

# Blender - Python API

**#3**



Serdar ARITAN

Department of Computer Graphics  
Hacettepe University, Ankara, Turkey



- The **BCO602 Toolkit** for Creation and Transformation program is a minimal set of tools for creating, selecting, and transforming objects. The script at the page 10 runs some example transformations.
- Before we begin, give this toolkit a Python filename of **bco602tk.py** using the bar at the top of the Text Editor. Now, click the plus sign in the base of the Text Editor to create a new script. The file **bco602tk.py** is now a linked script within the Blender Python environment, and we can import it into other scripts within the environment.



```
View Text Edit Select Format Templates
1 import bpy
2 tk = bpy.data.texts["bco602tk.py"].as_module()
3 # Create a cube
4 tk.create.cube('PerfectCube')
5 # Differential transformations combine
6 tk.sel.translate((0, 1, 2))
7 tk.sel.scale((1, 1, 2))
8 tk.sel.scale((0.5, 1, 1))
9 tk.sel.rotate_x(3.1415 / 8)
10 tk.sel.rotate_x(3.1415 / 7)
11 tk.sel.rotate_z(3.1415 / 3)
12 # Create a cone
13 tk.create.cone('PointyCone')
14 # Declarative transformations overwrite
15 tk.act.location((-2, -2, 0))
16 tk.spec.scale('PointyCone', (1.5, 2.5, 2))
17 # Create a Sphere
18 tk.create.sphere('SmoothSphere')
19 # Declarative transformations overwrite
20 tk.spec.location('SmoothSphere', (2, 0, 0))
21 tk.act.rotation((0, 0, 3.1415 / 3))
22 tk.act.scale((1, 3, 1))
23
```

tryTk.py

bco602tk.py  
tryTk.py

🔍



## Blender/Python API

### Transformations with bpy

```
# BCO602 Toolkit for Creation and Transformation (bco602tk.py )
import bpy

# Selecting objects by name
def select(objName):
    bpy.ops.object.select_all(action='DESELECT')
    # Set the 'select' property of the datablock to True
    bpy.data.objects[objName].select_set(True)

# Activating objects by name
def activate(objName):
    # Make the objName the active object
    bpy.context.view_layer.objects.active = bpy.data.objects[objName]

class sel:..... page(5)
class act:..... page(6)
class spec:..... page(7)
class create:..... page(8)
```





Under `sel` Namespace

`ops`

```
class sel:
    """Function Class for operating on SELECTED objects"""
    # Differential
    def translate(v):
        bpy.ops.transform.translate(
            value=v, constraint_axis=(True, True, True))
    # Differential
    def scale(v):
        bpy.ops.transform.resize(value= v) # value is tuple (v, v, v)
    # Differential
    def rotate_x(v):
        bpy.ops.transform.rotate(value=v, orient_axis='X')
    # Differential
    def rotate_y(v):
        bpy.ops.transform.rotate(value=v, orient_axis='Y')
    # Differential
    def rotate_z(v):
        bpy.ops.transform.rotate(value=v, orient_axis='Z')
```



Under `act` Namespace

`context`

```
class act:
```

```
    """Function Class for operating on ACTIVE objects"""
```

```
    # Declarative
```

```
    def location(v):
```

```
        bpy.context.object.location = v
```

```
    # Declarative
```

```
    def scale(v):
```

```
        bpy.context.object.scale = v
```

```
    # Declarative
```

```
    def rotation(v):
```

```
        bpy.context.object.rotation_euler = v
```

```
    # Declarative
```

```
    def dimension(v):
```

```
        bpy.context.object.dimensions = v
```

```
    # Rename the active object
```

```
    def rename(objName):
```

```
        bpy.context.object.name = objName
```



Under `spec` Namespace

`data`

```
class spec:
    """Function Class for operating on SPECIFIED objects"""

    # Declarative
    def scale(objName, v):
        bpy.data.objects[objName].scale = v

    # Declarative
    def location(objName, v):
        bpy.data.objects[objName].location = v

    # Declarative
    def rotation(objName, v):
        bpy.data.objects[objName].rotation_euler = v
```



Under `create` Namespace

`ops`

```
class create:
```

```
    """Function Class for CREATING Objects"""
```

```
    def cube(objName):
```

```
        bpy.ops.mesh.primitive_cube_add(radius=0.5, location=(0, 0, 0))
```

```
        act.rename(objName)
```

```
    def sphere(objName):
```

```
        bpy.ops.mesh.primitive_uv_sphere_add(size=0.5, location=(0, 0, 0))
```

```
        act.rename(objName)
```

```
    def cone(objName):
```

```
        bpy.ops.mesh.primitive_cone_add(radius1=0.5, location=(0, 0, 0))
```

```
        act.rename(objName)
```



Under GLOBAL

```
# Delete an object by name
```

```
def delete(objName):  
    select(objName)  
    bpy.ops.object.delete(use_global=False)
```

```
# Delete all objects
```

```
def delete_all():  
    if(len(bpy.data.objects) != 0):  
        bpy.ops.object.select_all(action='SELECT')  
        bpy.ops.object.delete(use_global=False)
```



```
import bpy
tk = bpy.data.texts["bco602tk.py"].as_module()
# Create a cube
tk.create.cube('PerfectCube')
# Differential transformations combine
tk.sel.translate((0, 1, 2))
tk.sel.scale((1, 1, 2))
tk.sel.scale((0.5, 1, 1))
tk.sel.rotate_x(3.1415 / 8)
tk.sel.rotate_y(3.1415 / 7)
tk.sel.rotate_z(3.1415 / 3)
# Create a cone
tk.create.cone('PointyCone')
tk.act.location((-2, -2, 0))
tk.spec.scale('PointyCone', (1.5, 2.5, 2))
# Create a Sphere
tk.create.sphere('SmoothSphere')
tk.spec.location('SmoothSphere', (2, 0, 0))
tk.act.rotation((0, 0, 3.1415 / 3))
tk.act.scale((1, 3, 1))
```



# SCRIPT LANGUAGES FOR ANIMATION

## Blender/Python API

The screenshot displays the Blender 2.80 interface. The top menu bar includes File, Edit, Render, Window, Help, Layout, Modeling, Sculpting, UV Editing, Texture Paint, Shading, Animation, Rendering, Compositing, and Scripting. The Scripting editor is active, showing a Python script in the try Tk.py file. The 3D viewport shows a scene with a cone and a sphere. The Properties panel on the right shows the Scene Collection and Current File properties.

```
1 import bpy
2 tk = bpy.data.texts["bco602tk.py"].as_module()
3 # Create a cube
4 tk.create_cube('PerfectCube')
5 # Differential transformations combine
6 tk.sel.translate((0, 1, 2))
7 tk.sel.scale((1, 1, 2))
8 tk.sel.scale((0.5, 1, 1))
9 tk.sel.rotate_x(3.1415 / 8)
10 tk.sel.rotate_x(3.1415 / 7)
11 tk.sel.rotate_z(3.1415 / 3)
12 # Create a cone
13 tk.create_cone('PointyCone')
14 # Declarative transformations overwrite
15 tk.act.location((-2, -2, 0))
16 tk.spec.scale('PointyCone', (1.5, 2.5, 2))
17 # Create a Sphere
18 tk.create_sphere('SmoothSphere')
19 # Declarative transformations overwrite
20 tk.spec.location('SmoothSphere', (2, 0, 0))
21 tk.act.rotation((0, 0, 3.1415 / 3))
22 tk.act.scale((1, 3, 1))
23
```

PYTHON INTERACTIVE CONSOLE 3.7.4 (default, Feb 17 2020, 16:23:28) [MSC v.1916 64 bit (AMD64)]

Builtin Modules: bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, ma, thutils

Convenience Imports: from mathutils import \*, from math import \*

Convenience Variables: C = bpy.context, D = bpy.data

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

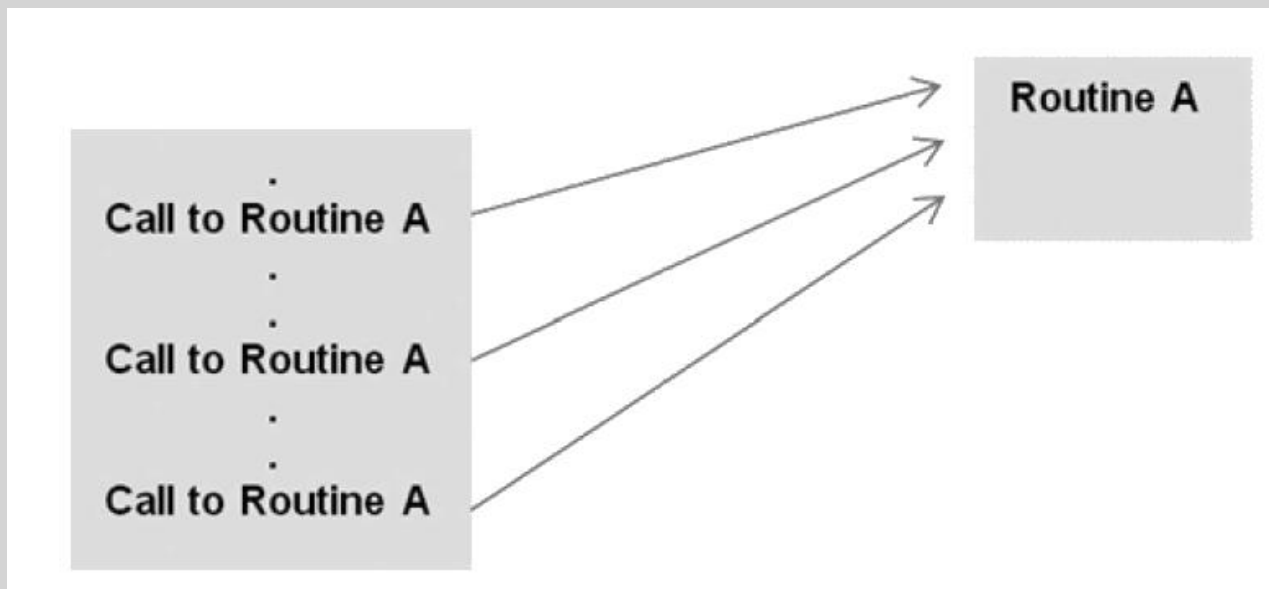
File: C:\Lectures\BCO 602 Animasyon İçin Betik Diller\Hafta\_03\try\_tk.py

Collection | SmoothSphere | Verts:523 | Faces:551 | Tris:1,034 | Objects:1/5 | Mem: 17.2 MiB | 2.83.4





## Functions



**A program routine is a named group of instructions that accomplishes some task. A routine may be invoked (called) as many times as needed in a given program. A function is Python's version of a program routine.**



## Functions

The first step to code reuse is the function: **a named piece of code**, separate from all others. A function can take **any number and type of input parameters** and **return any number and type of output results**. You can do two things with a function:

- Define it
- Call it

```
>>> def do_nothing():  
    pass  
  
>>> do_nothing()  
>>> type(do_nothing)  
<class 'function'>  
>>> do_nothing  
<function do_nothing at 0x00000001036AFD90>  
>>>
```



## Functions

Just like a value can be associated with a name, a piece of logic can also be associated with a name by defining a function.

```
>>> def square(x):  
    return x * x
```



```
>>> square(5)  
25
```



The body of the function is indented. Indentation is the Python's way of grouping statements.

The functions can be used in any expressions.

```
>>> square(2) + square(3)  
13  
>>> square(square(3))  
81
```



## Functions

```
def name(arg1, arg2, ... argN):  
    statements
```

As with all compound Python statements, **def** consists of a header line followed by a block of statements, usually indented (or a simple statement after the colon).

Function Header ▶	<code>def avg(n1, n2, n3):</code>	<code>&gt;&gt;&gt; num1 = 10</code>
	<code>    -----</code>	<code>&gt;&gt;&gt; num2 = 25</code>
Function Body (suite) ▶	<code>    -----</code>	<code>&gt;&gt;&gt; num3 = 16</code>
	<code>    -----</code>	<code>&gt;&gt;&gt; avg(num1, num2, num3)</code>
	<code>    -----</code>	

Function names have the same rules as variable names (they must start with a letter or `_` and contain only letters, numbers, or `_`).



## Functions

### Function Definition

```
def avg(n1, n2, n3):  
    return (n1 + n2 + n3) / 3.0
```

### Function Value

17.0

```
result = avg(10, 25, 16) * factor
```

### Function Call

## Call to Value-Returning Function



## Functions

define and call another function that has **no parameters** but prints a single word:

```
>>> def make_a_sound():  
    print('quack')
```

```
>>> make_a_sound()  
quack
```

When you called the `make_a_sound()` function, Python ran the code inside its definition. A function that has **no parameters** but **returns** a value:

```
>>> def agree():  
    return True
```



We can call this function and test its returned value by using if:

```
>>> if agree(): # if True:
    print('Splendid!')
else:
    print('That was unexpected.')
```

Splendid!





it's time to put something between those parentheses

```
>>> def echo(anything):  
    return anything + ' ' + anything
```

```
>>> echo('Is there anybody')  
'Is there anybody Is there anybody'
```

```
>>> echo(1500)  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```



a function that takes an input argument and does something with it

```
def commentary(color):  
    if color == 'red':  
        return "It's a tomato."  
    elif color == "green":  
        return "It's a green pepper."  
    elif color == 'bee purple':  
        return "I don't know what it is, but only bees can see  
it."  
    else:  
        return "I've never heard of the color " + color + "."
```



## Functions



Functions are generally defined at the top of a program. However, every function **must be defined before it is called**.

**def** Executes at Runtime



The Python **def** is a true executable statement: when it runs, it creates a new function object and assigns it to a name. Because it's a statement, a **def** can appear anywhere a statement can—even nested in other statements

```
if test:
```

```
    def func():    # Define func this way
```

```
    ...
```

```
else:
```

```
    def func():    # Or else this way
```

```
    ...
```

```
func()
```

```
    # Call the version selected and built
```



## Functions

Generally, **defs** are not evaluated until they are reached and run, and the code inside **defs** is not evaluated until the functions are later called. Because function definition happens at runtime, there's nothing special about the function name. What's important is the object to which it refers:

```
othername = func      # Assign function object  
othername()           # Call func again
```



Here, the function was assigned to a different name and called through the new name. Like everything else in Python, functions are just objects; they are recorded explicitly in memory at program execution time.

```
def func(): ...           # Create function object
func()                   # Call object
func.attr = value        # Attach attribute
```



The first statement in the body of a function is usually a string, which can be accessed with `function_name.__doc__`. This statement is called **Docstring**.

```
>>> def hello(name):  
    """ Greets a person """  
    print('Hello', name + '!')  
  
>>>  
>>> hello(name = 'Serdar')  
>>>  
>>> print("The docstring of the function : "+ Hello.__doc__)
```



**Functions in Python are first-class objects.** Programming language theorists define a “*first-class object*” as a program entity that can be:

- Created at runtime
- Assigned to a variable or element in a data structure
- Passed as an argument to a function
- Returned as the result of a function





### Treating a Function Like an **Object**

```
>>> def factorial(n):  
    '''returns n!'''  
    return 1 if n < 2 else n * factorial(n - 1)  
  
>>> factorial(42)  
1405006117752879898543142606244511569936384000000000  
>>> factorial.__doc__  
'returns n!'  
>>> type(factorial)  
<class 'function'>
```



## When to Use a Function

**Only one purpose:** A function should be the encapsulation of a single, identifiable operation.

**Readable:** A function should be readable.

**Not too long:** A function shouldn't be too long.

**Reusable:** A function should be reusable in contexts other than the program it was written for originally.

**Complete:** A function should be complete, in that it works in all potential situations. If you write a function to perform one thing, you should make sure that *all the cases* where it might be used are taken into account.

**Able to be refactored:** Refactoring is the process of taking existing code and modifying it such that its structure is somehow improved but the functionality of the code remains the same.



## Functions

We can even create more functions using the existing ones.

```
>>> def sum_of_squares(x, y):  
    return square(x) + square(y)
```

```
>>> sum_of_squares(2, 3)  
13
```

```
>>> def square(x):  
    return x * x
```

Functions are just like other values, they can assigned, passed as arguments to other functions etc.

```
>>> f = square
```

```
>>> f(4)
```

```
16
```


```
>>> def fxy(f, x, y):  
    return f(x) + f(y)
```

```
>>> fxy(square, 2, 3)
```

```
13
```



```
>>>def cube(x):  
    return x * x * x  
>>> fxy(cube, 2, 3)  
35
```



There is another way of creating functions, using the *lambda* operator.

```
>>>forthpower = lambda x: x ** 4  
>>> fxy(forthpower, 2, 3)  
97  
>>> fxy(lambda x: x ** 5, 2, 3)  
275
```

The *lambda* operator becomes handy when writing small functions to be passed as arguments etc.



$n! = n * (n-1)!, \text{ if } n > 1 \text{ and } f(1) = 1$

Example:

$4! = 4 * 3!$

$3! = 3 * 2!$

$2! = 2 * 1$

```
>>> def factorial(n):  
    f = 1  
    while (n > 0):  
        f = f * n  
        n = n - 1  
    return f
```

Recursion has  
something to  
do with infinity



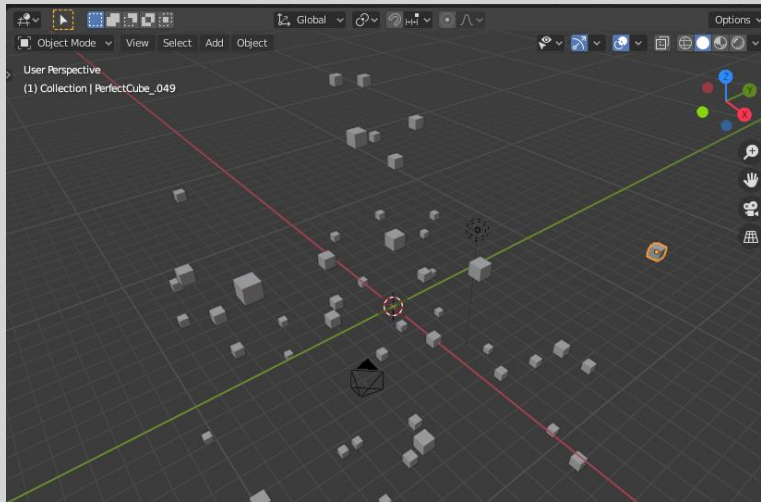
```
>>> def factorial(n):  
    if n == 0:  
        return 1  
    else:  
        return n * factorial(n-1)
```

```
>>> f= lambda x: x and x * f(x - 1) or 1
```



```
import bpy
from random import randint
tk = bpy.data.texts["bco602tk.py"].as_module()

# Generate 50 cubes in random locations
for i in range(50):
    # Create a cube
    tk.create.cube('PerfectCube_')
    # Differential transformations combine
    x, y, z = (randint(-10, 10), randint(-10, 10), randint(-10, 10))
    tk.sel.translate((x, y, z))
```





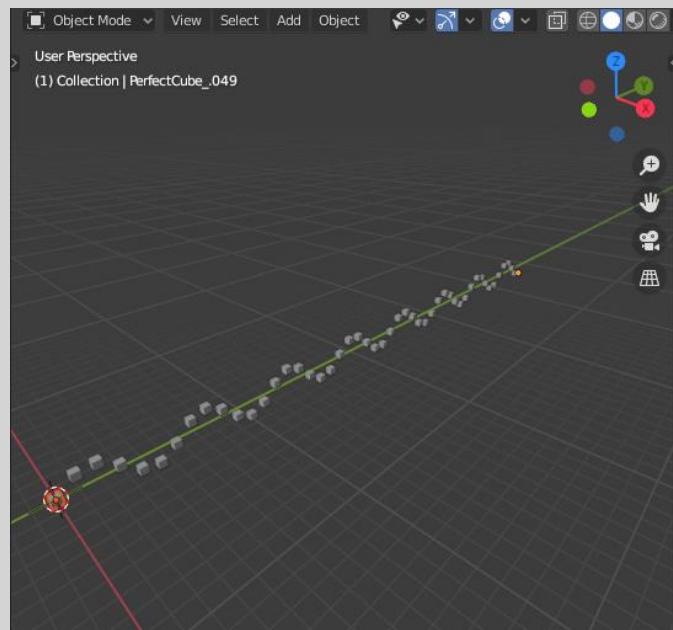


# SCRIPT LANGUAGES FOR ANIMATION

## Blender/Python API

```
import bpy
from math import sin
tk = bpy.data.texts["bco602tk.py"].as_module()
```

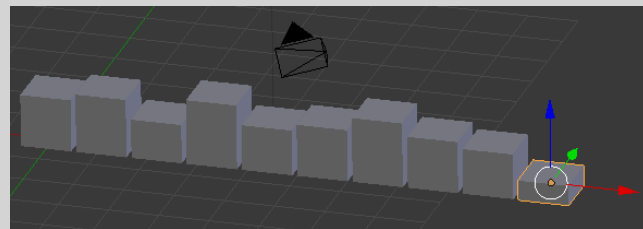
```
# Generate cubes on Sinus signal
for i in range(50):
    # Create a cube
    tk.create.cube('PerfectCube_')
    # Differential transformations combine
    x, y, z = 0, i, sin(i)
    tk.sel.translate((x, y, z))
```





```
import bpy
from random import randint
tk = bpy.data.texts["bco602tk.py"].as_module()

# Generate 10 random numbers in a list
data = [randint(1, 10) for _ in range(10)]
sumdata = sum(data)
for i in range(len(data)):
    # Set cube physical height as proportion of this random number from the total X 10
    cubeHeight = (data[i] / sumdata) * 10
    # Create a cube
    tk.create.cube('DataCube_')
    # Differential transformations combine
    tk.sel.translate((i * 1.1, 0, cubeHeight/2))
    tk.act.dimension((1, 1, cubeHeight))
```





```
import bpy
from random import randint
from random import radians
tk = bpy.data.texts["bco602tk.py"].as_module()
```

```
# Generate 5 random numbers in a list
```

```
data = [randint(1, 5) for _ in range(5)]
```

```
...
```

```
...
```

```
...
```

```
# Add text object
```

```
bpy.ops.object.text_add(location = ( i * 1.1, 1.5, 0 ))
```

```
# Rotate text by 90 degrees along Z axis
```

```
bpy.context.object.rotation_euler.z = radians(180)
```

```
# Add depth to the text
```

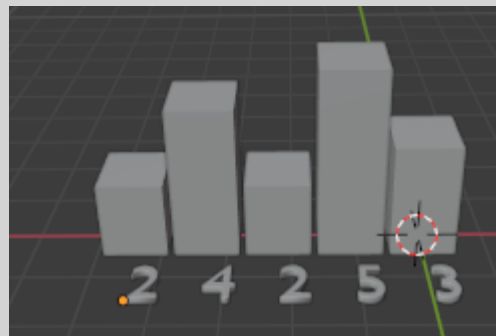
```
bpy.context.object.data.extrude = 0.05
```

```
# Add a nice bevel effect to smooth the text's edges
```

```
bpy.context.object.data.bevel_depth = 0.01
```

```
# Set the text to be the current data
```

```
bpy.context.object.data.body = str(data[i])
```





<https://fivethirtyeight.com/features/should-travelers-avoid-flying-airlines-that-have-had-crashes-in-the-past/>

← → ↺ 🏠 🔒 https://fivethirtyeight.com/features/should-travelers-avoid-fly/ 📄 ⋮

⚙️ Most Visited 🌐 Problem loading page 🌐 Getting Started 📌 Suggested Sites 🌐 Problem loading page 🌐 Web Slice Gall

---

## FiveThirtyEight

Politics Sports Science & Health Economics **Culture**

JUL 18, 2014 AT 8:20 PM

### Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past?

By [Nate Silver](#)

Filed under [Airline Safety](#)

Get the data on [GitHub](#)

📌

📌

📌



A memorial with candles that read "MH17" glows silently as unseen churchgoers pray for the victims of Malaysia Airlines flight MH17 from Amsterdam to Kuala Lumpur, at a church in Kuala Lumpur on Friday. NICOLAS ASFOURI / AFP / GETTY IMAGES

<https://github.com/fivethirtyeight/data>

AIRLINE	AVAILABLE SEAT KM PER WEEK	1985-1999			2000-2014		
		REVENUE	LOADING	AVAILABILITY	REVENUE	LOADING	AVAILABILITY
Aer Lingus	321e	2	0	0	0	0	0
Aeroflot*	1.198	76	14	128	6	1	89
Aerolineas	386	6	0	0	1	0	0
Argentinian	597	3	1	64	5	0	0
Aeromexico*	1.665	2	0	0	2	0	0
Air Canada	3.004	14	4	79	6	2	337
Air India*	869	2	1	329	4	1	158
Air New Zealand*	710	3	0	0	5	1	7
Alaska Airlines*	965	5	0	0	5	1	88
Alitalia	698	7	2	50	4	0	0
All Nippon	1.843	3	1	1	7	0	0
American*	5.228	21	5	101	17	3	418
Austrian Airlines	358	1	0	0	1	0	0
Avianca	397	5	3	323	0	0	0
British Airways*	3.180	4	0	0	6	0	0
Cathay Pacific*	2.582	0	0	0	2	0	0
China Airlines	815	12	6	535	2	1	225
Condor	418	2	1	16	0	0	0
COPA	550	3	1	47	0	0	0
Delta / Northwest*	6.526	24	12	497	24	2	51
EgyptAir	558	8	3	282	4	1	14
EI AI	335	1	1	4	1	0	0
Ethiopian Airlines	489	25	5	167	5	2	62
Finnair	506	1	0	0	0	0	0
Garuda Indonesia	613	10	3	280	4	2	22
Gulf Air	391	1	0	0	3	1	143
Hawaiian Airlines	494	0	0	0	1	0	0
Iberia	1.173	4	1	148	5	0	0
Japan Airlines	1.574	3	1	520	0	0	0
Kenya Airways	277	2	0	0	2	2	283
KLM*	1.875	7	1	3	1	0	0
Korean Air	1.735	12	5	435	1	0	0
LAN Airlines	1.002	3	2	21	0	0	0
Lufthansa*	3.427	6	1	2	3	0	0
Malaysia Airlines	1.039	3	1	34	3	2	537
Pakistan International	349	8	3	234	10	2	46
Philippine Airlines	413	7	4	74	2	1	1
Qantas*	1.917	1	0	0	5	0	0
Royal Air Maroc	286	5	3	51	3	0	0
SAS*	683	5	0	0	6	1	110
Saudi Arabian	860	7	2	313	11	0	0
Singapore Airlines	2.277	2	2	6	2	1	83
South African	652	2	1	159	1	0	0
Southwest Airlines	3.277	1	0	0	8	0	0
Sri Lankan Airlines / Air Lanka	326	2	1	14	4	0	0
SWISS*	793	2	1	229	3	0	0
TACA	299	3	1	3	1	1	3
TAM	1.509	8	3	98	7	2	188
TAP - Air Portugal	610	0	0	0	0	0	0
Thai Airways	1.703	8	4	308	2	1	1
Turkish Airlines	1.846	8	3	64	8	2	84
United / Continental*	7.139	19	8	319	14	2	109
US Airways / America West*	2.456	16	7	224	11	2	23
Vietnam Airlines	625	7	3	171	1	0	0
Virgin Atlantic	1.005	1	0	0	0	0	0
Xiamen Airlines	430	9	1	82	2	0	0

Join GitHub today

GitHub is home to over 20 million developers working together to host and review code, manage projects, and build software together.

Sign up

Data and code behind the stories and interactives at FiveThirtyEight

833 commits   1 branch   0 releases   32 contributors

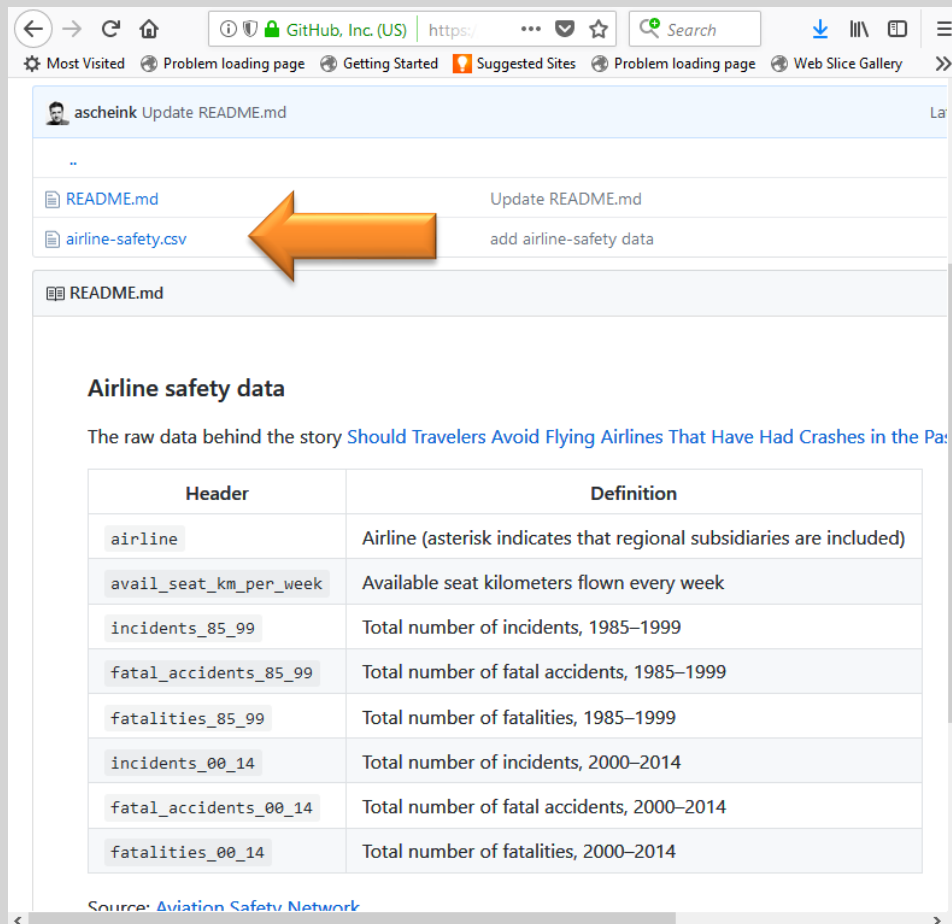
Branch: master   New pull request

dmil add redirecting to index

- ahca-polls   update README for ahca-polls
- airline-safety   Update README.md
- alcohol-consumption   Revert "Update drinks.csv"
- antiquities-act   Update README.md
- avengers   add avengers data
- bachelorette   Update README.md
- bad-drivers   add bad drivers data
- bechdel   format email address
- biopics   for race\_known as unknown, make subject\_race blank. not White



## Visualizing Three Dimensions of Data



The screenshot shows a GitHub repository page for the file 'airline-safety.csv'. An orange arrow points to the file name in the file list. Below the file list, the README content is visible, including a section titled 'Airline safety data' and a table defining the data headers.

### Airline safety data

The raw data behind the story [Should Travelers Avoid Flying Airlines That Have Had Crashes in the Past](#)

Header	Definition
airline	Airline (asterisk indicates that regional subsidiaries are included)
avail_seat_km_per_week	Available seat kilometers flown every week
incidents_85_99	Total number of incidents, 1985–1999
fatal_accidents_85_99	Total number of fatal accidents, 1985–1999
fatalities_85_99	Total number of fatalities, 1985–1999
incidents_00_14	Total number of incidents, 2000–2014
fatal_accidents_00_14	Total number of fatal accidents, 2000–2014
fatalities_00_14	Total number of fatalities, 2000–2014

Source: [Aviation Safety Network](#)



## Visualizing Three Dimensions of Data

airline-safety.csv - Microsoft Excel

	A	B	C	D	E	F	G	H	I	J
1	airline	avail_seat_km	incidents	fatal_acci	fatalities	incidents	fatal_acci	fatalities_00_14		
2	Aer Lingus	320906734	2	0	0	0	0	0		
3	Aeroflot*	1197672318	76	14	128	6	1	88		
4	Aerolineas Argentinas	385803648	6	0	0	1	0	0		
5	Aeromexico*	596871813	3	1	64	5	0	0		
6	Air Canada	1865253802	2	0	0	2	0	0		
7	Air France	3004002661	14	4	79	6	2	337		
8	Air India*	869253552	2	1	329	4	1	158		
9	Air New Zealand*	710174817	3	0	0	5	1	7		
10	Alaska Airlines*	965346773	5	0	0	5	1	88		
11	Alitalia	698012498	7	2	50	4	0	0		
12	All Nippon Airways	1841234177	3	1	1	7	0	0		
13	American*	5228357340	21	5	101	17	3	416		
14	Austrian Airlines	358239823	1	0	0	1	0	0		
15	Avianca	396922563	5	3	323	0	0	0		
16	British Airways*	3179760952	4	0	0	6	0	0		
17	Cathay Pacific*	2582459303	0	0	0	2	0	0		
18	China Airlines	813216487	12	6	535	2	1	225		
19	Condor	417982610	2	1	16	0	0	0		
20	COPA	550491507	3	1	47	0	0	0		
21	Delta / Northwest*	6525658894	24	12	407	24	2	51		
22	Egyptair	557699891	8	3	282	4	1	14		
23	El Al	335448023	1	1	4	1	0	0		
24	Ethiopian Airlines	488560643	25	5	167	5	2	92		
25	Finnair	506464950	1	0	0	0	0	0		
26	Garuda Indonesia	613356665	10	3	260	4	2	22		





## How to Read CSV File

from a web site

```
import csv
import urllib.request
# Read airline-safety.csv from file repository
url_str = 'http://yunus.hacettepe.edu.tr/~serdar.aritan/bco602/hafta_03/airline-safety.csv'
airline_csv = urllib.request.urlopen(url_str)
airline_ob = csv.reader(airline_csv.read().decode('utf-8').splitlines())
# Store header as list, and data as list of lists
airline_header = []
airline_data = []
for v in airline_ob:
    if not airline_header:
        airline_header = v
    else:
        # Columns
        v = [str(v[0]), # airline
             int(v[1]), # avail_seat_km_per_week
             int(v[2]), # incidents_85_99
             int(v[3]), # fatal_accidents_85_99
             int(v[4]), # fatalities_85_99
             int(v[5]), # incidents_00_14
             int(v[6]), # fatal_accidents_00_14
             int(v[7])] # fatalities_00_14
        airline_data.append(v)
```





```
for i in range(len(airline_data)):
    if airline_data[i][0] == "Turkish Airlines" \
       or airline_data[i][0] == "Lufthansa*" \
       or airline_data[i][0] == "Air France":
        print("{} incident report from 2000 to 2014".format(airline_data[i][0]))
        print("Incidents          : {}".format(airline_data[i][5]))
        print("Fatal Accidents : {}".format(airline_data[i][6]))
        print("Fatalities       : {}".format(airline_data[i][7]))
```

Air France incident report from 2000 to 2014

Incidents : 6

Fatal Accidents : 2

Fatalities : 337

Lufthansa\* incident report from 2000 to 2014

Incidents : 3

Fatal Accidents : 0

Fatalities : 0

Turkish Airlines incident report from 2000 to 2014

Incidents : 8

Fatal Accidents : 2

Fatalities : 84



```
import bpy
from random import randint
from math import radians
import csv
import urllib.request
tk = bpy.data.texts["bco602tk.py"].as_module()

# Read airline-safety.csv from file repository
url_str = 'http://yunus.hacettepe.edu.tr/~serdar.aritan/bco602/hafta_03/airline-safety.csv'
airline_csv = urllib.request.urlopen(url_str)
airline_ob = csv.reader(airline_csv.read().decode('utf-8').splitlines())
# Store header as list, and data as list of lists
airline_header = []
airline_data = []
for v in airline_ob:
    if not airline_header:
        airline_header = v
    else:
        # Columns:
        v = [str(v[0]), # airline
            int(v[1]), # avail_seat_km_per_week
            int(v[2]), # incidents_85_99
            int(v[3]), # fatal_accidents_85_99
            int(v[4]), # fatalities_85_99
            int(v[5]), # incidents_00_14
            int(v[6]), # fatal_accidents_00_14
            int(v[7])] # fatalities_00_14
        airline_data.append(v)
```

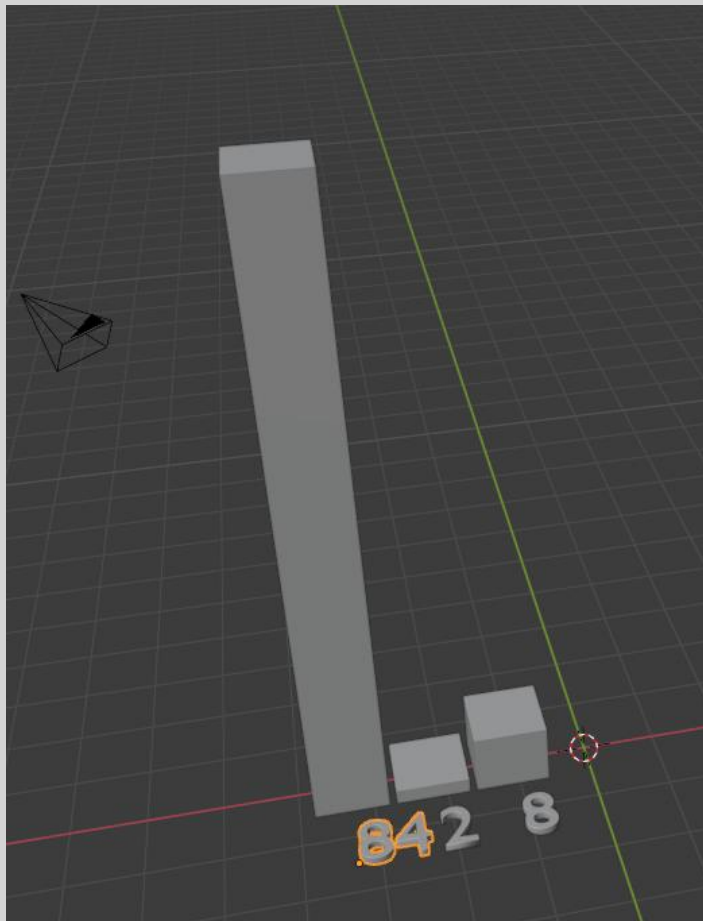


```
# print the airline data you want to look
for i in range(len(airline_data)):
    if airline_data[i][0]== "Turkish Airlines":
        print("{} incident report from 2000 to 2014".format(airline_data[i][0]))
        print("Incidents      : {}".format(airline_data[i][5]))
        print("Fatal Accidents : {}".format(airline_data[i][6]))
        print("Fatalities       : {}".format(airline_data[i][7]))

data = [airline_data[i][0], airline_data[i][5], airline_data[i][6], airline_data[i][7]]
sumdata = sum(data[1:])
for i in range(1, len(data)):
    # Set cube physical height as proportion of this random number from the total X 10
    cubeHeight = (data[i] / sumdata ) * 10
    # Create a cube
    tk.create.cube('DataCube_')
    # Differential transformations combine
    tk.sel.translate((i * 1.1, 0, cubeHeight/2))
    tk.act.dimension(( 1, 1, cubeHeight))

    # Add text object
    bpy.ops.object.text_add(location = ( i * 1.1, 1.5, 0 ))
    bpy.context.object.rotation_euler.z = radians(180) # Rotate text by 90 degrees along Z axis
    bpy.context.object.data.extrude      = 0.05      # Add depth to the text
    bpy.context.object.data.bevel_depth  = 0.01      # Add a nice bevel effect to smooth the text's edges
    bpy.context.object.data.body         = str(data[i]) # Set the text to be the current row's date
```

# Read the Data and Plot





## How to Read Excell File

```
.\python.exe -m pip install --upgrade pip
```

- PIP is a package manager for Python packages
- A package contains all the files you need for a module..

```
Select Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin

C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin>.\python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\sa-lenovo\appdata\roaming\python\python311\site-packages (24.2)

C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin>
```



- **Openpyxl 3.1.5**
- A Python library to read/write Excel 2010 xlsx/xlsm files

**`.\python.exe -m pip install openpyxl`**

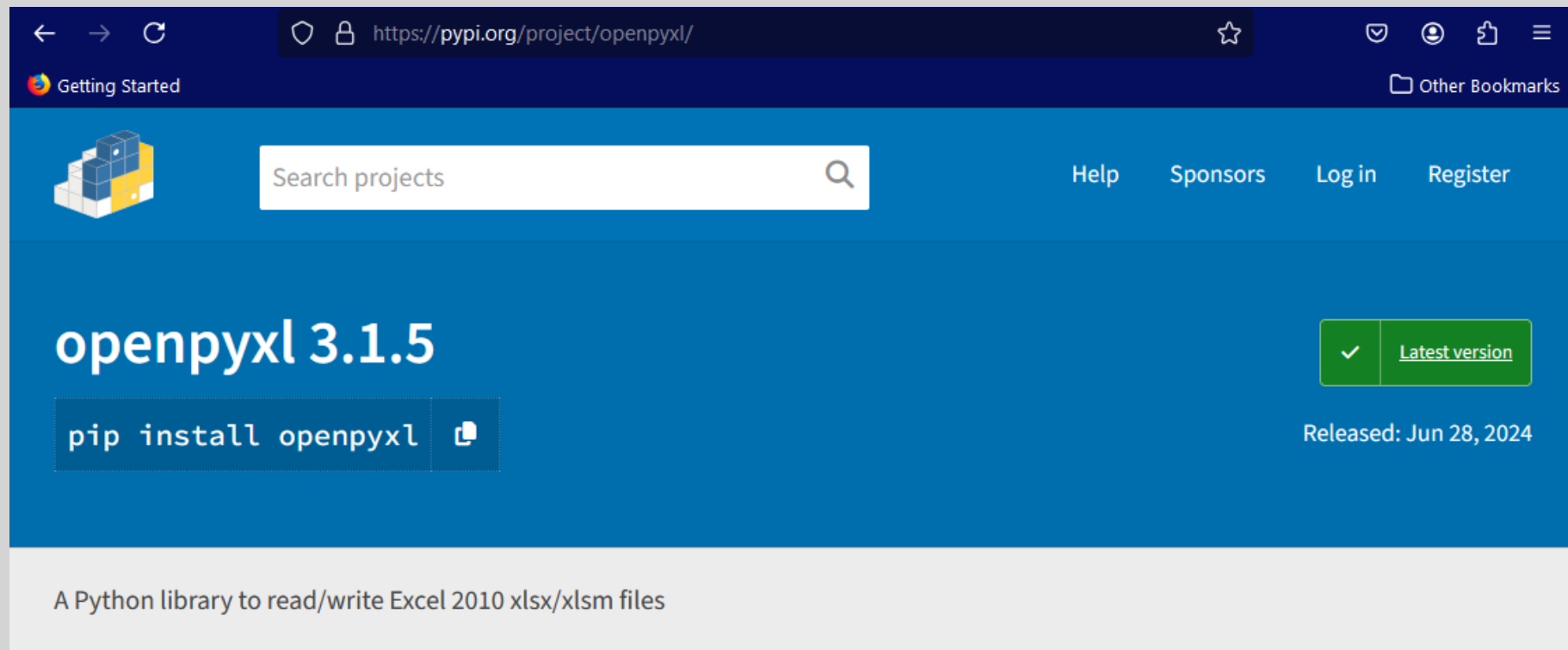
```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\System32>cd C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin

C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin>.\python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\sa-lenovo\appdata\roaming\python\python311\site-packages (24.2)

C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin>.\python.exe -m pip install openpyxl
Collecting openpyxl
  Downloading openpyxl-3.1.5-py2.py3-none-any.whl.metadata (2.5 kB)
Collecting et_xmlfile (from openpyxl)
  Downloading et_xmlfile-1.1.0-py3-none-any.whl.metadata (1.8 kB)
Downloading openpyxl-3.1.5-py2.py3-none-any.whl (250 kB)
Using cached et_xmlfile-1.1.0-py3-none-any.whl (4.7 kB)
Installing collected packages: et_xmlfile, openpyxl
Successfully installed et_xmlfile-1.1.0 openpyxl-3.1.5

C:\Program Files\Blender Foundation\Blender 4.2\4.2\python\bin>
```



The screenshot shows the PyPI project page for openpyxl. The browser address bar displays the URL `https://pypi.org/project/openpyxl/`. The page header includes navigation links: [Getting Started](#), [Other Bookmarks](#), [Help](#), [Sponsors](#), [Log in](#), and [Register](#). A search bar labeled "Search projects" is present. The main content area features the package name **openpyxl 3.1.5**, a green badge indicating it is the [Latest version](#), and a code block with the command `pip install openpyxl`. Below this, it states "Released: Jun 28, 2024". At the bottom, a description reads: "A Python library to read/write Excel 2010 xlsx/xlsm files".



← → ↺

🔒 https://openpyxl.readthedocs.io/en/stable/

📖 ☆

🔍 🧑 📁 ☰

🔥 Getting Started

📖 Other Bookmarks

openpyxl 3.1.3 documentation » openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files

next | modules | index

OpenPyXL

Table of Contents

openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files

- Introduction
- Security
- Mailing List
- Documentation
- Support
- How to Contribute
  - Other ways to help
- API Documentation
  - Key Classes

Indices and tables

Next topic

Tutorial

This Page

Show Source

Quick search

Go

openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files

Author:

Eric Gazoni, Charlie Clark

Source code:

<https://foss.heptapod.net/openpyxl/openpyxl>

Issues:

<https://foss.heptapod.net/openpyxl/openpyxl/-/issues>

Generated:

May 29, 2024

License:

MIT/Expat

Version:

3.1.3

coverage 96%

Introduction

openpyxl is a Python library to read/write Excel 2010 xlsx/xlsm/xltx/xltn files.

It was born from lack of existing library to read/write natively from Python the Office Open XML format.

All kudos to the PHPExcel team as openpyxl was initially based on PHPExcel.

Security

By default openpyxl does not guard against quadratic blowup or billion laughs xml attacks. To guard against these attacks install defusedxml.

Serdar ARITAN

48





## How to Read Excell File

```
View Text Edit Select Format Templates readExcell.py
1 from openpyxl import load_workbook
2
3 # Read airline-safety.xlsx from file repository
4 file_name = 'C:\\Lectures\\BC0 602 Animasyon Icin Betik Diller\\Hafta_03\\airline-safety.xlsx'
5
6 wb = load_workbook(file_name)
7 ws = wb[wb.sheetnames[0]]
8 for i, row in enumerate(ws.rows):
9     for cell in row:
10         print(f'{i}.row : {cell.value}')
11
12
```

```
C:\Program Files\Blender Foundation\Blender 3.3\blender.exe
53.row : 7
53.row : 224
53.row : 11
53.row : 2
53.row : 23
54.row : Vietnam Airlines
54.row : 625084918
54.row : 7
54.row : 3
54.row : 171
54.row : 1
54.row : 0
54.row : 0
55.row : Virgin Atlantic
55.row : 1005248585
55.row : 1
55.row : 0
55.row : 0
55.row : 0
55.row : 0
55.row : 0
56.row : Xiamen Airlines
56.row : 430462962
56.row : 9
56.row : 1
56.row : 82
56.row : 2
56.row : 0
56.row : 0
```



### Inserting an image

```
>>> from openpyxl import Workbook
>>> from openpyxl.drawing.image import Image
>>>
>>> wb = Workbook()
>>> ws = wb.active
>>> ws['A1'] = 'You should see three logos below'
```

```
>>> # create an image
>>> img = Image('logo.png')
```

```
>>> # add to worksheet and anchor next to cells
>>> ws.add_image(img, 'A1')
>>> wb.save('logo.xlsx')
```



Mac<sup>™</sup>OS

- PIP is a package manager for Python packages
- A package contains all the files you need for a module..

```
>>> import os
>>> import sys
>>> import subprocess
>>> print(os.path.join(sys.prefix, 'bin', 'python'))
/Applications/Blender.app/Contents/Resources/4.2/python/bin/python
>>> python_exec = os.path.join(sys.prefix, 'bin', 'python3.11')
>>> subprocess.call([python_exec, "-m", "pip", "install", "--upgrade", "pip"])
0
>>> subprocess.call([python_exec, "-m", "pip", "install", "--upgrade",
"scipy"])
0
>>> import scipy
```



Mac<sup>TM</sup> OS

- PIP is a package manager for Python packages
- A package contains all the files you need for a module..

```
>>> python_exec = os.path.join(sys.prefix, 'bin', 'python')
>>> subprocess.call([python_exec, "-m", "pip", "install", "--upgrade", "pip"])
Traceback (most recent call last):
  File "<blender_console>", line 1, in <module>
    File "/Applications/Blender.app/Contents/Resources/4.2/python/lib/python3.11/subprocess.py", line 389, in
    call
      with Popen(*popenargs, **kwargs) as p:
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/Applications/Blender.app/Contents/Resources/4.2/python/lib/python3.11/subprocess.py", line 1026, i
n __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
  File "/Applications/Blender.app/Contents/Resources/4.2/python/lib/python3.11/subprocess.py", line 1950, i
n _execute_child
    raise child_exception_type(errno_num, err_msg, err_filename)
FileNotFoundError: [Errno 2] No such file or directory: '/Applications/Blender.app/Contents/Resources/4.2/p
ython/bin/python'

>>> python_exec = os.path.join(sys.prefix, 'bin', 'python3.11')
>>> subprocess.call([python_exec, "-m", "pip", "install", "--upgrade", "pip"])
0

>>> subprocess.call([python_exec, "-m", "pip", "install", "--upgrade", "scipy"])
0

>>> import scipy
```



### Create objects with data method

The data method closely mimics how data are stored internally in Blender. Add the data, and then the object. For a mesh:

```
me = bpy.data.meshes.new(meshName)  
ob = bpy.data.objects.new(obName, me)
```

Link the object to the current scene and make it active. Optionally, we can make the newly created object active or selected. This code is the same for all kinds of objects.

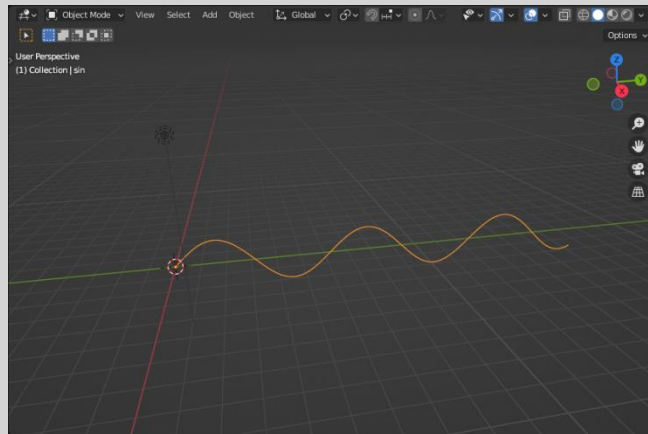
```
bpy.context.scene.collection.objects.link( ob )  
bpy.context.view_layer.objects.active = ob  
ob.select_set(True)
```



```
import bpy
from math import sin

n = 100                # Number of vertices
vertices = []          # Add n vertices
edges = []             # Add n-1 edges

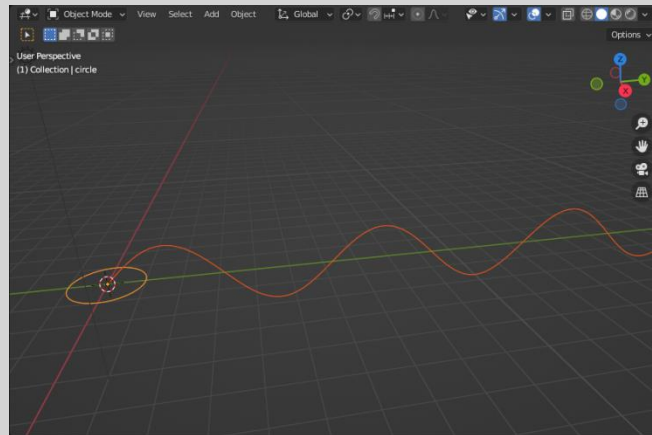
# Generate 100 y values
yVals = [y*0.18 for y in range(0, n)]
# Iterate over y values and generate 3D vertex coordinates
for i, y in enumerate( yVals ):
    vertices.append(( 0, y, sin( y ) ) )    # Set (x,y,z) vertex coordinate
    if i < n - 1:
        edges.append(( i, i+1 ) )           # Set edge vertices => this vertex and the next
# Generate a new mesh
me = bpy.data.meshes.new( 'sin' )
# Create mesh
me.from_pydata( vertices, edges, ( ) )
# Generate an object to contain an instance of this mesh 'me'
ob = bpy.data.objects.new( 'sin', me )
# Link this virtual object to an actual scene (the active or 'context' scene)
bpy.context.scene.collection.objects.link( ob )
bpy.context.view_layer.objects.active = ob
ob.select_set(True)
```





```
import bpy
from math import sin, cos

n = 36 # Number of vertices
vertices = [] # Add n vertices
edges = [] # Add n-1 edges
# Generate 36 y values
yVals = [y*0.18 for y in range(0, n)]
# Iterate over y values and generate 3D vertex coordinates
for i, y in enumerate ( yVals ):
    vertices.append((cos( y ), sin( y ), 0 )) # Set (x, y, z)
    if i < n - 1:
        # Set edge vertices => this vertex and the next
        edges.append( ( i, i+1 ))
# Generate a new mesh
me = bpy.data.meshes.new( 'circle' )
# Create mesh
me.from_pydata( vertices, edges, ( ) )
# Generate an object to contain an instance of this mesh 'me'
ob = bpy.data.objects.new( 'circle', me )
# Link this virtual object to an actual scene (the active or 'context' scene)
bpy.context.scene.collection.objects.link( ob )
bpy.context.view_layer.objects.active = ob
ob.select_set(True)
```





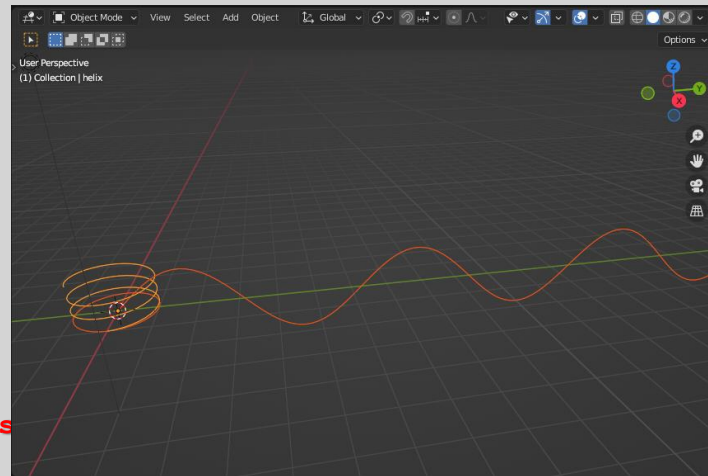


```
import bpy
from math import sin, cos

# Generate 100 y values
n = 100                                # Number of vertices
verts = []                             # n vertices
edges = []                             # n-1 edges
yVals = [y*0.18 for y in range(0, n)]
zVals = [z*0.01 for z in range(0, n)]

# Iterate over y values and generate 3D vertex coordinates
for i, y in enumerate( yVals ):
    verts.append(( cos( y ), sin( y ), zVals[i]))    # Set (x,y,z) vertex coordinate
    if i < n - 1:
        # Set edge vertices => this vertex and the next
        edges.append(( i, i+1 ))

# Generate a new mesh
mesh = bpy.data.meshes.new( 'helix' )
# Create mesh
mesh.from_pydata( verts, edges, ( ) )
# Generate an object to contain an instance of this mesh 'mesh'
ob = bpy.data.objects.new( 'helix', mesh )
# Link this virtual object to an actual scene (the active or 'context' scene)
bpy.context.collection.objects.link(ob)
ob.select_set(True)
```







## Blender Mesh Definition

### Simple Mesh Definition 4-Corner Plane

```
import bpy
```

```
#Define vertices, faces
```

```
#The vertex array contains 4 items with X, Y, and Z definitions
```

```
verts = [(0,0,0), (0,5,0), (5,5,0), (5,0,0)]
```

```
# the faces array contains 1 item.
```

```
# The number sequence refers to the vertex array items.
```

```
# The order will determine how the face is constructed.
```

```
faces = [(0,1,2,3)]
```

```
#Define mesh and object
```

```
me = bpy.data.meshes.new('Plane')
```

```
#the mesh variable is then referenced by the object variable
```

```
ob = bpy.data.objects.new('Plane', me)
```

```
#Set location and scene of object
```

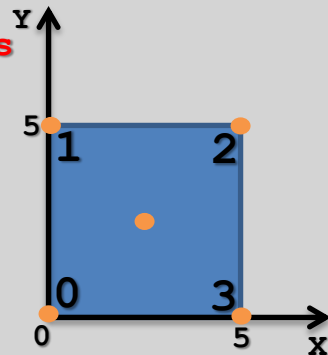
```
ob.location = bpy.context.scene.cursor.location # the cursor location
```

```
bpy.context.collection.objects.link(ob) # linking the object to the collection
```

```
#Create mesh
```

```
me.from_pydata(verts, [], faces)
```

```
me.update(calc_edges = True) # Calculate edges
```



No Edge Data





## Dictionaries

A **dictionary** is like a list, but more general. In a list, the indices have to be integers; in a dictionary they can be (almost) **any type**. You can think of a dictionary as a mapping between a set of indices (which are called **keys**) and a set of **values**. Each key maps to a value. The association of a key and a value is called a **key-value** pair or sometimes an item.

The squiggly-brackets, **{ }**, represent an empty dictionary



## Dictionary vs List

Values in lists are accessed by means of integers called indices, which indicate where in the list a given value is found.

Dictionaries access values by means of integers, strings, or other Python objects called keys, which indicate where in the dictionary a given value is found.

Both lists and dictionaries can store objects of any type.

```
>>> x = [ ]
```

```
>>> y = { }
```



## Dictionaries

As an example, we'll build a dictionary that maps from **English to Spanish** words, so the keys and the values are all strings. The function **dict** creates a new dictionary with no items. Because **dict** is the name of a built-in function, you should avoid using it as a variable name.

```
>>> eng2sp = dict()  
>>> print(eng2sp)  
{}  
>>> eng2sp['one'] = 'uno'
```

This line creates an item that maps from the key **'one'** to the value **'uno'**.



```
>>> print(eng2sp)  
{ 'one': 'uno' }
```

This output format is also an input format. For example, you can create a new dictionary with three items:

```
>>> eng2sp = { 'one': 'uno', 'two': 'dos', 'three': 'tres' }
```

```
>>> print(eng2sp)  
{ 'two': 'dos', 'three': 'tres', 'one': 'uno' }
```

The order of the key-value pairs is not the same. In fact, if you type the same example on your computer, you might get a different result. In general, the order of items in a dictionary is unpredictable.



But that's not a problem because the elements of a dictionary are never indexed with integer indices. Instead, you use the keys to look up the corresponding values:

```
>>> print(eng2sp['two'])
```

```
dos
```

If the key isn't in the dictionary, you get an exception:

```
>>> print(eng2sp['four'])
```

```
Traceback (most recent call last):
```

```
  File "<pyshell#24>", line 1, in <module>
```

```
    print(eng2sp['four'])
```

```
KeyError: 'four'
```



## Dictionaries

The `len` function works on dictionaries; it returns the number of key-value pairs:

```
>>> len(eng2sp)
3
```

The `in` operator works on dictionaries; it tells you whether something appears as a *key* in the dictionary.

```
>>> 'one' in eng2sp
True
>>> 'uno' in eng2sp
False
```



## Dictionaries

To see whether something appears as a value in a dictionary, you can use the method `values`, which returns the values as a list, and then use the `in` operator:

```
>>> vals = eng2sp.values()  
>>> 'uno' in vals  
True
```

The `in` operator uses different algorithms for lists and dictionaries.





You can obtain all the keys in the dictionary with the `keys` method.

```
>>> list(eng2sp.keys())  
['two', 'three', 'one']
```

It's also possible to obtain all the values stored in a dictionary, using `values`:

```
>>> list(eng2sp.values())  
['dos', 'tres', 'uno']
```

You can use the `items` method to return all keys and their associated values as a sequence of tuples:

```
>>> list(eng2sp.items())  
[('two', 'dos'), ('three', 'tres'), ('one', 'uno')]
```



You can obtain a copy of a dictionary using the copy method:

```
>>> x = {0: 'zero', 1: 'one'}  
>>> y = x.copy()  
>>> y  
{0: 'zero', 1: 'one'}
```

This makes a shallow copy of the dictionary. This will likely be all you need in most situations. The update method updates a first dictionary with all the key/value pairs of a second dictionary. For keys that are common to both, the values from the second dictionary override those of the first:

```
>>> z = {1: 'One', 2: 'Two'}  
>>> x = {0: 'zero', 1: 'one'}  
>>> x.update(z)  
>>> x  
{0: 'zero', 1: 'One', 2: 'Two'}
```



## Dictionaries and Lists

```
>>> d = { 'deniz': 'mavi', 'ağaç': 'yeşil', 'ateş': 'kırmızı' }
```

```
>>> for k in d:  
    print(k)
```

```
deniz  
ağaç  
ateş
```

```
>>> for k in d:  
    print(k, '-->', d[k])
```

```
ağaç --> yeşil  
deniz --> mavi  
ateş --> kırmızı
```



```
>>> for k, v in d.items():  
    print(k, '-->', v)
```

```
ağaç --> yeşil  
deniz --> mavi  
ateş --> kırmızı
```

```
>>> names = ['deniz', 'ağaç', 'ateş']  
>>> colors = ['mavi', 'yeşil', 'kırmızı']
```

```
>>> d = dict(zip(names, colors))
```

```
>>> print(d)  
{ 'ateş': 'kırmızı', 'ağaç': 'yeşil', 'deniz': 'mavi' }
```