

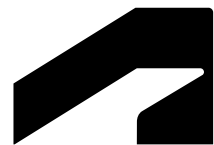
Dynamo の「イロハ」 ~ 番外編 :

Dynamo ! Python ! API ! 実は簡単 ? !
経験ゼロの現役大学生が徹底解説 ~

末廣 隆介, 日下部 達哉

本日の主な講師は、弊社 Intern（末廣さん）です

- 本日の流れ
 - セミナー本編 by 末廣 （Dynamo for Revit 上での、Python の使い方 を解説）
 - 自由討論 by 末廣 & 日下部 （Dynamo 上で Python を「初めて触ってみた」感想）
 - 補足説明 by 日下部 （Dynamo for Civil 3D 上での、Python の使い方 に関して補足）
- このセミナーを通したメッセージ：**Dynamo を使用している方**
 - このセミナーの手順に従うと、Dynamo 上で Python が書けます。
- このセミナーを通したメッセージ：**Dynamo を使用したことがない方（調査中を含む）**
 - Dynamo 未経験の学生さんでも、一か月でここまで出来ます。あなたにも、きっと出来ます。
 - Dynamo 上で Python が書けると、Dynamo をもっと便利に使えます。
 - Dynamo の基礎を学んで、Python の世界に踏み出したくなったら、このセミナーがあります。



セミナー 本編

Dynamo の「イロハ」～番外編～

今日のコンテンツ

自己紹介

API とは？

API 実装後の最終形態

API リファレンス (API Docs) を読み解く

Python を書き始める下準備

Revit タイプと Dynamo タイプの変換方法

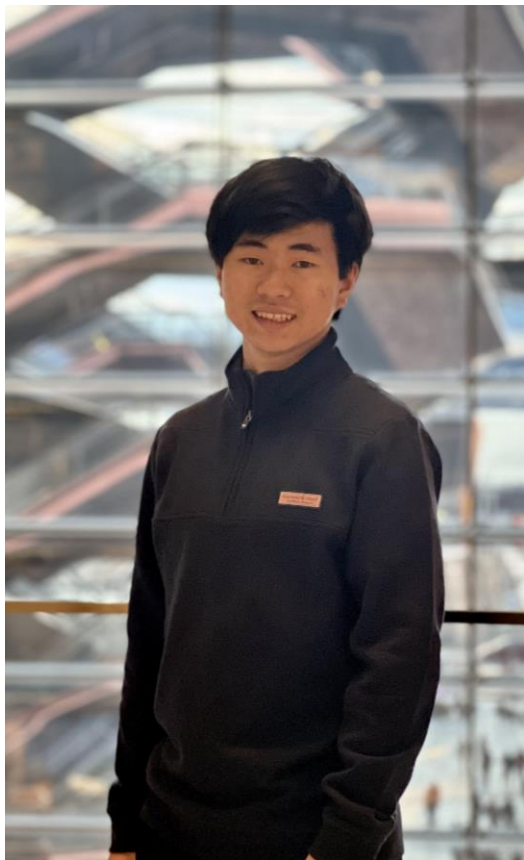
デバッグのテクニック

Python を使って Dynamo を実装するメリット



自己紹介：末廣隆介

技術営業 (テクニカルセールス) のインターンです！



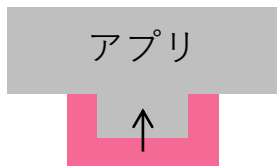
基本的なバックグラウンド

- **在籍校:** カリフォルニア大学バークレー校 (UC Berkeley)
- **学年:** 二年生
- **専攻:** 数学、コンピューターサイエンス、ビジネス
- **今まで住んだ事がある場所:** ボストン (生まれ)、ロンドン (6年間)、ニューヨーク (5年)、東京 (5年)、バークレー (1年)
- **経験:** コンピュータービジョンを用いてマスク着用の有無、また正しくつけているか検知するプログラム、ゲーム会社のユーザー分析
- **趣味:** テニス、ゴルフ、旅行、食べること



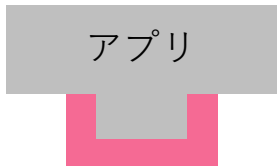
API (Application Programming Interface) とは

GUI：人間が
アプリに指示



これ
やって

API：プログラムが
アプリに指示



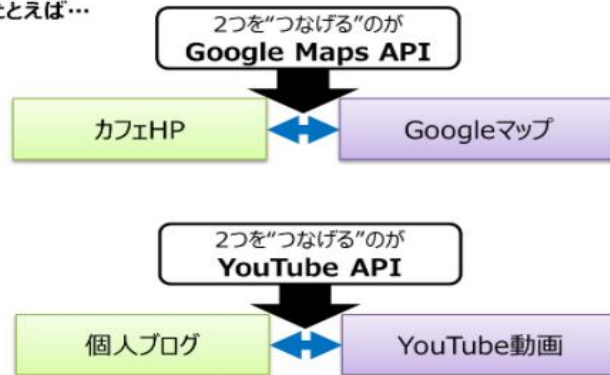
これ
やって

 **AUTODESK**
Revit



私たちが Dynamo 上で
Python を書き、Revit の
サービスを使うよう指示す
る必要がある

たとえば…

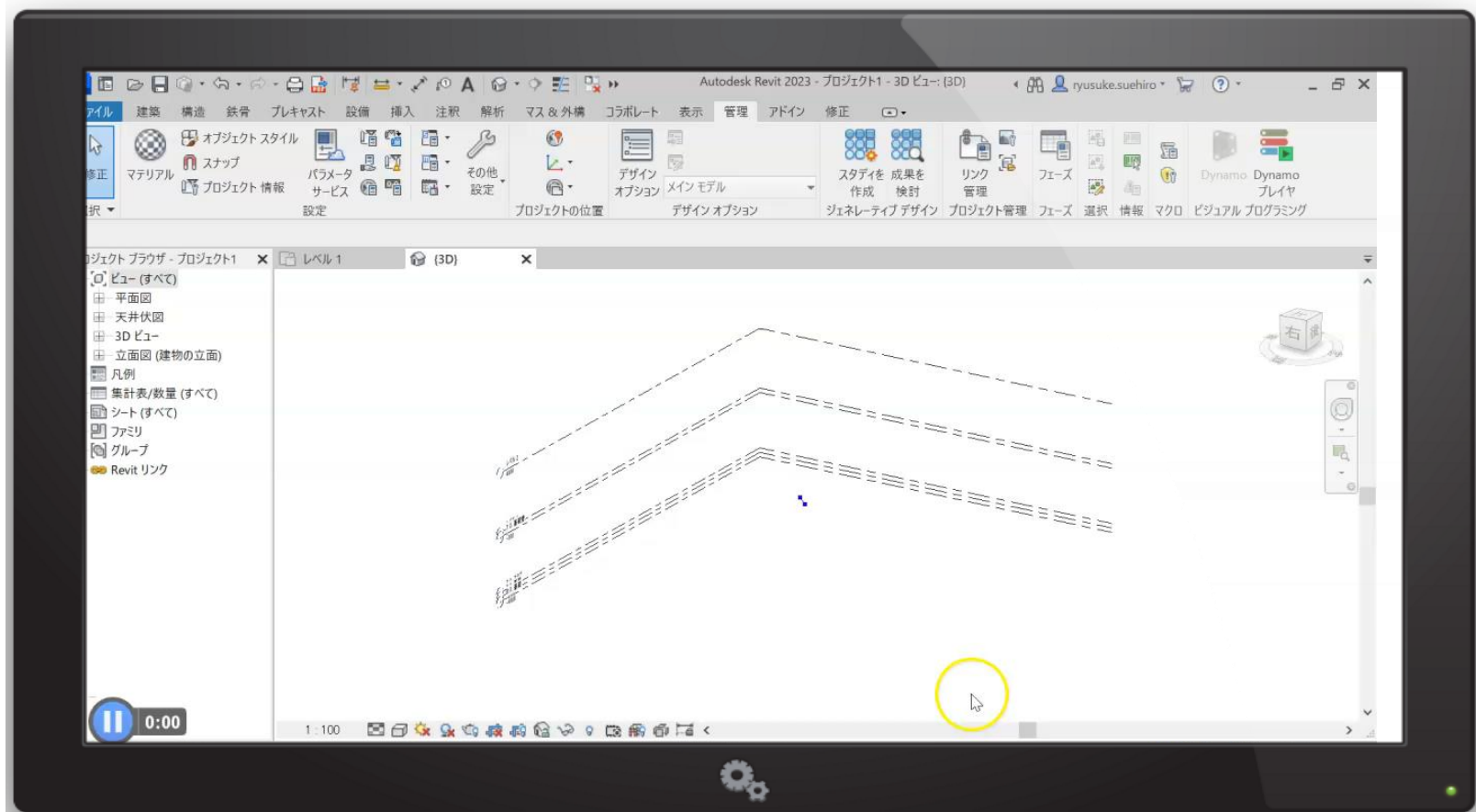


API があると、複数のサービスを
自動で接続できるようになる

Revit にある膨大な
サービスを自動的に使
えるようになれば**無限**
の可能性がある！

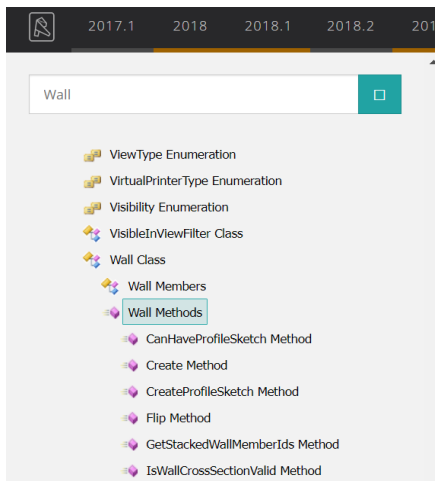
結果動画

Dynamo と Python Script を使うと、一瞬で壁が作れる！



Revit API Docs の検索、読み方

Revit API Docs で使いたい Method (関数) を検索



注意：壁などを作りたい場合は作りたいものが所属する“Wall”というクラスから検索し、その後“Create”を検索する

複数同じメソッドがある場合は、必要なパラメータ情報と生成結果の説明 (Description) をもとに選択

Overload List

Name	Description
<code>Create(Document, IList Curve, Boolean)</code>	Creates a non rectangular profile wall within the project using the default wall type.
<code>Create(Document, Curve, ElementId, Boolean)</code>	Creates a new rectangular profile wall within the project using the default wall style.
<code>Create(Document, IList Curve, ElementId, ElementId, Boolean)</code>	Creates a non rectangular profile wall within the project using the specified wall type.
<code>Create(Document, IList Curve, ElementId, ElementId, Boolean, XYZ)</code>	Creates a non rectangular profile wall within the project using the specified wall type and normal vector.
<code>Create(Document, Curve, ElementId, ElementId, Double, Double, Boolean, Boolean)</code>	Creates a new rectangular profile wall within the project using the specified wall type, height, and offset.

注意：同じメソッドでもパラメータの種類によって作る壁の種類が変わる

パラメータの詳しい情報を確認

```
public static Wall Create(  
    Document document,  
    Curve curve,  
    ElementId wallTypeId,  
    ElementId levelId,  
    double height,  
    double offset,  
    bool flip,  
    bool structural  
)
```

注意：太字のものは必須。その他は指定しないとデフォルトの値が使われる

後ほどこのように Python を実装するためにはどんな情報を用意すべきか分かる

```
wall = Wall.Create(doc, curve, wallTypeId, levelId, 250.0, 0.0, False, True)
```




Revit API Docs



All Images Videos News Shopping More Tools

About 401,000 results (0.33 seconds)

https://www.revitapidocs.com

Revit API Docs

Online Documentation for Autodesk's **Revit API**: 2015, 2016, 2017, 2017.1, 2018.

2020

Use the search box or collapsible namespace navigation to ...

2021.1

Use the search box or collapsible namespace navigation to ...

2019

Use the search box or collapsible namespace navigation to ...

2022

Use the search box or collapsible namespace navigation to ...

More results from [revitapidocs.com](https://www.revitapidocs.com) »



https://apidocs.co

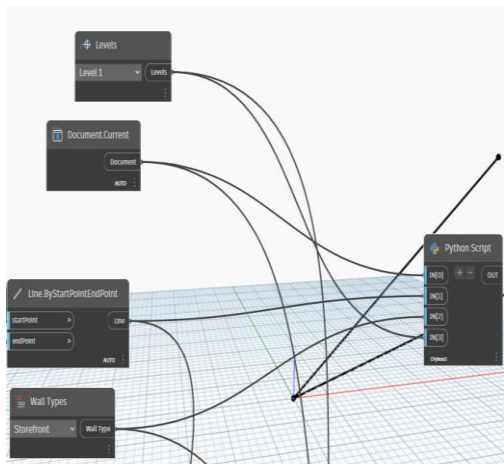


Python Script で API を実装する下準備

API Docs にあるパラメータを Dynamo で作り、Python Script が使えるような準備が必要

Python Script の Input の設定

API Docs で見た
パラメータを
Dynamo で作り、
Python Script の
IN につなげる



Python Script ライブラリーの読み込み

```
RevitBoilerplate.py
1 import clr
2 import sys
3 sys.path.append('C:\Program Files (x86)\IronPython 2.7\Lib')
4 import System
5 from System import Array
6 from System.Collections.Generic import *
7 clr.AddReference("ProtoGeometry")
8 from Autodesk.DesignScript.Geometry import *
9 clr.AddReference("RevitNodes")
10 import Revit
11 clr.ImportExtensions(Revit.Elements)
12 clr.ImportExtensions(Revit.GeometryConversion)
13 clr.AddReference("RevitServices")
14 import RevitServices
15 from RevitServices.Persistence import DocumentManager
16 from RevitServices.Transactions import TransactionManager
17
18 clr.AddReference("RevitAPI")
19 clr.AddReference("RevitAPIUI")
20
21 import Autodesk
22 from Autodesk.Revit.DB import *
23 from Autodesk.Revit.UI import *
24
25 doc = DocumentManager.Instance.CurrentDBDocument
26 uiapp = DocumentManager.Instance.CurrentUIApplication
27 app = uiapp.Application
28 uidoc = uiapp.ActiveUIDocument
29
30 #####OK NOW YOU CAN CODE#####
```

ボイラープレート
からコピーして
Python が Revit の
API を認識できる
ようセットアップ。
このステップを忘
れると必ずエラー
が出る

* IronPython2, CPython3
どちらでも使用可能

- Search Term
- Create Method (Document, IList(Curve), Boc
 - Create Method (Document, Curve, ElementId
 - Create Method (Document, IList(Curve), Eler
 - Create Method (Document, IList(Curve), Eler
 - Create Method (Document, Curve, ElementId
 - CreateProfileSketch Method
 - Flip Method
 - GetStackedWallMemberIds Method
 - IsWallCrossSectionValid Method
 - RemoveProfileSketch Method
 - Wall Properties
 - WallCrossSection Enumeration
 - WallFoundation Class
 - WallFoundationType Class
 - WallFunction Enumeration
 - WallKind Enumeration
 - WallLocationLine Enumeration
 - WallSide Enumeration

Create Method (Document, Curve, ElementId, ElementId, Double, Double, ...

[Wall Class](#) [Example](#) [See Also](#)

Creates a new rectangular profile wall within the project using the specified wall type, height, and offset.

Syntax

C#

```
public static Wall Create(  
    Document document,  
    Curve curve,  
    ElementId wallTypeId,  
    ElementId levelId,  
    double height,  
    double offset,  
    bool flip,  
    bool structural  
)
```



Visual Basic



Your new development career awaits. Check out the latest listings.

ADS VIA CARBON



Dynamo の様々なタイプから Revit への変換方法

現在あるエレメントが何タイプか確認



Dynamo 上で一つずつのコンポーネントが何のタイプなのかが見られる。

Revit と書いてない場合は Revit への変換が必要
Revit と書いてある場合は Unwrap が必要 (後述)

API 実装に必要な Revit Type (オレンジ) と照合

```
public static Wall Create(  
    Document document,  
    Curve curve,  
    ElementId wallTypeId,  
    ElementId levelId,  
    double height,  
    double offset,  
    bool flip,  
    bool structural  
)
```

変換方法の検索

ネットや Dynamo フォーラム
で変換する前と後のタイプを
キーワードにして検索！もし
しくは API Docs でエレメント
の継承ストラクチャーを見る

変換方法が分からない場合：

変換方法がすぐわかる場合：

先程 IN に繋げた Dynamo エレメントを一つずつ読み込み、必須パラメータ
の数と一致するか確認し、それぞれ適正な Revit へタイプ変換を行う

```
35 document = IN[0]  
36 curve = IN[1].ToRevitType()  
37 wallTypeId = UnwrapElement(IN[2]).Id  
38 levelId = UnwrapElement(IN[3]).Id  
39
```

Dynamo (D) ファイル(F) 編集(E) 表示(V) パッケージ(P) ジェネレーティブデザイン ヘルプ(H) 拡張機能(X) ! イメージとして書き出し

Wall.dyn

Library

Search

- > Dictionary
- > Display
- > Geometry
- > ImportExport
- > Input
- > List
- > Math
- > Revit

The workflow consists of the following nodes and connections:

- Levels** (Revit) - Set to T.O. 基礎壁. Connected to the **Python Script** node.
- Document.Current** (Revit) - Connected to the **Python Script** node. This node is highlighted with a yellow circle.
- Line.ByStartPointEndPoint** (Geometry) - Connected to the **Python Script** node.
- Wall.Types** (Revit) - Set to 保持 - 300 mm コンクリート. Connected to the **Python Script** node.
- Python Script** (Python) - Contains a script that takes inputs from the other nodes and outputs a wall object.

3D Viewport: Shows a grid with a red line and a black arrow pointing to a wall element.

0:00

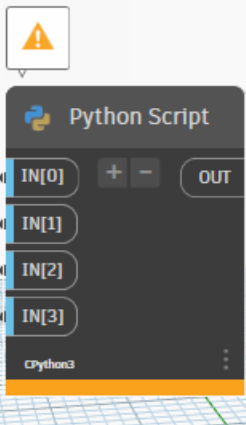
laborate, build, and
success with Shortcut.
free.

ADS VIA CARBON



Python Script のデバッグの仕方

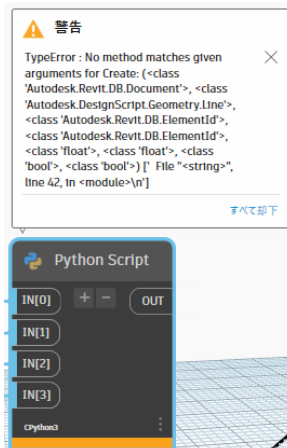
Dynamo の Python Script がオレンジ色で警告サインが見える



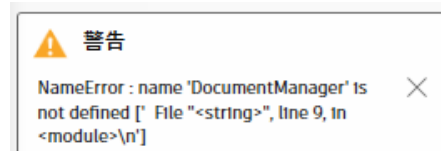
警告サインを押す



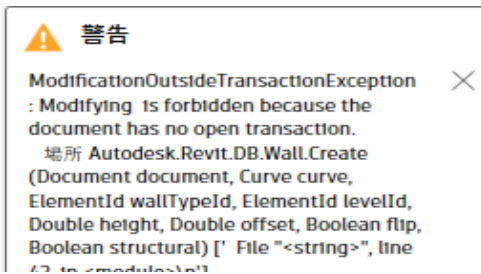
エラーの種類、エラー箇所の行数が見える



Not Defined など明確なエラーの場合は指定された行に行き、対処



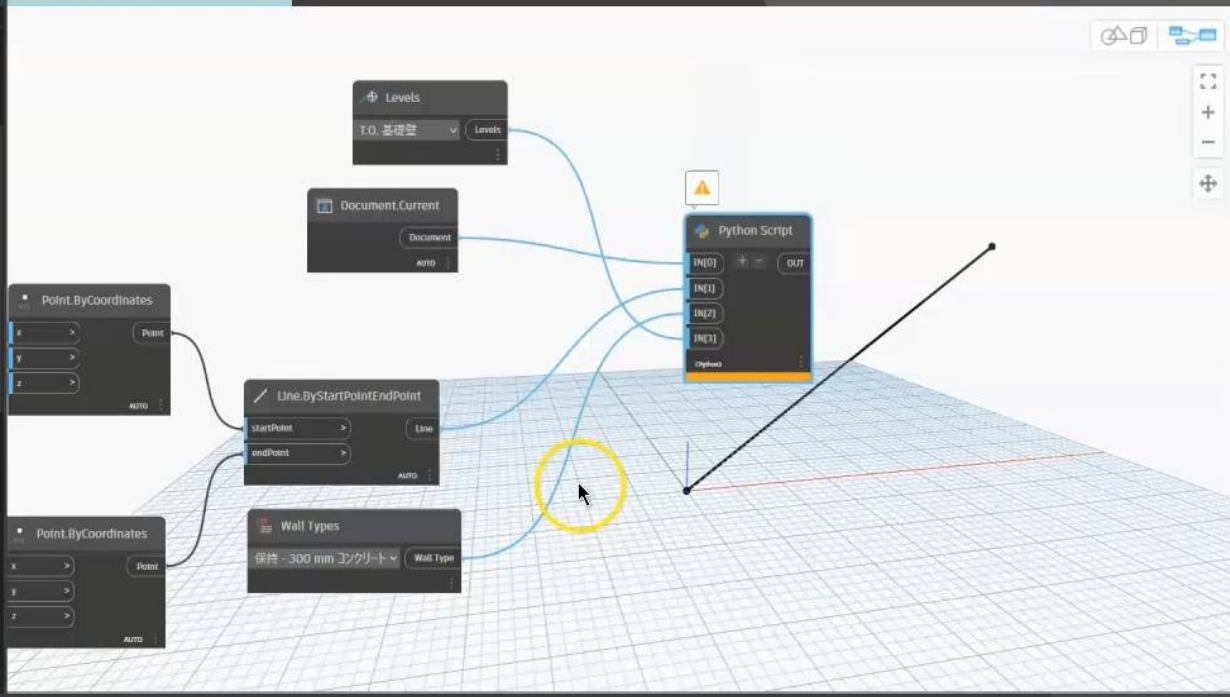
複雑なエラーの場合はすぐ検索！



Wall.dyn

イメージとして書き出し

- Geometry
- ByBestFitThroughPoints
Line + Geometry
- ByTangency
Line + Geometry
- ByStartPointDirectionLength
Line + Geometry
- Direction
Line ? Geometry
- ByControlPoints (points)
NurbsCurve + Geometry
- ByControlPoints (points, degree)
NurbsCurve + Geometry
- ByControlPoints
(points, degree, closeCurve)
NurbsCurve + Geometry
- ByControlPointsWeightsKnots
NurbsCurve + Geometry
- ByPoints (points)
NurbsCurve + Geometry
- ByPoints (points, closeCurve)
NurbsCurve + Geometry



なぜ Python と Dynamo を試してみるべきか

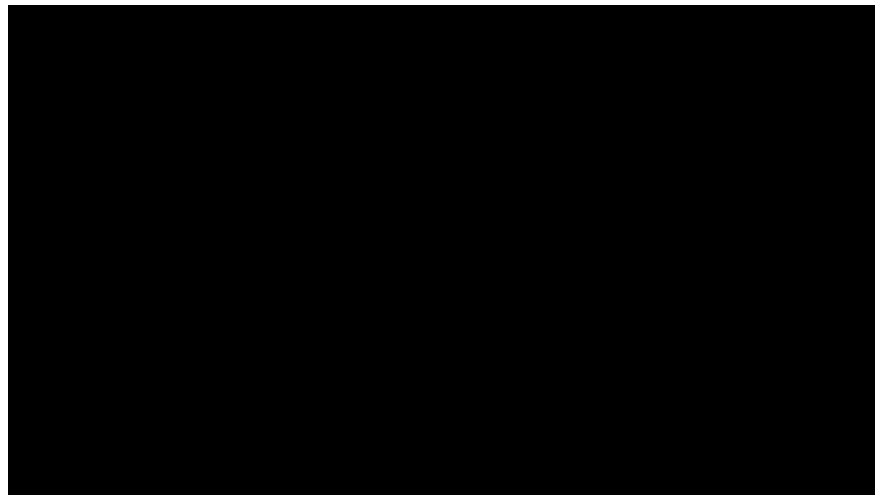
反復作業を効率化できる

例：モデル作成に要する時間を
2週間から2日に短縮 ([こちら](#))

簡単に壁の間隔や高さもカスタマイズできる

Dynamo ノードだけでは不可能な
Revit カスタマイズもできる

- ・ Dynamo で未対応の要素の処理
(CAD リンク, MEP 製造用パーツ, ...)
- ・ 複雑なリスト処理 (for, while, ...)
- ・ Python ライブラリを活用した計算
(NumPy, Pandas, ...)

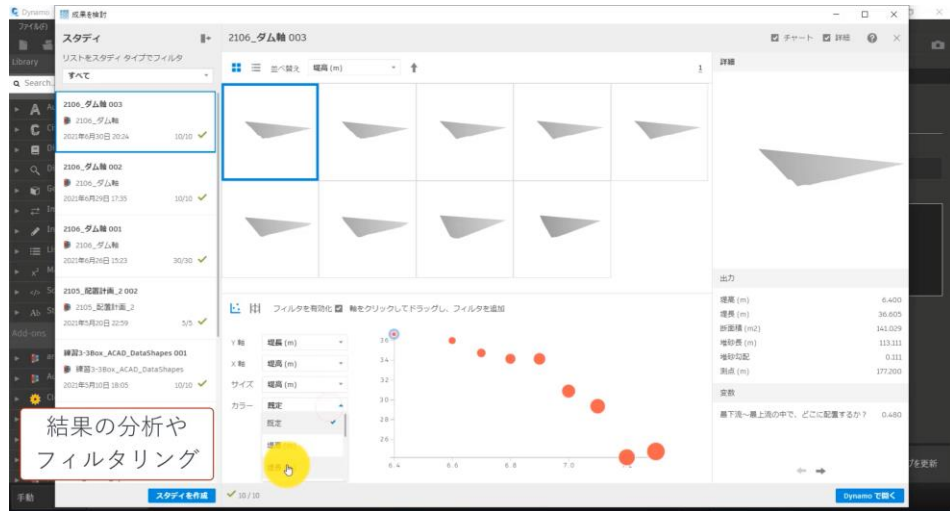


なぜ Python と Dynamo を試してみるべきか 2

オートデスクの最先端技術を
試せるので、楽しい

例：Dynamo for Civil 3D, Python に加えて
Generative Design の技術も活用

→ 砂防施設の最適配置 (詳細は [こちら](#))





Free Talk Session

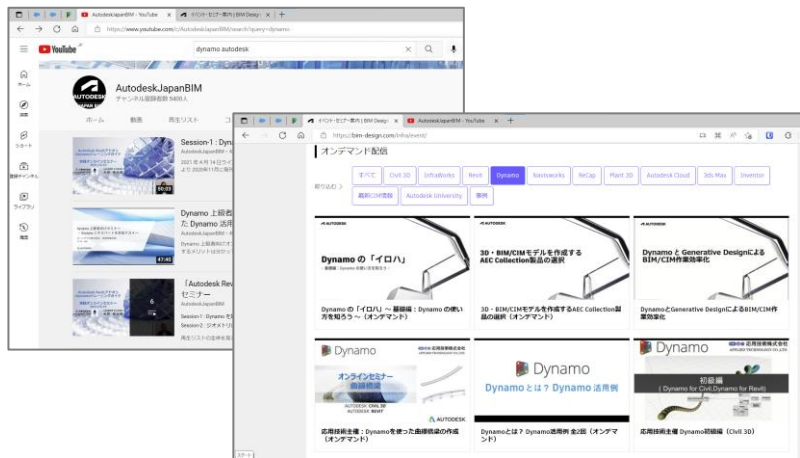
Dynamo 学習、最初はどこから？

様々な動画コンテンツ

Dynamo の操作方法を確認する、「手順書」

- YouTube でいろいろ検索 ([こちら](#))
- Autodesk Japan のセミナー
(建築編は [こちら](#)、土木編は [こちら](#))

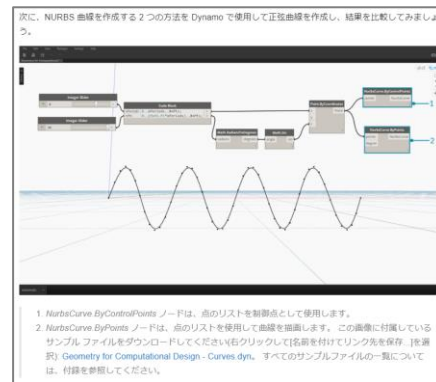
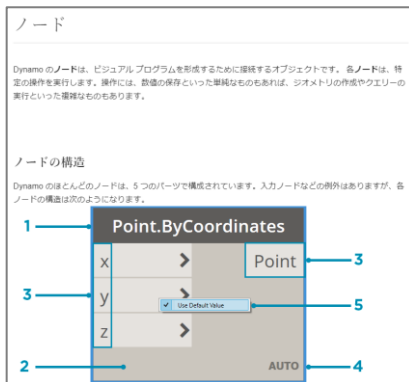
* 今回のセミナーのアーカイブは [こちら](#) に掲載予定



Dynamo Primer

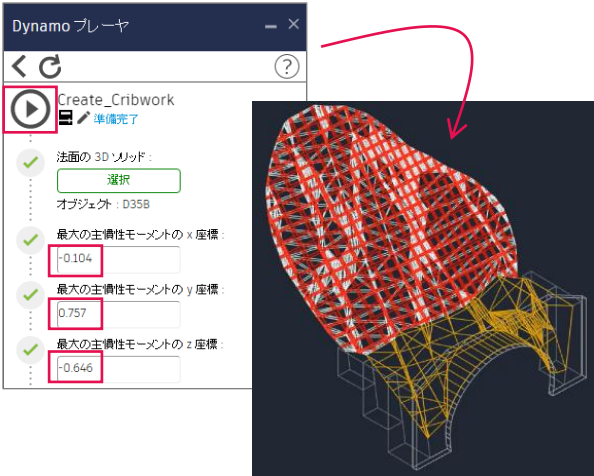
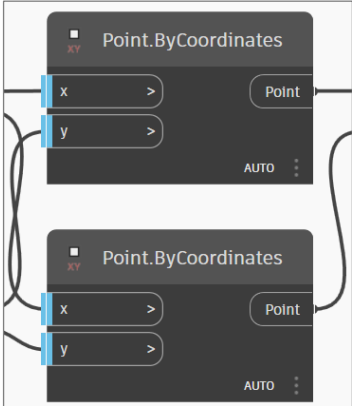
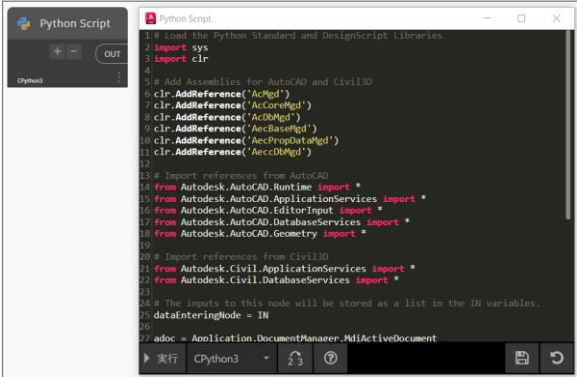
Dynamo の機能を網羅的に解説した、「辞書」

- Dynamo の起動、UI
- ビジュアルプログラムの構造
- 形状を作成 (ジオメトリの操作)
- データを整理 (リスト、ディクショナリ)
- プログラムを記述 (Code Block, Python)
- ...



Dynamo の特色は何だと思うか？

- ビジュアルプログラミングの「分かりやすさ」と、Python の「拡張性」を両立
- 使い分けのカギは、ハンズオン動画を見たり自分で実際に触って試行錯誤をすること！

Dynamo Player	Dynamo	Dynamo + Python
<p>既にある Dynamo のプログラムに、入力値だけ指定して、処理を実行</p> 	<p>ビジュアルプログラミングでプログラムを作成</p> 	<p>Dynamo のノードだけでは実現できない機能を、Python を書いて実現</p> 

オートデスクの特色は何だと思うか？

- オープンなところ
 - API が公開されているので、自由にカスタマイズができる（Web API も！）
 - ユーザ同士のコミュニティがあるので、知見が Web 上に蓄積されている

Design Automation API Documentation / Design Automation API / Developer's Guide

Version 3

Design Automation API

The Design Automation API provides the ability to use the core API's of your favorite CAD Platform to run automated jobs. These jobs could be highly repetitive or frequent, or require power. With the Design Automation API, you can offload that processing to the Forge Platform and scale and efficiency.

Currently, the Design Automation API supports AutoCAD, 3ds Max, Inventor and Revit. Read on for details about each:

Design Automation API for AutoCAD

Developer's Guide

Step-by-Step Tutorials

Code Samples & Blog Posts

API Reference

Change History

Step-by-Step Tutorials

- > Design Automation API for 3ds Max
- > Design Automation API for AutoCAD
- > Design Automation API for Inventor
- > Design Automation API for Revit

About this Tutorial

- Task 1 - Convert Revit Add-in
- Task 2 - Obtain an Access Token
- Task 3 - Create a Nickname
- Task 4 - Upload an AppBundle
- Task 5 - Publish an Activity
- Task 6 - Prepare Cloud Storage
- Task 7 - Submit a WorkItem

Postman is a popular tool that provides an easy-to-use interface to send HTTP requests. The Postman Collection at <https://github.com/Autodesk-Forge/forge-tutorial-postman/tree/master/DAARevit> collates all the HTTP requests used in this Tutorial. If you are familiar with Postman, you can import this Postman Collection and use Postman to send requests instead of curl.

On the Postman Collection, requests are grouped by task. The group has the same name as the corresponding task in the tutorial.

Design Automation for Revit

- About this Tutorial
- Task 1 - Convert Revit Add-in
- Task 2 - Obtain an Access Token
- Task 3 - Create a Nickname
- Task 4 - Upload an AppBundle
- Task 5 - Publish an Activity
- Task 6 - Prepare Cloud Storage
- Task 7 - Submit a WorkItem

Task 4 - Upload AppBundle

AUTODESK Knowledge Network

製品 サポート 学習 コミュニティ

投稿者	投稿日時	最終投稿日時	件の返信	件の閲覧数
sakoumih, yasuyuki_kido	10-26-2021 07:25 AM	最終投稿日時: 10-28-2021 07:10 PM	12	205
Kantarou_Nagami, ACIX	10-22-2021 04:40 PM	最終投稿日時: 10-28-2021 02:37 PM	17	308
sn2019, marishimode	10-05-2021 10:24 AM	最終投稿日時: 10-28-2021 01:46 PM	3	123
sakco102	10-26-2021 02:54 PM	最終投稿日時: 10-27-2021 07:51 AM	2	73

python last visit

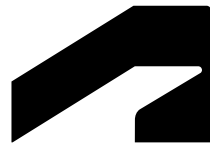
Organized cogopoint list by number

Points from Alignment projected on featurelines/offset alignments or polylines

Autocad polyline to a polycurve

Multileaders Generated from Dynamo

dynamo, python



補足説明

Dynamo for “Civil 3D” & Python の場合

結果動画

Dynamo と Python Script を使うと、一瞬でサブアセンブリが作れる！

The screenshot displays the Dynamo software interface with a workflow titled "SectionView.dyn". The workflow is divided into four main stages, each highlighted with a colored background:

- アセンブリを作成 (Create Assembly):** A green box containing a "Name" block, a "Point.ByCoordinates" block, and an "Assembly.ByNamePoint" block. The "Name" block outputs "1. '左端';", which is connected to the "Assembly.ByNamePoint" block. The "Point.ByCoordinates" block takes inputs from an "XY" block (values: 1: -35000, 2: -36000) and outputs "x" and "y", which are connected to the "Assembly.ByNamePoint" block.
- サブアセンブリを追加 (Add Subassembly):** A green box containing an "Assembly.AddSubassembly" block and a "Code Block" block. The "Code Block" block contains the text "1. '中心';", which is connected to the "Assembly.AddSubassembly" block.
- サブアセンブリを読み込み (Load Subassembly):** A grey box containing a "Code Block" block and a "Subassembly.ImportSubassembly" block. The "Code Block" block contains the text "1. '配置_セッパ';", which is connected to the "Subassembly.ImportSubassembly" block.
- Excelを読み込み (Load Excel):** A grey box containing a "File Path" block, a "File From Path" block, a "Data.ImportExcel" block, a "List.Transpose" block, and a "Code Block" block. The "File Path" block outputs "1. 'TubeParam.xlsx'", which is connected to the "File From Path" block. The "File From Path" block outputs "file", which is connected to the "Data.ImportExcel" block. The "Data.ImportExcel" block outputs "data", which is connected to the "List.Transpose" block. The "List.Transpose" block outputs "lists", which is connected to the "Code Block" block. The "Code Block" block contains the text "1. 'Sheet13';", which is connected to the "Data.ImportExcel" block.

The workflow is connected to a "Subassembly.SetParameterByName" block, which is highlighted in pink. This block takes inputs from the "Assembly.AddSubassembly" block and the "Code Block" block in the "Excelを読み込み" stage. The "Subassembly.SetParameterByName" block outputs "subassembly", "parameter", "value", and "restart", which are connected to the "Code Block" block in the "サブアセンブリを読み込み" stage.

The interface includes a menu bar at the top with options like "Dynamo(D)", "ファイル(F)", "編集(E)", "表示(V)", "パッケージ(P)", "ヘルプ(H)", and "拡張機能(X)". A sidebar on the left lists various categories: "ImportExport", "Input", "List", "Math", "Script", "String", "Units", "Add-ons", "Autodesk", "MeshToolkit", and "Springs". The bottom status bar shows "手動" (Manual) and "実行" (Execute) buttons, along with the message "Dynamo を使用できるようになりました。" (Dynamo is now usable).

API Reference の読み方

- API Reference のありか
 - <https://help.autodesk.com/view/CIV3D/2023/ENU/?guid=89ffd413-aada-d770-e322-89dfa7b99369>
 - 年数 (2023) を変えれば、各 Ver の Reference を入手可能
- API Reference の読み方
 - 基本的には Revit API Docs と同じ
 - 使いたい Method (関数) を検索
 - 同じメソッドが複数ある場合は、説明書きを基に絞り込み
 - パラメータの詳しい情報を確認

The screenshot displays the Autodesk Civil 3D 2023 API Reference interface. The top header shows the Autodesk logo and the product name 'AUTODESK Civil 3D 2023'. The main content area is divided into two panes. The left pane shows a tree view of the API structure, with the following items expanded:

- Namespace
 - Autodesk.Civil.DatabaseServices Namespace
 - + AdditionalAppliedAssemblyInfo Class
 - Alignment Class
 - Alignment Members
 - Alignment Methods
 - + CopyToSite Method
 - Create Method

The right pane shows the code for the 'Create' method in three languages: C#, Visual Basic, and Visual C++.

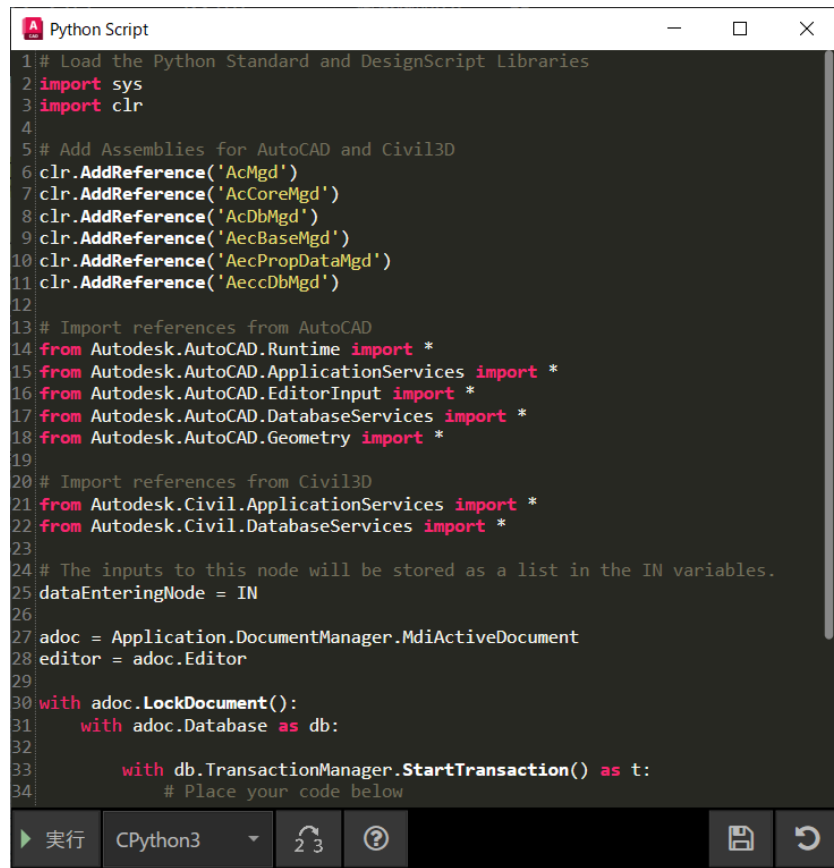
```
C#
public static ObjectId Create(
    CivilDocument document,
    string alignmentName,
    ObjectId siteId,
    ObjectId layerId,
    ObjectId styleId,
    ObjectId labelSetId
)
```

```
Visual Basic
Public Shared Function Create ( _
    document As CivilDocument, _
    alignmentName As String, _
    siteId As ObjectId, _
    layerId As ObjectId, _
    styleId As ObjectId, _
    labelSetId As ObjectId _
) As ObjectId
```

```
Visual C++
public:
static ObjectId Create(
    CivilDocument^ document,
    String^ alignmentName,
    ObjectId siteId,
```


Python Script で API を実装する下準備

- Python Script の Input の設定
 - Revit に同じ
- Python Script ライブラリーの読み込み
 - Python Script を作成した段階で、ボイラープレートを設定済



```
Python Script
1 # Load the Python Standard and DesignScript Libraries
2 import sys
3 import clr
4
5 # Add Assemblies for AutoCAD and Civil3D
6 clr.AddReference('AcMgd')
7 clr.AddReference('AcCoreMgd')
8 clr.AddReference('AcDbMgd')
9 clr.AddReference('AecBaseMgd')
10 clr.AddReference('AecPropDataMgd')
11 clr.AddReference('AeccDbMgd')
12
13 # Import references from AutoCAD
14 from Autodesk.AutoCAD.Runtime import *
15 from Autodesk.AutoCAD.ApplicationServices import *
16 from Autodesk.AutoCAD.EditorInput import *
17 from Autodesk.AutoCAD.DatabaseServices import *
18 from Autodesk.AutoCAD.Geometry import *
19
20 # Import references from Civil3D
21 from Autodesk.Civil.ApplicationServices import *
22 from Autodesk.Civil.DatabaseServices import *
23
24 # The inputs to this node will be stored as a list in the IN variables.
25 dataEnteringNode = IN
26
27 adoc = Application.DocumentManager.MdiActiveDocument
28 editor = adoc.Editor
29
30 with adoc.LockDocument():
31     with adoc.Database as db:
32
33         with db.TransactionManager.StartTransaction() as t:
34             # Place your code below
```

Dynamo タイプから Civil 3D タイプへの変換

- Revit と違い、変換不可
 - 下記の二つによる対応が基本
 - 最初から最後まで Dynamo ノードを使用
 - 最初から最後まで Python Script を使用
- Civil3DToolkit パッケージを使うと、Dynamo ノードだけで行える処理を飛躍的に増やすことが可能
 - パラメータの“参照”だけでなく、“作成”“更新”に対応できるように

デフォルトのノード



Civil3DToolkit 内のノード

