Mathutils:

1. Cách Import?

from mathutils import Vector, Matrix

* Tất cả các tọa độ đỉnh, ma trận các thứ của Object trong Blender đều có kiểu Vector hoặc Matrix của Mathutils

1. Vector?

* Tạo 1 Vector

<Vector> = Vector(<Tọa Độ>)

* Ví dụ

foo = Vector((1, 2, 3))

* Tính tích có hướng của <Vector 1> với <Vector 2>

<Tích Có Hướng> = <Vectorr 1>.cross(<Vector 2>)

* Chuẩn hóa <Vector>

<Vector>.normalize()

* Ví dụ

foo.normalize()

* Ta có

|  |  |
| --- | --- |
| foo trước | [2, 4, 9] |
| foo sau | [0.199, 0.398, 0.896] |

* Trả về độ dài <Vector>

<Độ Dài> = <Vector>.length

1. Ma Trận?

* Tạo 1 ma trận

<Ma Trận> = Matrix(<Mảng 2D>)

* Ví dụ

foo = Matrix([

[1, 2, 3],

[5, 6, 7]

])

1. Toán Tử @?

* Nhân 2 ma trận hoặc 1 ma trận với 1 Vector

Context – Ngữ Cảnh:

1. Cách Import?

from bpy import context

1. Trả Về List Các Object Đang Được Chọn?

<List Các Object> = context.selected\_objects

1. Trả Về Object Mà Bạn Đã Chọn Trong Lần Chọn Gần Nhất?

<Object> = context.active\_object

Operation – Toán Tử:

1. Cách Import?

from bpy.ops import object as ob

1. Cách Chọn Và Bỏ Chọn Tất Cả Object?

ob.select\_all(action = <Kiểu Chọn>)

* Các cơ chế chọn tương ứng

|  |  |
| --- | --- |
| <Kiểu Chọn> | Cơ chế |
| "SELECT" | Chọn tất cả Object trong Scene hiện tại |
| "DESELECT" | Bỏ chọn tất cả Object trong Scene hiện tại |
| "TOGGLE" | "DESELECT" khi có ít nhất 1 Object được chọn, nếu không thì "SELECT" |
| "INVERT" | Chọn tất cả Object không được chọn và bỏ chọn tất cả Object được chọn trong Scene hiện tại |

Data – Dữ Liệu:

1. Cách Import?

from bpy import data

1. Trả Về Object Có Tên Nào Đó?

<Object> = data.objects[<Tên Object>]

* Trả về kiểu của <Object>

<Kiểu Object> = <Object>.type

* Các giá trị

|  |  |
| --- | --- |
| <Kiểu Object> |  |
| "MESH" | Các Object có đỉnh cạnh mặt các thứ |
| "CAMERA" | Cái Camera |
| "LIGHT" | Cái bóng đèn |

Object:

1. Mesh?

* Trả về Mesh của 1 Object

<Mesh> = <Object>.data

* Trả về 1 Data Collection chứa tất cả đỉnh của <Mesh>

<Các Đỉnh> = <Mesh>.vertices

* Trả về Vector tọa độ của 1 đỉnh theo hệ qui chiếu Local

<Tọa Độ Đỉnh Local> = <Đỉnh>.co

* Trả về trọng số Bevel của 1 đỉnh

<Trọng Số Bevel Đỉnh> = <Đỉnh>.bevel\_weight

* Trả về ma trận thế giới của Object, ma trận này có tác dụng chuyển vật từ hệ quy chiều Local sang Global

<Ma Trận Thế Giới> = <Object>.matrix\_world

* Dùng ma trận này để chuyển đổi hệ quy chiếu Local thành Global

<Tọa Độ Đỉnh Global> = <Ma Trận Thế Giới> @ <Tọa Độ Đỉnh Local>

1. Các Thuộc Tính Hay Dùng?

* Để trả về Vector có giá trị là độ phóng to thu nhỏ của Object theo 3 phương hoành, tung, cao ứng với hệ trục Local

<Scale Vector> = <Object>.scale

* Để trả về Vector tọa độ tâm của Object

<Location Vector> = <Object>.location

Blender Mesh:

1. Cách Import?

import bmesh

1. Cách Hoạt Động?

* Copy 1 Mesh vào 1 Blender Mesh rồi chỉnh sửa rồi áp cái chỉnh sửa vào Mesh đã Copy
* Để tạo 1 Blender Mesh rỗng

<Blender Mesh> = bmesh.new()

* Để Copy 1 Mesh vào Blender Mesh

<Blender Mesh>.from\_mesh(<Mesh>)

* Để áp Blender Mesh vào lại Mesh

<Blender Mesh>.to\_mesh(<Mesh>)

* Để giải phóng Blender Mesh khỏi bộ nhớ

<Blender Mesh>.free()

1. Đỉnh Trong Blender Mesh?

* Trả về 1 Data Sequence chứa tất cả các đỉnh của <Blender Mesh>

<Các Đỉnh> = <Blender Mesh>.verts

* Để thêm 1 đỉnh
* <Các Đỉnh>.new(<>)
* Khi bạn chỉnh sửa <Blender Mesh> như thêm đỉnh, mặt, xóa cạnh, … thì Index của những thứ này không được cập nhật, và bạn sẽ không thể truy cập 1 đỉnh, mặt, … thông qua Index của nó, để cập nhật lại Index của chúng sau khi chỉnh sửa, lí do không cập nhật mỗi khi chỉnh sửa là để giảm việc tính toán quá nhiều

<Các Đỉnh Hoặc Các Mặt Hoặc Các Cạnh>.ensure\_lookup\_table()

* Trả về Vector tọa độ của 1 đỉnh theo hệ qui chiếu Local

<Tọa Độ Đỉnh Local> = <Đỉnh>.co

Addon – Tiện Ích:

1. Viết 1 File Python Tiện Ích?

* Để 1 File Python trở thành 1 File Python tiện ích, bạn cần khai báo biến toàn cục sau

bl\_info = <Dictionary Chứa Thông Tin>

* <Dictionary Chứa Thông Tin> chỉ được phép có các Key sau

|  |  |
| --- | --- |
| Key |  |
| "category" và "name" | Giá trị String của 2 Key này sẽ là tiêu đề của tiện ích trong danh sách tiện ích  Nếu không chỉ định giá trị cho "category" thì nó mặc định = ""  Nếu không chỉ định giá trị cho "name" thì nó mặc định = tên File Python tiện ích  Cú pháp tiêu đề tiện ích là  <Category Value>:<Name Value> |
| "version" | Key này nếu được chỉ định, thì nó sẽ được hiển ở phần mô tả tiện ích  Nếu giá trị của Key này là String, thì giữa mỗi cặp kí tự liền kề trong nó sẽ thêm 1 dấu chấm  Nếu giá trị Key này là 1 List, Tuple, …, thì chúng sẽ được nối lại thành String và thêm 1 dấu chấm ngăn cách  Cú pháp trong mô tả tiện ích là  Version: <Version Value> |
| "description", "location", "author" | Các Key này nếu được chỉ định, thì nó sẽ được hiển ở phần mô tả tiện ích  Giá trị của các Key này phải là String  Cú pháp trong mô tả tiện ích là  <Tên Key>: <Value Tương Ứng> |

* Ví dụ

bl\_info = {

"name": "Suc Dau Cu",

"author": "Nguyen Van Duc",

"version": (2, 3),

"blender": (3, 5, 0),

"location": "Cho Nao Day Tao Quen Roi",

"description": "Dung Xai Addon Nay",

"category": "Addon Rac",

}

* Đồng thời khi bạn kích hoạt 1 tiện ích trong danh sách tiện ích, hàm register trong File Python tiện ích tương ứng sẽ được chạy, và khi bạn bất hoạt 1 tiện ích trong danh sách tiện ích, hàm unregister trong File Python tiện ích tương ứng sẽ được chạy
* Ví dụ

def register():

print("Chào thằng ngu")

def unregister():

print("Tạm biệt thằng ngu")

1. Panel?

* Tất cả Panel trong Blender đều kế thừa, hay là Class con của bpy.types.Panel
* Mặc định có khoảng 1165 Panel
* Mỗi Panel 1 khi đã được Register thì sẽ được thêm vào danh sách Registered Panel, và bạn không thể xóa nó khỏi danh sách, và nó sẽ ở trong File Blender vĩnh viễn, tuy nhiên bạn có thể Unregister nó, khi này nó vẫn nằm trong danh sách, chỉ là không được hiển thị ra ngoài thôi, bạn có thể Register lại lúc nào cũng được
* Để trả về danh sách Registered Panel dưới dạng 1 List chứa các Panel

<Danh Sách Registered Panel> = bpy.types.Panel.\_\_subclass\_\_()

* Để trả về 1 Panel

<Panel> = <Danh Sách Registered Panel>[<Index>]

* Để Register 1 Panel, Panel nào đã Register rồi thì báo lỗi

bpy.utils.register\_class(<Panel Muốn Register>)

* Để Unregister 1 Panel, Panel nào đã Unregister rồi thì báo lỗi

bpy.utils.unregister\_class(<Panel Muốn Unregister>)

* Để tạo 1 Panel

class <Panel>(bpy.types.Panel):

<Danh Sách Các Thuộc Tính>

def draw(<Self>, <Context>):

<Thiết Kế Panel>

* <Self> và <Context> bạn có thể đặt tên tùy ý, <Context> hoạt động y chang bpy.context
* Mới khai báo Class như này thì <Panel> chưa được vào <Danh Sách Registered Panel>, bạn cần Register nó như đã nói

|  |  |
| --- | --- |
| <Danh Sách Các Thuộc Tính> |  |
| bl\_label | Giá trị String tiêu đề của Panel  Bắt buộc phải chỉ định |
| bl\_space\_type | Giá trị String là tên của cửa sổ con Panel này thuộc về  Bắt buộc phải chỉ định |
| bl\_region\_type | Giá trị String là tên của vùng chứa Panel trong cửa sổ con  Bắt buộc phải chỉ định |
| bl\_category | Giá trị String là tên Tab chứa Panel, nếu Tab chưa tồn tại thì tạo Tab mới  Nếu không chỉ định thì mặc định nó là "Misc" |
| bl\_idname | Giá trị String là Panel ID  Không chỉ định cũng được |
| bl\_parent\_id | Giá trị String là Panel ID của Panel chứa nó, nghĩa là đây sẽ là Panel con của 1 Panel  Không chỉ định cũng được |
| bl\_options | Là 1 Set chứa các phần tử là String, mỗi String sẽ có 1 hiệu ứng khác nhau  Không chỉ định cũng được |

* Bảng giá trị cho bl\_space\_type và bl\_region\_type

|  |  |  |  |
| --- | --- | --- | --- |
| bl\_space\_type | Cửa sổ con | bl\_region\_type | Vùng chứa |
| "VIEW\_3D" | 3D Viewport | "UI" | Viewport Panel |
| "PROPERTIES" | Properties |  |  |

* Bảng hiệu ứng cho bl\_options

|  |  |
| --- | --- |
| "DEFAULT\_CLOSED" | Mặc định khi mới Register, Panel sẽ thu gọn lại, Click nút tam giác trên tiêu đề Panel sẽ mở ra |

* Trong phần <Thiết Kế Panel>
* Tạo 1 hàng mới trong Panel và trả về hàng này

<Hàng> = <Self>.layout.row()

* Để thêm 1 nút bấm vào 1 hàng, khi bấm vào thì 1 Operation sẽ được chạy

<Hàng>.operator(<Operation ID>, text = <String Trên Nút Bấm>, icon = <Tên Biểu Tượng>)

* Nếu <String Trên Nút Bấm> không được chỉ định, thì mặc định nó là nhãn của Operation
* Biểu tượng có tên là <Tên Biểu Tượng> sẽ nằm bên trái <String Trên Nút Bấm>
* Để thêm 1 biểu tượng + 1 dòng chữ bên cạnh biểu tượng vào 1 hàng

<Hàng>.label(icon = <Tên Biểu Tượng>, text = <Dòng Chữ>)

* Để thêm các ô Input liết kết với 1 thuộc tính của Object nào đó vào 1 hàng

<Hàng>.prop(<Object>, <Tên Thuộc Tính>)

* Ví dụ

foo.prop(context.active\_object, "scale")

* Khi này trong hàng foo sẽ có thêm 3 ô Input ứng với độ phóng to thu nhỏ của Object đang Active theo 3 phương
* Để thêm 1 cột mới trong Panel và trả về cột này

<Cột> = <Self>.layout.column()

* <Cột> y chang hàng, chỉ có điều các phần tử khác nhau như dòng chữ, dãy ô Input,… hiển thị từ trái qua phải thì bây giờ ở cột nó hiển thị từ trên xuống dưới
* Để tạo nhiều cột cạnh nhau như 1 bảng

<Cột Split> = <Cột>.split()

* Từ đây ta có thể sử dụng <Cột Split> thay cho <Cột>, khi thêm 1 phần tử khác vào

<Cột Split>, thì phần tử đó sẽ vẫn nằm dọc, nhưng đặt cạnh bên phải phần tử trước đó

1. Operation?

* Tất cả Operation trong Blender đều kế thừa, hay là Class con của bpy.types.Operator
* Mặc định có khoảng 214 Operation
* Mỗi Operation 1 khi đã được Register thì sẽ được thêm vào danh sách Registered Operation, và bạn không thể xóa nó khỏi danh sách, và nó sẽ ở trong File Blender vĩnh viễn, tuy nhiên bạn có thể Unregister nó, khi này nó vẫn nằm trong danh sách, chỉ là không dùng được thôi, bạn có thể Register lại lúc nào cũng được
* Để trả về danh sách Registered Operation dưới dạng 1 List chứa các Operation

<Danh Sách Registered Operation> = bpy.types.Operator.\_\_subclass\_\_()

* Để trả về 1 Operation

<Operation> = <Danh Sách Registered Operation>[<Index>]

* Để Register 1 Operation, Operation nào đã Register rồi thì báo lỗi

bpy.utils.register\_class(<Operation Muốn Register>)

* Để Unregister 1 Operation, Operation nào đã Unregister rồi thì báo lỗi

bpy.utils.unregister\_class(<Operation Muốn Unregister>)

* Operation có thể đặt chung với Panel để Register hoặc Unregister một thể
* Để tạo 1 Operation

class <Operation>(bpy.types.Operator):

<Danh Sách Các Thuộc Tính>

<Danh Sách Các Thông Số>

def \_\_init\_\_(<Self 5>):

<Khởi Tạo>

def \_\_del\_\_(<Self 6>):

<Sau Khi Hoàn Thành>

@classmethod

def poll(<Class Này>, <Context 1>):

<Có Thực Hiện Operation Không>

def execute(<Self 1>, <Context 2>):

<Thiết Kế Chức Năng Của Operation>

def invoke(<Self 2>, <Context 3>, <Event 1>):

<Xử Lí Đầu Tiên>

def draw(<Self 3>, <Context 4>):

<Thiết Kế Pop Up Panel>

def modal(<Self 4>, <Context 5>, <Event 2>:

<Lắng Nghe Sự Kiện>

* <Class Này>, <Self 1 2 3 4 5 6>, <Context 1 2 3 4 5> và <Event 1 2> bạn có thể đặt tên tùy ý
* Mới khai báo Class như này thì <Operation> chưa được vào <Danh Sách Registered Operation>, bạn cần Register nó như đã nói
* Tất cả các Operation đã Register đều có thể được chạy bằng lệnh sau

bpy.ops.<Operation ID>(<Danh Sách Tham Số>)

* Ta có

|  |  |
| --- | --- |
| <Danh Sách Các Thuộc Tính> |  |
| bl\_idname | Giá trị String là <Operation ID>  Phải chứa đúng 1 dấu chấm  Bắt buộc phải chỉ định |
| bl\_label | Giá trị String là nhãn của Operation  Bắt buộc phải chỉ định |

* Cơ chế hoạt động như sau
* Khi <Operation> được gọi thông qua bpy.ops, trước nhất nó sẽ gọi hàm poll, và Pass chính Class <Operation> và bpy.context cho hàm này, nếu kết quả trả về là False, thì mọi thứ dừng lại và <Operation> không được thực hiện, nếu <Operation> là 1 nút bấm trong 1 Panel, thì nó sẽ bị bất hoạt, ngược lại nếu kết quả là True, ta sẽ thực hiện tiếp các bước tiếp theo, trong trường hợp không chỉ định hàm poll thì ta cũng sẽ thực hiện các bước tiếp theo
* Hàm \_\_init\_\_ sẽ là hàm tiếp theo được chạy
* Tiếp theo, sẽ là 1 loạt các hàm được chạy, bắt đầu từ hàm invoke, nếu không có invoke thì bắt đầu từ execute, nếu không có execute nữa thì lỗi, ta có quy luật sau
* Khi đang chạy mà gặp lệnh return {"FINISHED"}, thì dừng lại, và áp dụng những gì <Operation> đã thay đổi
* Nếu gặp lệnh return {"CANCELLED"}, thì cũng dừng lại, nhưng hủy bỏ những gì <Operation> đã thay đổi, giữ nguyên hiện trạng như trước khi <Operation> xảy ra
* Nếu gặp lệnh return {"RUNNING\_MODAL"}, thì ngay lập tức chuyển sang chạy hàm modal, nên bạn có thể chạy hàm modal nhiều lần
* Khi 1 trong các hàm invoke, modal, execute được chạy, thì trong chúng bắt buộc phải có lệnh return, và return 1 trong 3 cái đã nói ở trên
* Ngoài 3 kiểu trên, ta có thể return bằng cách gọi 1 hàm khác, ví dụ

return self.execute(context)

* Sau khi <Operation> kết thúc, hàm \_\_del\_\_ sẽ được chạy
* Trong các hàm của <Operation>, bạn có thể sử dụng lệnh sau để Log ra thông tin ở cửa sổ con Info

<Self>.report({<Loại Thông Tin>}, <Tin Nhắn>)

* Ta có

|  |  |
| --- | --- |
| <Loại Thông Tin> |  |
| "INFO" | Loại thông tin thuần văn bản |
| "ERROR" | Loại thông tin lỗi |

* Bạn có thể sử dụng <Event> để biết được trạng thái con trỏ chuột và bàn phím ngay lúc hàm có tham số là <Event> được chạy

|  |  |
| --- | --- |
| <Event>.mouse\_x | khoảng cách đo bằng Pixel từ con trỏ đến cạnh trái màn hình, là 1 số nguyên |
| <Event>.mouse\_y | khoảng cách đo bằng Pixel từ con trỏ đến cạnh trên màn hình, là 1 số nguyên |
| <Event>.type | Hành động của chuột, có thể là 1 trong các giá trị sau  "LEFTMOUSE", bạn đã Click chuột trái  "RIGHTMOUSE", bạn đã Click chuột phải  "MOUSEMOVE", bạn đã di chuột  "ESC", bạn đã nhấn phím “ESC” |

* Để chỉ định <Danh Sách Các Thông Số>, ta sử dụng cú pháp

<Thông Số>: bpy.props.<Kiểu Dữ Liệu>(default = <Giá Trị Mặc Định>)

* Ta có

|  |  |
| --- | --- |
| <Kiểu Dữ Liệu> |  |
| IntProperty | Số nguyên |
| FloatProperty | Số thập phân |
| StringProperty | Chuỗi |

* Nếu không chỉ định <Giá Trị Mặc Định> thì nó tự động = 0 nếu là kiểu số
* Ví dụ

foo: bpy.props.IntProperty(default = 10)

* Bạn có thể sử dụng lệnh sau trong các hàm của <Operation> để mở cửa sổ mở File, điều đặc biệt ở cửa sổ này là chữ “Open” sẽ thay bằng nhãn của <Operation>, các lệnh tiếp theo vẫn sẽ được chạy, nhưng nếu gặp lệnh return {"RUNNING\_MODAL"}, thì sẽ tạm dừng cho đến khi bạn đóng cửa sổ mở File hoặc mở thành công 1 File, nếu bạn đóng cửa sổ mở File và không mở File nào, thì <Operation> sẽ kết thúc ngay, nếu bạn có mở File, thì đường dẫn tuyệt đối tới File này có phần mở rộng sẽ được lưu trong thông số filepath ở

<Danh Sách Các Thông Số>, nên bạn cần khai báo thông số này, đồng thời hàm execute mới là hàm sẽ được chạy tiếp theo, thay vì hàm modal

<Context>.window\_manager.fileselect\_add(<Self>)

* Ngoài ra, bạn cũng có thể sử dụng lệnh sau trong các hàm của <Operation> để mở Pop Up Panel

<Context>.window\_manager.invoke\_props\_dialog(<Self>)

* Lệnh trên cũng sẽ cần return {"RUNNING\_MODAL"} để tạm dừng, hoặc bạn return thẳng câu lệnh luôn cũng sẽ tạm dừng

return <Context>.window\_manager.invoke\_props\_dialog(<Self>)

* Sau khi đã nhập giá trị cho các thông số ở Pop Up Panel và nhấn “OK”, các thông số sẽ cập nhật và hàm execute sẽ được chạy tiếp theo
* Cấu trúc Pop Up Panel mặc định
* Thanh trên cùng là thanh tiêu đề, ghi nhãn của <Operation>
* Thanh dưới cùng là thanh xác nhận, ghi chữ “OK”
* Phần ở giữa và khu vực nhập giá trị cho các thông số trong <Danh Sách Các Thông Số>, trước khi nhập thì nó cũng sẽ hiển thị giá trị mặc định của các thông số
* Bạn có thể thiết kế lại Pop Up Panel trong phần <Thiết Kế Pop Up Panel>, nếu không có phần này thì nó dùng thiết kế mặc định như trên, cách thiết kế y chang như thiết kế Panel bình thường
* Ví dụ, để tạo 1 hàng có ô Input dành cho thông số foo

row = self.layout.row()

row.prop(self, "foo")

* Hơn nữa, bạn cũng có thể sử dụng lệnh sau trong các hàm của <Operation> để mở Search Panel

<Context>.window\_manager.invoke\_search\_popup (<Self>)

* Lệnh trên cũng sẽ cần return {"RUNNING\_MODAL"} để tạm dừng
* Sau khi đã chọn 1 trong các giá trị được liệt kê ra, thông số my\_search trong <Danh Sách Các Thông Số> sẽ cập nhật và hàm execute sẽ được chạy tiếp theo
* Cấu trúc Search Panel
* Thanh trên cùng là thanh Filter
* Phần bên dưới là danh sách các tùy chọn bạn có thể chọn, danh sách này được ghi trong thông số my\_search như sau

my\_search: bpy.props.EnumProperty(items = (

(<String Giá Trị Thật 1>, <Tùy chọn 1 Hiện Ở Search Panel>, ""),

(<String Giá Trị Thật 2>, <Tùy chọn 2 Hiện Ở Search Panel>, ""),

(<String Giá Trị Thật 3>, <Tùy chọn 3 Hiện Ở Search Panel>, ""),

…

))

* Lưu ý danh sách của my\_search chỉ hiển thị ở Search Panel khi bạn chỉ định thêm thuộc tính bl\_property = "my\_search", đồng thời giá trị mặc định cho my\_search sẽ là <String Giá Trị Thật 1>
* Ví dụ

bl\_property = "my\_search"

my\_search: bpy.props.EnumProperty(items = (

("Con Cac", "daucac", ""),

("Con Cu", "daucu", ""),

("Con Buom", "daubuom", "")

))

Quick Code – Mã Nhanh:

1. Tích Tổng Moment Động Lượng?

from bpy import context

from mathutils import Vector, Matrix

from random import random

[obj] = context.selected\_objects

vertex\_masses = []

total\_mass = 0

center = Vector((0, 0, 0))

for vertex in obj.data.vertices:

vertex\_mass = random()

total\_mass += vertex\_mass

vertex\_masses.append(vertex\_mass)

center += vertex\_mass \* vertex.co

center = center / total\_mass + Vector(<Offset>)

omega\_direction = <Vector Vận Tốc Góc>

omega = Vector(omega\_direction)

normalized\_omega = Vector(omega\_direction)

normalized\_omega.normalize()

angular\_momentum = Vector((0, 0, 0))

distances = []

for idx, vertex in enumerate(obj.data.vertices):

r = vertex.co - center

d = r - normalized\_omega \* (r.dot(normalized\_omega))

m = vertex\_masses[idx]

angular\_momentum += m \* r.cross(omega.cross(d))

distances.append(d.length)

axis\_angular\_momentum = angular\_momentum.dot(normalized\_omega) \* normalized\_omega

print("Method 1 :")

print("Raw Angular Momentum : ", angular\_momentum)

print("Axis Angular Momentum : ", axis\_angular\_momentum)

moment\_of\_inertia = Matrix([

[0, 0, 0],

[0, 0, 0],

[0, 0, 0]

])

axis\_moment\_of\_inertia = 0

for idx, vertex in enumerate(obj.data.vertices):

x, y, z = vertex.co - center

m = vertex\_masses[idx]

moment\_of\_inertia += m \* Matrix([

[y\*\*2 + z\*\*2, -x\*y, -x\*z],

[-x\*y, x\*\*2 + z\*\*2, -y\*z],

[-x\*z, -y\*z, x\*\*2 + y\*\*2]

])

axis\_moment\_of\_inertia += m \* distances[idx]\*\*2

angular\_momentum = moment\_of\_inertia @ omega

axis\_angular\_momentum = axis\_moment\_of\_inertia \* omega

print("")

print("Method 2 : ")

print("Raw Angular Momentum : ", angular\_momentum)

print("Axis Angular Momentum : ", axis\_angular\_momentum)

print(moment\_of\_inertia)

* <Offset>, <Vector Vận Tốc Góc> là các Tuple gồm 3 phần tử là tọa độ
* Đoạn Code trên sẽ tính trọng điểm của Object đang chọn = cách coi nó là hệ chất điểm với các đỉnh là chất điểm với khối lượng ngẫu nhiên, lưu ý hệ quy chiếu Local, sau đó tìm mốc để tính Moment động lượng = trọng tâm + <Offset>
* Giả định rằng tất cả các đỉnh của Object đang quay quanh mốc với trục có hướng của <Vector Vận Tốc Góc>
* Đoạn Code trên cũng sẽ tính Moment động lượng = phương pháp nâng cao là sử dụng Moment quán tính, với mốc trùng với khi tính Moment động lượng và cũng in ra Moment quán tính
* Đoạn Code trên cũng sẽ trả về Moment động lượng theo trục quay = 2 phương pháp

1. Dùng Ma Trận Biến Đổi Hình Dạng Của 1 Object?

import bpy

from mathutils import Matrix

ob = bpy.context.object

me = ob.data

M = Matrix(<Ma Trận 4 x 4>)

me.transform(M)

me.update()

* <Ma Trận 4 x 4> là ma trận tích hợp quay và tịnh tiến bạn muốn dùng để biến đổi hình dạng Object
* Ví dụ

M = Matrix((

(1, 0, 0, 1),

(0, 1, 0, 0),

(0, 0, 1, 0),

(0, 0, 0, 1),

))

1. Plot Đồ Thị 3D Của Hàm Bất Kì?

* Đoạn code này xóa tất cả vật thể hiện tại, rồi tạo Object mới là đồ thị

import bpy

from math import \*

bpy.ops.object.select\_all(action='DESELECT')

bpy.ops.object.select\_by\_type(type='MESH')

bpy.ops.object.delete()

mesh = bpy.data.meshes.new("graph")

obj = bpy.data.objects.new("GraphObj", mesh)

scene = bpy.context.scene

scene.objects.link(obj)

vertices = []

edges = []

faces = []

num\_points = <Độ Phân Giải>

lim\_x = <Giới Hạn Giá Trị Của X>

lim\_y = <Giới Hạn Giá Trị Của Y>

x\_step = (lim\_x[1] - lim\_x[0]) / num\_points

y\_step = (lim\_y[1] - lim\_y[0]) / num\_points

for i in range(num\_points):

x = lim\_x[0] + i \* x\_step

for j in range(num\_points):

y = lim\_y[0] + j \* y\_step

z = <Biểu Thức Hàm>

vertices.append((x, y, z))

for i in range(num\_points - 1):

for j in range(num\_points - 1):

v0 = i \* num\_points + j

v1 = v0 + 1

v2 = v0 + num\_points

v3 = v2 + 1

edges.extend([(v0, v1), (v1, v3), (v3, v2), (v2, v0)])

faces.append((v0, v1, v3, v2))

mesh.from\_pydata(vertices, edges, faces)

mesh.update()

* <Giới Hạn Giá Trị Của X> và <Giới Hạn Giá Trị Của Y> là List bao gồm 2 giá trị cận có tác dụng giới hạn hiển thị hàm số
* Ví dụ

lim\_x = [-5, 5]

lim\_y = [-10, 10]

* <Độ Phân Giải> càng lớn thì đồ thị càng mượt
* <Biểu Thức Hàm> là hàm bạn muốn Plot
* Ví dụ hàm của bạn là z = 2x + 4y2

z = 2 \* x + 4 \* y \*\* 2

1. Plot Đường Cong?

* Đoạn Code này sẽ xóa tất cả Object hiện tại sau đó tạo đường cong có dạng

z = f(x, y), trong đó y = g(x)

import bpy

import numpy as np

from math import pow

def create\_curve():

curve\_data = bpy.data.curves.new(name='My Curve', type='CURVE')

curve\_data.dimensions = '3D'

spline = curve\_data.splines.new(type='POLY')

x\_values = np.linspace(<Cận Trái>, <Cận Phải>, <Độ Phân Giải>).tolist()

y\_values = [<Biểu Thức Y Theo X> for x in x\_values]

z\_values = [<Biểu Thức Đường Cong> for x, y in zip(x\_values, y\_values)]

spline.points.add(len(x\_values) - 1)

for i, (x, y, z) in enumerate(zip(x\_values, y\_values, z\_values)):

spline.points[i].co = (x, y, z, 1)

obj = bpy.data.objects.new('My Object', curve\_data)

scene = bpy.context.scene

scene.objects.link(obj)

obj.select = True

scene.objects.active = obj

bpy.ops.object.select\_all(action='DESELECT')

bpy.ops.object.select\_by\_type(type='CURVE')

bpy.ops.object.delete()

create\_curve()

* <Cận Trái> và <Cận Phải> là giới hạn hiển thị đường cong theo x
* <Độ Phân Giải> càng cao thì đường cong càng mượt
* <Biểu Thức Đường Cong> là f(x, y)
* Ví dụ z = f(x, y) = 2x + 4y2

2 \* x + 4 \* y \*\* 2

* <Biểu Thức Y Theo X> là g(x)
* Ví dụ y = g(x) = 5x

5 \* x

1. Render Từng Khoảng Frame Rời Rạc?

import bpy

FRAME\_RANGES = [<Các Khoảng Frame>]

OUTPUT\_PATH = <Đường Dẫn Tới Thư Mục Lưu Ảnh>

for frame\_range in FRAME\_RANGES:

start\_frame, end\_frame = frame\_range

for frame\_number in range(start\_frame, end\_frame + 1):

bpy.context.scene.frame\_set(frame\_number)

bpy.context.scene.render.filepath = OUTPUT\_PATH + str(frame\_number)

bpy.ops.render.render(write\_still=True)

* <Các Frame Range> là các Tuple chứa 2 phần tử lần lượt là Start Frame và End Frame ứng với mỗi khoảng Frame
* Ví dụ

[(0, 10), (20, 30), (50, 70)]

* Chạy đoạn Code trên sẽ Render Scene hiện tại dưới nền theo cài đặt ở giao diện người dùng

1. Xóa Toàn Bộ Key Frame Liên Quan Đến Chuyển Động Của Tất Cả Object Được Chọn?

import bpy

SELECTED\_OBJECTS = bpy.context.selected\_objects

for obj in SELECTED\_OBJECTS:

obj.animation\_data\_clear()

1. Xuất Tọa Độ Của Tất Cả Các Đỉnh Trong Object Được Chọn Sang File CSV?

import bpy

CSV\_FILE = <Đường Dẫn Tuyệt Đối Tới File CSV Có Phần Mở Rộng>

VERTICES = [

bpy.context.object.matrix\_world @ x.co

for x in bpy.context.object.data.vertices

]

f = open(CSV\_FILE, 'w')

f.writelines([";".join([str(x) for x in coord]) + "\n" for coord in VERTICES])

f.close()

* Trong File CSV, mỗi dòng sẽ là 1 bộ 3 hoành, tung, cao độ ngăn cách nhau bằng dấu chấm phẩy, ví dụ

1.0;1.0;-1.0

1.0;-1.0;1.0

1.0;-1.0;-1.0