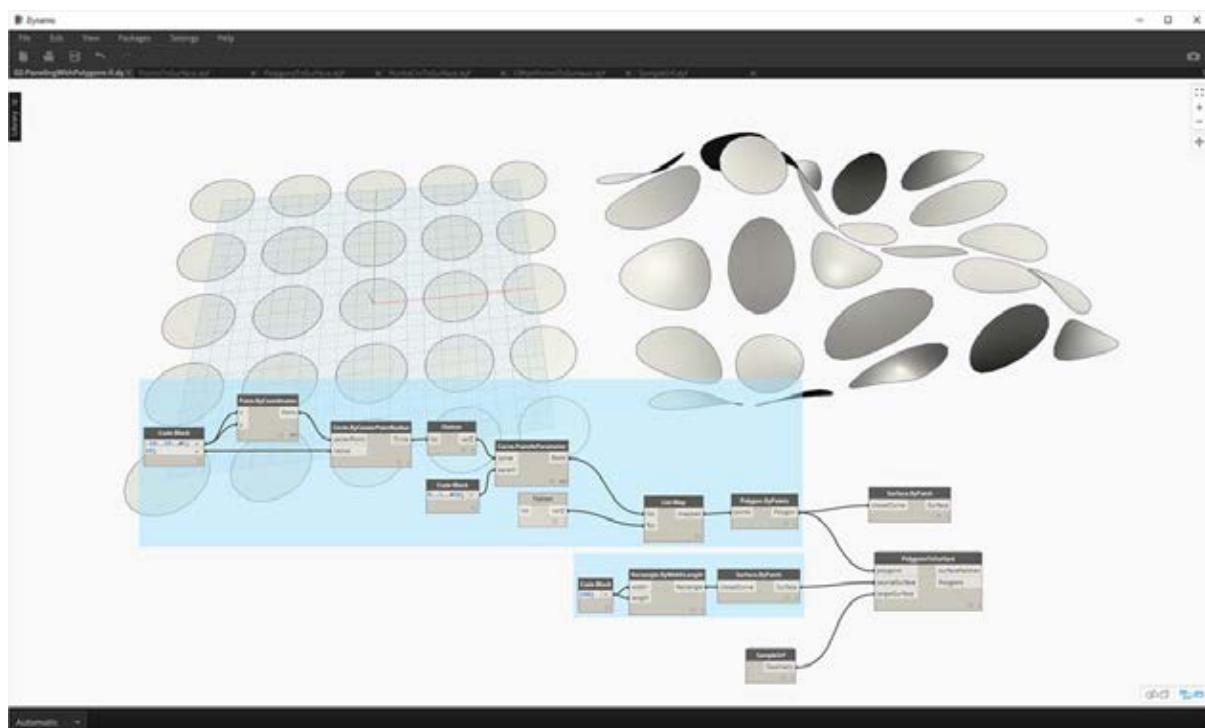
Klicken im Fenster zur Verwaltung von Paketen auf die drei senkrechten Punkte rechts neben *MapToSurface* und wählen Sie *Stammverzeichnis anzeigen*.

Navigieren Sie im geöffneten Stammverzeichnis zum Ordner *extra*, in dem sich alle Dateien des Pakets befinden, die keine

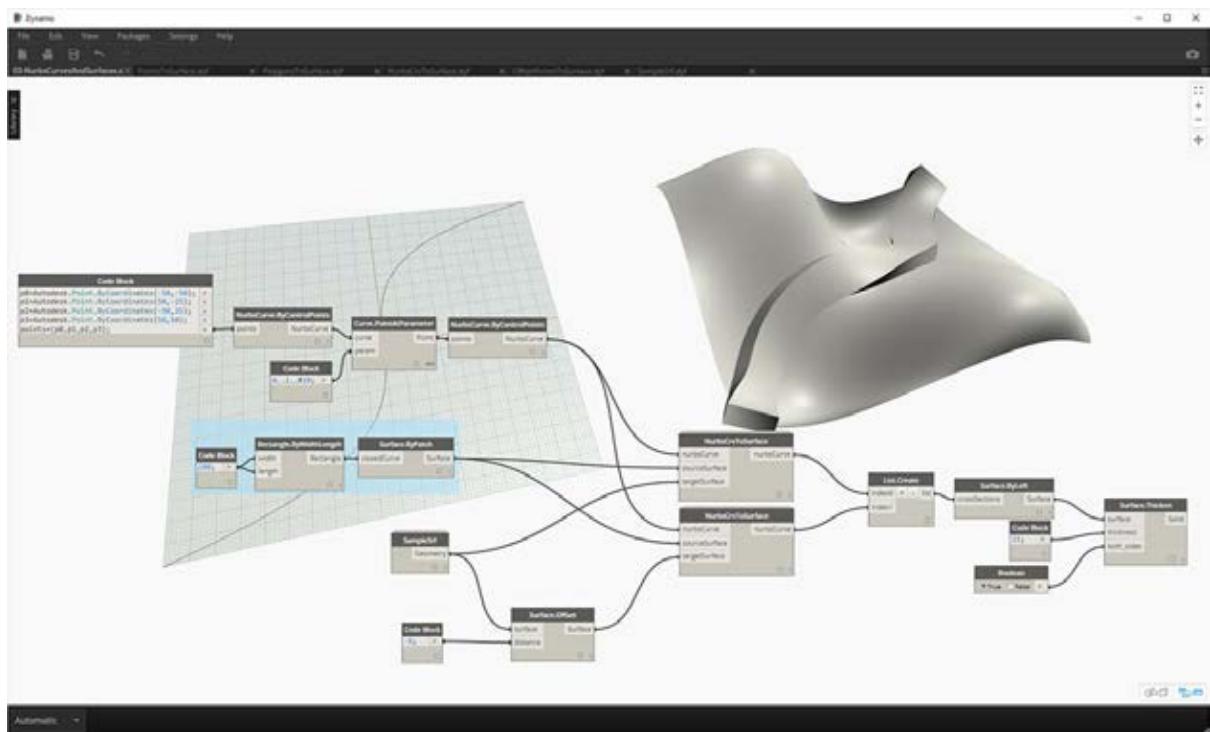
benutzerdefinierten Blöcke sind. Hier werden Beispieldateien (sofern vorhanden) für Dynamo-Pakete abgelegt. Mit den im Folgenden gezeigten Abbildungen werden die in den einzelnen Beispieldateien gezeigten Konzepte erläutert.



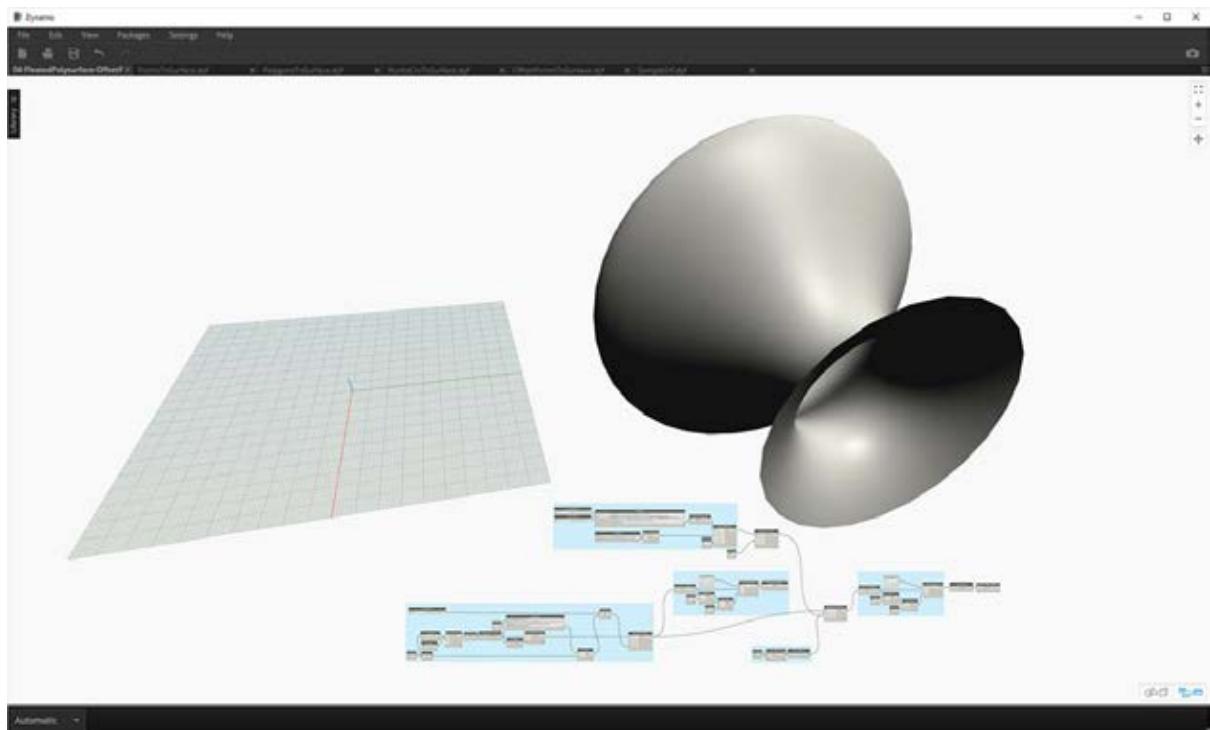
**01-PanelingWithPolygons:** Dieses Beispiel zeigt die Unterteilung einer Oberfläche anhand eines Rasters aus Rechtecken mithilfe von *PointsToSurface*. Dieser Vorgang ist Ihnen wahrscheinlich vertraut, da ein ähnlicher Arbeitsablauf im [vorigen Kapitel](#) gezeigt wurde.



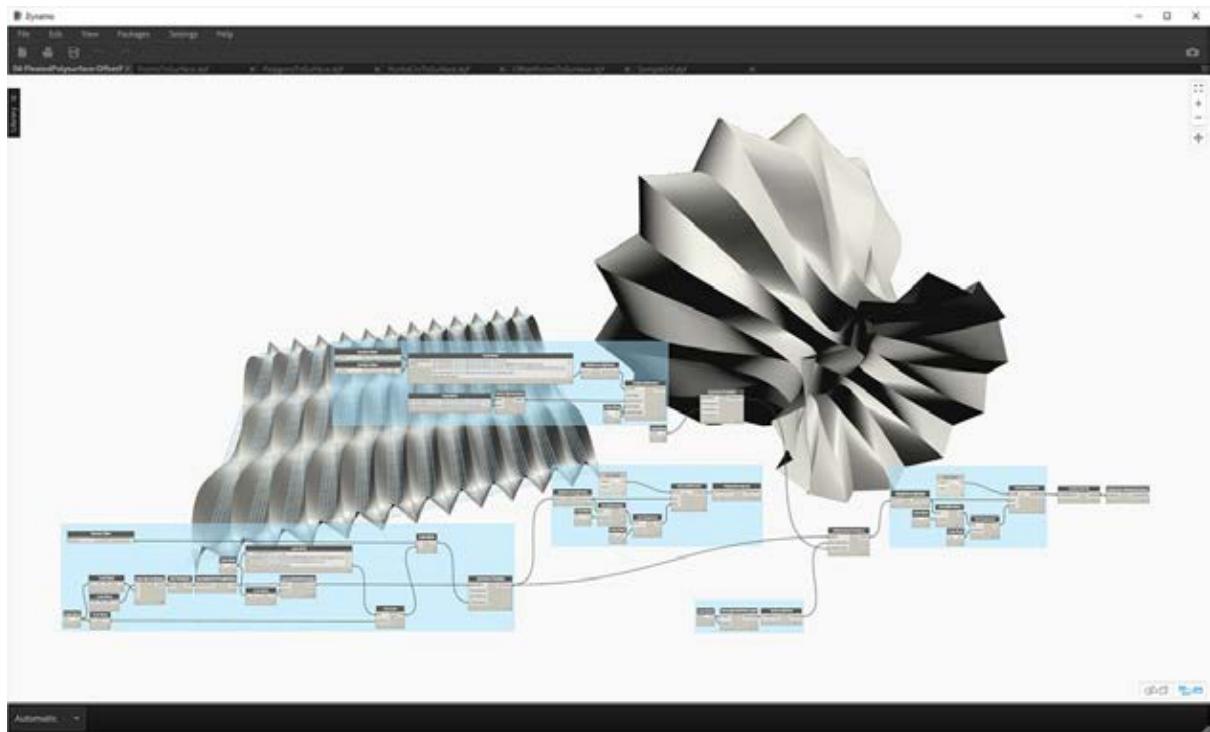
**02-PanelingWithPolygons-II:** In dieser Übungsdatei wird eine Einrichtung zum Zuordnen von Kreisen (bzw. Polygonen, die Kreise repräsentieren) von einer Oberfläche zu einer anderen gezeigt. Hierfür wird der *PolygonsToSurface*-Block verwendet.



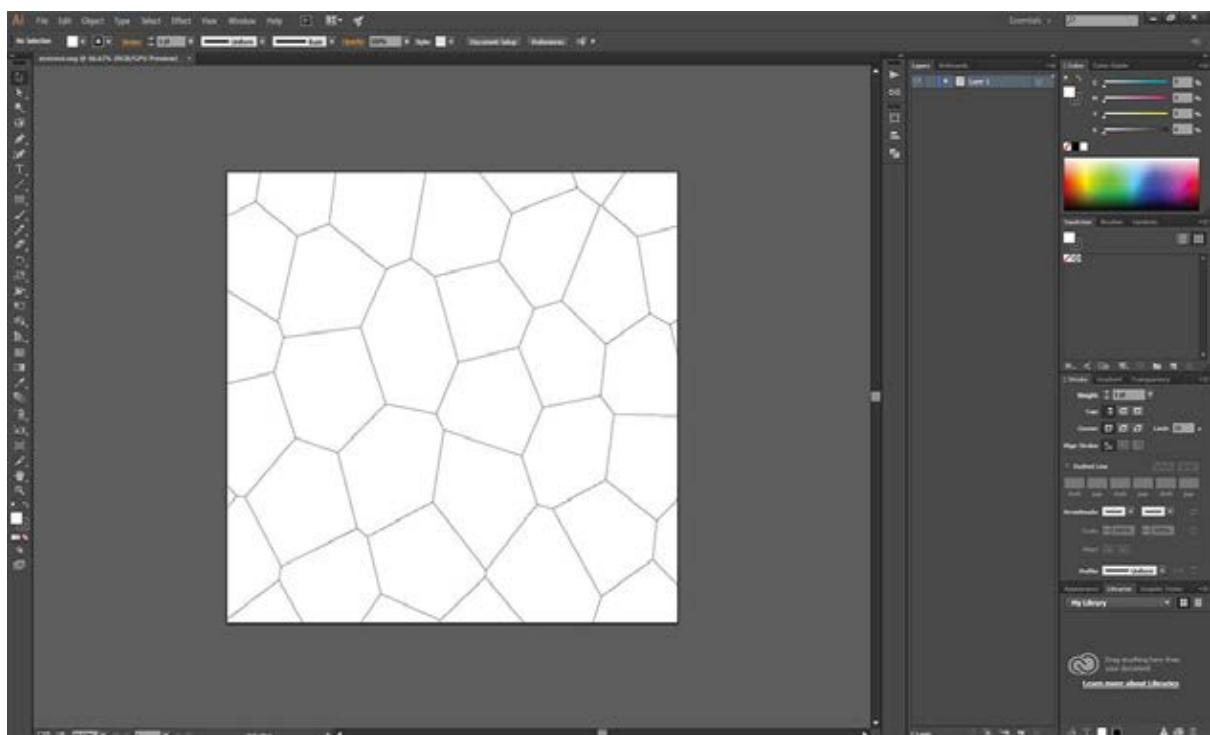
**03-NurbsCrvsAndSurface:** Diese Beispieldatei zeigt einen komplexeren Vorgang unter Verwendung des NurbsCrvToSurface-Blocks. Die Zieloberfläche wird um eine angegebene Strecke versetzt und die Nurbs-Kurve wird sowohl der ursprünglichen Zieloberfläche als auch der versetzten Oberfläche zugeordnet. Die beiden zugeordneten Kurven werden anschließend durch eine Erhebung verbunden, sodass eine Oberfläche entsteht, die anschließend verdickt wird. Der resultierende Körper weist eine gewellte Form auf, die den Normalen der Zieloberfläche folgt.



**04-PleatedPolysurface-OffsetPoints:** In dieser Beispieldatei wird die Zuordnung einer gefalteten Polysurface von einer Quell- zu einer Zieloberfläche gezeigt. Die Quell- und Zieloberfläche sind eine rechteckige, das ganze Raster umfassende Oberfläche und ein Rotationskörper.



**04-PleatedPolysurface-OffsetPoints:** Zuordnung der Quell-Polysurface von der Quell- zur Zieloberfläche.

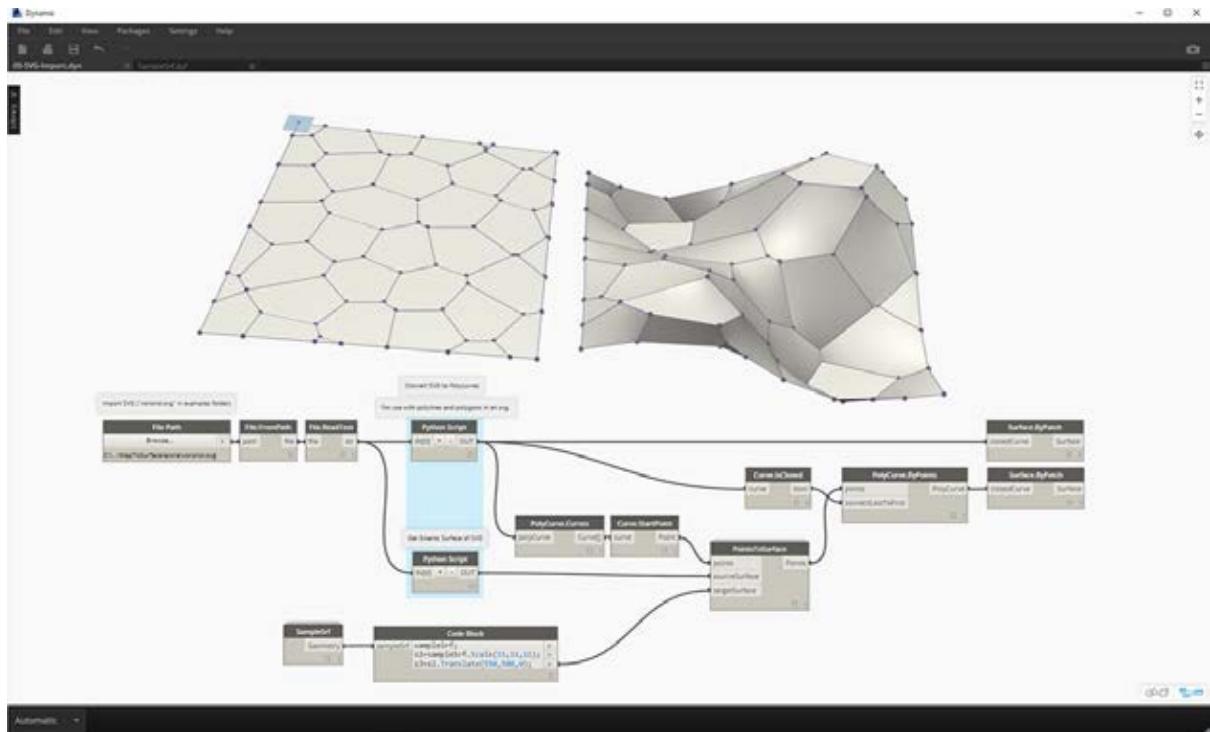


**05-SVG-Import:** Mithilfe der benutzerdefinierten Blöcke können Kurven unterschiedlicher Typen zugeordnet werden. In dieser letzten Datei wird daher eine aus Illustrator exportierte SVG-Datei referenziert und die importierten Kurven werden einer Zieloberfläche zugeordnet.

The screenshot shows a Python script editor window titled "Edit Python Script...". The script contains code for parsing SVG files into Dynamo Polycurves. It includes functions for extracting segments from paths and points from polygons, and a function for getting the viewBox. The code uses regular expressions and the System.Xml namespace. At the bottom right of the editor window are two buttons: "Accept Changes" and "Cancel".

```
1 import clr
2 import re
3 clr.AddReference("System.Xml")
4 import System.Xml
5
6 xmlDoc = System.Xml.XmlDocument()
7 xmlDoc.Load(IN[0])
8
9 OUT=[]
10
11 #for shorthanded quadratic and cubic bezier curves
12 svgDict={}
13 svgDict["et"]=[]
14 svgDict["ed"]=[]
15
16 def segsFromPath(data):
17     subOUT=[]
18     splitPath=re.findall('[A-Za-z][^A-Za-z]*',"".join(line.strip() for line in data))
19     return splitPath
20
21 def viewBox():
22     items = xmlDoc.GetElementsByTagName("svg")
23     for item in items:
24         box=item.getAttribute("viewBox")
25     nums=box.split(' ')
26     floats=[]
27     for num in nums:
28         floats.append(float(num))
29     OUT.append(["viewBox",floats])
30
31
32 def ptsFromPoly(data):
33     subOUT=[]
34     pts=data.split(" ")
35     for points in pts:
36         ptList=points.split(",")
37         if len(ptList)>1:
38             numX=float(ptList[0])
39             numY=float(ptList[1])
40             geoPt=[numX,numY]
41             subOUT.append(geoPt)
```

**05-SVG-Import:** Durch Analyse der Syntax einer SVG-Datei werden die Kurven aus dem XML-Format in Dynamo-Polykurven konvertiert.

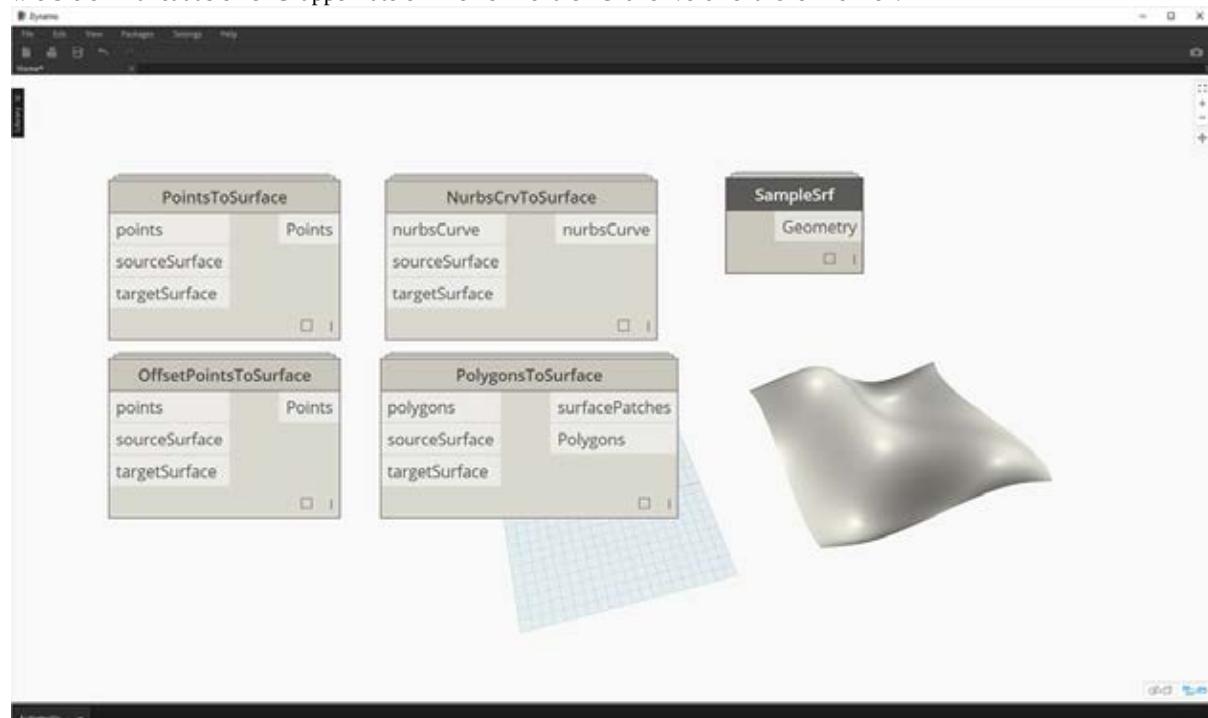


**05-SVG-Import:** Die importierten Kurven werden einer Zieloberfläche zugeordnet. Dieser Vorgang ermöglicht es, eine Unterteilung explizit (durch Zeigen und Klicken) in Illustrator zu entwerfen, und sie anschließend in Dynamo zu importieren und auf eine Zieloberfläche anzuwenden.

# Veröffentlichen von Paketen

## Veröffentlichen von Paketen

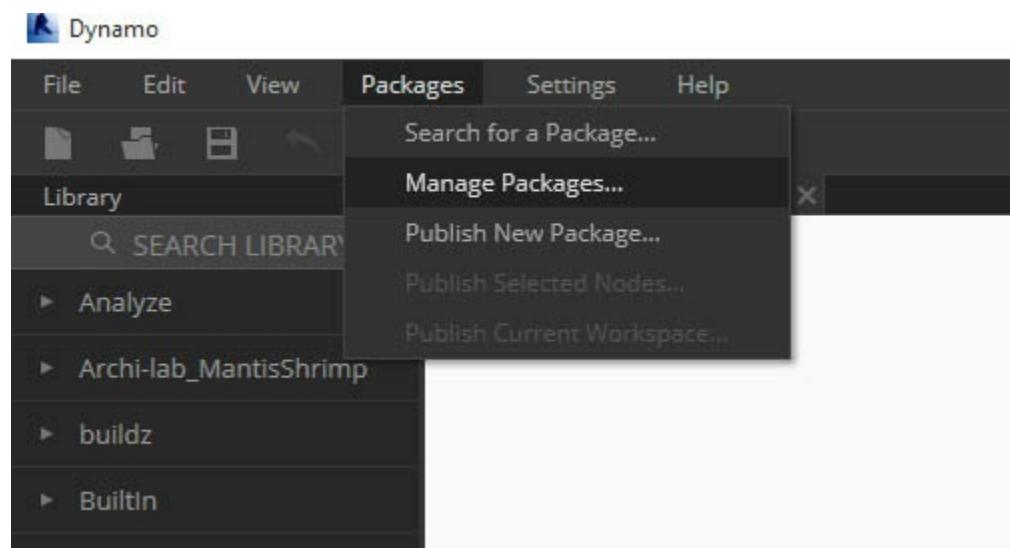
In den vorigen Abschnitten wurde gezeigt, wie das *MapToSurface*-Paket sich aus benutzerdefinierten Blöcken und Beispieldateien zusammensetzt. Aber wie veröffentlichen Sie ein Paket, das lokal entwickelt wurde? Diese Fallstudie zeigt, wie Sie ein Paket aus einer Gruppe Dateien in einem lokalen Ordner veröffentlichen können.



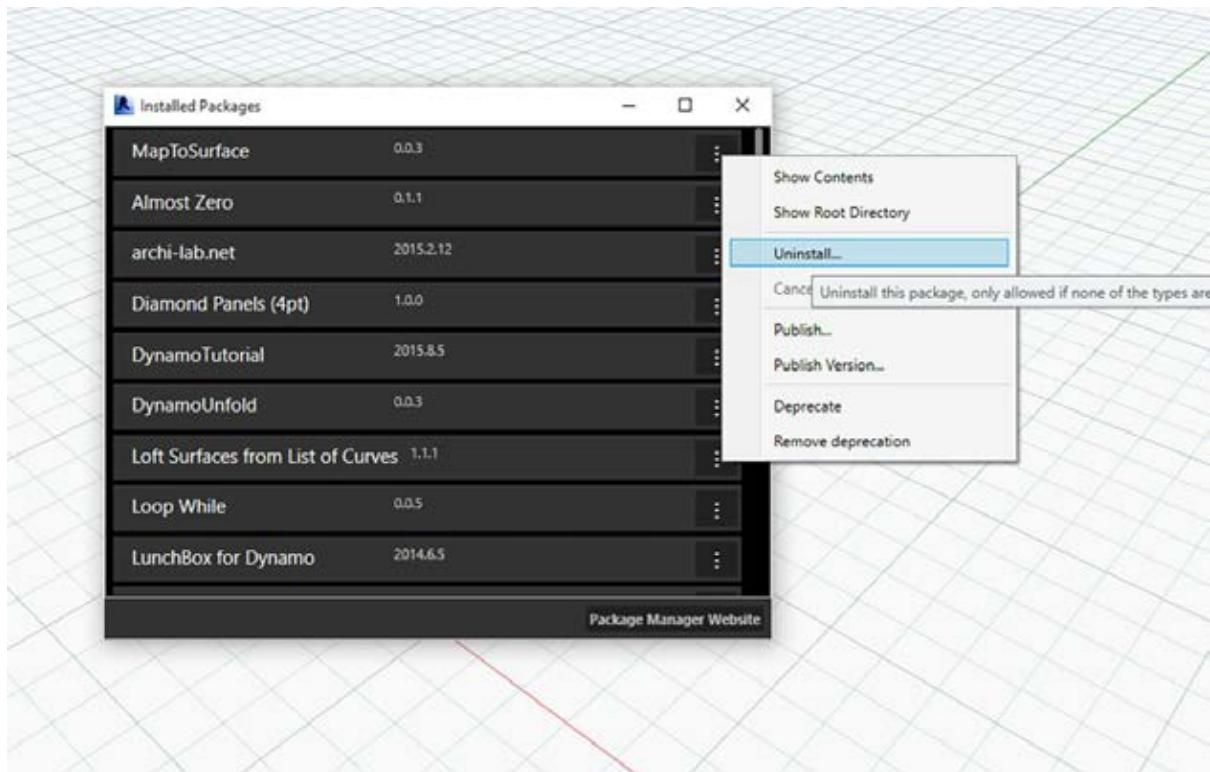
Es gibt mehrere Möglichkeiten zum Veröffentlichen von Paketen. Im Folgenden wird der von uns empfohlene Prozess beschrieben: **Sie veröffentlichen lokal, entwickeln lokal und veröffentlichen schließlich online**. Sie beginnen mit einem Ordner, der sämtliche Dateien im Paket enthält.

### Deinstallieren eines Pakets

Bevor Sie mit der Veröffentlichung des *MapToSurface*-Pakets beginnen, deinstallieren Sie das Paket aus der vorigen Lektion, falls Sie es installiert haben. Dadurch vermeiden Sie, mit identischen Paketen zu arbeiten.



Beginnen Sie, indem Sie *Pakete > Pakete verwalten* wählen.

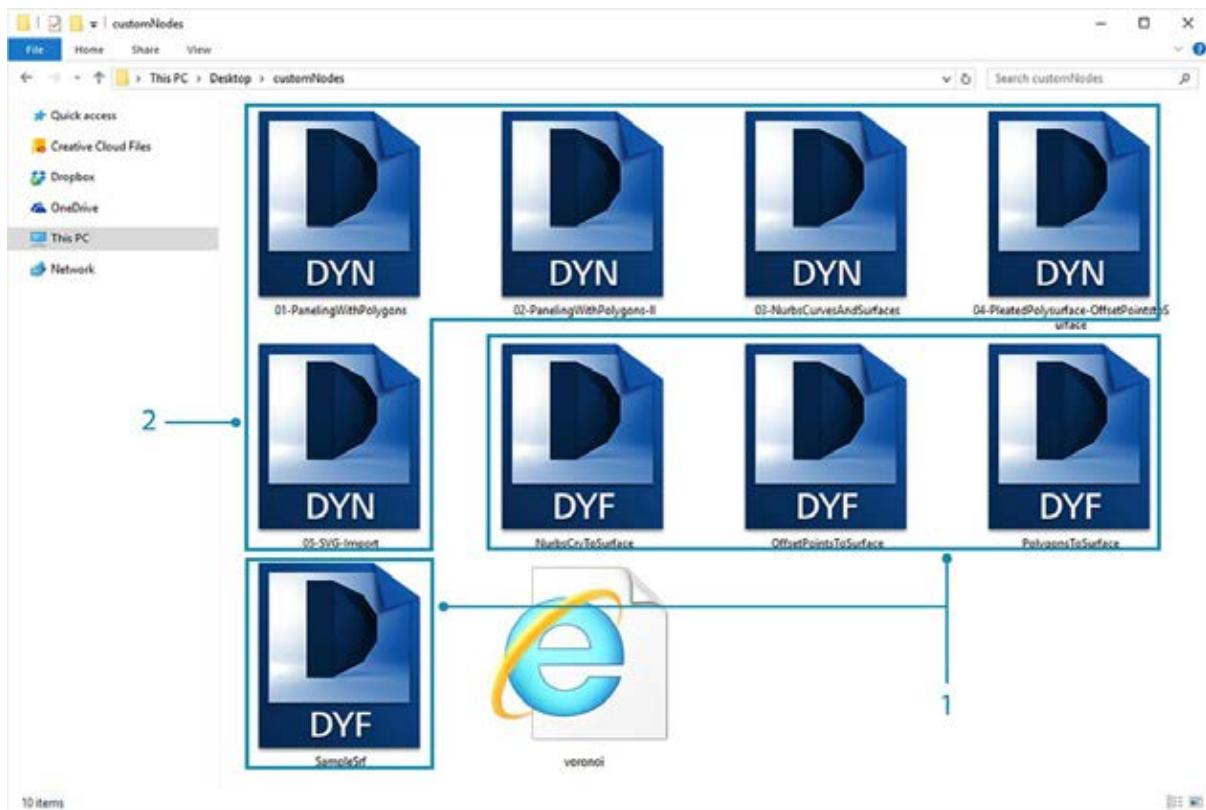


Wählen Sie die Schaltfläche für *MapToSurface* und wählen Sie *Deinstallieren*. Starten Sie dann Dynamo erneut. Wenn Sie beim erneuten Öffnen des Fenster *Pakete verwalten* überprüfen, darf *MapToSurface* dort nicht mehr vorhanden sein. Jetzt können Sie den Vorgang von Anfang an durchführen.

## Lokale Veröffentlichung von Paketen

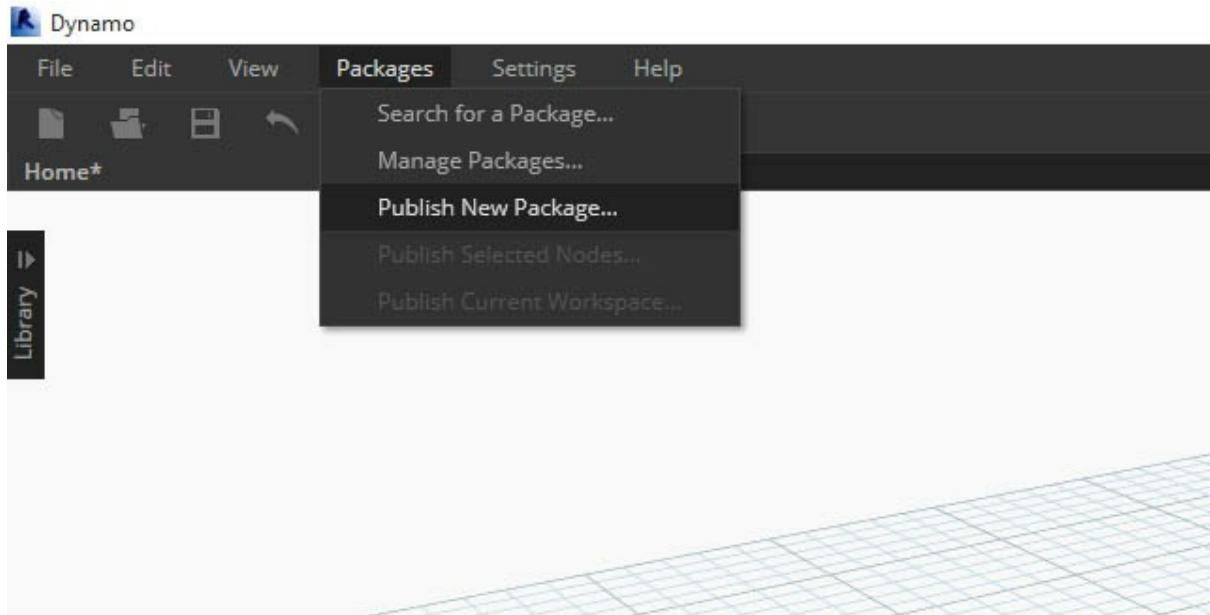
Anmerkung: Zu dem Zeitpunkt, als dieses Dokument verfasst wurde, war die Veröffentlichung von Dynamo-Paketen nur in *Dynamo Studio* und *Dynamo for Revit* aktiviert. *Dynamo Sandbox* verfügte nicht über Veröffentlichungsfunktionen.

Laden Sie die Beispieldateien für die Fallstudie zu diesem Paket herunter (durch Rechtsklicken und Wahl von Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [MapToSurface.zip](#)

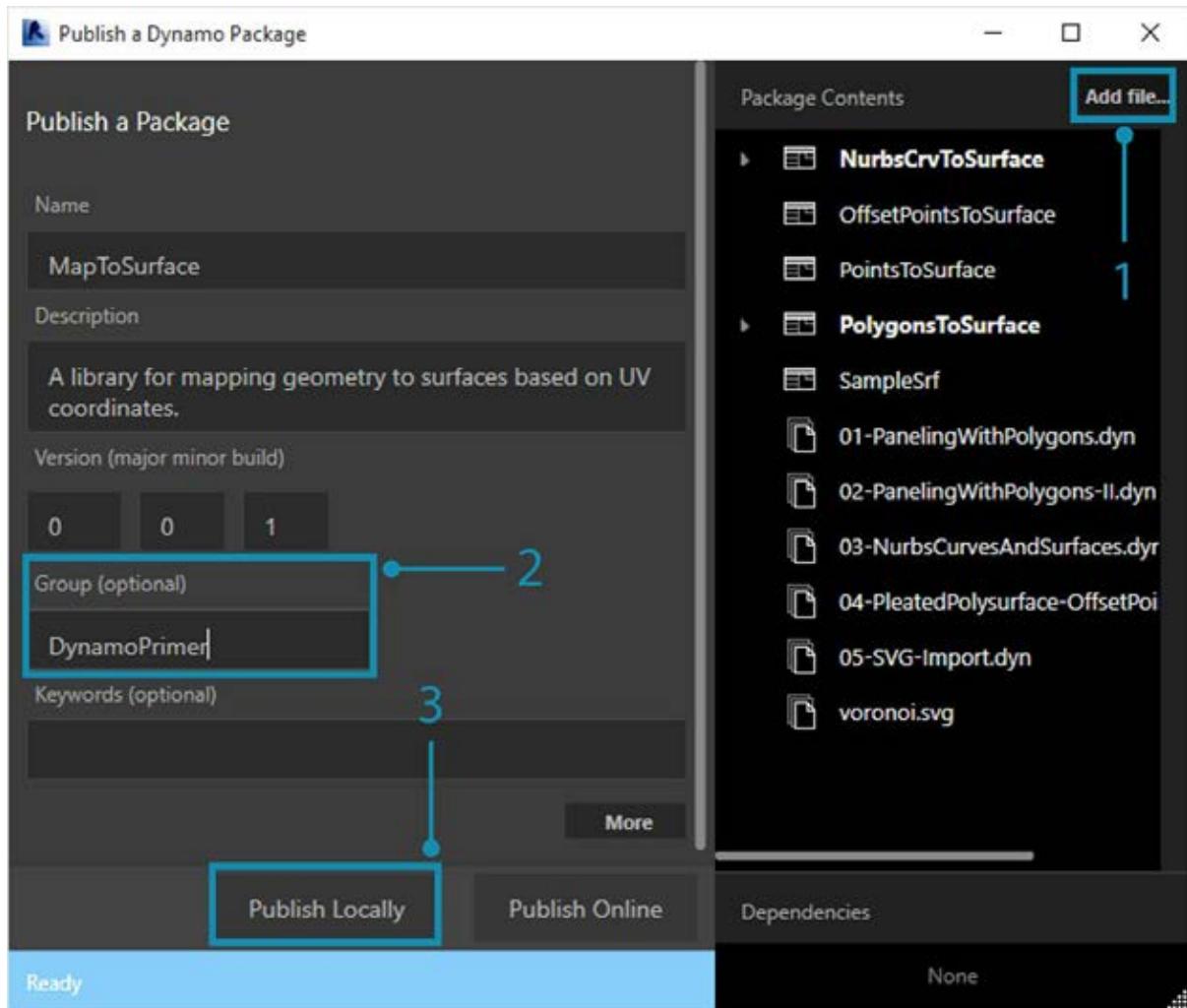


Sie übermitteln Ihr Paket zum ersten Mal und alle Beispieldateien und benutzerdefinierten Blöcke befinden sich im selben Ordner. Nachdem dieser Ordner vorbereitet ist, können Sie jetzt auf Dynamo Package Manager hochladen.

1. Der Ordner enthält fünf benutzerdefinierte Blöcke (.dyf).
2. Er enthält außerdem fünf Beispieldateien (.dyn) und eine importierte Vektordatei (.svg). Diese Dateien dienen als einführende Übungen, die dem Benutzer die Arbeit mit den benutzerdefinierten Blöcken erläutern sollen.

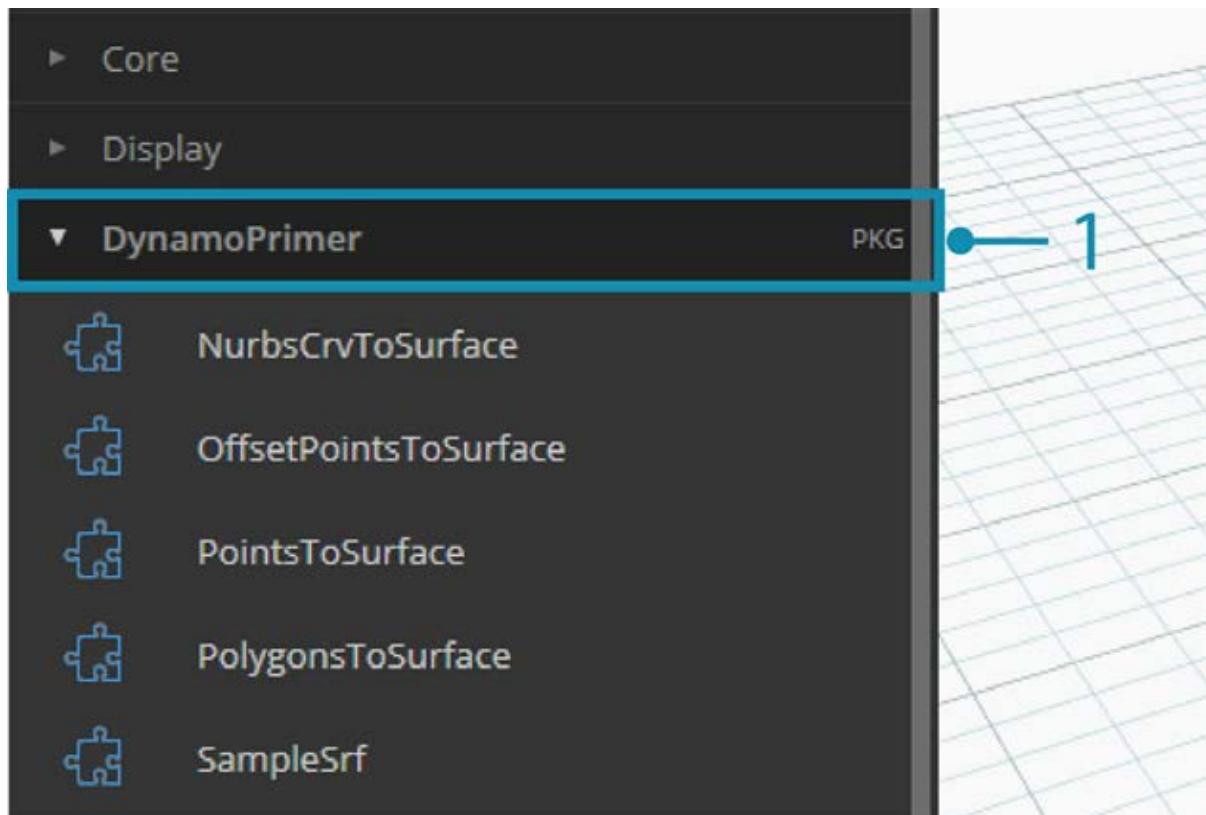


Klicken Sie in Dynamo zunächst auf *Pakete > Neues Paket veröffentlichen*.

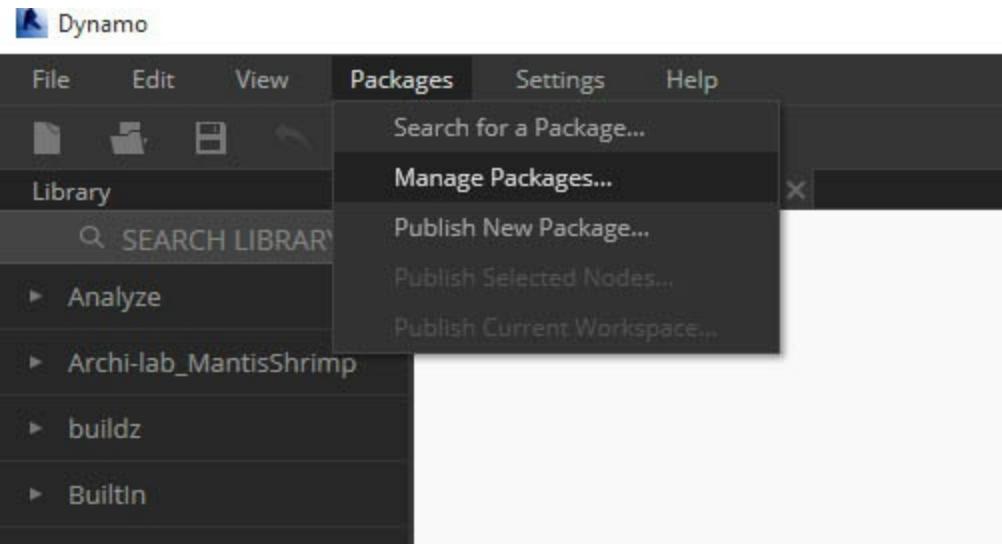


In Fenster *Paket publizieren* ist das relevante Formular im linken Bereich bereits ausgefüllt.

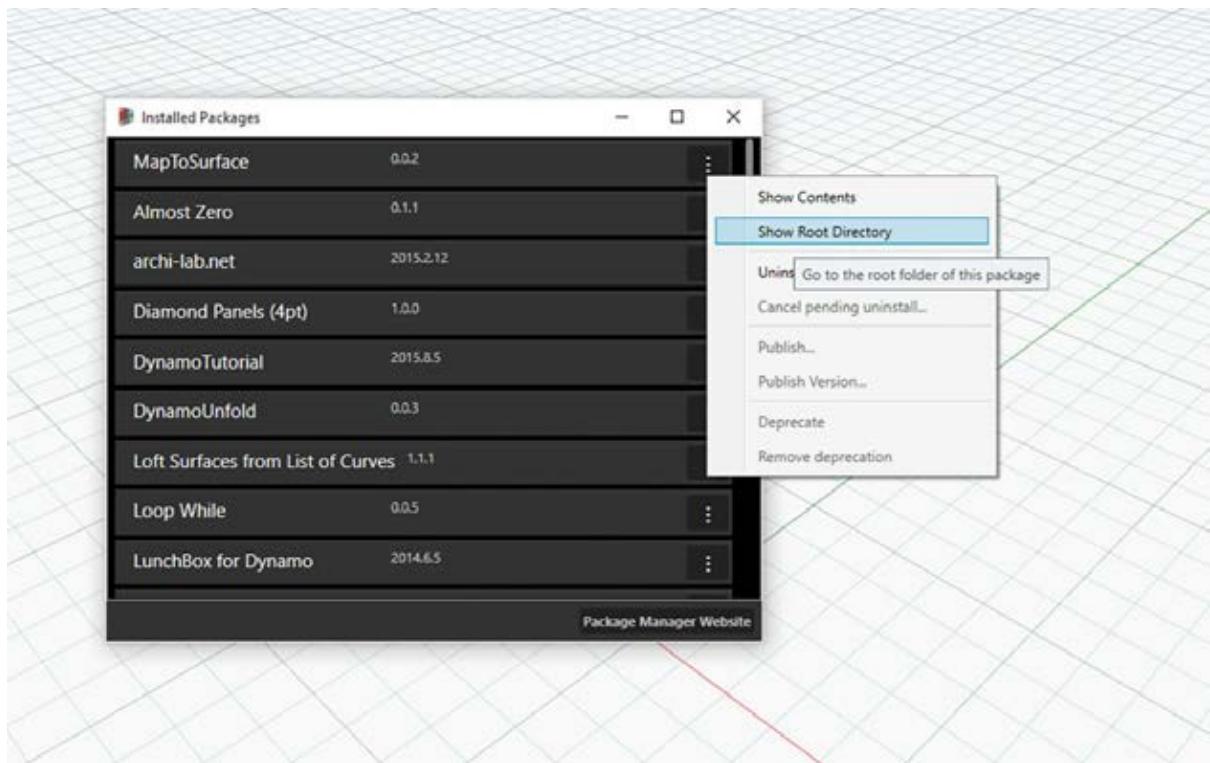
1. Durch Klicken auf *Datei hinzufügen* wurden außerdem die Dateien aus der Ordnerstruktur rechts auf dem Bildschirm hinzugefügt. (Um andere Dateien als DYF-Dateien hinzuzufügen, ändern Sie den Dateityp im Browserfenster in **Alle Dateien (.)**). Beachten Sie, dass sämtliche Dateien ohne Unterscheidung zwischen benutzerdefinierten Blöcken (.dyf) und Beispieldateien (.dyn) hinzugefügt wurden. Dynamo ordnet diese Objekte beim Veröffentlichen des Pakets in Kategorien ein.
2. Im Feld Gruppe wird definiert, in welcher Gruppe die benutzerdefinierten Blöcke in der Benutzeroberfläche von Dynamo abgelegt werden.
3. Veröffentlichen Sie das Paket, indem Sie auf Lokal publizieren klicken. Achten Sie darauf, auf *Lokal publizieren* und **nicht** auf *Online publizieren* zu klicken: Es sollen keine Duplikate im Package Manager erstellt werden.



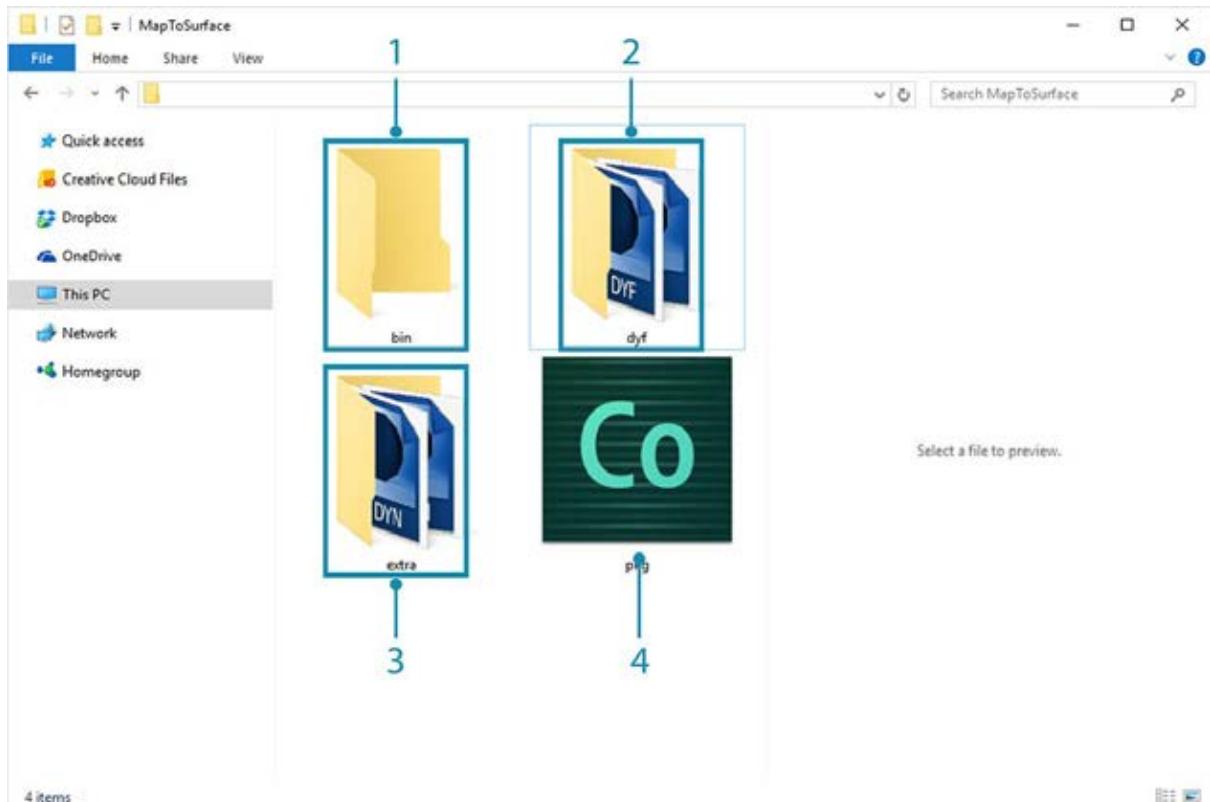
1. Nach der Veröffentlichung werden die benutzerdefinierten Blöcke in der Gruppe **DynamoPrimer** oder in Ihrer **Dynamo-Bibliothek** angezeigt.



Sehen Sie jetzt im Stammverzeichnis nach, wie **Dynamo** das eben erstellte Paket formatiert hat. Klicken Sie dazu auf **Pakete > Pakete verwalten**.



Klicken im Fenster zur Verwaltung von Paketen auf die drei senkrechten Punkte rechts neben *MapToSurface* und wählen Sie *Stammverzeichnis anzeigen*.

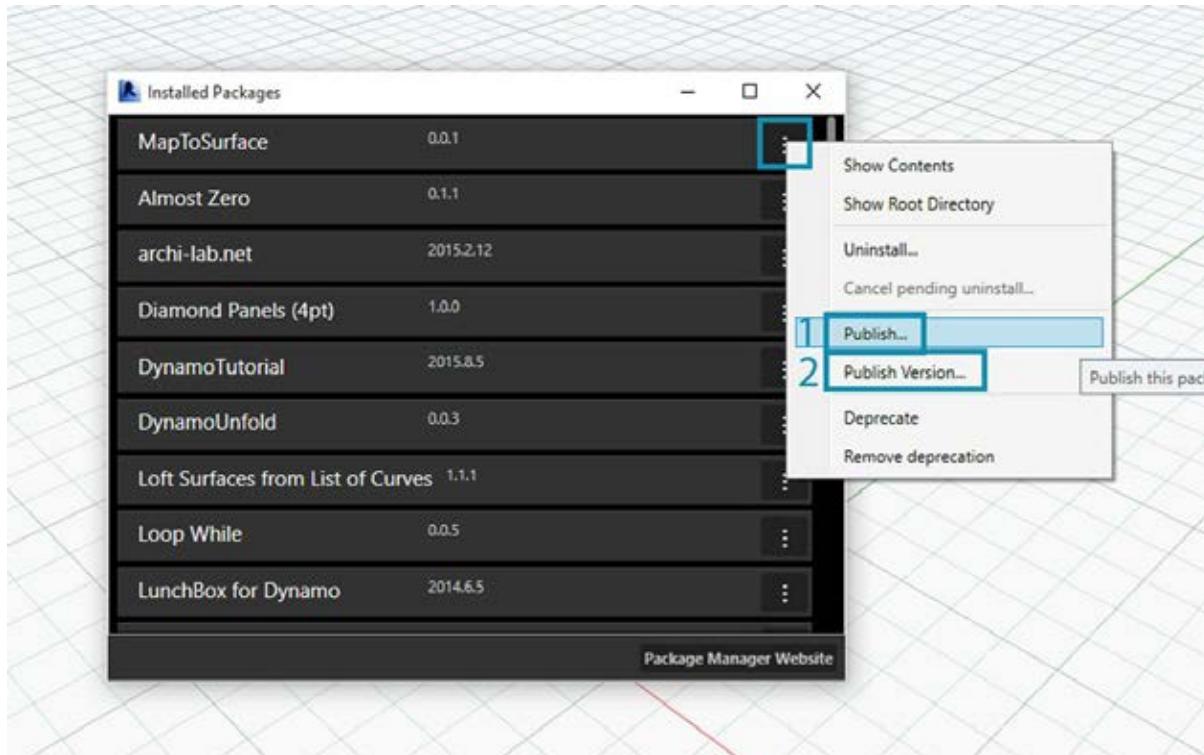


Das Stammverzeichnis befindet sich am lokalen Speicherort des Pakets (da Sie es lokal veröffentlicht haben). Dynamo greift derzeit zum Lesen benutzerdefinierter Blöcke auf diesen Ordner zu. Aus diesem Grund müssen Sie das Verzeichnis lokal an einem dauerhaften Speicherort ablegen (d. h. nicht auf Ihrem Desktop). Der Ordner mit dem Dynamo-Paket ist wie folgt gegliedert:

1. Im Ordner *bin* befinden sich DLL-Dateien, die mit C#- oder Zero Touch-Bibliotheken erstellt wurden. Dieses

- Paket enthält keine solchen Dateien; dieser Ordner ist also in diesem Beispiel leer.
2. Im Ordner `dyf` befinden sich die benutzerdefinierten Blöcke. Wenn Sie ihn öffnen, werden alle benutzerdefinierten Blöcke (DYF-Dateien) für das Paket angezeigt.
  3. Im Ordner `extra` befinden sich alle zusätzlichen Dateien. Dies sind wahrscheinlich Dynamo-Dateien (`.dyn`) oder sonstige erforderliche Zusatzdateien (`.svg`, `.xls`, `.jpeg`, `.sat` usw.).
  4. Die Datei `pkg` ist eine einfache Textdatei, die die Einstellungen des Pakets definiert. Diese Datei wird in Dynamo automatisch erstellt, Sie können Sie jedoch bearbeiten, wenn Sie detaillierte Einstellungen benötigen.

## Online-Veröffentlichung von Paketen



**Anmerkung:** Führen Sie diesen Schritt bitte nicht aus, es sei denn, Sie möchten tatsächlich ein eigenes Paket veröffentlichen.

1. Wenn das Paket zur Veröffentlichung bereit ist, wählen Sie im Fenster Pakete verwalten die Schaltfläche rechts von `MapToSurface` und wählen Sie *Veröffentlichen*.
2. Um ein bereits veröffentlichtes Paket zu aktualisieren, wählen Sie *Version veröffentlichen*. Dynamo aktualisiert dann Ihr Paket online mit den neuen Dateien im Stammverzeichnis des Pakets. Dieser einfache Schritt genügt.

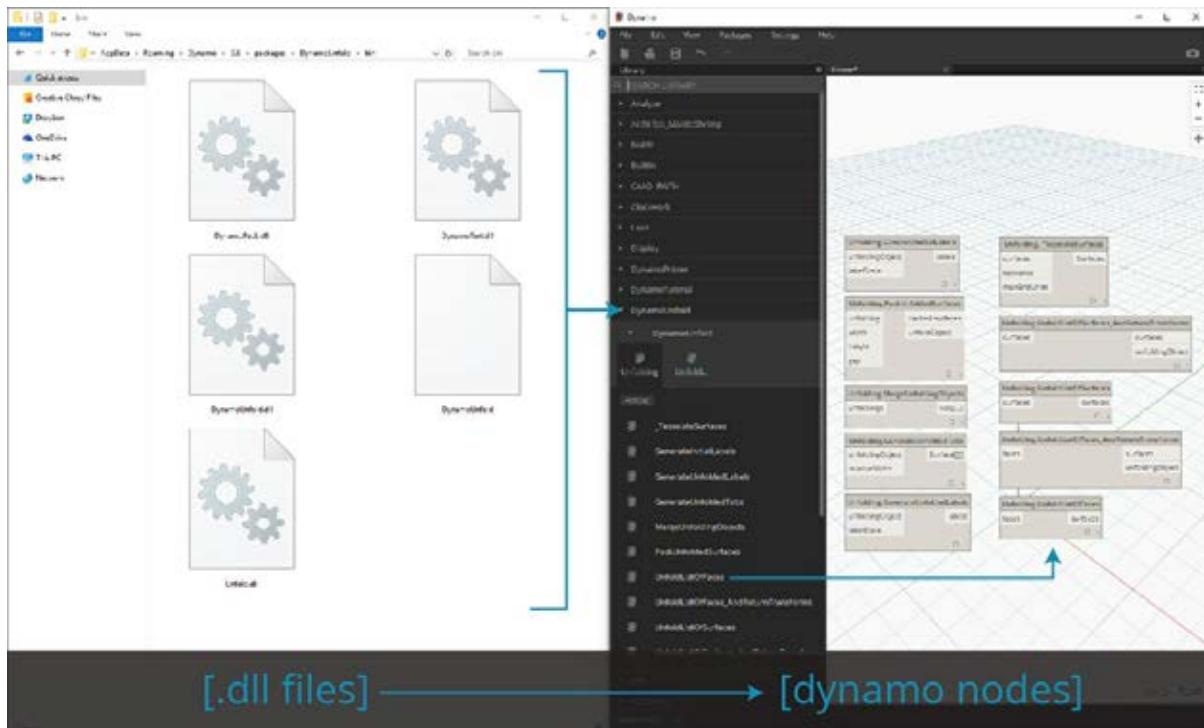
## Version veröffentlichen

Wenn Sie die Dateien im Stammverzeichnis des veröffentlichten Pakets aktualisieren, können Sie über *Version veröffentlichen* im Fenster *Pakete verwalten* eine neue Version des Pakets veröffentlichen. Auf diese Weise können Sie nahtlos erforderliche Aktualisierungen Ihrer Inhalte vornehmen und sie für die Community bereitstellen. Die Funktion *Version veröffentlichen* kann nur verwendet werden, wenn Sie der Verwalter des Pakets sind.

# Was ist Zero-Touch?

## Was ist Zero-Touch?

Unter Zero-Touch-Import versteht man ein einfaches Verfahren zum Importieren von C#-Bibliotheken durch Zeigen und Klicken. Dynamo liest die öffentlichen Methoden einer *DLL*-Datei und konvertiert sie in Dynamo-Blöcke. Sie können mithilfe von Zero-Touch Ihre eigenen benutzerdefinierten Blöcke und Pakete entwickeln sowie externe Bibliotheken in die Dynamo-Umgebung importieren.



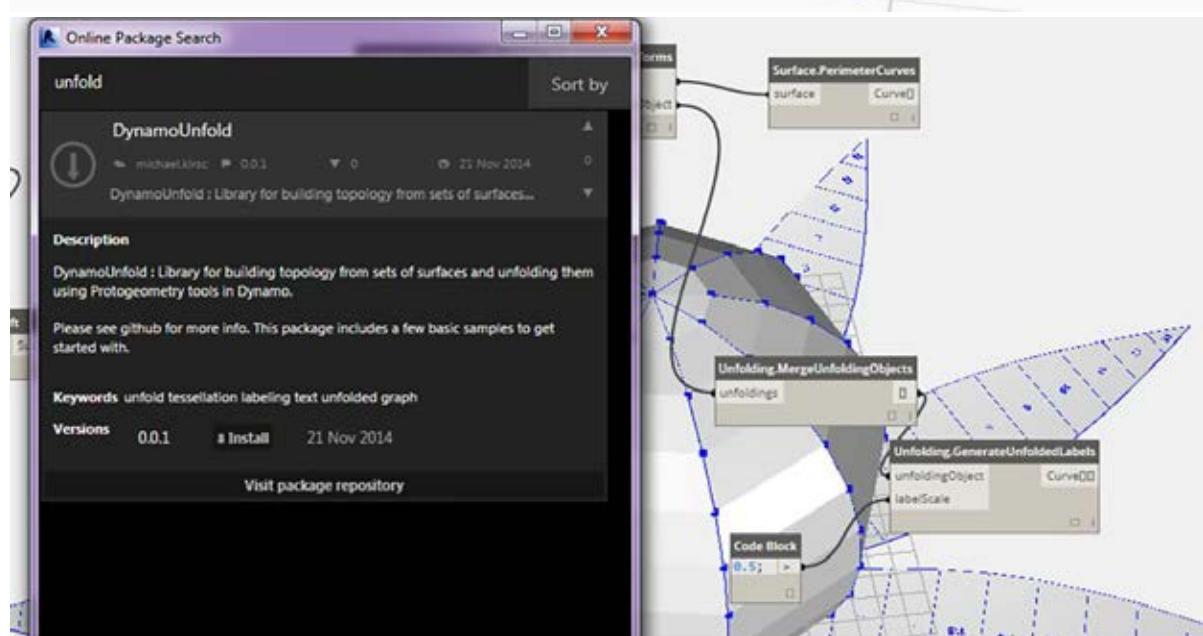
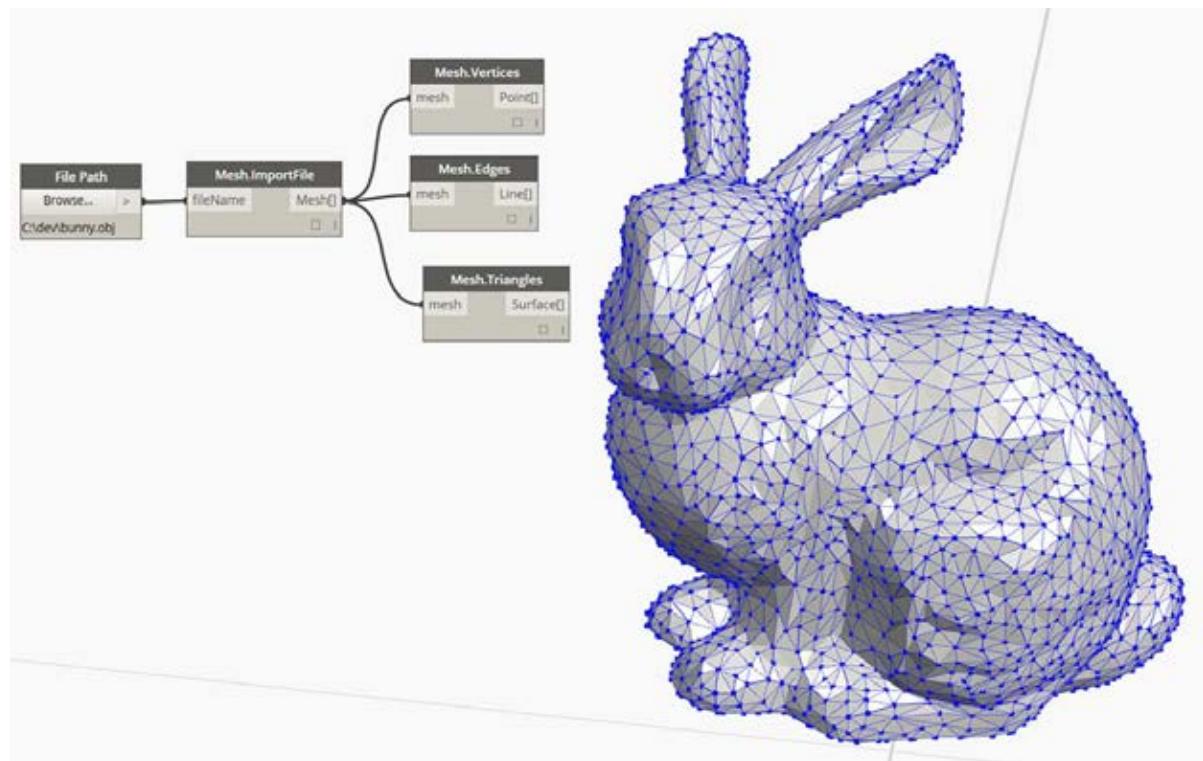
Mit Zero-Touch können Sie Bibliotheken importieren, die nicht unbedingt für Dynamo entwickelt wurden, und Suites mit neuen Blöcken erstellen. Die aktuelle Zero-Touch-Funktion zeigt das plattformübergreifende Konzept des Dynamo-Projekts.

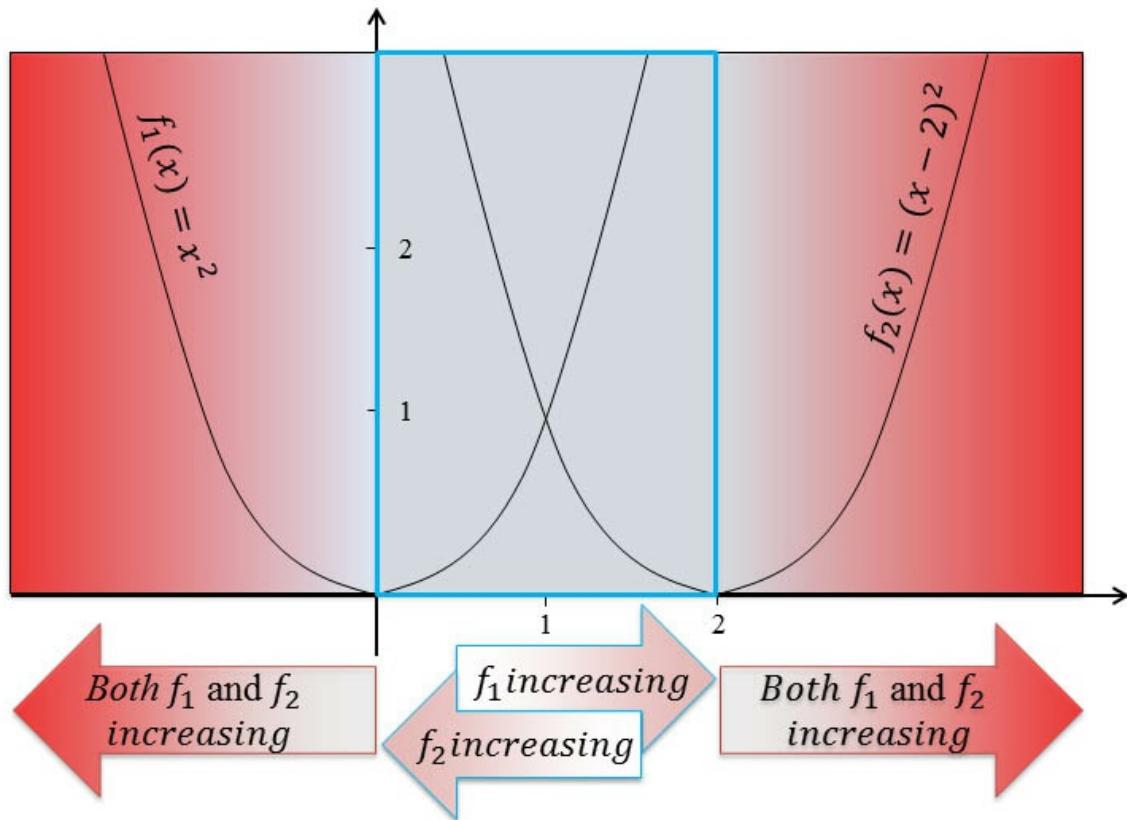
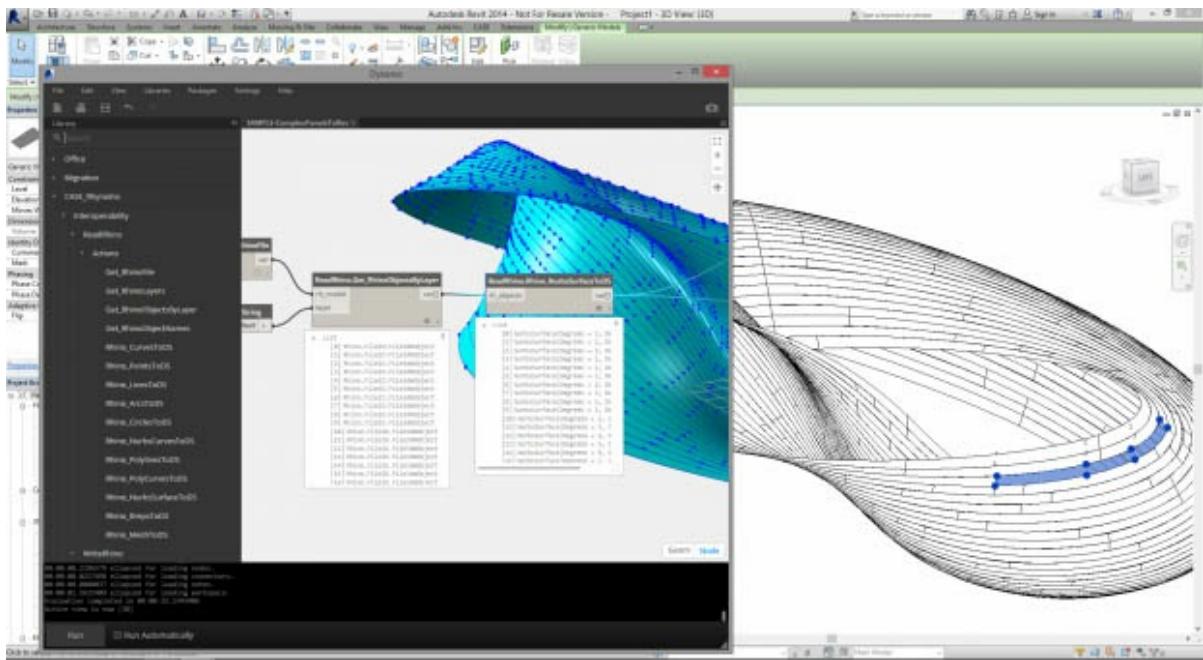
In diesem Abschnitt wird gezeigt, wie Sie mithilfe von Zero-Touch externe Bibliotheken importieren können. Informationen zum Entwickeln eigener Zero-Touch-Bibliotheken finden Sie auf der [Dynamo-Wiki-Seite](#).

## Zero-Touch-Pakete

Zero-Touch-Pakete sind eine gute Ergänzung für benutzerdefinierte Blöcke. In der folgenden Tabelle sind einige Pakete angegeben, in denen C#-Bibliotheken verwendet werden. Genaue Informationen über die Pakete finden Sie im [Abschnitt zu Paketen](#) im Anhang.

### Logo/Abbildung





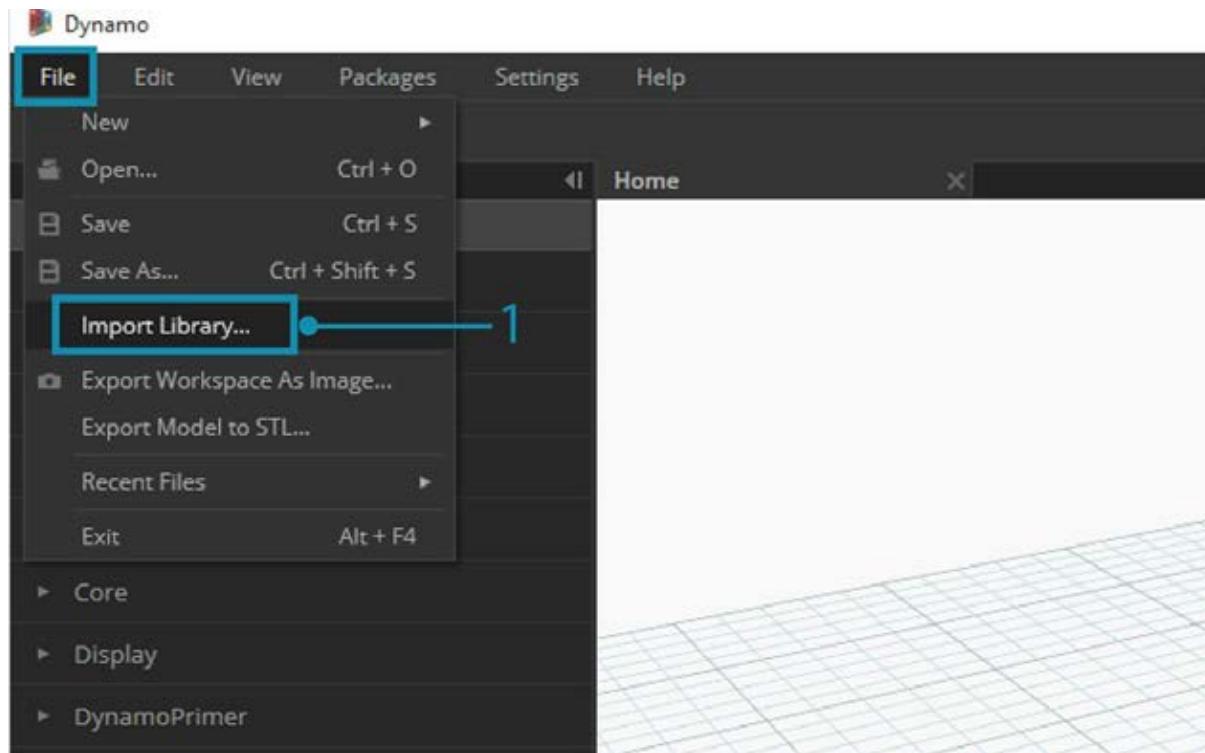
### Fallstudie – Importieren von AForge

In dieser Fallstudie wird der Import der externen [AForge-DLL-Bibliothek](#) gezeigt. AForge ist eine zuverlässige Bibliothek mit einem breiten Spektrum von Funktionen, angefangen mit der Bildbearbeitung und bis hin zu künstlicher Intelligenz. Hier referenzieren Sie die Bildverarbeitungsklasse von AForge für die im weiteren Verlauf dieses Abschnitts folgenden Übungen zur Bildverarbeitung.

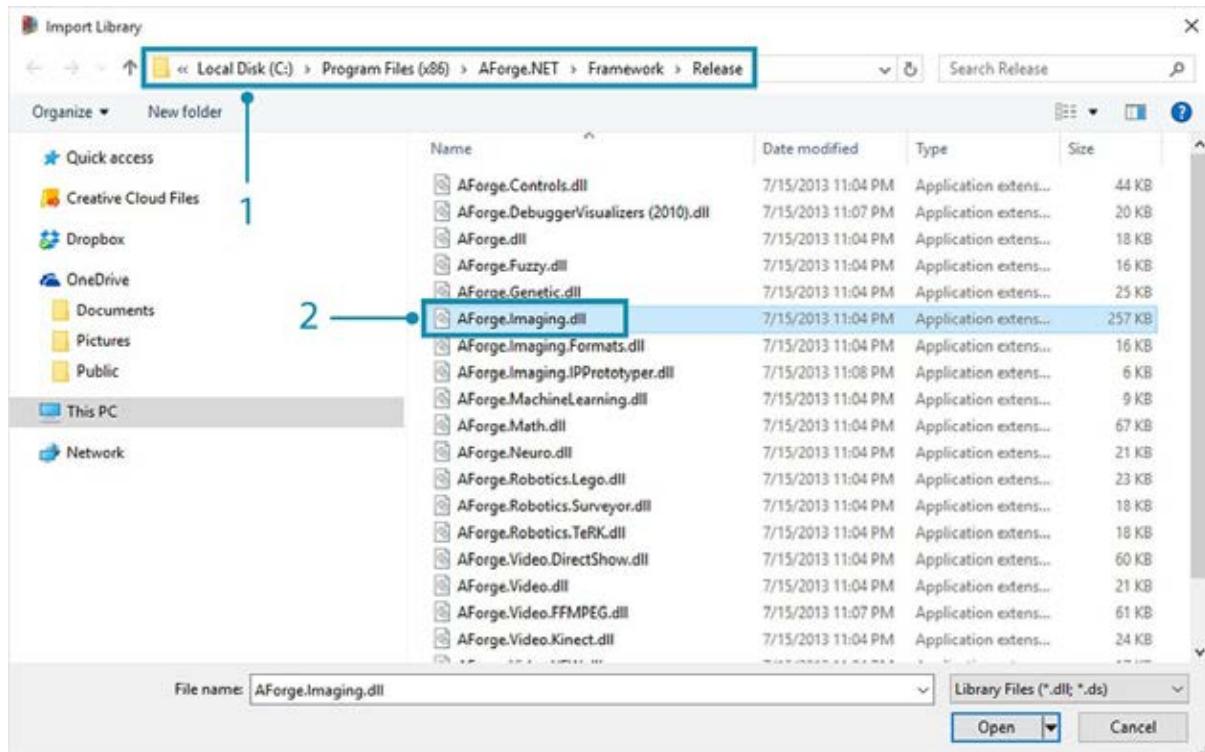
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Zero-Touch-Examples.zip](#).

1. Beginnen Sie, indem Sie AForge herunterladen. Wählen Sie auf der [Download-Seite von AForge](#) die Option

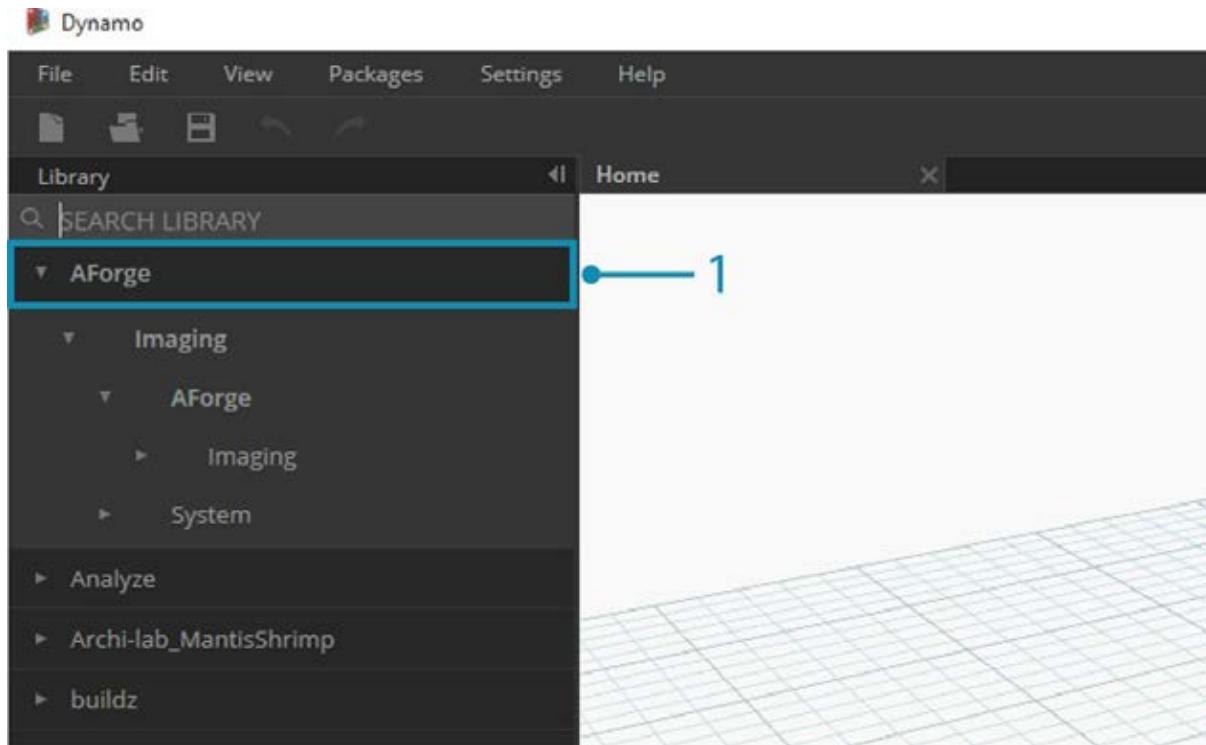
[Download Installer]. Wenn der Download abgeschlossen ist, installieren Sie das Programm.



1. Erstellen Sie in Dynamo eine neue Datei und wählen Sie *Datei > Bibliothek importieren*.



1. Navigieren Sie im Popup-Fenster zum Release-Ordner in Ihrer AForge-Installation. Dies ist wahrscheinlich ein Ordner wie der folgende: C:\Program Files (x86)\AForge.NET\Framework\Release.
2. **AForge.Imaging.dll:** Für diese Fallstudie benötigen Sie nur diese eine Datei aus der AForge-Bibliothek. Wählen Sie diese DLL-Datei aus und klicken Sie auf Öffnen.



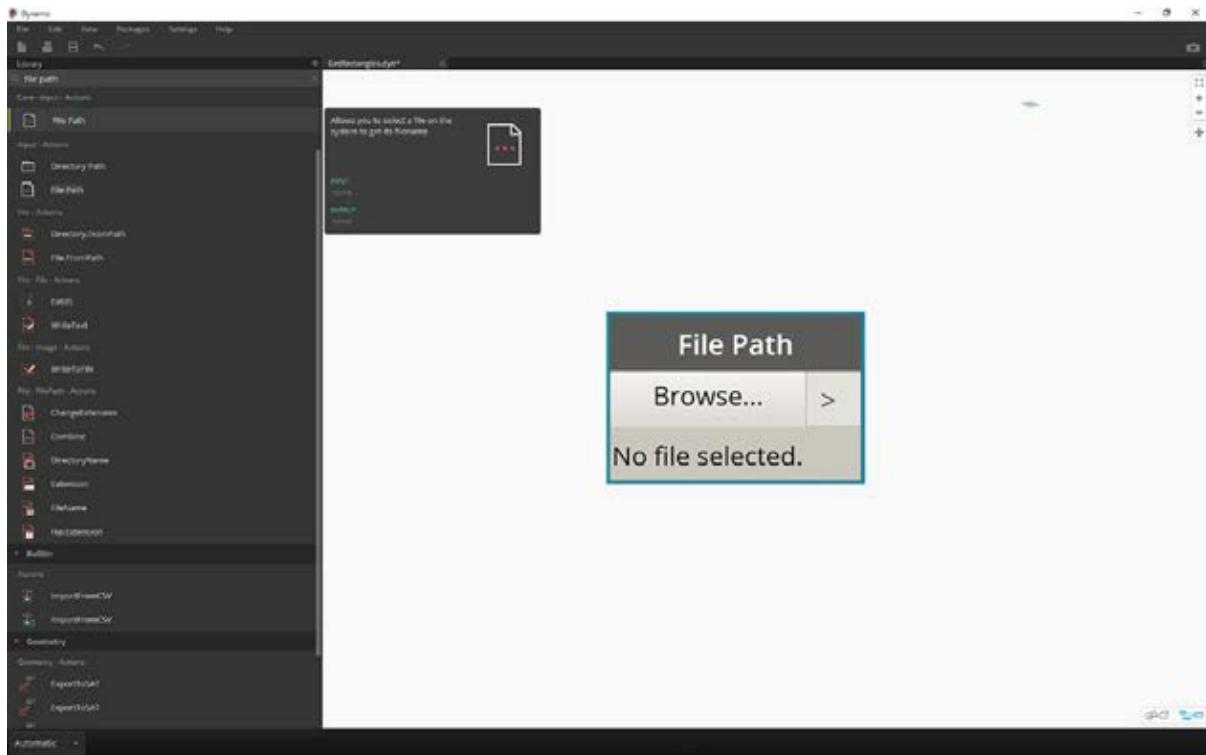
1. In Dynamo sollte im Werkzeugkasten der Bibliothek jetzt eine Gruppe von Blöcken namens *AForge* hinzugekommen sein. Damit haben Sie innerhalb des visuellen Programms Zugriff auf die AForge-Bildbearbeitungsbibliothek.

## Übung 1 – Kantenerkennung

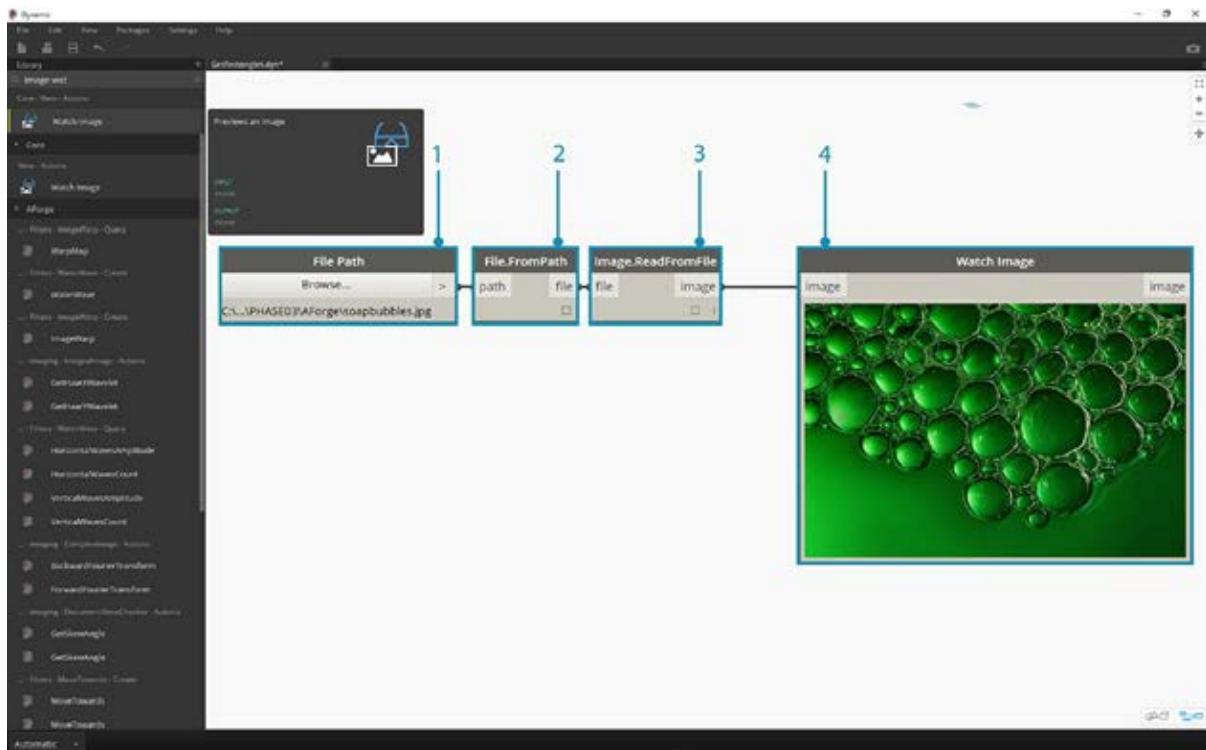
Nachdem Sie die Bibliothek importiert haben, beginnen Sie mit der ersten einfachen Übung. Sie führen einige einfache Bildverarbeitungsvorgänge an einem Beispielbild durch, um die Funktionsweise der Bildfilter in AForge zu demonstrieren. Dabei zeigen Sie die Ergebnisse in einem *Watch Image*-Block an und wenden in Dynamo Filter an, die denen von Photoshop ähnlich sind.

Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [ZeroTouchImages.zip](#)

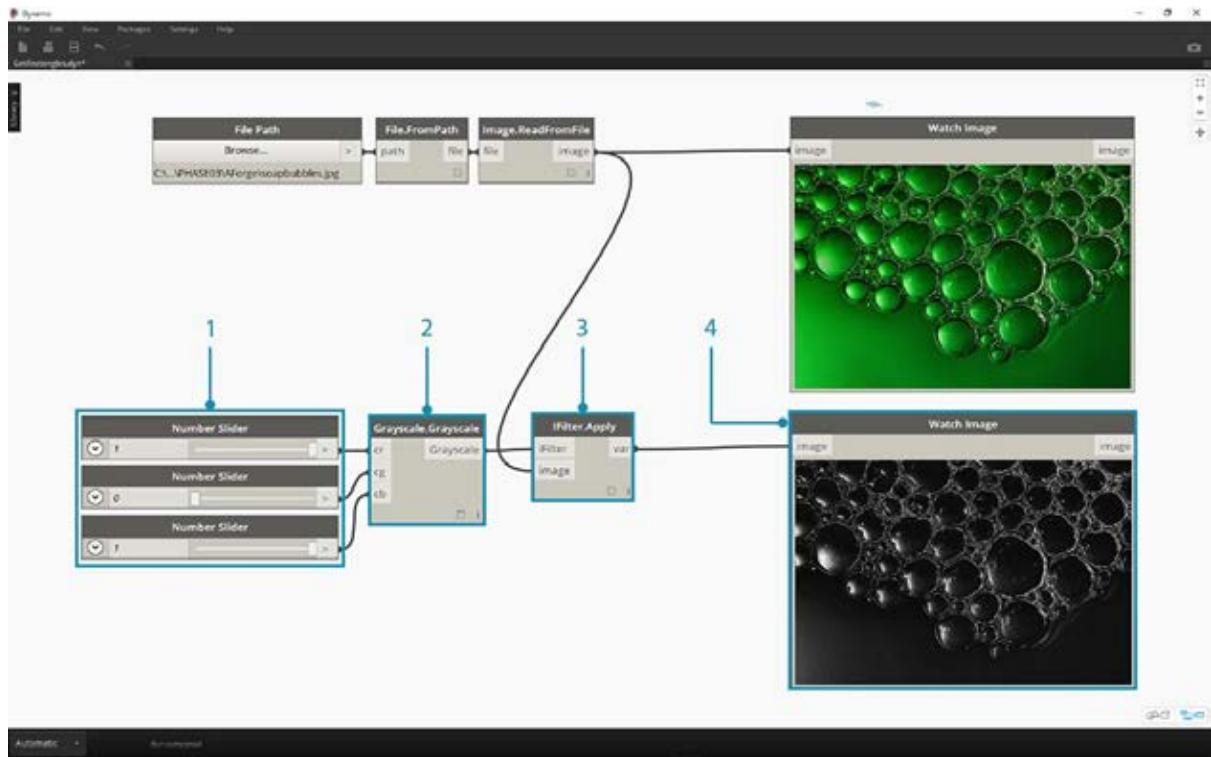
Nachdem Sie die Bibliothek importiert haben, beginnen Sie mit der ersten einfachen Übung (*01-EdgeDetection.dyn*). Sie führen einige einfache Bildverarbeitungsvorgänge an einem Beispielbild durch, um die Funktionsweise der Bildfilter in AForge zu demonstrieren. Dabei zeigen Sie die Ergebnisse in einem *Watch Image*-Block an und wenden in Dynamo Filter an, die denen von Photoshop ähnlich sind.



Als Erstes importieren Sie das Bild, mit dem Sie arbeiten möchten. Fügen Sie einen *File Path*-Block im Ansichtsbereich hinzu und wählen Sie die Datei "soapbubbles.jpg" aus dem heruntergeladenen Übungsordner (Foto: [flickr](#)).

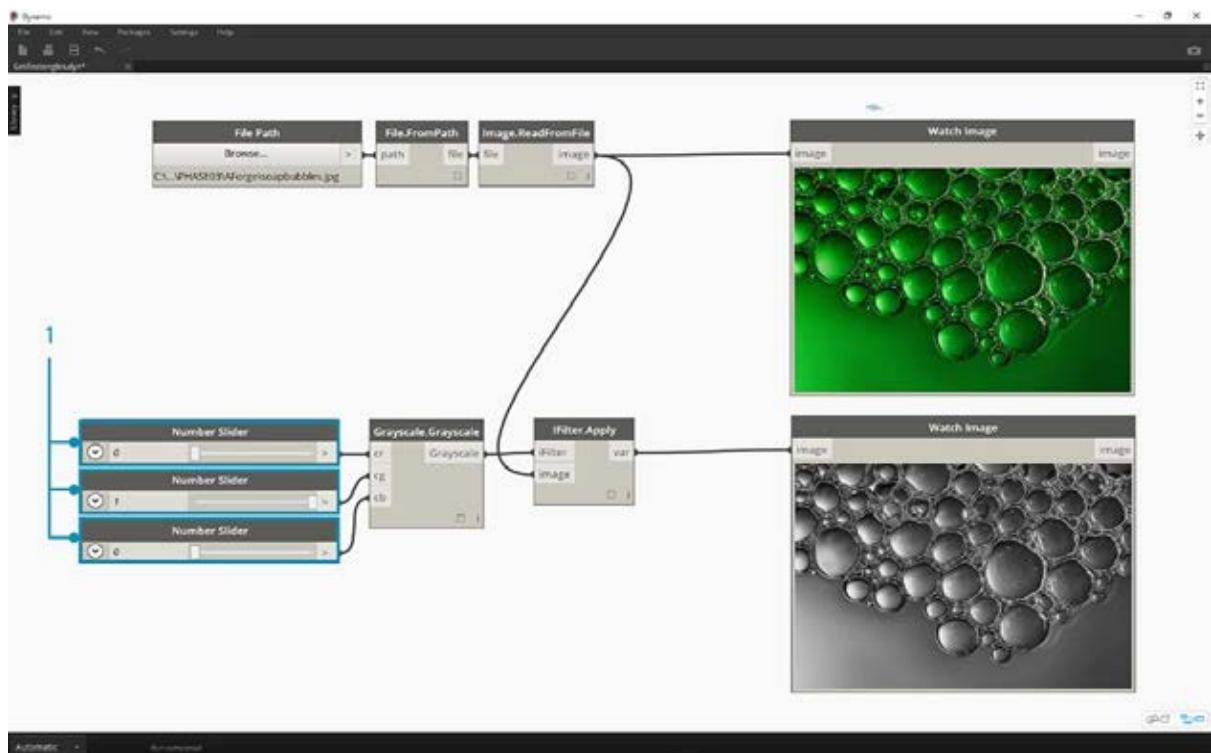


1. Im File Path-Block wird lediglich der Pfad zum ausgewählten Bild als Zeichenfolge angegeben. Diesen Pfad müssen Sie in ein Bild in der Dynamo-Umgebung konvertieren.
2. Verbinden Sie den File Path-Block mit einem File.FromPath-Block.
3. Um diese Datei in ein Bild zu konvertieren, verwenden Sie den Image.ReadFromFile-Block.
4. Schließlich zeigen Sie das Ergebnis an. Fügen Sie einen Watch Image-Block in den Ansichtsbereich ein und verbinden Sie ihn mit dem Image.ReadFromFile-Block. Sie haben AForge noch nicht verwendet, aber das Bild erfolgreich in Dynamo importiert.



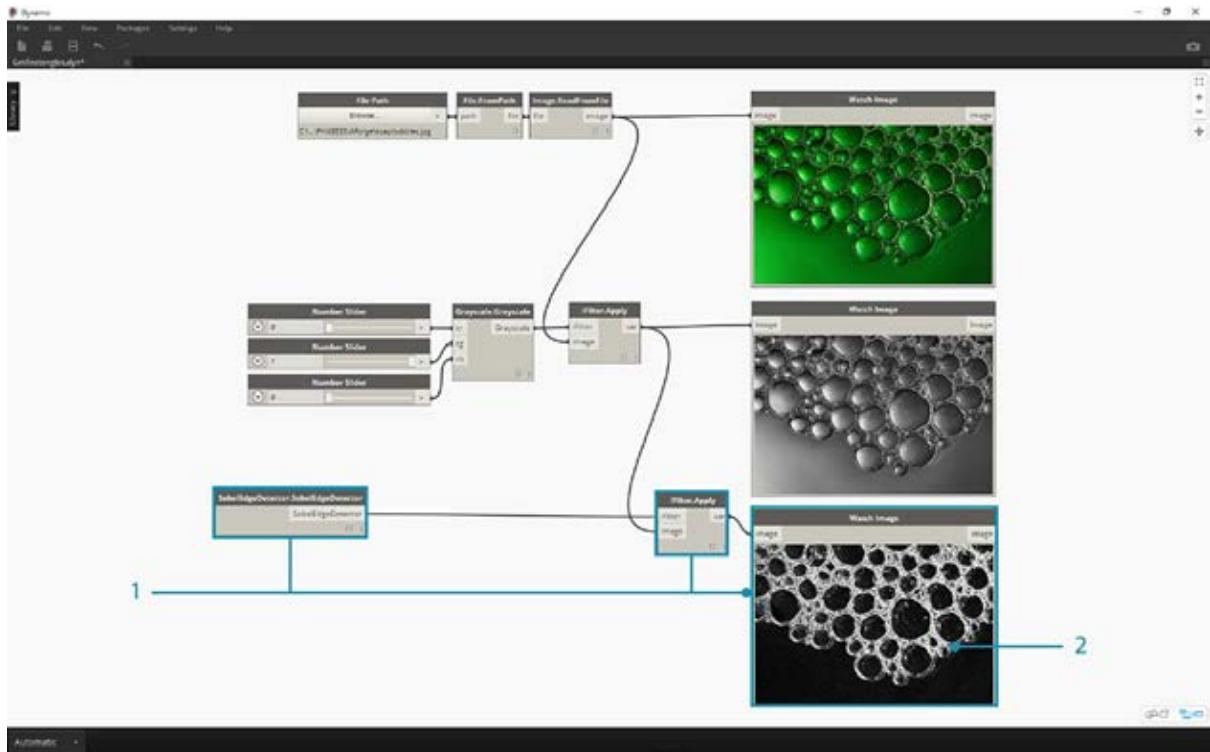
Unter AForge.Imaging.AForge.Filters (im Navigationsmenü) stehen zahlreiche Filter zur Verfügung. Sie reduzieren mithilfe eines dieser Filter die Farbsättigung des Bildes anhand von Schwellenwerten.

1. Fügen Sie drei Schiebereglern in den Ansichtsbereich ein und ändern Sie ihre Bereiche in 0 bis 1 und ihre Schrittweite in 0,01.
2. Fügen Sie den Grayscale.Grayscale-Block in den Ansichtsbereich ein. Dies ist ein AForge-Filter, der einen Graustufenfilter auf das Bild anwendet. Verbinden Sie die drei Schiebereglern aus Schritt 1 mit cr, cg und cb. Legen Sie im oberen und unteren Schiebereglern jeweils den Wert 1 und im mittleren den Wert 0 fest.
3. Damit der Graustufenfilter angewendet wird, benötigen Sie eine Aktion für das Bild. Verwenden Sie hierfür IFilter.Apply. Verbinden Sie das Bild mit der image-Eingabe und Grayscale.Grayscale mit der IFilter-Eingabe.
4. Wenn Sie hier einen Watch Image-Block verbinden, erhalten Sie ein entsättigtes Bild.



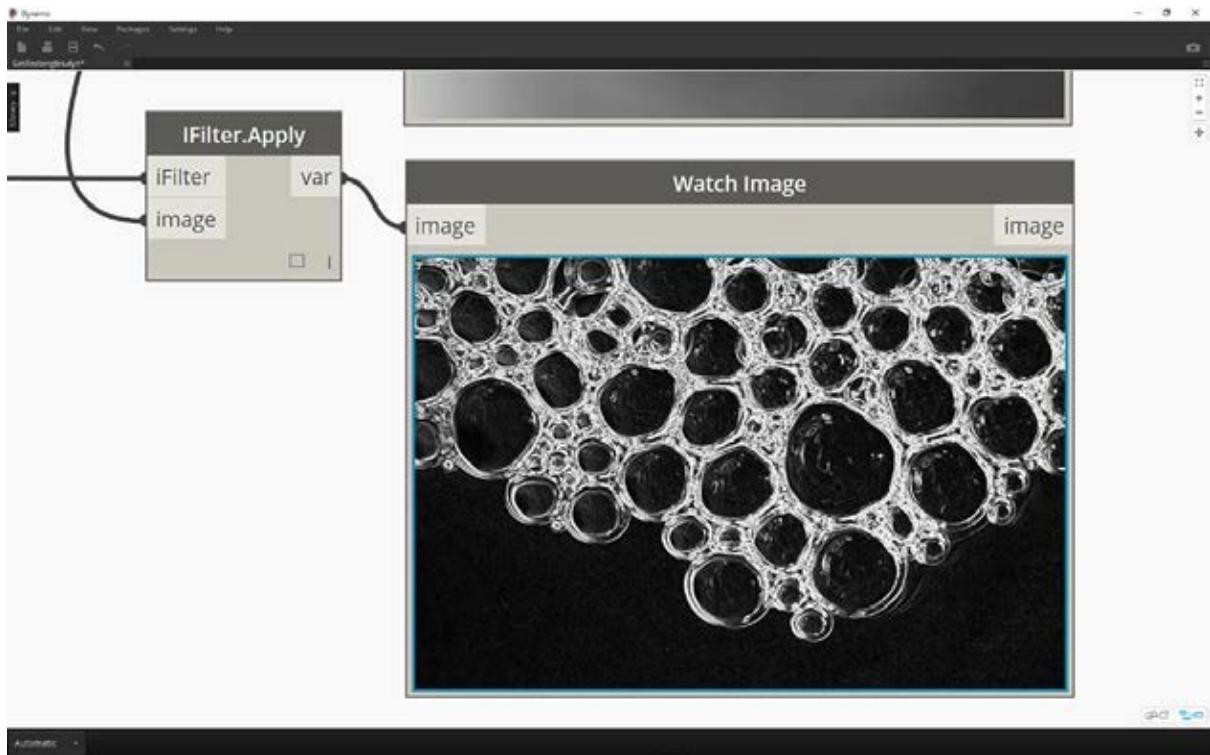
Sie können mithilfe von Schwellenwerten für Rot, Grün und Blau steuern, wie das Bild entsättigt werden soll. Diese Werte werden über die Eingaben des Grayscale.Grayscale-Blocks definiert. Das Bild wirkt recht dunkel. Der Grund dafür ist, dass im Schieberegler für den Grün-Wert auf 0 eingestellt ist.

1. Legen Sie im oberen und unteren Schieberegler jeweils den Wert 0 und im mittleren den Wert 1 fest. Auf diese Weise erhalten Sie deutlicheres entsättigtes Bild.



Als Nächstes wenden Sie einen zusätzlichen Filter auf das entsättigte Bild an. Das entsättigte Bild weist einen gewissen Kontrast auf. Testen Sie daher jetzt die Kantenerkennung.

1. Fügen Sie im Ansichtsbereich einen SobelEdgeDetector.SobelEdgeDetector-Block hinzu. Verbinden Sie diesen als IFilter mit einem neuen IFilter-Block und das entsättigte Bild mit der image-Eingabe des IFilter-Blocks.
2. Die Sobel-Kantenerkennung hebt in einem neuen Bild die Kanten hervor.

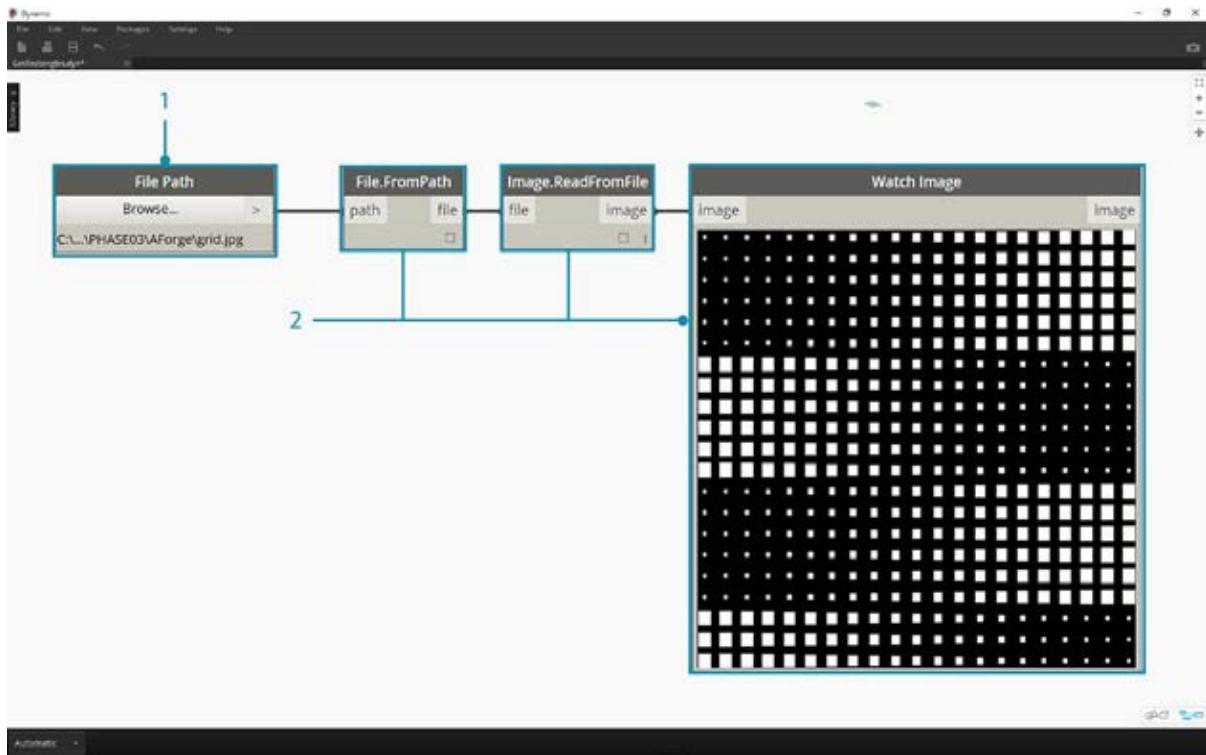


Die vergrößerte Darstellung zeigt, dass die Kantenerkennung die Umrisse der Blasen mit Pixeln markiert. In der AForge-Bibliothek stehen Werkzeuge zur Verfügung, mit denen Sie aus Ergebnissen wie diesem Dynamo-Geometrie erstellen können. Dies wird in der nächsten Übung genauer betrachtet.

## Übung 2 – Rechtecke erstellen

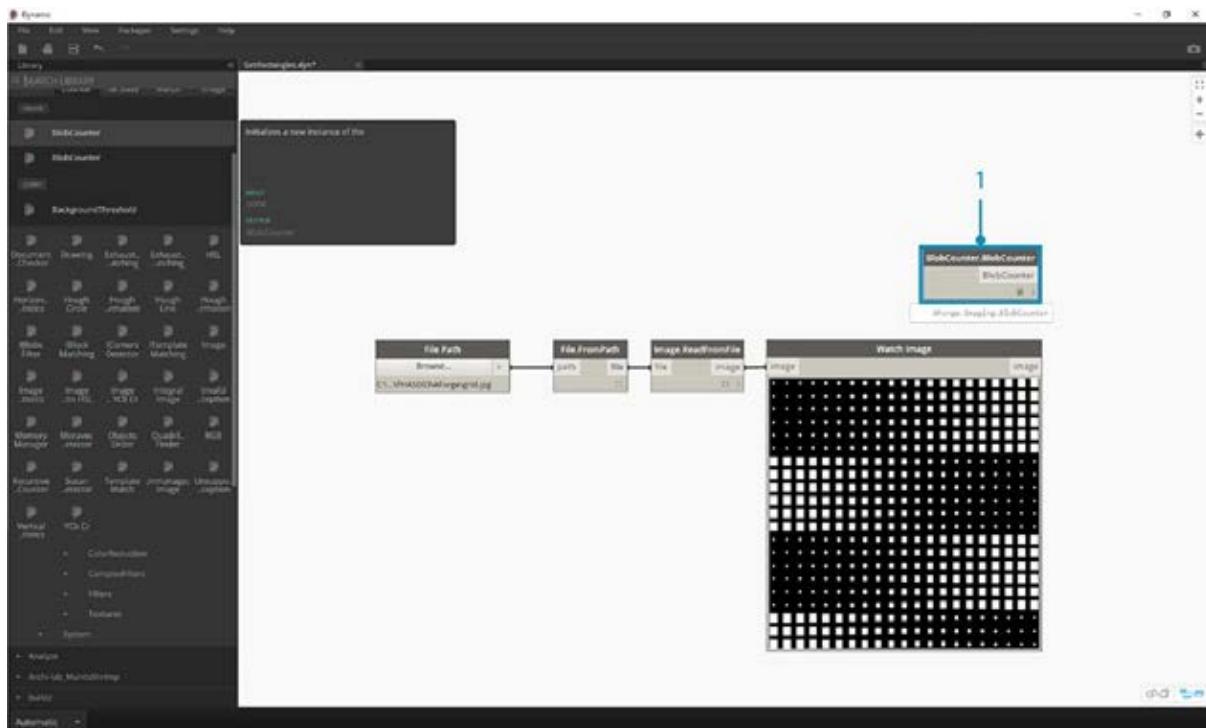
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option "Save Link As..."). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [ZeroTouchImages.zip](#)

Nach dieser Einführung in die einfache Bildverarbeitung wird hier gezeigt, wie Sie ein Bild dazu verwenden können, um Dynamo-Geometrie zu steuern. In dieser Übung führen Sie mithilfe von AForge und Dynamo einen einfachen *Live Trace*-Vorgang für ein Bild durch. Dieses Beispiel ist relativ einfach: Aus einem Referenzbild werden Rechtecke extrahiert. In AForge stehen jedoch auch Werkzeuge für komplexere Operationen zur Verfügung. Sie arbeiten mit der Datei *02-RectangleCreation.dyn* aus den heruntergeladenen Übungsdateien.

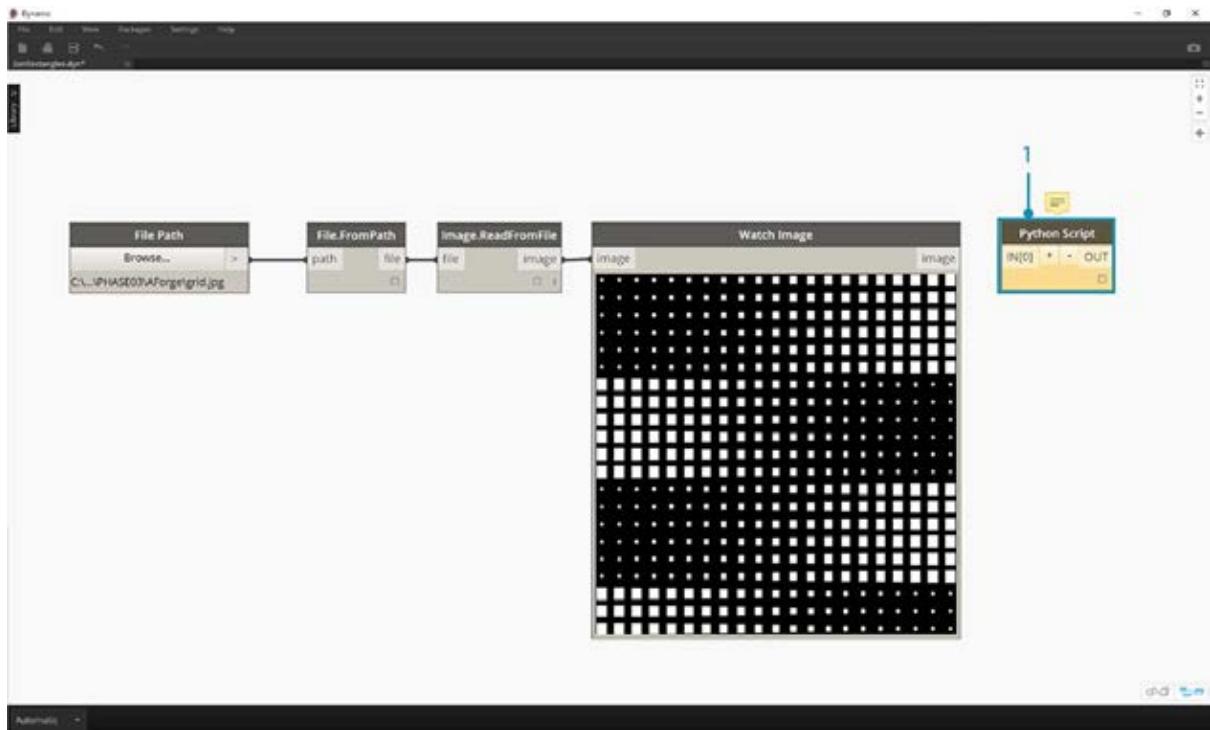


1. Navigieren Sie im File Path-Block zu grid.jpg im Übungsordner.
2. Verbinden Sie die übrigen Blöcke der oben gezeigten Folge, um ein durch Verlaufsparameter definiertes Raster anzuseigen.

Im nächsten Schritt sollen die weißen Quadrate in diesem Bild referenziert und in Dynamo-Geometrie konvertiert werden. AForge bietet eine Vielfalt leistungsstarker Computer Vision-Werkzeuge. Hier verwenden Sie ein besonders wichtiges Werkzeug in der Bibliothek: [BlobCounter](#).



1. Nachdem Sie einen BlobCounter-Block im Ansichtsbereich hinzugefügt haben, benötigen Sie eine Funktion zur Verarbeitung des Bildes (ähnlich dem IFilter-Werkzeug in der vorigen Übung). Der Process Image-Block ist jedoch nicht direkt in der Dynamo-Bibliothek sichtbar. Der Grund hierfür ist, dass die Funktion eventuell nicht im AForge-Quellcode sichtbar ist. Dies müssen Sie mit einer Umgehungslösung beheben.

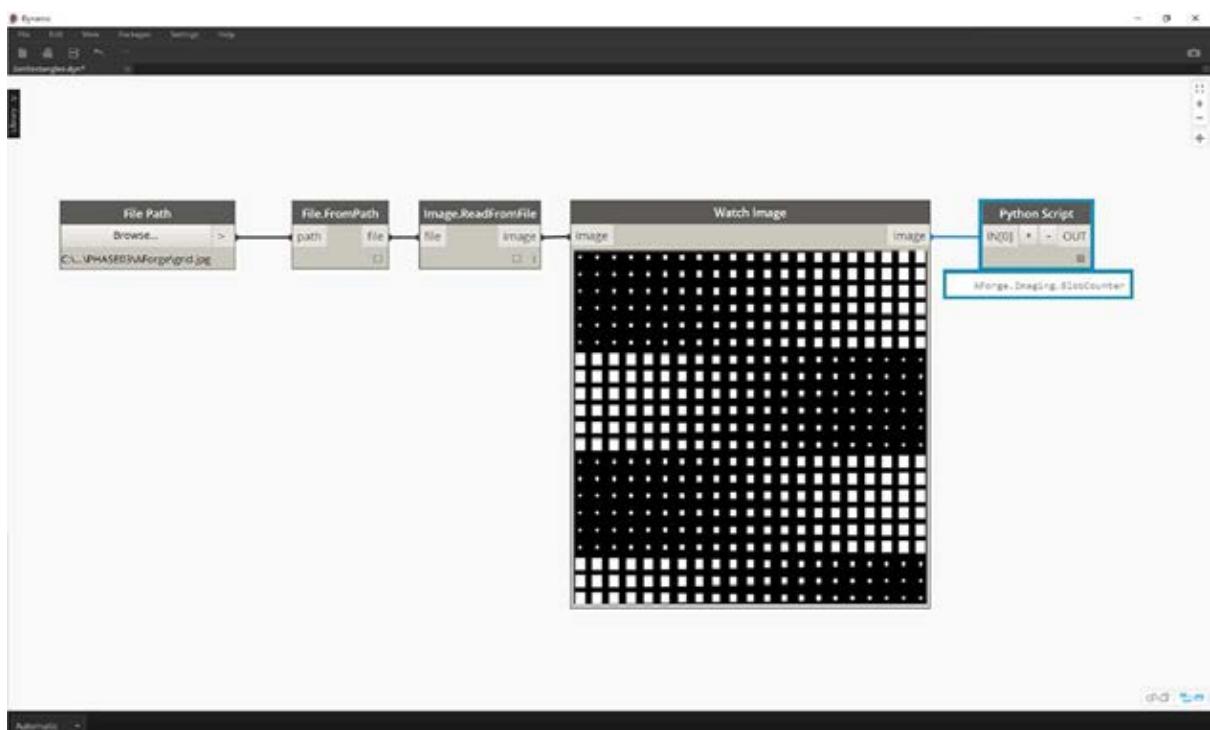


1. Fügen Sie im Ansichtsbereich einen Python-Block hinzu.

```
import clr
clr.AddReference('AForge.Imaging')
from AForge.Imaging import *

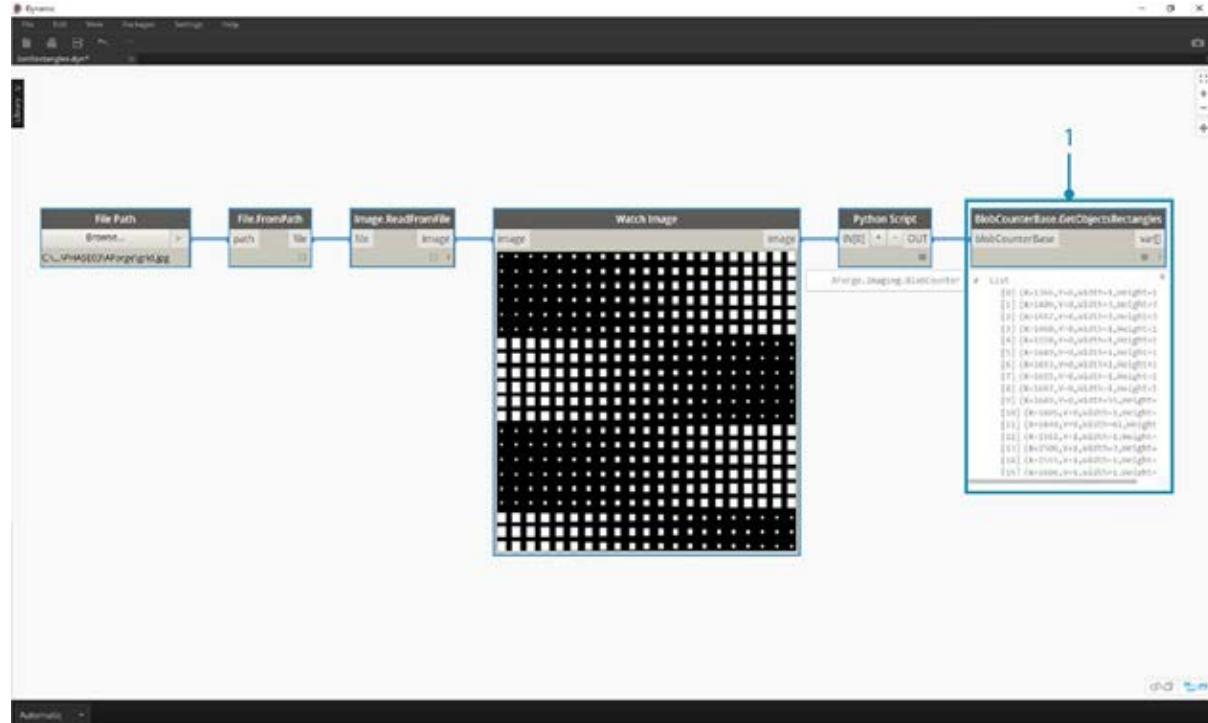
bc= BlobCounter()
bc.ProcessImage(IN[0])
OUT=bc
```

Fügen Sie den oben gezeigten Code in den Python-Block ein. Dieser Code importiert die AForge-Bibliothek und verarbeitet dann das importierte Bild.

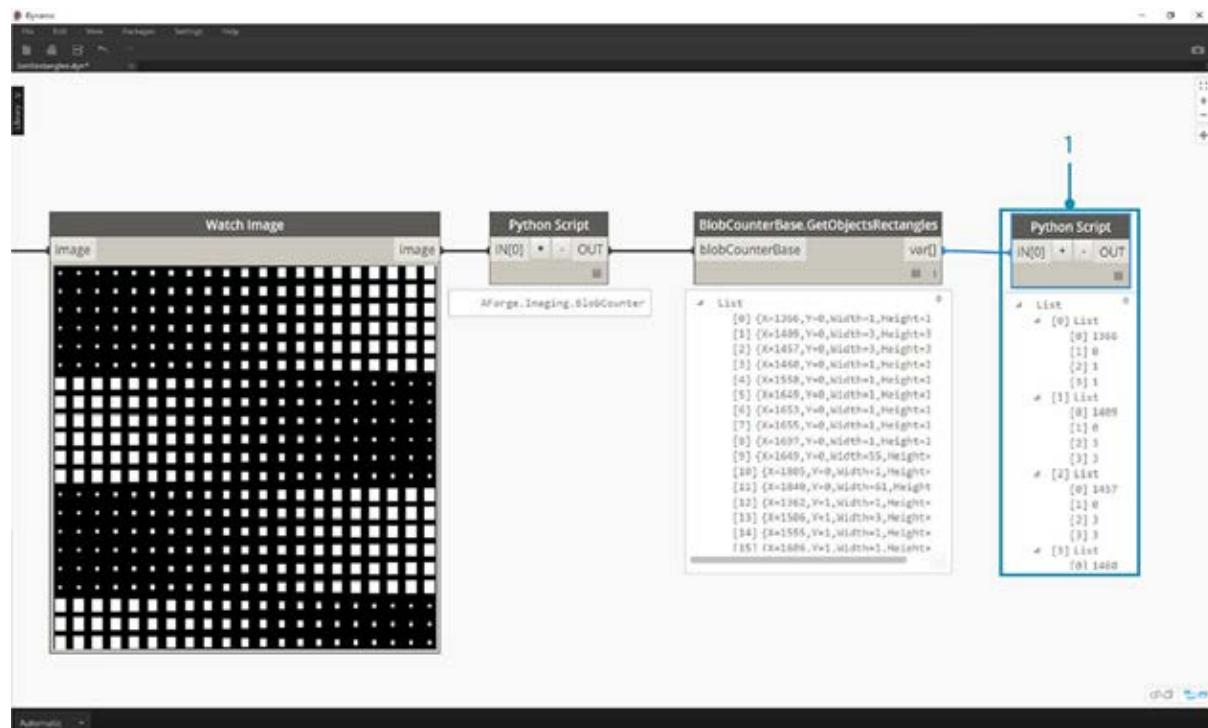


Indem Sie die image-Ausgabe mit der Eingabe des Python-Blocks verbinden, erhalten Sie ein AForge.Imaging.BlobCounter-Ergebnis aus dem Python-Block.

Die Vorgänge in den nächsten Schritten setzen eine gewisse Kenntnis der [AForge Imaging API](#) voraus. Für die Arbeit mit Dynamo müssen Sie diese nicht komplett erlernen. Dies dient mehr zur Demonstration der Arbeit mit externen Bibliotheken innerhalb der Dynamo-Umgebung mit ihrer großen Flexibilität.



1. Verbinden Sie die Ausgabe des Python-Skripts mit BlobCounterBase.GetObjectRectangles. Dieser Block liest Objekte in einem Bild anhand eines Schwellenwerts und extrahiert quantifizierte Rechtecke aus dem Pixelraum.

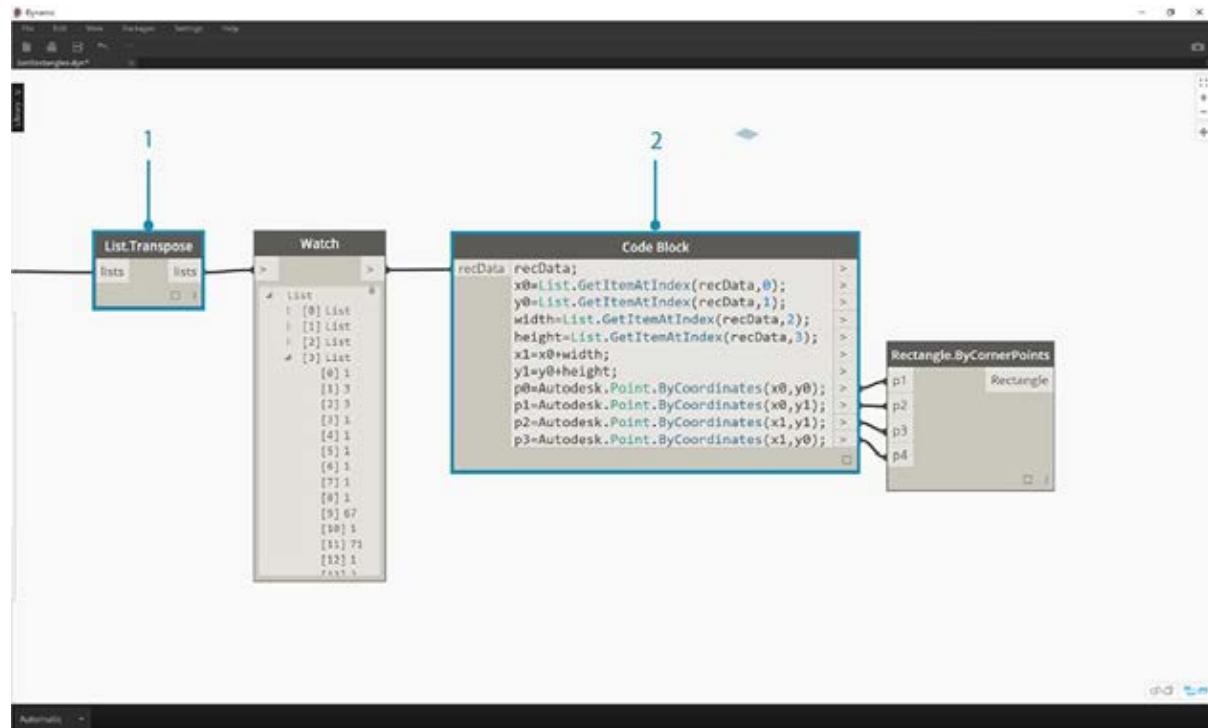


1. Fügen Sie einen weiteren Python-Block in den Ansichtsbereich ein, verbinden Sie ihn mit GetObjectRectangles und geben Sie den unten stehenden Code ein. Dadurch erhalten Sie eine strukturierte Liste von Dynamo-Objekten.

```

OUT = []
for rec in IN[0]:
    subOUT=[]
    subOUT.append(rec.X)
    subOUT.append(rec.Y)
    subOUT.append(rec.Width)
    subOUT.append(rec.Height)
    OUT.append(subOUT)

```

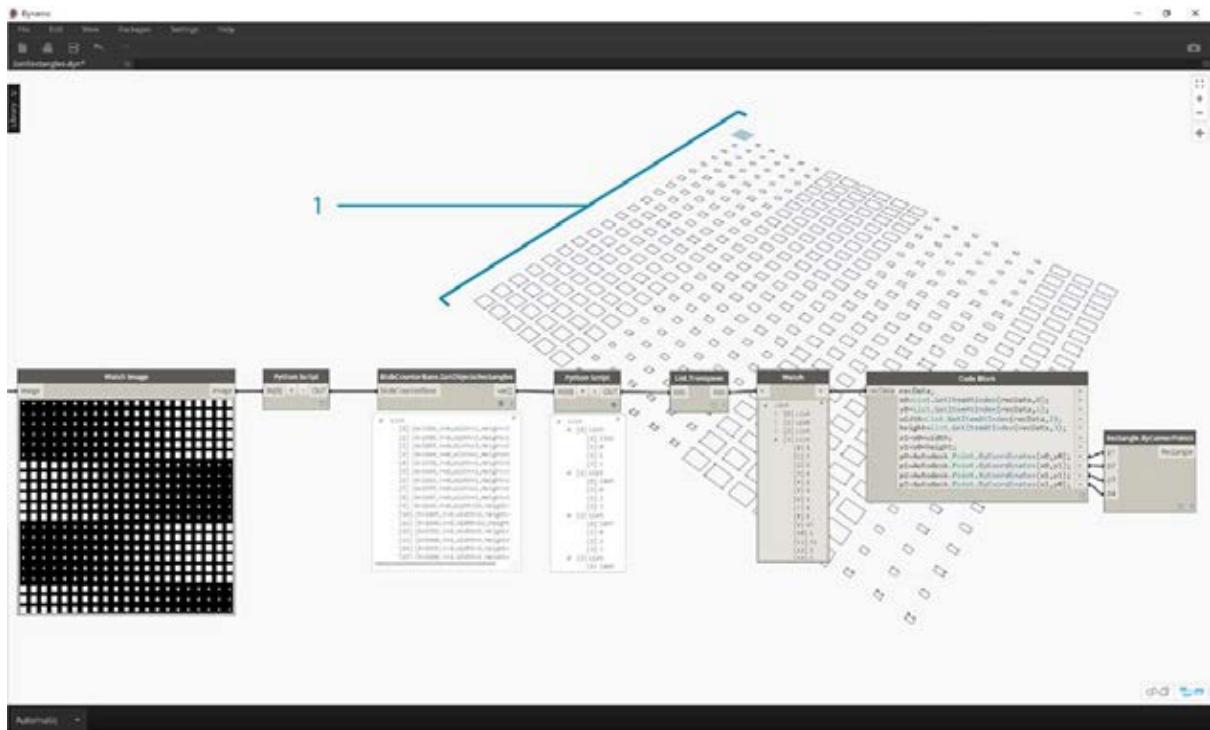


- Vertauschen Sie die Listenebenen aus der Ausgabe des Python-Blocks aus dem vorigen Schritt mithilfe von Transpose. Dadurch erhalten Sie vier Listen, jeweils mit den x-, y-, Breiten- und Höhenwerten der einzelnen Rechtecke.
- Mithilfe eines Codeblocks ordnen Sie die Daten in einer für den Rectangle.ByCornerPoints-Block geeigneten Struktur (mithilfe des folgenden Codes).

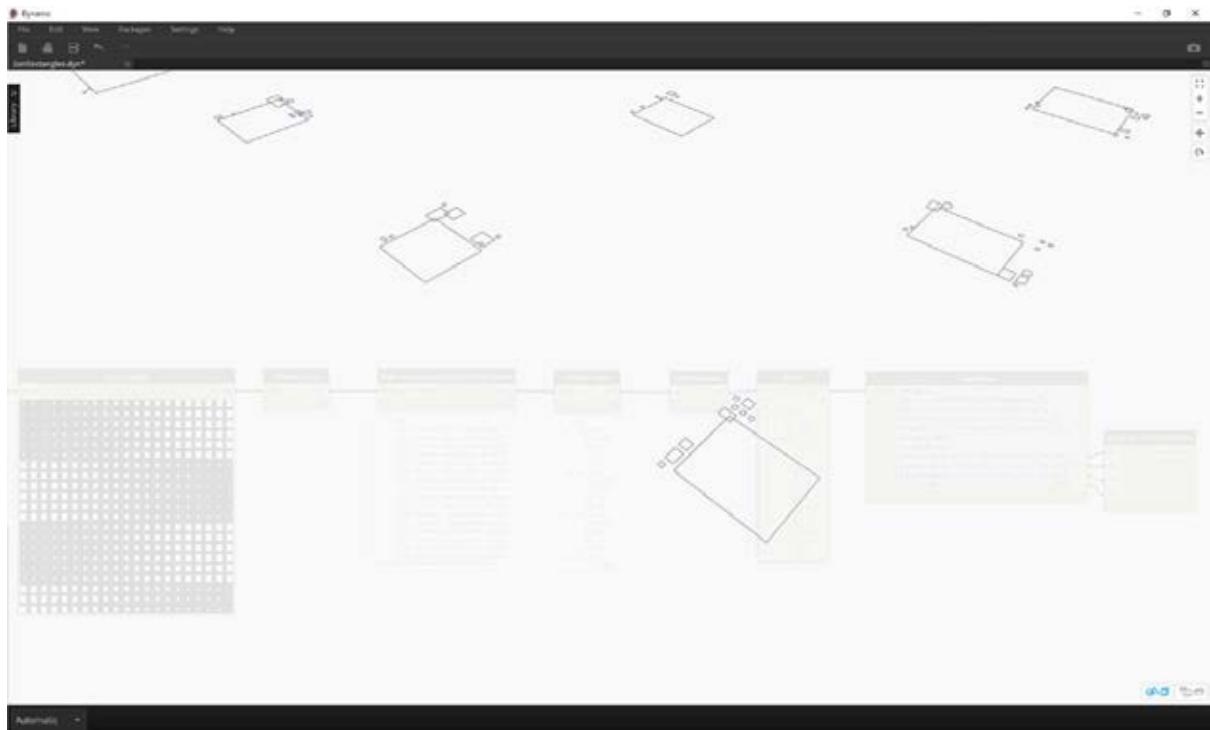
```

recData;
x0=List.GetItemAtIndex(recData,0);
y0=List.GetItemAtIndex(recData,1);
width=List.GetItemAtIndex(recData,2);
height=List.GetItemAtIndex(recData,3);
x1=x0+width;
y1=y0+height;
p0=Autodesk.Point.ByCoordinates(x0,y0);
p1=Autodesk.Point.ByCoordinates(x0,y1);
p2=Autodesk.Point.ByCoordinates(x1,y1);
p3=Autodesk.Point.ByCoordinates(x1,y0);

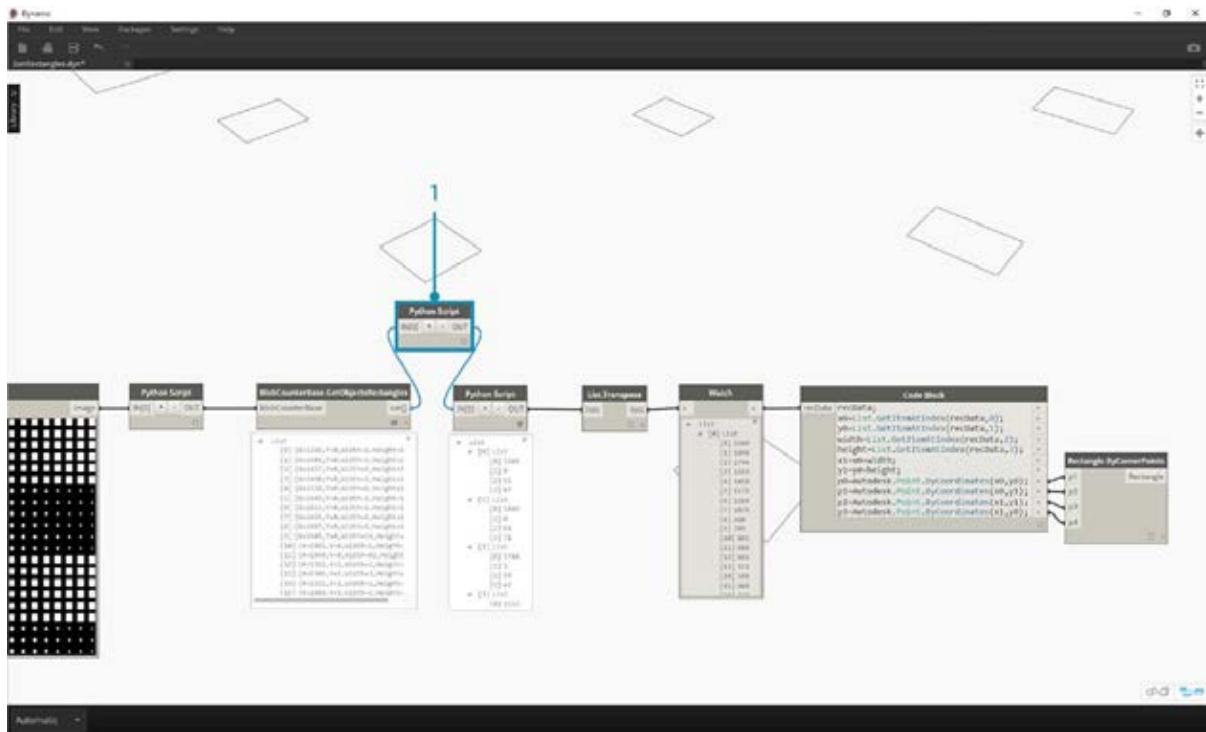
```



Die verkleinerte Darstellung zeigt eine Reihe aus Rechtecken, die die weißen Quadrate aus dem Bild darstellen. Durch diese Programmierung haben Sie einen Vorgang durchgeführt, der der Live Trace-Funktion von Illustrator recht nahe kommt.

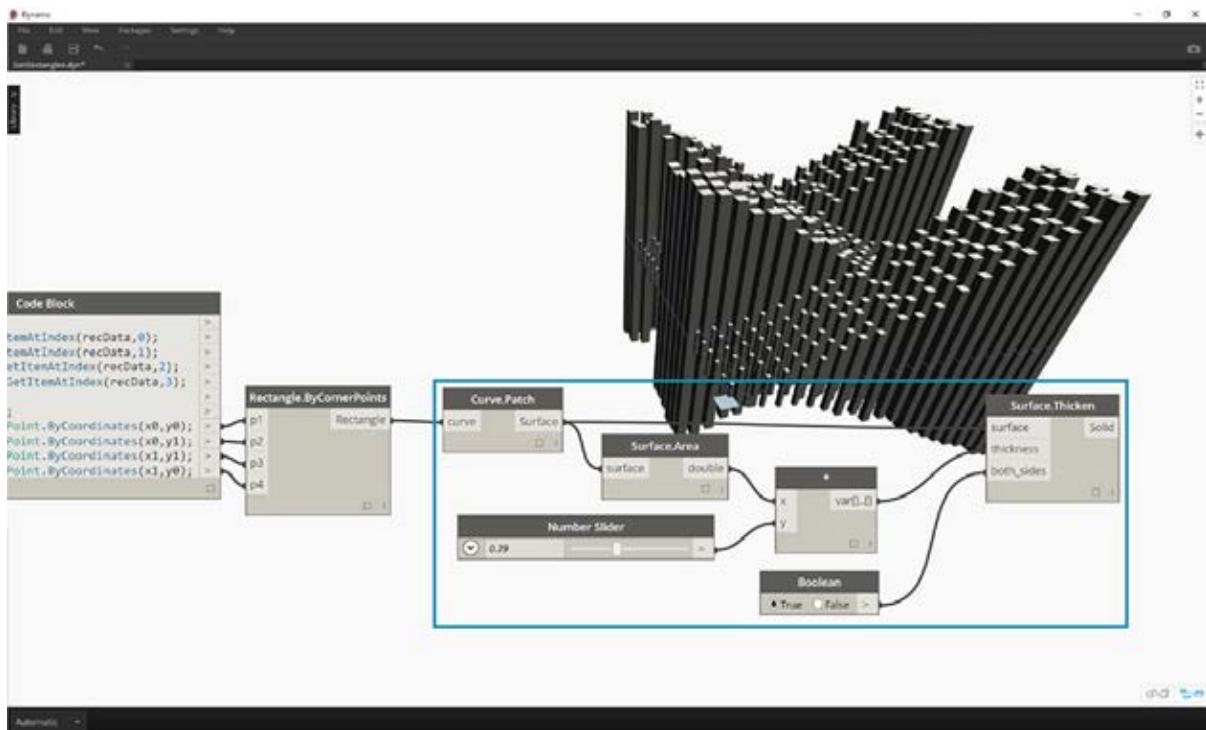


Das Ergebnis muss jedoch noch bereinigt werden. Die vergrößerte Darstellung zeigt, dass einige unerwünschte kleine Rechtecke vorhanden sind.

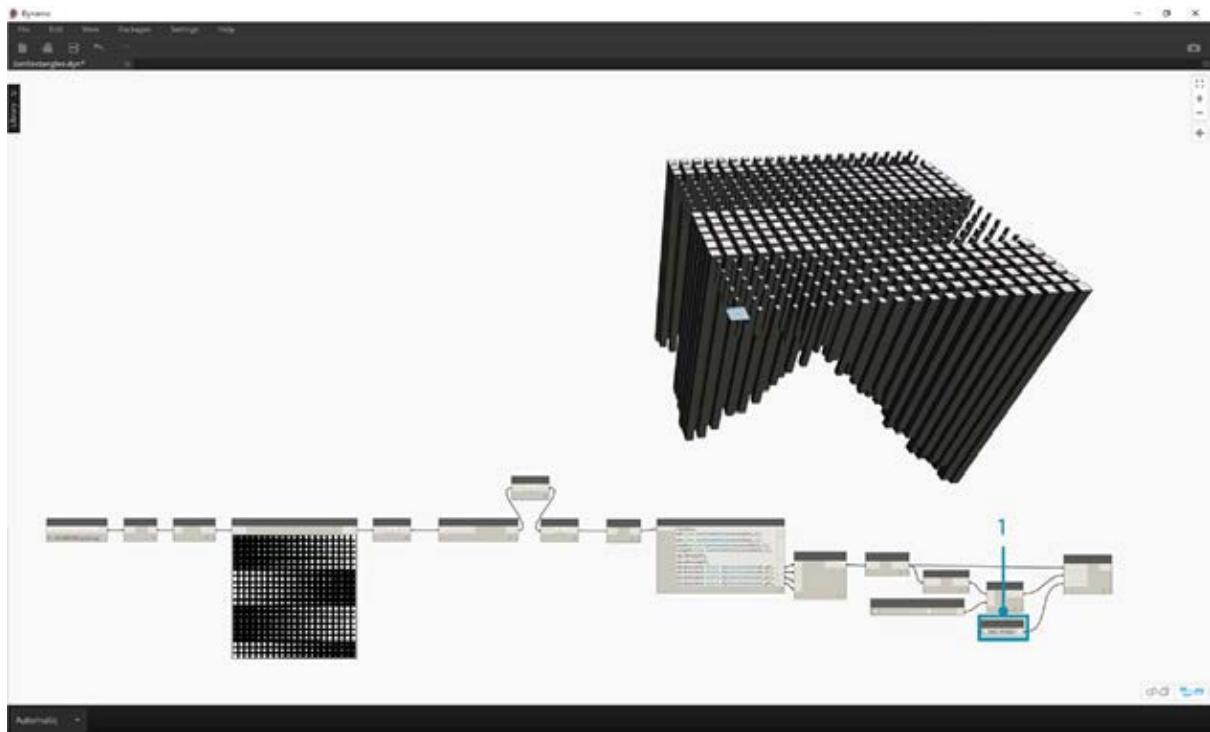


- Sie entfernen die unerwünschten Rechtecke, indem Sie einen weiteren Python-Block zwischen dem GetObjectRectangles-Block und dem bestehenden Python-Block einfügen. Der Code für diesen Block wird unten gezeigt. Er entfernt alle Rechtecke, die kleiner als die angegebene Größe sind.

```
rectangles=IN[0]
OUT=[]
for rec in rectangles:
if rec.Width>8 and rec.Height>8:
OUT.append(rec)
```



Damit haben Sie die überflüssigen Rechtecke beseitigt. Erstellen Sie jetzt interessehalber eine Oberfläche aus diesen Rechtecken und extrudieren Sie diese um eine Entfernung in Abhängigkeit von ihrer Fläche.



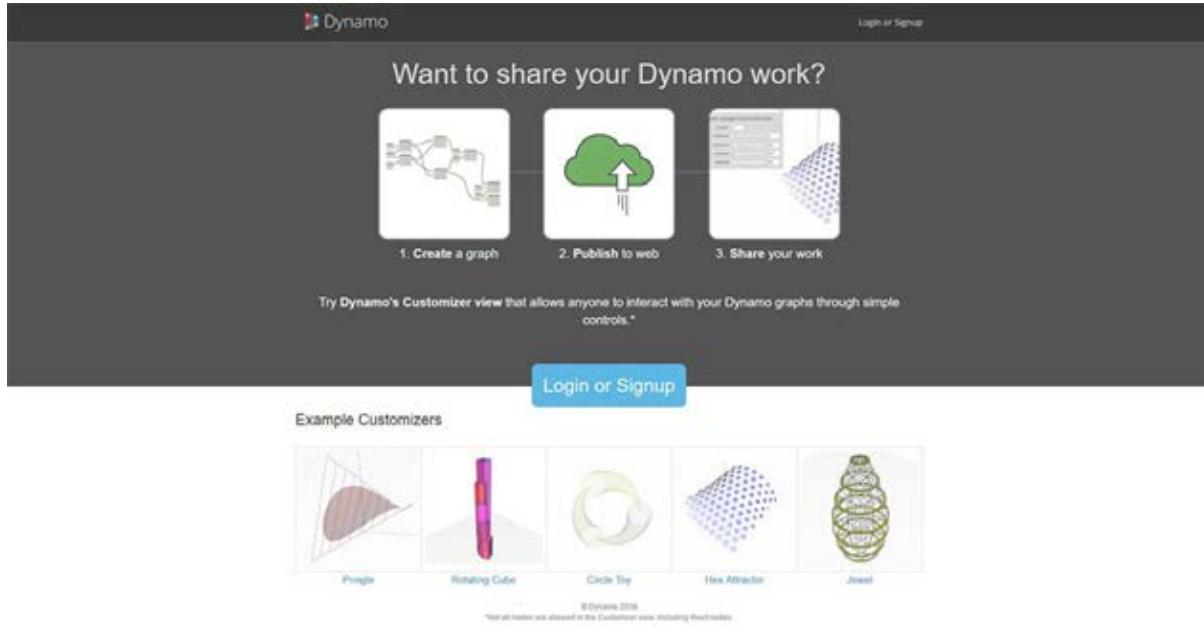
1. Ändern Sie zum Schluss die both\_sides-Eingabe in "false". Damit erhalten Sie eine Extrusion in nur eine Richtung. Tauchen Sie diese Form in Kunstharz, um den perfekten Tisch für Nerds zu erhalten!

Die hier gezeigten Beispiele sind relativ einfach, die beschriebenen Konzepte jedoch lassen sich auf faszinierende reale Anwendungen übertragen. Computer Vision kann für eine Vielzahl von Prozessen verwendet werden. Hierzu gehören, um nur einige Beispiele zu nennen, Barcode-Scanner, Perspective-Matching, [Projektionsmapping](#) und [erweiterte Realität](#). Erweiterte Themen mit AForge für diese Übung finden Sie in [diesem Artikel](#).

# Dynamo im Internet

## Dynamo im Internet

In Dynamo Studio steht jetzt eine Funktion zur Übermittlung ins Internet zur Verfügung, die Ihnen die Arbeit mit Dynamo über das Internet ermöglicht. Diese Funktion ermöglicht es anderen Benutzern, über eine auf die hierfür nötigen Funktionen (zulässige Eingaben wie Schieberegler, Zahlen und boolesche Werte) beschränkte Benutzeroberfläche, die Customizer-Ansicht, mit Ihren Skripts zu interagieren. Dadurch werden Ihre Skripts für größere Benutzergruppen zugänglich, die eventuell nicht mit Dynamo oder visueller Programmierung vertraut sind.



Dynamo im Internet

# Ins Internet senden (mit Dynamo Studio)

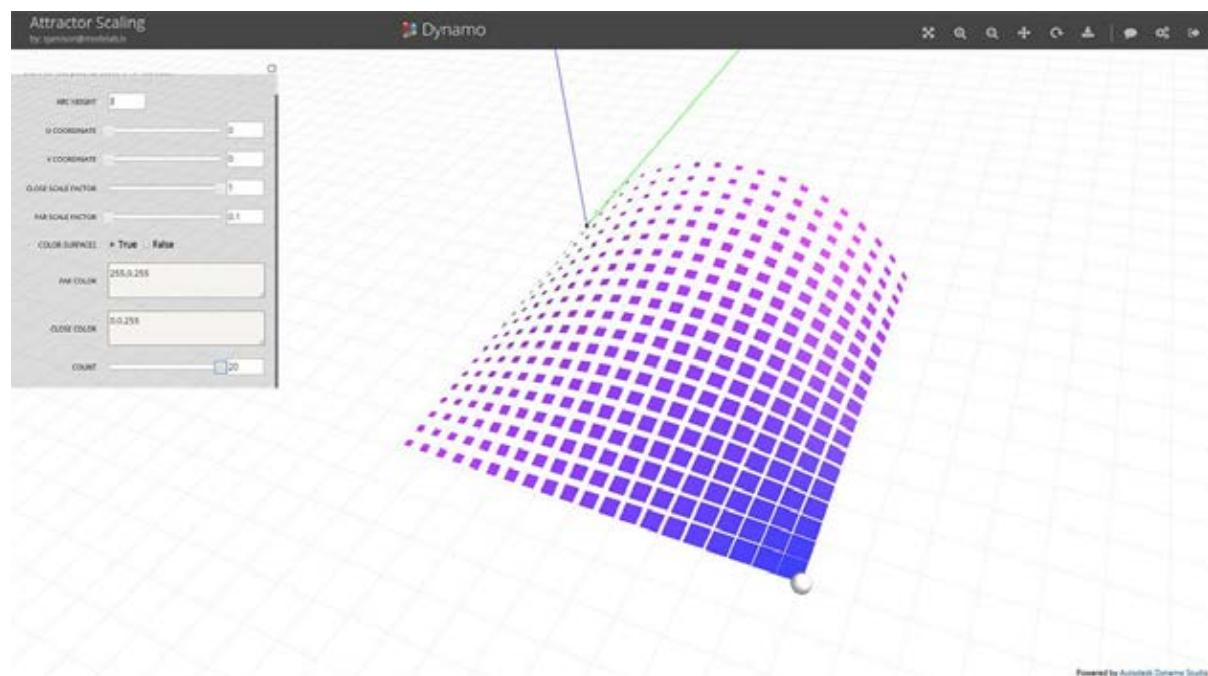
## Ins Internet senden (mit Dynamo Studio)

Mit Dynamo Studio können Sie Ihre Dateien schnell und leicht im Internet veröffentlichen. Sie sollten sich jedoch eventuell etwas Zeit nehmen, um Ihre Eingaben auszuwählen und zu beschriften und Ihre Datei benutzerfreundlich zu gestalten. Wenn Sie Dynamo Studio herunterladen müssen, besuchen Sie die [Website von Autodesk](#), wo Sie weitere Informationen erhalten.

### Übung: Vorbereiten der Übermittlung ins Internet.

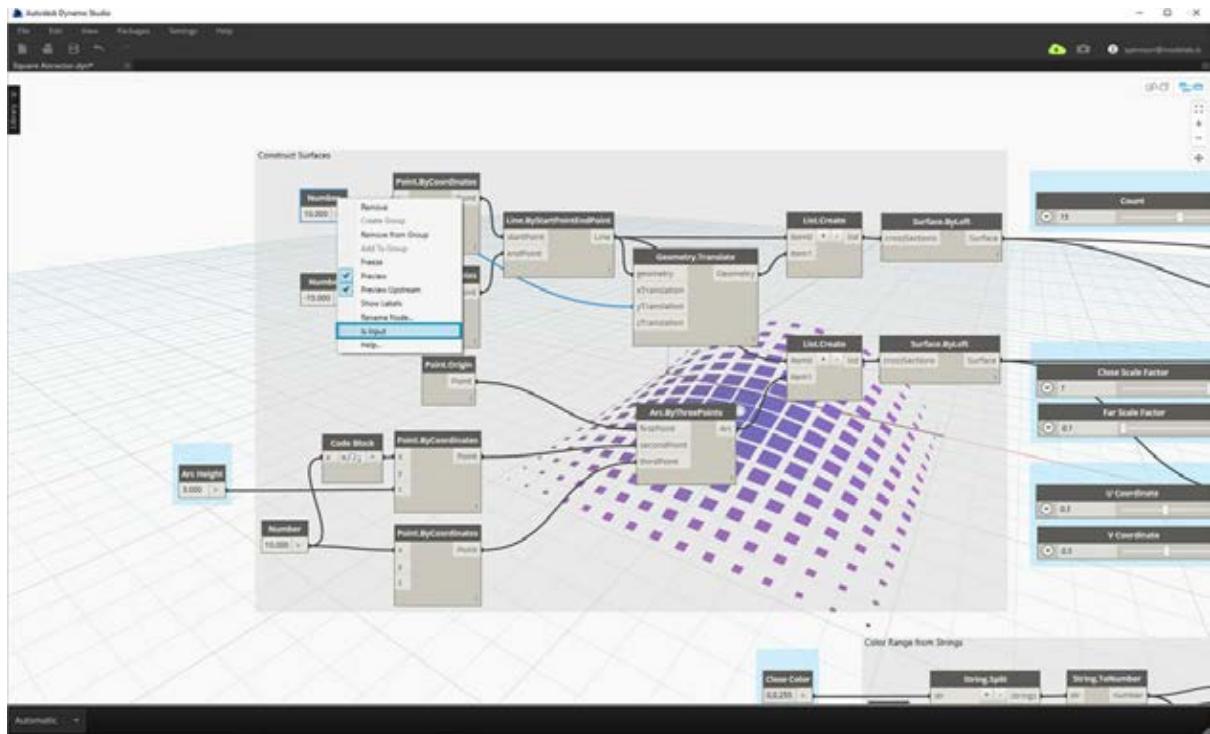
Laden Sie die zu dieser Übungslektion gehörigen Beispieldateien herunter (durch Rechtsklicken und Wahl der Option Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [Attractor Scale-Beispiel herunterladen](#)

In dieser Übung veröffentlichen Sie ein Dynamo-Diagramm im Internet. In dieser Datei wird ein Raster aus Rechtecken erstellt, die mithilfe eines Attraktors skaliert und von einer Basisoberfläche ausgehend einer Zieloberfläche zugeordnet werden. Aus den einzelnen Rechtecken werden Oberflächenfelder erstellt und in Abhängigkeit von ihrer Entfernung zum Attraktor eingefärbt.



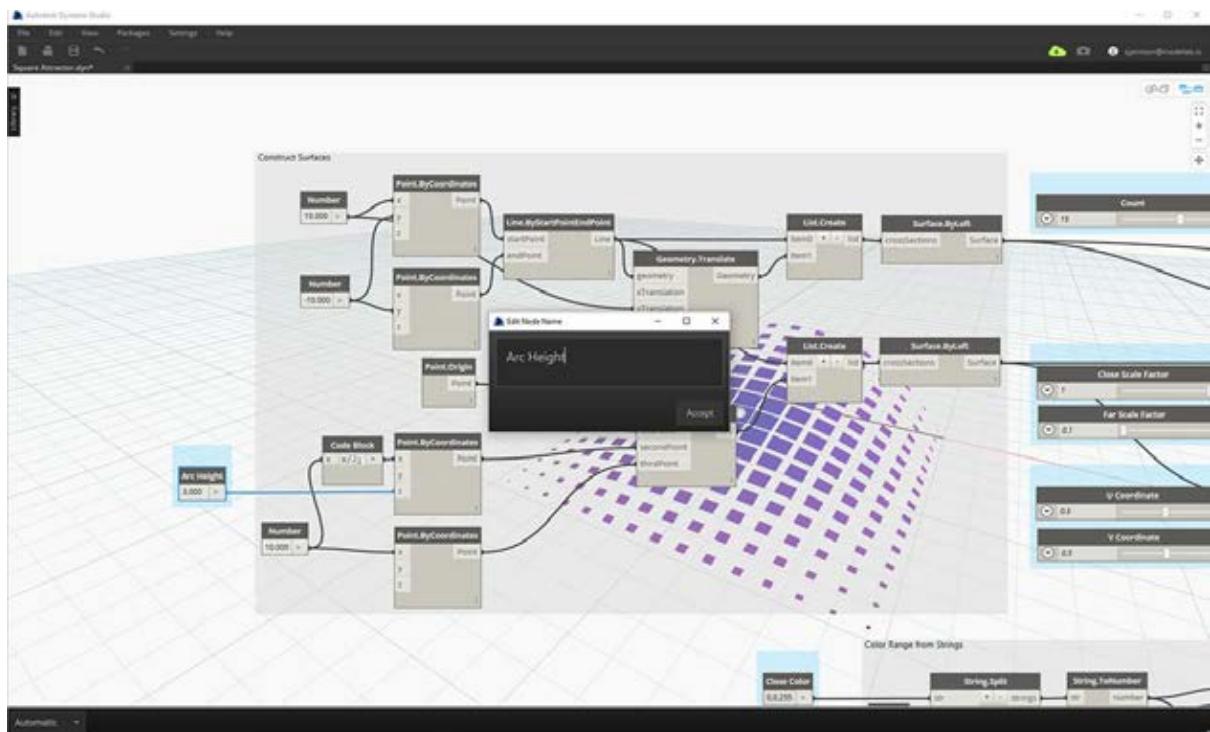
Dies ist der Customizer, den Sie hier erstellen. Sehen Sie dieses Beispiel im [Internet](#) an.

Um Ihr Skript für die Veröffentlichung vorzubereiten, entscheiden Sie zunächst, welche Eingaben für die Benutzer zugänglich sein sollen. Zu den zulässigen Eingaben gehören Schieberegler, Zahlen, Zeichenfolgen und boolesche Werte. Codeblöcke und Dateipfade können nicht als Eingaben verwendet werden. Deaktivieren Sie über das Kontextmenü die Option Ist Eingabe für alle Eingaben, die nicht in der Customizer-Ansicht angezeigt werden sollen. Achten Sie darauf, in allen Schieberegler-Eingaben angemessene Minimal- und Maximalwerte anzugeben.



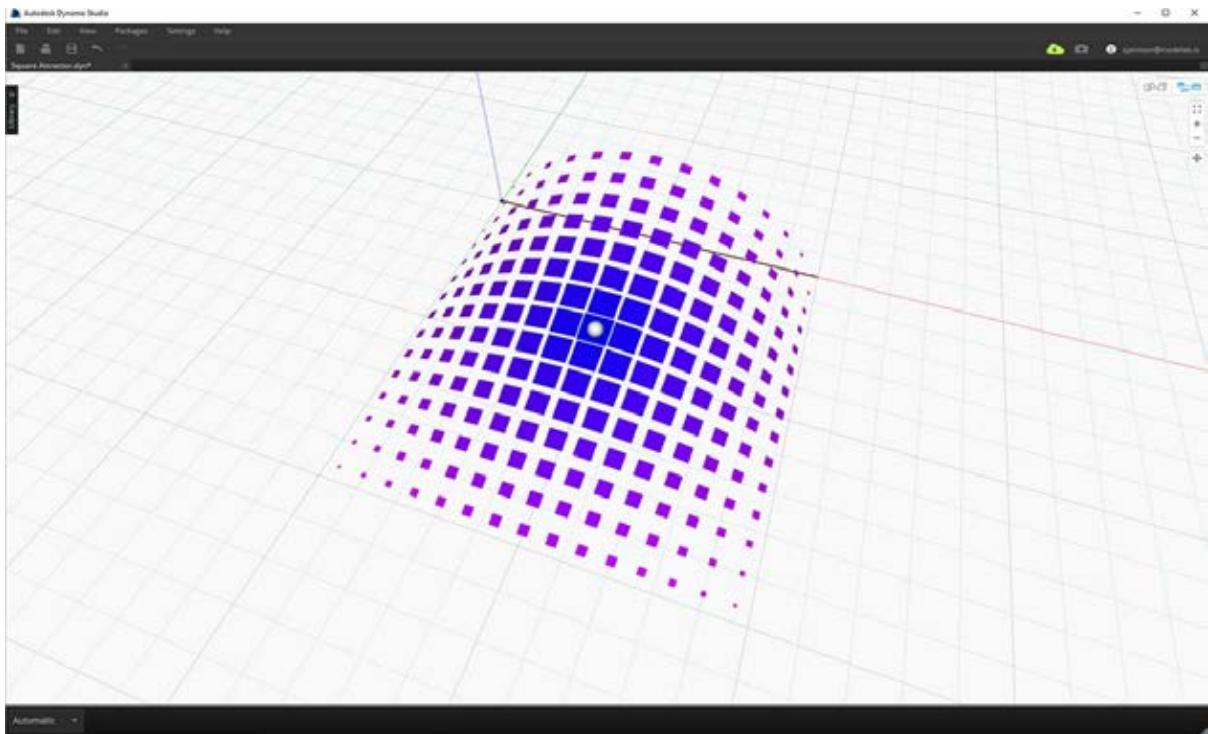
Deaktivieren Sie über das Kontextmenü die Option Ist Eingabe für alle Eingaben, die nicht im Customizer angezeigt werden sollen.

Zweitens müssen Sie sicherstellen, dass alle Eingaben unmissverständlich beschriftet sind.



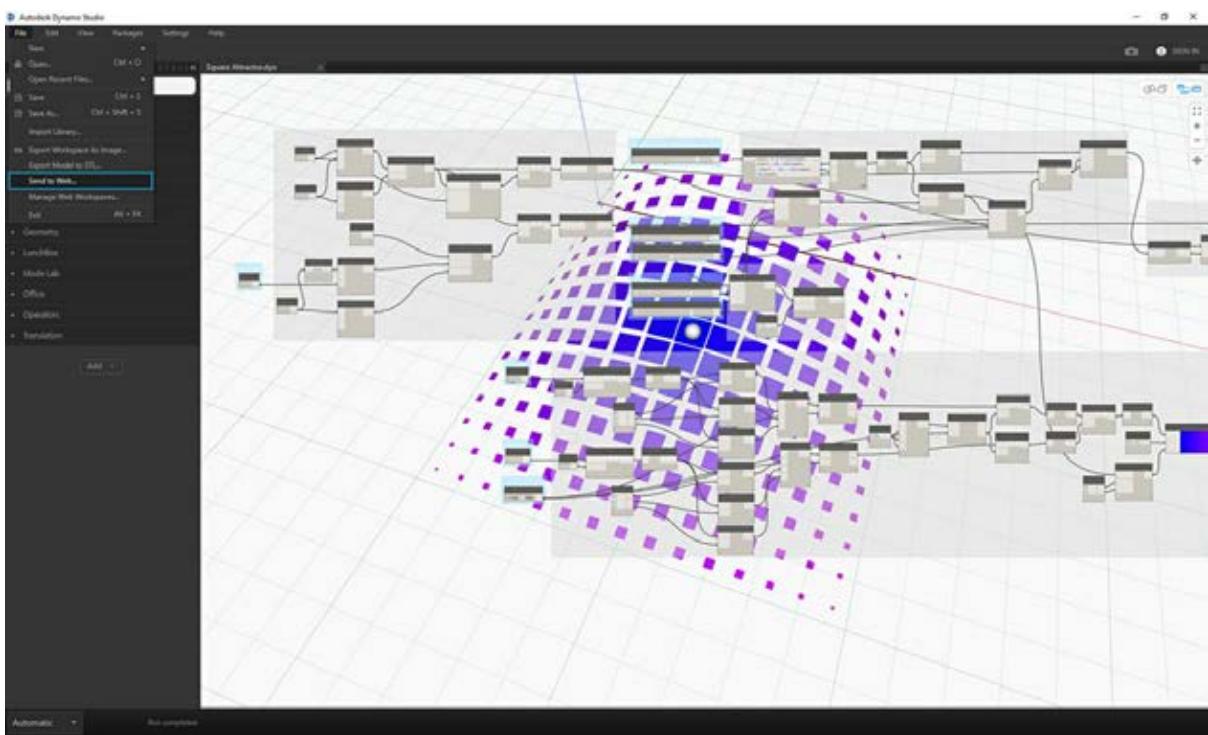
Beschriften Sie die Eingaben, indem Sie auf den Namen des Blocks doppelklicken, um ihn zu bearbeiten.

Beziehen Sie Vorschageometrie ein, um Ihr Skript leichter verständlich zu machen. In diesem Beispiel wird die Position des Attraktors durch eine Kugel markiert und die Oberflächen werden in Abhängigkeit von ihrer Entfernung zum Attraktor gefärbt. Dadurch ist die Wirkung des Attraktors leicht zu visualisieren und zu verstehen.

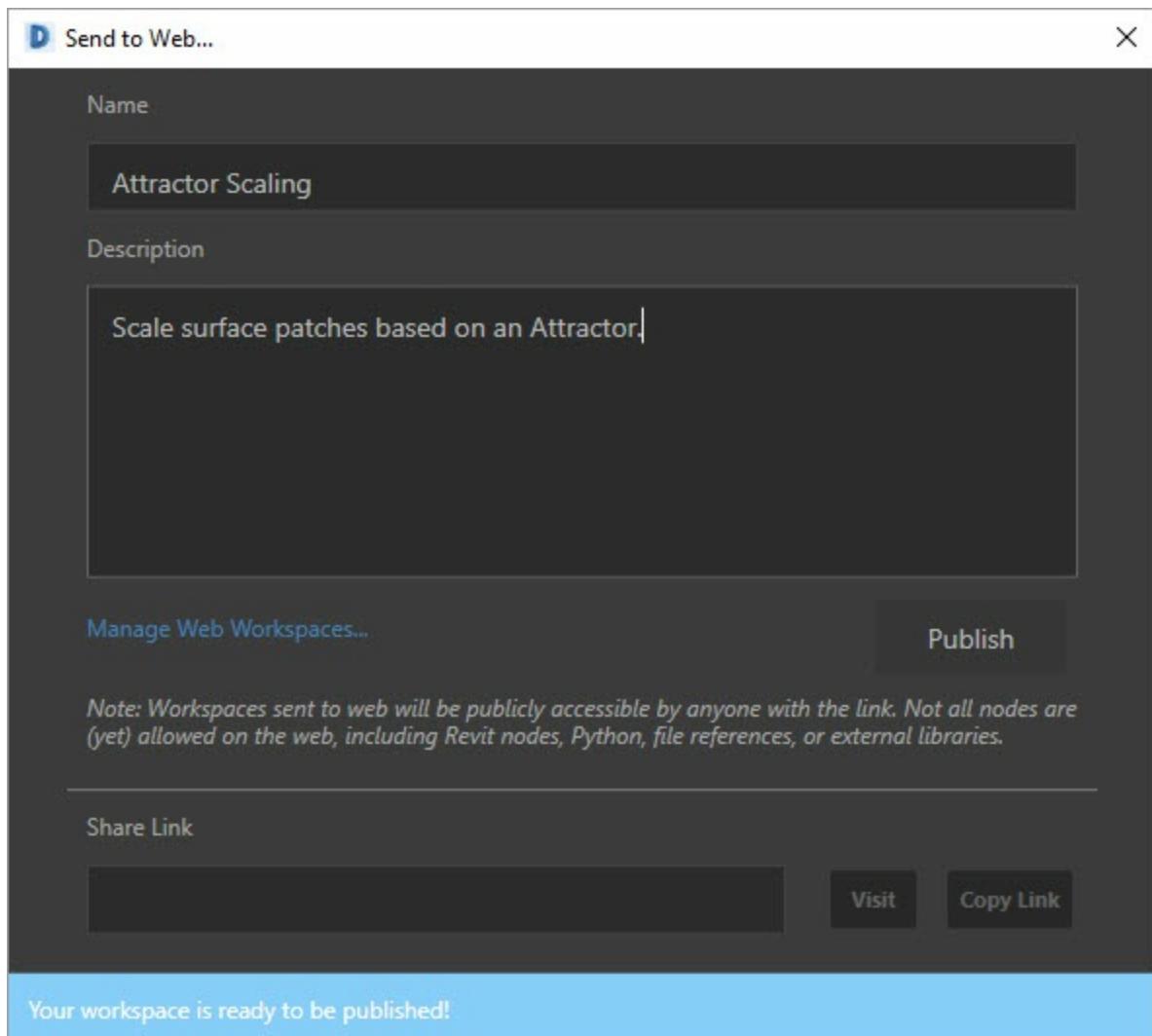


## Datei veröffentlichen

Wenn Ihre Datei zur Veröffentlichung bereit ist, wählen Sie im Menü Datei die Option zum Senden ins Internet.



Geben Sie eine Beschreibung der Datei sowie gegebenenfalls Anweisungen zum Einstieg ein. Nachdem Sie Ihre Datei veröffentlicht haben, können Sie einen Link an beliebige Benutzer senden, die über ein Autodesk-Konto verfügen. Die Datei wird mit den aktuellen Eingabewerten und Vorschauen veröffentlicht.



Beispiel im [Internet](#) anzeigen

## Verwalten veröffentlichter Dateien

Um Ihre veröffentlichten Skripts zu verwalten, besuchen Sie <https://dynamo.autodesk.com> und melden Sie sich bei Ihrem Konto an. Wählen Sie in der Dropdown-Liste rechts oben die Option Manage. Auf dieser Seite können Sie Arbeitsbereiche, die Sie bereits veröffentlicht haben, bearbeiten, freigeben oder löschen. Sie können diese Seite auch über die Option zum Verwalten von Web-Arbeitsbereichen im Menü Datei von Dynamo Studio aufrufen.

Name	Description	Last Modified ↑
Attractor Scaling	Scale surface patches based on an attractor.	1 minute ago

© Dynamo 2015

1. Arbeitsbereich bearbeiten
2. Arbeitsbereich löschen
3. Link freigeben

# Customizer-Ansicht

## Customizer-Ansicht

Die Dynamo Customizer-Ansicht ermöglicht anderen Benutzern die Interaktion mit Ihren Dynamo-Skripts über das Internet in einer vereinfachten Benutzeroberfläche mit Eingaben wie Schiebereglern, Zahlen und booleschen Werten.

In der Customizer-Ansicht werden komplexe Diagramme in einer einfachen Benutzeroberfläche zusammengefasst und dadurch für eine größere Community von Benutzern zugänglich gemacht, die eventuell nicht mit Dynamo, visueller Programmierung oder 3D-Modellierung vertraut sind. Jeder Benutzer, der über ein Autodesk-Konto verfügt, kann in einer Customizer-Ansicht über einen freigegebenen Link auf Ihr Dynamo-Skript zugreifen und damit arbeiten, ohne eine Dynamo-Lizenz zu benötigen.

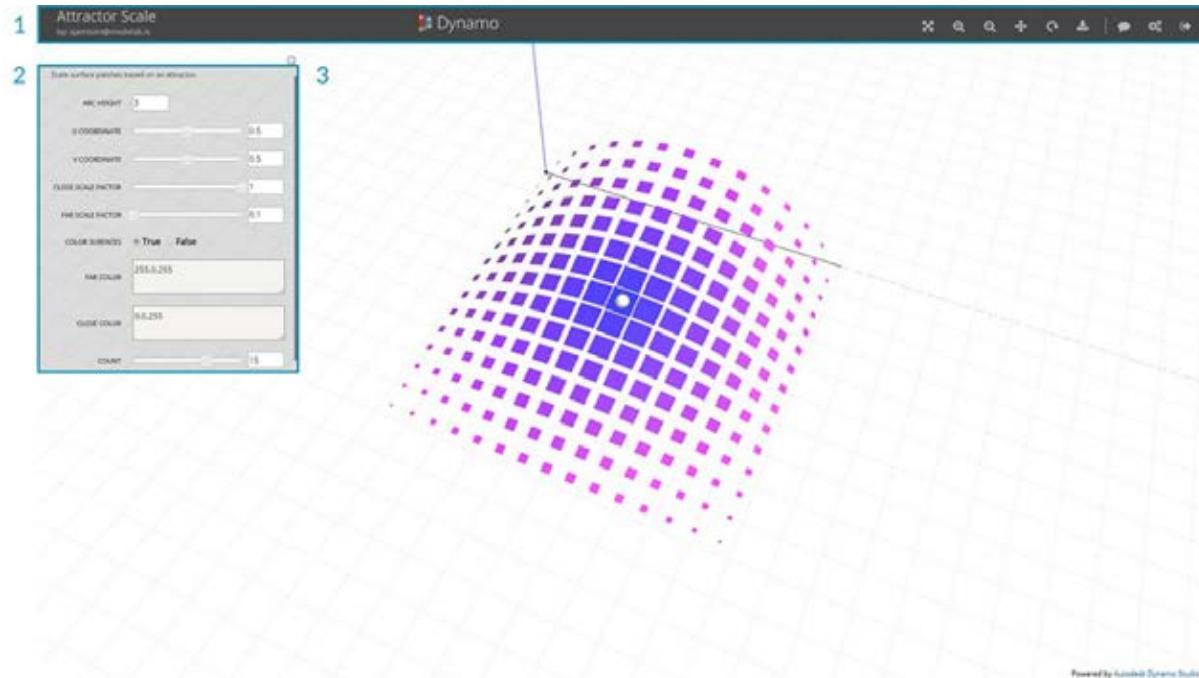
In der Customizer-Ansicht können Benutzer Geometrie als STL-Netz für die rasche Erstellung von Prototypen oder als Dynamo-Netzdatei exportieren.



Beispiele für die Customizer-Ansicht stehen auf <https://dynamo.autodesk.com/> zur Verfügung.

## Benutzeroberfläche der Customizer-Ansicht

Die Customizer-Ansicht besteht aus einer Menüleiste, einem Flyout-Menü mit einer Beschreibung der Datei und Benutzereingaben sowie einer 3D-Ansicht, ähnlich dem Dynamo-Arbeitsbereich.



1. Menüleiste
2. Steuerelemente
3. 3D-Vorschau

## Menü der Customizer-Ansicht

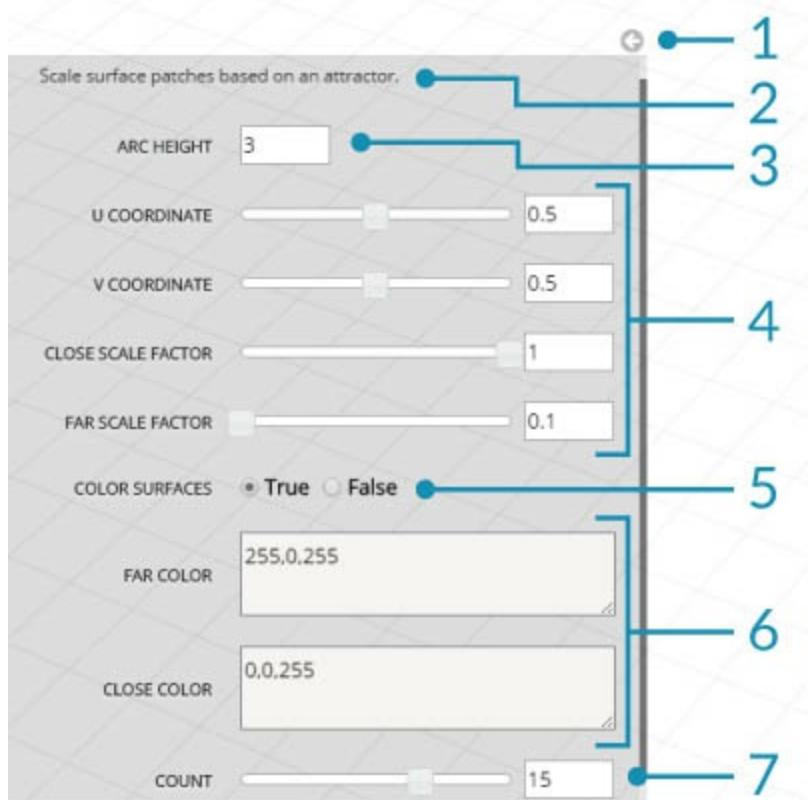
Die Menüleiste der Customizer-Ansicht umfasst Informationen zur Datei, Steuerelemente zur Navigation sowie Download-Optionen.



1. Titel: Name der Datei
2. Autor: Eigentümer der Datei
3. Zoom Grenzen: Zoom auf die Grenzen der Geometrie
4. Vergrößern/Verkleinern: Steuerung der Zoom-Einstellung
5. Schwenken/Umkreisen: Wechseln zwischen Schwenken und Umkreisen
6. Herunterladen: Speichern der Datei als STL- oder DYN-Datei
7. Feedback: Senden von Kommentaren, Vorschlägen oder Problemen
8. Verwalten/Info: Nützliche Informationen über Dynamo
9. Abmelden: Abmelden vom Konto und Beenden der Customizer-Ansicht

## Steuerelemente

Das Menü mit den Steuerelementen enthält die Eingaben für das Dynamo-Skript. Dazu gehören Zahlen, Schieberegler, Zeichenfolgen und boolesche Werte sowie eine kurze Beschreibung der Datei. Die Steuerelemente entsprechen den in der Dynamo-Originaldatei enthaltenen Eingaben und sind daher von Skript zu Skript unterschiedlich. Dieses Menü kann durch Klicken auf das Pfeilsymbol ausgeblendet werden.

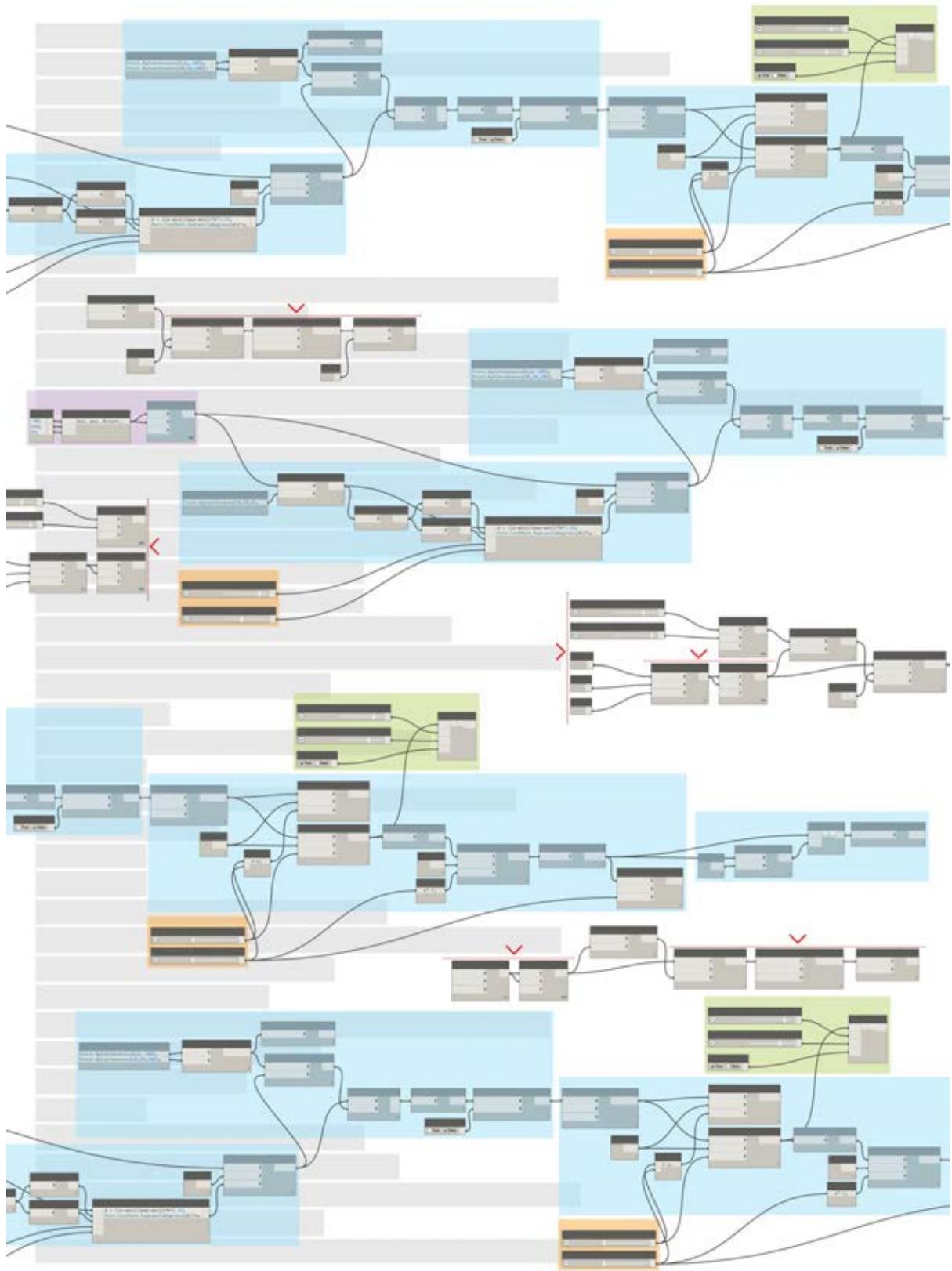


1. Steuerelemente ausblenden/erweitern
2. Dateibeschreibung
3. Number-Eingabe
4. Number Sliders
5. Boolescher Wert
6. Zeichenfolgen
7. Integer Slider

# **Optimale Verfahren**

## **Optimale Verfahren**

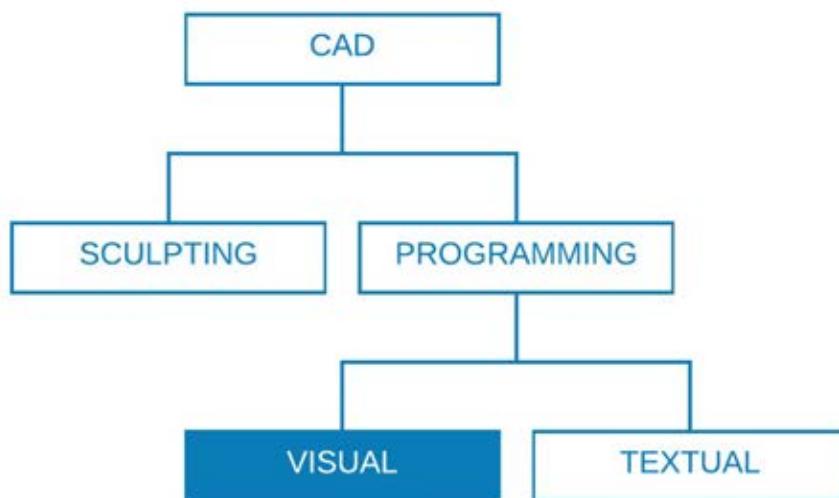
Dieser Teil des Leitfadens ist quasi ein laufendes Protokoll für optimale Verfahren. Wir stellen hier eine Reihe von Vorgehensweisen vor, die sich unserer Erfahrung nach und bei Recherchen als besonders nützlich für hochwertige parametrische Arbeitsabläufe erwiesen haben. Als Designer und Programmierer legen wir einen Qualitätsmaßstab mit den Hauptkriterien einfache Wartung, Zuverlässigkeit, Benutzerfreundlichkeit und Effizienz unserer Werkzeuge zugrunde. In diesen optimalen Verfahren werden zwar spezielle Beispiele für visuelle oder textbasierte Skripterstellung gegeben, ihre Prinzipien gelten jedoch für alle Programmierumgebungen und können als Anregung für viele berechnungsbasierte Arbeitsabläufe dienen.



# Vorgehensweisen für Diagramme

## Vorgehensweisen für Diagramme

In den vorangegangenen Kapiteln dieses Handbuchs wurde bereits behandelt, wie Sie die leistungsstarken Funktionen zur visuellen Programmierung in Dynamo einsetzen können. Ein gutes Verständnis dieser Funktionen ist eine solide Grundlage und der erste Schritt bei der Erstellung zuverlässiger visueller Programme. Bei der Verwendung visueller Programme in der Praxis, der Weitergabe an Kollegen, der Behebung von Fehlern oder beim Testen von Grenzen müssen zusätzliche Aspekte berücksichtigt werden. Wenn andere Benutzer mit Ihrem Programm arbeiten sollen oder Sie damit rechnen, es z. B. sechs Monate später erneut zu öffnen, müssen seine Grafik und seine Logik unmittelbar verständlich sein. Dynamo stellt zahlreiche Werkzeuge zur Verfügung, die Ihnen helfen, die Komplexität Ihres Programms zu bewältigen. In diesem Kapitel finden Sie Richtlinien zu ihren Verwendungszwecken.

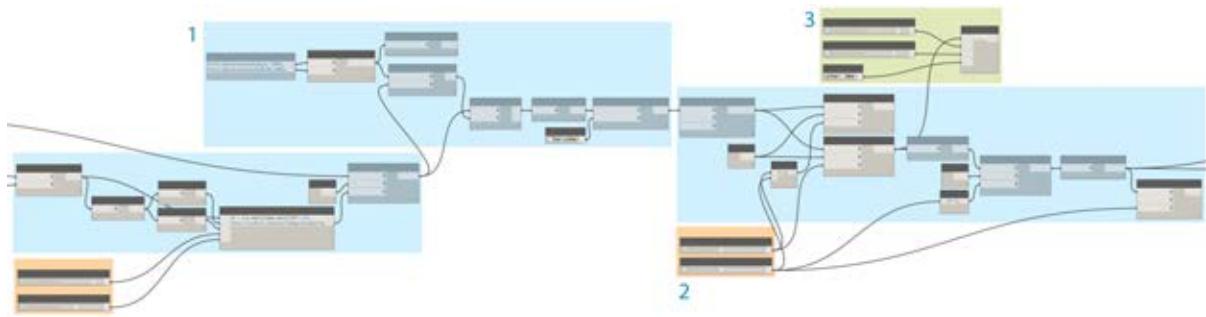


## Komplexität reduzieren

Während Sie Ihr Dynamo-Diagramm entwickeln und Ihre Ideen testen, kann es rasch an beachtlicher Größe und Komplexität zunehmen. Ein funktionsfähiges Programm zu erstellen, ist entscheidend, es ist jedoch auch wichtig, dies auf möglichst einfachem Weg zu erreichen. Das Diagramm lässt sich so nicht nur schneller und besser vorhersehbar ausführen, sondern seine Logik ist dadurch für Sie und andere Benutzer problemlos verständlich. Im Folgenden werden einige Methoden beschrieben, mit denen Sie die Logik Ihres Diagramms verdeutlichen können.

### Modularisieren mit Gruppen

- Gruppen ermöglichen es, bei der Entwicklung eines Programms **separate Teile mit unterschiedlichen Funktionen** zu erstellen.
- Mithilfe von Gruppen können Sie darüber hinaus **große Teile des Programms verschieben**, wobei die Modularität und Ausrichtung erhalten bleiben.
- Sie können die **Farbe einer Gruppe zur Differenzierung** ihres Verwendungszwecks (Eingaben oder Funktionen) ändern.
- Gruppen können als Ausgangspunkt beim **Organisieren des Diagramms zur Vereinfachung der Erstellung benutzerdefinierter Blöcke** verwendet werden.

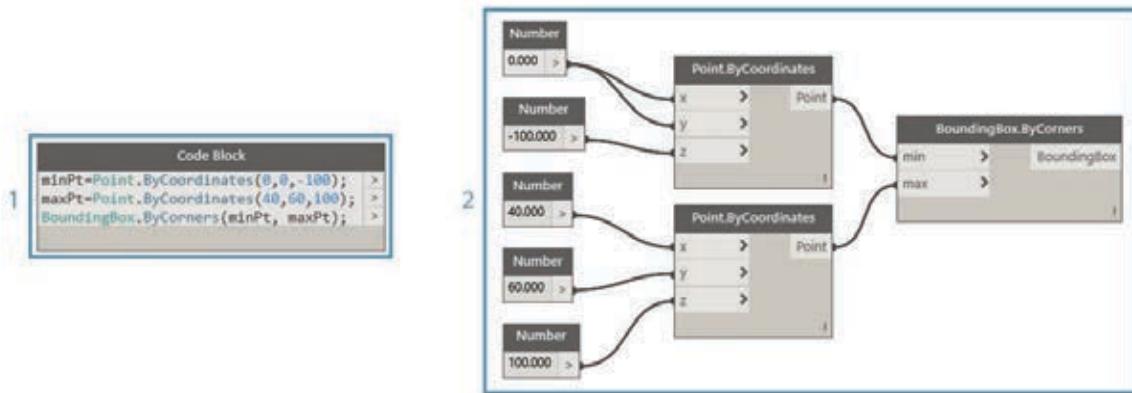


Die Farben in diesem Programm kennzeichnen den Verwendungszweck der einzelnen Gruppen. Mithilfe dieses Verfahrens können Sie eine Hierarchie in den von Ihnen entwickelten Grafikstandards oder -vorlagen erstellen.

1. Funktionsgruppe (blau)
2. Eingabengruppe (orange)
3. Skriptgruppe (grün) Informationen zur Verwendung von Gruppen finden Sie unter [Verwalten Ihres Programms](#).

### Effizientere Entwicklung mit Codeblöcken

- In manchen Fällen können Sie in einem Codeblock **eine Methode für eine Zahl oder einen Block schneller eingeben, als Sie nach ihr suchen könnten** (Point.ByCoordinates, Number, String, Formula).
- Codeblöcke sind nützlich zum **Definieren benutzerdefinierter Funktionen in DesignScript, damit weniger Blöcke im Diagramm benötigt werden**.



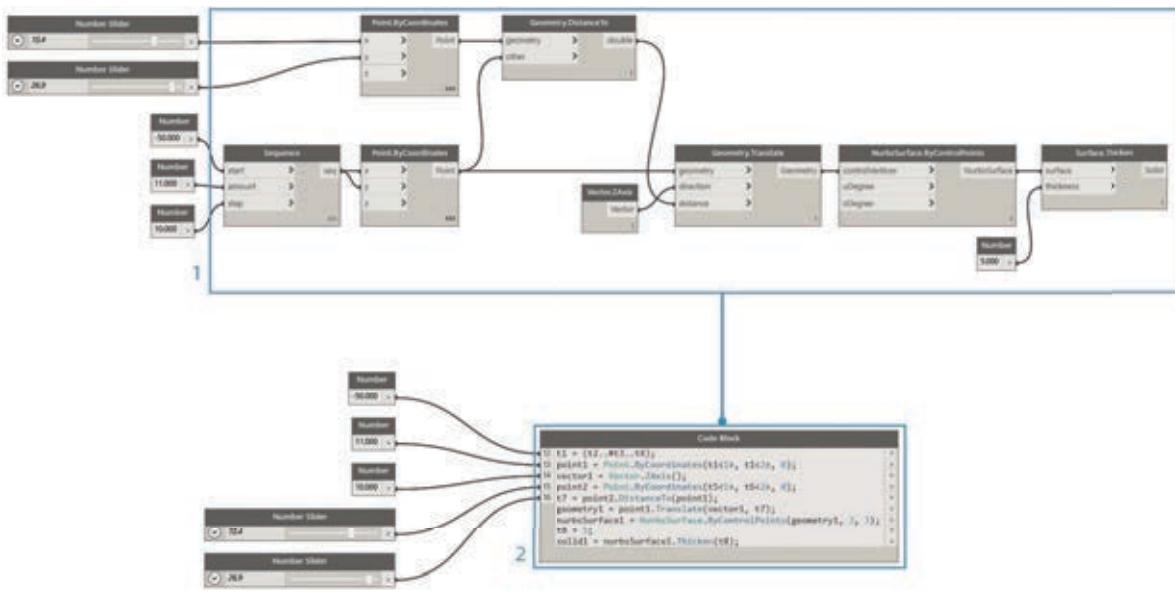
1 und 2 führen dieselbe Funktion aus. Dabei nahm das Schreiben einiger Codezeilen wesentlich weniger Zeit in Anspruch als das Suchen und Hinzufügen jedes einzelnen Blocks. Die Angaben im Codeblock sind darüber hinaus wesentlich prägnanter.

1. In Codeblock geschriebenes DesignScript
2. Entsprechendes Programm in Blöcken Informationen zur Verwendung von Codeblöcken finden Sie unter [Was ist ein Codeblock](#).

### Komprimieren mit Block zu Code

- Sie können **mithilfe von Block zu Code die Komplexität eines Diagramms reduzieren**, wobei eine Gruppe einfacher Blöcke zusammengefasst und das entsprechende DesignScript in einen einzigen Codeblock geschrieben wird.
- Block zu Code kann **Code komprimieren, ohne die Verständlichkeit des Programms zu beeinträchtigen**.
- Die Verwendung von Block zu Code bietet die folgenden **Vorteile**:
- Einfache Komprimierung von Code in eine einzige Komponente, die nach wie vor bearbeitet werden kann
- Vereinfachung eines großen Teils eines Diagramms
- Nützlich, wenn das „Mini-Programm“ nicht oft bearbeitet werden muss
- Nützlich für die Integration anderer Codeblock-Funktionalität, z. B. Funktionen
- Die Verwendung von Block zu Code bringt die folgenden **Nachteile** mit sich:

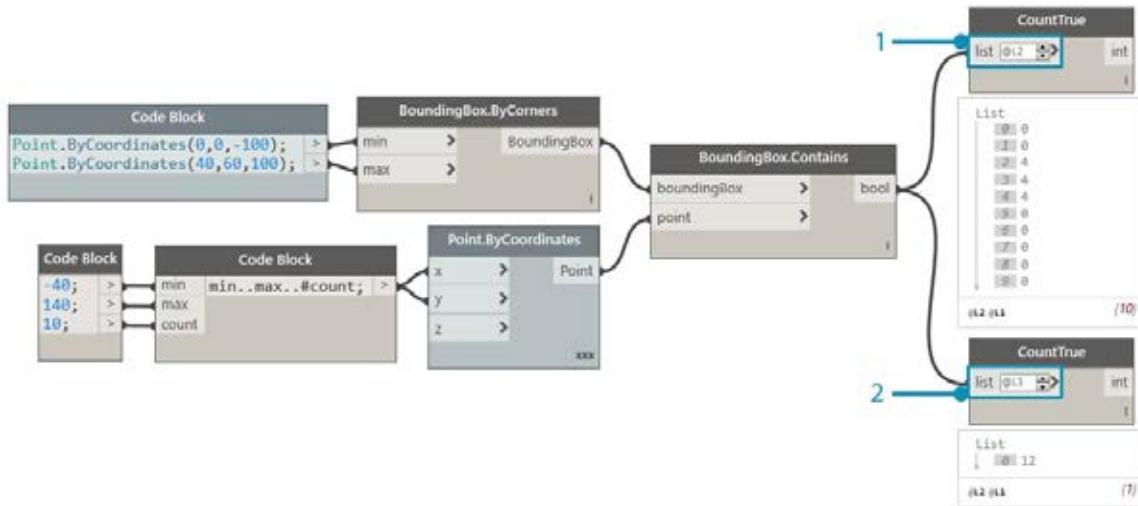
- Schlechtere Lesbarkeit wegen allgemeiner Benennung
- Für andere Benutzer schwieriger zu verstehen
- Keine einfache Möglichkeit, zur Version aus der visuellen Programmierung zurückzuwechseln



1. Vorhandenes Programm
2. Mithilfe von Block zu Code erstellter Codeblock Informationen zur Verwendung von Block zu Code finden Sie unter [DesignScript-Syntax](#).

### Flexibler Zugriff auf Daten mit List@Level

- List@Level kann es Ihnen erleichtern, Ihr Diagramm durch Ersetzen der Blöcke List.Map und List.Combine zu vereinfachen, die viel Platz im Ansichtsbereich beanspruchen können.
- List@Level bietet ein schnelleres Verfahren zum Konstruieren von Blocklogik als List.Map/List.Combine, indem es den Zugriff auf Daten auf einer beliebigen Ebene einer Liste direkt über den Eingabearschluss eines Blocks ermöglicht.



Sie können überprüfen, wie viele True-Werte BoundingBox.Contains zurückgibt und in welchen Listen diese enthalten sind, indem Sie List@Level für den list-Eingang von CountTrue aktivieren. List@Level ermöglicht es, die Ebene festzulegen, auf der die Eingabe Daten übernimmt. List@Level ist flexibel und effizient und wird gegenüber anderen Verfahren, die List.Map und List.Combine nutzen, dringend empfohlen.

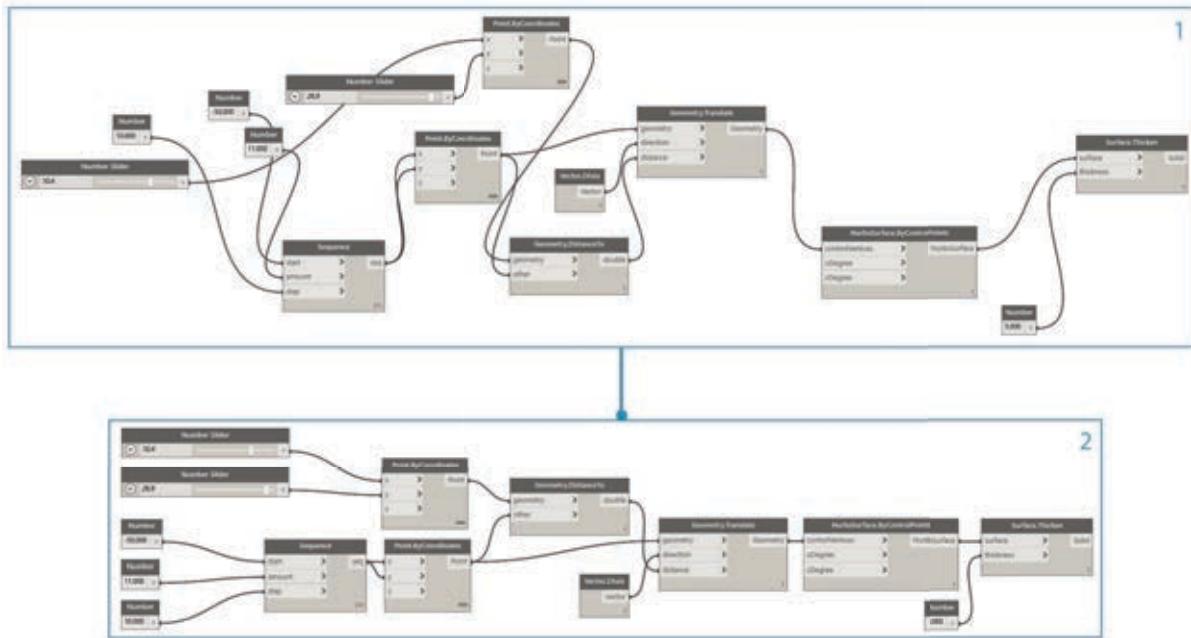
1. Zählen der True-Werte auf Listenebene 2
2. Zählen der True-Werte auf Listenebene 3 Informationen zur Verwendung von List@Level finden Sie unter [Listen von Listen](#).

### Lesbarkeit gewährleisten

Gestalten Sie Ihr Diagramm nicht nur so einfach und effizient wie möglich, sondern streben Sie auch eine übersichtliche grafische Darstellung an. Beziehungen sind trotz Ihrer Bemühungen, das Diagramm intuitiv mit logischen Gruppen zu gestalten, eventuell nicht ohne Weiteres zu erkennen. Ein einfacher Block innerhalb einer Gruppe oder das Umbenennen eines Schiebereglers kann Ihnen oder anderen Benutzern unnötige Verwirrung oder das Suchen im gesamten Diagramm ersparen. Im Folgenden werden mehrere Verfahren beschrieben, mit deren Hilfe Sie eine einheitliche Grafik innerhalb eines Diagramms und diagrammübergreifend erzielen können.

#### **Visuelle Kontinuität durch Ausrichten der Blöcke**

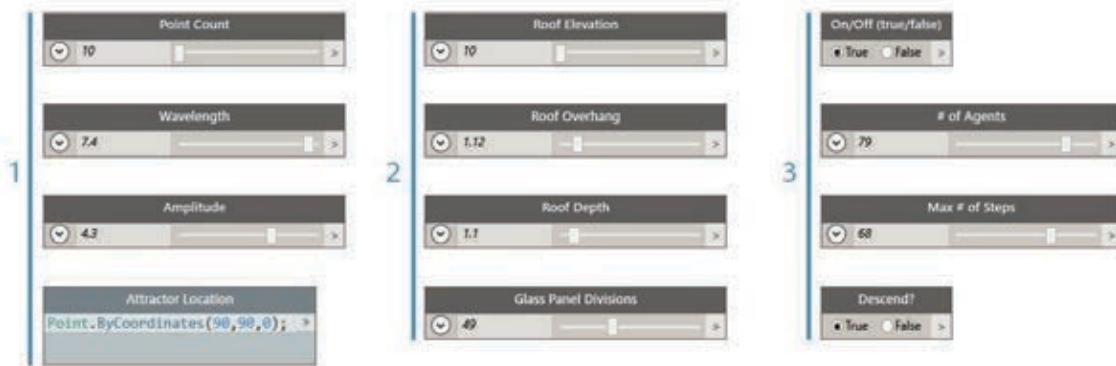
- Um den Arbeitsaufwand nach dem Erstellen des Diagramms zu reduzieren, achten Sie auf eine gute Leserlichkeit des Blocklayouts, indem Sie die **Blöcke während der Arbeit häufig ausrichten**.
- Wenn andere Benutzer mit Ihrem Diagramm arbeiten sollen, **sorgen Sie vor der Bereitstellung für ein Layout mit einem leicht verständlichen Ablauf aus Blöcken und Drähten**.
- Um die Ausrichtung zu erleichtern, **verwenden Sie die Funktion Blocklayout bereinigen zur automatischen Ausrichtung** des Diagramms. Durch manuelles Ausrichten erzielen Sie allerdings präzisere Ergebnisse.



1. Ungeordnetes Diagramm
2. Ausgerichtetes Diagramm Informationen zur Verwendung der Blockausrichtung finden Sie unter [Verwalten von Programmen](#).

#### Aussagekräftige Beschriftung durch Umbenennen

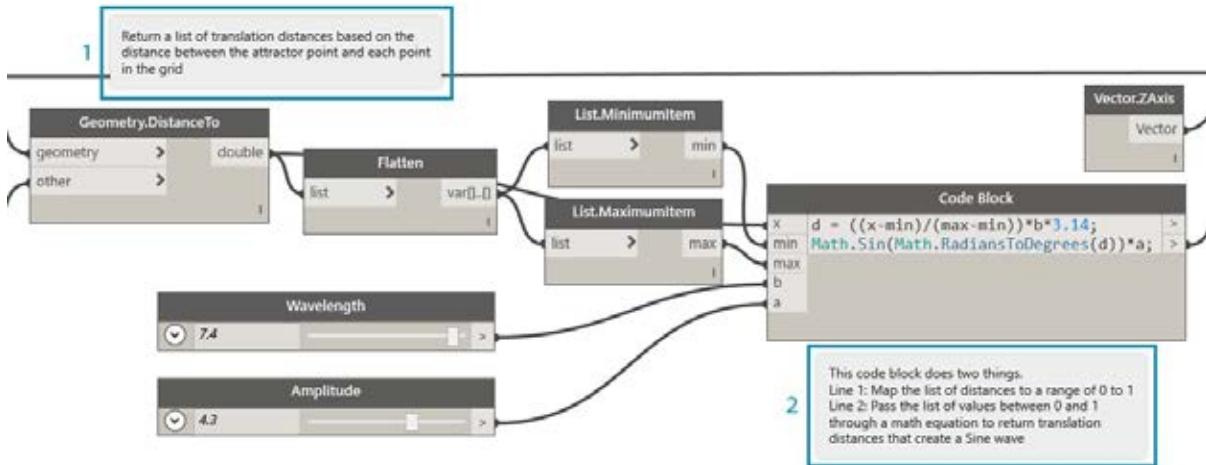
- Durch Umbenennen von Eingaben machen Sie Ihr Diagramm für andere Benutzer leicht verständlich, **insbesondere, wenn Objekte, die sich außerhalb des Bildschirms befinden, verbunden werden sollen.**
- **Benennen Sie nach Möglichkeit nicht Blöcke, sondern Eingaben um.** Als Alternative dazu können Sie einen benutzerdefinierten Block aus einer Gruppe von Blöcken erstellen und ihn umbenennen. Dabei ist ersichtlich, dass andere Elemente darin enthalten sind.



1. Eingaben für die Bearbeitung der Oberfläche
2. Eingaben für Architekturparameter
3. Eingaben für das Skript zur Simulation der Entwässerung Um einen Block umzubenennen, klicken Sie mit der rechten Maustaste auf seinen Namen, und wählen Sie Block umbenennen.

#### Erläuterungen durch Anmerkungen

- Fügen Sie eine Anmerkung hinzu, wenn ein Bestandteil des **Diagramms eine Erläuterung in Klartext benötigt**, die nicht in den Blöcken selbst gegeben werden kann.
- Fügen Sie eine Anmerkung hinzu, wenn eine Sammlung von **Blöcken oder eine Gruppe zu groß oder zu komplex ist und nicht direkt verstanden werden kann.**



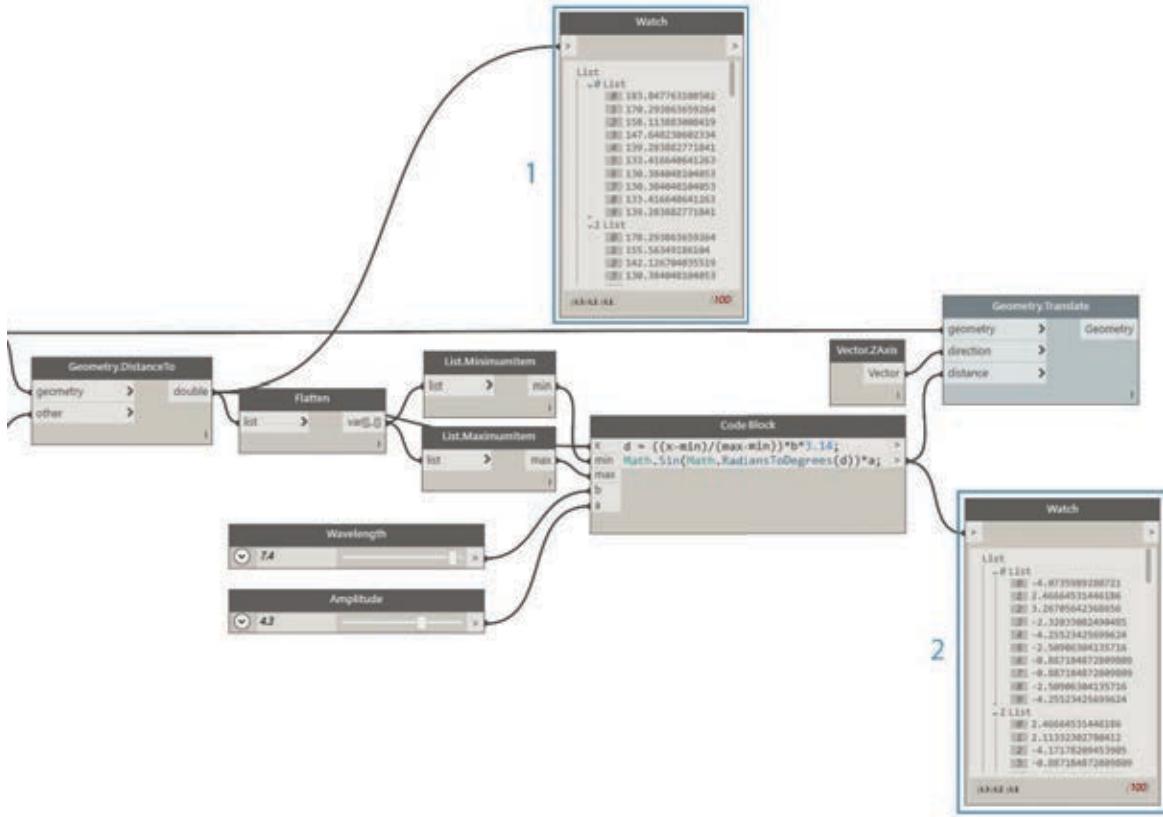
1. Anmerkung zur Beschreibung des Teils des Programms, der Rohwerte der Verschiebungsstrecken zurückgibt
2. Anmerkung zur Beschreibung des Codes, der diese Werte einer Sinuswelle zuordnet Informationen zum Hinzufügen einer Anmerkung finden Sie unter [Verwalten von Programmen](#).

## Laufendes Testen

Es ist wichtig, während der Entwicklung des visuellen Skripts zu überprüfen, ob die zurückgegebenen Ergebnisse Ihren Erwartungen entsprechen. Nicht alle Fehler oder Probleme lassen das Programm sofort fehlschlagen, dies gilt insbesondere für Nullwerte, die sich erst viel später im weiteren Verlauf auswirken können. Diese Vorgehensweise wird auch im Zusammenhang mit Textskripts unter [Vorgehensweisen zur Skripterstellung](#) beschrieben. Das folgende Verfahren hilft Ihnen, sicherzustellen, dass Sie das gewünschte Ergebnis erzielen:

### Überwachen von Daten mit Beobachtungs- und Vorschaublöcken

- Verwenden Sie während der Entwicklung des Programms Beobachtungs- oder Vorschaublocke, um zu **überprüfen, ob wichtige Ausgaben das erwartete Ergebnis zurückgeben**.



Mithilfe der Beobachtungsblöcke werden verglichen:

1. Die Rohwerte der Verschiebungsstrecken
2. Die durch die Sinusgleichung geleiteten Werte Informationen zur Verwendung der Beobachtungsfunktion finden Sie unter [Bibliothek](#).

## Wiederverwendbarkeit sicherstellen

Ihr Programm wird sehr wahrscheinlich irgendwann auch von anderen Benutzern geöffnet werden, selbst wenn Sie unabhängig voneinander arbeiten. Diese Benutzer sollten in der Lage sein, anhand der Ein- und Ausgaben rasch zu bestimmen, was das Programm benötigt und was es produziert. Dies ist besonders bei der Entwicklung benutzerdefinierter Blöcke wichtig, die an die Dynamo-Community weitergegeben und in Programmen anderer Benutzer verwendet werden sollen. Mit diesen Vorgehensweisen erhalten Sie zuverlässige, wiederverwendbare Programme und Blöcke.

## Verwalten der Ein- und Ausgaben

- Für eine optimale Lesbarkeit und Skalierbarkeit sollten Sie **die Ein- und Ausgaben auf ein Minimum beschränken**.
- Versuchen Sie, **eine Strategie zur Entwicklung der Logik zu erarbeiten, indem Sie zunächst einen groben Plan** ihrer Funktionsweise erstellen, bevor Sie den ersten Block im Ansichtsbereich einfügen. Behalten Sie während der Arbeit an diesem Plan im Auge, welche Ein- und Ausgaben in den Skripts verwendet werden sollen.

## Verwenden von Voreinstellungen zum Einbetten von Eingabewerten

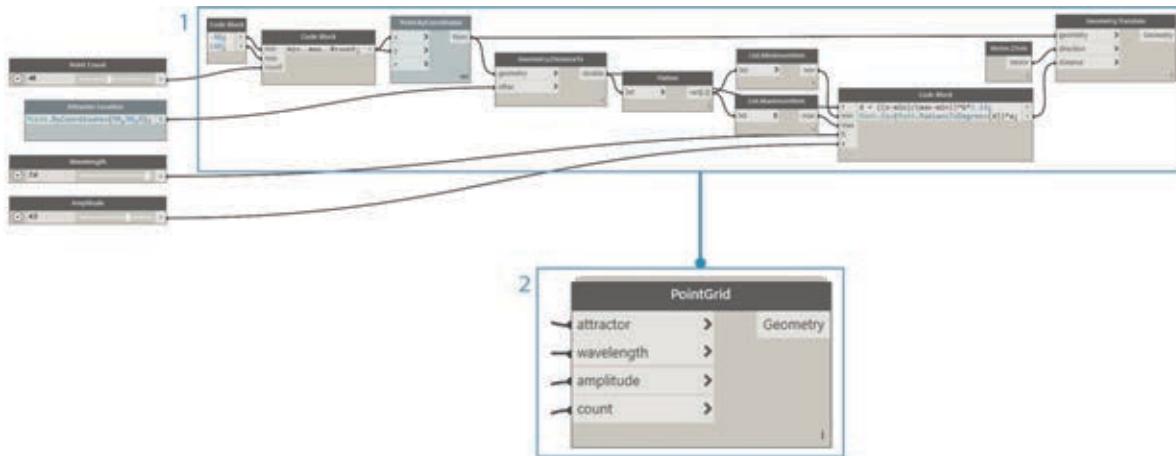
- Falls **bestimmte Optionen oder Bedingungen vorhanden sind, die Sie in das Diagramm einbetten möchten**, empfiehlt es sich, Voreinstellungen für den schnellen Zugriff zu verwenden.
- Mithilfe von Voreinstellungen können Sie darüber hinaus **durch Caching spezifischer Schiebereglerwerte die Komplexität** in Diagrammen mit langen Laufzeiten reduzieren.

Informationen zur Verwendung von Voreinstellungen finden Sie unter [Verwalten von Daten mit Voreinstellungen](#).

## Verwenden von benutzerdefinierten Blöcken als Container für Programme

- Verwenden Sie einen benutzerdefinierten Block, wenn das **Programm in einem einzelnen Container zusammengefasst werden kann**.

- Verwenden Sie einen benutzerdefinierten Block, wenn ein Teil des Diagramms oft in anderen Programmen wiederverwendet werden soll.
- Verwenden Sie einen benutzerdefinierten Block, wenn Sie eine Funktion für die Dynamo-Community bereitstellen möchten.

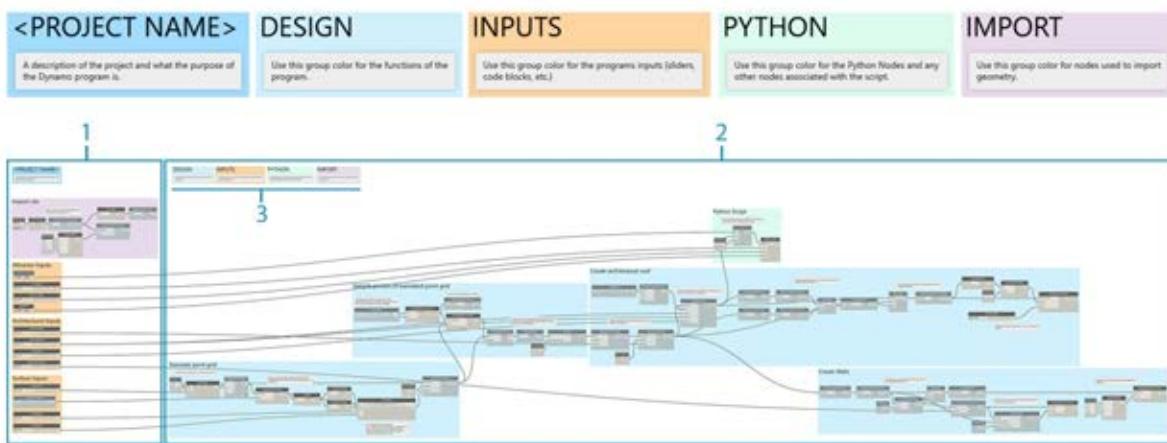


Indem Sie das Programm zur Verschiebung von Punkten in einem benutzerdefinierten Block zusammenfassen, wird dieses zuverlässige, spezielle Programm portierbar und wesentlich leichter verständlich. Aussagekräftige Namen für die Eingabeanschlüsse erleichtern es anderen Benutzern, die Verwendungsweise des Blocks zu verstehen. Achten Sie darauf, für jede Eingabe eine Beschreibung und den erforderlichen Datentyp anzugeben.

1. Bestehendes Programm für Attraktor
2. Benutzerdefinierter Block, in dem dieses Programm, PointGrid, enthalten ist Weitere Informationen zur Verwendung benutzerdefinierter Blöcke finden Sie unter [Einführung zu benutzerdefinierten Blöcken](#).

### Vorlagen erstellen

- Mithilfe von Vorlagen können Sie **Grafikstandards für alle Ihre visuellen Diagramme einrichten, um sie Ihren Kollegen in einheitlicher, verständlicher Weise bereitzustellen.**
- Beim Erstellen einer Vorlage können Sie **Gruppenfarben und Schriftgrößen** standardisieren, um Typen von Arbeitsabläufen oder Datenaktionen zu kategorisieren.
- Sie können beim Erstellen einer Vorlage sogar **Beschriftung, Farbe oder Stil für die Unterscheidung zwischen Frontend- und Backend-Arbeitsabläufen** in Ihrem Diagramm standardisieren.



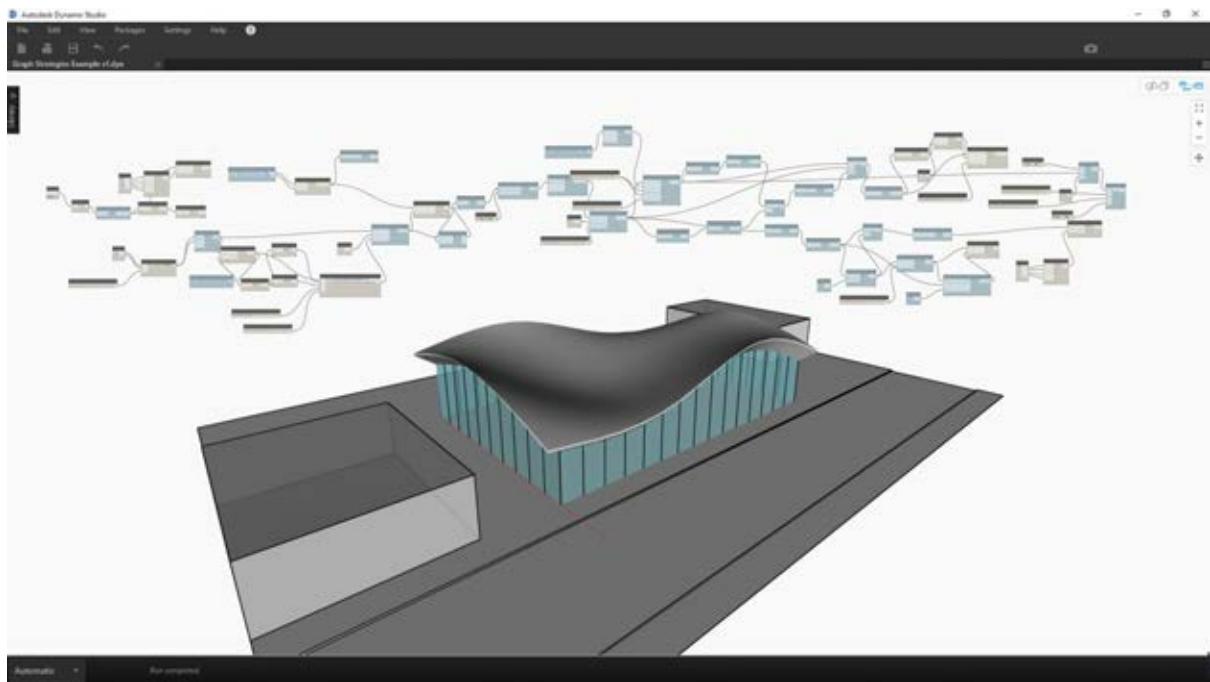
1. Die Benutzeroberfläche (das Frontend) des Programms umfasst den Projektnamen, die Eingabe-Schiebereglern und die Importgeometrie.
2. Backend des Programms.
3. Kategorien für Gruppenfarben (allgemeines Design, Eingaben, Python-Skripts, importierte Geometrie)

### Übung – Dach in der Architektur

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save

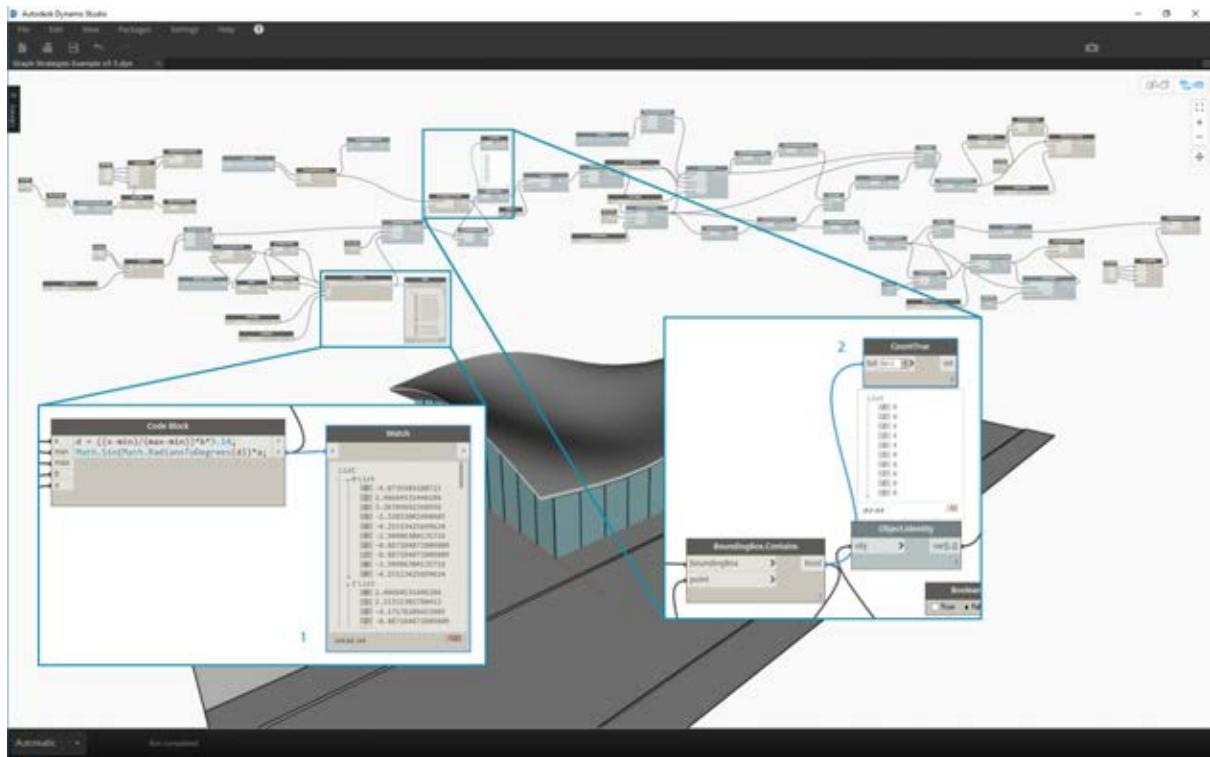
Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [RoofDrainageSim.zip](#)

Sie haben eine Reihe optimaler Verfahren festgelegt und wenden diese jetzt auf ein rasch zusammengestelltes Programm an. Das Programm erstellt zwar wie vorgesehen das Dach, das Diagramm stellt jedoch eher eine „Mind-Map“ des Autors dar. Ihm fehlt die Struktur, und es gibt keine Beschreibung des Verwendungszwecks. Sie ordnen, beschreiben und analysieren das Programm unter Verwendung der optimalen Verfahren so, dass andere Benutzer seine Verwendungsweise verstehen.



Das Programm funktioniert, aber dem Diagramm fehlt Struktur.

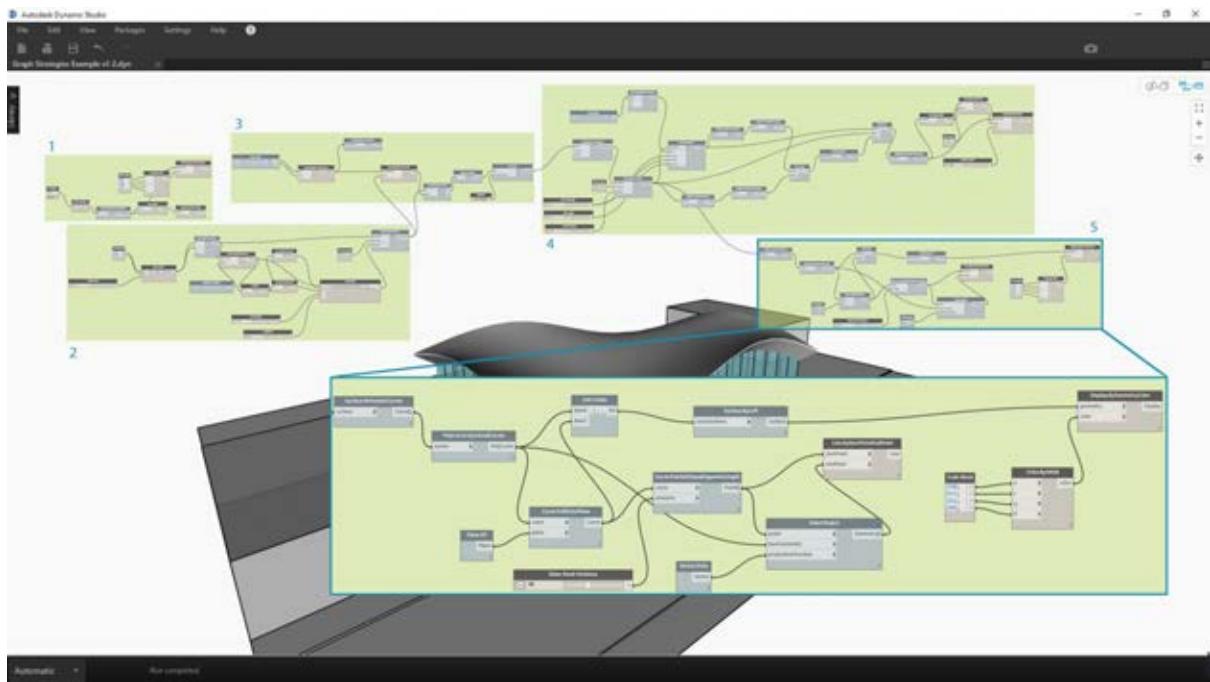
Bestimmen Sie als Erstes die Daten und die Geometrie, die das Programm zurückgibt.



Um logische Unterteilungen, d. h. Modularität zu erzielen, müssen Sie die Stellen kennen, an denen wesentliche Änderungen an den Daten erfolgen. Analysieren Sie den Rest des Programms mithilfe von Beobachtungsböckchen, um festzustellen, ob Gruppen erkennbar sind, bevor Sie mit dem nächsten Schritt fortfahren.

1. Dieser Codeblock mit einer mathematischen Gleichung scheint ein wichtiger Bestandteil des Programms zu sein. Im Beobachtungsblock ist zu erkennen, dass er Listen von Verschiebungsstrecken zurückgibt.
2. Der Zweck dieses Bereichs ist nicht ohne Weiteres ersichtlich. Die Anordnung der True-Werte auf der Listenebene L2 aus BoundingBox.Contains und das Vorhandensein von List.FilterByBoolMask lassen darauf schließen, dass ein Teil des Punktrasters als Beispiel entnommen wird.

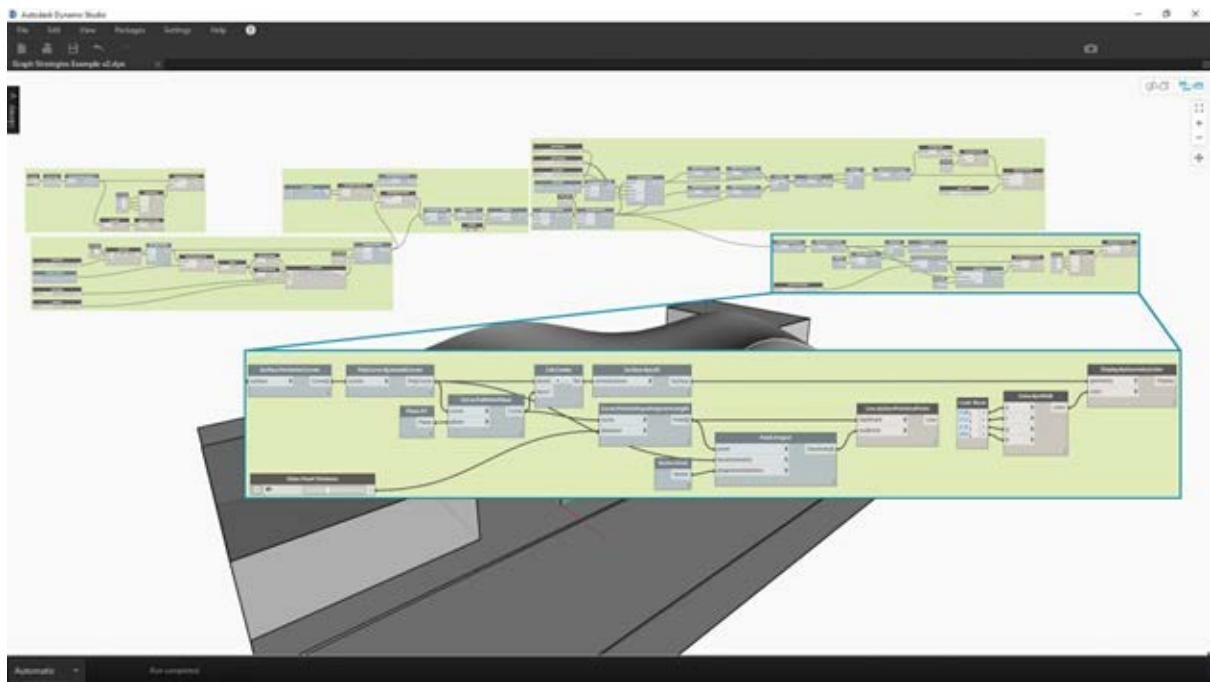
Nachdem Sie die zugrunde liegenden Bestandteile des Programms verstanden haben, fassen Sie sie in Gruppen zusammen.



Gruppen ermöglichen dem Benutzer die visuelle Unterscheidung der Programmbestandteile.

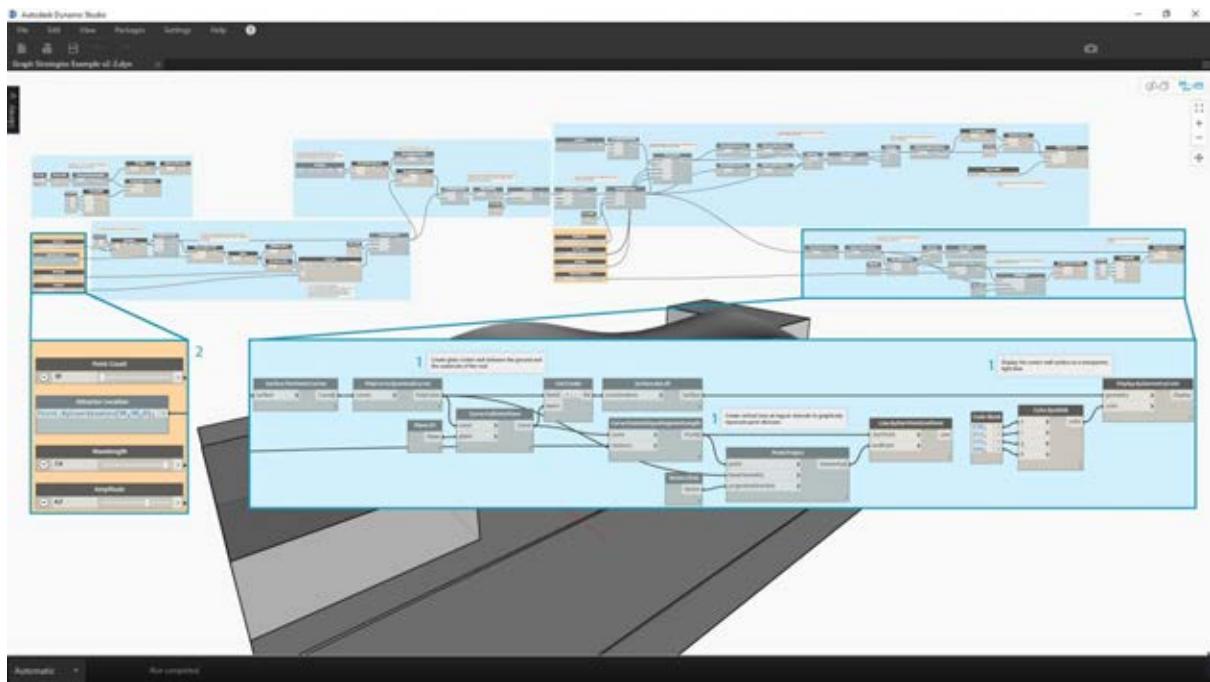
1. 3D-Grundstücksmodell importieren
2. Punkt raster entsprechend der Sinusgleichung verschieben
3. Bestandteil des Punktrasters als Beispiel
4. Dachoberfläche der Architektur erstellen
5. Glasfassade erstellen

Nachdem Sie die Gruppen eingerichtet haben, richten Sie die Blöcke innerhalb des Diagramms auf einheitliche Weise aus.



Eine einheitliche Darstellung macht den Programmablauf und die impliziten Beziehungen zwischen den Blöcken für den Benutzer leichter erkennbar.

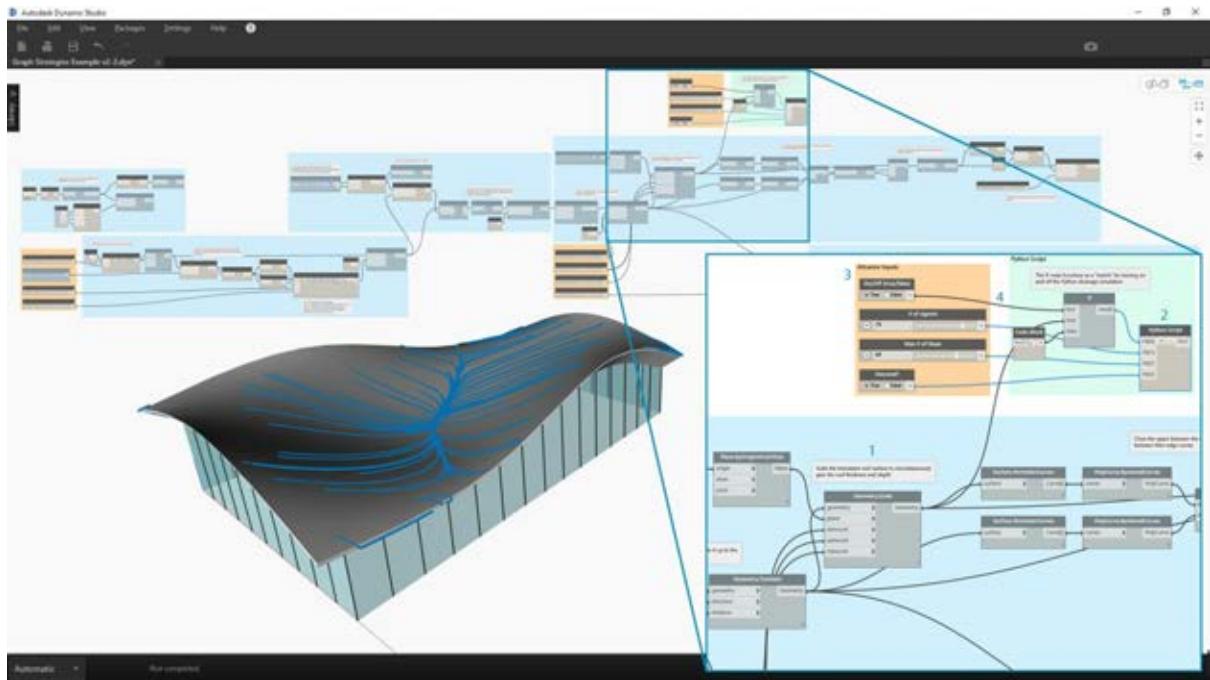
Machen Sie das Programm noch leichter verständlich, indem Sie eine weitere Ebene grafischer Verbesserungen hinzufügen. Fügen Sie Anmerkungen hinzu, mit denen Sie die Funktionsweise eines bestimmten Programmteils beschreiben, geben Sie den Eingaben benutzerdefinierte Namen, und weisen Sie verschiedenen Typen von Gruppen Farben zu.



Diese grafischen Verbesserungen geben dem Benutzer genaueren Aufschluss über den Verwendungszweck des Programms. Die unterschiedlichen Farben der Gruppen helfen bei der Unterscheidung von Eingaben und Funktionen.

1. Anmerkungen
2. Eingaben mit aussagekräftigen Namen

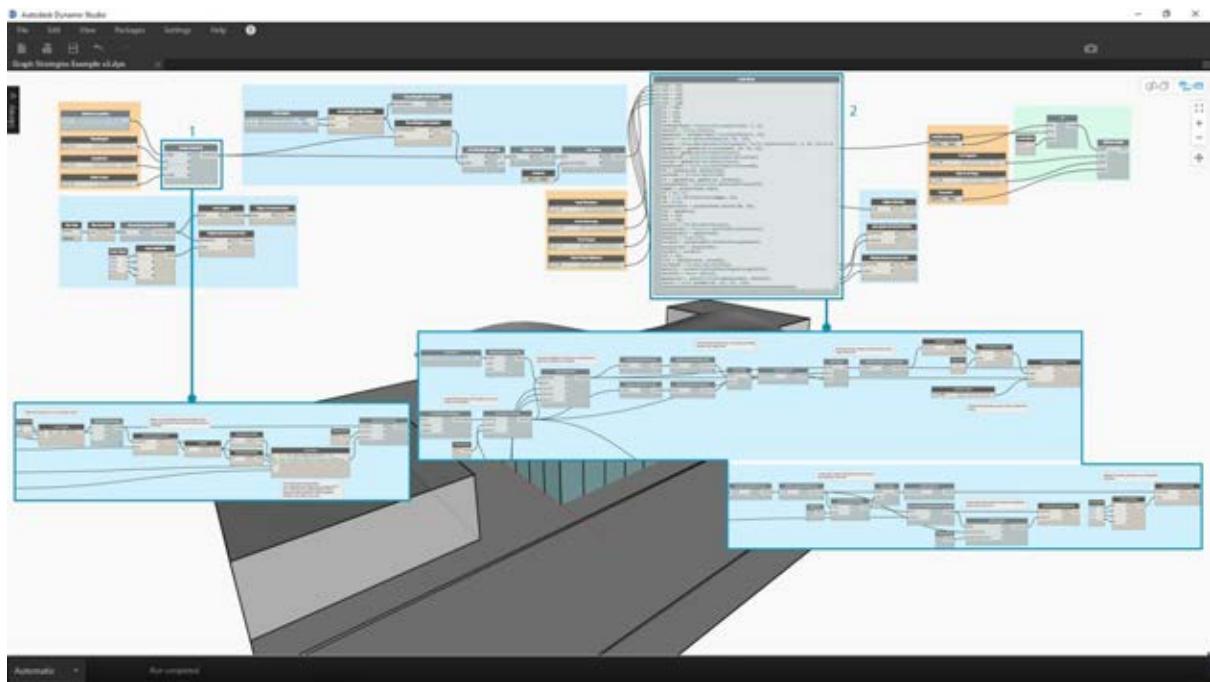
Bevor Sie damit beginnen, das Programm zusammenzufassen, suchen Sie nach einem geeigneten Platz für den Python-Skript-Entwässerungssimulator. Verbinden Sie die Ausgabe der ersten skalierten Dachoberfläche mit der dazugehörigen Skripteingabe.



Durch die Entscheidung, das Skript an dieser Stelle des Programms zu integrieren, wird erreicht, dass die Entwässerungssimulation für die einfache Originaloberfläche des Dachs durchgeführt wird. Diese spezielle Oberfläche wird nicht in der Vorschau angezeigt, aber durch diesen Schritt entfällt die separate Auswahl der oberen Fläche in der gefassten PolySurface.

1. Quellgeometrie für Skripteingabe
2. Python-Block
3. Eingabe-Schieberegler
4. „Schalter“ Ein-Aus

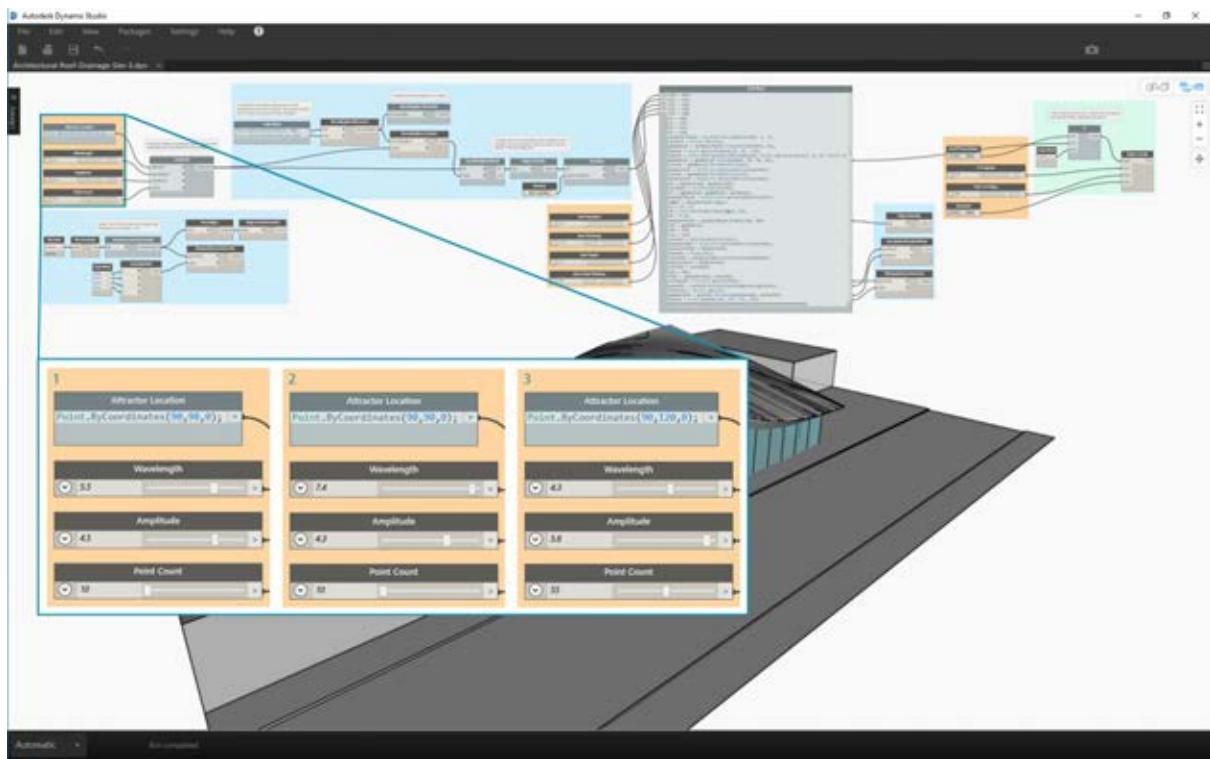
Damit befinden sich alle Elemente an ihrem Platz, und als Nächstes vereinfachen Sie das Diagramm.



Durch Zusammenfassen des Programms mit Block zu Code und benutzerdefinierten Blöcken haben Sie das Diagramm erheblich verkleinert. Die Gruppen für die Erstellung der Dachoberfläche und der Wände wurden in Code konvertiert, da sie für dieses Programm hochspezifisch sind. Die Gruppe zur Verschiebung von Punkten ist in einem benutzerdefinierten Block eingeschlossen, da sie auch in anderen Programmen verwendet werden könnte. Erstellen Sie in der Beispieldatei Ihren eigenen benutzerdefinierten Block aus der Gruppe zur Verschiebung von Punkten.

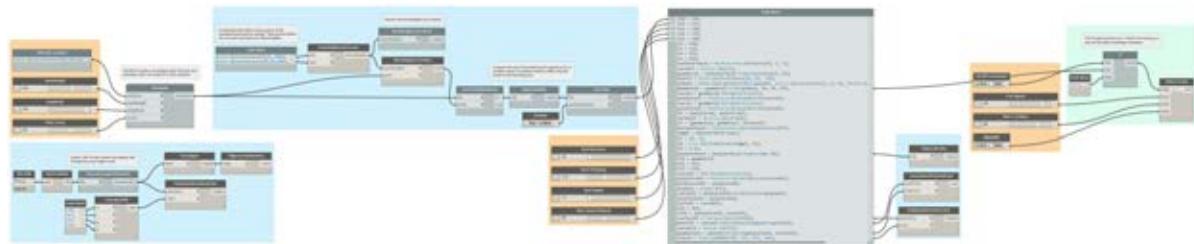
1. Benutzerdefinierter Block als Container für die Gruppe zur Verschiebung von Punkten
2. Block zu Code für die Zusammenfassung der Gruppen zum Erstellen der Oberfläche für das Dach in der Architektur und der Wände

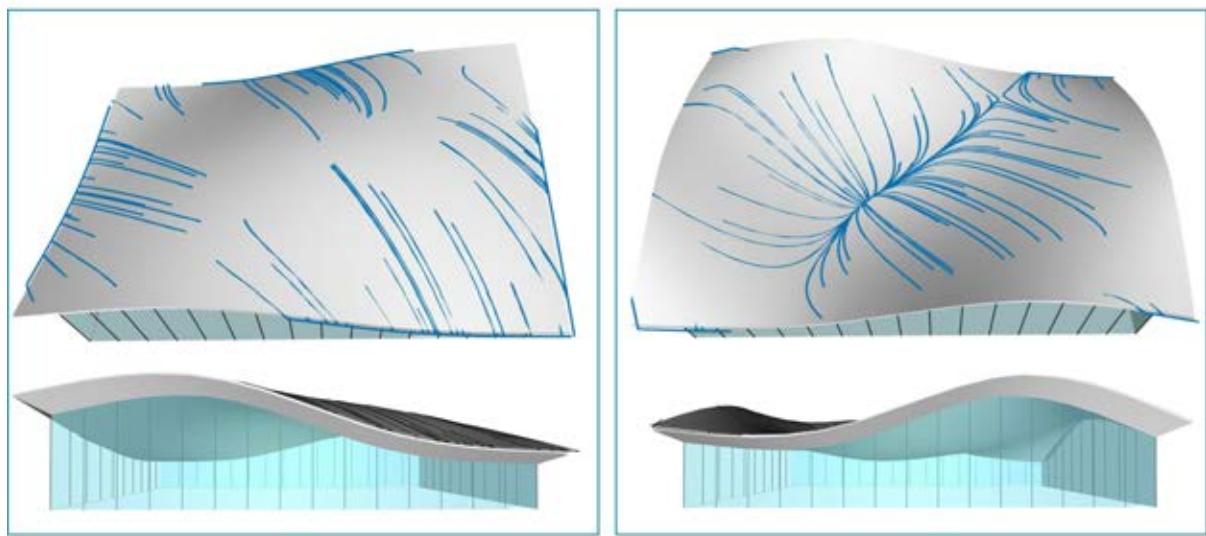
Im letzten Schritt erstellen Sie Voreinstellungen für als Beispiele zu verwendende Dachformen.



Diese Eingaben sind die wesentlichen Angaben zum Steuern der Dachform und geben den Benutzern Hinweise auf die Möglichkeiten des Programms.

Das Programm mit Ansichten zweier Voreinstellungen.





1

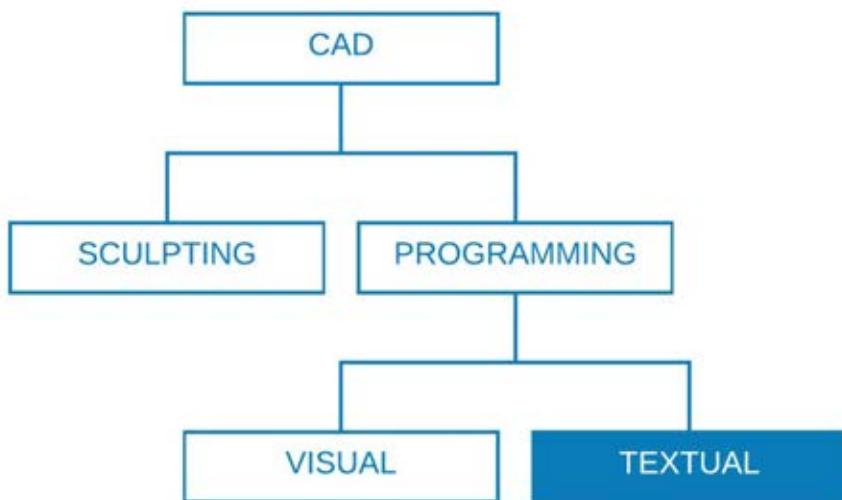
2

Die Muster der Dachentwässerung bieten dem Benutzer eine analytische Ansicht der jeweiligen Voreinstellungen.

# Vorgehensweisen zur Skripterstellung

## Vorgehensweisen zur Skripterstellung

Die textbasierte Skripterstellung innerhalb der Umgebung für visuelle Skripterstellung ermöglicht leistungsstarke und visuelle Beziehungen mithilfe von DesignScript, Python und ZeroTouch (C#). Die Benutzer können in ein und demselben Arbeitsbereich Elemente wie Eingabe-Schieberegler bereitstellen, umfangreiche Abläufe in DesignScript zusammenfassen sowie in Python oder C# auf leistungsstarke Werkzeuge und Bibliotheken zugreifen. Diese Vorgehensweisen können bei effizientem Einsatz die Möglichkeiten zur benutzerdefinierten Anpassung, die Verständlichkeit und die Effizienz des gesamten Programms erheblich steigern. Die im Folgenden beschriebenen Richtlinien helfen Ihnen dabei, Ihre visuellen Skripts um Textskripte zu ergänzen.



### Wann sollten Skripts verwendet werden?

Mit Textskripts können Sie komplexere Beziehungen erstellen als durch visuelle Programmierung, jedoch sind beiden Verfahren auch zahlreiche Funktionen gemeinsam. Dies ist sinnvoll, da Blöcke letzten Endes vorgefertigter Code sind. Es ist wahrscheinlich möglich, ein ganzes Dynamo-Programm in DesignScript oder Python zu schreiben. Die visuellen Skripte kommen jedoch zum Einsatz, weil die Benutzeroberfläche mit Blöcken und Drähten eine intuitive Darstellung des Ablaufs in grafischer Form bietet. Indem Sie sich klarmachen, inwiefern die Möglichkeiten von Textskripts über diejenigen visueller Skripts hinausgehen, erhalten Sie eine wichtige Grundlage für die Entscheidung, wann Textskripts verwendet werden sollten, ohne dabei die intuitive Arbeitsweise mit Blöcken und Drähten aufgeben zu müssen. Die folgenden Richtlinien beschreiben, wann und in welcher Sprache Skripts erstellt werden sollten.

#### Verwenden Sie Textskripts für:

- Schleifen
- Rekursionen
- Zugriff auf externe Bibliotheken

#### Wählen Sie eine Sprache:

| |Schleifen|Rekursionen|Zusammenfassen von Blöcken|Ext. Bibliotheken|Kurzschrifweis| | -- | -- | -- | -- | -- |  
| **DesignScript**|Ja|Ja|Ja|Nein|Ja| **Python**|Ja|Ja|Teilweise|Ja|Nein| **ZeroTouch (C#)**|Nein|Nein|Nein|Ja|Nein|

Unter [Referenz für die Skripterstellung](#) ist aufgelistet, worauf Sie mit welcher Dynamo-Bibliothek zugreifen können.

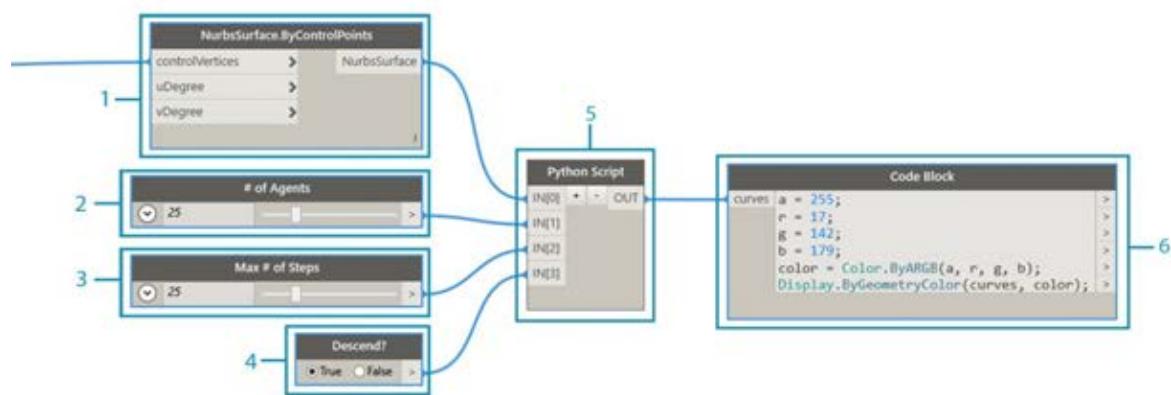
#### Parametrischer Denkansatz

Bei der Skripterstellung in Dynamo, einer zwangsläufig parametrischen Umgebung, ist es sinnvoll, Ihren Code bezogen auf

die Blöcke und Drähte zu strukturieren, in denen er zum Einsatz kommt. Betrachten Sie den Block mit Ihrem Textskript als einen normalen Block wie andere Blöcke im Programm mit spezifischen Eingaben, einer Funktion und einer erwarteten Ausgabe. Dadurch stellen Sie dem Code im Block direkt eine kleine Gruppe von Variablen zur Verarbeitung zur Verfügung, was für ein ordnungsgemäßes parametrisches System entscheidend ist. Im Folgenden finden Sie einige Richtlinien für eine bessere Integration von Code in ein visuelles Programm.

#### Identifizieren der externen Variablen:

- Versuchen Sie, die gegebenen Parameter in der Designaufgabe so festzulegen, dass Sie ein Modell direkt auf Basis dieser Daten konstruieren können.
- Identifizieren Sie die Variablen, bevor Sie mit dem Erstellen von Code beginnen:
- Eine auf das absolut Notwendige beschränkte Gruppe von Eingaben
- Die beabsichtigte Ausgabe
- Konstanten



Vor dem Schreiben des Codes wurden mehrere Variablen festgelegt.

1. Die in der Simulation dem Regen ausgesetzte Oberfläche.
2. Die gewünschte Anzahl Regentropfen (Agents).
3. Die von den Regentropfen zurückgelegte Strecke.
4. Umschalten zwischen steilstmöglichem Pfad und Überqueren der Oberfläche.
5. Python-Block mit der entsprechenden Anzahl Eingaben.
6. Ein Codeblock, mit dem die zurückgegebenen Kurven blau gefärbt werden.

#### Entwerfen der internen Beziehungen:

- Das parametrische Prinzip ermöglicht die Bearbeitung bestimmter Parameter oder Variablen, um das Endergebnis einer Gleichung zu ändern oder zu beeinflussen.
- Versuchen Sie stets, Objekte in Ihrem Skript, die logisch miteinander verbunden sind, als Funktionen voneinander zu definieren. Auf diese Weise wird bei einer Änderung eines der Objekte auch das andere proportional dazu aktualisiert.
- Beschränken Sie die Anzahl der Eingaben, indem Sie nur die wichtigsten Parameter bereitstellen:
- Wenn eine Gruppe von Parametern aus Parametern auf einer höheren Hierarchieebene abgeleitet werden kann, stellen Sie nur die übergeordneten Parameter als Skripteingaben bereit. Auf diese Weise vereinfachen Sie die Benutzeroberfläche des Skripts und machen dieses dadurch benutzerfreundlicher.

```

1 import clr
2 clr.AddReference('ProtoGeometry')
3 from Autodesk.DesignScript.Geometry import *
4
5 solid = IN[0]
6 seed = IN[1]
7 xCount = IN[2]
8 yCount = IN[3]
9
10 solids = []
11 yDist = solid.BoundingBox.MaxPoint.Y - solid.BoundingBox.MinPoint.Y
12 xDist = solid.BoundingBox.MaxPoint.X - solid.BoundingBox.MinPoint.X
13
14 for i in xRange:
15     for j in yRange:
16         fromCoord = solid.ContextCoordinateSystem
17         toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin, Vector.ByCoordinates((0,0,1),(0*(i+j*xVal),0)))
18         vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
19         toCoord = toCoord.Translate(vec)
20         solids.append(solid.Transform(fromCoord,toCoord))
21
22 OUT = solids

```

The screenshot shows a Python script editor window with three numbered callouts pointing to specific parts of the code:

- Callout 1:** Points to the first three lines of the script.
- Callout 2:** Points to the calculation of `yDist` and `xDist`.
- Callout 3:** Points to the nested loop structure where a coordinate system is rotated and translated to create multiple instances of the input solid.

Die Codemodule aus dem Beispiel unter [Python-Block](#).

1. Eingaben.
2. Interne Variablen für das Skript.
3. Eine Schleife, die diese Eingaben und Variablen für ihre Funktion nutzt. Tipp: Wenden Sie für den Prozess dieselbe Sorgfalt an wie für die Lösung.

#### Vermeiden von Wiederholungen (das DRY-Prinzip – Don't repeat yourself):

- Falls es mehrere Möglichkeiten für denselben Vorgang in Ihrem Skript gibt, werden die doppelt vorhandenen Darstellungen schließlich asynchron, was die Wartung extrem schwierig macht, die Faktorisierung verschlechtert und interne Widersprüche verursacht.
- Das DRY-Prinzip besagt, dass jede für jede Information genau eine eindeutige und maßgebliche Darstellung im System vorhanden sein muss.
- Wird dieses Prinzip erfolgreich angewendet, erfolgen Änderungen an allen verbundenen Elementen in Ihrem Skript in vorhersehbarer und einheitlicher Weise und Elemente, die nicht miteinander verbunden sind, haben keine logischen Auswirkungen aufeinander.

```

### BAD
for i in range(4):
for j in range(4):
point = Point.ByCoordinates(3*i, 3*j, 0)
points.append(point)

### GOOD
count = IN[0]
pDist = IN[1]

for i in range(count):
for j in range(count):
point = Point.ByCoordinates(pDist*i, pDist*j, 0)
points.append(point)

```

Tipp: Bevor Sie Objekte in Ihrem Skript duplizieren (wie die Konstante im obigen Beispiel), überlegen Sie, ob Sie stattdessen eine Verknüpfung zur Quelle erstellen können.

#### Modulares Strukturieren

Mit zunehmender Länge und Komplexität des Codes ist das Grundkonzept bzw. der Gesamタルgorithmus immer schwieriger zu entziffern. Es wird zudem schwieriger, zu verfolgen, welche spezifischen Vorgänge ablaufen (und wo dies geschieht), bei Fehlfunktionen den Fehler zu finden, anderen Code zu integrieren und Entwicklungsaufgaben zuzuweisen. Um diese Probleme zu vermeiden, ist es sinnvoll, Code in Modulen zu erstellen: Bei diesem Organisationsverfahren wird Code anhand der jeweils ausgeführten Aufgaben aufgegliedert. Im Folgenden finden Sie einige Tipps, die es Ihnen erleichtern sollen, durch Modularisierung leichter zu verwaltende Skripts zu erstellen.

#### Schreiben von Code in Modulen:

- Ein Modul ist eine Sammlung von Code, der eine bestimmte Aufgabe ausführt, ähnlich wie ein Dynamo-Block im Arbeitsbereich.
- Dabei kann es sich um beliebigen Code handeln, der visuell von angrenzendem Code abgetrennt sollte (eine Funktion, eine Klasse, eine Gruppe von Eingaben oder die von Ihnen importierten Bibliotheken).
- Durch die Entwicklung von Code in Form von Modulen können Sie sowohl die visuellen, intuitiven Eigenschaften von Blöcken als auch die komplexen Beziehungen nutzen, die nur mit Textskripts zu erreichen sind.

```

69 # INITIALIZE AGENTS
70 agents = []
71 for i in range(numAgents):
72     u = float(random.randrange(1000))/1000
73     v = float(random.randrange(1000))/1000
74     agent = Agent(u,v)
75     agents.append(agent)
76
77
78 # UPDATE AGENTS
79 for i in range(maxSteps):
80     for eachAgent in agents:
81         eachAgent.update()
82
83 # DRAW TRAILS
84 trails = []
85 for eachAgent in agents:
86     trailPts = eachAgent.trailPts
87     if (len(trailPts) > 1):
88         trail = PolyCurve.ByPoints(trailPts)
89         trails.append(trail)
90
91 # OUTPUT TRAILS
92 OUT = trails

```

Accept Changes Cancel

Diese Schleifen rufen eine Klasse namens „agent“ auf, die in dieser Übung entwickelt wird.

1. Ein Code-Modul, das den Startpunkt jedes Agents definiert.
2. Ein Code-Modul, das den Agent aktualisiert.
3. Ein Code-Modul, das den Pfad zeichnet, dem der Agent folgt.

#### Erkennen der Wiederverwendung von Code:

- Wenn Sie feststellen, dass Ihr Code dieselbe (oder eine sehr ähnliche) Aufgabe an mehr als einer Stelle ausführt, suchen Sie nach Möglichkeiten, ihn zu einer Funktion zusammenzufassen, die aufgerufen werden kann.
- „Manager“-Funktionen steuern den Programmablauf und enthalten in erster Linie Aufrufe an „Worker“-Funktionen, die Details auf unteren Ebenen, etwa das Verschieben von Daten zwischen Strukturen, verarbeiten.



In diesem Beispiel werden Kugeln erstellt, deren Radien und Farben vom Z-Wert ihrer Mittelpunkte abhängig sind.

1. Zwei übergeordnete „Worker“-Funktionen zum Erstellen von Kugeln mit Radien und mit Anzeigefarben anhand des Z-Werts des Mittelpunkts.
2. Eine übergeordnete „Manager“-Funktion, die die beiden „Worker“-Funktionen kombiniert. Durch Aufrufen dieser Funktion werden beide darin enthaltenen Funktionen aufgerufen.

#### Beschränken der Anzeige auf das Nötige:

- Die Schnittstelle eines Moduls gibt die vom Modul bereitgestellten und benötigten Elemente an.
- Nachdem Sie die Schnittstellen zwischen den Einheiten definiert haben, kann die detaillierte Entwicklung jeder Einheit separat erfolgen.

#### Möglichkeit zum Trennen bzw. Ersetzen:

- Module erkennen oder berücksichtigen einander nicht.

#### Allgemeine Formen der Modularisierung :

- Codegruppierung:

```
# IMPORT LIBRARIES
```

```

import random
import math
import clr
clr.AddReference('ProtoGeometry')
from Autodesk.DesignScript.Geometry import *

# DEFINE PARAMETER INPUTS
surfIn = IN[0]
maxSteps = IN[1]

• Funktionen:

def get_step_size():
area = surfIn.Area
stepSize = math.sqrt(area)/100
return stepSize

stepSize = get_step_size()

• Klassen:

class MyClass:
i = 12345

def f(self):
return 'hello world'

numbers = MyClass.i
greeting = MyClass.f

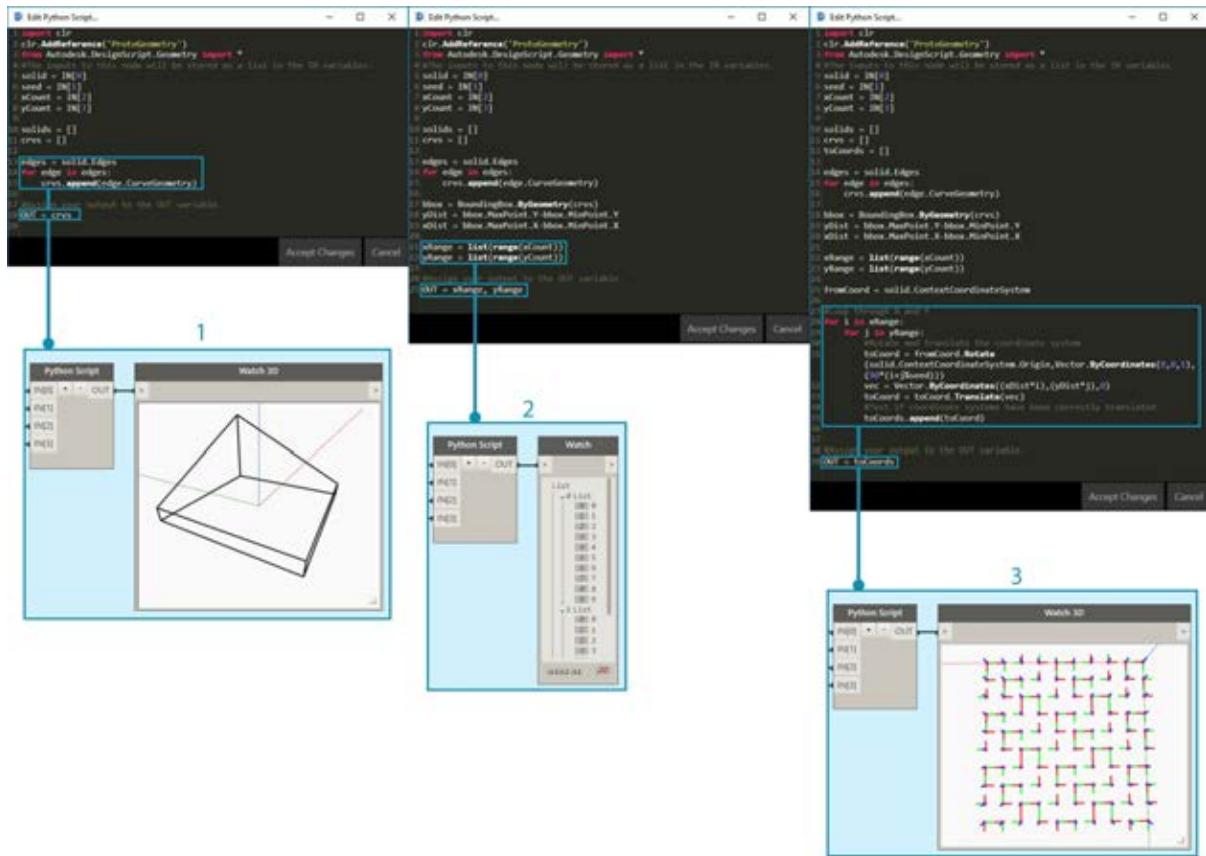
```

## Laufendes Testen

Es ist sinnvoll, während der Entwicklung von Textskripts in Dynamo laufend zu überprüfen, ob die tatsächlich erstellten Funktionen Ihren Erwartungen entsprechen. Dadurch stellen Sie sicher, dass unvorhergesehene Ereignisse wie Syntaxfehler, logische Diskrepanzen, falsche Werte, regelwidrige Ausgaben usw. nicht erst zum Schluss alle zusammen, sondern direkt bei ihrem Auftreten schnell erkannt und korrigiert werden. Da die Textskripts sich innerhalb der Blöcke im Ansichtsbereich befinden, sind sie bereits in den Datenfluss des visuellen Programms integriert. Die laufende Überwachung des Skripts gestaltet sich dadurch so einfach wie das Zuweisen von Daten für die Ausgabe, die Ausführung des Programms und die Auswertung des Skriptergebnisses über einen Beobachtungsblock. Im Folgenden finden Sie einige Tipps für die laufende Überwachung Ihrer Skripts, während Sie sie konstruieren.

### Testen Sie während der Entwicklung:

- Wenn Sie eine Gruppe von Funktionen erstellt haben:
- Überprüfen Sie Ihren Code aus distanzierter Sicht.
- Seien Sie dabei kritisch. Ist die Funktionsweise für einen Teamkollegen verständlich? Brauche ich das wirklich? Kann diese Funktion effizienter durchgeführt werden? Werden unnötige Duplikate oder Abhängigkeiten erstellt?
- Führen Sie rasch Tests durch, um sich zu überzeugen, dass plausible Daten zurückgegebenen werden.
- Weisen Sie die aktuellsten Daten, mit denen Sie arbeiten, in Ihrem Skript als Ausgabedaten zu, damit der Block bei einer Aktualisierung des Skripts immer relevante Daten ausgibt:



Testen des Beispielcodes aus dem [Python-Block](#).

1. Überprüfen Sie, ob alle Kanten des Volumenkörpers als Kurven zurückgegeben werden, damit ein Begrenzungsrahmen darum erstellt wird.
2. Überprüfen Sie, ob die Count-Eingaben in Ranges konvertiert werden.
3. Überprüfen Sie, ob Koordinatensysteme in dieser Schleife ordnungsgemäß verschoben und gedreht wurden.

#### Vorwegnehmen von Grenzfällen:

- Geben Sie während der Skripterstellung die Mindest- und Höchstwerte der Eingabeparameter innerhalb ihrer zugewiesenen Domäne an, um zu testen, ob das Programm auch unter extremen Bedingungen funktioniert.
- Überprüfen Sie auch dann, wenn das Programm mit seinen Extremwerten funktioniert, ob es unbeabsichtigte Nullwerte oder leere Werte zurückgibt.
- Bugs und Fehler, die auf grundlegende Probleme mit dem Skript hinweisen, werden zuweilen nur in solchen Grenzfällen erkennbar.
- Ermitteln Sie die Fehlerursache, und entscheiden Sie, ob sie intern behoben werden muss oder ob zur Vermeidung des Problems eine Parameterdomäne neu definiert werden muss.

Tipp: Gehen Sie stets davon aus, dass die Benutzer jede mögliche Kombination sämtlicher für sie bereitgestellten Eingabewerte verwenden werden. Auf diese Weise vermeiden Sie unangenehme Überraschungen.

#### Effiziente Fehlersuche

Debugging ist der Prozess der Beseitigung von Fehlern („Bugs“) in Ihrem Skript. Bugs können Fehler, Ineffizienzen, Ungenauigkeiten oder beliebige nicht beabsichtigte Ergebnisse sein. Um einen Bug zu beheben, kann ein einfacher Schritt wie die Korrektur eines falsch geschriebenen Variablenamens genügen, es können jedoch auch tiefergehende, die Struktur des Skripts betreffende Probleme vorhanden sein. Im Idealfall erkennen Sie solche potenziellen Probleme frühzeitig, indem Sie das Skript während des Erstellens testen, eine Garantie für Fehlerfreiheit ist dadurch jedoch nicht gegeben. Im Folgenden werden einige der oben genannten optimalen Verfahren genauer beschrieben, um Ihnen die systematische Beseitigung von Bugs zu erleichtern.

#### Verwenden Sie den Beobachtungsblock:

- Überprüfen Sie die an verschiedenen Stellen des Codes zurückgegebenen Daten, indem Sie ihn ähnlich wie beim Testen des Programms der OUT-Variablen zuweisen:

#### Schreiben Sie aussagekräftige Kommentare:

- Das Debugging eines Codemoduls ist erheblich einfacher, wenn das beabsichtigte Ergebnis klar beschrieben ist.

```
# Loop through X and Y
for i in range(xCount):
    for j in range(yCount):

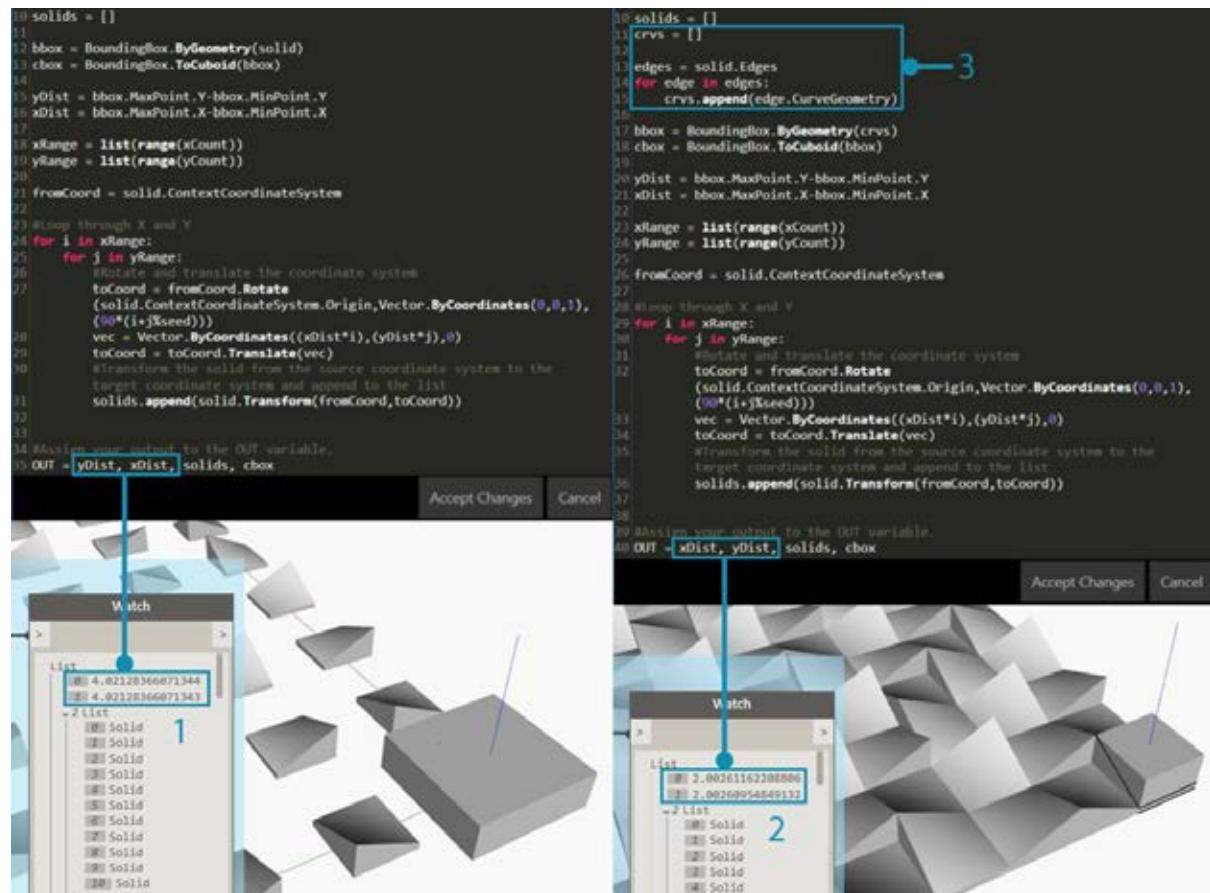
        # Rotate and translate the coordinate system
        toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin, Vector.ByCoordinates
        vec = Vector.ByCoordinates((xDist*i),(yDist*j),0)
        toCoord = toCoord.Translate(vec)

        # Transform the solid from the source coord system to the target coord system and ap
        solids.append(solid.Transform(fromCoord,toCoord))
```

Normalerweise wäre dies ein Übermaß an Kommentaren und leeren Zeilen, beim Debugging kann es jedoch hilfreich sein, den Code in überschaubare Abschnitte aufzuteilen.

#### Nutzen Sie die Modularität des Codes:

- Die Ursache eines Problems kann gezielt auf bestimmte Module zurückgeführt werden.
- Nachdem Sie das fehlerhafte Modul identifiziert haben, lässt sich das Problem wesentlich leichter beheben.
- Wenn ein Programm geändert werden muss, ist Code, der in Form von Modulen entwickelt wurde, erheblich leichter zu ändern.
- Sie können neue oder korrigierte Module in ein bestehendes Programm einfügen und sich dabei darauf verlassen, dass der Rest des Programms unverändert bleibt.



Debuggen der Beispieldatei aus dem [Python-Block](#).

1. Die Eingabegeometrie gibt einen Begrenzungsrahmen zurück, der größer ist als sie selbst, wie durch Zuweisen von xDist und yDist zu OUT zu erkennen ist.
2. Die Kantenkurven der Eingabegeometrie geben einen passenden Begrenzungsrahmen mit den richtigen Entfernungen für xDist und yDist zurück.
3. Das zur Behebung des Problems mit den xDist- und yDist-Werten eingefügte Codemodul.

## Übung – Steilstmöglicher Pfad

Laden Sie die zu dieser Übungslektion gehörige Beispieldatei herunter (durch Rechtsklicken und Wahl der Option Save Link As). Eine vollständige Liste der Beispieldateien finden Sie im Anhang. [SteepestPath.dyn](#)

In dieser Übung schreiben Sie unter Beachtung der optimalen Verfahren für Textskripts ein Skript zur Simulation von Regen. Im Kapitel „Strategien für Diagramme“ war es möglich, optimale Verfahren auf ein unstrukturiertes visuelles Programm anzuwenden. Bei Textskripts ist dies jedoch wesentlich schwieriger. Logische Beziehungen, die in Textskripts erstellt werden, sind weniger sichtbar und können in unzureichend strukturiertem Code kaum unterschieden werden. Die Leistungsfähigkeit der Textskripts bringt einen größeren Organisationsbedarf mit sich. Hier werden die einzelnen Schritte gezeigt und während des ganzen Vorgangs die optimalen Verfahren angewendet.



Das auf eine durch einen Attraktor verformte Oberfläche angewendete Skript.

Als Erstes müssen Sie die benötigten Dynamo-Bibliotheken importieren. Indem Sie dies zu Anfang durchführen, erhalten Sie globalen Zugriff auf Dynamo-Funktionen in Python.



Sie müssen hier sämtliche Bibliotheken importieren, die Sie verwenden möchten.

Anschließend müssen Sie die Eingaben und die Ausgabe definieren, die als Eingabeanschlüsse am Block angezeigt werden. Diese externen Eingaben sind die Grundlage für das Skript und der Schlüssel zum Erstellen einer parametrischen Umgebung.



Sie müssen Eingaben definieren, die Variablen im Python-Skript entsprechen, und die gewünschte Ausgabe bestimmen:

1. Die Oberfläche, über die die Bewegung nach unten erfolgen soll.
2. Die Anzahl der Agents, die sich bewegen sollen.
3. Die maximale Anzahl an Schritten, die die Agents ausführen dürfen.
4. Eine Option, mit der entweder der kürzeste Weg abwärts auf der Oberfläche oder eine seitliche Bewegung gewählt werden kann.
5. Der Python-Block mit Eingabe-IDs, die den Eingaben im Skript (IN[0], IN[1]) entsprechen.
6. Ausgabekurven, die in einer anderen Farbe angezeigt werden können.

Erstellen Sie jetzt unter Beachtung des Modularitätsprinzips den Hauptteil des Skripts. Die Simulation des kürzesten Wegs abwärts auf einer Oberfläche ausgehend von unterschiedlichen Startpunkten ist eine größere Aufgabe, die mehrere Funktionen erfordert. Anstatt die verschiedenen Funktionen im Verlauf des ganzen Skripts aufzurufen, können Sie den Code modularisieren, indem Sie sie in einer einzigen Klasse, dem Agent, sammeln. Die verschiedenen Funktionen dieser Klasse – des „Moduls“ – können über unterschiedliche Variablen aufgerufen oder sogar in einem anderen Skripts wiederverwendet werden.



Sie müssen eine Klasse bzw. eine Vorlage für einen Agent definieren, der eine Abwärtsbewegung über eine Oberfläche darstellt, wobei bei jedem Schritt der Bewegung jeweils die Richtung mit der maximalen Neigung gewählt wird:

1. Name.
2. Globale, allen Agents gemeinsame Attribute.
3. Exemplarattribute, die für jeden Agent spezifisch sind.
4. Eine Funktion zum Ausführen eines Schritts.
5. Eine Funktion zum Katalogisieren der Position jedes Schritts in einer Liste für den Pfad.

Initialisieren Sie die Agents, indem Sie ihre Startpositionen definieren. Dies ist eine gute Gelegenheit, das Skript zu testen und sicherzustellen, dass die Agent-Klasse funktioniert.



Sie müssen alle Agents, deren Abwärtsbewegung über die Oberfläche Sie beobachten möchten, instanziieren und ihre Anfangsattribute definieren.

1. Eine neue, leere Pfadliste.
2. Ausgangspunkt der Bewegung über die Oberfläche.
3. Die Liste der Agents wird als Ausgabe zugewiesen, um zu überprüfen, was das Skript hier zurückgibt. Die richtige Anzahl der Agents wird zurückgegeben, das Skript muss jedoch später erneut getestet werden, um die zurückgegebene Geometrie zu prüfen.

Aktualisieren Sie die einzelnen Agents bei jedem Schritt.



Anschließend müssen Sie eine verschachtelte Schleife starten, wobei die Position für jeden Agent und jeden Schritt aktualisiert und in der jeweiligen Pfadliste aufgezeichnet wird. Bei jedem Schritt wird außerdem geprüft, ob der Agent einen Punkt auf der Oberfläche erreicht hat, von dem aus keine weiteren Abwärtsschritte mehr möglich sind. Ist diese Bedingung erfüllt, wird die Bewegung des Agent beendet.

Nachdem die Agents vollständig aktualisiert sind, können Sie Geometrie ausgeben, durch die sie dargestellt werden.



Nachdem alle Agents entweder den Endpunkt ihrer Abwärtsbewegung erreicht oder die maximale Anzahl Schritte ausgeführt haben, erstellen Sie eine Polykurve durch die Punkte in ihrem Pfad und geben die Polykurvenpfade aus.

Das Skript zur Ermittlung der steilstmöglichen Pfade.



1. Voreinstellung zur Simulation von Regen auf der zugrunde liegenden Oberfläche.
2. Die Agents können umgeschaltet werden, sodass sie nicht die steilstmöglichen Pfade suchen, sondern die Oberfläche überqueren.



Das vollständige Python-Textskript.

# Referenz für die Skripterstellung

## Referenz für die Skripterstellung

Diese Referenzseite bietet ausführlichere Informationen zu Codebibliotheken, Beschriftungen und Stil aus den optimalen Verfahren, die unter „Vorgehensweisen zur Skripterstellung“ beschrieben wurden. Zur Verdeutlichung der im Folgenden beschriebenen Konzepte wird hier Python verwendet, es gelten jedoch dieselben Prinzipien sowohl in Python als auch in C# (Zerotouch), allerdings mit unterschiedlicher Syntax.

### Wahl geeigneter Bibliotheken

Standardbibliotheken sind außerhalb von Dynamo verfügbar und liegen in den Programmiersprachen Python und C# (Zerotouch) vor. Dynamo verfügt darüber hinaus über eigene Bibliotheken, die direkt seiner Blockhierarchie entsprechen und es dem Benutzer ermöglichen, alle Abläufe, die aus Blöcken und Drähten konstruiert werden können, auch in Form von Code zu erstellen. Der folgende Leitfaden zeigt, worauf Sie in den einzelnen Dynamo-Bibliotheken Zugriff erhalten und wann Sie eine Standardbibliothek verwenden sollten.



### Standardbibliotheken und Dynamo-Bibliotheken

- Standardbibliotheken in Python und C# können zum Erstellen erweiterter Daten- und Ablaufstrukturen in der Dynamo-Umgebung genutzt werden.
- Dynamo-Bibliotheken entsprechen direkt der Blockhierarchie zum Erstellen von Geometrie und anderer Dynamo-Objekte.

### Dynamo-Bibliotheken

- ProtoGeometry
- Funktionen: Bogen, Begrenzungsrahmen, Kreis, Kegel, Koordinatensystem, Quader, Kurve, Zylinder, Kante, Ellipse, elliptischer Bogen, Fläche, Geometrie, Spirale, Indexgruppe, Linie, Netz, NURBS-Kurve, NURBS-Oberfläche, Ebene, Punkt, Polygon, Rechteck, Volumenkörper, Kugel, Oberfläche, Topologie, TSpline, UV, Vektor, Scheitelpunkt.
- Importverfahren: `import Autodesk.DesignScript.Geometry`
- **Beachten Sie bei der Verwendung von ProtoGeometry in Python oder C#, dass Sie hierbei nicht verwaltete Objekte erstellen, deren Speicher manuell verwaltet werden muss, wie weiter unten im Abschnitt Nicht verwaltete Objekte genauer beschrieben.**
- DSCoreNodes
- Funktionen: Farbe, Farbbereich 2D, Datum und Uhrzeit, Zeitraum, IO, Formel, Logik, Liste, mathematische Funktionen, Quadtree, Zeichenfolge, Thread.
- Importverfahren: `import DSCore`
- Tessellieren
- Funktionen: konvexe Hülle, Delaunay, Voronoi.
- Importverfahren: `import Tessellation`
- DSOFFice
- Funktion: Excel.
- Importverfahren: `import DSOFFice`

### Sorgfältige Kennzeichnung

Beim Erstellen von Skripts werden Elemente wie Variablen, Typen, Funktionen und andere Objekte laufend mit IDs gekennzeichnet. Durch dieses System der symbolischen Schreibweise können Sie beim Entwickeln von Algorithmen ganz einfach über Beschriftungen, die in der Regel aus einer Folge von Zeichen bestehen, auf Informationen verweisen. Die aussagekräftige Benennung von Elementen spielt eine wichtige Rolle beim Erstellen von Code, den sowohl andere Benutzer als auch Sie selbst zu einem späteren Zeitpunkt problemlos lesen und verstehen können. Beachten Sie beim Benennen von

Elementen in Ihrem Skript die folgenden Tipps:

**Sie können Abkürzungen verwenden, müssen diese jedoch in einem Kommentar erläutern:**

```
### BAD
csfX = 1.6
csfY= 1.3
csfZ = 1.0

### GOOD
# column scale factor (csf)
csfX = 1.6
csfY= 1.3
csfZ = 1.0
```

**Vermeiden Sie überzählige Beschriftungen:**

```
### BAD
import car
seat = car.CarSeat()
tire = car.CarTire()

### GOOD
import car
seat = car.Seat()
tire = car.Tire()
```

**Verwenden Sie für Variablennamen positive anstatt negativer Logik :**

```
### BAD
if 'mystring' not in text:
print 'not found'
else:
print 'found'
print 'processing'

### GOOD
if 'mystring' in text:
print 'found'
print 'processing'
else:
print 'not found'
```

**Geben Sie der „Rückwärtsschreibweise“ den Vorzug:**

```
### BAD
agents = ...
active_agents = ...
dead_agents ...

### GOOD
agents = ...
agents_active = ...
agents_dead = ...
```

Dies ist unter dem Aspekt der Struktur sinnvoller.

**Verwenden Sie Aliases zur Verkürzung überlanger und häufig wiederholter Ketten:**

```
### BAD
from RevitServices.Persistence import DocumentManager

DocumentManager = DM

doc = DM.Instance.CurrentDBDocument
uiapp = DM.Instance.CurrentUIApplication

### GOOD
```

```
from RevitServices.Persistence import DocumentManager as DM  
  
doc = DM.Instance.CurrentDBDocument  
uiapp = DM.Instance.CurrentUIApplication
```

Aliases führen rasch zu verwirrenden und nicht standardmäßigen Programmen.

#### Verwenden Sie nur die erforderlichen Wörter:

```
### BAD  
rotateToCoord = rotateFromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.E  
  
### GOOD  
toCoord = fromCoord.Rotate(solid.ContextCoordinateSystem.Origin,Vector.ByCoordinates
```

„Man muss die Dinge so einfach wie möglich machen. Aber nicht einfacher.“ – Albert Einstein

#### Einheitlicher Stil

Im Allgemeinen gibt es beim Programmieren von Anwendungen jeder Art mehrere Möglichkeiten. Ihr „persönlicher Stil“ beim Schreiben von Skripts ist daher das Ergebnis zahlloser Detailentscheidungen für oder gegen einzelne Schritte während der Arbeit. Wie leserlich und leicht zu warten Ihr Code ist, ist dennoch gleichermaßen das direkte Ergebnis seiner internen Kohärenz und der Einhaltung allgemeiner Stilkonventionen. Als Faustregel gilt, dass Code, der an zwei unterschiedlichen Stellen gleich aussieht, auch dieselbe Funktion ausführen muss. Die folgenden Tipps sollen beim Schreiben von verständlichem und einheitlichem Code helfen.

**Namenskonventionen:** (Wählen Sie eine der folgenden Konventionen für jede Art von Element in Ihrem Code und behalten Sie sie konsequent bei.)

- Variablen, Funktionen, Methoden, Pakete, Module:  
`lower_case_with_underscores`
- Objektklassen und Ausnahmen:  
`CapWords`
- Geschützte Methoden und interne Funktionen:  
`_single_leading_underscore(self, ...)`
- Private Methoden:  
`__double_leading_underscore(self, ...)`
- Konstanten:  
`ALL_CAPS_WITH_UNDERSCORES`

Tipp: Vermeiden Sie Variablen, die aus nur einem Buchstaben bestehen (insbesondere L, O, I), ausgenommen in sehr kurzen Blöcken, wenn die Bedeutung unmissverständlich aus dem unmittelbaren Kontext hervorgeht.

#### Verwendung leerer Zeilen:

- Fügen Sie vor und nach Definitionen von Funktionen auf oberster Ebene und von Klassen je zwei leere Zeilen ein.
- Schließen Sie Methodendefinitionen innerhalb einer Klasse in einfache leere Zeilen ein.
- Zusätzliche leere Zeilen können (in Maßen) dazu verwendet werden, Gruppen zusammengehöriger Funktionen voneinander zu trennen.

#### Vermeiden Sie überflüssigen Leerraum an den folgenden Stellen:

- direkt in runden, geschweiften oder eckigen Klammern:

```
### BAD  
function( apples[ 1 ], { oranges: 2 } )
```

```
### GOOD:  
function(apples[1], {oranges: 2})
```

- unmittelbar vor einem Komma, Semikolon oder Doppelpunkt:

```
### BAD
```

```
if x == 2 : print x , y ; x , y = y , x  
### GOOD  
if x == 2: print x, y; x, y = y, x
```

- unmittelbar vor der öffnenden Klammer am Anfang der Liste der Argumente für einen Funktionsaufruf:

```
### BAD  
function (1)  
  
### GOOD  
function(1)
```

- unmittelbar vor der öffnenden Klammer am Anfang von Indizierungen und Teilbereichen:

```
### BAD  
dict ['key'] = list [index]  
  
### GOOD  
dict['key'] = list[index]
```

- Fügen Sie vor und nach diesen Binäroperatoren immer jeweils ein Leerzeichen ein:

```
assignment ( = )  
augmented assignment ( += , -= etc.)  
comparisons ( == , < , > , != , <> , <= , >= , in , not in , is , is not )  
Booleans ( and , or , not )
```

#### Beachten Sie die Länge:

- Sie sollte ca. 79 Zeichen möglichst nicht überschreiten.
- Indem Sie die Breite der benötigten Editor-Fenster beschränken, können Sie mehrere Dateien nebeneinander anzeigen. Dies ist besonders bei der Verwendung von Codeprüfungs-Tools hilfreich, die die beiden Versionen in benachbarten Spalten zeigen.
- Lange Zeilen können umbrochen und auf mehrere Zeilen verteilt werden, indem Sie Ausdrücke in Klammern setzen:

#### Vermeiden Sie allzu offensichtliche und überflüssige Kommentare:

- Durch weniger Kommentare erhalten Sie zuweilen leichter lesbaren Code. Dies gilt insbesondere, wenn Sie infolgedessen auf aussagekräftige Symbolnamen achten müssen.
- Durch sinnvolle Arbeitsgewohnheiten beim Schreiben von Code benötigen Sie weniger Kommentare:

```
### BAD  
# get the country code  
country_code = get_country_code(address)  
  
# if country code is US  
if (country_code == 'US'):  
# display the form input for state  
print form_input_state()  
  
### GOOD  
# display state selection for US users  
country_code = get_country_code(address)  
if (country_code == 'US'):  
print form_input_state()
```

Tipp: Kommentare beantworten die Frage nach dem Warum, Code nach dem Wie.

#### Checken Sie Open Source-Code aus:

- Open Source-Projekte werden durch die Zusammenarbeit vieler Entwickler vorangetrieben. In diesen Projekten ist die leichte Verständlichkeit des Codes unverzichtbar, damit das Team so effizient wie möglich zusammenarbeiten kann. Aus diesem Grund empfiehlt es sich, den Quellcode dieser Projekte durchzusehen und aus der Arbeitsweise dieser Entwickler Anregungen zu schöpfen.

- Verbessern Sie Ihre Konventionen:
- Überprüfen Sie für jede einzelne Konvention, ob sie für den aktuellen Verwendungszweck geeignet ist.
- Kommt es zu Beeinträchtigungen der Funktionsfähigkeit/Effizienz?

## Standards für C# (Zerotouch)

Auf den folgenden Wiki-Seiten finden Sie Anweisungen zum Schreiben von C# für Zerotouch und wie Sie zu Dynamo beitragen können:

- Im folgenden Wiki werden allgemeine Coding-Standards zum Dokumentieren und Testen des Codes beschrieben: <https://github.com/DynamoDS/Dynamo/wiki/Coding-Standards>
- Im folgenden Wiki wird speziell auf Namenskonventionen für Bibliotheken, Kategorien, Blocknamen, Anschlussnamen und Abkürzungen eingegangen: <https://github.com/DynamoDS/Dynamo/wiki/Naming-Standards>

### Nicht verwaltete Objekte:

Wenn Sie die Geometriebibliothek von Dynamo (*ProtoGeometry*) in Python oder C# verwenden, werden von Ihnen erstellte Geometrieobjekte nicht durch die virtuelle Maschine verwaltet, und der Speicher für viele dieser Objekte muss manuell bereinigt werden. Zum Bereinigen nativer oder nicht verwalteter Objekte können Sie die **Dispose**-Methode oder das **using**-Schlüsselwort verwenden. Einen Überblick hierzu finden Sie in diesem Wiki-Eintrag: <https://github.com/DynamoDS/Dynamo/wiki/Zero-Touch-Plugin-Development#dispose--using-statement>.

Sie müssen nur diejenigen nicht verwalteten Ressourcen beseitigen, die Sie nicht in das Diagramm ausgeben und auf die keine Verweise gespeichert werden. Für den Rest dieses Abschnitts werden diese Objekte als *temporäre Geometrie* bezeichnet. Das Codebeispiel unten zeigt ein Beispiel für diese Klasse von Objekten. Diese Zerotouch C#-Funktion namens **singleCube** gibt einen einzelnen Würfel zurück, erstellt jedoch während ihrer Ausführung 10.000 zusätzliche Würfel. Nehmen Sie an, dass diese zusätzliche Geometrie als temporäre Geometrie zur Konstruktion verwendet wurde.

**Diese Zerotouch-Funktion bringt Dynamo mit großer Wahrscheinlichkeit zum Absturz.** Es wurden 10.000 Volumenkörper erstellt, jedoch nur einer davon gespeichert, und nur dieser wurde zurückgegeben. Stattdessen sollten alle temporären Würfel ausgenommen derjenige, der zurückgegeben werden soll, beseitigt werden. Der zurückzugebende Würfel darf nicht beseitigt werden, da er an das Diagramm weitergeleitet und von anderen Blöcken verwendet werden soll.

```
public Cuboid singleCube(){

var output = Cuboid.ByLengths(1,1,1);

for(int i = 0; i<10000;i++){
output = Cuboid.ByLengths(1,1,1);
}
return output;
}
```

Der feste Code sieht ungefähr so aus:

```
public Cuboid singleCube(){

var output = Cuboid.ByLengths(1,1,1);
var toDispose = new List<Geometry>();

for(int i = 0; i<10000;i++){
toDispose.Add(Cuboid.ByLengths(1,1,1));
}

foreach(IDisposable item in toDispose ){
item.Dispose();
}

return output;
}
```

Im Allgemeinen muss nur Geometrie der Typen **Surface**, **Curve**, **Solid** usw. beseitigt werden. Sie können jedoch zur Sicherheit alle Geometrietypen (**Vector**, **Point**, **CoordinateSystem**) beseitigen.

# **Anhang**

## **Anhang**

In diesem Abschnitt finden Sie Informationen zu zusätzlichen Ressourcen, die Sie dabei unterstützen, Ihre Arbeit mit Dynamo zu vertiefen. Sie finden hier auch einen Index wichtiger Blöcke, eine Sammlung nützlicher Pakete und ein Repository der in diesem Primer enthaltenen Beispieldateien. Sie können diesen Abschnitt gerne um weitere Informationen ergänzen, schließlich ist dieser [Dynamo Primer](#) eine Open-Source-Dokumentation.

**Resources**

**Dynomo Language Guide**  
Programming languages are created to express ideas, usually involving logic and calculation. In addition to these objectives, the Dynomo visual language—formerly Dynoblock—has been created to express design intent. If it generally requires that computational designing is explanatory, and Dynomo does this intent well, we hope that the language looks like and feels enough to take a design from concept through design realization, to your final build. This manual is structured to give a user who is new to either programming or architecture geometry full exposure to a variety of topics in these two intersecting disciplines.

[http://www.dynamoblocks.com/guide/dynomo\\_guide.html](http://www.dynamoblocks.com/guide/dynomo_guide.html)

**Zero Touch Development for Dynamo**  
This page outlines the process of developing a `C#` interface, in most cases, to static methods and classes, which only needs to run function, and not construct an object. Methods (Dynamo objects) need `DL`, if they are static, methods. When objects need `DL`, it's fairly easy to do.

[http://www.dynamoblocks.com/guide/zero\\_touch.html](http://www.dynamoblocks.com/guide/zero_touch.html)

**Python for Beginners**

Python is an interpreted, interactive, object-oriented, dynamic-hybrid, very high-level dynamic language with remarkable power with every static type. It has many applications in machine learning, and is available in most environments, and is extensible in `C/C++`. It's very easy to learn, and is available on Mac, Linux, Windows, and on Windows 2008 and later. It's introductory tutorials and resources for learning Python.

<http://www.python.org/doc/essays/intro/index.html>

**AForge**

AForge.NET is an open-source C# framework designed to support Computer Vision and Artificial Intelligence – image processing, logic, machine learning, robotics, etc.

<http://www.aforgenet.com/>

**MathNet**

MathNet is an online mathematics resource, spanning thousands of subjects. Since its contents test spans a need of mathematical information in both the mathematics extensively referenced in journals and books spanning all fields of study.

<http://mathnet.sourceforge.net/>

**Dynamo Packages**

Here are a list of some of the more popular packages in the Dynamo community. Developers, please add to the list! Contributors, the `Dynamo` Primer is open-sourced!

**BUBBLEZES FOR DYNAMO**

Bubblez is an Excel and Dynamo interoperability project that easily improves Dynamo's ability to read and write Excel files.

<http://www.bubblez.com/>

**CLOCKWORK FOR DYNAMO**

Clockwork is a collection of custom nodes for the Dynamo programming environment. If Dynamo can't read certain nodes, but also lots of nodes for various other purposes (join, intersection, mathematical operations, using ranges and conversions, geometric operations, creating bounding boxes, planes, points, surfaces, UVs and vectors) are provided.

**DYNAMIC-SAP**

DynamicSAP is a parametric interface for SAP2000, but of Dynamo. The project aims to develop engineers and designers to generatively design and analyse structural systems. It uses Dynamo to drive the SAP model. The project provides new creation workflows which are described in the new example workflow, and provides a wide range of opportunities for automation of typical tasks in SAP.

**DYNAFORM**

This library extends Dynomodel's functionality by enabling users to edit surface and poly-topology geometry. This allows users to first define surfaces into plane facets, then edit them using Polytopology tools. This package also includes some experimental nodes as well as a few basic samples files.

**Resources**

**Dynomo Language Guide**  
Programming languages are created to express ideas, usually involving logic and calculation. In addition to these objectives, the Dynomo visual language—formerly Dynoblock—has been created to express design intent. If it generally requires that computational designing is explanatory, and Dynomo does this intent well, we hope that the language looks like and feels enough to take a design from concept through design realization, to your final build. This manual is structured to give a user who is new to either programming or architecture geometry full exposure to a variety of topics in these two intersecting disciplines.

[http://www.dynamoblocks.com/guide/dynomo\\_guide.html](http://www.dynamoblocks.com/guide/dynomo_guide.html)

**Zero Touch Development for Dynamo**  
This page outlines the process of developing a `C#` interface, in most cases, to static methods and classes, which only needs to run function, and not construct an object. Methods (Dynamo objects) need `DL`, if they are static, methods. When objects need `DL`, it's fairly easy to do.

[http://www.dynamoblocks.com/guide/zero\\_touch.html](http://www.dynamoblocks.com/guide/zero_touch.html)

**Python for Beginners**

Python is an interpreted, interactive, object-oriented, dynamic-hybrid, very high-level dynamic language with remarkable power with every static type. It has many applications in machine learning, and is available in most environments, and is extensible in `C/C++`. It's very easy to learn, and is available on Mac, Linux, Windows, and on Windows 2008 and later. It's introductory tutorials and resources for learning Python.

<http://www.python.org/doc/essays/intro/index.html>

**AForge**

AForge.NET is an open-source C# framework designed to support Computer Vision and Artificial Intelligence – image processing, logic, machine learning, robotics, etc.

<http://www.aforgenet.com/>

**MathNet**

MathNet is an online mathematics resource, spanning thousands of subjects. Since its contents test spans a need of mathematical information in both the mathematics extensively referenced in journals and books spanning all fields of study.

<http://mathnet.sourceforge.net/>

**INDEX OF NODES**

These index pages provide additional information on all the nodes used in this primer, as well as other might find useful. This is just an introduction to some of the 600 nodes available.

**functions**

Return the 10 list of the `multinomial` function.

**INDEX OF NODES**

These index pages provide additional information on all the nodes used in this primer, as well as other components you might find useful. This is just an introduction to some of the 600 nodes available.

**Builtin Functions**

<b>Count</b>	Returns the number of items in the specified list.
<b>ForEach</b>	Iterates over the input <code>list</code> .
<b>Map</b>	Maps a value into an input range.

**Core**

**Core Color**

<b>Color</b>	Creates a color gradient between a start and end color.
<b>Color By Angle</b>	Creates a color at a given angle and the components.
<b>Color Range</b>	Get a color from a color gradient between start and end colors.
<b>Color Brightness</b>	Gets the brightness value for this color.

**Dynamo Example Files**

These example files accompany the `Dynamo` Primer, and are organized according to Chapter and Section.

**Blocks of Programs**

Right click here and use "Save Link As..."

**Introduction**

What is Visual Programming    What Programming    Click Through First App

**The Building Blocks of Programs**

<b>Section</b>	<b>Download File</b>
Date	Building Blocks of Programs - Date.dyn
Math	Building Blocks of Programs - Math.dyn
Logic	Building Blocks of Programs - Logic.dyn
Ranges	Building Blocks of Programs - Range.dyn
Color	Building Blocks of Programs - Color.dyn

**Computational Design**

<b>Section</b>	<b>Download File</b>
Geometry	Building Blocks of Programs - Geometry.dyn
Geometry By Context	Building Blocks of Programs - Geometry By Context.dyn
Geometry As Context	Building Blocks of Programs - Geometry As Context.dyn
Surface	Building Blocks of Programs - Surface.dyn

**Lists**

<b>Section</b>	<b>Download File</b>
Lists	Building Blocks of Programs - Lists.dyn
List Append	Building Blocks of Programs - List Append.dyn
List RemoveItem	Building Blocks of Programs - List RemoveItem.dyn
List ContainsItem	Building Blocks of Programs - List ContainsItem.dyn
Surface	Building Blocks of Programs - Surface.dyn

**Designing with Lists**

<b>Section</b>	<b>Download File</b>
What's a List	Designing with Lists - What's a List.dyn
Working with Lists	Designing with Lists - Working with Lists.dyn
List Append	Designing with Lists - List Append.dyn
List Count	Designing with Lists - List Count.dyn
List RemoveItem	Designing with Lists - List RemoveItem.dyn
List ContainsItem	Designing with Lists - List ContainsItem.dyn
Surface	Designing with Lists - Surface.dyn

# Ressourcen

## Ressourcen

### Dynamo Wiki

"Dieses Wiki vermittelt Informationen zur Entwicklung mit der Dynamo-API sowie zu unterstützenden Bibliotheken und Werkzeugen."

<https://github.com/DynamoDS/Dynamo/wiki>

### Dynamo Blog

Dieser Blog stellt die aktuellste Sammlung an Artikeln des Dynamo-Teams dar, das neue Funktionen, Arbeitsabläufe und andere Dinge in Bezug auf Dynamo diskutiert.

<http://dynamobim.com/blog/>

### DesignScript Guide

Programmiersprachen werden entwickelt, um Ideen auszudrücken, die normalerweise Logik und Berechnungen einschließen. In Ergänzung dazu wurde die textuelle Sprache von Dynamo (früher DesignScript) entwickelt, um Konstruktionsabsichten auszudrücken. Es ist allgemein anerkannt, dass die computergestützte Konstruktion eine Herausforderung darstellt, bei der Dynamo Unterstützung bietet: Wir hoffen, dass die Sprache für Sie flexibel und schnell genug ist, um eine Konstruktion von der Konzeption über Designiterationen bis hin zur endgültigen Form zu entwickeln. Dieses Handbuch ist so strukturiert, dass es Benutzern ohne Kenntnisse in der Programmierung oder architektonischen Geometrie eine breite Palette an Themen in diesen beiden sich überschneidenden Bereichen bietet.

[http://dynamobim.org/wp-content/uploads/forum-assets/colin-mccroneautodesk-com/07/10/Dynamo\\_language\\_guide\\_version\\_1.pdf](http://dynamobim.org/wp-content/uploads/forum-assets/colin-mccroneautodesk-com/07/10/Dynamo_language_guide_version_1.pdf)

### Das Dynamo Primer-Projekt

Dynamo Primer ist ein Open-Source-Projekt, das von Matt Jezyk und dem Dynamo-Entwicklungsteam bei Autodesk initiiert wurde. Die erste Version des Primers wurde von Mode Lab entwickelt. Um zu diesem Projekt beizutragen, spalten Sie das Repository ab, fügen eigene Inhalte hinzu und reichen eine Pull-Anforderung ein.

<https://github.com/DynamoDS/DynamoPrimer>

### Entwicklung von Zero-Touch-Plugins für Dynamo

Auf dieser Seite wird die Entwicklung eines benutzerdefinierten Dynamo-Blocks in C# mithilfe der "Zero Touch"-Oberfläche erläutert. In den meisten Fällen können statische C#-Methoden und -Klassen ohne Änderung importiert werden. Wenn die Bibliothek nur Funktionen aufrufen und keine neuen Objekte konstruieren muss, kann dies sehr einfach mit statischen Methoden erreicht werden. Wenn Dynamo Ihre DLL lädt, wird der Namensraum Ihrer Klassen entfernt und werden alle statischen Methoden als Blöcke bereitgestellt.

<https://github.com/DynamoDS/Dynamo/wiki/Zero-Touch-Plugin-Development>

### Python für Einsteiger

Python ist eine interpretierte, interaktive, objektorientierte Programmiersprache. Sie enthält Module, Exceptions, dynamische Typisierung, sehr hohe dynamische Datentypen und Klassen. Python kombiniert bemerkenswerte Stärke mit einer sehr klaren Syntax. Sie bietet Schnittstellen zu vielen Systemen und Bibliotheken sowie zu zahlreichen Windows-Systemen und ist auf C und C++ erweiterbar. Sie kann auch als Erweiterungssprache für Anwendungen verwendet werden, die eine programmierbare Benutzeroberfläche erfordern. Schließlich ist Python portierbar: Sie kann auf vielen UNIX-Varianten sowie auf Mac-Systemen und unter Windows 2000 und späteren Versionen ausgeführt werden. Das Einsteigerhandbuch zu Python bietet Verknüpfungen zu anderen einführenden Übungslektionen und Ressourcen zum Erlernen von Python.

<https://www.python.org/about/gettingstarted>

### AForge

AForge.NET ist ein C#-Open-Source-Framework, das für Entwickler und Forscher in den Bereichen Computer Vision und Künstliche Intelligenz – Bildverarbeitung, neuronale Netze, genetische Algorithmen, Fuzzy-Logik, maschinelles Lernen,

Robotik usw. – konzipiert ist.

<http://www.aforgenet.com/framework/>

### **Wolfram MathWorld**

MathWorld ist eine Online-Ressource zur Mathematik, die von Eric W. Weisstein mit Unterstützung von tausenden Beitragenden zusammengetragen wurde. MathWorld, dessen Inhalte zum ersten Mal 1995 online veröffentlicht wurden, hat sich zum Nexus für mathematische Informationen sowohl in der Mathematik- als auch in der Bildungs-Community entwickelt. Die Einträge in dieser Online-Ressource werden in hohem Maße in Fachzeitschriften und Büchern über alle Bildungsebenen zitiert.

<http://mathworld.wolfram.com/>

### **Revit-Ressourcen**

#### **buildz**

"Diese Beiträge, in denen es in erster Linie um die Revit-Plattform geht, enthalten Empfehlungen zur optimalen Nutzung."

<http://buildz.blogspot.com/>

#### **Nathan's Revit API Notebook**

"Mit diesem Notebook wird versucht, ein paar 'Ressourcenmängel' beim Erlernen und Anwenden der Revit-API im Kontext eines Konstruktionsablaufs zu beheben."

<http://wiki.theprovingground.org/revit-api>

#### **Revit Python Shell**

"Die RevitPythonShell fügt einen IronPython-Interpreter zu Autodesk Revit und Vasari hinzu." Dieses Projekt wurde schon vor Dynamo gestartet und ist eine hervorragende Referenz für die Python-Entwicklung. RPS-Projekt:

<https://github.com/architecture-building-systems/revitpythonshell> Developer's Blog: <http://darenatwork.blogspot.com/>

#### **The Building Coder**

Ein solider Katalog mit Revit API-Arbeitsabläufen von einem der führenden BIM-Experten.

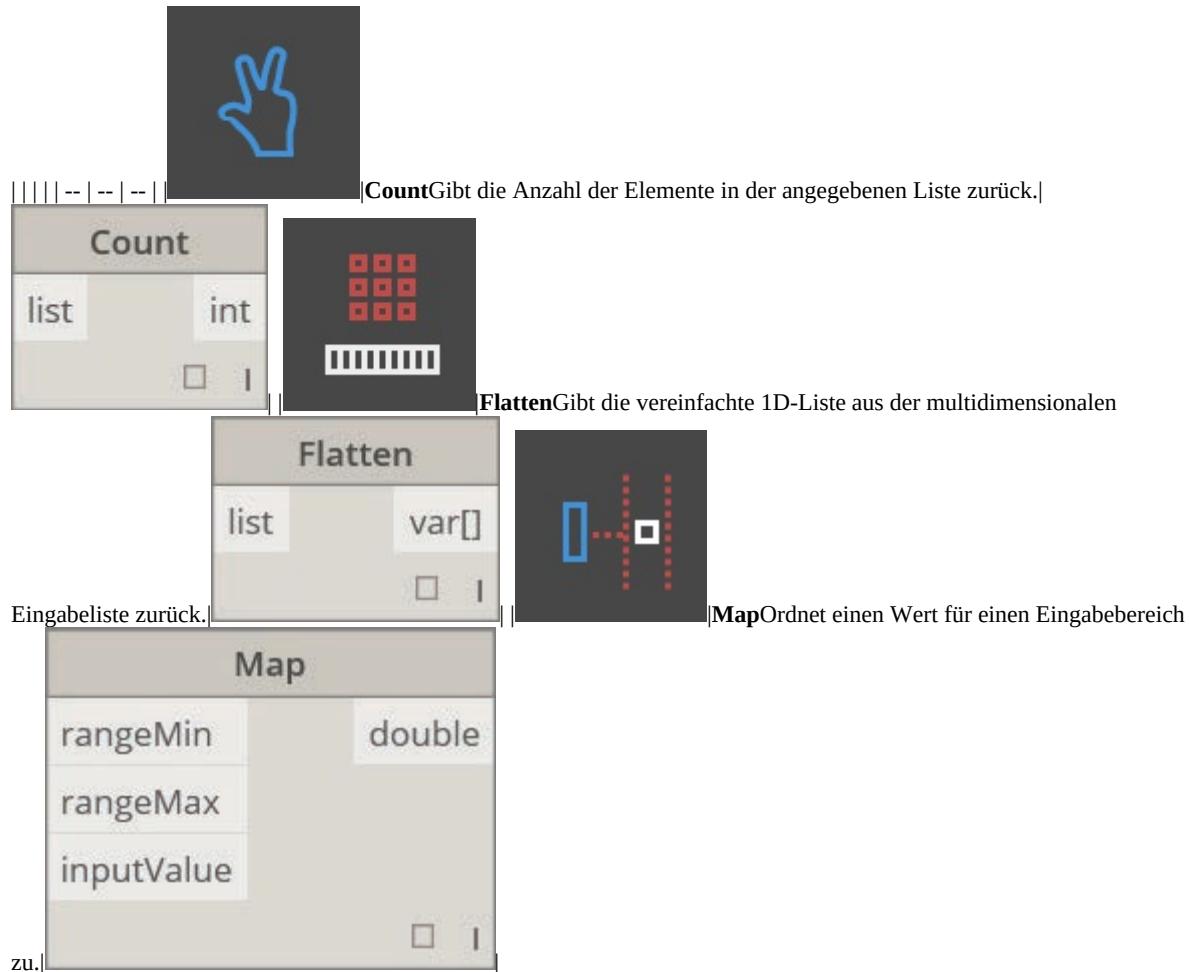
<http://thebuildingcoder.typepad.com/>

# Index: Blöcke

## Index: Blöcke

Dieser Index enthält weitere Informationen zu allen in dieser Einführung verwendeten Blöcken sowie zu anderen Komponenten, die für Sie eventuell nützlich sind. Dabei ist dies nur eine Einführung in die über 500 Blöcke, die in Dynamo zur Verfügung stehen.

### Integrierte Funktionen



### Core

#### Core.Color



A screenshot of a software interface showing the 'Color.ByARGB' node. The interface has a light gray background. On the left, there is a vertical stack of four input fields labeled 'a', 'r', 'g', and 'b', each containing a single character. To the right of these fields is a large, semi-transparent gray rectangle representing the color preview. In the bottom right corner of the preview area, there is a small color swatch showing a gradient from blue at the top to red at the bottom. At the very bottom of the interface, there are two small, semi-transparent square icons.

Komponenten.|||Color RangeRuft eine Farbe aus einem Farbverlauf zwischen einer Anfangs- und einer Endfarbe ab.|||

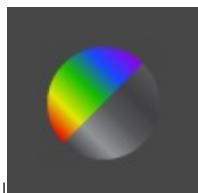
Color Range	
colors	color
indices	
value	

**Color Range** Ruft eine Farbe aus einem Farbverlauf

**|Color.Brightness**Ruft den Helligkeitswert für die Farbe ab.

**Color.Components** Listet die Komponenten für die Farbe in der Reihenfolge Alpha, Rot, Grün, Blau auf.

Color.Components	
c	a
	r
	g
	b



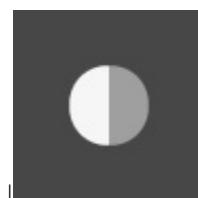
**Color.Saturation** Ruft den Sättigungswert für die Farbe ab.

**Color.Saturation**



**Color.Hue** Ruft den Farbtonwert für die Farbe ab.

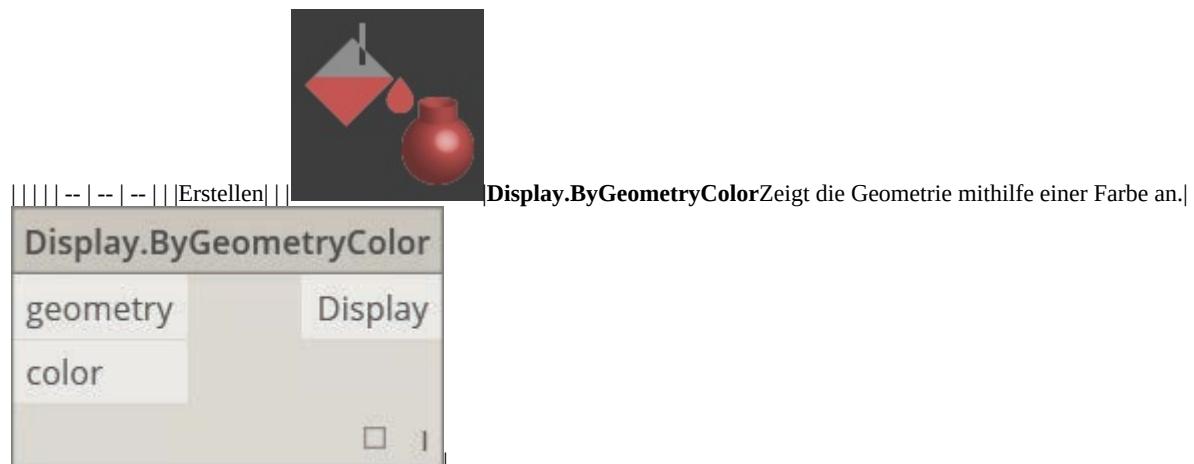
Color.Hue



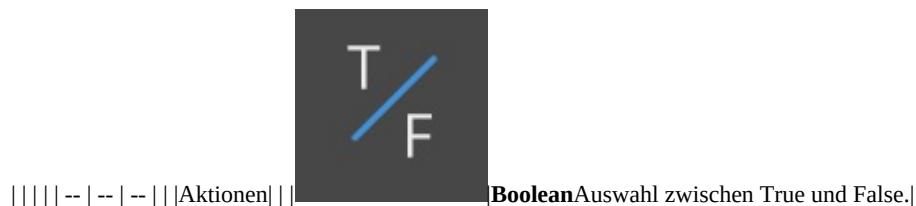
|Color.AlphaAlpha-Komponente der Farbe ermitteln: 0



### Core.Display



### Core.Input



**Boolean**

True   False >

**Code Block**

Your code goes here

Code.

**Code Block** Ermöglicht die direkte Erstellung von DesignScript-

**Directory Path**

Browse... >

No file selected.

Verzeichnisses auf dem System, um seinen Pfad abzurufen.

**Directory Path** Ermöglicht die Auswahl einen

**File**

Browse... >

No file selected.

**File Path**

Ermöglicht die Auswahl einer Datei auf dem System, um ihren Dateinamen abzurufen.

**Path**

**Integer Slider**

Ein Schieberegler zur Erzeugung ganzzahliger Werte.

**Number**

123

**Number** Erstellt eine

**Number**

0.000 >

Zahl.

**Number Slider**

Ein Schieberegler zur Erzeugung numerischer Werte.

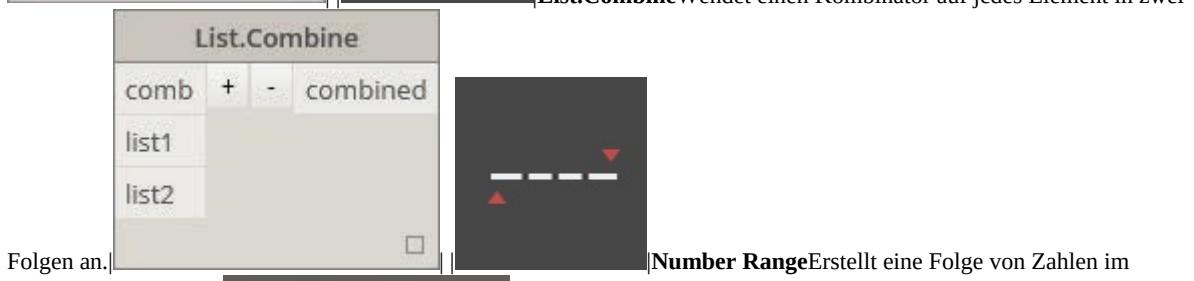
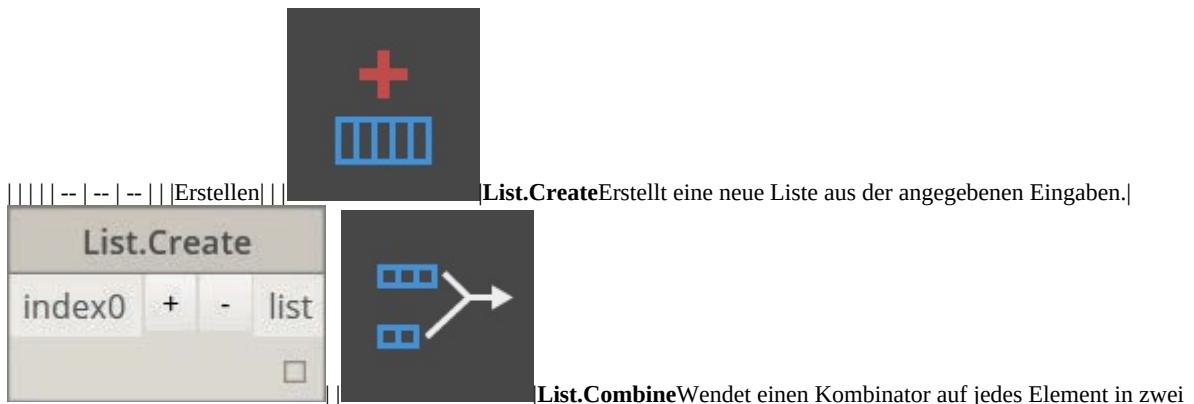
**String**

ABC

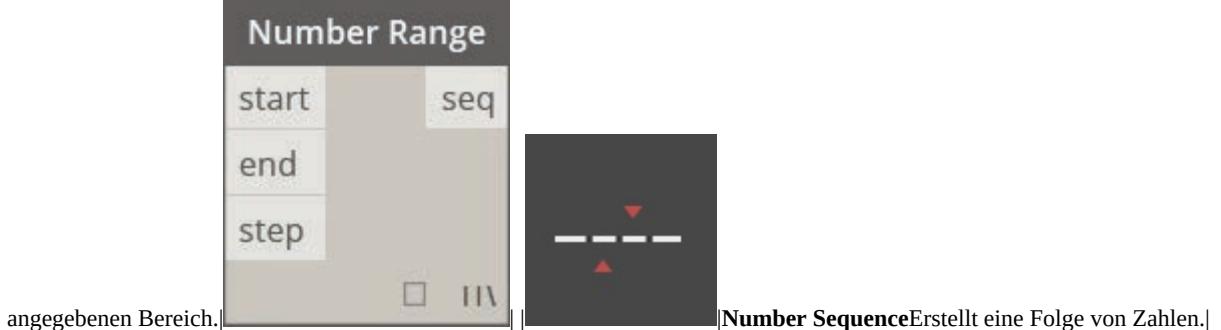
**String** Erstellt eine

Zeichenfolge.

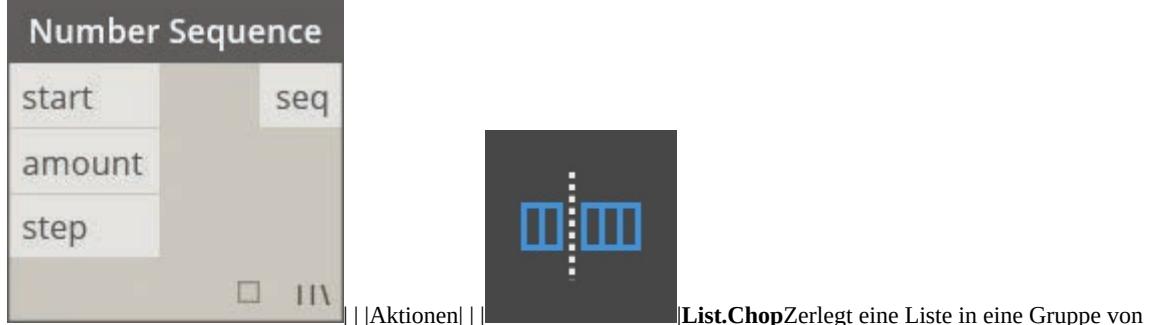
**Core.List**



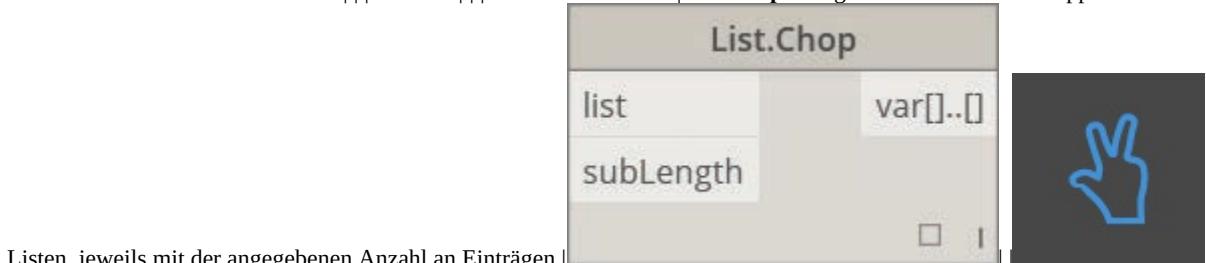
Folgen an.



angegebenen Bereich.



Aktionen |



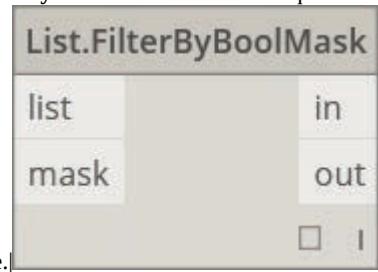
Listen, jeweils mit der angegebenen Anzahl an Einträgen.



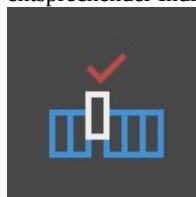
List.Flatten Vereinfacht eine verschachtelte Liste von Listen um eine bestimmte Anzahl von Ebenen.



**List.FilterByBoolMask** Filtert eine Sequenz durch Abrufen



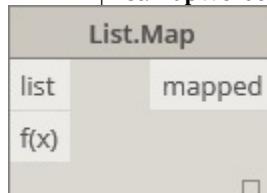
entsprechender Indizes in einer separaten Liste boolescher Werte.



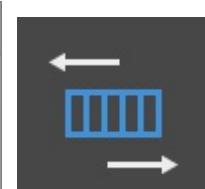
**List.GetItemAtIndex** Ruft ein Element aus der angegebenen Liste am angegebenen Index ab.



**List.Map** Wendet eine Funktion für alle Elemente einer Liste an



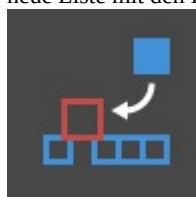
und erstellt aus den Ergebnissen eine neue Liste.



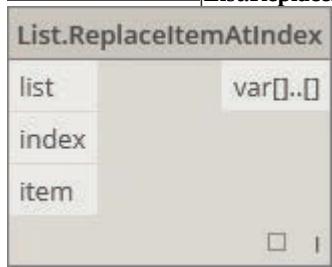
**List.Reverse** Erstellt eine



neue Liste mit den Elementen der angegebenen Liste, jedoch in umgekehrter Reihenfolge.



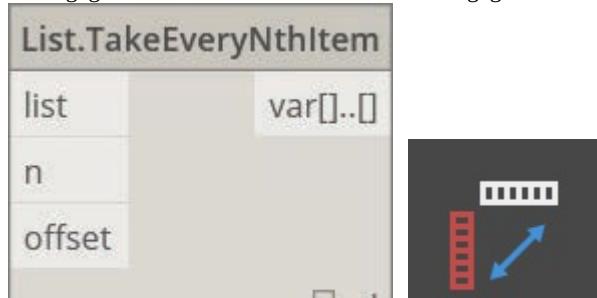
**List.ReplaceItemAtIndex** Ersetzt ein Element am angegebenen Index in der angegebenen Liste.



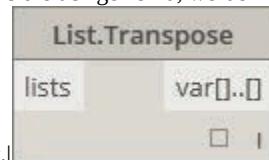
**List.ShiftIndices** Verschiebt die Indizes in der Liste um den



angegebenen Betrag nach rechts. | **List.TakeEveryNthItem** Ruft unter Einhaltung des angegebenen Versatzes Elemente aus der angegebenen Liste ab, deren Indizes Vielfache des angegebenen



Werts sind. | **List.Transpose** Vertauscht Zeilen und Spalten in einer Liste von Listen. Wenn einige Zeilen kürzer als die übrigen sind, werden Nullwerte als Platzhalter in das resultierende

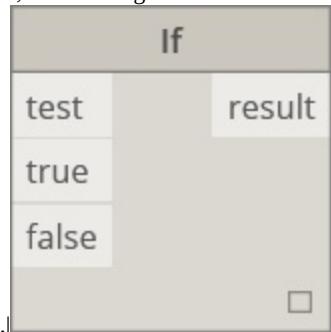


Array eingefügt, sodass dieses immer rechteckig ist. |

## Core.Logic



||||| -- | -- | -- || |Aktionen|| | **If** Bedingte Anweisung. Prüft den Booleschen Wert des eingegebenen Tests. Wenn der eingegebene Test den Wert True hat, wird als Ergebnis der zur Alternative True gehörige Wert ausgegeben,

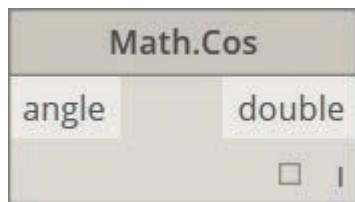


andernfalls der zur Alternative False gehörige Wert. |

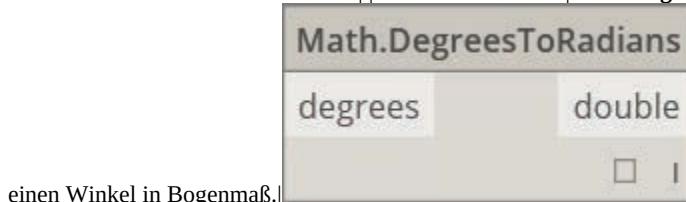
## Core.Math



||||| -- | -- | -- || |Aktionen|| | **Math.Cos** Ermittelt den Kosinus eines Winkels. |



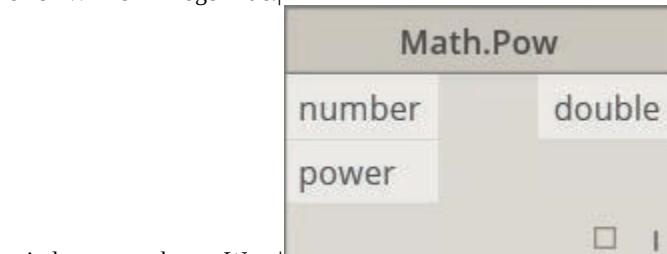
**Math.DegreesToRadians** Konvertiert einen Winkel in Grad in



einen Winkel in Bogenmaß.



**Math.Pow** Potenziert eine Zahl

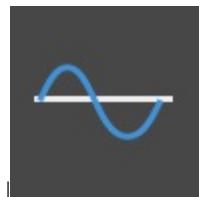
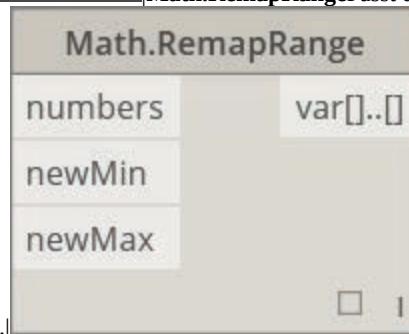


mit dem angegebenen Wert.

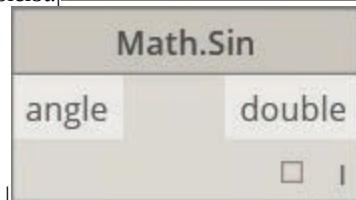
**Math.RadiansToDegrees** Konvertiert einen Winkel in Bogenmaß in einen Winkel in Grad.



**Math.RemapRange** Passt den Bereich einer Liste von

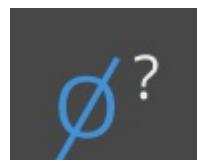


Zahlen an, wobei deren Verteilung erhalten bleibt.



**Math.Sin** Ermittelt den Sinus eines Winkels.

## Core.Object



**Object.IsNull** Bestimmt, ob das angegebene Objekt Null ist.

||||| -- | -- | -- || Aktionen ||



### Core.Scripting



||||| -- | -- | -- || Aktionen ||

|**Formula**Wertet mathematische Formeln aus. Dabei wird NCalc für die



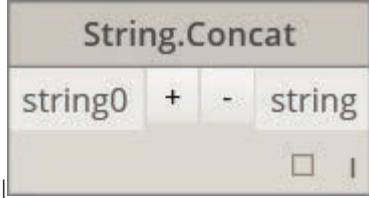
Auswertung verwendet. Weitere Informationen finden Sie unter <http://ncalc.codeplex.com>.

### Core.String



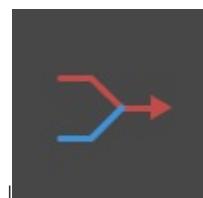
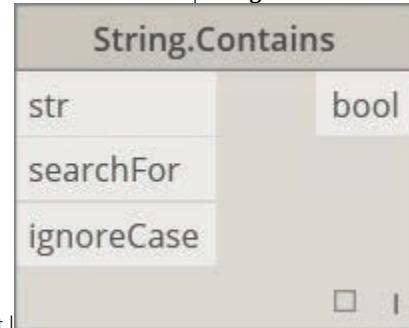
||||| -- | -- | -- || Aktionen ||

|**String.Concat**Verkettet mehrere Zeichenfolgen zu einer einzigen



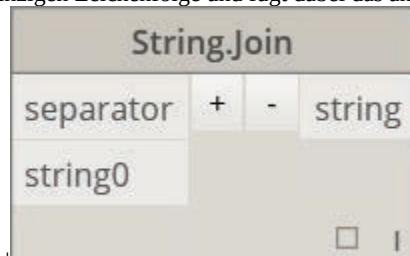
Zeichenfolge.|

|**String.Contains**Bestimmt, ob die angegebene



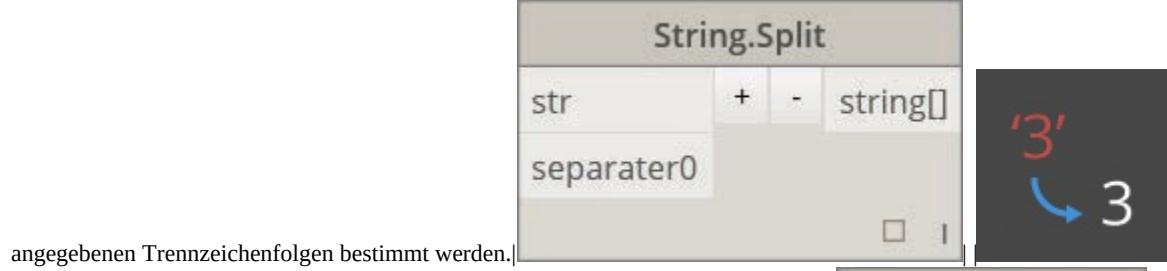
Zeichenfolge die angegebene Teilzeichenfolge enthält.|

|**String.Join**Verkettet mehrere Zeichenfolgen zu einer einzigen Zeichenfolge und fügt dabei das angegebene Trennzeichen



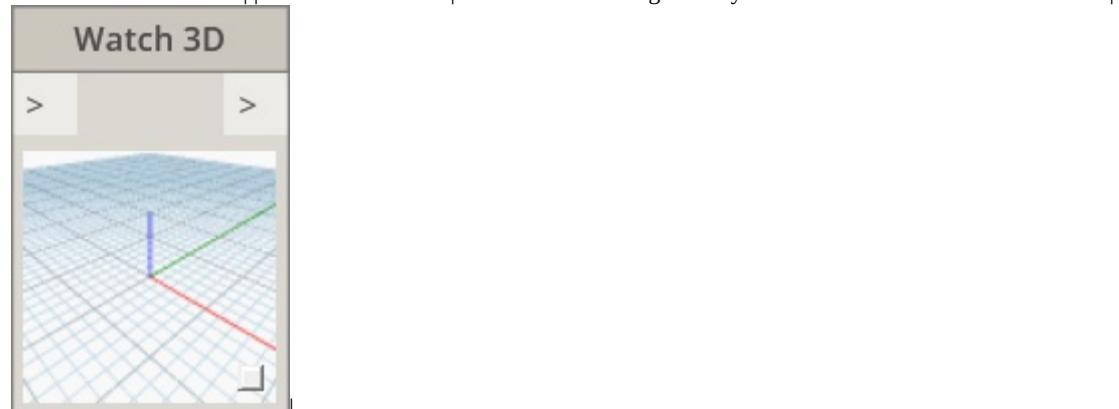
zwischen den einzelnen verbundenen Zeichenfolgen ein.|

|**String.Split**Teilt eine einzelne Zeichenfolge in eine Liste von Zeichenfolgen auf, wobei die Unterteilungen durch die



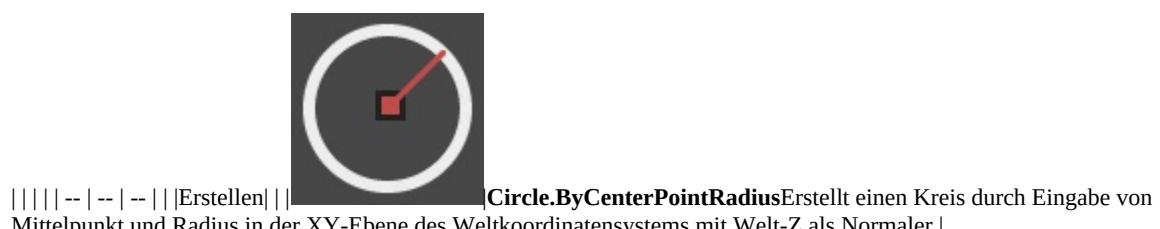
|**String.ToString**Konvertiert eine Zeichenfolge in einen integer- oder double-Wert.|

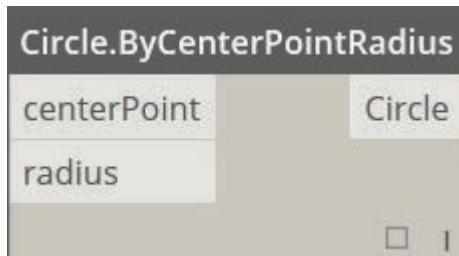
### Core.View



## Geometrie

### Geometry.Circle

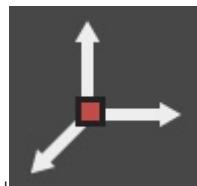




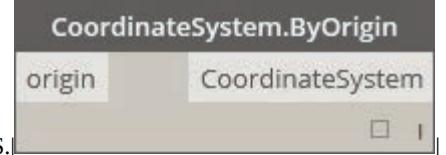
**Circle.ByPlaneRadius**Erstellt einen Kreis zentriert am Ursprung der Eingabeebene, innerhalb der Eingabeebene und mit dem angegebenen Radius.|



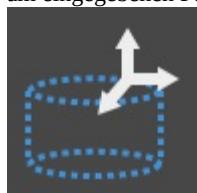
#### Geometry.CoordinateSystem



||||| -- | -- | -- | | Erstellen || | **CoordinateSystem.ByOrigin**Erstellt ein CoordinateSystem mit Ursprung



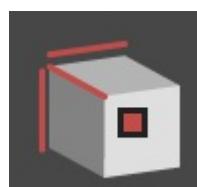
am eingegebenen Punkt mit X- und Y-Achse als X- und Y-Achse des WKS.|



**CoordinateSystem.ByCylindricalCoordinates**Erstellt ein CoordinateSystem mit den angegebenen zylindrischen Koordinatenparametern in Bezug auf das angegebene Koordinatensystem.|



#### Geometry.Cuboid



||||| -- | -- | -- | | Erstellen || | **Cuboid.ByLengths(Ursprung)**Erstellt einen Quader mit Mittelpunkt am

Cuboid.ByLengths	
origin	Cuboid
width	
length	
height	

Eingabepunkt und Angaben für Breite, Länge und Höhe.

#### Geometry.Curve



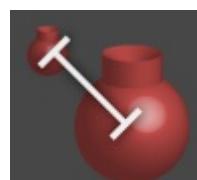
||||| -- | -- | -- || Aktionen || | Curve.Extrude (Abstand) Extrudiert eine Kurve in Richtung des

Curve.Extrude	
curve	Surface
distance	

Normalenvektors.|| | Curve.PointAtParameter Ruft einen Punkt auf der Kurve am angegebenen Parameter zwischen StartParameter() und EndParameter() ab.||

Curve.PointAtParameter	
curve	Point
param	

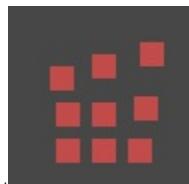
#### Geometry.Geometry



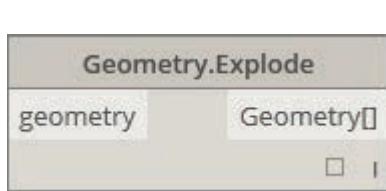
||||| -- | -- | -- || Aktionen || | Geometry.DistanceTo Ruft den Abstand zwischen dieser und anderer

Geometry.DistanceTo	
geometry	double
other	

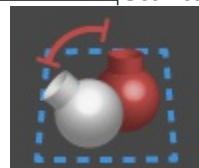
Geometrie ab.|| | Geometry.Explode Trennt zusammengesetzte oder nicht getrennte Elemente in die Teile, aus denen sie bestehen.||



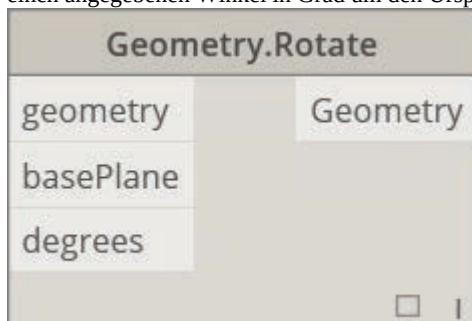
Geometry.Explode Trennt



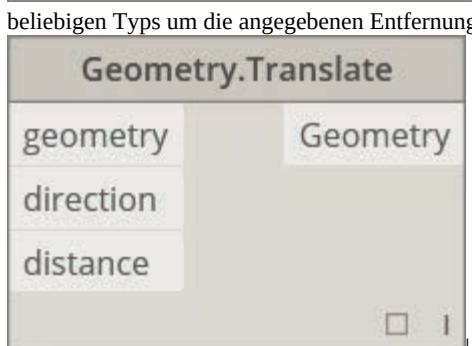
**Geometry.ImportFromSAT** Liste der importierten



**Geometry.Rotate** (basePlane) Dreht ein Objekt um einen angegebenen Winkel in Grad um den Ursprung und die Normale der Ebene.

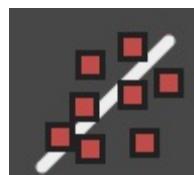


**Geometry.Translate** Verschiebt Geometrie



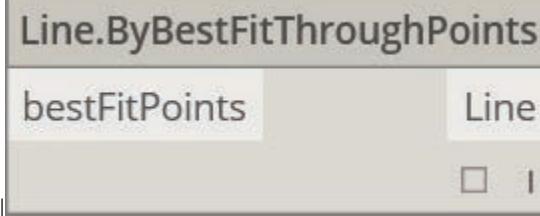
beliebigen Typs um die angegebenen Entfernung in die angegebene Richtung.

### Geometry.Line

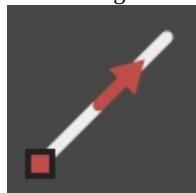


||||| -- | -- | -- | | Erstellen |||

**Line.ByBestFitThroughPoints** Erstellt eine Linie mit der bestmöglichen



Annäherung an ein Streudiagramm aus Punkten.



**Line.ByStartPointDirectionLength** Erstellt eine gerade Linie mit der angegebenen Länge vom

### Line.ByStartPointDirectionLength

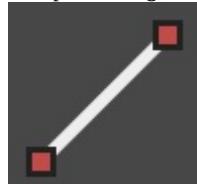
startPoint

Line

direction

length

|Startpunkt ausgehend in Vektorrichtung.|



### Line.ByStartPointEndPoint

startPoint

Line

endPoint



|Line.ByTangency|Erstellt eine zur eingegebenen

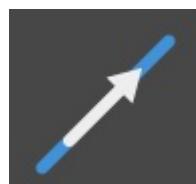
### Line.ByTangency

curve

Line

parameter

|Kurve tangentiale Linie am Parameterpunkt der eingegebenen Kurve.|



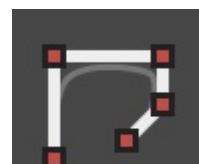
|Abfrage| | Line.Direction|Die Richtung der Kurve.

### Line.Direction

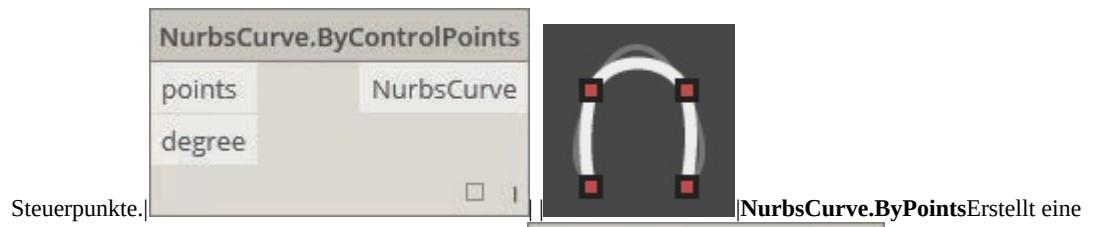
line

Vector

### Geometry.NurbsCurve



||||| -- | -- | -- | | Erstellen| | NurbsCurve.ByControlPoints|Erstellt ein BSplineCurve über explizite

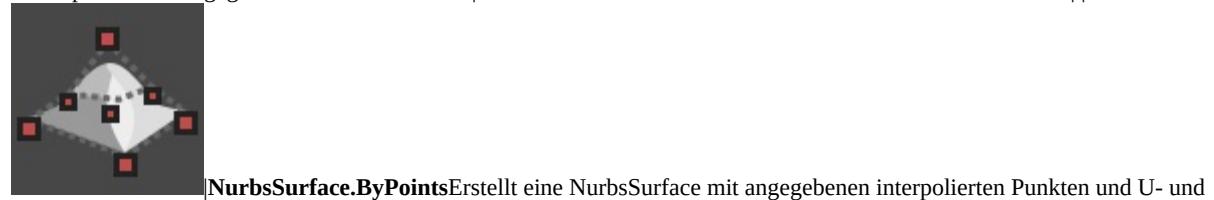


B-Spline-Curve durch Interpolation zwischen Punkten.

### Geometry.NurbsSurface



Steuerpunkte mit angegebenem U- und V-Grad.



V-Graden. Die resultierende Oberfläche verläuft durch alle Punkte.

### Geometry.Planes





mit eingegebenem Normalenvektor.

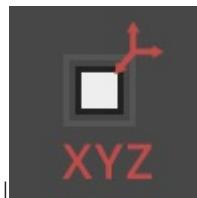


**Plane.XY** Erstellt eine



Ebene in der Welt-XY-Ebene.

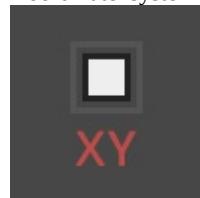
### Geometry.Point



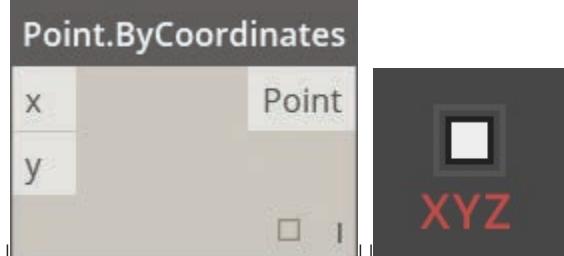
||||| -- | -- | -- || |Erstellen|| | **Point.ByCartesianCoordinates** Legt einen Punkt im angegebenen



Koordinatensystem mithilfe dreier kartesischer Koordinaten fest.



**Point.ByCoordinates (2D)** Legt einen Punkt in der XY-Ebene nach Angabe zweier kartesischer



Koordinaten fest. Die Z-Komponente hat den Wert 0.

**Point.ByCoordinates (3D)** Legt einen Punkt nach Angabe dreier kartesischer Koordinaten fest.

**Point.ByCoordinates**

x  
y  
z

Point

000

**Point.Origin**

Point

000

**Point.Add**

Aktionen

point  
vectorToAdd

Point.Add

X

**Point.X**

Abfrage

point  
double

Y

**Point.Y**

ab.

point  
double

Y

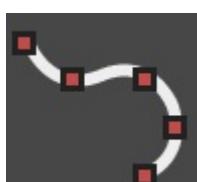
**Point.Z**

Point.ZRuft die Z-Komponente des Punkts ab.

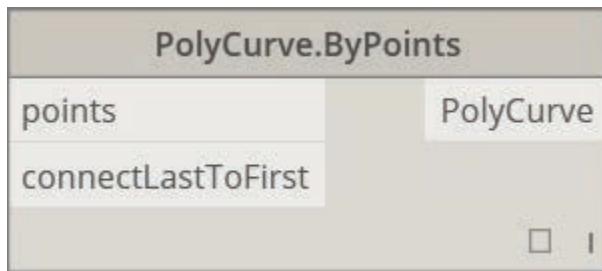
point  
double

X

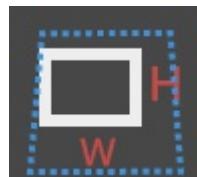
## Geometry.Polycurve



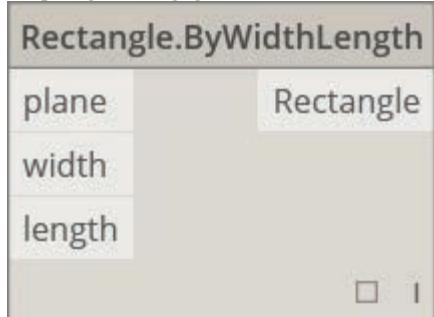
||||| -- | -- ||| Erstellen ||| Polycurve.ByPoints Erstellt PolyCurve aus einer Folge von Linien durch Verbinden von Punkten. Für geschlossene Kurven muss der letzte Punkt sich an derselben Stelle befinden wie der erste.|||



#### Geometry.Rectangle



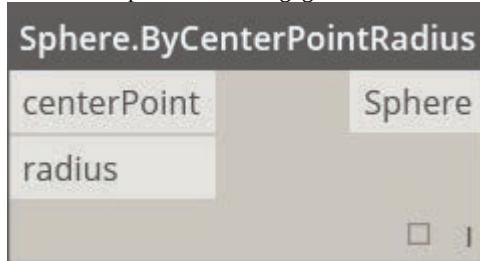
||||| -- | -- | -- | | Erstellen || | **Rectangle.ByWidthLength(Ebene)** Erstellt ein Rectangle zentriert am Ursprung der eingegebenen Plane mit der eingegebenen Breite (X-Achse) und Länge (Y-Achse).|



#### Geometry.Sphere



||||| -- | -- | -- | | Erstellen || | **Sphere.ByCenterPointRadius** Erstellt einen kugelförmigen Volumenkörper mit dem eingegebenen Punkt als Mittelpunkt und dem angegebenen Radius.|



#### Geometry.Surface



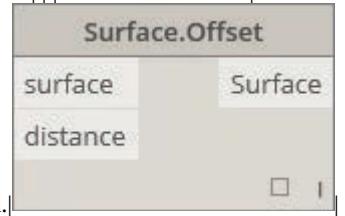
||||| -- | -- | -- | | Erstellen || | **Surface.ByLoft** Erstellt eine Oberfläche durch Erhebung zwischen den



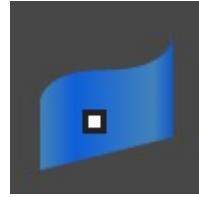
eingegebenen Querschnittskurven.] | Surface.ByPatchErstellt eine Oberfläche durch Ausfüllen des Bereichs innerhalb einer durch die eingegebenen Kurven definierten geschlossenen



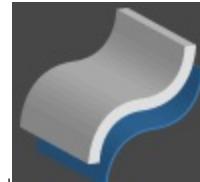
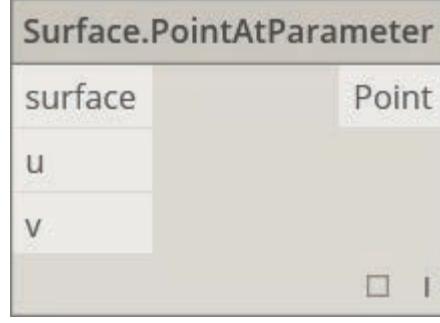
Begrenzung.] || Aktionen] | Surface.OffsetVersetzt die



Oberfläche in Richtung ihrer Normalen um den angegebenen Abstand.] ||



Surface.PointAtParameterGibt den Punkt für die angegebenen U- und V-Parameter zurück.]



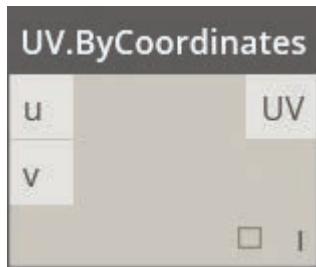
Extrusion in Richtung ihrer Normalen auf beiden Seiten in einen Volumenkörper um.] | Surface.ThickenWandelt eine Oberfläche durch



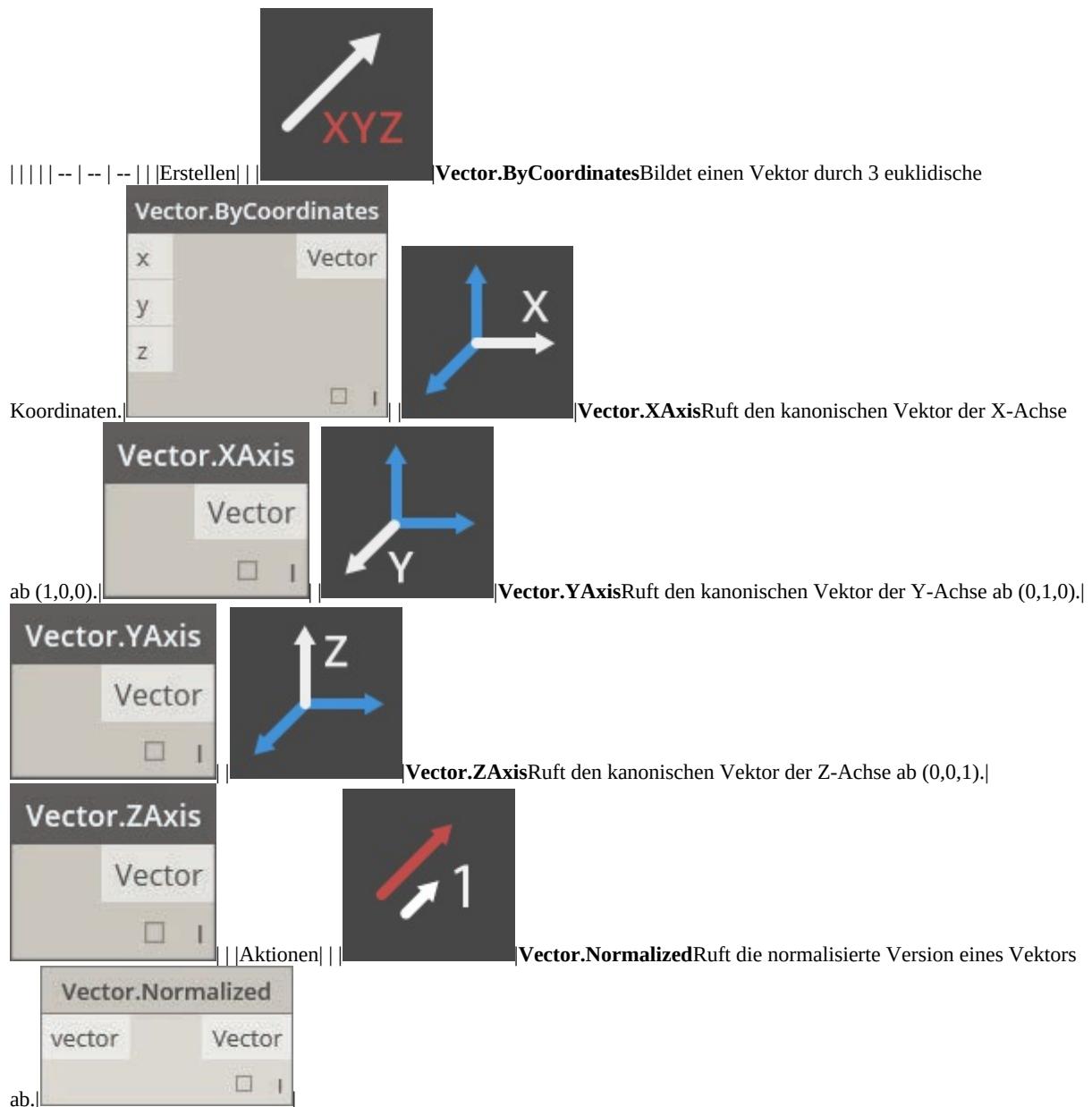
## Geometry.UV



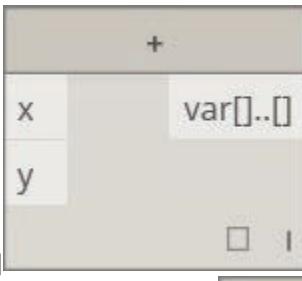
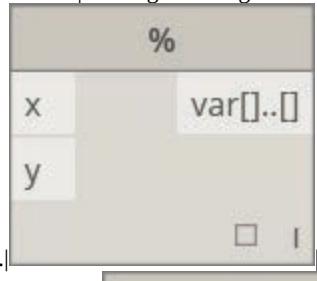
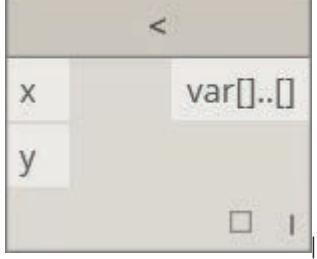
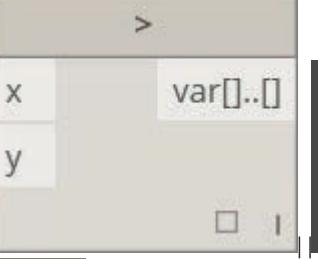
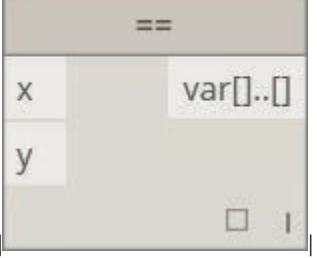
||||| -- | -- | -- | | Erstellen] | UV.ByCoordinatesErstellt UV aus zwei double-Werten.]



### Geometry.Vector



### Operatoren

	+ Addition 		- Subtraktion 
	* Multiplikation 		/ Division 
	% Die ganzzahlige Division ermittelt den bei der Division 		< der ersten Eingabe durch die zweite verbleibenden Rest. 
	> Größer als 		== Gleichheitstest: Vergleich zweier Werte. 

# Dynamo-Pakete

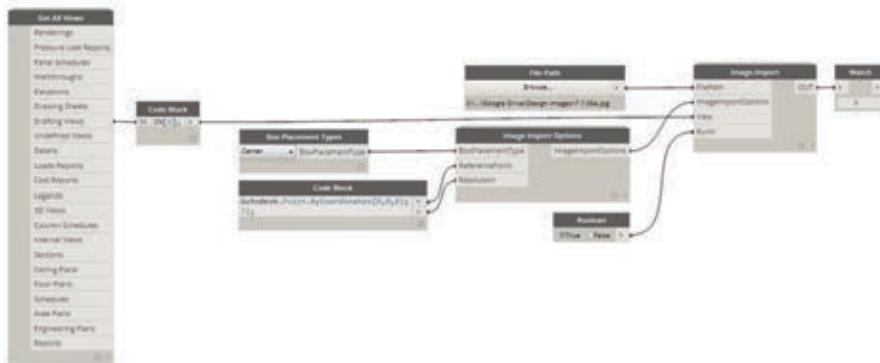
## Dynamo-Pakete

Im Folgenden sind einige der häufiger in der Dynamo-Community verwendeten Pakete aufgeführt. Entwickler sind aufgefordert, diese Liste zu ergänzen. Denken Sie daran, dass es sich bei [Dynamo Primer](#) um ein Open-Source-Projekt handelt.



ARCHI-LAB

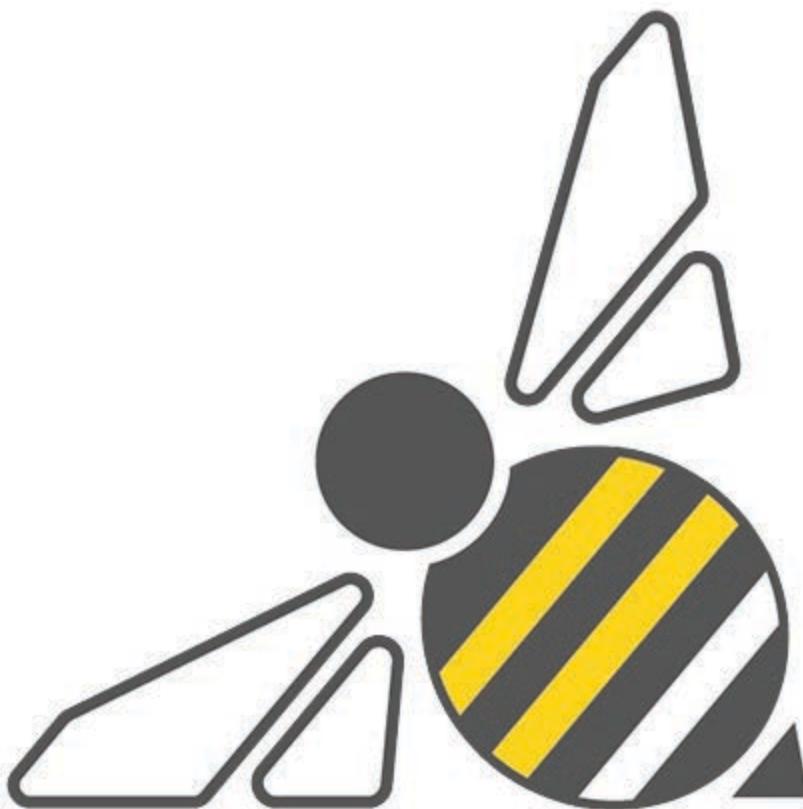
archi-lab ist eine Sammlung mit über 50 benutzerdefinierten Paketen, die die Möglichkeiten zur Interaktion zwischen Dynamo und Revit erheblich erweitern. Die archi-lab-Pakete enthalten Blöcke mit höchst unterschiedlichen Funktionen von einfachen Listenoperationen bis hin zu Analysis Visualization Framework-Blöcken für Revit.



## **BIMORPH-BLÖCKE**

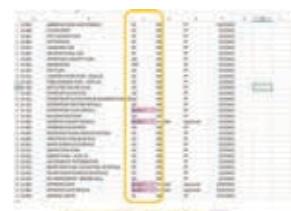
BimorphNodes ist eine vielseitige Sammlung mit leistungsfähigen Versorgungsnetzwerk-Blöcken. Zu den wichtigsten Neuerungen des Pakets gehören die hoch effiziente Kollisionserkennung, Geometrieschnittpunkt-Blöcke, ImportInstance (CAD) zur Konvertierung von Kurven und Blöcken sowie Kollektoren für verknüpfte Elementen, die Einschränkungen in der

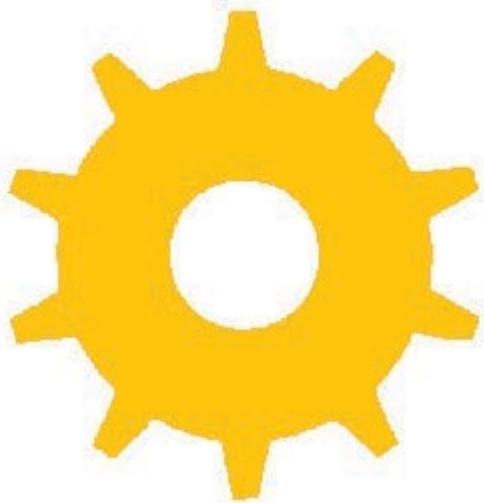
Einschrnkungen in der Revit-API aufheben.  
Weitere Informationen ber die gesamte Bandbreite an verfgbaren Blcken finden Sie im Wrterbuch auf BimorphNodes.



## **BUMBLEBEE FOR DYNAMO**

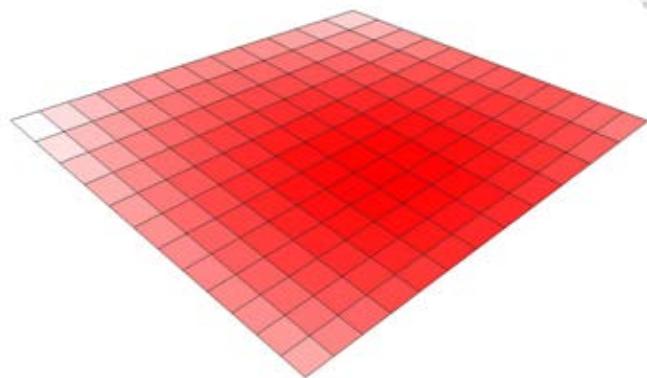
BumbleBee ist ein Interoperabilitäts-Plugin für Excel und Dynamo, das die Möglichkeiten von Dynamo zum Lesen und Schreiben von Excel-Dateien erheblich verbessert.





### CLOCKWORK FOR DYNAMO

Clockwork ist eine Sammlung benutzerdefinierter Blöcke für die visuelle Programmierungsumgebung von Dynamo. Es enthält viele Revit-bezogene Blöcke, aber auch zahlreiche Blöcke für verschiedene andere Zwecke wie Listenverwaltung, mathematische Operationen, Zeichenkettenoperationen, Einheitenumrechnungen, geometrische Operationen (vor allem Rahmen, Netze, Ebenen, Punkte, Oberflächen, UVs und Vektoren) und Unterteilung von Oberflächen.



**DATA**

ds

DataShapes ist ein Paket zur Erweiterung der Benutzerfunktionen von Dynamo-Skripts. Der Schwerpunkt liegt dabei auf der Bereitstellung zusätzlicher Funktionen in Dynamo Player. Weitere Informationen finden Sie auf <https://data-shapes.net/>. Sie möchten beeindruckende Arbeitsabläufe für Dynamo Player erstellen? Verwenden Sie dieses Paket.



DYNAM  
SAP



DynamoSAP ist eine parametrische Benutzeroberfläche für SAP2000, die auf Dynamo aufsetzt. Das Projekt versetzt Konstrukteure und Ingenieure in die Lage, strukturelle Systeme auf generative Weise in SAP zu entwickeln und zu analysieren, indem das SAP-Modell mit Dynamo betrieben wird. Das Projekt schreibt einige alltägliche Arbeitsabläufe vor, die in den eingeschlossenen Beispieldateien beschrieben sind, und bietet eine breite Palette an Möglichkeiten zur Automatisierung typischer Aufgaben in SAP.

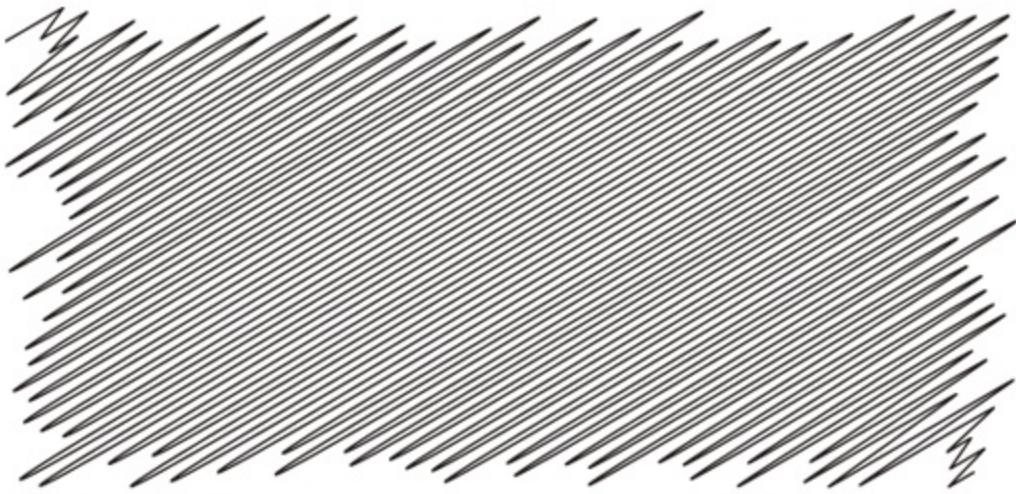


DYNAM  
UNFOLI



Diese Bibliothek erweitert die Funktionalität von Dynamo/Revit, indem Benutzer in die Lage versetzt werden, Oberflächen und Poly-Oberflächengeometrie abzuwickeln. Mithilfe dieser Bibliothek können Benutzer Oberflächen zunächst in eine planare Tessellationstopologie überführen und sie dann mithilfe der Protogeometrie-Werkzeuge von Dynamo abwickeln. Dieses Paket beinhaltet zudem einige experimentelle Blöcke und einige grundlegende Beispieldateien.





DYNAST

# DYNASTRATOR

Importieren Sie Vektorillustrationen aus Illustrator oder dem Internet mit .svg. Auf diese Weise können Sie manuell erstellte Zeichnungen für parametrische Operationen in Dynamo importieren.



Energy Analysis for Dynamo ermöglicht die parametrische Energiemodellierung und ganzheitliche Energieanalyseabläufe in Dynamo 0.8. Energy Analysis for Dynamo ermöglicht es dem Benutzer, ein Energiemodell in Autodesk Revit zu konfigurieren, es für die Green Building Studio for DOE2-Energieanalyse einzureichen und die von der Analyse zurückgegebenen Ergebnisse weiterzuverarbeiten. Das Paket wird in Studio CORE von Thornton Tomasetti entwickelt.





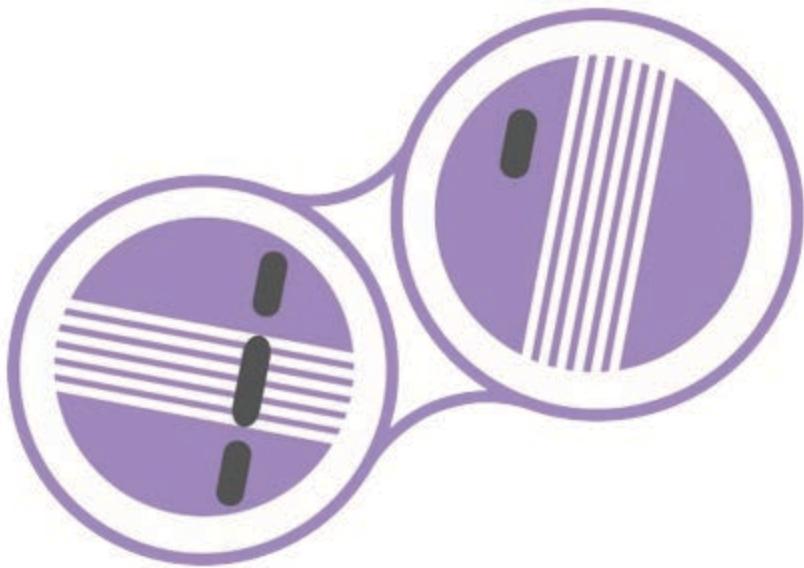
Firefly ist eine Sammlung von Blöcken, die es Dynamo ermöglichen mit Eingabe-/Ausgabegeräten wie dem Arduino Micro Controller zu kommunizieren. Da der Datenfluss "live" erfolgt, eröffnet Firefly viele Möglichkeiten für interaktives Prototyping zwischen digitalen und physischen Welten über Webcams, Mobiltelefone, Gamecontroller, Sensoren und vielem mehr.



LunchBox ist eine Sammlung von wiederverwendbaren Geometrie- und Datenverwaltungsblöcken. Die Werkzeuge wurden mit Dynamo 0.8.1 und Revit 2016 getestet. Das Werkzeug enthält Blöcke für Oberflächenverkleidungen, Geometrie, Revit-Datensammlungen und vieles mehr.



## MANTIS SHRIMP



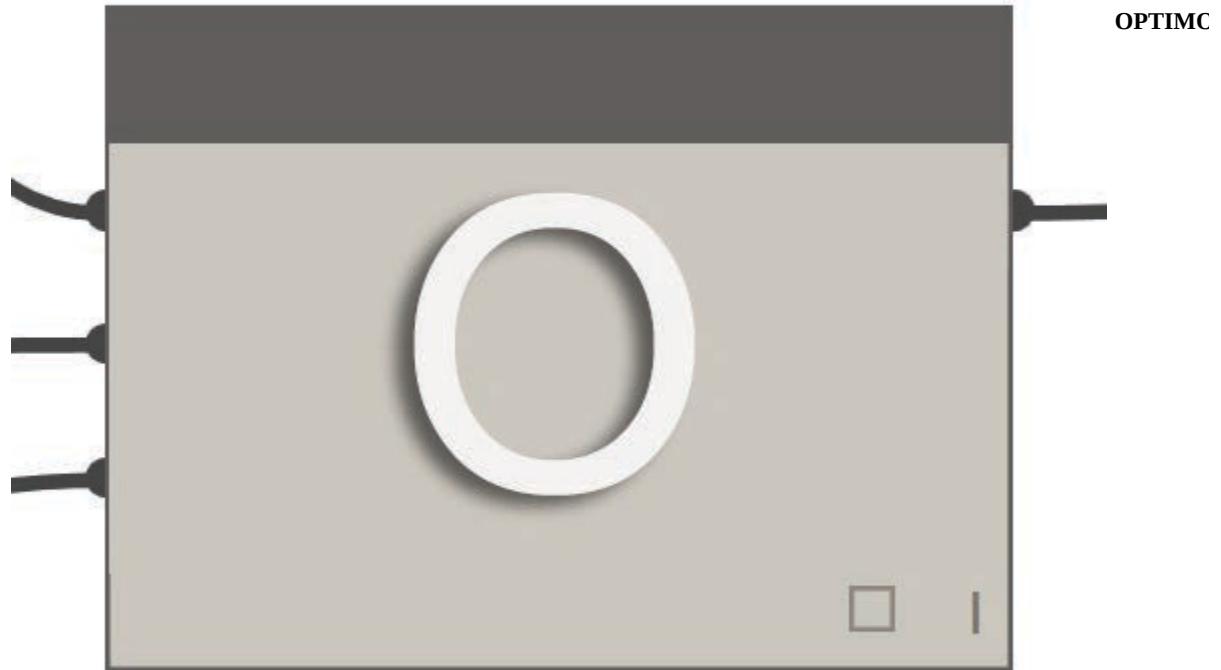
Mantis Shrimp ist ein Interoperabilitätsprojekt, das Ihnen den problemlosen Import von Grasshopper- und/oder Rhino-Geometrie in Dynamo ermöglicht.



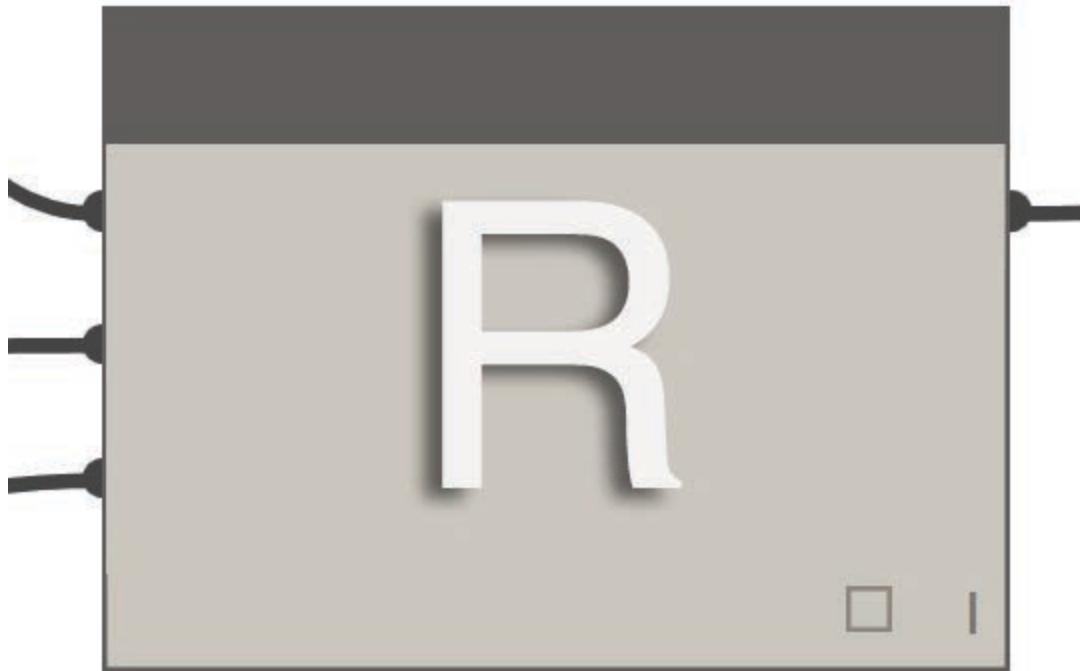


Das Dynamo Mesh Toolkit enthält viele nützliche Werkzeuge für die Arbeit mit Netzgeometrie. Zu den Funktionen dieses Pakets gehören die Möglichkeiten zum Importieren von Netzen mit externen Dateiformaten, zum Erstellen von Netzen aus bereits vorhandenen Dynamo-Geometrieobjekten und zum manuellen Erstellen von Netzen aus Scheitelpunkten und Verbindungsinformationen. Darüber hinaus enthält dieses Toolkit Werkzeuge zum Ändern und Reparieren von Netzgeometrie.



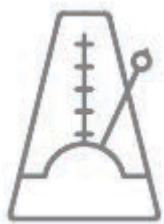


Optimo bietet Dynamo-Benutzern die Möglichkeit, selbstdefinierte Konstruktionsprobleme mithilfe verschiedener evolutionärer Algorithmen zu optimieren. Die Benutzer können die Ziele eines Problems sowie spezielle Eignungsfunktionen definieren.



**RHYNAL**

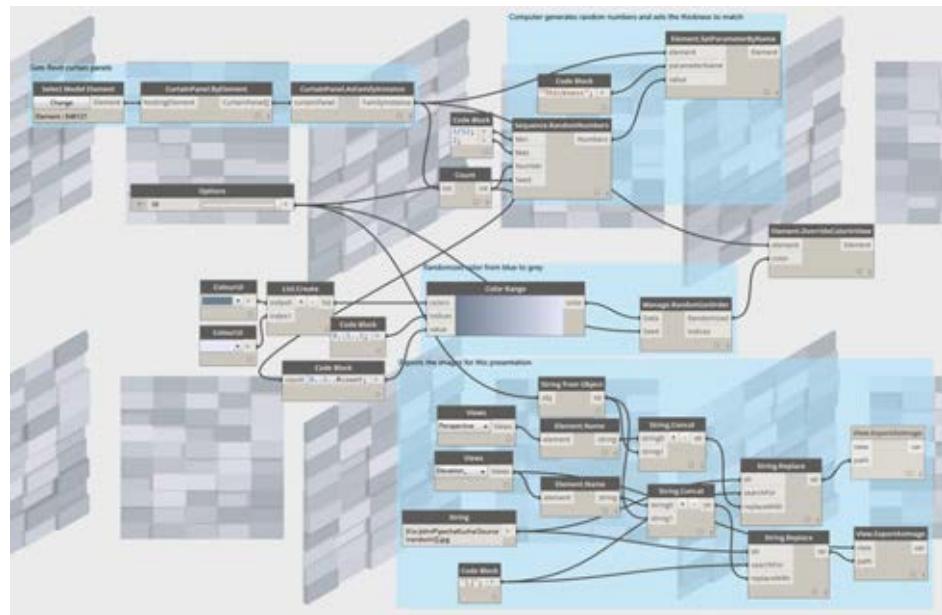
Die Rhynamo-Blockbibliothek bietet Benutzern die Möglichkeit, Rhino 3DM-Dateien in Dynamo einzulesen und zu bearbeiten. Rhynamo übersetzt Rhino-Geometrie mithilfe der OpenNURBS-Bibliothek von McNeel in Geometrie, die in Dynamo verwendbar ist, und ermöglicht dadurch neue Arbeitsabläufe, mit denen Geometrie und Daten fließend zwischen Rhino und Revit ausgetauscht werden können. Dieses Paket enthält auch einige experimentelle Blöcke, die den "Live"-Zugriff auf die Rhino-Befehlszeile ermöglichen.



**RHYTHM**

Auf den ersten Blick ist  
Rhythm nichts  
Besonderes. Es bietet  
keinen komplexen Code  
oder andere aufregende

Dinge. Rhythm steht jedoch für das Ergebnis einer praktischen Denkweise und Gewissenhaftigkeit. Die Idee hinter diesem Paket besteht darin, Benutzer bei der Verwendung von Rhythm in Revit mit Dynamo zu unterstützen. Rhythm besteht in erster Linie aus sofort einsatzfähigen Dynamo-Blöcken, die in der Revit-Umgebung auf intelligente Weise genutzt werden können.



# Dynamo-Beispieldateien

## Dynamo-Beispieldateien

Die im Dynamo Primer enthaltenen Beispieldateien sind nach Kapitel und Abschnitt aufgeführt.

Klicken Sie mit der rechten Maustaste auf die Dateien und wählen Sie "Speichern unter..." aus.

### Einführung

Abschnitt	Datei herunterladen
-----------	---------------------

Was ist die visuelle Programmierung? [Visual Programming - Circle Through Point.dyn](#)

### Anatomie einer Dynamo-Definition

Abschnitt	Datei herunterladen
-----------	---------------------

Voreinstellungen [Presets.dyn](#)

### Die Bausteine von Programmen

Abschnitt	Datei herunterladen
-----------	---------------------

Daten [Building Blocks of Programs - Data.dyn](#)  
Math [Building Blocks of Programs - Math.dyn](#)  
Logik [Building Blocks of Programs - Logic.dyn](#)  
Zeichenfolgen [Building Blocks of Programs - Strings.dyn](#)  
Farbe [Building Blocks of Programs - Color.dyn](#)

### Geometrie für computergestützte Konstruktion

Abschnitt	Datei herunterladen
-----------	---------------------

Geometrie – Überblick [Geometry for Computational Design - Geometry Overview.dyn](#)  
Vektoren [Geometry for Computational Design - Vectors.dyn](#)  
[Geometry for Computational Design - Plane.dyn](#)  
[Geometry for Computational Design - Coordinate System.dyn](#)  
Punkte [Geometry for Computational Design - Points.dyn](#)  
Kurven [Geometry for Computational Design - Curves.dyn](#)  
Oberflächen [Geometry for Computational Design - Surfaces.dyn](#)  
[Surface.sat](#)

### Konstruieren mit Listen

Abschnitt	Datei herunterladen
-----------	---------------------

Was ist eine Liste? [Lacing.dyn](#)  
Arbeiten mit Listen [List-Count.dyn](#)  
[List-FilterByBooleanMask.dyn](#)  
[List-GetItemAtIndex.dyn](#)  
[List-Operations.dyn](#)  
[List-Reverse.dyn](#)  
[List-ShiftIndices.dyn](#)

Listen von Listen	<a href="#">Chop.dyn</a>
<a href="#">Combine.dyn</a>	
<a href="#">Flatten.dyn</a>	
<a href="#">Map.dyn</a>	
<a href="#">ReplaceItems.dyn</a>	
<a href="#">Top-Down-Hierarchy.dyn</a>	
<a href="#">Transpose.dyn</a>	
n-dimensionale Listen	<a href="#">n-Dimensional-Lists.dyn</a>
<a href="#">n-Dimensional-Lists.sat</a>	

### Codeblöcke und DesignScript

Abschnitt	Datei herunterladen
DesignScript-Syntax	<a href="#">Dynamo-Syntax Attractor-Surface.dyn</a>
Kurzschrifweisnen	<a href="#">Obsolete-Nodes Sine-Surface.dyn</a>
Funktionen	<a href="#">Functions_SphereByZ.dyn</a>

### Dynamo für Revit

Abschnitt	Datei herunterladen
Auswählen	<a href="#">Selecting.dyn</a>
<a href="#">ARCH-Selecting-BaseFile.rvt</a>	
Bearbeiten	<a href="#">Editing.dyn</a>
<a href="#">ARCH-Editing-BaseFile.rvt</a>	
Erstellen	<a href="#">Creating.dyn</a>
<a href="#">ARCH-Creating-BaseFile.rvt</a>	
Anpassen	<a href="#">Customizing.dyn</a>
<a href="#">ARCH-Customizing-BaseFile.rvt</a>	
Dokumentation	<a href="#">Documenting.dyn</a>
<a href="#">ARCH-Documenting-BaseFile.rvt</a>	

### Wörterbücher in Dynamo

Abschnitt	Datei herunterladen
Raum-Wörterbuch	<a href="#">RoomDictionary.dyn</a>

### Benutzerdefinierte Blöcke

Abschnitt	Datei herunterladen
Erstellen eines benutzerdefinierten Blocks	<a href="#">UV-CustomNode.zip</a>
In der Bibliothek publizieren	<a href="#">PointsToSurface.dyf</a>
Python-Blöcke	<a href="#">Python-CustomNode.dyn</a>
Python und Revit	<a href="#">Revit-Doc.dyn</a>
Python und Revit	<a href="#">Revit-ReferenceCurve.dyn</a>
Python und Revit	<a href="#">Revit-StructuralFraming.zip</a>

### Pakete

Abschnitt	Datei herunterladen
Fallstudie zu Paketen: Mesh Toolkit	<a href="#">MeshToolkit.zip</a>

Publizieren eines Pakets

[MapToSurface.zip](#)

Zero-Touch-Import

[ZeroTouchImages.zip](#)