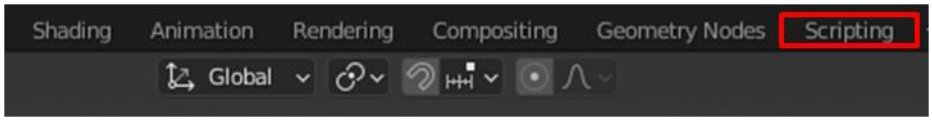
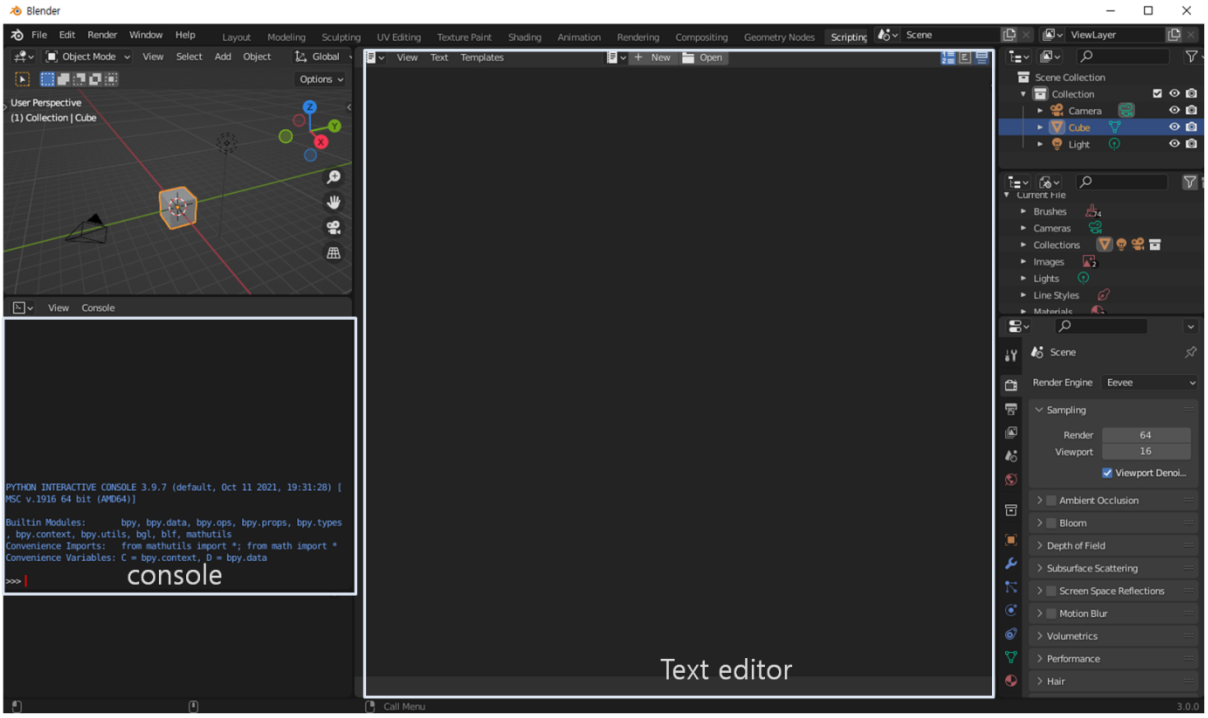
블렌더에서 파이선 실행하기

상단메뉴애서 Scripting 클릭

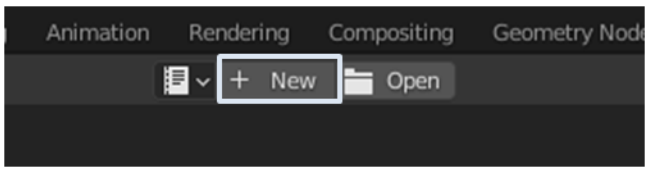


레이아웃이 아래와 같이 변경

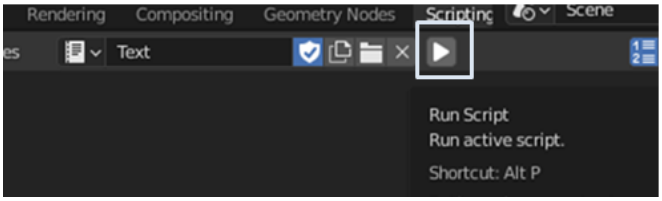


파이썬 콘솔에는 스크립트의 실행 로그를 확인
그 아래 인포편집기에는 실행된 기능의 명령문이 출력
텍스트 에디터는 스크립트를 입력하는 화면이다.

우선 스크립트 입력을 위해 New 버튼을 누른다



스크립트 실행은 오른쪽 화살표를 클릭 혹은 Alt + P



블렌더의 기능들을 사용하기 위해서는 종속된 클래스에 대해 알아야 한다.

```
>>> list(bpy.data.objects)
[bpy.data.objects['Camera'], bpy.data.objects['Cub
e'], bpy.data.objects['Light']]
```

기본으로 생성된 큐브의 해당 클래스를 호출해 보면 3 가지 객체에 대한 리스트 정보를 얻을 수 있다.
카메라, 큐브, 라이트에 대한 객체 정보이다.
리스트 카메라는 0 번, 큐브 1 번, 라이트 2 번의 인덱스 정보를 가진다.

```
>>> bpy.data . objects [ 'Cube' ] . location
Vector((0.0, 0.0, 0.0))

>>> bpy.data . objects [1] . location
Vector((0.0, 0.0, 0.0))

>>> bpy.data . objects [1] . location. x
0.0
```

객체의 이름으로 찾거나 인덱스 번호로 위치를 찾을 수 있고

```
>>> bpy.data.objects[1].location.x = 2
>>> bpy.data.objects[1].location.y = -1
>>> bpy.data.objects[1].location = (1, 2, 1)
```

큐브 객체의 위치 x, y, z 값을 변경해 주어서 큐브의 위치를 바꿀 수 있다.

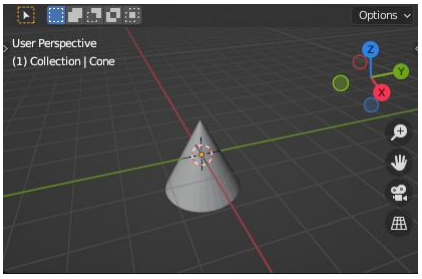
블렌더 오브젝트 만드는법

큐브 생성

```
import bpy

bpy.ops.mesh.primitive_cube_add(size=2, enter_editmode=False,
location=(0,0,0))
bpy.ops.object.delete(use_global=False)
```

원뿔 생성

```
import bpy


bpy.ops.mesh.primitive_cone_add(radius1=1, radius2=0, depth=2,
enter_editmode=False, align='WORLD', location=(0, 0, 0), scale=(1, 1, 1))
```



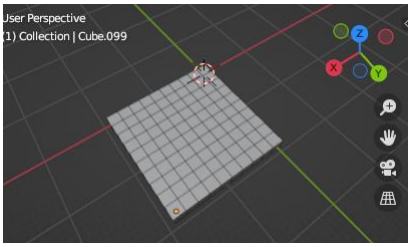
import bpy 는 블렌더를 접근하기 위한 블렌더 모듈을 불러오는 명령어
bpy 는 블렌더 파이썬에 관련된 총칭이고
ops 는 오퍼레이터의 약어이다.
그 중에서도 bpy.ops.mesh 는 메쉬 오브젝트에 대한 유틸리티 같은 것

primitive_cube_add 는 큐브를 추가하는 명령어
size 는 크기를 의미하고
editmode 는 생성한 후 바로 편집모드로 진입하는 것을 의미
location 은 공간상의 x, y, z 좌표를 지정

primitive_cone_add 는 원뿔의 원형을 추가하는 명령어
radius1 파라미터는 원뿔의 반지름 (원뿔의 반지름이 2 개가 있다.)
depth 는 높이를 의미하고
align 은 월드좌표를 기준으로 할지 로컬좌표를 기준으로 할지 결정

for문 사용

```
import bpy


bpy.ops.object.select_all(action="SELECT")
bpy.ops.object.delete()
for x in range(0, 10):
        for y in range(0, 10):
                bpy.ops.mesh.primitive_cube_add(location=(x * 2, y * 2, 0))
```



bpy.ops.object.select_all(action="SELECT")는 모든 오브젝트를 선택하는 명령어
bpy.ops.object.delete() 선택된 오브젝트를 삭제하는 명령어

큐브를 생성해보기 위해서 bpy.ops.mesh.primitive_cube_add() 실행

```
PYTHON INTERACTIVE CONSOLE 3.9.7 (default, Oct 11 2021, 19:31:28) [M
SC v.1916 64 bit (AMD64)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types,
 bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> bpy.ops.mesh.primitive_cube_add()
{'FINISHED'}

>>>
```

```
bpy.ops.mesh.primitive_cube_add(enter_editmode=False,
align='WORLD', location=(0, 0, 0), scale=(1, 1, 1))
```

Set Selection Pan View

bpy.ops.mesh.primitive_cube_add() 인수값이 없는데
아래 인포편집기에서 조금 전 실행한 명령문에는 괄호 안이 비어있지 않은걸 볼 수 있다.
기본값이라는 기능이 있기 때문

오브젝트의 이동, 회전, 확대

이동

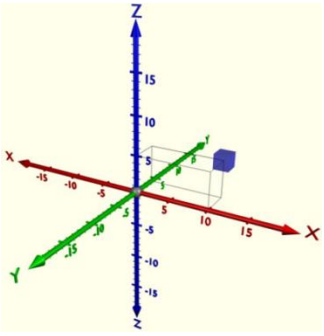
x, y, z 값을 변경하면 Cube 의 중심좌표가 이동

bpy.context.object.locaton[0] = 1 # X 축 (1,0,0) 이동

bpy.context.object.locaton[1] = 1 # Y 축 (1,1,0) 이동

bpy.context.object.locaton[2] = 1 # Z 축 (1,1,1) 이동

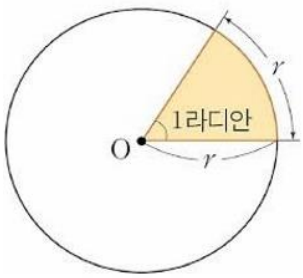
블렌더의 좌표계는 아래와 같다.



회전

Rotation X 의 값을 90 도로 입력하면 다음과 같은 스크립트가 생성된다.

bpy.context.object.rotation_euler[0] = 1.5708



90 도=1.5708 라디언이다.

확대

Scale X 의 값을 2 로 하면 2m*2m*2m 의 크기가 되는 직사각형 cube 가 되는 것을 확인할 수 있다.

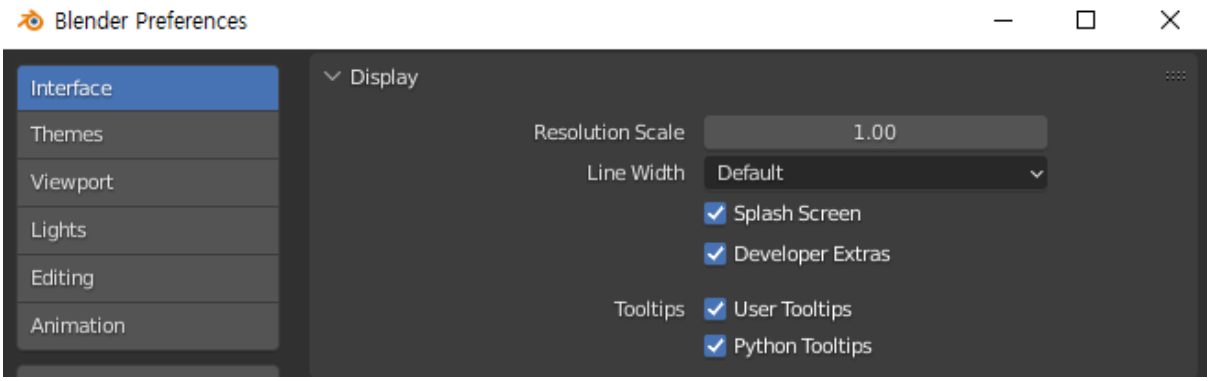
bpy.context.object.scale[0] = 2

bpy.context.object.scale[1] = 2

bpy.context.object.scale[1] = 2

스크립트로 다른 도형을 추가하거나 메뉴의 기능을 실행하려면

Edit - Perferences



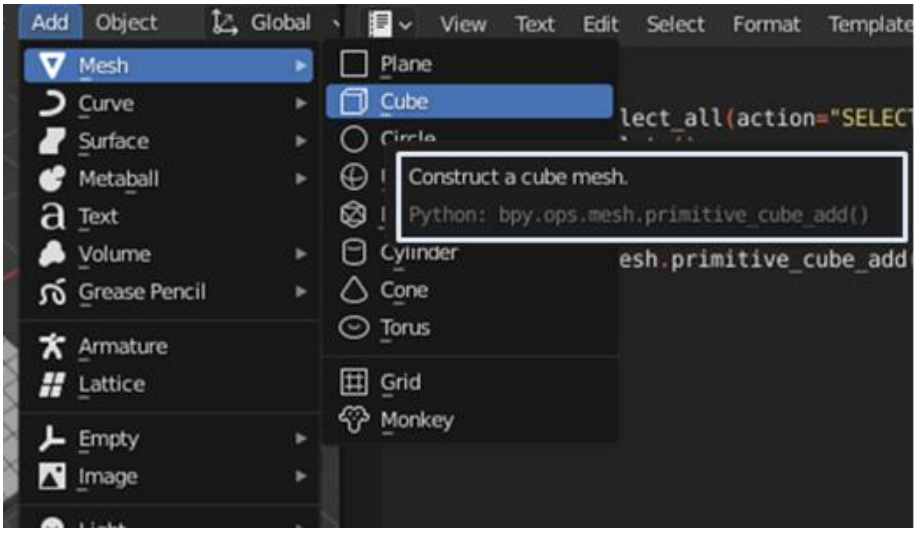
Interface 탭에서 python tooltip 항목 User Tooltips, develper extra 체크

마우스를 올리고 잠시 기다려보면 툴팁이 나타나는데

해당 파이썬 명령문을 확인할 수 있음

이 명령문을 파이썬 콘솔에서 입력하고 엔터를 누르면

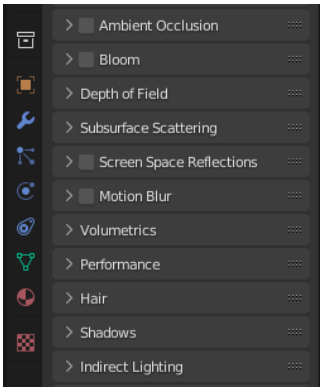
메뉴를 누른 것과 같은 기능이 실행된다.



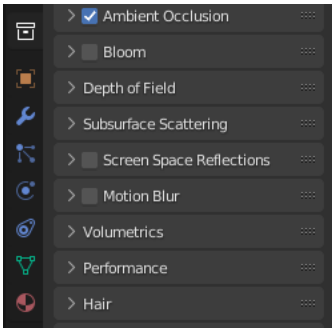
명령문을 복사하려면 마우스 우클릭 Copy Python Command 클릭

이처럼 모든 기능을 스크립트로 사용할 수 있다.

오른쪽 메뉴의 값 켜고 끄기




```
>>> bpy.data.scenes['Scene'].eevee.use_gtao
False

>>> bpy.data.scenes['Scene'].eevee.use_gtao = True
>>> bpy.data.scenes['Scene'].eevee.use_gtao
True
```



오퍼레이터 만드는 법

기본 템플릿 이용해서 나만의 기능을 만들 수 있다.
템플릿은 Templetes – Operator Simple 사용

```
import bpy


def main(context):
    for ob in context.scene.objects:
        print(ob)


class SimpleOperator(bpy.types.Operator):
    """Tooltip"""
    bl_idname = "object.simple_operator"
    bl_label = "Simple Object Operator"

    @classmethod
    def poll(cls, context):
        return context.active_object is not None

    def execute(self, context):
        main(context)
        return {'FINISHED'}

def menu_func(self, context):
    self.layout.operator(SimpleOperator.bl_idname, text=SimpleOperator.bl_label)

# Register and add to the "object" menu (required to also use F3 search "Simple Object
Operator" for quick access)

def register():
    bpy.utils.register_class(SimpleOperator)
    bpy.types.VIEW3D_MT_object.append(menu_func)


def unregister():
    bpy.utils.unregister_class(SimpleOperator)
    bpy.types.VIEW3D_MT_object.remove(menu_func)


if __name__ == "__main__":
    register()

    # test call
    bpy.ops.object.simple_operator()
```

```
if __name__ == "__main__":
    register()

    # test call
    bpy.ops.object.simple_operator()
```

register() 함수가 호출되면

```
def register():
    bpy.utils.register_class(SimpleOperator)
    bpy.types.VIEW3D_MT_object.append(menu_func)
```

bpy.utils 모듈에 있는 register_class 함수가 호출

register() 함수는 사용자가 작성한 오퍼레이터 클래스를 블렌더에 등록해주는 역할

새로 만들어진 오퍼레이터는 등록해줘야 사용할 수 있음

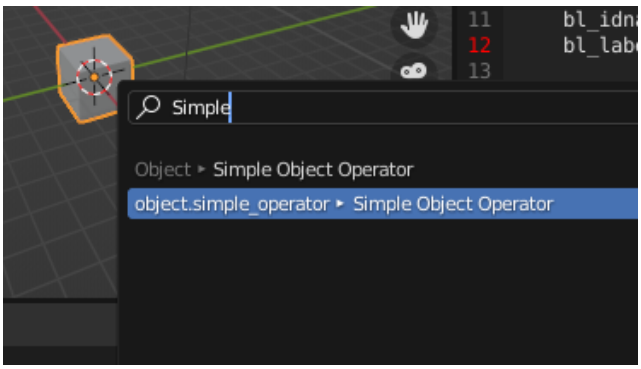
함수의 인수로 새로 만들어진 오퍼레이터 클래스 이름을 넣어주면 된다.

```
class SimpleOperator(bpy.types.Operator):
    """Tooltip"""
    bl_idname = "object.simple_operator"
    bl_label = "Simple Object Operator"

    @classmethod
    def poll(cls, context):
        return context.active_object is not None

    def execute(self, context):
        main(context)
        return {'FINISHED'}
```

클래스를 정의한 문장을 보면 사용자 정의 오퍼레이터를 어떻게 만들었는지 알 수 있다.

사용자 정의 오퍼레이터를 만드려면 bpy.types 모듈에서 제공하는 Operator 클래를 상속 받아야 한다.

poll 함수와 execute 함수는 기본 클래스의 함수를 재정의 하고 있음



3d 뷰포트에서 F3 을 누르면 만들어진 오퍼레이터를 검색할 수 있는데

bl_lable 의 대입된 문자열이 이름으로 표시

해당 항목을 클릭하면 오퍼레이터가 실행

```python
def main(context):
    for ob in context.scene.objects:
        print(ob)
```

main 함수의 내용을 보면

for 로 시작하는 반복문으로 되어있음

context.scene.objects 는 현재의 장면에 포함된 모든 오브젝트를 뜻하며

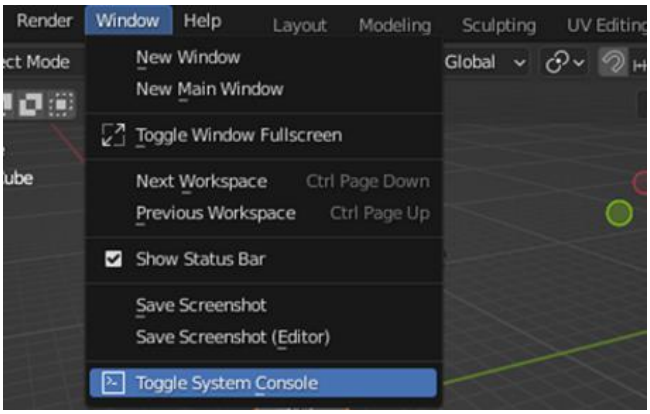
이 오브젝트의 숫자만큼 반복

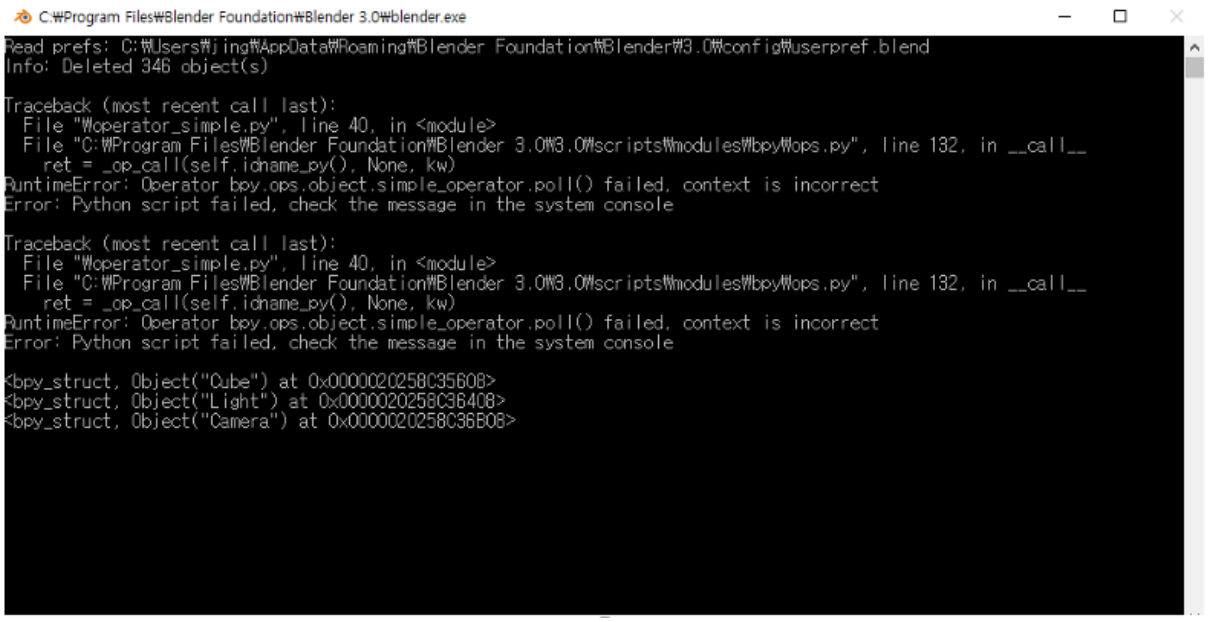
실행했기 때문에 레지스터 함수가 새로 만들어진 오퍼레이터에 등록되었고

이 오퍼레이터를 테스트로 한번 실행

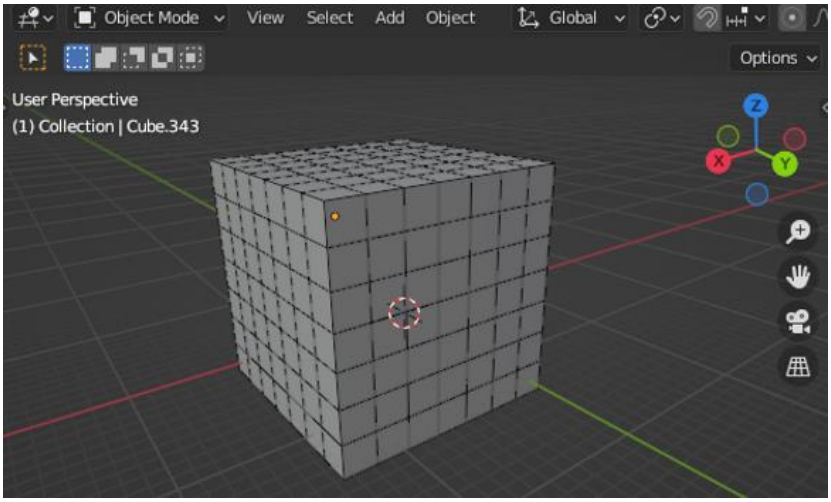
실행되면 모든 오브젝트의 정보가 하나씩 프린트 문으로 출력되는데

시스템 콘솔에 글자가 출력된다.





아래 세 줄은 오퍼레이터를 실행하면서 출력된 내용이다.




```
import bpy

number_of_cubes = 7
x_pos = 0
y_pos = 0
z_pos = 0
for i in range(number_of_cubes):
	y_pos = 0
	for j in range(number_of_cubes):
		z_pos = 0
		for k in range(number_of_cubes):

			bpy.ops.mesh.primitive_cube_add(enter_editmode=False,
                                                        location=(z_pos, y_pos, x_pos))
			z_pos += 2

			bpy.ops.rigidbody.object_add()
			bpy.context.object.rigid_body.collision_shape = 'BOX'
			bpy.context.object.rigid_body.use_margin = True
			bpy.context.object.rigid_body.collision_margin = 0
			bpy.context.object.rigid_body.linear_damping = 0.35
			bpy.context.object.rigid_body.angular_damping = 0.6
			bpy.context.object.rigid_body.mass = 20

		y_pos+=2

	x_pos+=2
```