



Tài Liệu Có Tham Khảo Tại Trang WWW.DYNAMOBIM.ORG

Biên soạn: Lê Đức Tuấn
08-2017

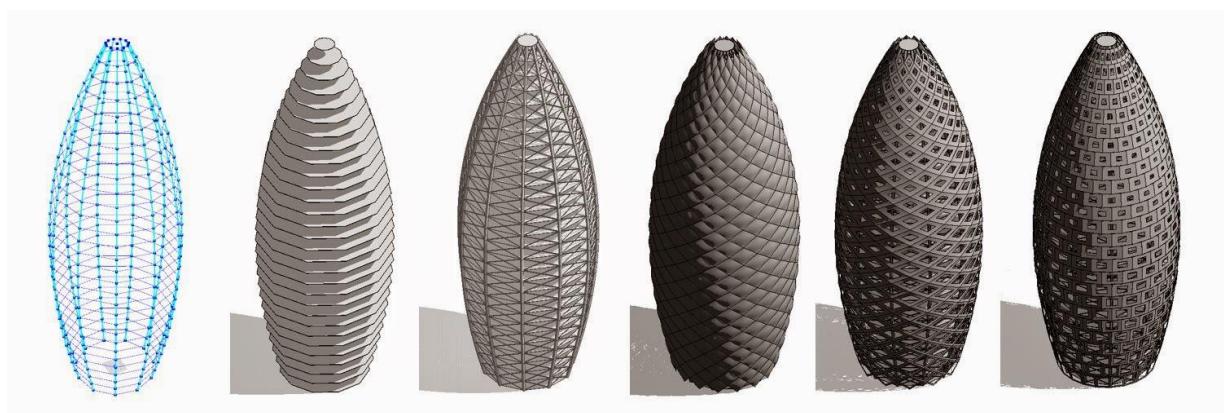
A.TỔNG QUAN.....	4
I. DYNAMO LÀ GÌ:	4
II. CÀI ĐẶT	5
B.CÁC THÀNH PHẦN CƠ BẢN	6
I. NODES	6
II. WIRES- DÂY	7
III. LIBRARY- THƯ VIỆN.....	7
IV. PROGRAM MANAGEMENT- QUẢN LÍ CHƯƠNG TRÌNH	8
V. NODE- THÀNH PHẦN CƠ BẢN XÂY DỰNG NÊN PROGRAM	9
a. Data	9
b. Geometry	9
c. List- Danh sách	10
d. Revit Nodes	10
e. Custom Packages- công cụ hữu hiệu miễn phí.....	11
C.GEOMETRY- ĐỐI TƯỢNG VẬT LÝ.....	11
I. POINT- ĐIỂM.....	11
II. CURVES- ĐƯỜNG.....	13
III. LINE- ĐƯỜNG THẲNG.....	14
IV. SURFACE- BỀ MẶT	16
V. SOLID- KHỐI ĐẶC	17
D.DATA IN DYNAMO- DỮ LIỆU TRONG DYNAMO	18
I. NUMBER	18
II. STRING	20
III. MATH- PHÉP TOÁN	21
IV.LOGIC.....	24
E.LÀM VIỆC VỚI LIST	26
I. Thao Tác List Cơ Bản.....	26
II. Thao Tác List Nâng Cao	31
F.REVIT TO EXCEL	33
I. Import To Excel.....	34
II. Export from Excel	36

G. CODEBLOCK VÀ DESIGNSCRIPT	36
I. Giới thiệu cơ bản về CodeBlock và DesignScript.....	36
II. Cách tạo lập và chức năng của Code Block và DesignScript.....	37
1. Code Block	37
2. DesignScript Syntax	38
3. Lacing trong Code Block.....	39
4. List@Level trong Code Block	43
5. Node To Code	45
6. Logic Trong Code Block.....	46
7. Tạo list và lấy giá trị trong list	46
8. Hàm Trong CodeBlock	48
H. PYTHON IN DYNAMO.....	48
I. Giới thiệu sơ lược về Python.....	48
II. Python trong Dynamo	49

A. TỔNG QUAN

I. DYNAMO LÀ GÌ:

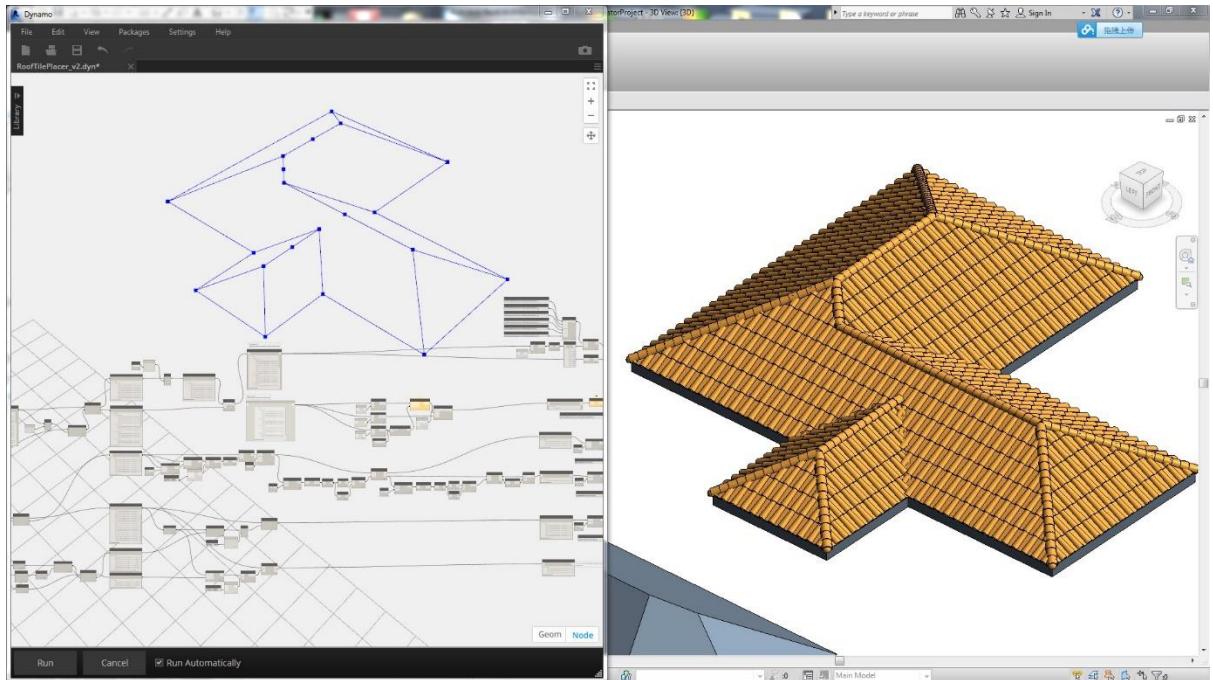
- Plus-in hỗ trợ revit và bắt đầu từ phiên bản 2017 đã được tích hợp vào Revit
- Nền tảng lập trình mang tính thị giác
- Mã nguồn mở hỗ trợ cho revit trong quá trình thiết kế, giúp tự động hóa trong quá trình thiết kế hoặc là giúp điều chỉnh phương án thiết kế một cách nhanh chóng và hiệu quả



Điều chỉnh phương án thiết kế lớp bao phủ

Dynamo Không phải là ý tưởng hay khi:

- Được tạo ra và phân phối tới nhiều người sử dụng mà chưa được training
- Đối với công việc phải thực hiện qua nhiều bước. Chỉ khi chạy một Graph(chuỗi node) một lần duy nhất
- Với các dữ liệu “live”



Graph – Môi trường dynamo

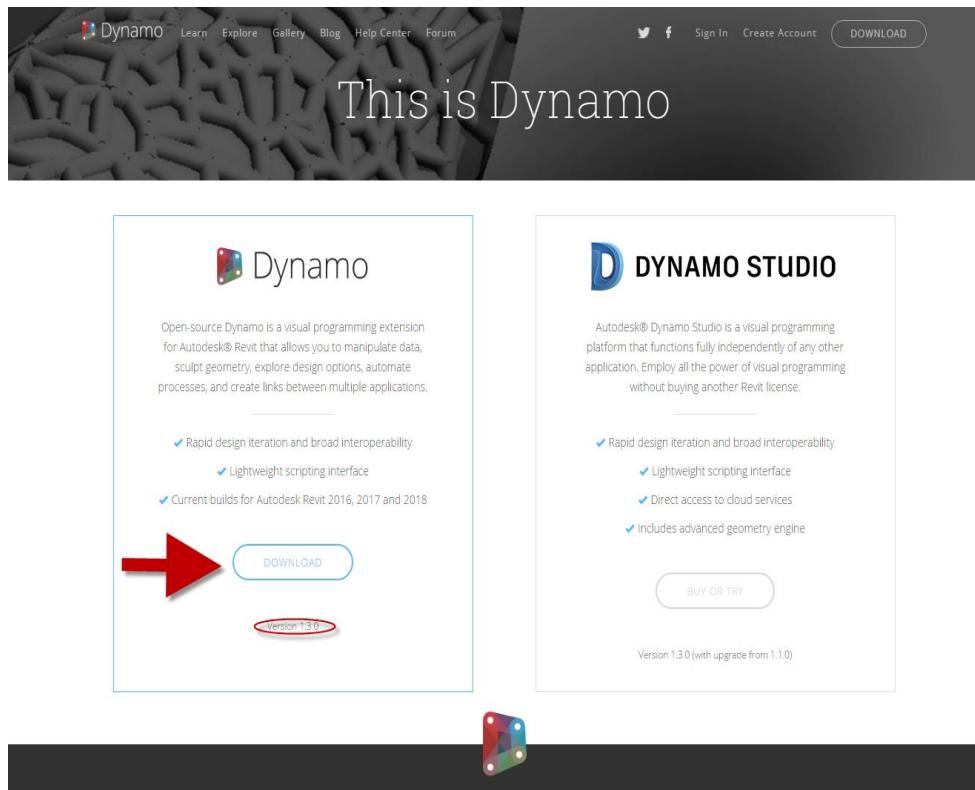
Môi trường Revit

Dynamo thực sự là hiệu quả khi:

- Tự động những thao tác lặp đi lặp lại và mất thời gian. Vd như: Fill thông tin Room từ Excel, Tạo view, tạo sheet.
- Copy thông tin. Vd: Copy và đặt Hanger đều cho ống MEP
- Tạo và sử dụng bởi một người

II. CÀI ĐẶT

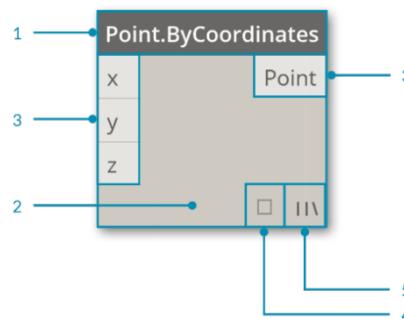
- Nếu Không được cài đặt sẵn như trong phiên bản 2017 thì bạn có thể vào địa chỉ này <http://dynamo.org/download>



B. CÁC THÀNH PHẦN CƠ BẢN

I. NODES

- Nodes là những đối tượng bạn kết nối lại để tạo chương trình ảo. Mỗi node chạy một chức năng, đôi khi đơn giản như là chứa giá trị của một số hay phức tạp hơn là tạo đối tượng hình học hay tính toán dữ liệu
- Hầu hết node trong dynamo được cấu tạo bởi 5 phần:

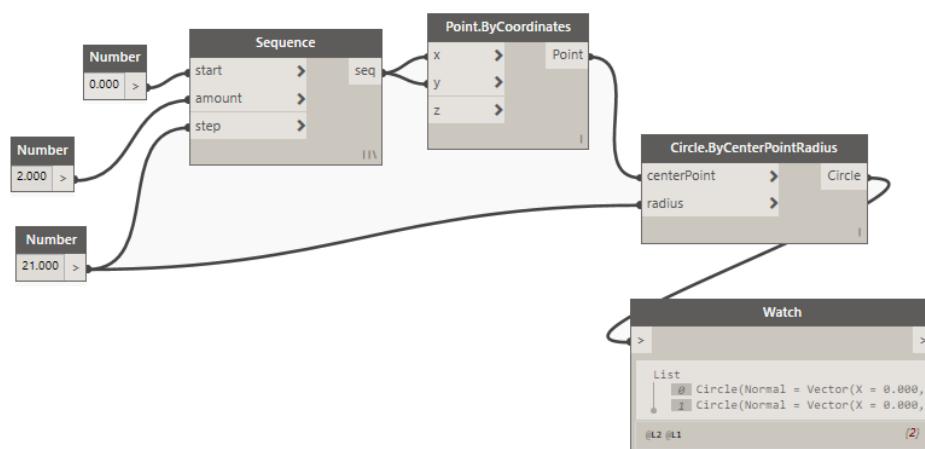


- Name** - tên của node với quy ước <category>.<Name>;
- Main** - Phần “body” chính của node
- Ports-Cổng (In và Out)** – Nơi cho gắn dây dữ liệu vào đối với Cổng In (bên trái Node) cũng như nơi xuất kết quả chạy từ node- Cổng Out(bên phải node);
- Data Preview** – rê chuột vào để xem kết quả chạy được từ node, có thể dùng node Watch để xem;

5. Lacing Icon- thể hiện trạng thái của Lacing- sự kết hợp của dữ liệu đầu vào

II. WIRES- DÂY

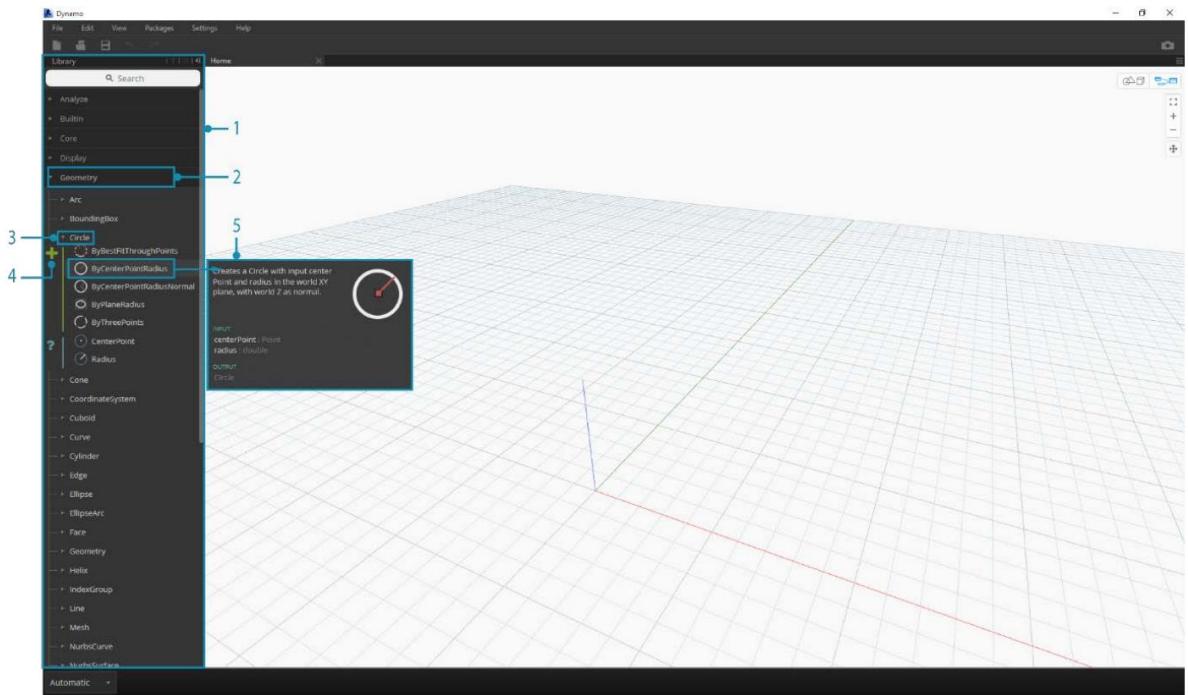
- Wires kết nối các node tạo nên quan hệ liên kết và tạo thành dòng dữ liệu liên tục cho chương trình. Có thể ví nó như dây điện mang dữ liệu từ đối tượng này tới đối tượng khác;
- Wires kết nối Cổng output từ một node tới Cổng InPut của node khác, tạo thành dòng dữ liệu liên tục. Mặc dù có thể sắp xếp vị trí các node nhưng nhìn chung chúng ta có thể thấy dòng dữ liệu chạy từ trái qua phải. Xem ví dụ bên dưới



- Tạo Wire bằng click chọn ở một cổng và click chọn cổng tiếp theo và node sẽ được kết nối. wire sẽ hiện thị từ nét chấm chấm dash sang nét liền solid khi 2 port dc kết nối thành công

III. LIBRARY- THƯ VIỆN

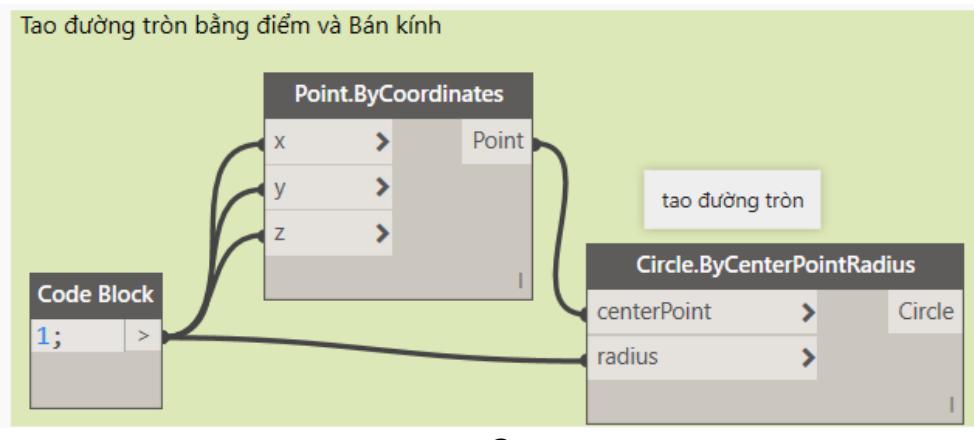
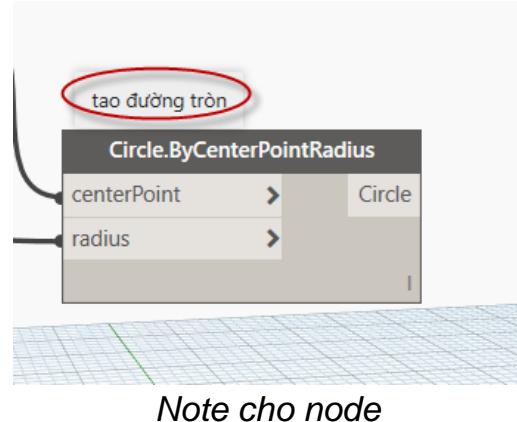
- Thư viện Dynamo là một bộ sưu tập của các hàm hay gói packages, mỗi node đều được nhóm lại theo Category



1. The Library- Giao diện Thư viện người dùng
2. A library-1 thư viện
3. A Category- Bộ sưu tập những node liên quan tới Circles
4. A Subcategory-Category phụ, điển hình là Creat (tạo), Action(Thực hiện), Query(trích xuất);
5. A node- đối tượng được đưa vào WorkSpace để thực hiện lệnh

IV. PROGRAM MANAGEMENT- QUẢN LÍ CHƯƠNG TRÌNH

- Làm việc với chương trình ảo trong Dynamo có thể rất mạnh mẽ và sáng tạo, nhưng mà nhanh chóng chương trình của bạn có thể trở nên rối và phức tạp bởi việc dàn trang thiếu khoa học. Có một vài quy tắc dàn trang đơn giản sau
 1. Alignment- quét chọn các node cần align, chuột phải vào Workspace và chọn align
 2. Notes: Ghi chú cho Nodes : File -> Create Note hoặc Shortcut Ctrl+W
 3. Grouping: Nhóm những node thực hiện ra 1 kết quả hay 1 chức năng. Group có thể được đặt tên và điều chỉnh màu sắc cho nổi



V. NODE- THÀNH PHẦN CƠ BẢN XÂY DỰNG NÊN PROGRAM

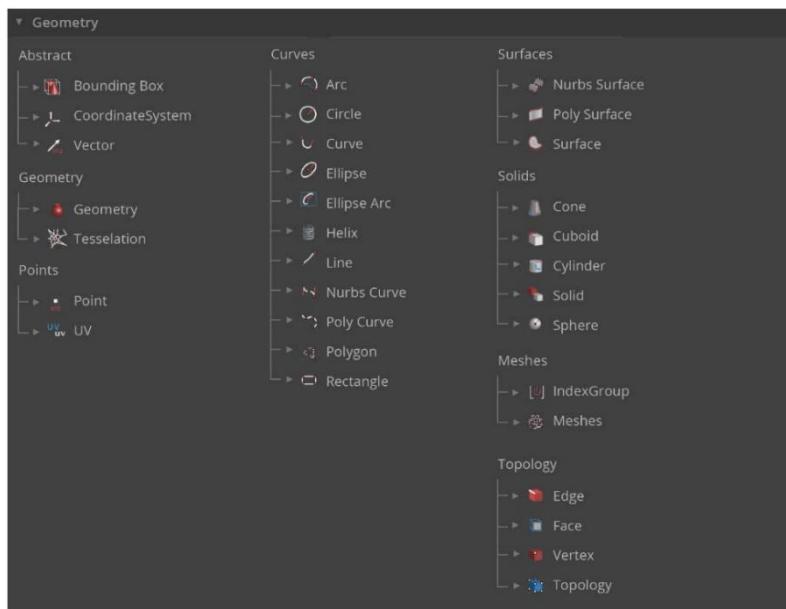
Node là thành phần cơ bản và quan trọng nhất, chính vì thế hiểu được ý nghĩa của node trong việc tạo chương trình trong Dynamo chính là **chìa khóa** để tiết kiệm thời gian và tiền bạc. Node được nhóm lại thành các categories theo các chức năng bao gồm DATA, GEOMETRY, LISTS, và chuỗi các chức năng đặc biệt trong Revit

a. Data

Chúng ta cần dữ liệu để đưa vào Port- có thể có dữ liệu mà ko cần chạy nhưng mà cần phải có dữ liệu thì mới có thể chạy được Node hiện diện. Dạng dữ liệu đơn giản nhất là **number** như 0, 3.14, hay 11. Tuy nhiên có rất nhiều loại dữ liệu: biến thay đổi giá trị number, kí tự hay một tên; dữ liệu hình học; hay một List đối tượng (1,2,3,5,6,12...). Node dữ liệu bao gồm các chức năng liên quan tới Toán tử, Logic(if this then), String (text) hay cả là màu sắc

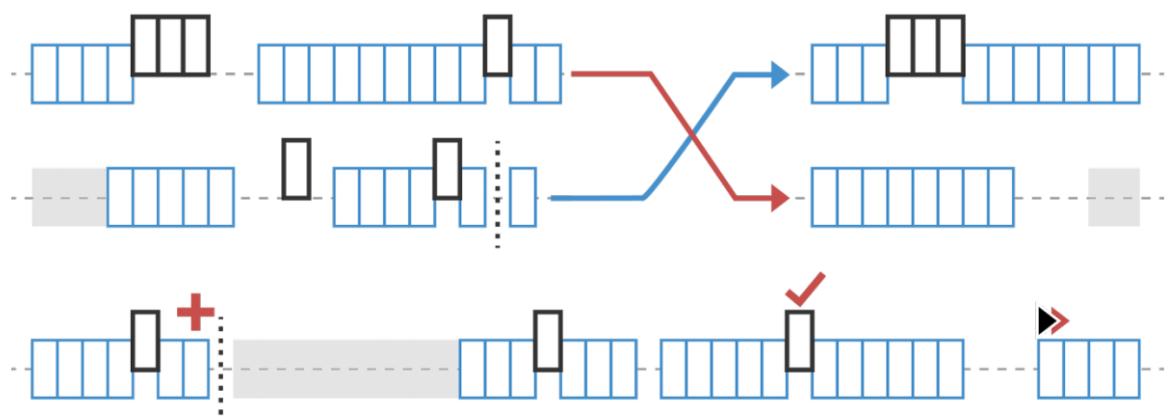
b. Geometry

Dynamo có rất nhiều node dùng để tạo và hiệu chỉnh Geometry, đặc biệt là đối tượng hình học trong revit. Geometry Node bao gồm Curves, Surfaces, Solids, Meshes, Topology, Points, Import Geometry và các chức năng trích xuất.



c. List- Danh sách

- List là dữ liệu được tổ chức trong Dynamo, hiểu và trích xuất cũng như xử lý dữ liệu trong List chính là chìa khóa tới thành công trong việc tạo chương trình trong Dynamo. Trong Dynamo bạn có thể Create, Modify và Trích xuất dữ liệu trong List



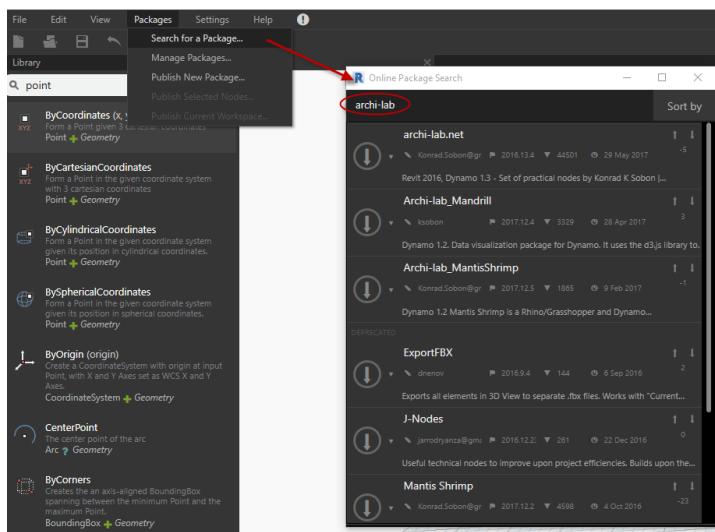
Dynamo List Node: nguồn Dynamoprimer.com

d. Revit Nodes

- *Dynamo cung cấp những node được tạo ra cho riêng Revit. Sử dụng những node này cho phép bạn mở rộng khả năng của Revit dường như không giới hạn. Revit Node bao gồm Selecting, Editing, Creating, Analysis, And Documenting function.*

e. Custom Packages- công cụ hữu hiệu miễn phí

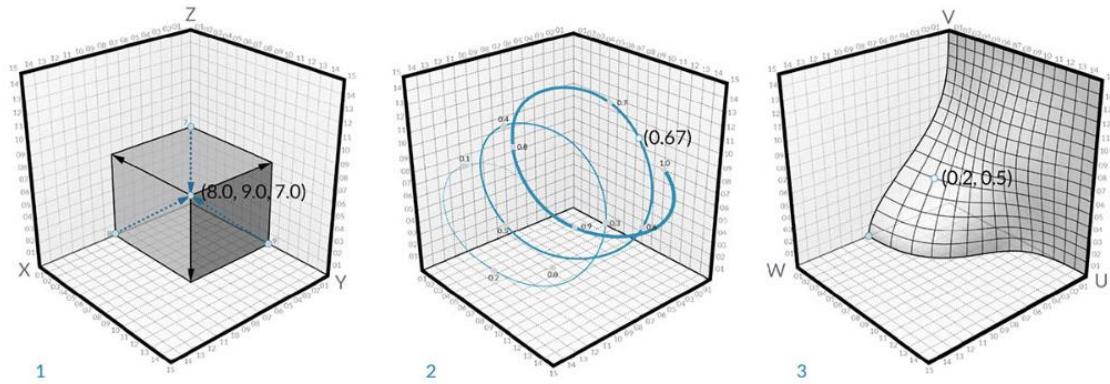
- Một trong những điều tạo nên khả năng phát triển của Dynamo chính là các Packets, Packet là gói các node mà các nhà phát triển tạo ra bằng code Python hoặc là nhóm node có sẵn trong Dynamo để thực hiện một chức năng không có sẵn trong Dynamo, bạn chỉ cần down về bản tương thích với phiên bản Dynamo hiện hành là có thể sử dụng các Packet này rồi
- Các Packet nên phải có trong Dynamo gồm
 - + Archi-lab
 - + Clockwork for Dynamo: hơn 360 node liên quan tới Revit và quản lý List, toán tử với hình học, tấm panels v.v..
 - + Lunchbox for Dynamo:
 - + Rhythm for Dynamo
- Cách download packet:



C. GEOMETRY- ĐỐI TƯỢNG VẬT LÝ

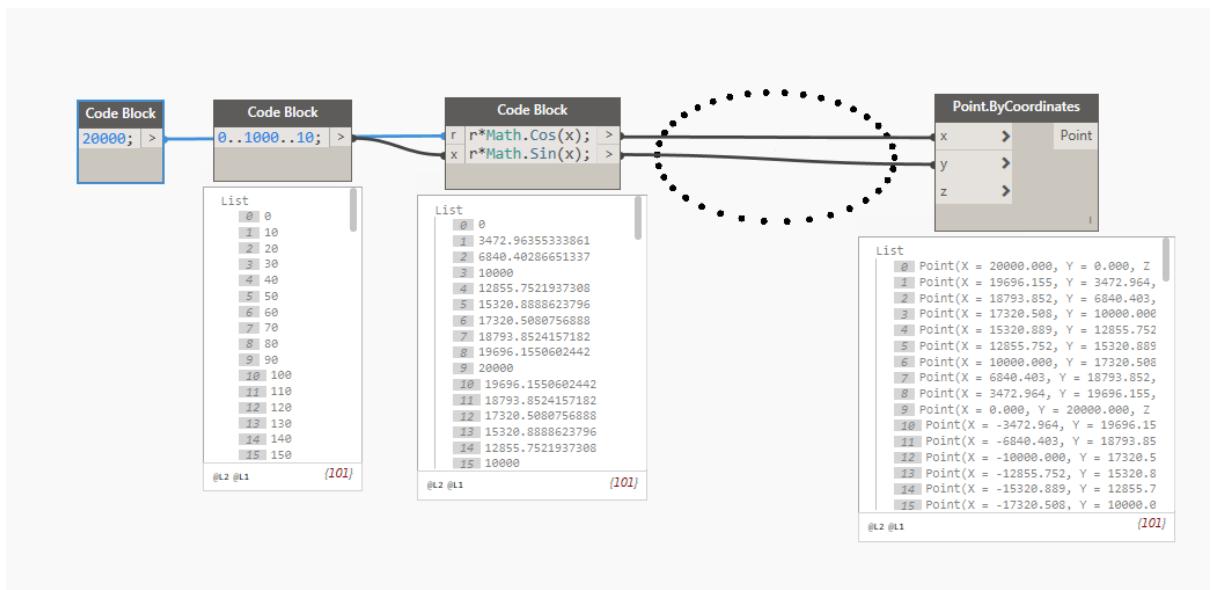
I. POINT- ĐIỂM

- Point là nền tảng của mọi đối tượng vật lý, cần ít nhất 2 điểm để tạo Curve, ít nhất 3 điểm để tạo đa giác hay mặt
- Quy ước về kí hiệu tọa độ của điểm phụ thuộc vào loại không gian làm việc

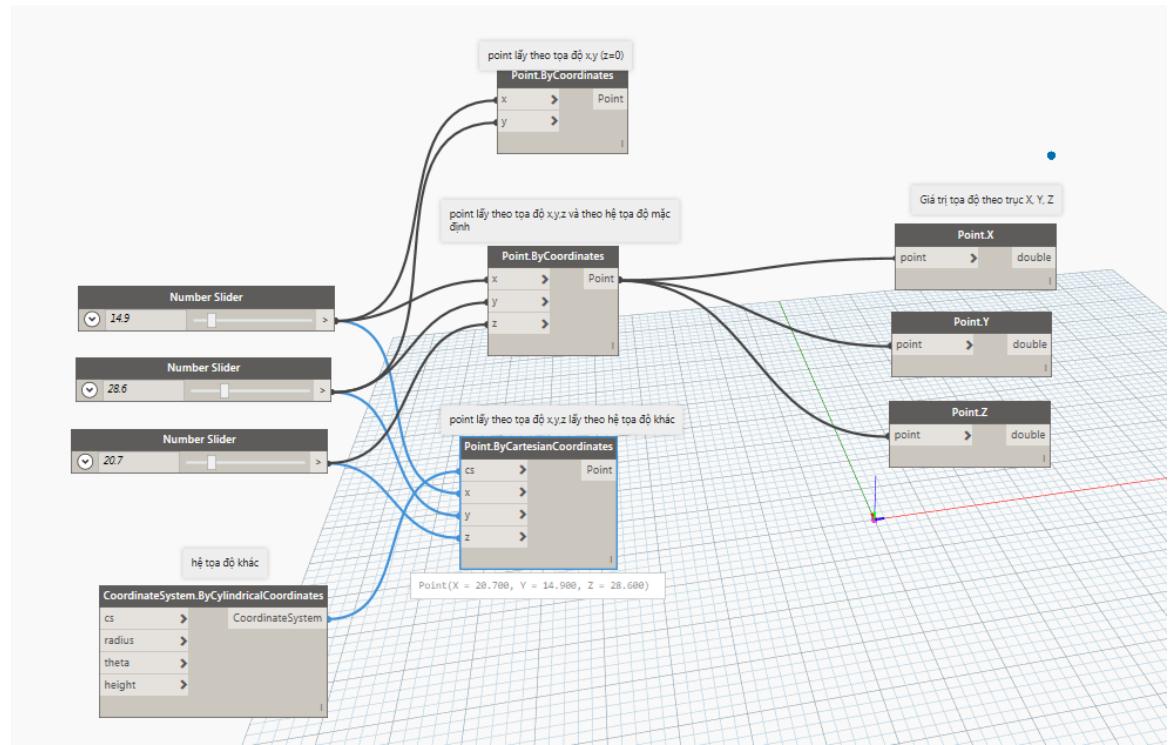


1. Point trong hệ tọa độ O-Clít [X,Y,Z]
2. Point trong hệ tọa độ tham biến của Curve [t]
3. Point trong hệ tọa độ tham biến Surface [U,V]

VD:

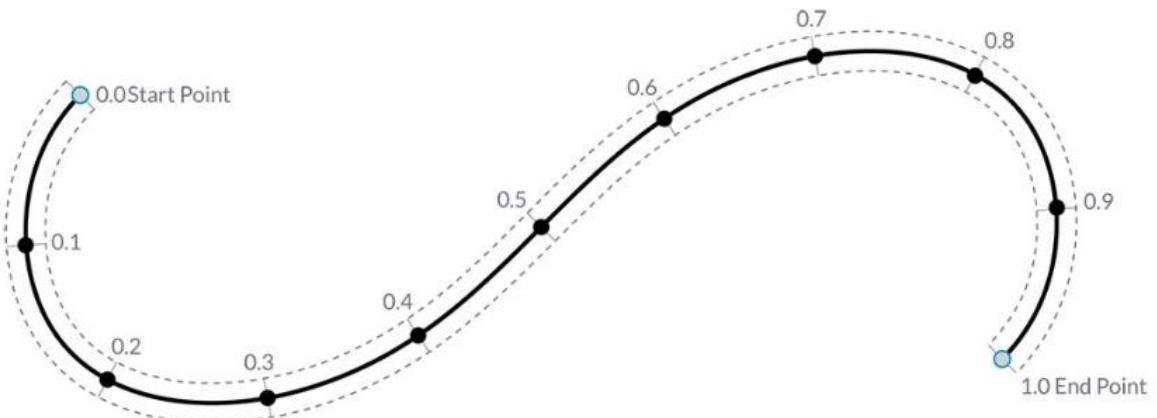


Đường tròn tạo bởi hàm $x=r\cos(x)$ và $y=r\sin(x)$



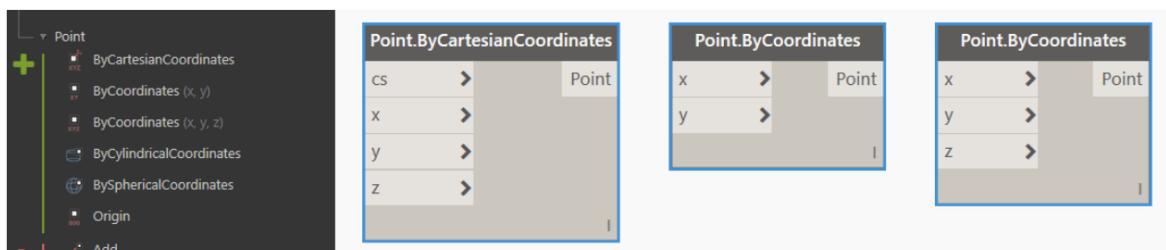
II. CURVES- ĐƯỜNG

- Curve là đối tượng hình học thân quen nhất trong các dạng hình học, curve được tạo bởi đối tượng cơ bản point
- Curve là tên chung cho mọi loại đường, kể cả đường thẳng, bao gồm: Line, Circle, Spline. Curve được cấu tạo từ tập hợp Point và để thể hiện Curve qua Point ta còn có thể dùng hàm (VD: $x=0.5*t$, $y=t$). Tham số t này luôn có thể tính toán được đối với bất kì loại Curve nào
- Nếu nhìn một cách khác thì Curve luôn có điểm đầu và điểm cuối, có thể xem là điểm có giá trị t max và t min tạo ra Curve.

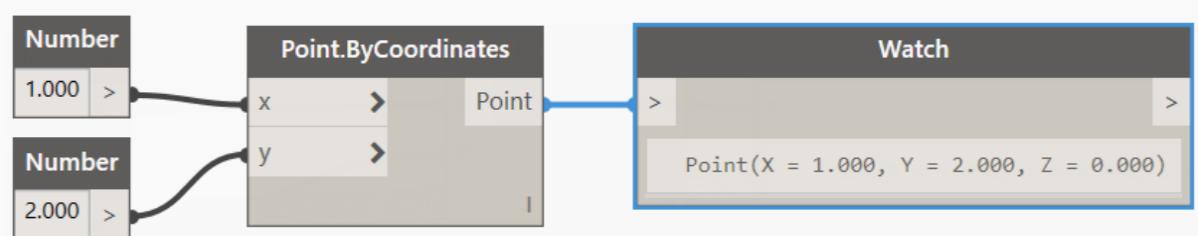


- Trong Dynamo tham số t của Curve được hiểu là 0.0 đến 1.0

- Điểm đầu và điểm cuối mà bằng nhau thì có nghĩa là curve này là đường kín (Closed)
 - Vẽ Point trong Dynamo
- Có nhiều cách tạo Point, vào Geometry > Point

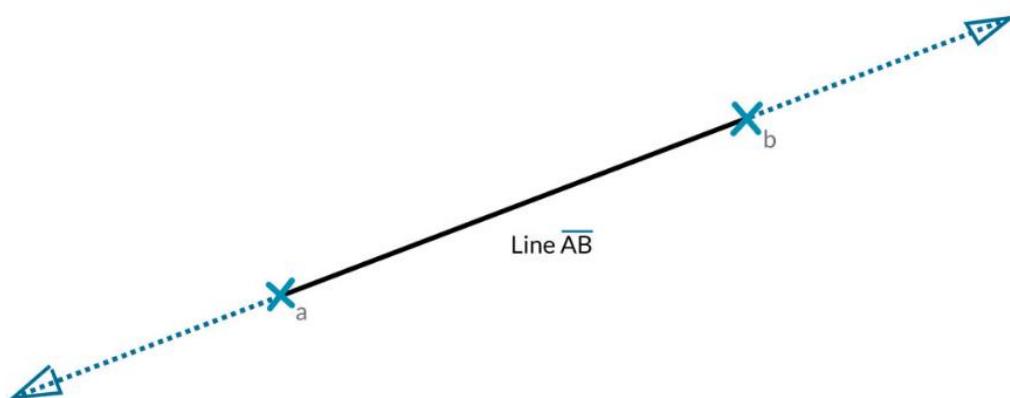
*Các cách tạo Point*

- Khác nhau ở đây là dữ liệu đầu vào, tùy vào cách chúng ta mong muốn cho điều kiện tạo ra điểm
- Rê chuột vào Input ta sẽ thấy phần mô tả dữ liệu cần phải đưa vào và giá trị mặc định (nếu có). Xét node đơn giản nhất là **Point.ByCoordinates**, giá trị mặc định nếu ta ko đưa dữ liệu đầu vào là 0,0

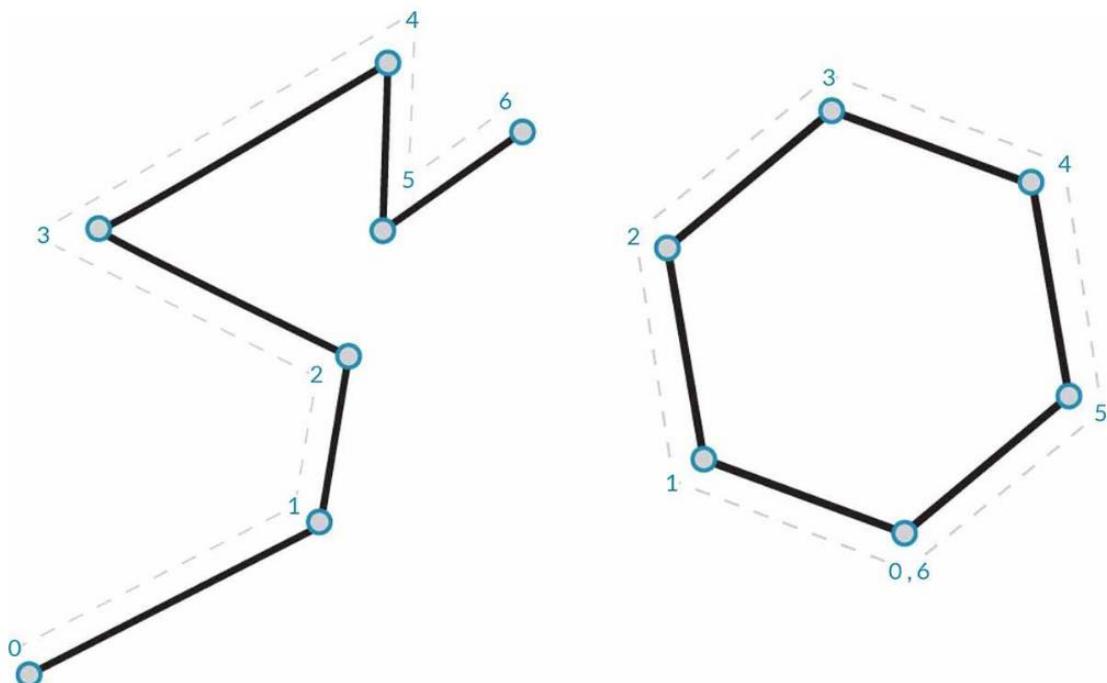
*Point được khởi tạo theo giá trị đầu vào*

III. LINE- ĐƯỜNG THẲNG

- Line là dạng đơn giản nhất của Curve, nó không phải đường cong nhưng cơ bản nó vẫn là Curve. Có nhiều cách để tạo Line nhưng đơn giản nhất là Point A tới Point B, hình dạng của Line AB được vẽ nằm giữa 2 điểm A, B. Nhưng trong toán học thì nó còn có thể được định nghĩa thêm là phần mở rộng về 2 hướng



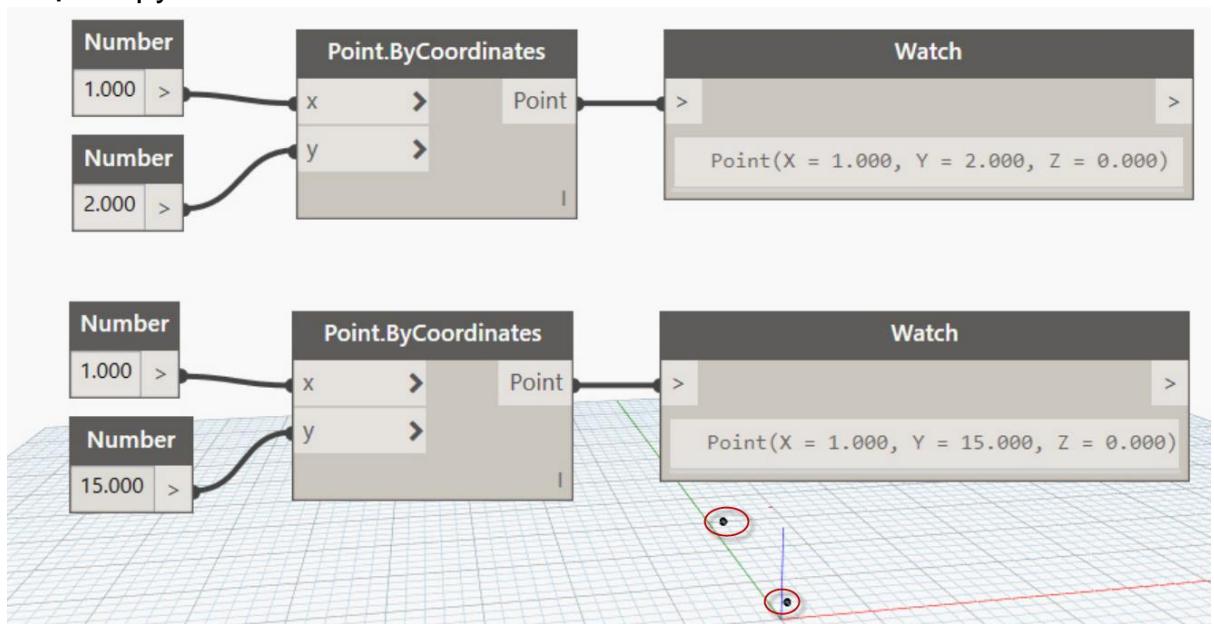
- Kết nối 2 Line lại ta được PolyLine, nếu Polyline kín thì được gọi là PolyGon



Polyline & Polygon

- **Tạo Line trong Dynamo**

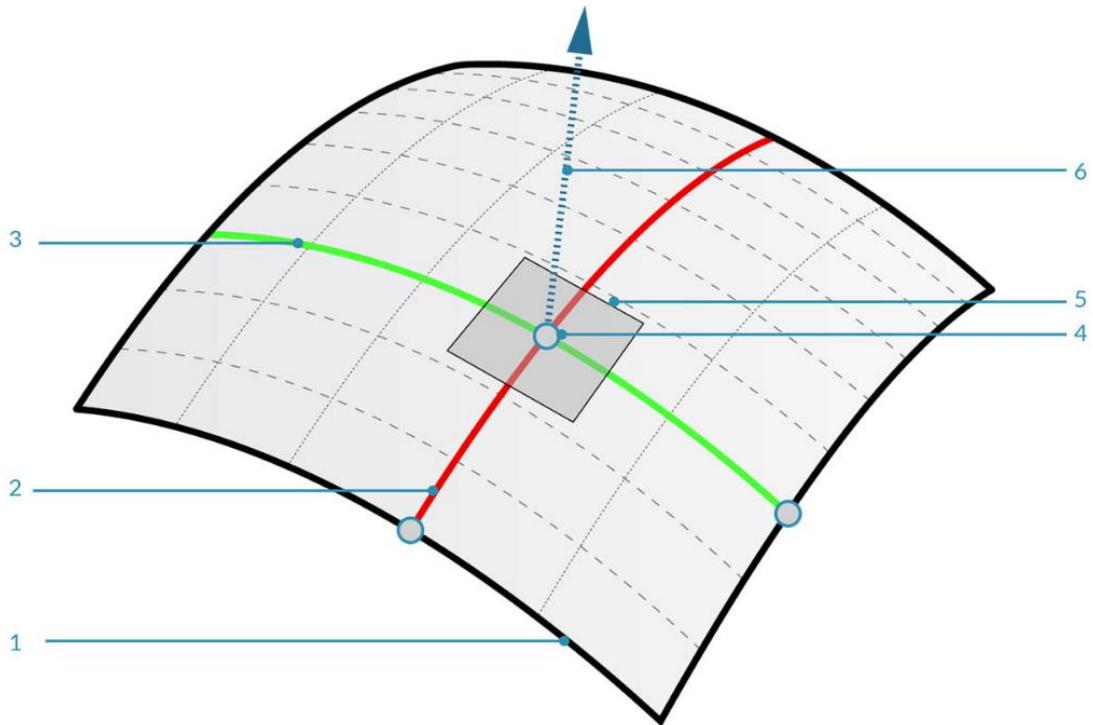
- Để sử dụng lại Graph ta có thể quét chọn copy và paste và trong khi những node vừa được tạo ra vẫn đang được Selected ta có thể dễ dàng kéo tới vị trí mong muốn. Thay đổi giá trị đầu vào của node mới được copy



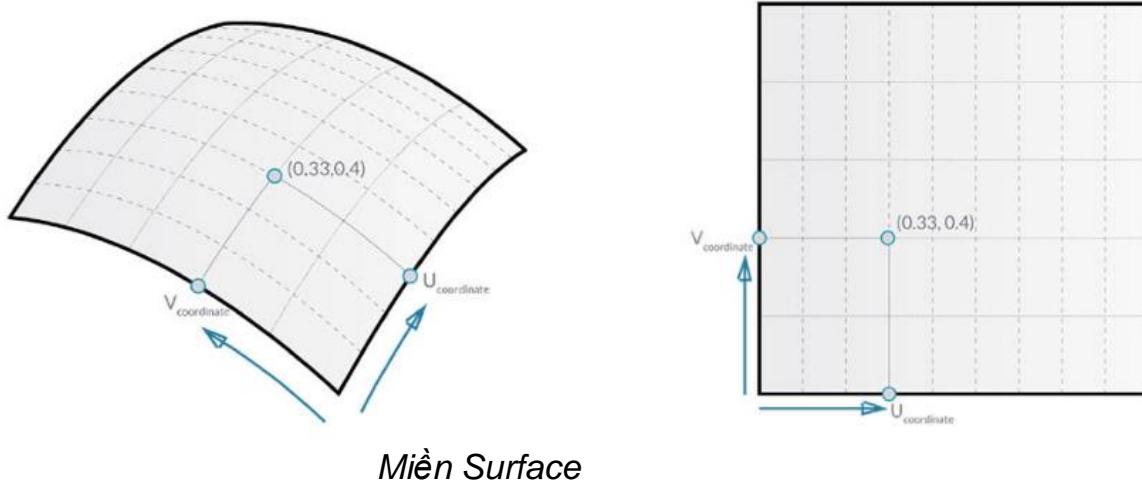
- Tiếp theo vào Library, Geometry > Line bạn sẽ thấy node **ByStartPointEndPoint**, kéo vào WorkSpace và nối lại ta được Line theo 2 điểm.

IV. SURFACE- BỀ MẶT

- Surface là hình dạng toán học được định nghĩa bởi hàm và 2 tham biến u và v , thay cho 1 tham biến t như của Curves
- Line thì có vector tiếp tuyến và mặt phẳng pháp tuyến còn Surface thì có vector pháp tuyến và mặt phẳng tiếp tuyến

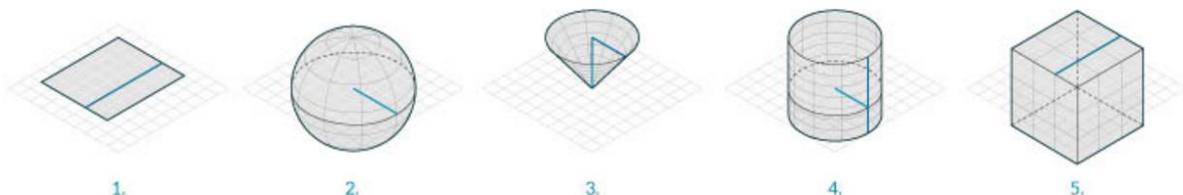


1. Surface
 2. Curve phương U
 3. Curve phương V
 4. Hệ trục tọa độ UV
 5. Mặt phẳng vuông góc
 6. Vector pháp tuyến
- Miền Surface được định nghĩa là dãy tham biến (U,V) mà mỗi cặp (U,V) định lượng cho 3 điểm theo 3 chiều trên Surface. Miền của mỗi chiều (U hoặc V) được mô tả bởi 2 số (U min tới U max) và (V Min tới V Max)
 -

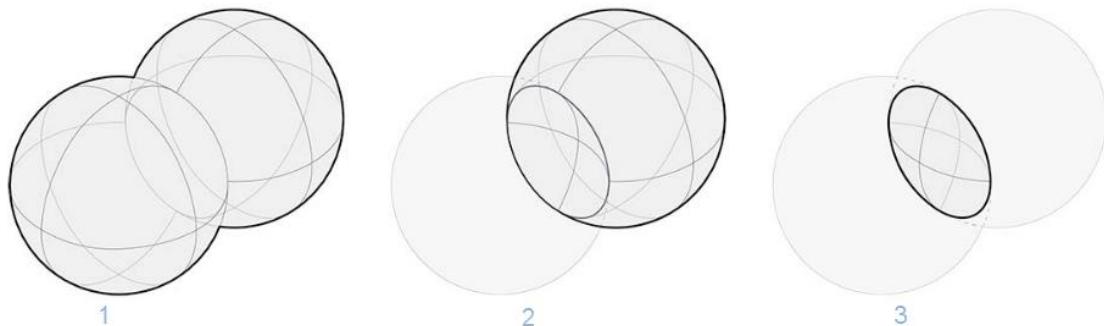


V. SOLID- KHỐI ĐẶC

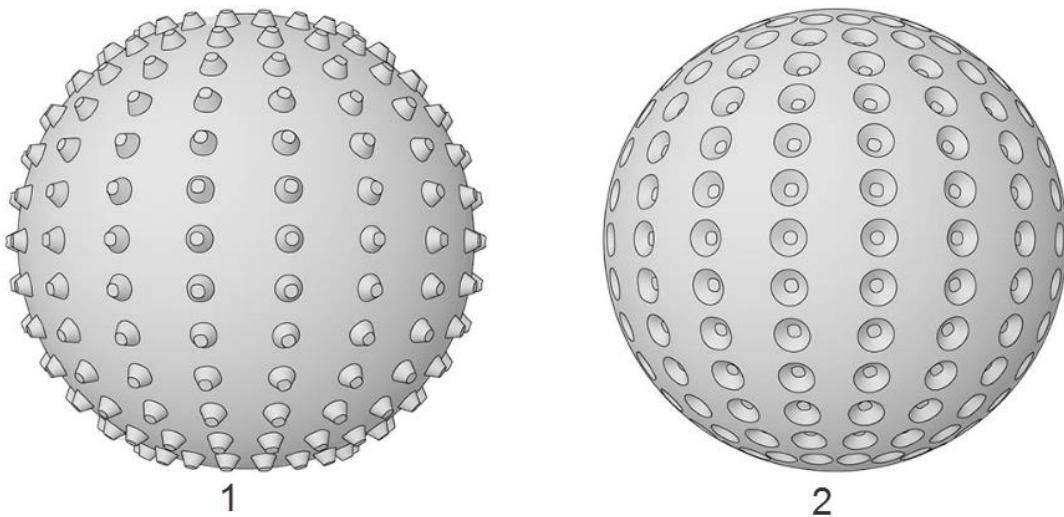
- Solids co thể coi một hay nhiều surface kín và có thể tích, có nghĩa là không cần biết có bao nhiêu surface hay polysurface liên kết lại miễn là hình dạng của nó phải “chứa được nước” thì mới được xem là solid
- Solid có thể tạo bởi các phương thức Loft, Sweep và Revolve. Sphere, Cube, Cone và Cylinder cơ bản là solids



1. Mặt phẳng tạo bởi Surface, không phải Solid
 2. Mặt cầu tạo bởi Surface, là Solid
 3. Hình Cone tạo bởi 2 Surface join với nhau, là Solid
 4. Hình trụ tạo bởi 3 Surface join với nhau, là Solid
 5. Hình hộp tạo bởi 6 Surface join với nhau, là Solid
- **Phép Toán Logic** : là phương thức kết hợp 2 hay nhiều solid lại, một phép toán logic với đối tượng hình học gồm 4 phương thức sau:
 1. Intersect : Giao đối tượng
 2. Split : Tách đối tượng giao nhau
 3. Delete : Xóa đối tượng không mong muốn
 4. Join : join mọi thứ lại với nhau
 - Có 4 phép toán Logic với Solid :



1. Union : Loại bỏ phần chồng lên nhau và join đối tượng thành 1 Solid
 2. Difference : Trừ đi phần solid từ một solid khác
 3. Intersection: Giữ lại phần khối tích giao nhau giữa 2 Solids
- Ngoài ra đối với nhiều Solids với nhau thì ta có nút **Solid.DifferenceAll** và **Solid.UnionAll**



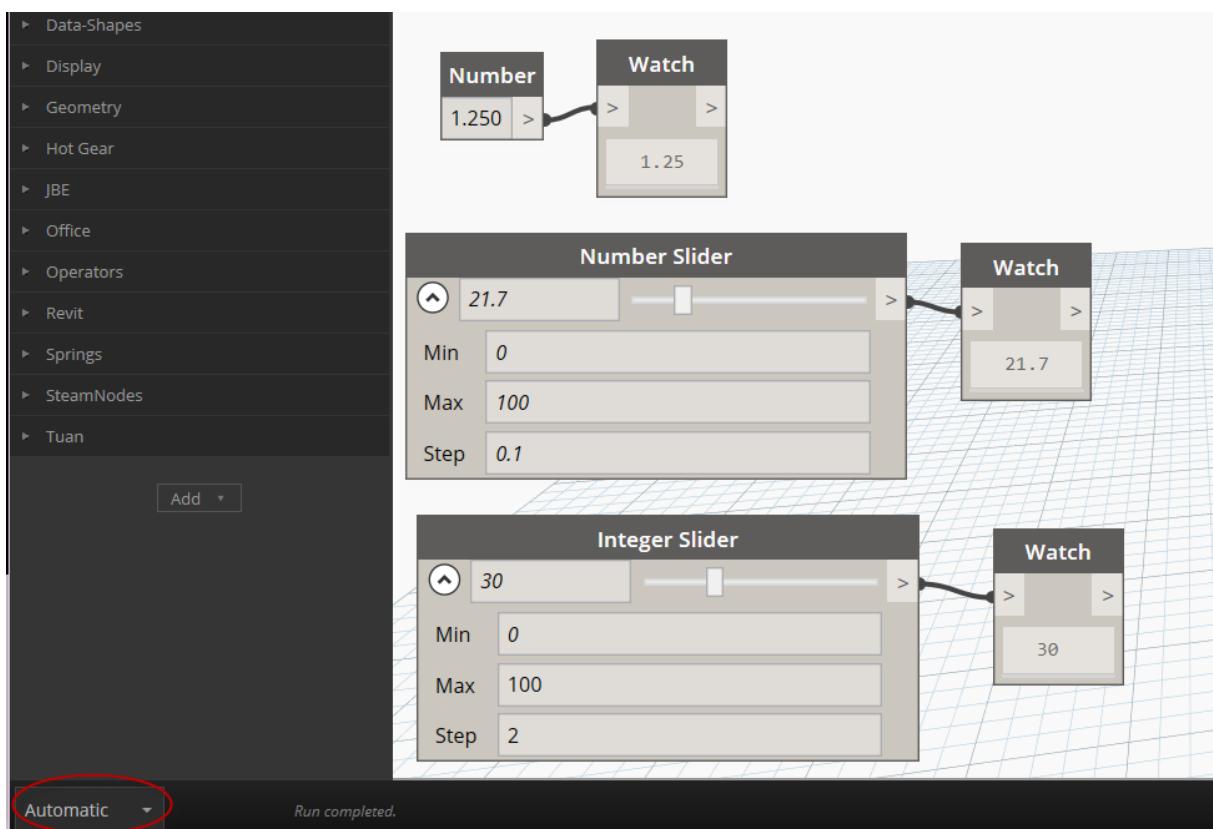
1. UnionAll: Lấy tất cả phần của các khối Solid cầu và các khối Solid cone giao nhau (bao gồm phần giao nhau)
2. DifferenceAll:Lấy phần khác nhau của khối Solid cầu so với các khối Solid cone

D. DATA IN DYNAMO- DỮ LIỆU TRONG DYNAMO

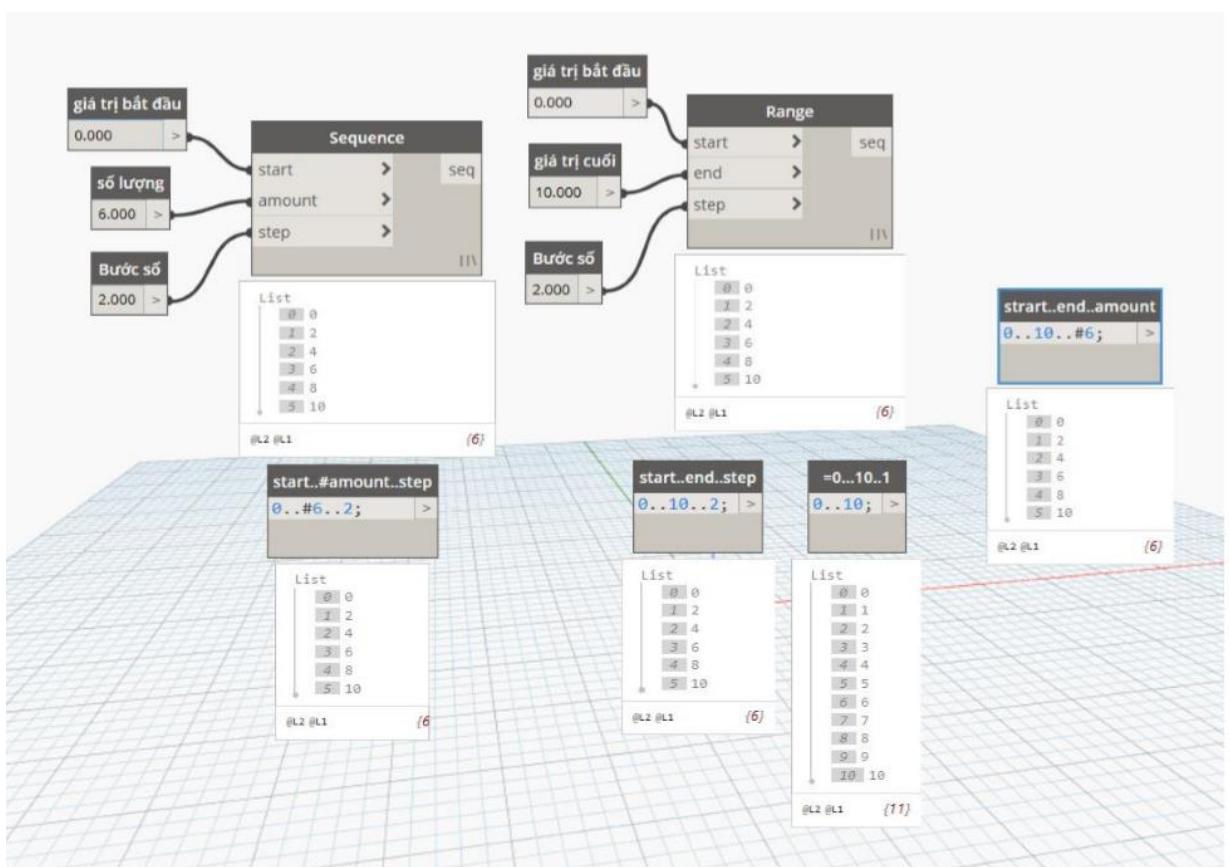
- Dữ liệu bao gồm number (số), Character (ký tự), geometry (hình học), hay là một list các thứ ở trên

I. NUMBER

- Tìm Number trong Library ta được 3 kiểu input là Number và Number Slider và Integer Slider

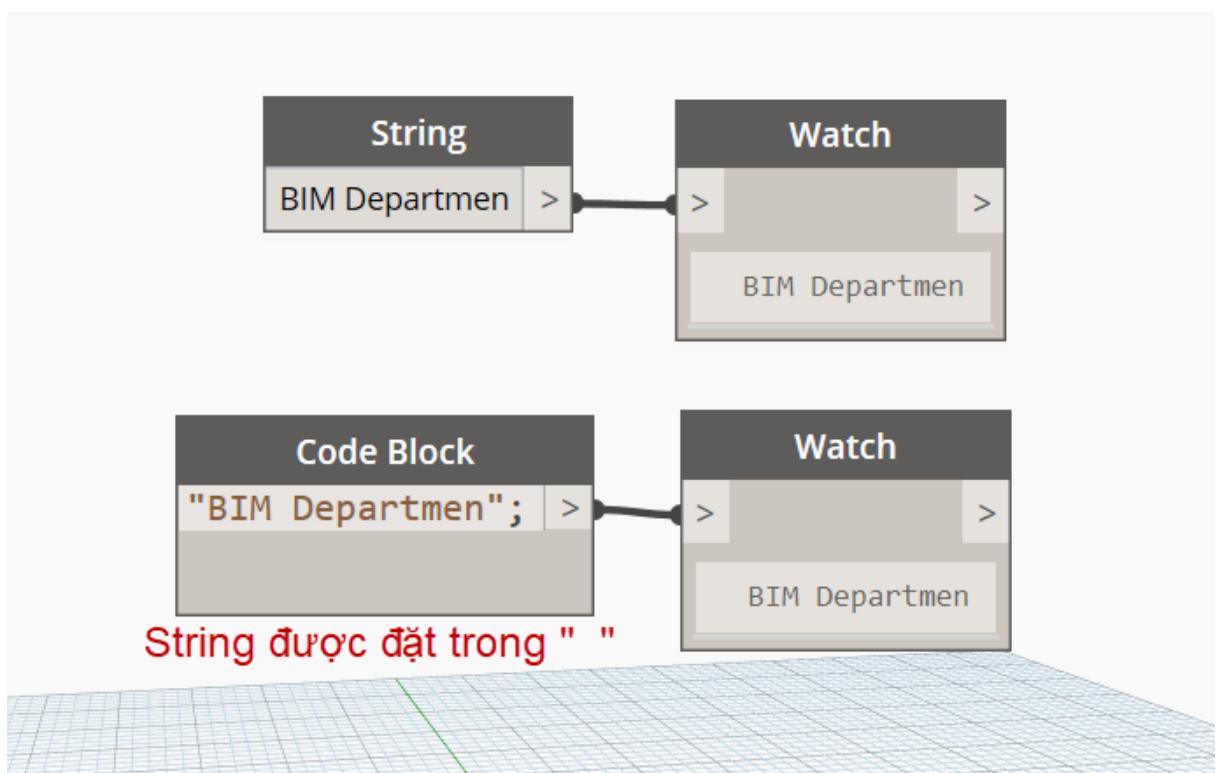


- + Node Number: cho phép điền trực tiếp giá trị vào
- + Node Number Slider: cho phép kéo theo thanh trượt với giá trị Max, Min, bước nhảy Step set trước. Với chế độ Run Automactic ta sẽ thấy được giá trị input thay đổi khi được kéo trên thanh trượt
- + Node Integer Slider: tương tự Number Slider nhưng với giá trị là số nguyên
- Phía trên là ví dụ về giá trị đơn, giá trị mảng(hay list) của number sẽ được tạo như sau:
 - + Node Range
 - + Node Sequence
 - + CodeBlock

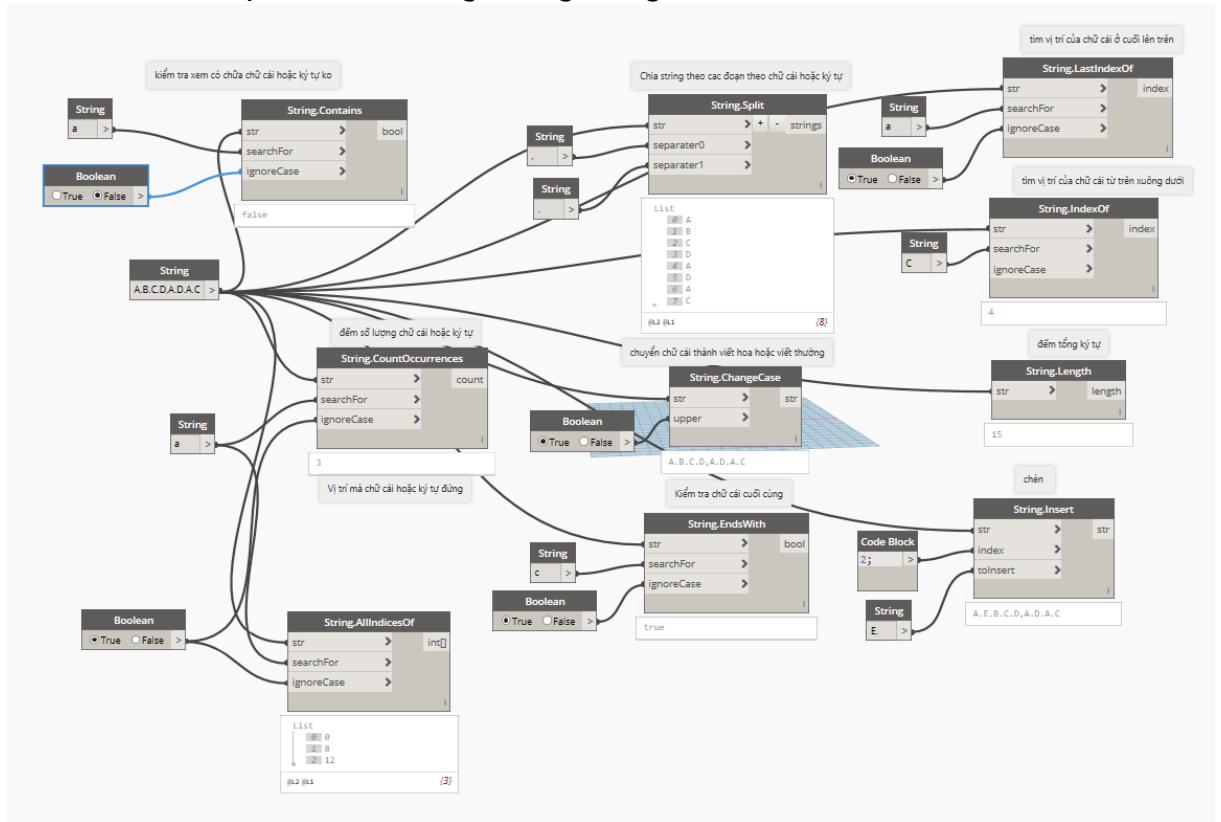


II. STRING

- Dữ liệu kiểu String là dữ liệu dạng ký tự, chuỗi ký tự và được input bởi node String hay Code Block



- Các Node liên quan đến String thông dụng



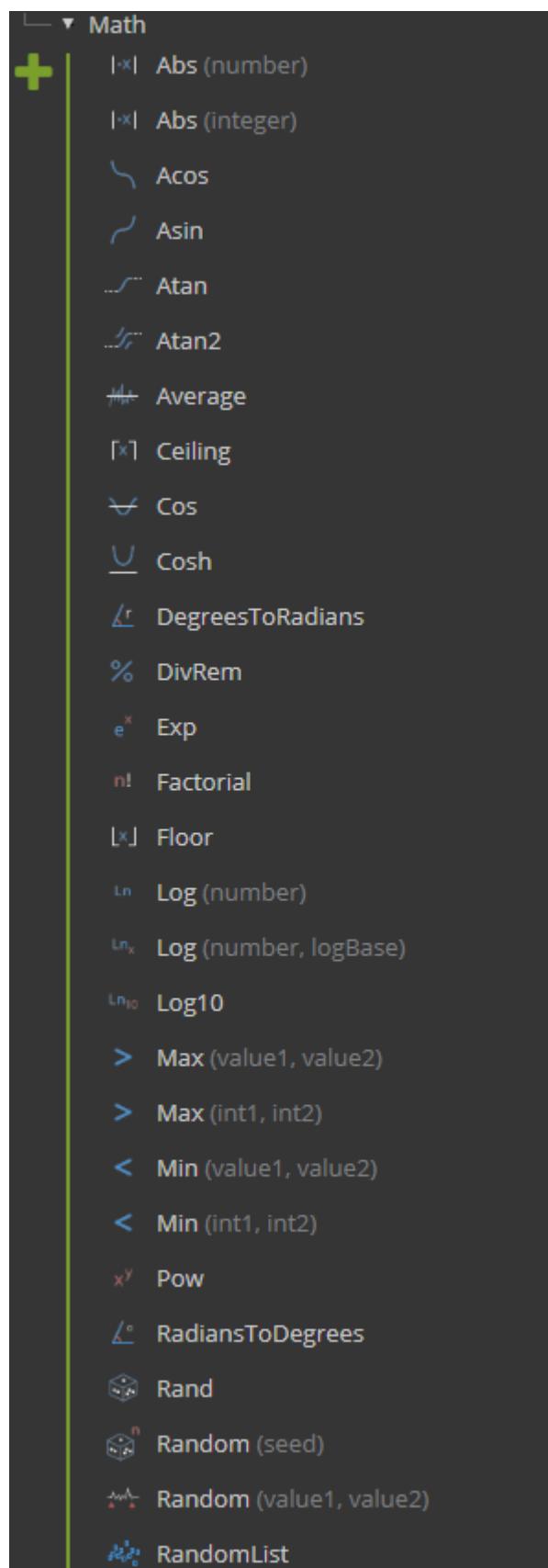
III. MATH- PHÉP TOÁN

- Phép toán số học cơ bản có thể tìm trong Operators>Actions hoặc gõ đúng tên hay kí hiệu phép toán

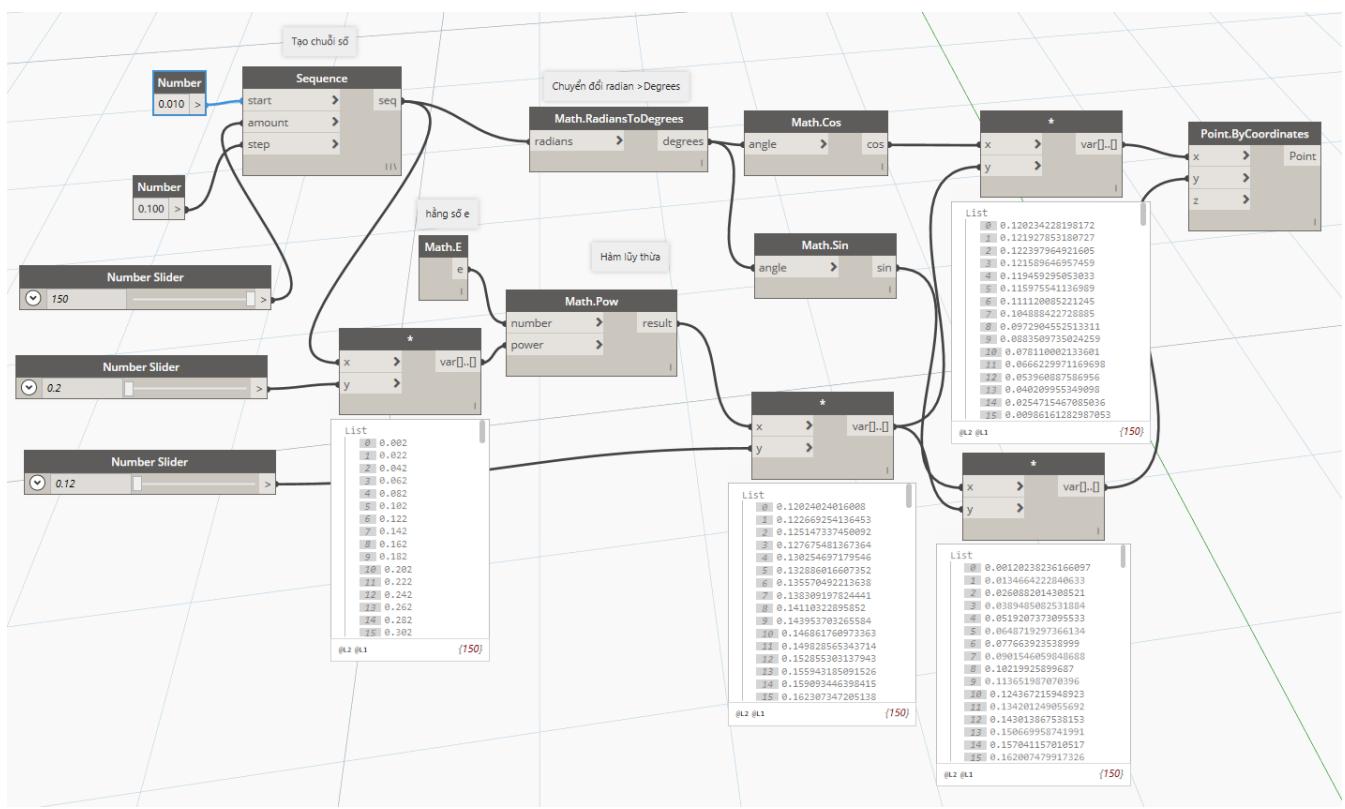
Icon	Name	Syntax	Inputs	Outputs
	Add	+	var[]...[], var[]...[]	var[]...[]
	Subtract	-	var[]...[], var[]...[]	var[]...[]
	Multiply	*	var[]...[], var[]...[]	var[]...[]
	Divide	/	var[]...[], var[]...[]	var[]...[]

Toán tử cơ bản

- Ngoài ra các hàm trong có thể tìm thấy trong Category Math



Ví dụ: Vẽ đường xoắn ốc Fibonaci



IV. LOGIC

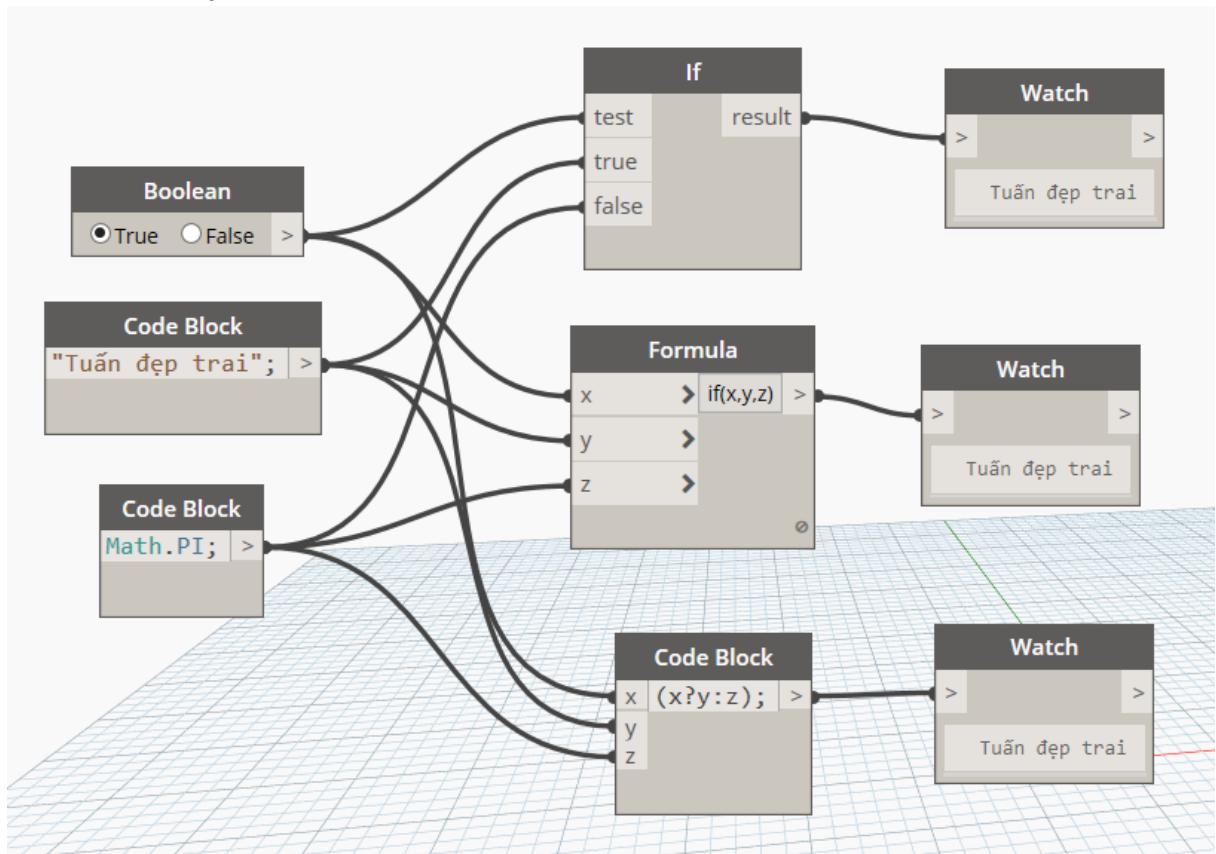
1. Booleans- Luận Lý

- Biến Luận Lý chỉ nhận 2 giá trị là **True/False** tương ứng Đúng/Sai hay 1/0 trong lập trình hay đơn giản là Có/Không
- Biến luận được sử dụng khi phép toán cần rẽ nhánh với cú pháp **If**
- Câu lệnh If có thể hiểu đơn giản là “ Nếu điều này đúng thì hãy thực hiện A còn không thì hãy làm B”

Icon	Name	Syntax	Inputs	Outputs
	If	If	test, true, false	result
	Formula	IF(x,y,z)	x, y, z	result
	Code Block	(x?y:z)	x, y, z	result

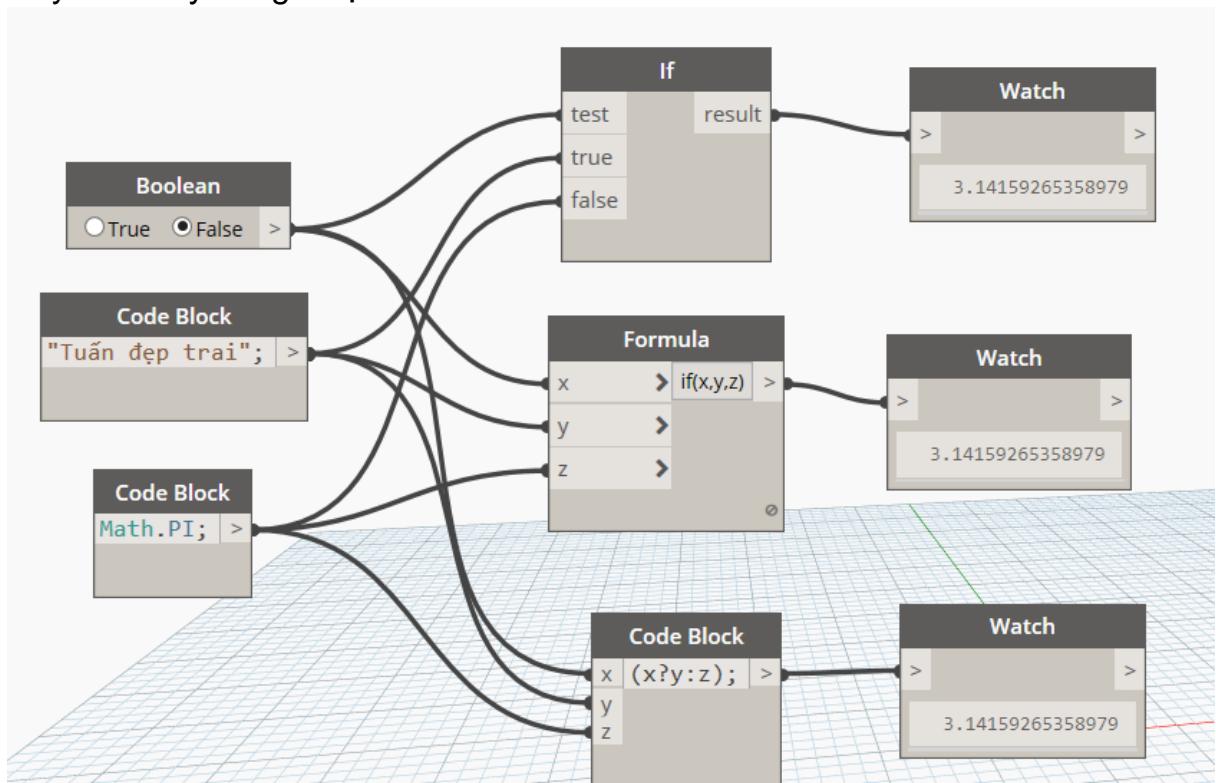
Cách thể hiện câu lệnh IF trong Dynamo

- Xem ví dụ sau



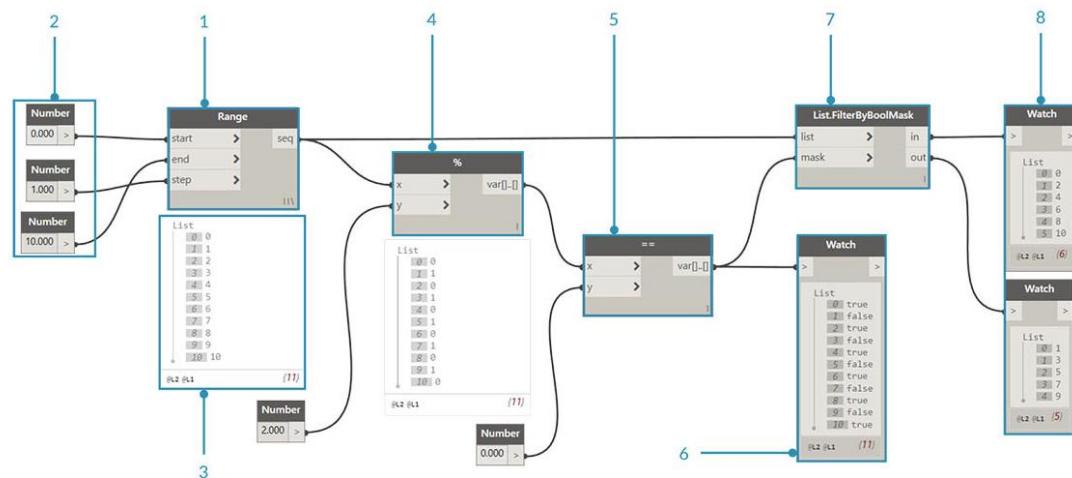
Giá trị Y xuất ra khi Điều kiện nhận TRUE

- Hãy thử thay đổi giá trị Boolean:



Giá trị Z xuất ra khi điều kiện FALSE

- Xét tiếp ví dụ phân loại số chẵn lẻ sau

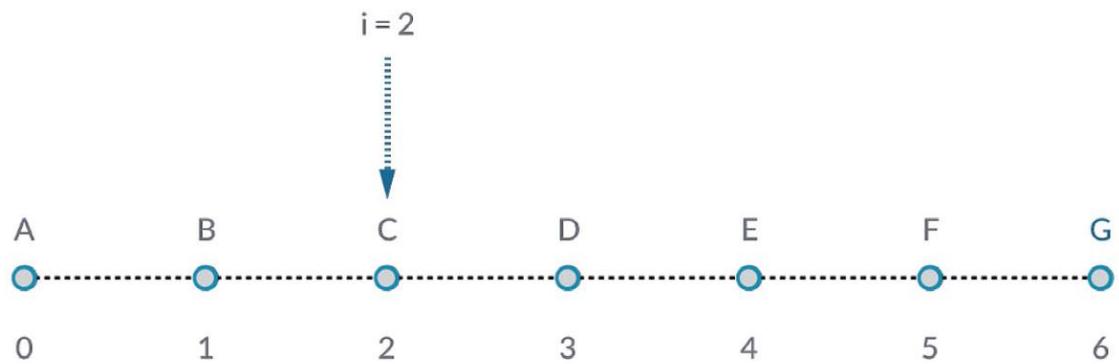


1. Number Range: tạo một dãy số
2. Number: Giá trị bắt đầu là 0, Bước nhảy là 1, giá trị cuối là 1
3. Output: List số giá trị được tạo ra
4. Phép chia lấy dư, ở đây chia lấy dư cho 2
5. Kiểm tra True/False, ở đây so sánh lần lượt giá trị dư của list số với số 0, nếu bằng thì trả về True, không bằng trả về False
6. Watch: Kiểm tra giá trị trả về của Node, ở đây chính là list true/false
7. List.FilterByBoolMask: Node này sẽ lọc ra một list thành 2 list dựa vào list true/false đầu vào, tương ứng giá trị nhận true sẽ vào list IN, giá trị tương ứng False sẽ vào OUT
8. Watch: Xem kết quả 2 list được phân loại, chính là list số chẵn và list số lẻ

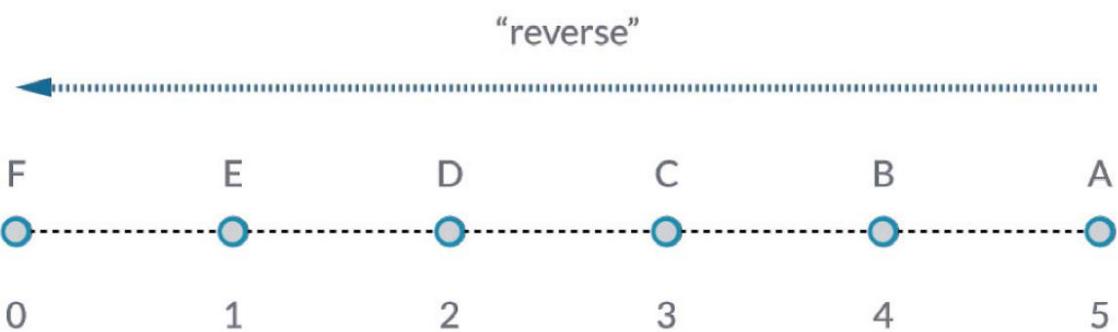
E. LÀM VIỆC VỚI LIST

I. Thao Tác List Cơ Bản

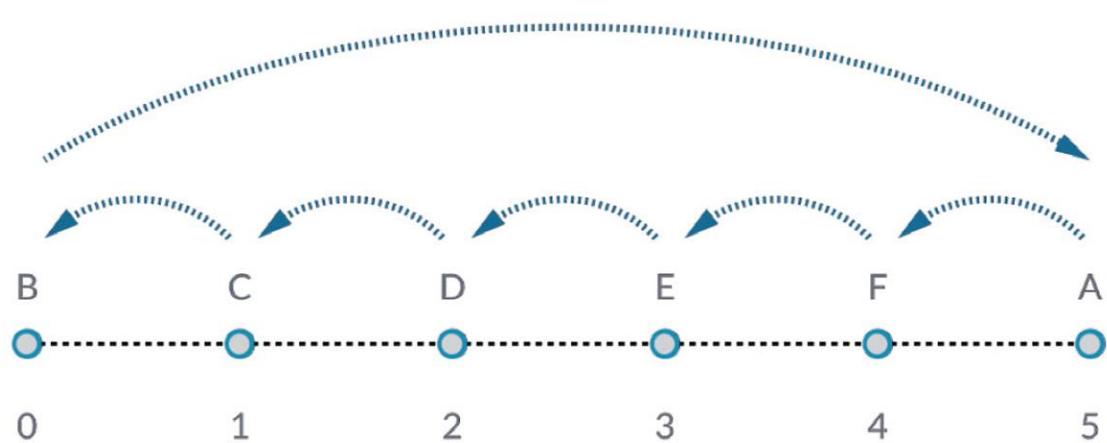
- **List.Count:** trả về một number có giá trị là số lượng đối tượng trong một list
- **List.GetItemAtIndex:** Lấy một phần tử trong list qua chỉ số Index của đối tượng đó trong list

*List.GetItemAtIndex*

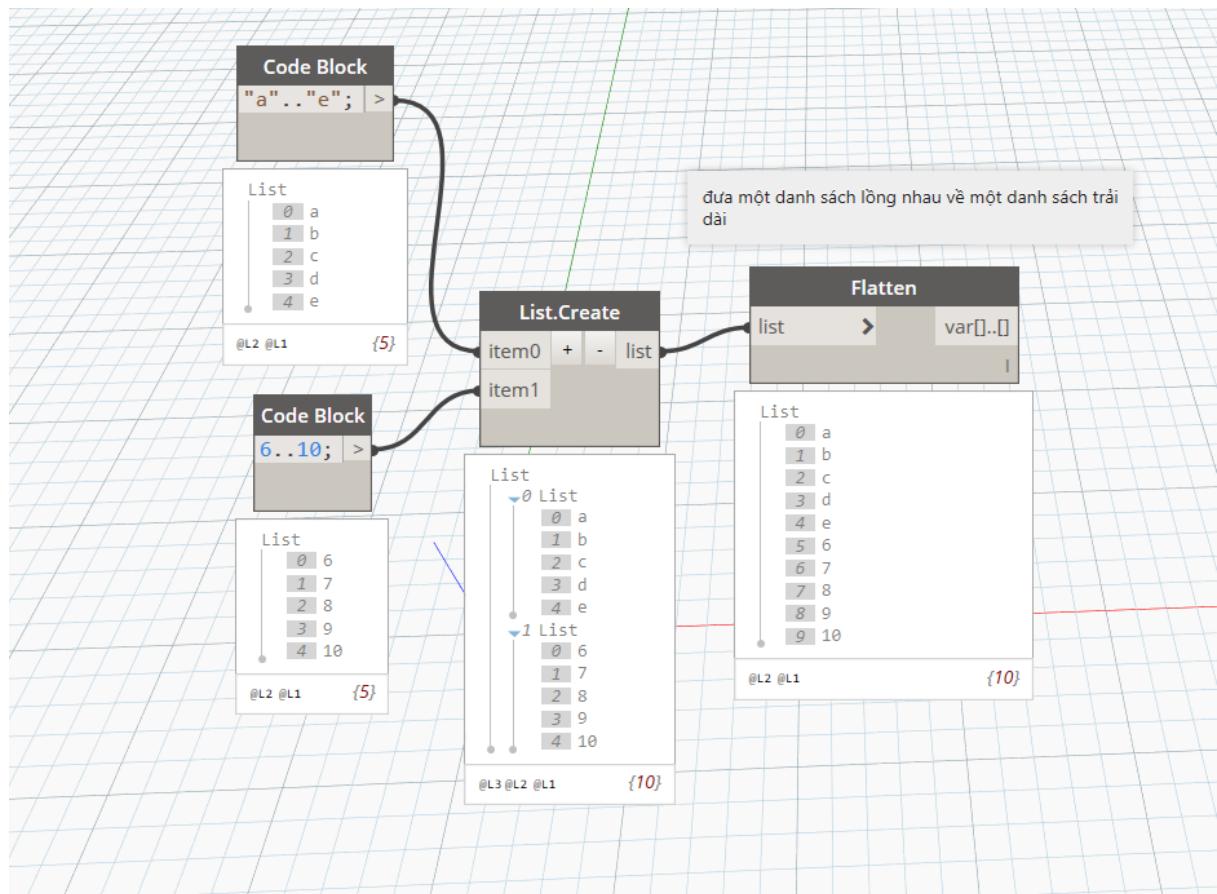
- **List.Reverse:** Đảo ngược trật tự của một list, nghĩa là đảo thứ tự của các phần tử từ chiều thuận sang chiều nghịch

*List.Reverse*

- **List.ShiftIndices:** Cắt một số lượng n phần tử đầu tiên của list và đưa xuống cuối List

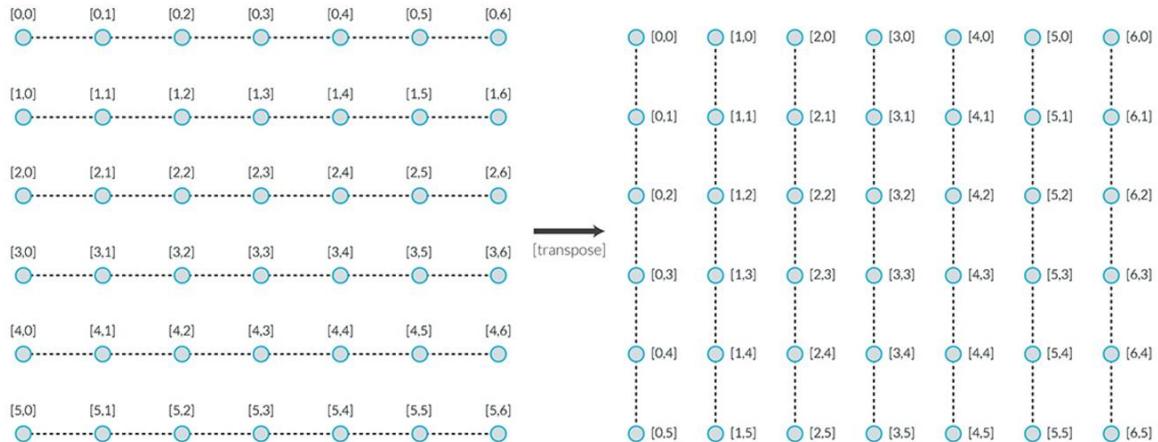


- **List.Flatten:** Giải phóng dữ liệu List trong list thành một list duy nhất, sử dụng trong trường hợp không cần phân cấp bậc dữ liệu, cần sử dụng cẩn trọng vì có thể mất thông tin



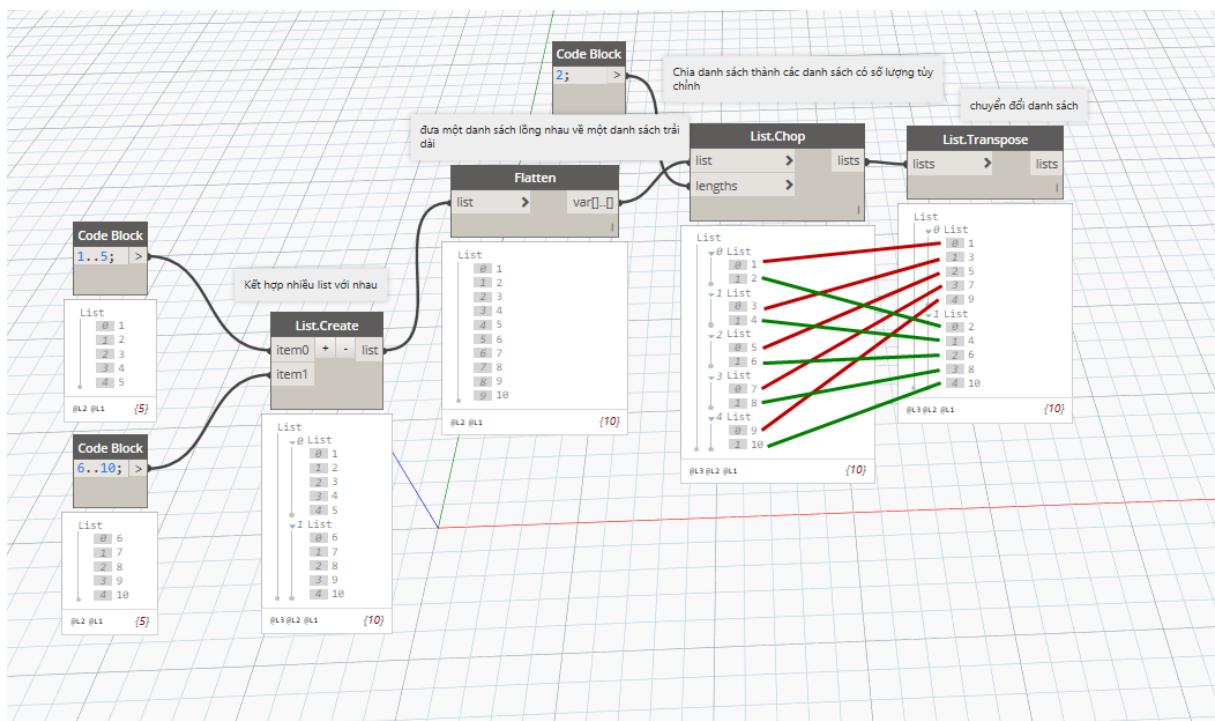
2 List con chỉ còn 1 sau khi được Flatten

- **ListTranspose:** đây là một trong những hàm cơ bản khi làm việc với List chằng List. Nếu xem như List trong list là những cấu trúc dữ liệu cột và hàng trong bảng Excel thì tác dụng của node này đó là chuyển đổi cột thành hàng và hàng thành cột.



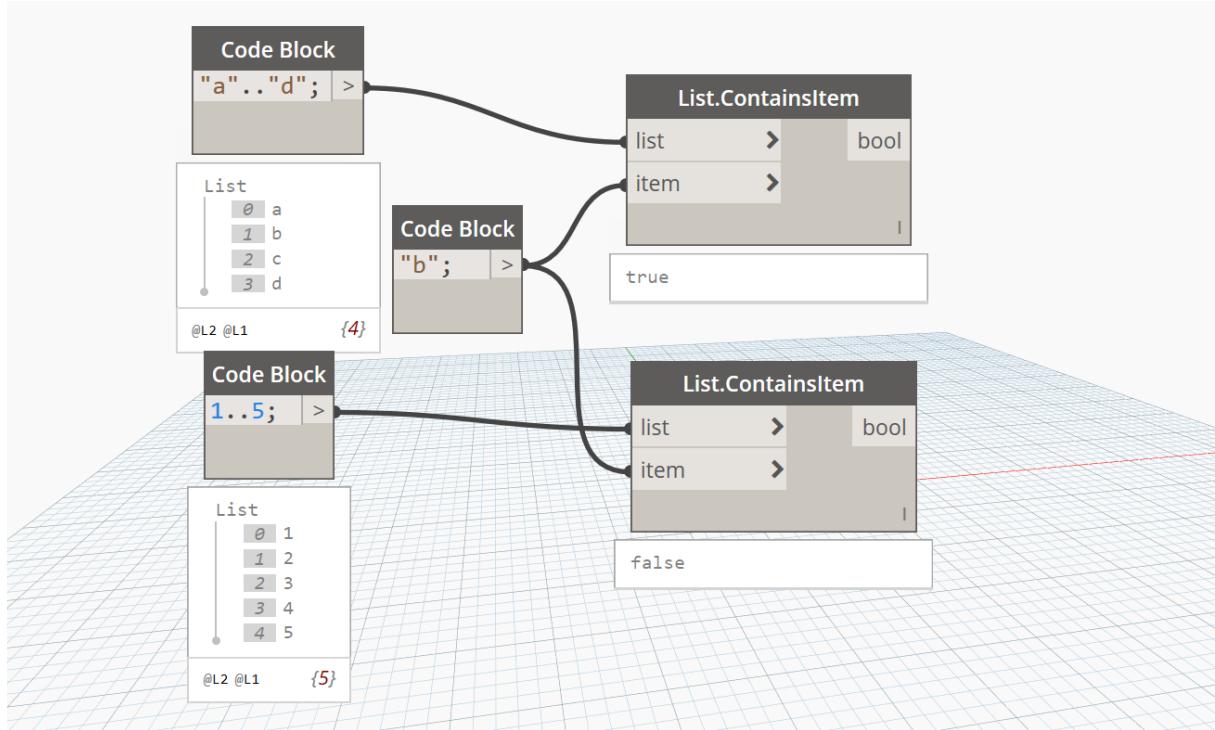
List.Transpose

+ Xem tiếp một ví dụ sau:



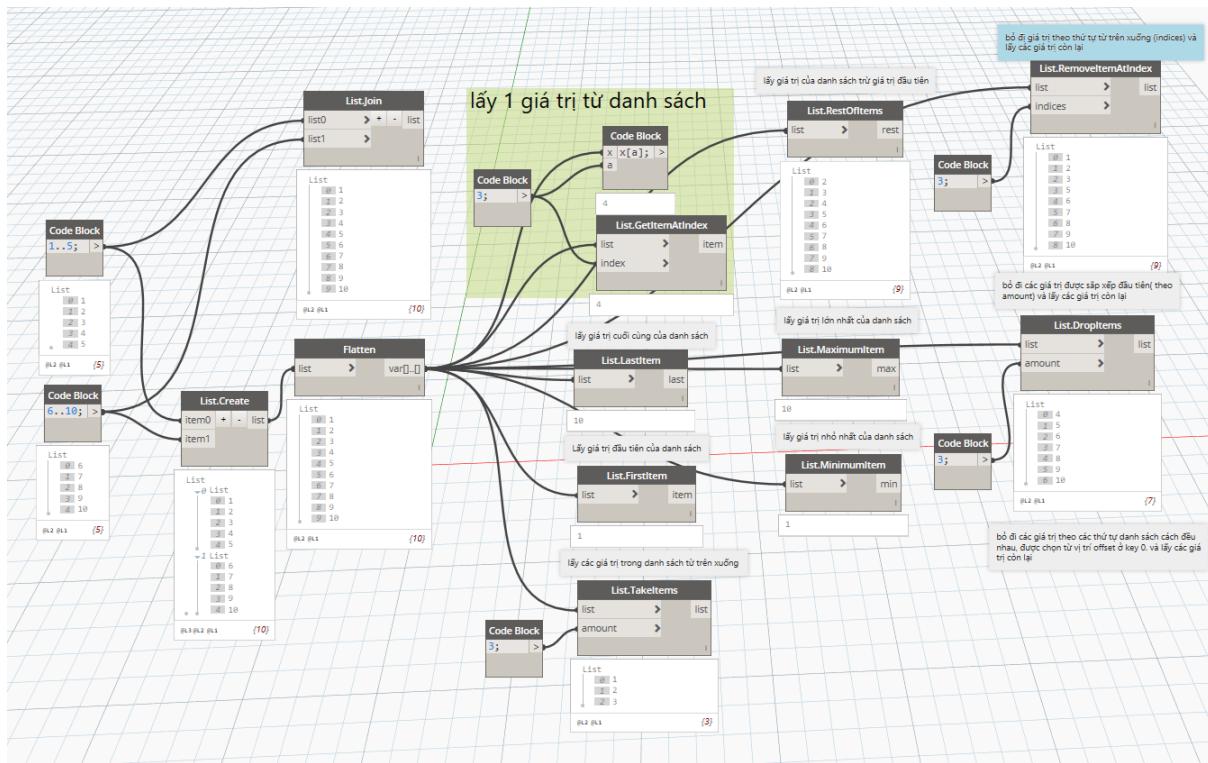
Transpose từ 5 list con 2 phần tử thành 2 list con 5 phần tử

- **List.ContainsItem**: Kiểm tra một giá trị có nằm trong list không và trả về true/false tương ứng có/không



Kiểm tra giá trị `b` trong 2 list

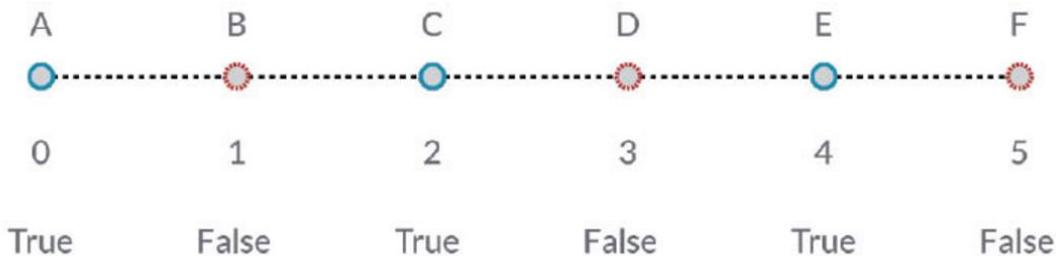
- **List.Sort:** Sắp xếp dữ liệu theo thứ tự từ nhỏ đến lớn
- **List.Join:** Gộp dữ liệu 2 hay nhiều list thành 1 list
- **List.MaximumItem:** Lấy giá trị lớn nhất trong 1 list
- **List.MinimumItem:** Lấy giá trị nhỏ nhất trong 1 list
- **List.FirstItem:** Lấy giá trị đầu tiên của list
- **List.LastItem:** Lấy giá trị cuối cùng của list
- **List.RestOfItems:** Lấy giá trị của danh sách trừ giá trị đầu tiên
- **List.RemoveItemAtIndex:** Bỏ đi giá trị có chỉ số index nhập vào, giữ các giá trị còn lại
- **List.DropItems:** Bỏ đi amount (nhập vào) phần tử theo thứ tự từ trên xuống và giữ các phần tử còn lại



Thao tác với List cơ bản

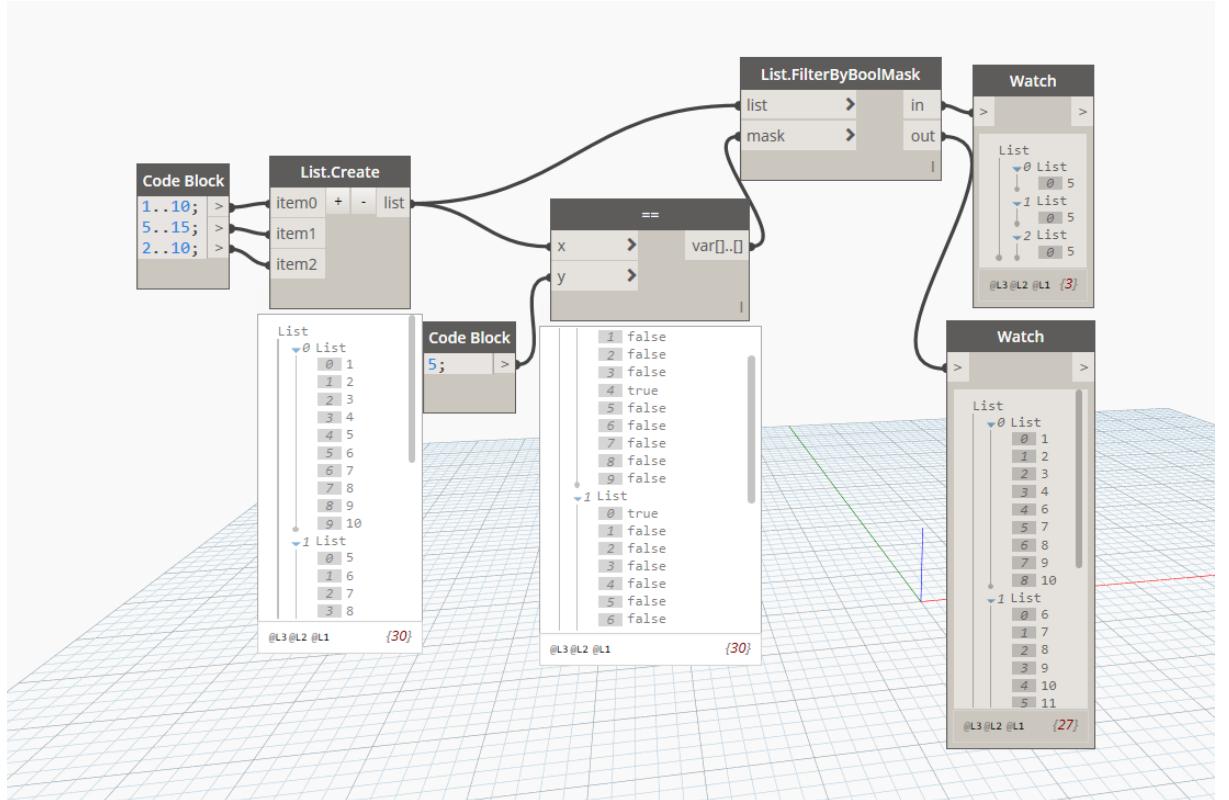
II. Thao Tác List Nâng Cao

- **List.FilterByBooleanMask**: Phân loại danh sách theo một danh sách true/false tương ứng



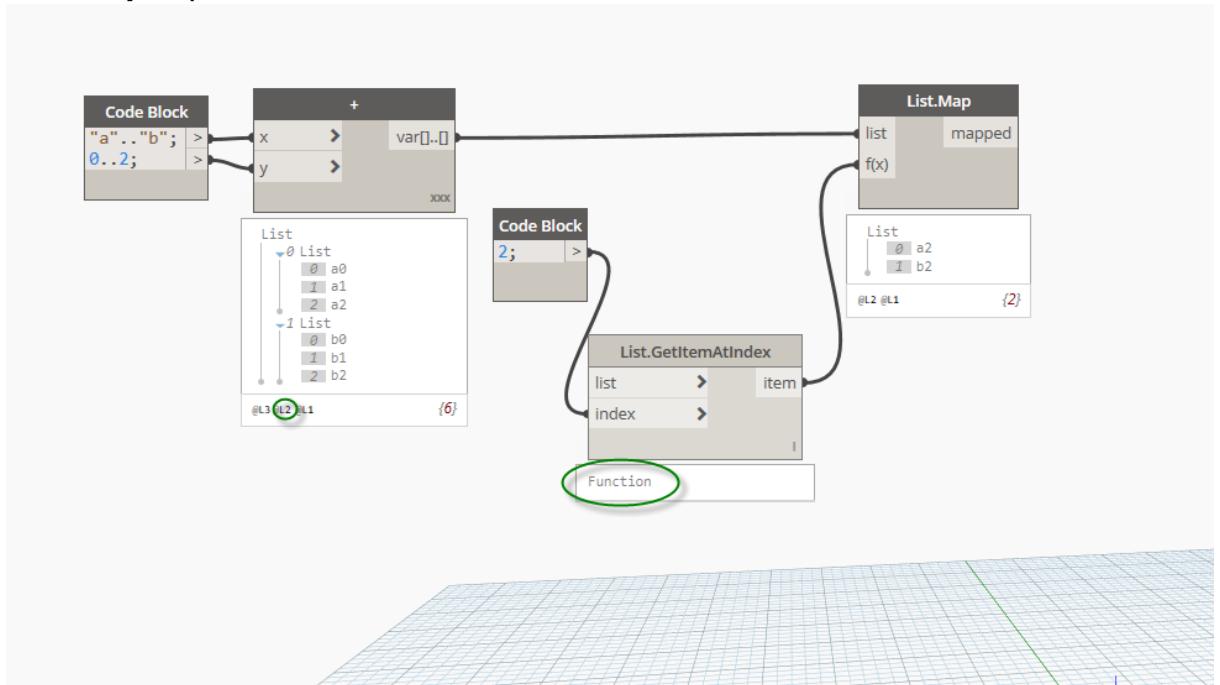
List.FilterByBooleanMask

+ Xem thêm ví dụ sau: Lọc các số có giá trị bằng 5 từ một list



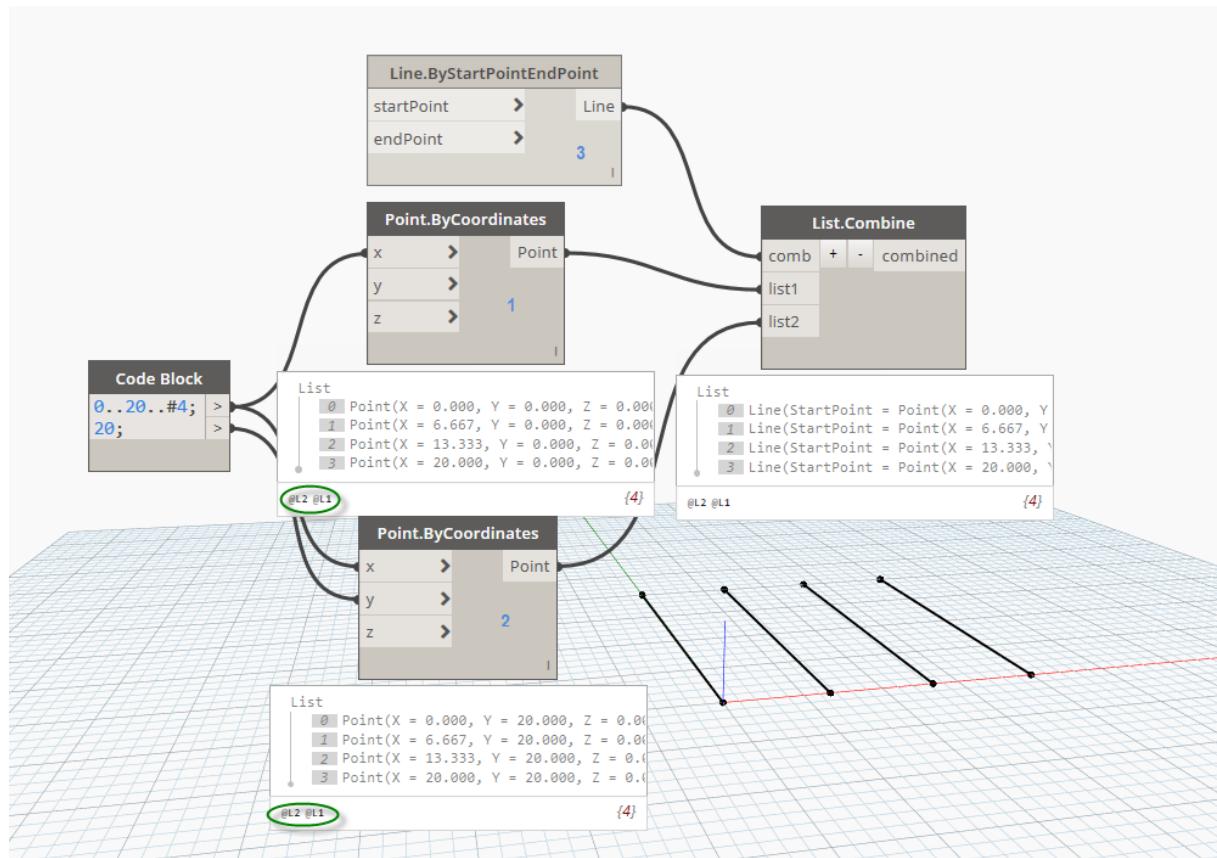
Lọc số 5 ra khỏi dãy số

- **List.Map:** Áp một function vào một list tại @L max-1.



- + List level cao nhất ở đây là @L3, vậy function sẽ được áp vào @L2, cho gồm cả 2 list con.

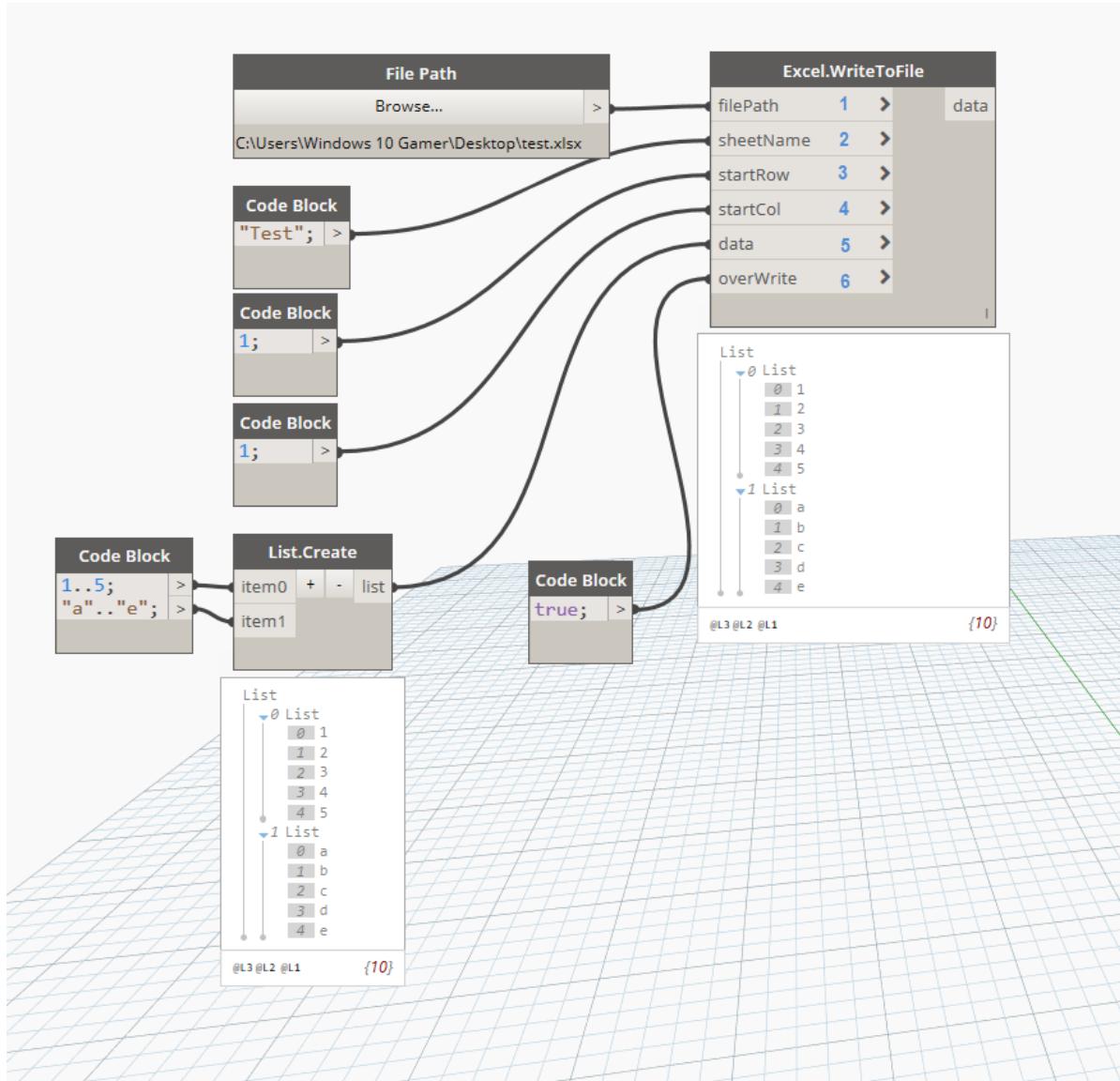
- + Node List.GetItemAtIndex khi ko có dữ liệu list truyền vào thì nó sẽ là một function, ở đây là function lấy giá trị có index là 2
- + Kết quả là từ mỗi list ta lấy được một giá trị và được một list mới.
- **List.Combine**: Tương tự như List.Map, chỉ khác là node này có thể áp nhiều list phù hợp với nhau.



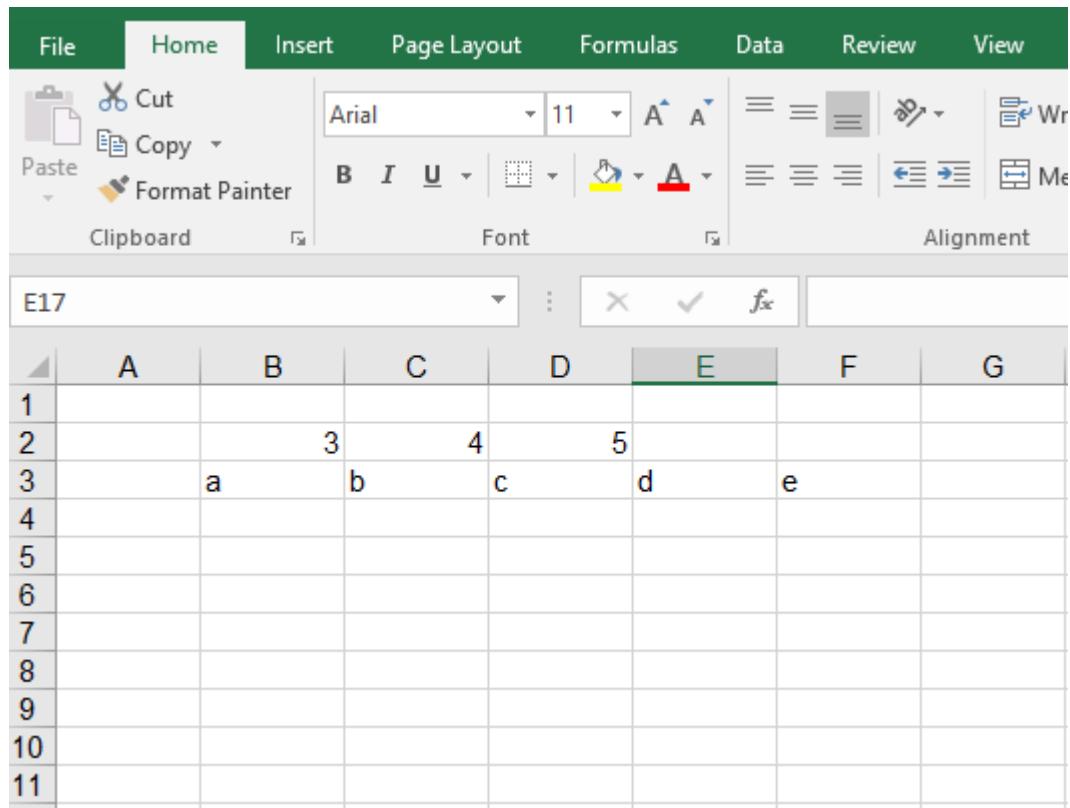
- + 1,2 list Point gồm 4 phần tử có @L giống nhau
- + 3: function tạo Line từ 2 điểm
- + Kết quả ta được 4 line tạo ra từ 4 cặp điểm tương ứng

F. REVIT TO EXCEL

I. Import To Excel



- + **1** filePath: đường dẫn đến file excel cần ghi dữ liệu, file này phải tạo sẵn
- + **2** sheetName: Tên Sheet ghi dữ liệu vào, ở trên là "Test"
- + **3** startRow: vị trí hàng bắt đầu ghi dữ liệu, hàng đầu tiên là 0, ở trên bắt đầu ghi ở hàng thứ 2
- + **4** startCol: vị trí cột bắt đầu ghi dữ liệu, cột đầu tiên là 0, ở trên bắt đầu ghi ở cột thứ 2
- + **5** data: Dữ liệu ghi vào
- + **6** overWrite: Có ghi đè file hay ko, tương ứng với giá trị True/False

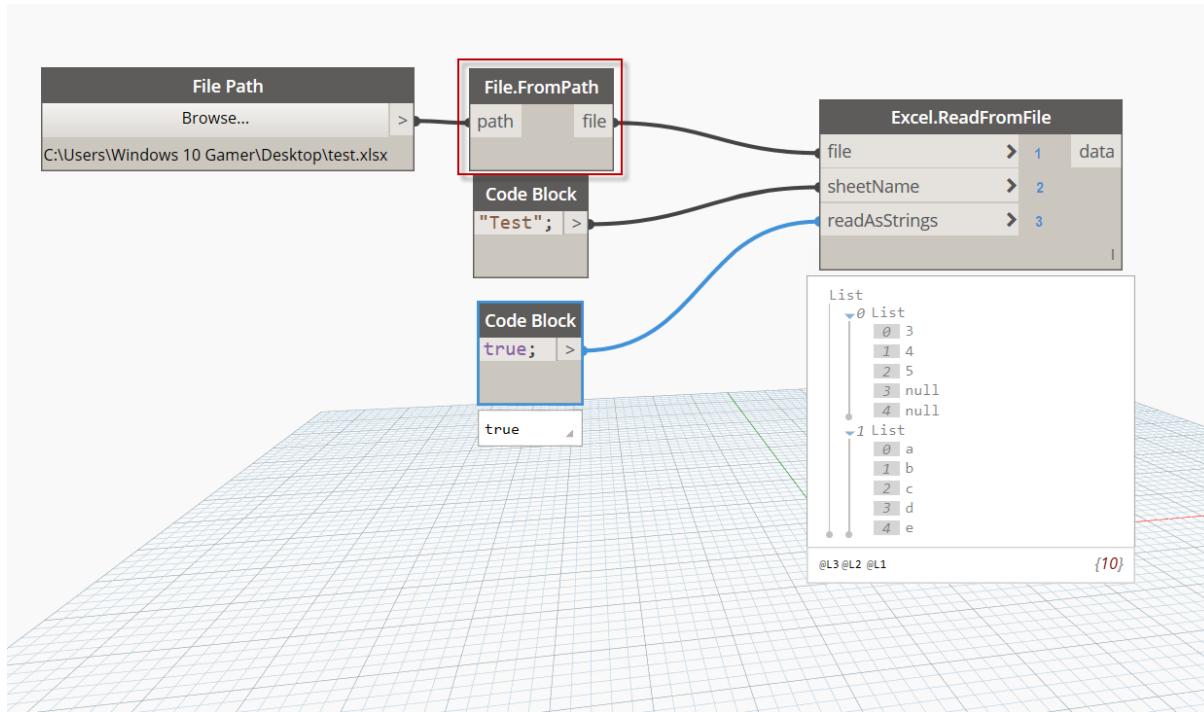


A screenshot of a Microsoft Excel spreadsheet. The grid consists of 25 cells arranged in 5 rows and 5 columns. The columns are labeled A through G, and the rows are labeled 1 through 5. The data entered into the cells is as follows:

	A	B	C	D	E	F	G
1							
2		3	4	5			
3	a	b	c	d	e		
4							
5							
6							
7							
8							
9							
10							
11							

File Excel xuất ra

II. Export from Excel



+ 1: file excel để đọc dữ liệu, ở đây là file chứ không phải file path như ở node **Excel.WriteToExcel** phía trên nên ta cần thêm một node lấy file từ filePath là **File.FromPath**

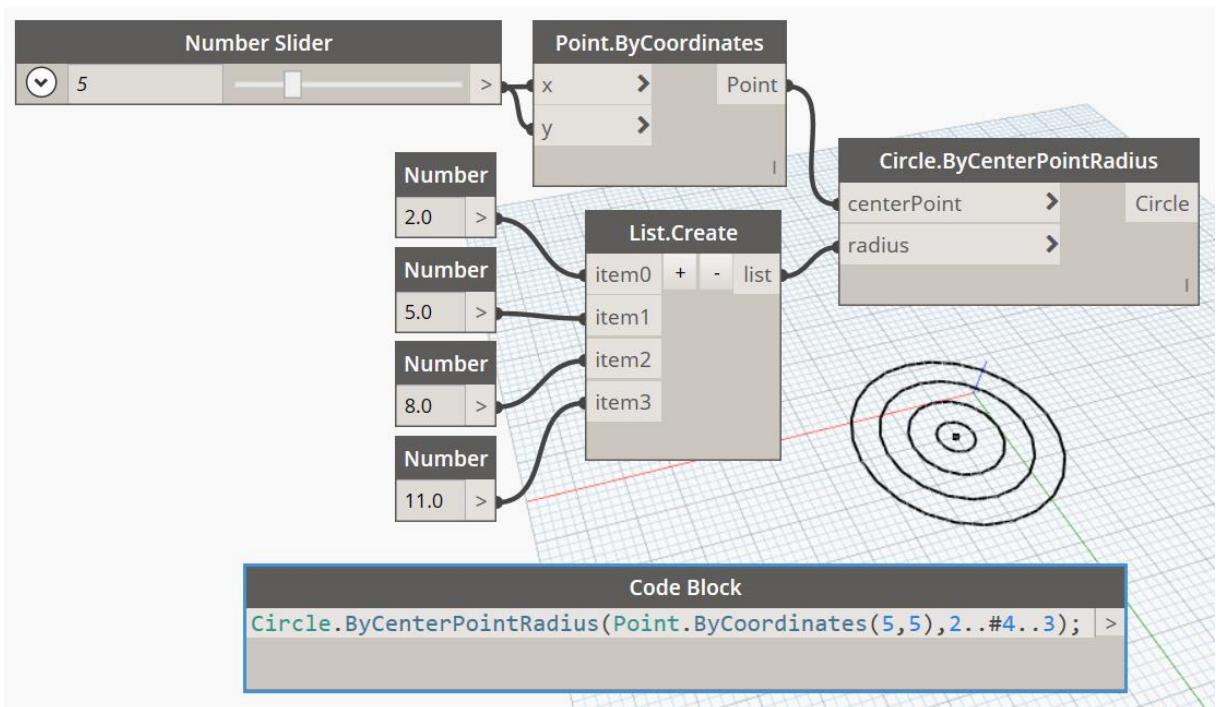
+ 2: tên sheet trong file cần đọc dữ liệu

+ 3: dữ liệu đọc được sẽ đưa về String hay không, tương ứng True/False

G.CODEBLOCK VÀ DESIGNSCRIPT

I. Giới thiệu cơ bản về CodeBlock và DesignScript

- Hiểu một cách cơ bản, Code Block là cửa sổ (nút) trong Dynamo có thể tái hiện được tất cả các nút, chuỗi các nút bằng những dòng lệnh. Nó là một nút giúp người đọc dễ dàng tiếp cận với ngôn ngữ lập trình, từ đó có thể lập trình một cách linh động theo ý của mình nhằm rút gọn chuỗi các nút không cần thiết trong quá trình sử dụng Dynamo.
- DesignScript về cơ bản nó là ngôn ngữ lập trình ngắn gọn cung cấp phương thức (hành động) tác động đến tất cả các đối tượng trong Dynamo, nó có thể đọc và phản hồi một cách nhanh chóng những dòng lệnh đơn giản cũng như phức tạp mà ta nhập vào ở nút Code Block.

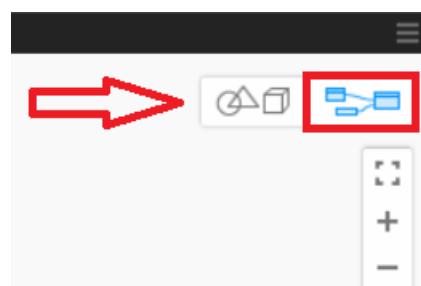


Hình minh họa cho Code Block

II. Cách tạo lập và chức năng của Code Block và DesignScript

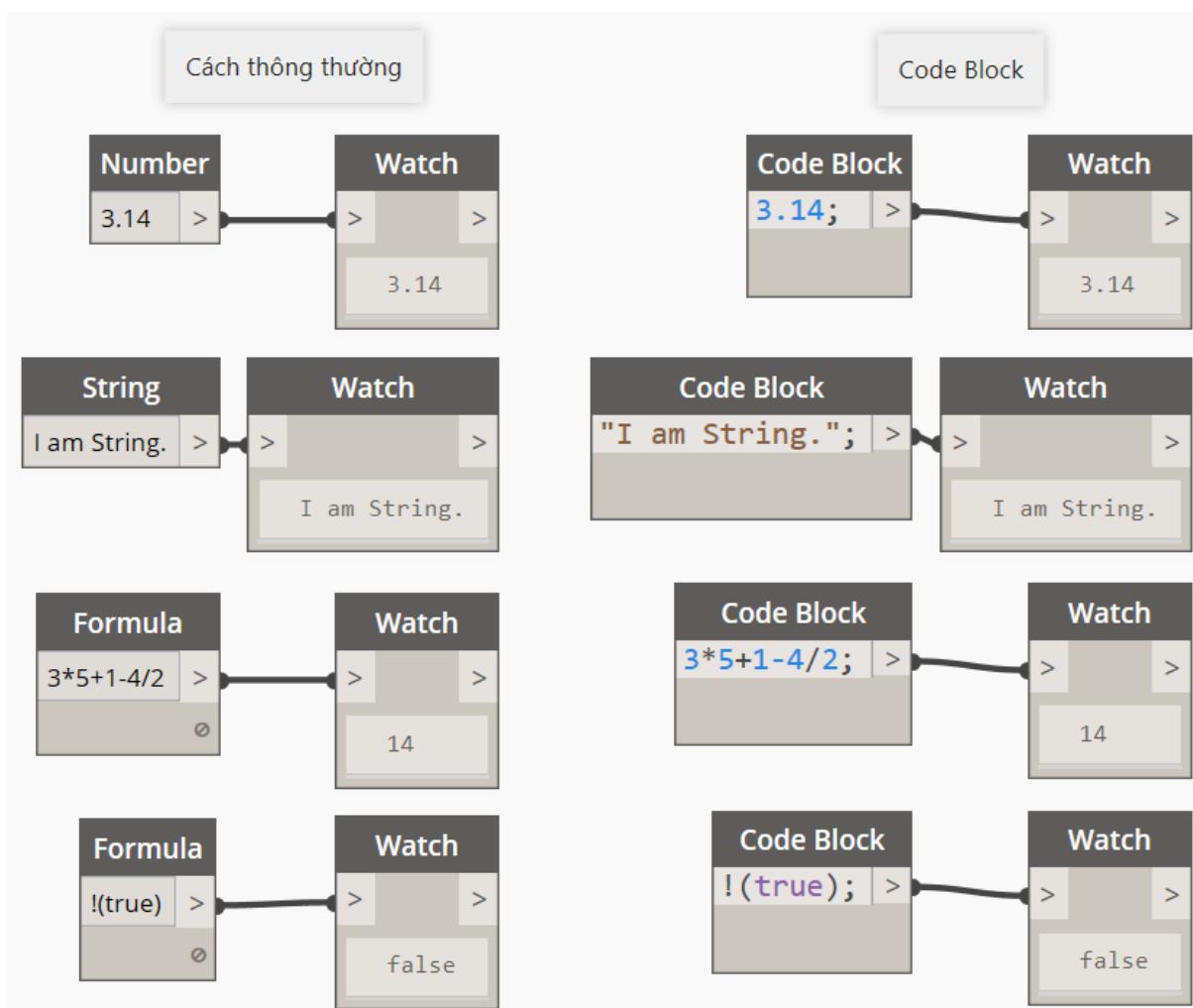
1. Code Block

- Có 2 cách tạo lập:
 - + Vào Library ⇒ Core ⇒ Input ⇒ Code Block.
 - + Kích đúp vào mặt phẳng làm việc (Workspace) ở chế độ tạo nút (Enable Graph View Navigation).



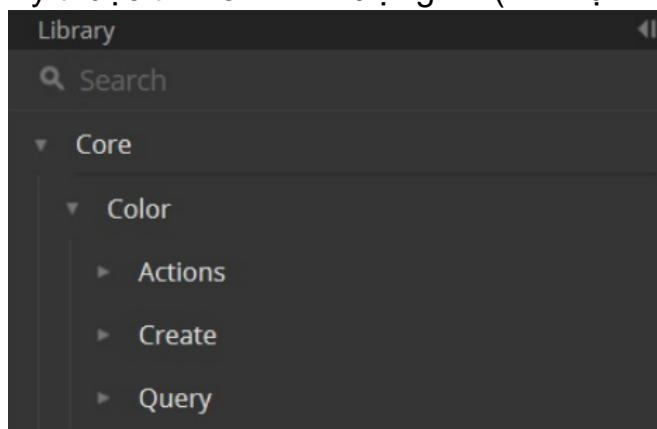
Kiểu dữ liệu hỗ trợ (sau mỗi dòng lệnh phải có **dấu chấm phẩy**)

- + Number (kiểu số): Số nguyên, số thực ...
- + String (kiểu chuỗi).
- + Formulas (dạng công thức: cộng, trừ, nhân, chia ...).

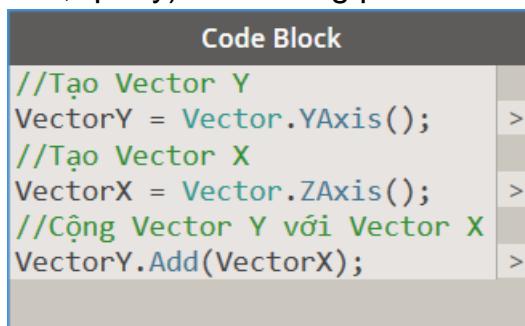


2. DesignScript Syntax

- Trong thư viện nút của Dynamo sắp xếp theo cấu trúc như sau (tùy đối tượng có hay không).
 - + Create: Tạo lập đối tượng (khu vực có dấu +).
 - + Action: Thực hiện hành động của đối tượng (khu vực có dấu tia chớp).
 - + Query: Lấy thuộc tính của đối tượng đó (khu vực có dấu chấm hỏi).



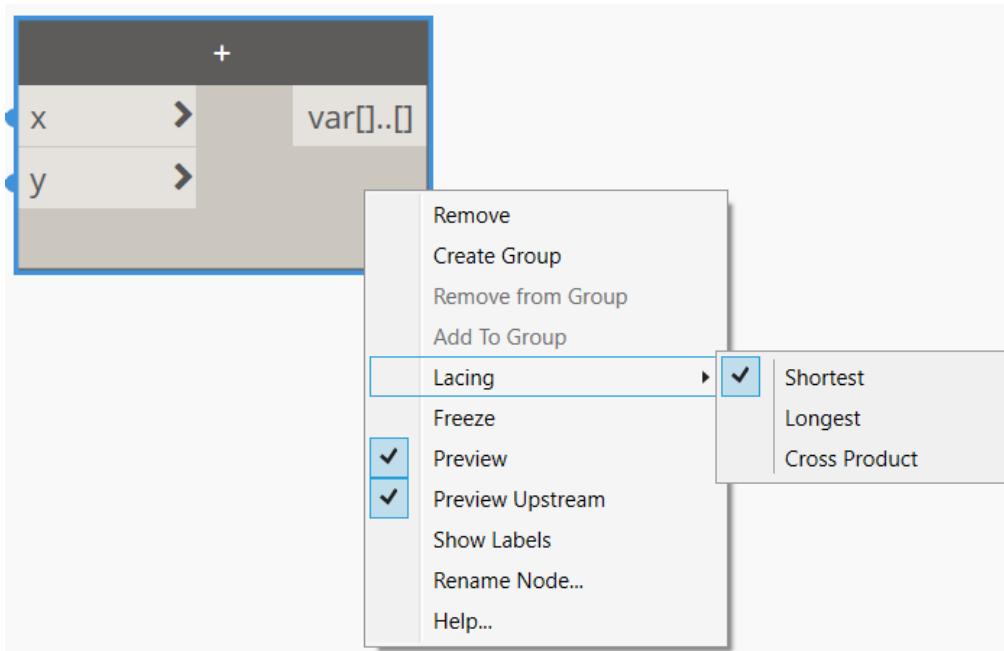
- Cú pháp DesignScript trong Code Block là giữa đối tượng với phương thức (action, create, query) của chúng phải có **dấu chấm**.



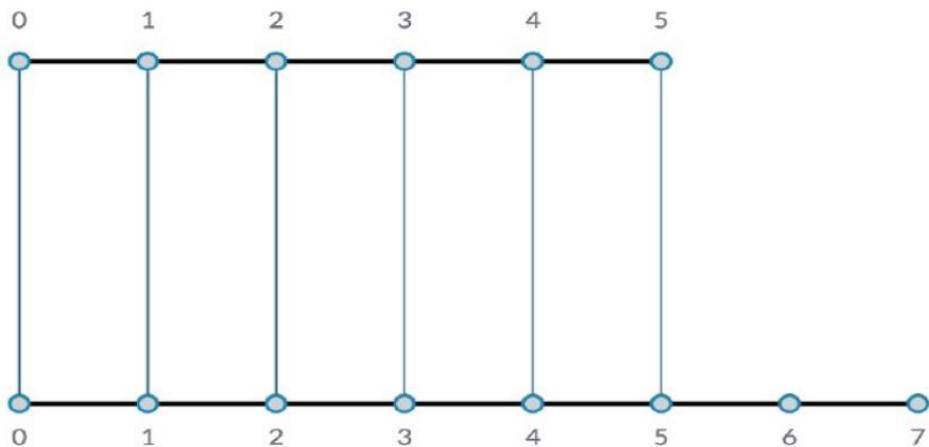
```
//Tạo Vector Y
VectorY = Vector.YAxis();
//Tạo Vector X
VectorX = Vector.ZAxis();
//Cộng Vector Y với Vector X
VectorY.Add(VectorX);
```

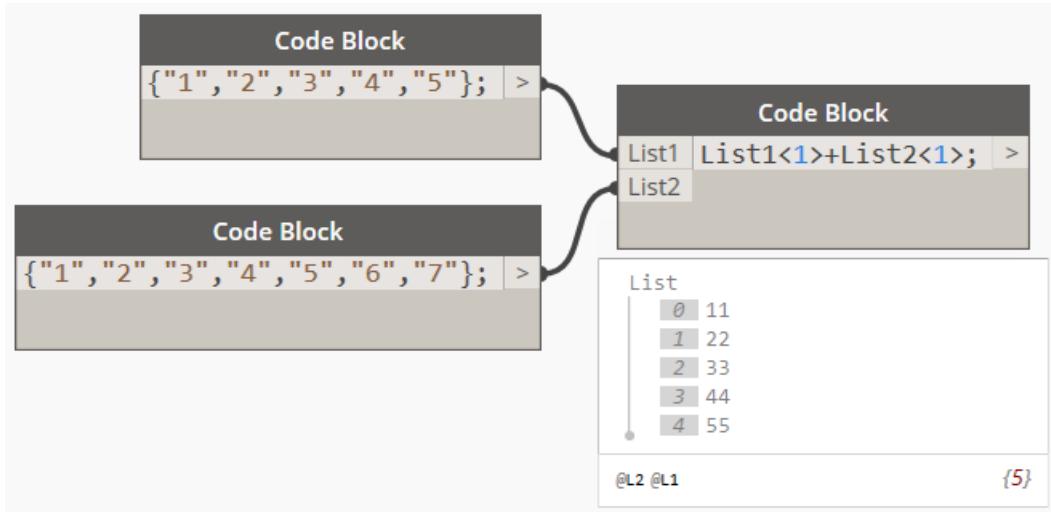
3. Lacing trong Code Block

- Nút trong Dynamo, chúng ta kích chuột phải có mục **Lacing**. Tính năng này khá hay và quan trọng.

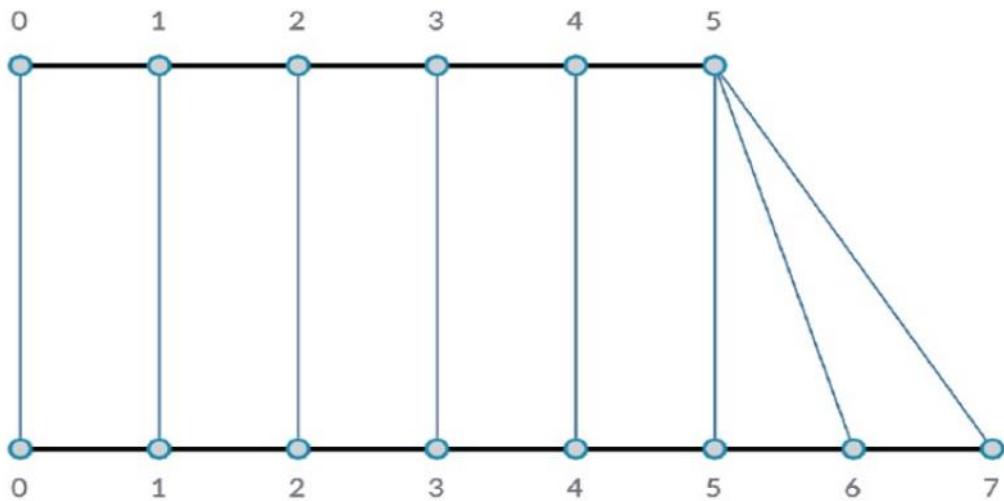


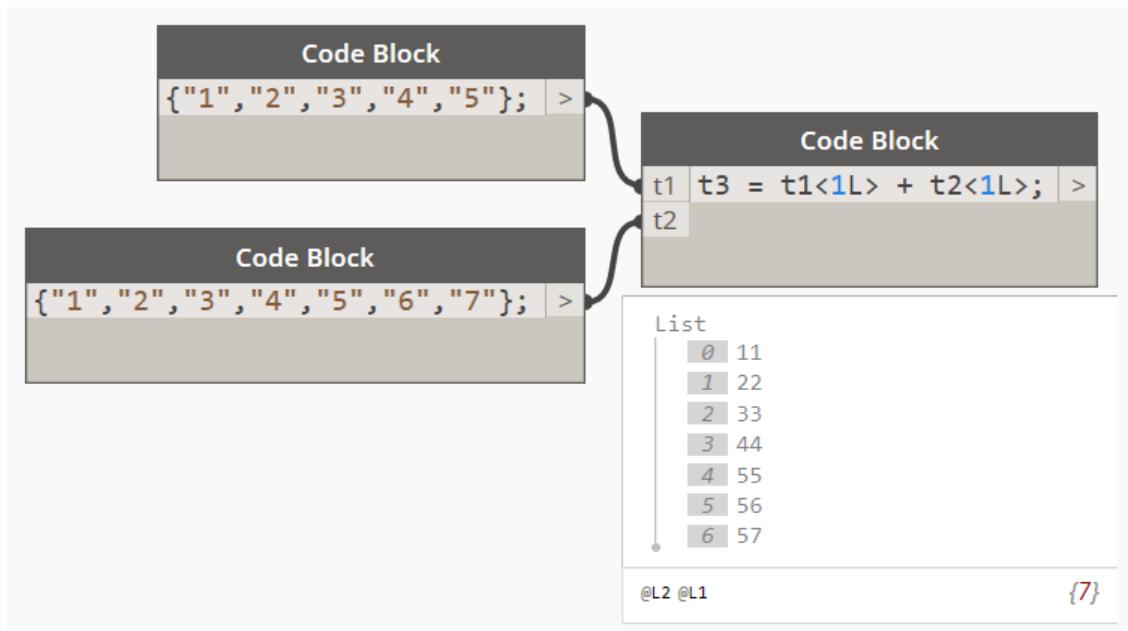
- Tương tự trong Code Block, Lacing có 3 chế độ
 - + Shortest (chế độ mặc định): Kết nối theo kiểu một - một bắt đầu từ phần tử thứ “0”, bên nào hết phần tử trước thì dừng. Như hình minh họa bên dưới.



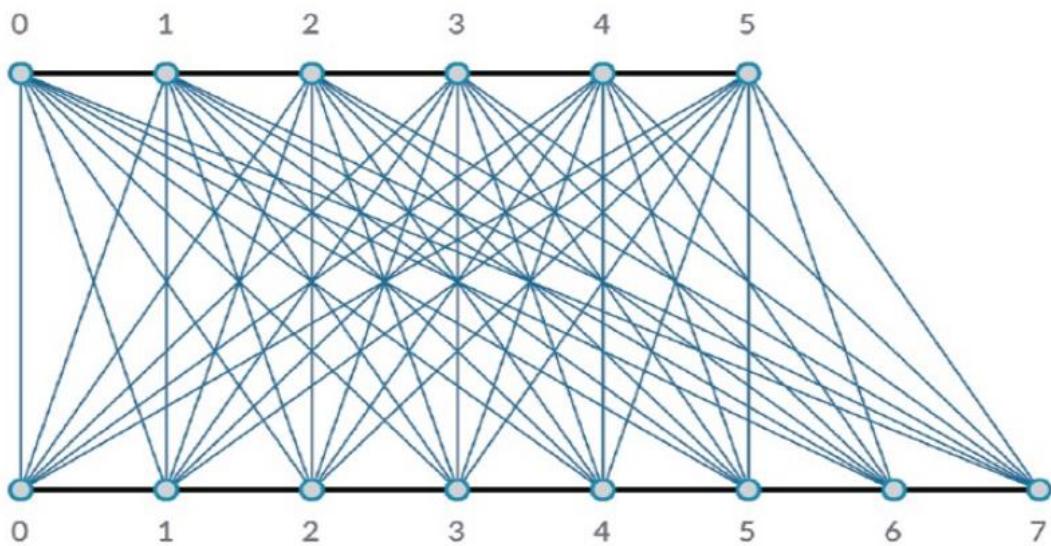


+ Longest: Tương tự như Shortest nhưng phần tử cuối cùng của List ngắn hơn kết nối với các phần tử còn lại của list dài hơn. Kí hiệu **<1L>**. Hình minh họa bên dưới.





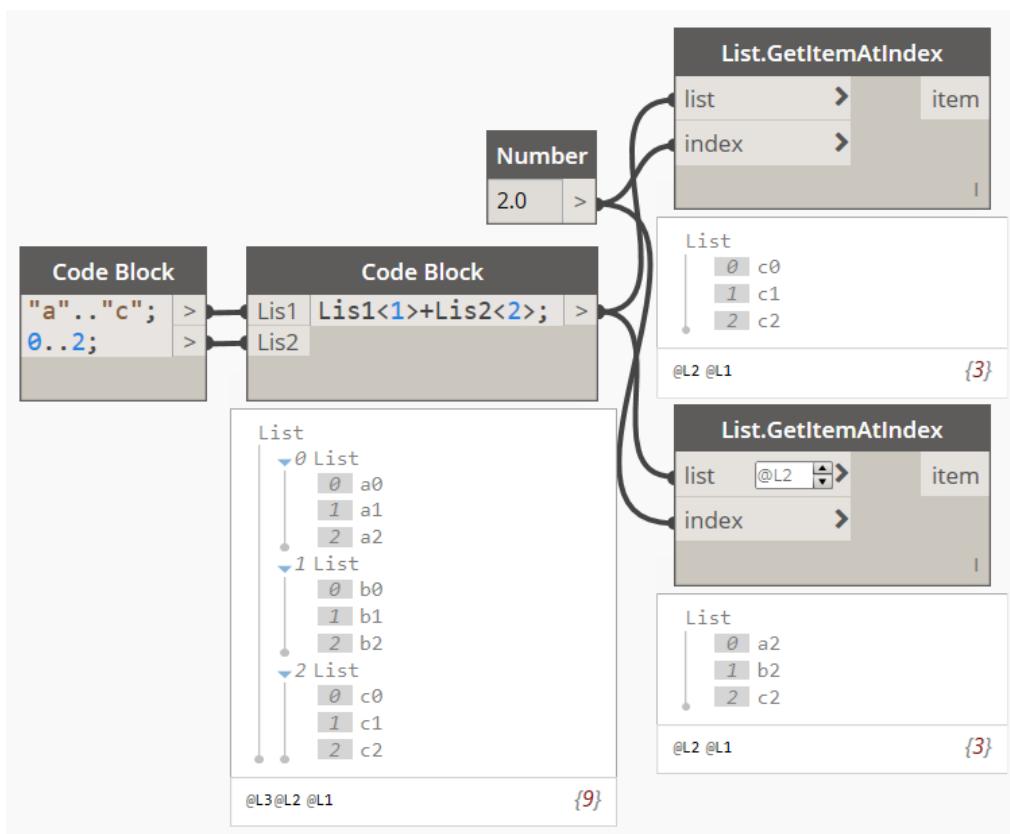
+ Cross Product: Kết nối theo kiểu tuần tự, một phần tử List này với tất cả phần tử List kia tương tự cho ngược lại nếu thay đổi trình tự đầu vào. Kí hiệu `<1><2>` hay `<2><1>`. Như hình minh họa bên dưới.



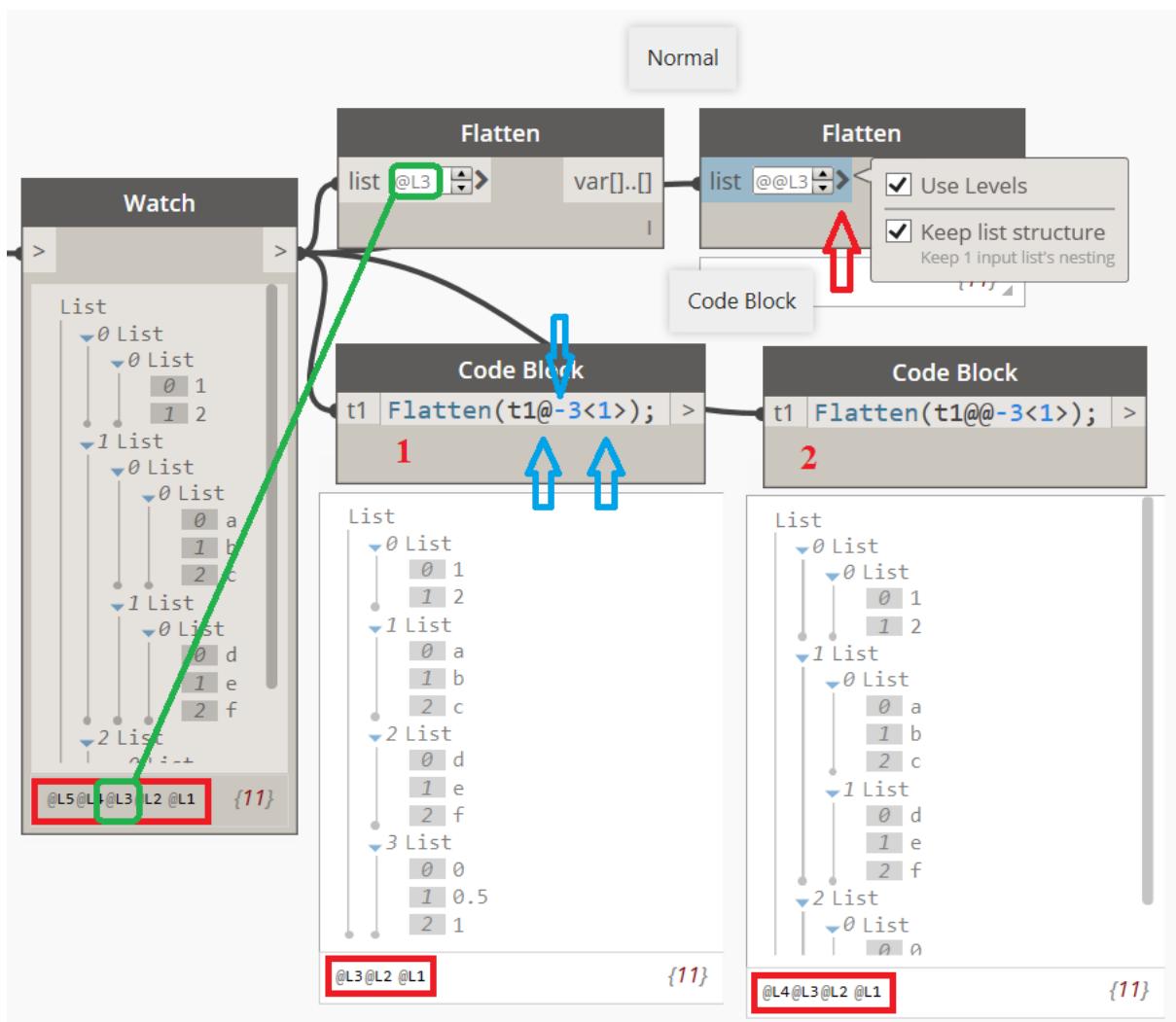


4. List@Level trong Code Block

- Tính năng này giúp cho thao tác với list có nhiều level (@L) cực kì dễ dàng. Được bổ sung từ phiên bản 1.2.
- Hướng dẫn từ Dynamo <http://dynamobim.org/introducing-listlevel-working-with-lists-made-easier/>
- Trong 1 list có nhiều level, khi kết nối chúng ở những level khác nhau thì ta dùng List@Level.



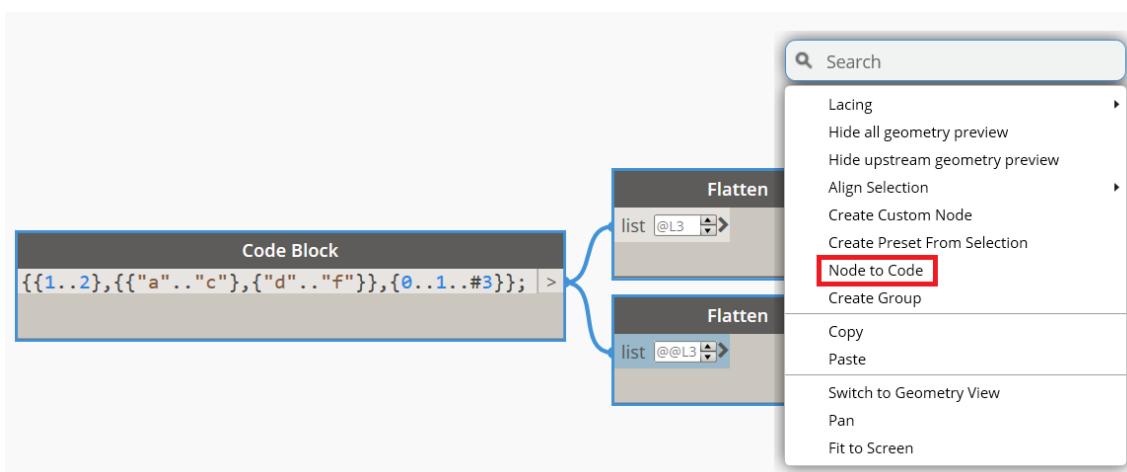
Ảnh minh họa dùng `List@Level`



- Cú pháp “**ListInput**” “**@@|@**” “-“ “**NumberLevel**“ “**Lacing**”
 - + @ (use levels) không giữ cấu trúc level ban đầu như hình 1.
 - + @@ (keep list structure) giữ cấu trúc ban đầu như hình 2.
 - + Dấu phân cách (-)
 - + NumberLevel, cấp level mà chúng ta tương tác ở đó.
 - + <> là Lacing như trình bày ở phần **Lacing**.

5. Node To Code

- Đây là cách thức học nhanh nhất cho những ai chưa quen lập trình, giống như ghi macro trong học lập trình excel, tức là từ những nút ta tạo ra giờ chuyển thành những dòng lệnh trong code block để học hỏi rồi phát triển thêm.
- Cú pháp: Quét chọn nút, kích chuột phải bên ngoài các nút, chọn Node To Code.



Trước khi chuyển thành code

```
Code Block
t1 = {{1..2}}, {"a".."c"}, {"d".."f"}}, {0..1..#3}}; >
t2 = Flatten(t1@-3<1>); >
t3 = Flatten(t1@@-3<1>); >
```

Sau khi chuyển thành code

6. Logic Trong Code Block

- Đôi khi dùng nút để thực hiện điều kiện khá dài dòng. Trong Code Block cung cấp cú pháp ngắn gọn như sau:
 - + Cho 1 điều kiện
điều kiện ? thực hiện nếu điều kiện đúng : thực hiện nếu điều kiện sai
 - + Cho nhiều điều kiện
điều kiện 1 ? thực hiện nếu điều kiện 1 đúng : điều kiện 2 ? thực hiện nếu điều kiện 2 đúng : thực hiện nếu điều kiện 2 sai



7. Tạo list và lấy giá trị trong list

- Tạo list
 - + Cú pháp cơ bản
 - 0..10..1** hay **0..10** tạo một danh sách các số từ 0 đến 10.
 - 0..#15..2** tạo một danh sách 15 số với số bắt đầu 0 và bước nhảy 2.
 - 0..15..#10** tạo một danh sách 10 số chia đều đoạn 0 đến 15.
 - + Cú pháp nâng cao

0..#5..(2..3) tạo ra 2 danh sách 5 phần tử bắt đầu từ số không với bước nhảy làn lượt 2 và 3.

0..5..(2..3) tạo 2 danh sách với số bắt đầu 0 không có giá trị quá 5 với bước nhảy làn lượt 2 và 3.

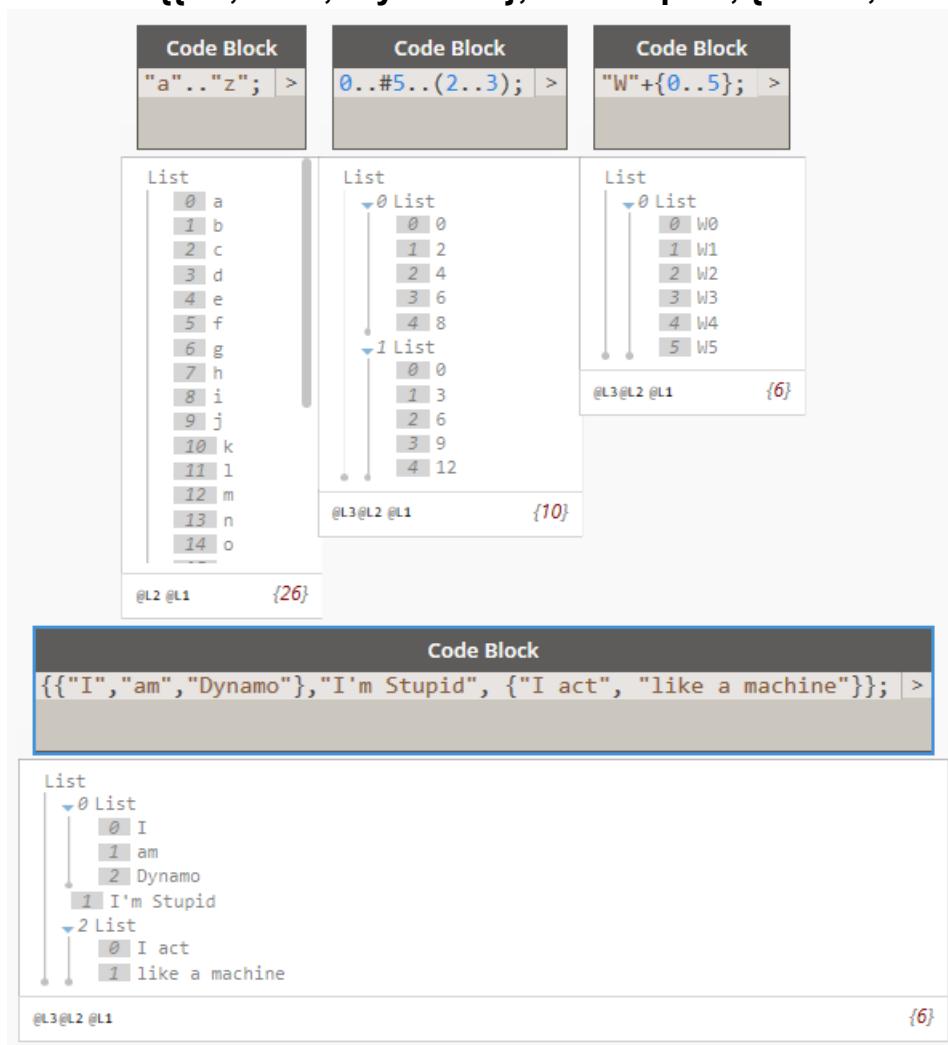
(0..1)..5..2 tạo 2 danh sách bắt đầu 0 và 1 không có giá trị quá 5 với bước nhảy là 2.

0..(2..4)..1 tạo ra 3 danh sách bắt đầu là 0 kết thúc làn lượt là 2; 3; 4 với bước nhảy là 1.

“a”..”z” tạo danh sách chữ cái từ a đến z.

“W”+{0..5} tạo danh sách W0 đến W5.

{“I”, “am”, “Dynamo”}, “I’m Stupid”, {"I act", "like a machine"}}}



The screenshot shows a complex Dynamo script structure. It starts with three code blocks:

- Code Block: "a".."z"; >
- Code Block: 0..#5..(2..3); >
- Code Block: "W"+{0..5}; >

These blocks feed into three parallel lists:

- List 1 (from "a".."z") contains 26 elements labeled a through z.
- List 2 (from 0..#5..(2..3)) contains 10 elements: 0, 2, 4, 6, 8, 0, 3, 6, 9, 12.
- List 3 (from "W"+{0..5}) contains 6 elements: W0, W1, W2, W3, W4, W5.

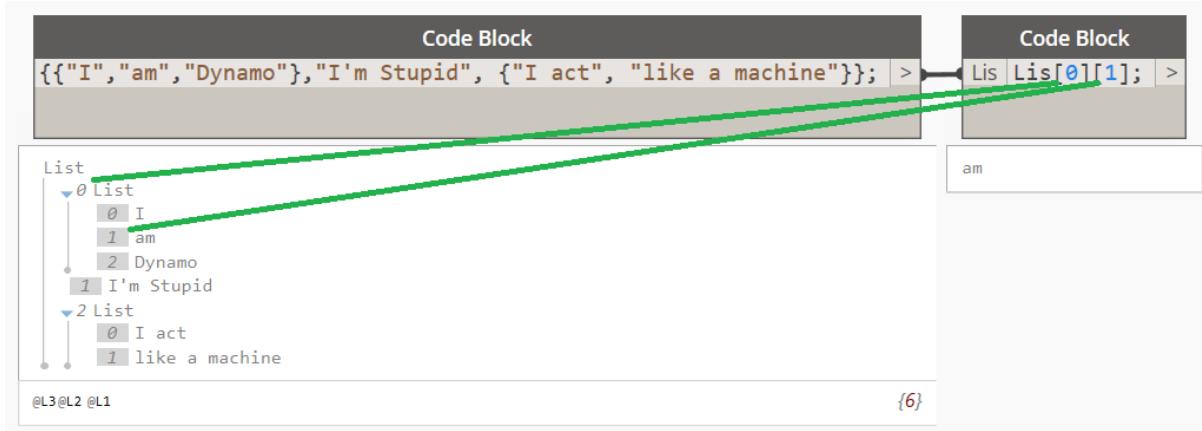
The final output is a single list containing the concatenated strings from the first list and the lists from the second and third blocks:

```
{{"I", "am", "Dynamo"}, "I'm Stupid", {"I act", "like a machine"}}; >
```

This results in a final list structure:

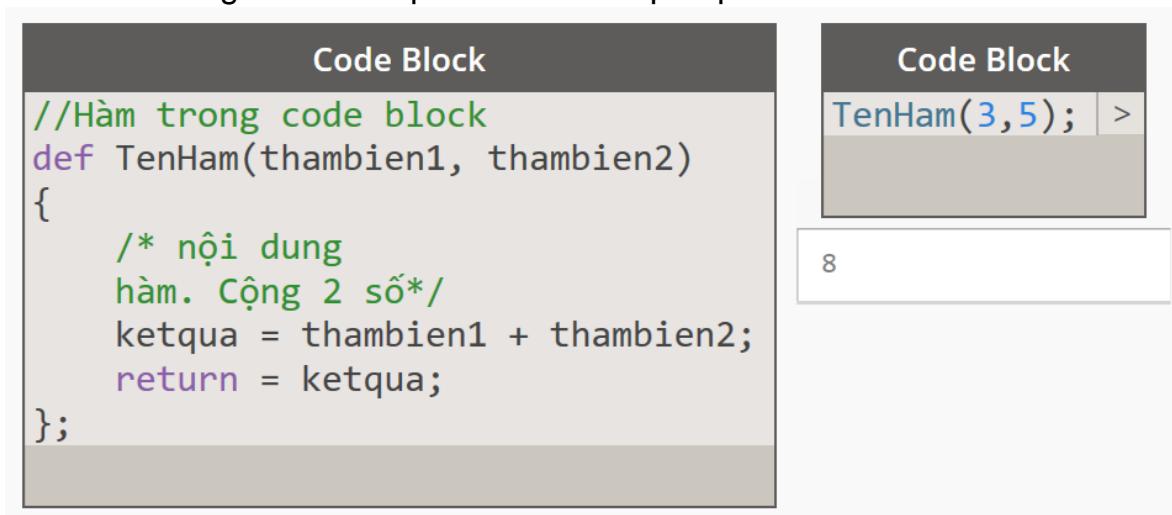
```
List
  0 List
    0 I
    1 am
    2 Dynamo
  1 I'm Stupid
  2 List
    0 I act
    1 like a machine
```

- Lấy giá trị
 - + Cú pháp: **List[Number1InLevel1][NumberInNumber1Level1]**
 - + Chỉ số bắt đầu là 0.



8. Hàm Trong CodeBlock

- Ghi chú trong Code Block
 - + Đơn dòng: **// comment**
 - + Đa dòng: **/* comment */**
- Trong trường hợp một thao tác nào đó lặp đi lặp lại nhiều lần và cú pháp phức tạp, dùng code block định nghĩa hàm với một tên riêng và dùng ở nhiều nơi trong Dynamo.
- Cú pháp và ví dụ như hình
 - + Có từ khóa **def** ở đầu dòng, tiếp đến **tên hàm** đến dấu **ngoặc tròn**.
 - + Trong dấu ngoặc tròn có thể có **nhiều biến hoặc không**.
 - + Thân code được bao bởi dấu **{ }** .
 - + Trong thân code phải trả về kết quả qua từ khóa **return**.



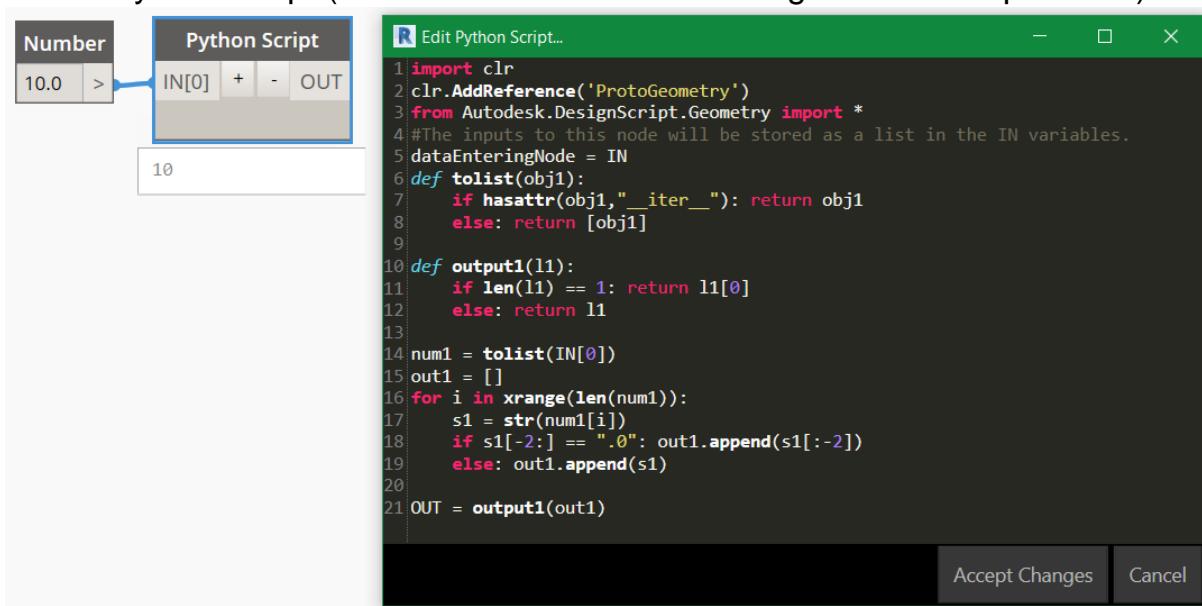
H. PYTHON IN DYNAMO

I. Giới thiệu sơ lược về Python

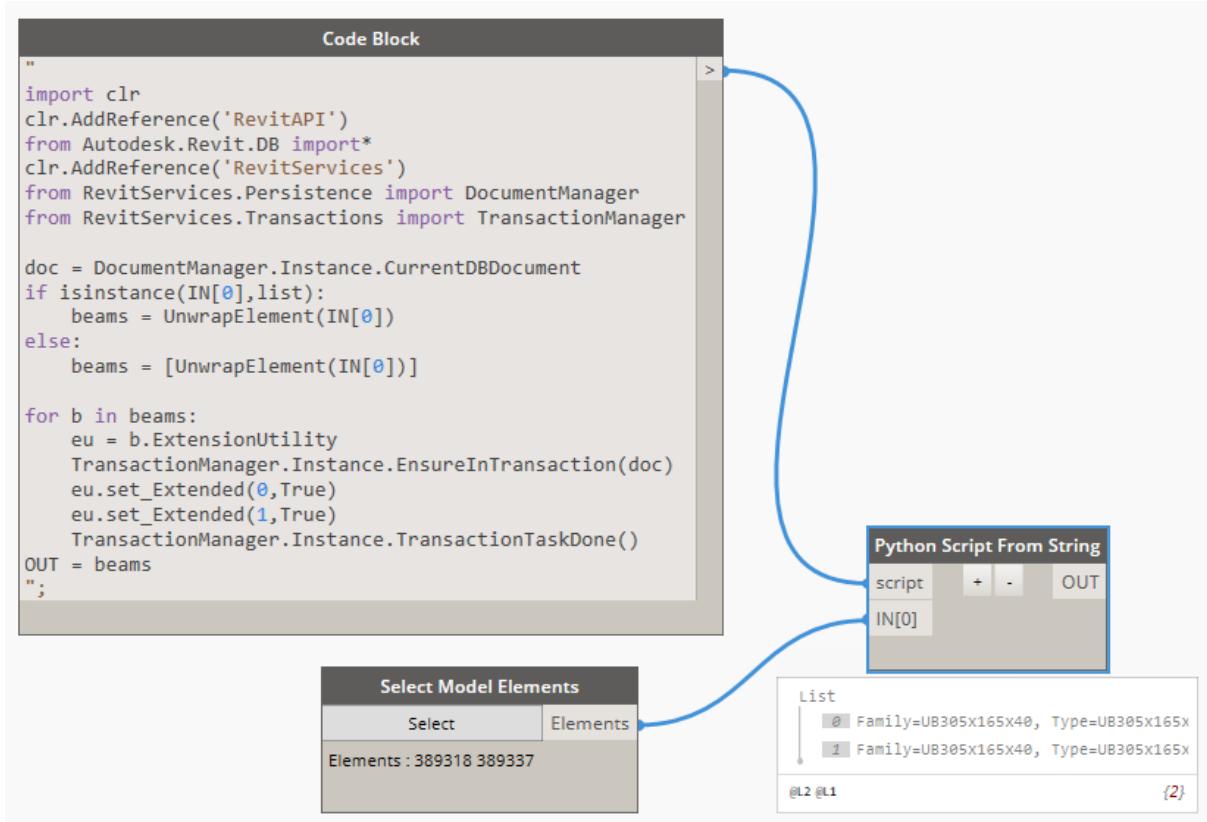
- Python là một ngôn ngữ lập trình hướng đối tượng, python hoàn toàn tạo kiểu động, cấp phát bộ nhớ động. Mục đích ra đời của Python là cung cấp một ngôn ngữ lập trình có cấu trúc rõ ràng, sáng sủa, thuận tiện cho người mới học lập trình. Python được phát triển bởi Guido và Rossum. Phiên bản đầu tiên được phát hành vào năm 1991. Python được lấy cảm hứng từ ABC, Haskell, Java, Lisp, Icon và Perl. Python là một ngôn ngữ thông dịch, đa nền tảng. Một trong những đặc điểm độc nhất của Python là ngôn ngữ này không dùng đến dấu chấm phẩy, dấu mở - đóng ngoặc {} để kết thúc câu lệnh hay khối lệnh, mà cách duy nhất để nó nhận biết một lệnh là dấu thụt đầu dòng.
- Hiện tại Python có hai dòng phiên bản là dòng 2.x và 3.x.
- Các thành phần của ngôn ngữ Python: Chú thích, biến, giá trị, toán tử, khối lệnh, kí hiệu, từ khóa.
- Các kiểu dữ liệu cơ bản của Python: Kiểu boolean, kiểu none, kiểu number, kiểu string, kiểu tuple, kiểu list, kiểu set, kiểu từ điển.
- Có hỗ trợ các module và packages dùng để quản lý code trong python.
- Phần này để hiểu sâu và chi tiết có thể truy cập website:
<https://www.python.org/>

II. Python trong Dynamo

- Trong Dynamo hỗ trợ lập trình Python thông qua 2 nút đó là Python Script và Python Script From String
 - Cú pháp như hình minh họa
- + Python Script (code biến number thành string và xóa số 0 phía sau).



+ Python Script From String (code join các góc của dầm)



- Giá trị đầu vào trong Python Script
Thông qua keyword **IN[0]**, **IN[1]** ... nó được tạo bằng cách nhập chuột vào dấu + trên nút Python Script.
- Revit API
Để tạo thuận lợi cho sự tương tác dễ dàng giữa RevitAPI và Dynamo, họ đã viết một thư viện toàn diện để tương tác với Revit, nghĩa là tất cả các kiểu dữ liệu truy cập giữa các nút Dynamo xoay quanh các gói kiểu dữ liệu cốt lõi, các gói này giúp Dynamo đồng bộ với Revit.

+ Document and Application

Revit Document/Application có thể được truy cập thông qua Dynamo bằng từ khóa DocumentManager.

```

import clr

# Import DocumentManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager

```

```
doc = DocumentManager.Instance.CurrentDBDocument
uiapp = DocumentManager.Instance.CurrentUIApplication
app = uiapp.Application
```

+ Elements

- Tất cả các Elements xuất phát từ Dynamo thực sự cũng chỉ xoay quanh gói Revit Elements. Bên trong Python Script bạn có thể thực hiện trực tiếp những kiểu này bằng cách gọi Revit.Elements

```
import clr

# Import RevitNodes
clr.AddReference("RevitNodes")
import Revit
# Import những kiểu bạn cần thay vì gọi từng kiểu
# bạn có thể gọi nó qua cú pháp 'from Revit.Elements import *'
from Revit.Elements import CurveByPoints, ReferencePoint

import System

startRefPt = IN[0]
endRefPt = IN[1]
refPtArray = System.Array[ReferencePoint]([startRefPt, endRefPt])
OUT = CurveByPoints.ByReferencePoints(refPtArray)
```

- Nếu muốn sử dụng Revit API trực tiếp bạn cần mở các gói Element trước khi cho nó hoạt động, sử dụng TransactionManager để chắc chắn rằng nó hoạt động bên trong Revit API Transaction và đóng gói bất kì Element nào bạn muốn import lại vào Revit.

```
import clr

# Import RevitAPI
clr.AddReference("RevitAPI")
import Autodesk
from Autodesk.Revit.DB import ReferencePointArray

# Import DocumentManager and TransactionManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

# Import ToDSType(bool) phương thức mở rộng
clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.Elements)

# Giải nén các gói Element đầu vào
startRefPt = UnwrapElement( IN[0] )
endRefPt = UnwrapElement( IN[1] )

# Bắt đầu chuyển đổi
doc = DocumentManager.Instance.CurrentDBDocument
```

```

TransactionManager.Instance.EnsureInTransaction(doc)

# Tạo CurveByPoints
arr = ReferencePointArray()
arr.Append(startRefPt)
arr.Append(endRefPt)
cbp = doc.FamilyCreate.NewCurveByPoints(arr)

# Kết thúc Transaction
TransactionManager.Instance.TransactionTaskDone()

# Đóng gói
OUT = cbp.ToDSType(false)
+ Unwrapping

wrappedElement = IN[0]
unwrappedElement = UnwrapElement( wrappedElement )
# Now I can use 'unwrappedElement' with the RevitAPI
+ Wrapping

import clr

# Import ToDSType(bool) extension method
clr.AddReference("RevitNodes")
import Revit
clr.ImportExtensions(Revit.Elements)

docPt = FetchRefPtFromDoc() #Fetch an existing ref pt (not a real method)
newPt = CreateNewRefPt()    #Create a new ref pt (not a real method)
OUT = [
    docPt.ToDSType(True), #Not created in script, mark as Revit-owned
    newPt.ToDSType(False) #Created in script, mark as non-Revit-owned
]

```

+ Units

- Dynamo sử dụng hệ đơn vị meter và Revit sử dụng đơn vị feet. Trong phiên bản hiện tại, Dynamo tự chuyển đổi đơn vị với các thông tin về hình học, còn trong khi sử dụng nút Python Script thi bắt buộc phải chuyển đổi giữa Dynamo và RevitAPI.

```

metersToFeet = 0.3048
feetToMeters = 1 / metersToFeet

# Chuyển đổi đơn vị từ Dynamo sang RevitAPI
dynamoUnitsLength = someDynamoLengthFunction()
revitUnitsAfterConvert = dynamoUnitsLength * metersToFeet

# Chuyển đổi đơn vị từ RevitAPI đến Dynamo
revitUnitsLength = someRevitLengthFunction()
dynamoUnitsLengthAfterConvert = revitUnitsLength * feetToMeters

```

- Phiên bản Dynamo 1.x không còn sử dụng đơn vị hệ m mà nó lấy trực tiếp từ RevitAPI, nếu chúng ta sử dụng hệ đơn vị ngoài m để đoạn code

không phức tạp Dynamo cũng cấp cú pháp chuyển đổi chính xác và khóa cứng lại bằng cách sử dụng phương thức RevitAPI **UnitUtils.ConvertFromInternalUnits()**

```
getDocUnits = doc.GetUnits()
getDisplayUnits =
getDocUnits.GetFormatOptions(UnitType.UT_Length).DisplayUnits
unitConversion = UnitUtils.ConvertFromInternalUnits(1, getDisplayUnits )
+ GeometryObjects
```

- Revit Geometry (khối, điểm, đường ...) là tất cả các loại đối tượng hình học (GeometryObject's). Tất cả các đối tượng hình học từ nút Dynamo không phải đối tượng hình học trong Revit (GeometryObject's) vì trong Dynamo sử dụng meter còn trong Revit sử dụng feet. Để chuyển đổi dễ dàng Dynamo cũng cấp cú pháp **GeometryConversion**.

Cách import GeometryConversion

```
import clr

clr.AddReference("RevitNodes")
import Revit

# Phương thức mở rộng
clr.ImportExtensions(Revit.GeometryConversion)
```

- Chuyển đổi đối tượng hình học trong revit sang đối tượng hình học trong hệ thống Dynamo

```
dynamoGeometry = revitGeometryObject.ToProtoType()
- Chuyển đổi tương đối tượng hình học của Dynamo sang Revit

revitGeometryObject = dynamoGeometry.ToRevitType()
- Revit XYZ không phải là GeometryObject's. Cách chuyển đổi Revit XYZ (qua vector hay point)
```

```
point = xyzToPoint()
vector = xyzToVector()
xyz = pointOrVector.ToXyz()
```

- Bạn có thể bỏ qua việc chuyển đổi đơn vị bằng cách để False ở các đối số ToRevitType()

ToProtoType() ToXyz(), ToPoint(), OR ToVector().

Ví dụ:

```
import clr

# Import RevitAPI
clr.AddReference("RevitAPI")
import Autodesk

clr.AddReference("RevitNodes")
import Revit
```

```

# Import DocumentManager and TransactionManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

# Import geometry conversion extension methods
clr.ImportExtensions(Revit.GeometryConversion)

# Import Element wrapper extension methods
clr.ImportExtensions(Revit.Elements)

# Unwrap the Point, yielding a Revit XYZ in Revit unit system
xyz = IN[0].ToXYZ()

# Start Transaction
doc = DocumentManager.Instance.CurrentDBDocument
TransactionManager.Instance.EnsureInTransaction(doc)

# Create a Reference Point
refPt = doc.FamilyCreate.NewReferencePoint(xyz)

# End Transaction
TransactionManager.Instance.TransactionTaskDone()

# Wrap ReferencePoint Element
OUT = refPt.ToDSType(false)

```

+ Transactions

- Nếu bạn viết code trong RevitAPI yêu cầu Transaction, thì trong Dynamo cũng yêu cầu đó bằng cú pháp TransactionManager
- Khởi tạo Dynamo Transaction:
`TransactionManager.EnsureInTransaction()`
- Kết thúc Transaction trong Dynamo:
`TransactionManager.TransactionTaskDone()`
- Transaction đang chuyển đổi trong Dynamo
`TransactionManager.ForceCloseTransaction()`
- Đóng quá trình chuyển đổi để làm việc `TransactionTaskDone()`
- Ví dụ:

```

import clr

# Import DocumentManager and TransactionManager
clr.AddReference("RevitServices")
import RevitServices
from RevitServices.Persistence import DocumentManager
from RevitServices.Transactions import TransactionManager

# Get the document
doc = DocumentManager.Instance.CurrentDBDocument

# "Start" the transaction

```

```
TransactionManager.Instance.EnsureInTransaction(doc)

# Create a reference point (requires a transaction)
refPt = doc.FamilyCreate.NewReferencePoint(XYZ(0, 0, 0))

# "End" the transaction
TransactionManager.Instance.TransactionTaskDone()
```