MBSI Python Coding Workshop 4

Functions & Libraries

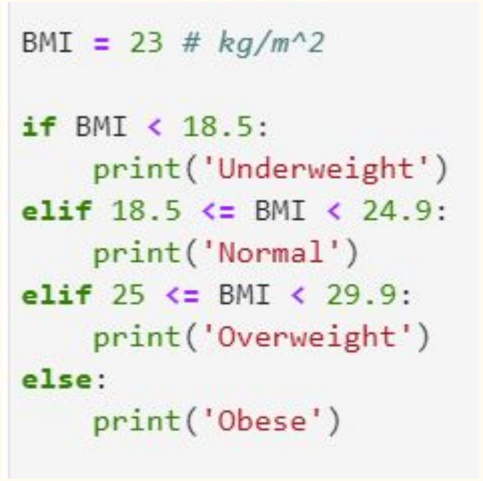
Revision questions!

pollev.com/danielgan042

Revision Questions!

Which one of the following is true?

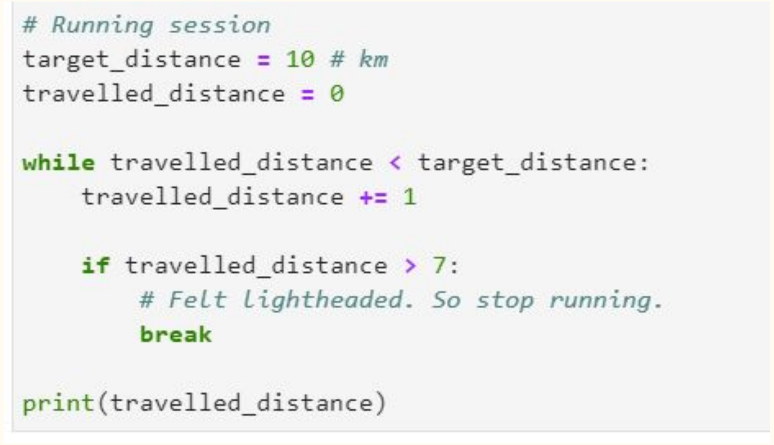
a) 1 and 0 == True
b) 10 >= 15
[✓ c) 1 != False]



```python
BMI = 23 # kg/m^2

if BMI < 18.5:
    print('Underweight')
elif 18.5 <= BMI < 24.9:
    print('Normal')
elif 25 <= BMI < 29.9:
    print('Overweight')
else:
    print('Obese')
```

What would be printed from the code block to the left?

a) Underweight
[✓ b) Normal]
c) Overweight
d) Obese



```python
# Running session
target_distance = 10 # km
travelled_distance = 0

while travelled_distance < target_distance:
    travelled_distance += 1

    if travelled_distance > 7:
        # Felt lightheaded. So stop running.
        break

print(travelled_distance)
```

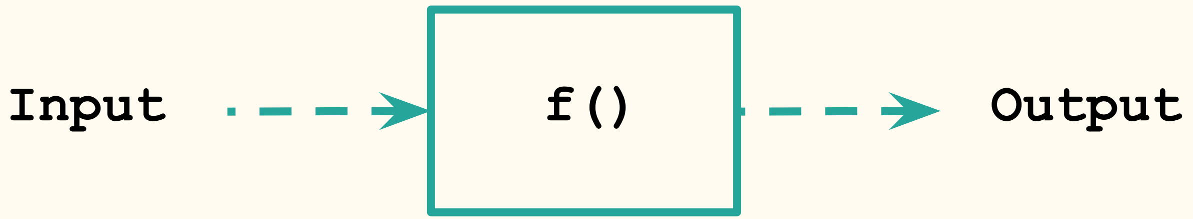
What is the travelled_distance at the end of the code block above?

a) 5
b) 7
[✓ c) 8]
d) Too far

4.1 Functions

Functions

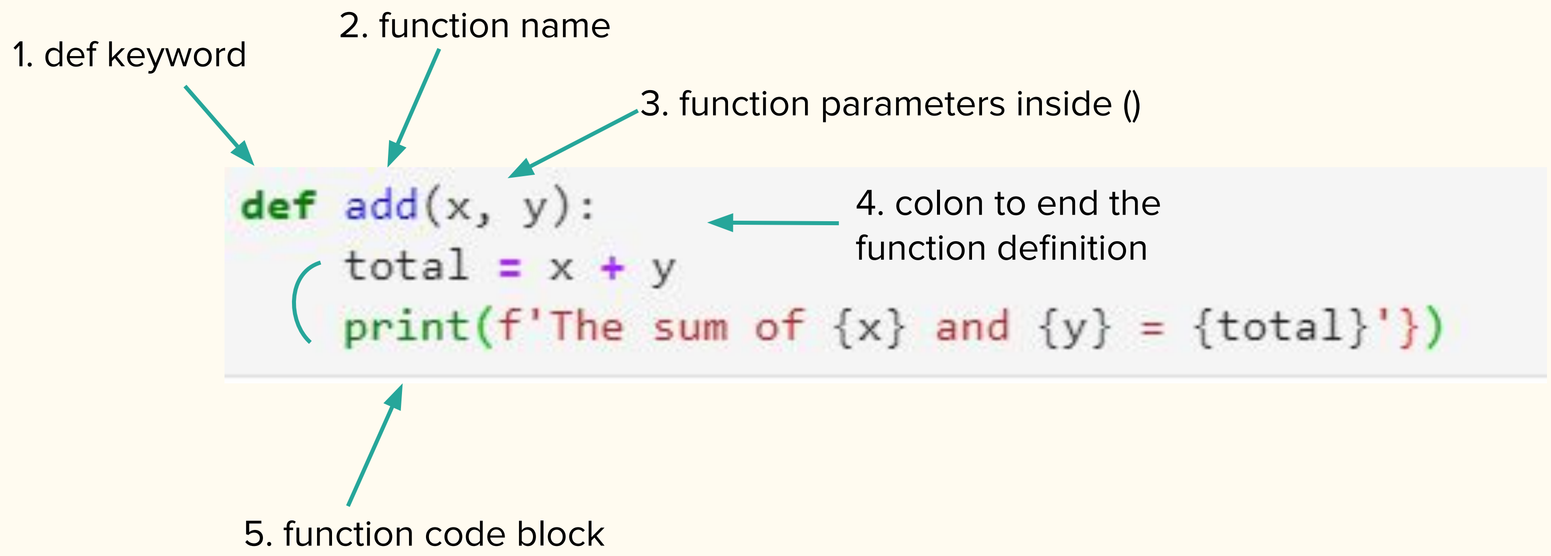
A function is a block of code that only runs when **called.**



What functions have we seen already?

- print()
- help()
- input()
- str()
- int()
- float()
- round()
- bool()
- type()

Defining functions



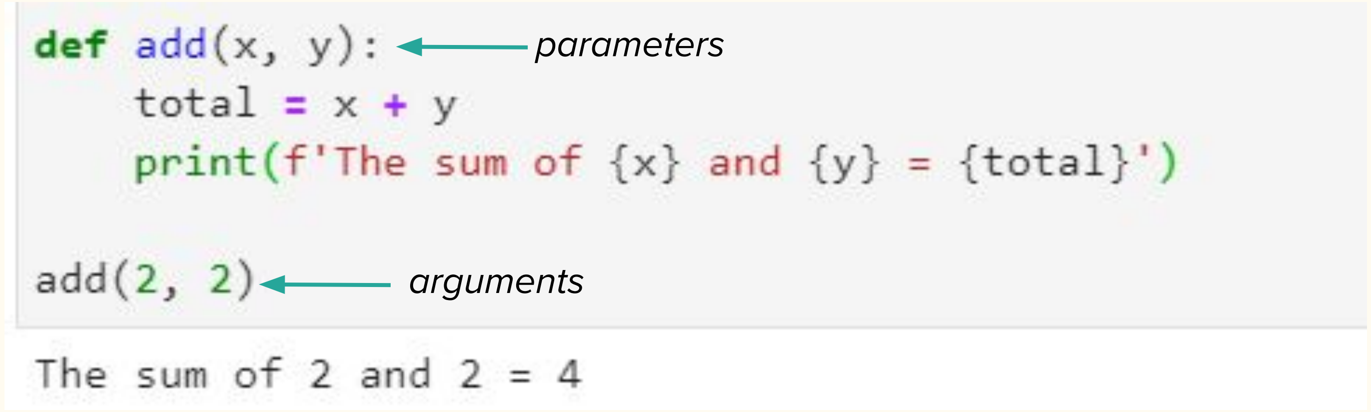
```
def add(x, y):
    total = x + y
    print(f'The sum of {x} and {y} = {total}')
```

1. def keyword
2. function name
3. function parameters inside ()
4. colon to end the function definition
5. function code block

Defining functions

Parameters are variables listed within the () of the function definition

Arguments are the actual values sent as inputs to the function when called.



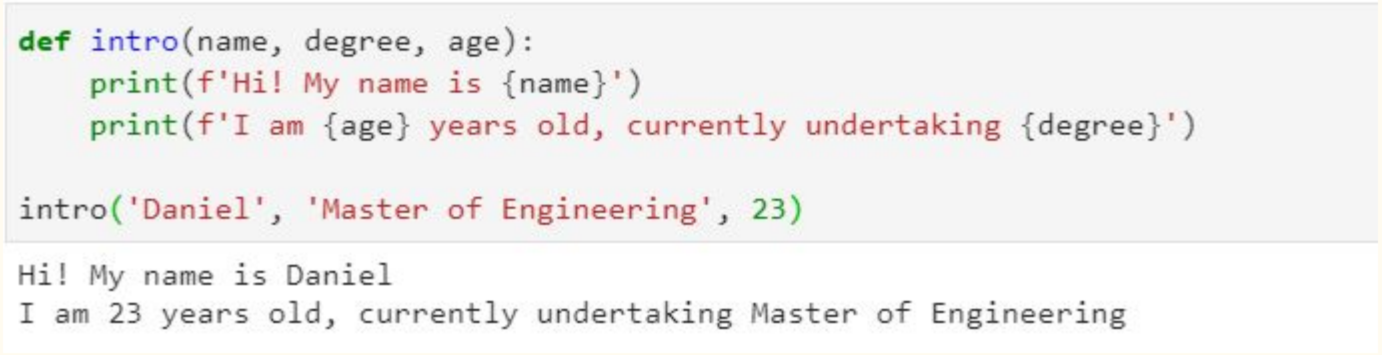
```
def add(x, y):
    total = x + y
    print(f'The sum of {x} and {y} = {total}')

add(2, 2)
```

```
The sum of 2 and 2 = 4
```

Functions with multiple parameters

Parameters must be separated by commas.

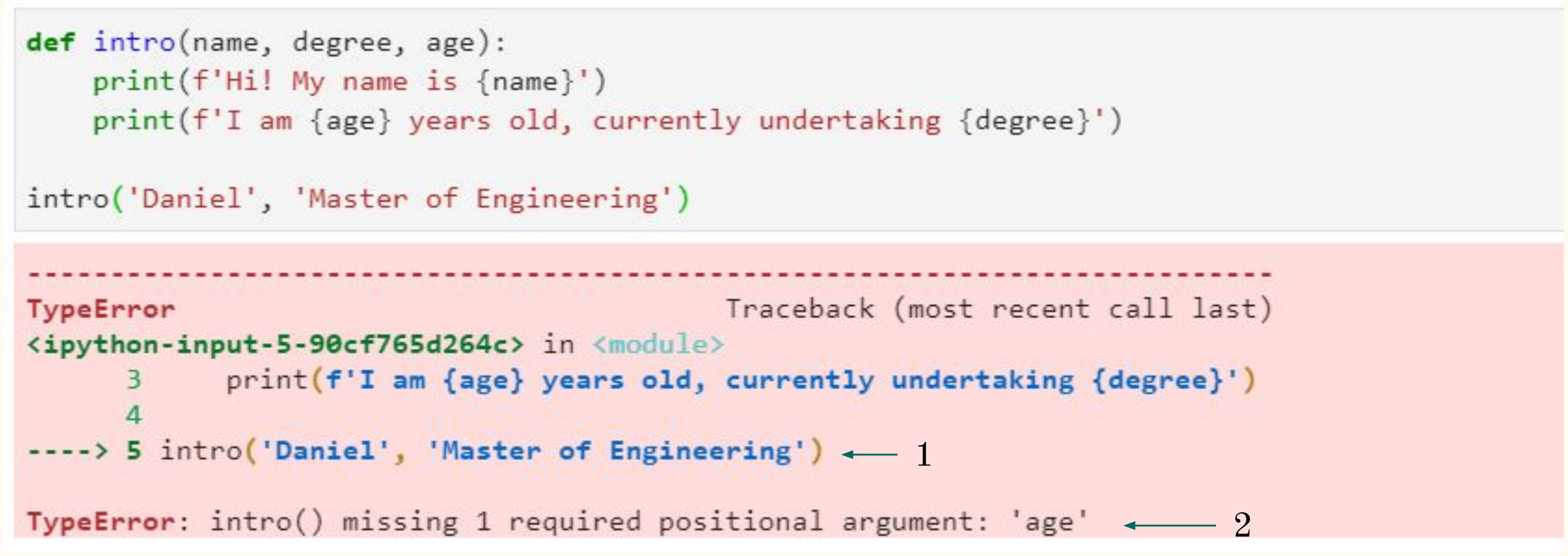
The function must be called with the correct # of arguments. (3 in this case)



```python
def intro(name, degree, age):
    print(f'Hi! My name is {name}')
    print(f'I am {age} years old, currently undertaking {degree}')

intro('Daniel', 'Master of Engineering', 23)
```

```
Hi! My name is Daniel
I am 23 years old, currently undertaking Master of Engineering
```

Functions with multiple parameters cont.



```
def intro(name, degree, age):
    print(f'Hi! My name is {name}')
    print(f'I am {age} years old, currently undertaking {degree}')

intro('Daniel', 'Master of Engineering')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-5-90cf765d264c> in <module>
      3     print(f'I am {age} years old, currently undertaking {degree}')
      4 
----> 5 intro('Daniel', 'Master of Engineering')

TypeError: intro() missing 1 required positional argument: 'age'
```

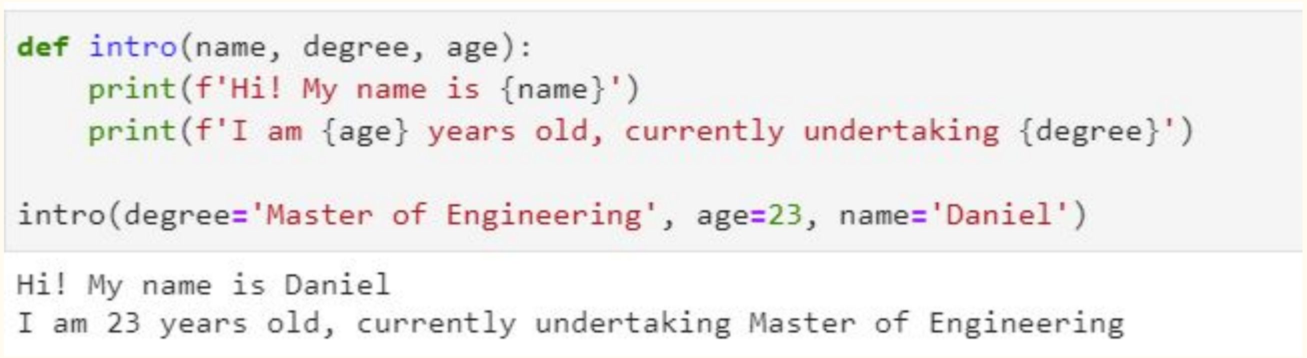
$\longleftarrow$ 1

$\longleftarrow$ 2

Keyword arguments

The arguments can also be sent as keywords when calling the function.

In this way, the order of the arguments doesn’t matter.



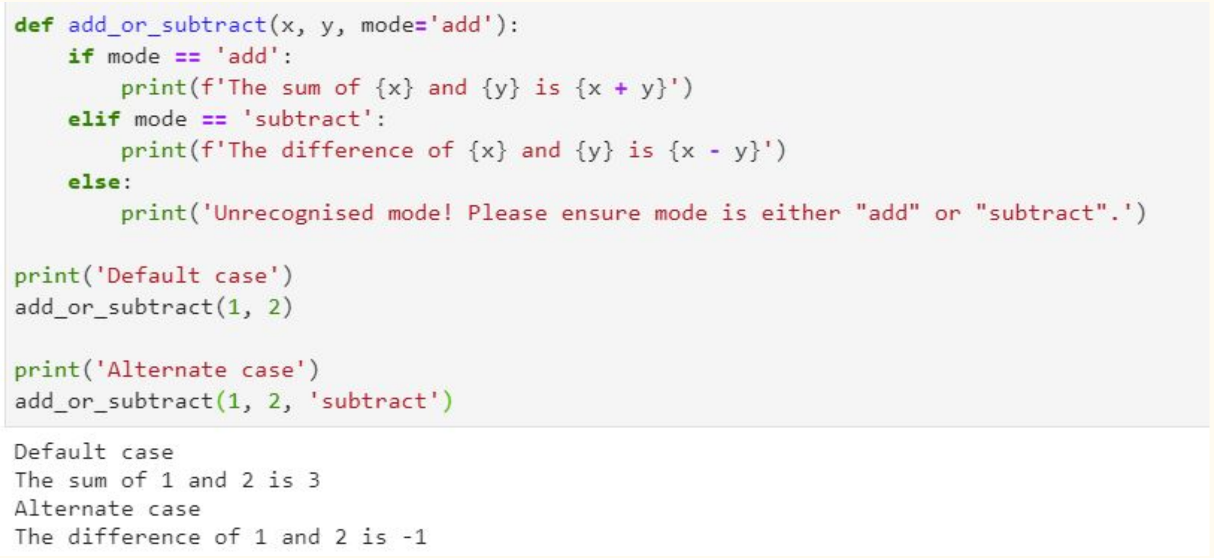
```python
def intro(name, degree, age):
    print(f'Hi! My name is {name}')
    print(f'I am {age} years old, currently undertaking {degree}')

intro(degree='Master of Engineering', age=23, name='Daniel')
```

```
Hi! My name is Daniel
I am 23 years old, currently undertaking Master of Engineering
```

Default arguments

There may be cases where a default behaviour is desired.

In these situations, we use what are known as *Default Arguments*.

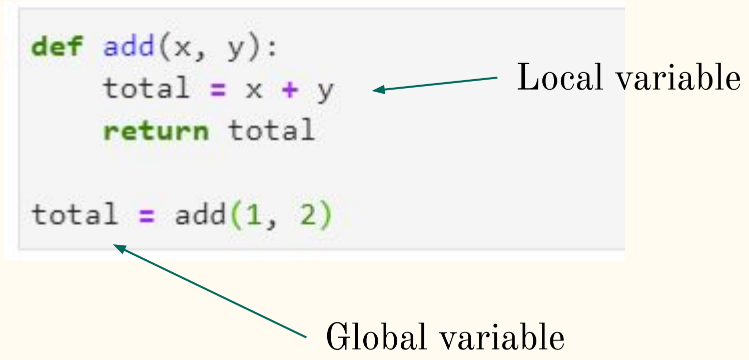


```python
def add_or_subtract(x, y, mode='add'):
    if mode == 'add':
        print(f'The sum of {x} and {y} is {x + y}')
    elif mode == 'subtract':
        print(f'The difference of {x} and {y} is {x - y}')
    else:
        print('Unrecognised mode! Please ensure mode is either "add" or "subtract".')

print('Default case')
add_or_subtract(1, 2)

print('Alternate case')
add_or_subtract(1, 2, 'subtract')
```

```
Default case
The sum of 1 and 2 is 3
Alternate case
The difference of 1 and 2 is -1
```

Returning values

A function can return a value using the `return` statement.



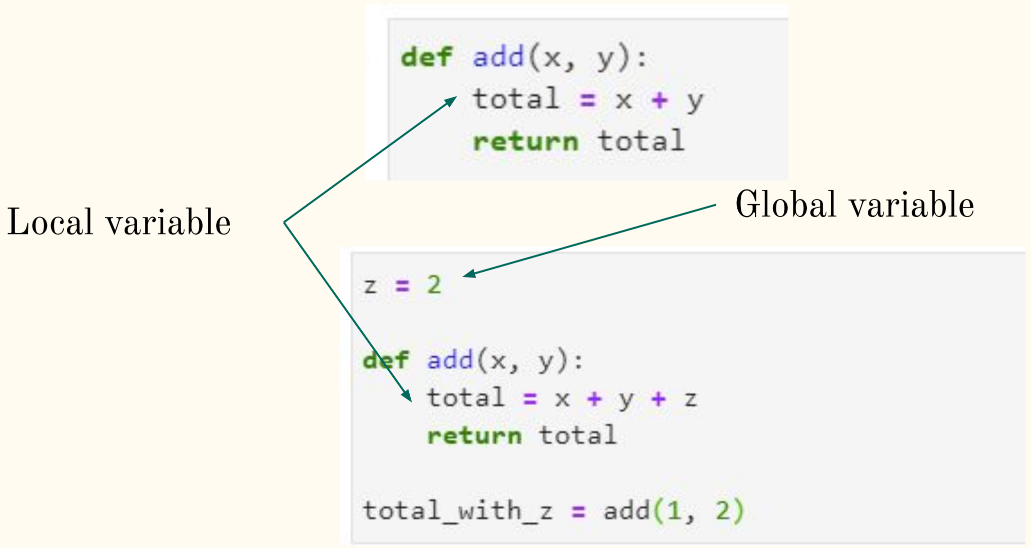
```python
def add(x, y):
    total = x + y
    return total

total = add(1, 2)
```

(The returned value must be printed to be seen.)

Variable scoping

Global variables = variables accessible by any code in the program

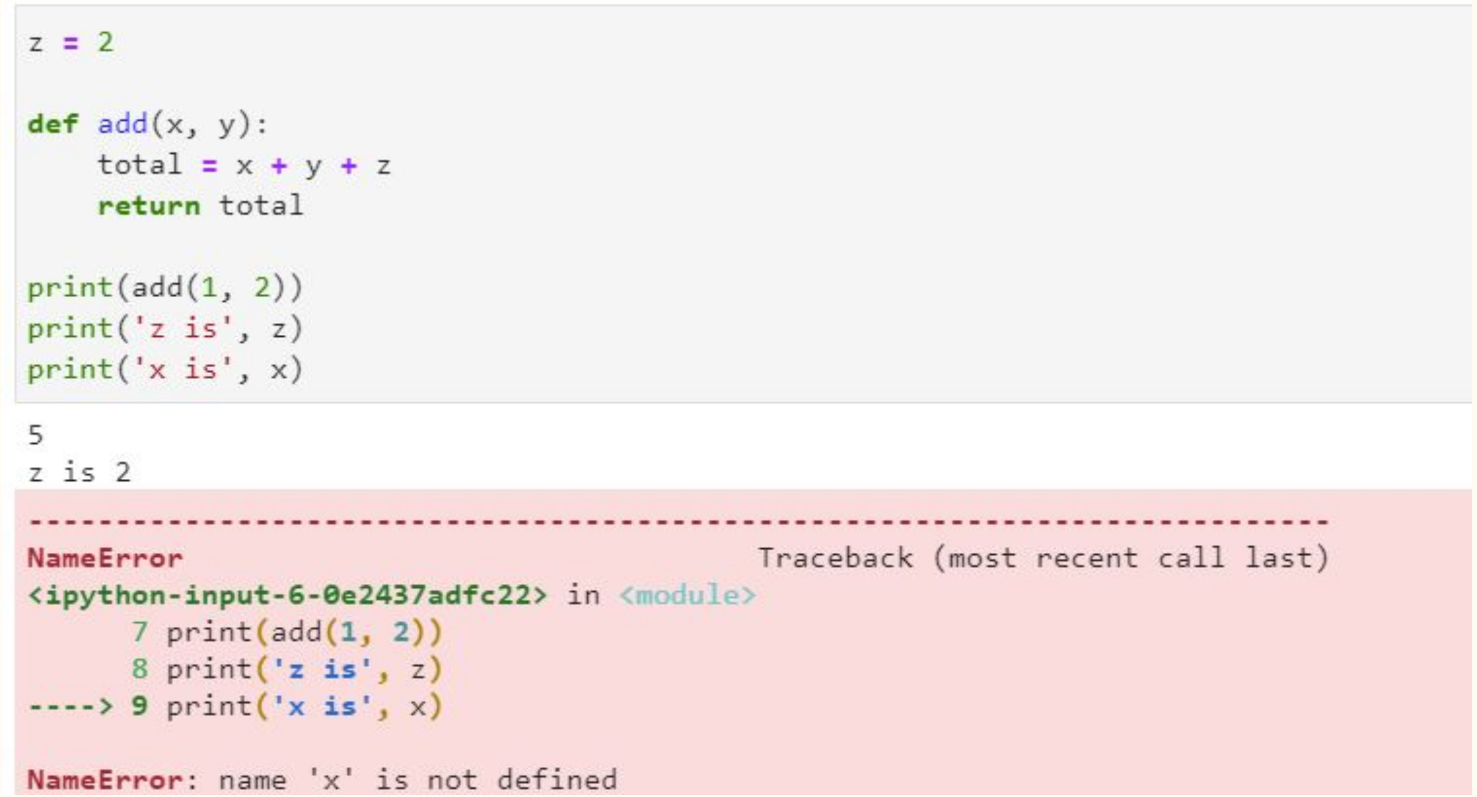
Local variables = variables accessible only within the *scope* of the variable



```
def add(x, y):
    total = x + y
    return total
```

```
z = 2

def add(x, y):
    total = x + y + z
    return total

total_with_z = add(1, 2)
```

Variable scoping cont.



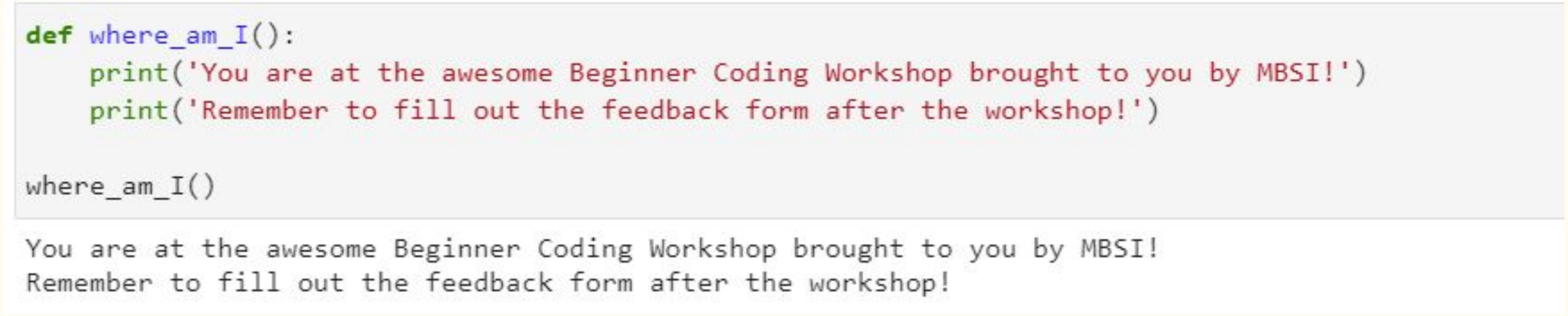
```
z = 2

def add(x, y):
    total = x + y + z
    return total

print(add(1, 2))
print('z is', z)
print('x is', x)
```

```
5
z is 2
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-6-0e2437adfc22> in <module>
      7 print(add(1, 2))
      8 print('z is', z)
----> 9 print('x is', x)

NameError: name 'x' is not defined
```

More on functions

A function can be defined without any parameters. (No input required)

You can also have a function that doesn't give any output. (No values returned)



```python
def where_am_I():
    print('You are at the awesome Beginner Coding Workshop brought to you by MBSI!')
    print('Remember to fill out the feedback form after the workshop!')

where_am_I()
```

```
You are at the awesome Beginner Coding Workshop brought to you by MBSI!
Remember to fill out the feedback form after the workshop!
```

4.2 Libraries

Do we need to code everything ourselves?

Can we use other people's code to make our development faster and easier?

Yes! That is the whole idea behind libraries.

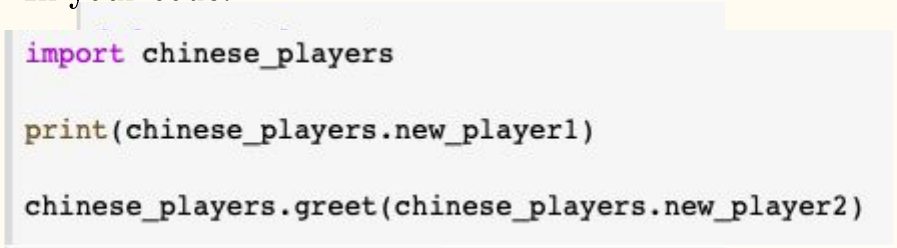
Monopoly expansion analogy

Simplest expansion -- Module

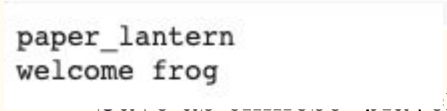
Chinese new year pieces

In python

In your code:



```python
import chinese_players

print(chinese_players.new_player1)

chinese_players.greet(chinese_players.new_player2)
```



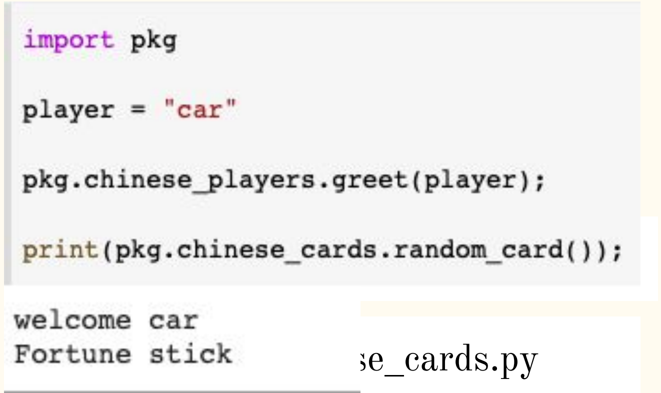
```
paper_lantern
welcome frog
```

Save as chinese_players.py

Collection of simple expansions -- package

Chinese new year player pieces, cards and dice.

In code:



```
import pkg

player = "car"

pkg.chinese_players.greet(player);

print(pkg.chinese_cards.random_card());
```

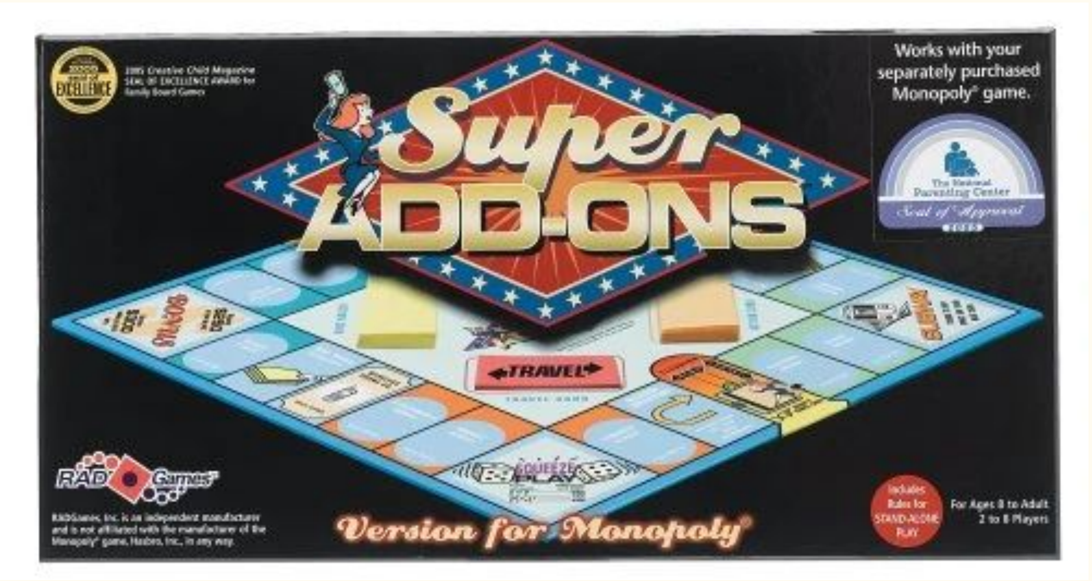
```
welcome car
Fortune stick
```

se_cards.py

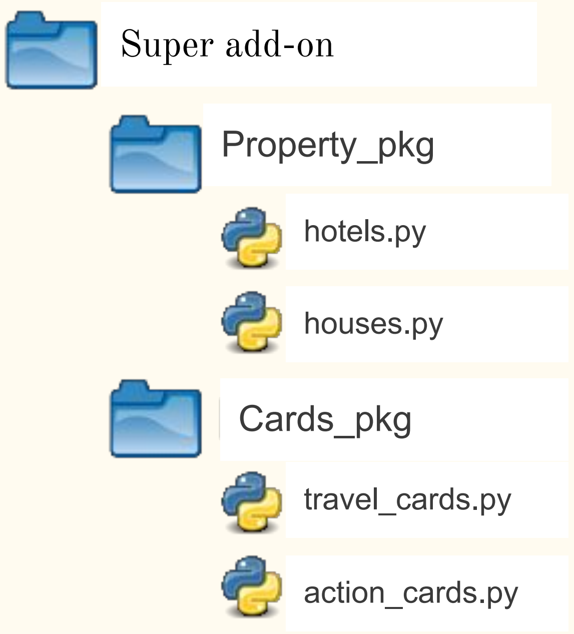
__init__.py

Bigger expansions -- Libraries

Super add-ons expansion



In python:



Same game but not really -- Frameworks

In Monopoly:

Collection of expansions making the game much different

In Python:

Collection of libraries

Definitions

Function/Variable	reusable code	Pieces in board game ↓
Module	file that contains Python functions and variables	More players expansion ↓
Package	collection of modules in a folder/directory	Chinese new year expansion ↓
Library	collection of packages	Super add on expansion ↓
Frameworks	collection of libraries	Mega monopoly

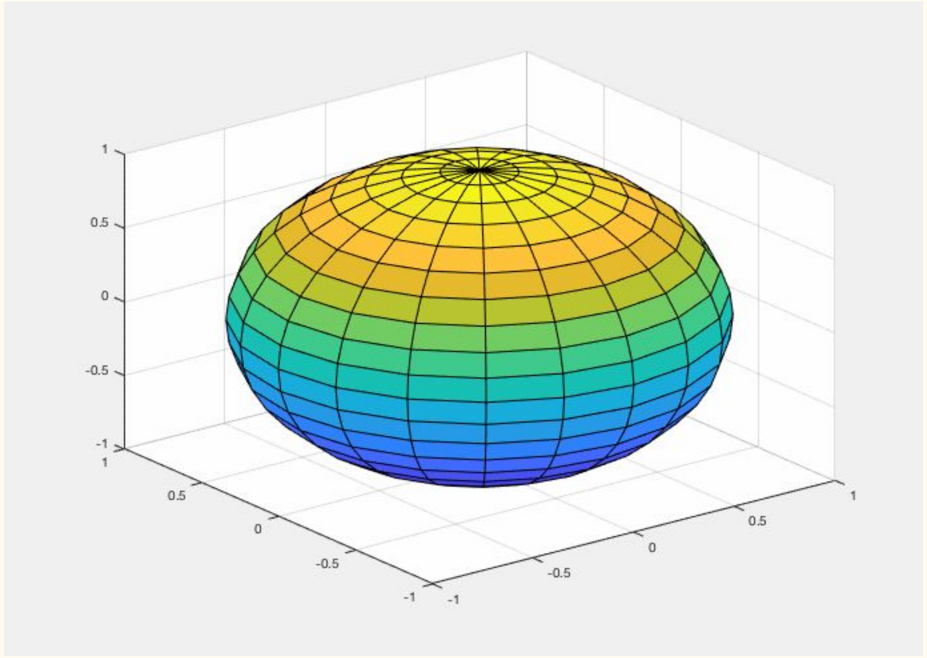
Modules that we will learn today

math — Mathematical functions (Module)

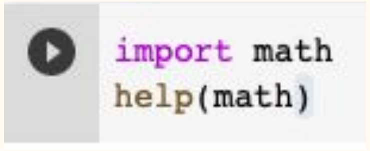
random — Generate random numbers (Module)

Libraries we will discuss today

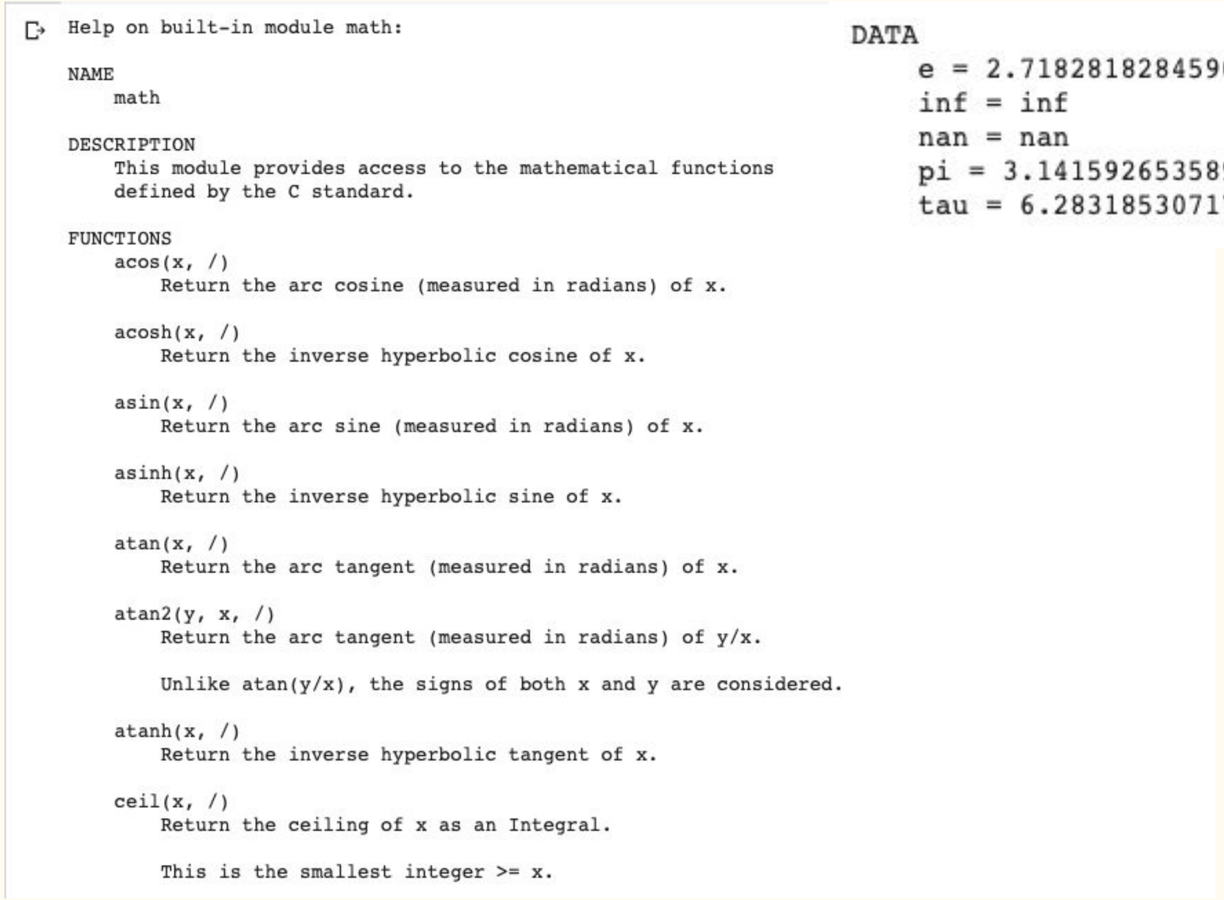
Matplotlib - Visualization and plots (Library)



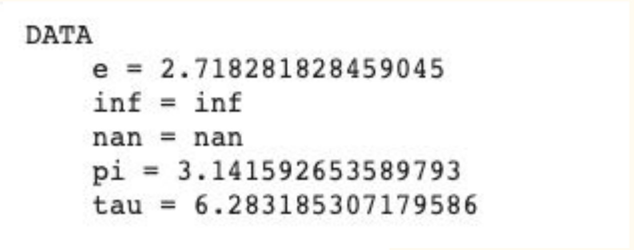
The math module



```python
import math
help(math)
```

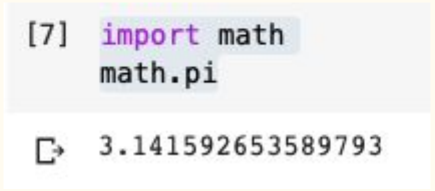


```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.
```

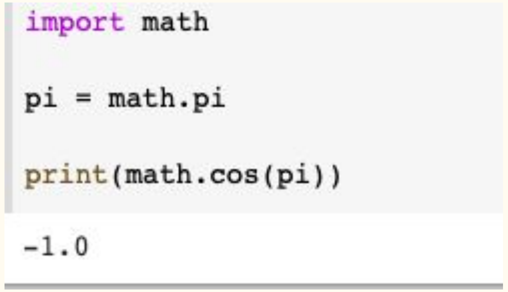


```
DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586
```

The math module

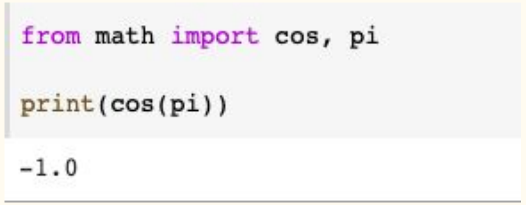


```
[7] import math
    math.pi

    3.141592653589793
```



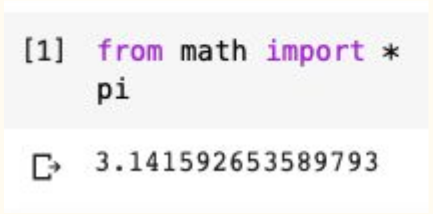
```
import math

pi = math.pi

print(math.cos(pi))
```

```
-1.0
```



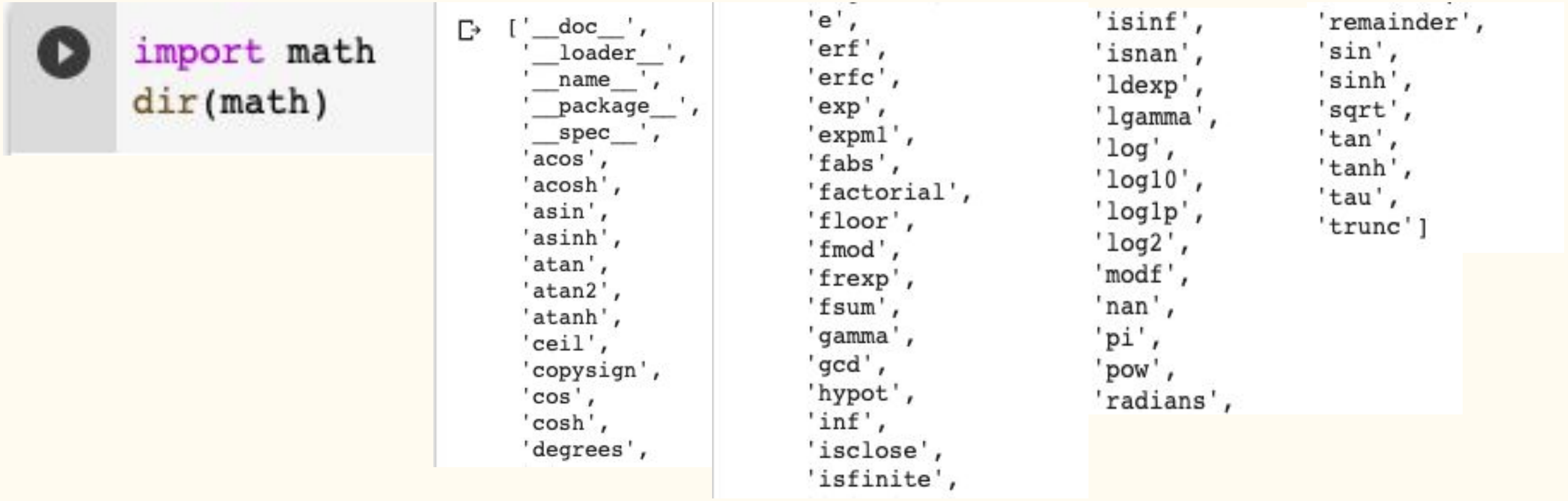
```
from math import cos, pi

print(cos(pi))
```

```
-1.0
```



```
[1] from math import *
    pi

    3.141592653589793
```

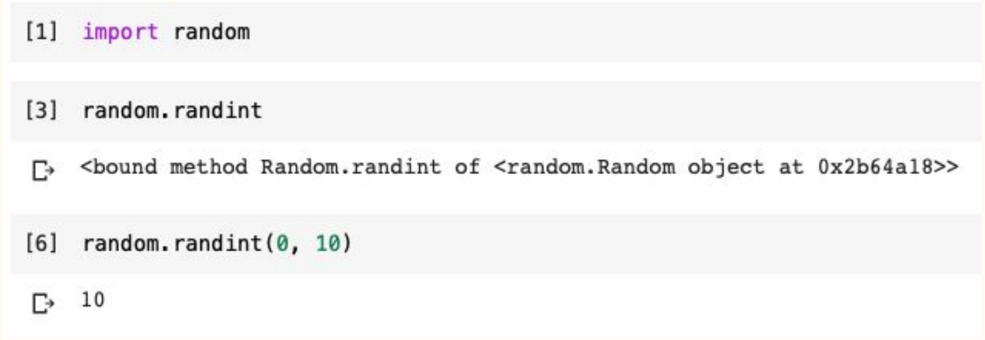
See all functions inside module



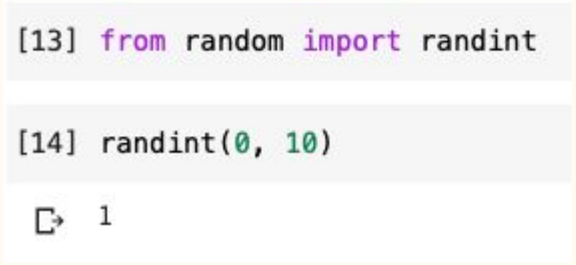
```
import math
dir(math)
```

```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'pi',
 'pow',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'tau',
 'trunc']
```

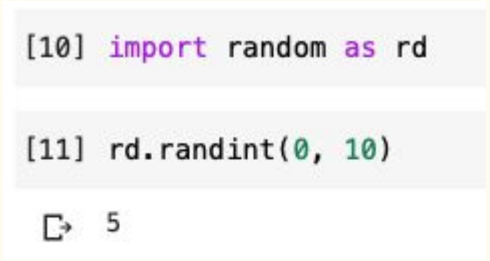
The random module



```
[1] import random

[3] random.randint

    <bound method Random.randint of <random.Random object at 0x2b64a18>>

[6] random.randint(0, 10)

    10
```

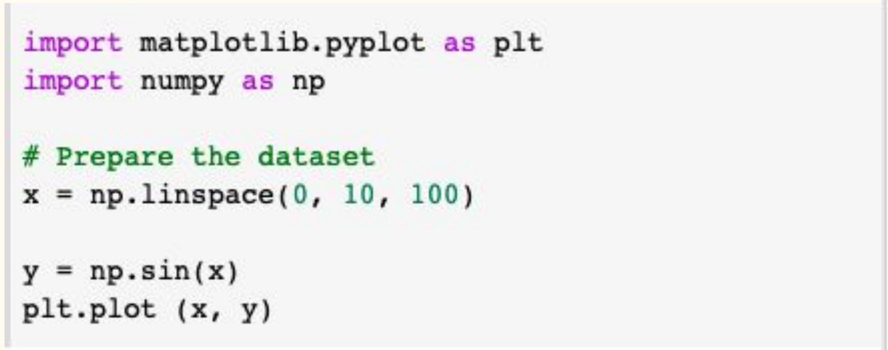


```
[13] from random import randint

[14] randint(0, 10)

     1
```

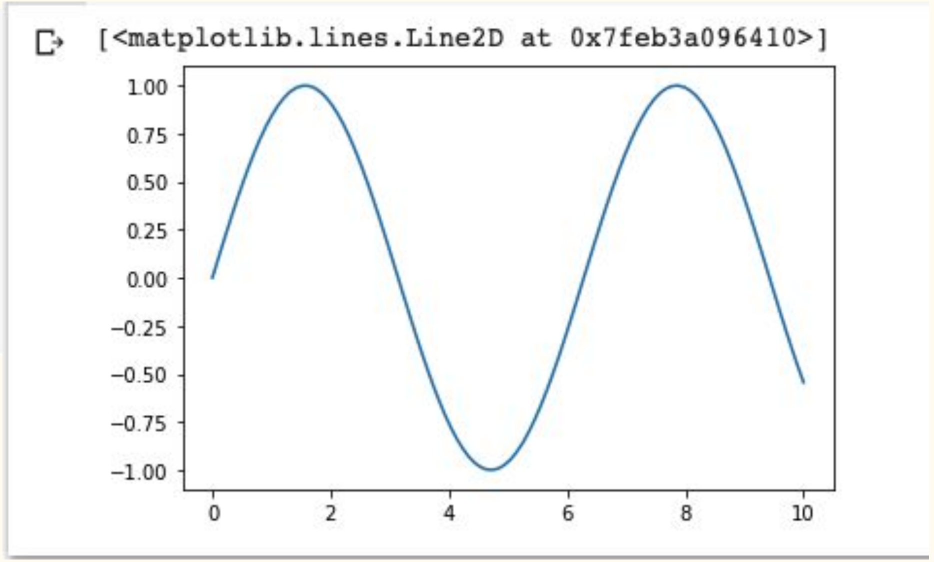


```
[10] import random as rd

[11] rd.randint(0, 10)

     5
```

The matplotlib library and pyplot



```
import matplotlib.pyplot as plt
import numpy as np

# Prepare the dataset
x = np.linspace(0, 10, 100)

y = np.sin(x)
plt.plot (x, y)
```

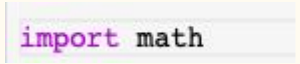


```
[<matplotlib.lines.Line2D at 0x7feb3a096410>]
```

How to use modules

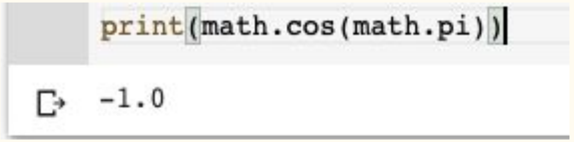
Recommended

Step 1- Import module by its name



```
import math
```

Step 2 - Use the "." operator to access the module's "methods"

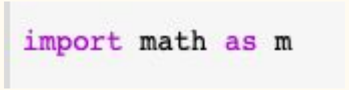


```
print(math.cos(math.pi))
-1.0
```

Or

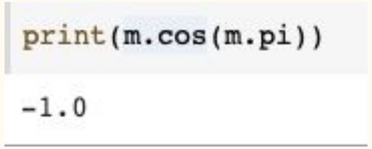
Recommended for big names

Step 1- Import module and give alias



```
import math as m
```

Step 2 - Use alias and "." to access the module's "methods"

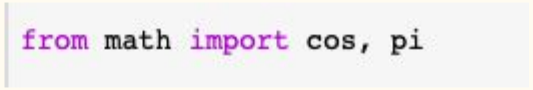


```
print(m.cos(m.pi))
-1.0
```

How to use modules

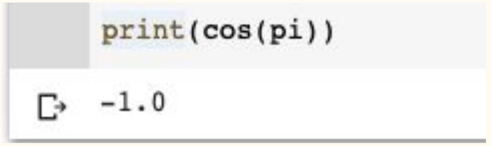
Recommended

Step 1- Import methods from module



```
from math import cos, pi
```

Step 2 - Use methods directly in code

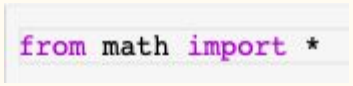


```
print(cos(pi))
-1.0
```

Or

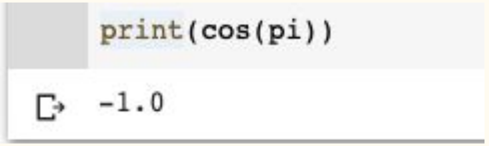
Not the best

Step 1- Import all methods in module



```
from math import *
```

Step 2 - Use methods directly in code



```
print(cos(pi))
-1.0
```

How to use libraries

Step 1- Import package or module from library.
Import <library.module>

```python
import matplotlib.pyplot as plt
```

Step 2 - Use alias and the "." access functions

```python
plt.plot (x, y)
```

Or

Alternative approach:

```python
from matplotlib.pyplot import plot

plot(x, y)
```

Popular libraries in Python

- os - miscellaneous operating system interfaces (Module)
- math - mathematical functions (Module)
- random - generate pseudo-random numbers (Module)
- time - provides various time-related functions (Module)
- string - common string operations (Module/built-in type with methods)
- numpy - manipulates large, multidimensional arrays and matrices with high-level mathematical functions (Library)
- scipy - scientific and technical computing for maths, science and engineering (Library)
- pandas - data manipulation and analysis in dataframes (Library)
- matplotlib - plotting for python and NumPy (Library)
- tensorflow - dataflow and differentiable programming (Library)

Breakout Time!

FEEDBACK FORM:

https://forms.gle/1GnsHhYUav7D281F8