Python Programming

TUYEN NGOC LE

Introduction

Outline

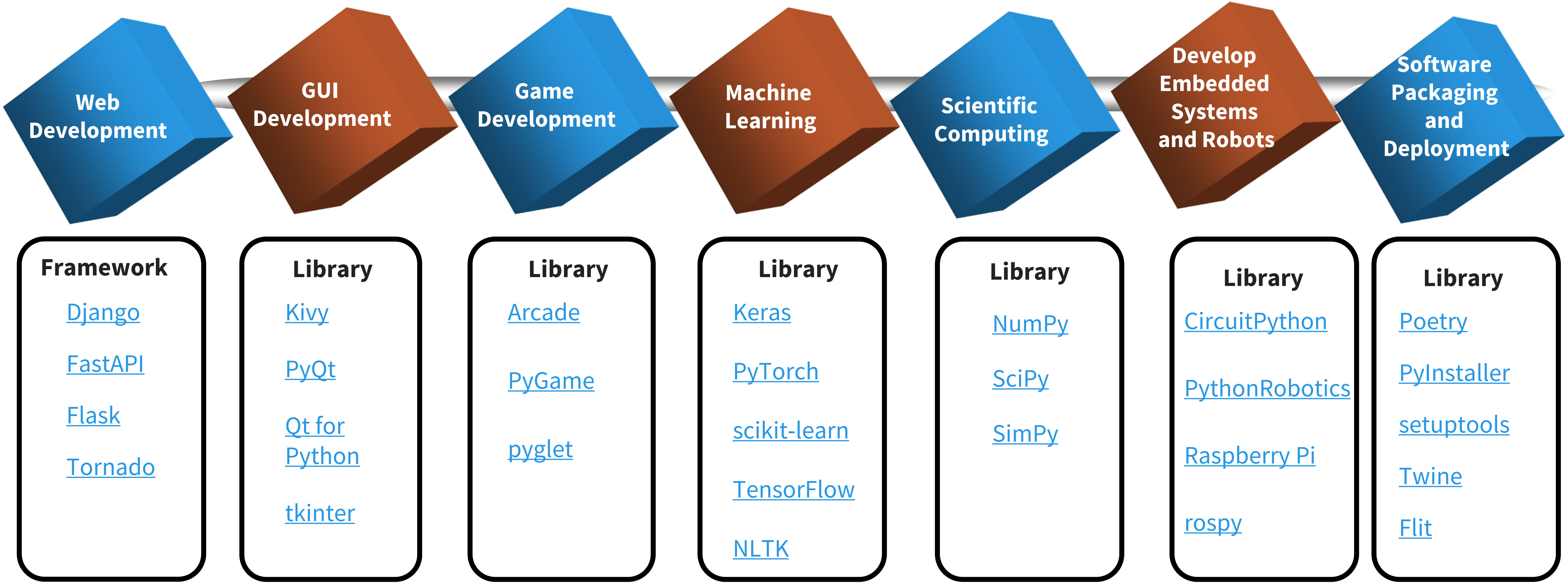
- ❑ **What is Python?**
- ❑ **What Can We Do With Python?**
- ❑ **Example: face detection**
- ❑ **Installing Python: Anaconda**
- ❑ **How to Get Started With Python? Your first Python Program**
- ❑ **References**

Guido van Rossum

What is Python?

- ❑ Python is a high-level, general-purpose programming language.
- ❑ Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis.
- ❑ Guido van Rossum designed python, which first appeared on 20 February 1991.
- ❑ **Did you know?** The name Python comes from Monty Python. When Guido van Rossum was creating Python, he read the scripts from BBC's Monty Python's Flying Circus. So, he thought the name Python was appropriately short and slightly mysterious.

What Can We Do With Python?



Web Development

Framework

- Django
- FastAPI
- Flask
- Tornado

GUI Development

Library

- Kivy
- PyQt
- Qt for Python
- tkinter

Game Development

Library

- Arcade
- PyGame
- pyglet

Machine Learning

Library

- Keras
- PyTorch
- scikit-learn
- TensorFlow
- NLTK

Scientific Computing

Library

- NumPy
- SciPy
- SimPy

Develop Embedded Systems and Robots

Library

- CircuitPython
- PythonRobotics
- Raspberry Pi
- rospy

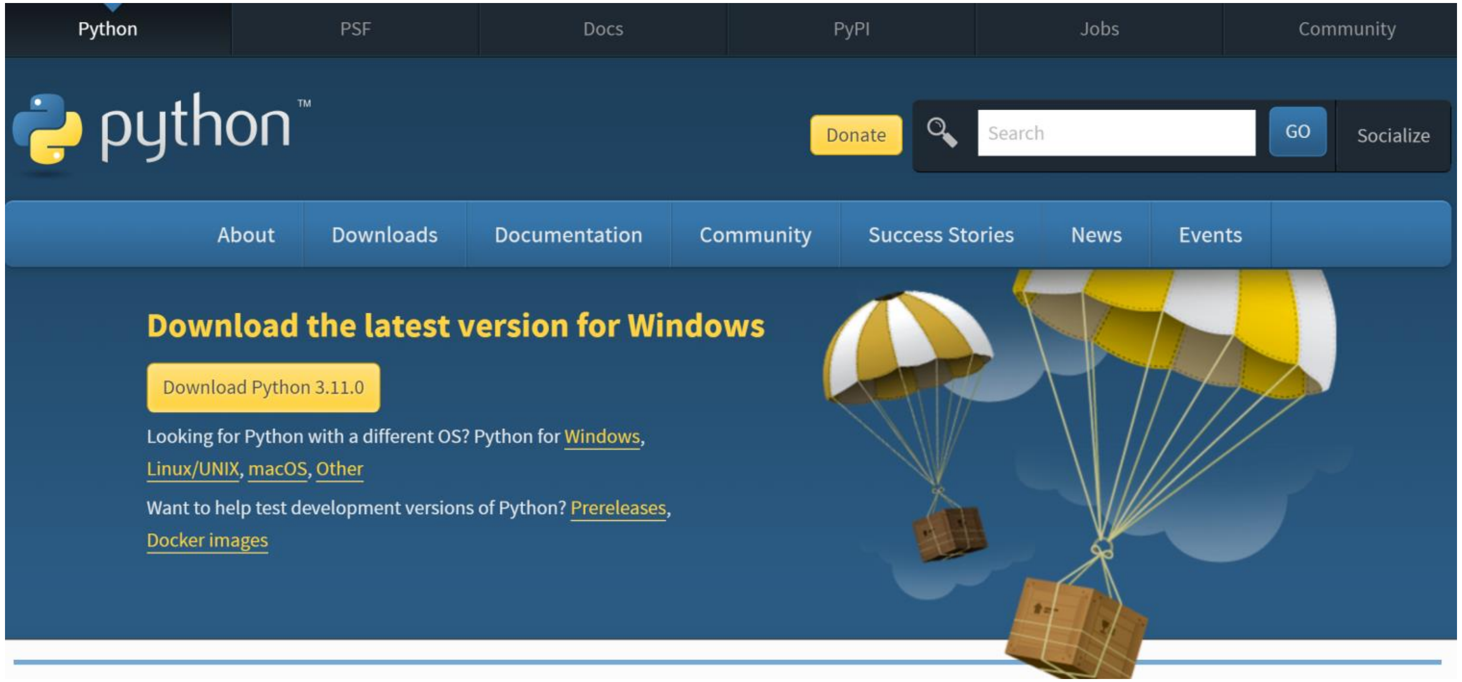
Software Packaging and Deployment

Library

- Poetry
- PyInstaller
- setuptools
- Twine
- Flit

Installing

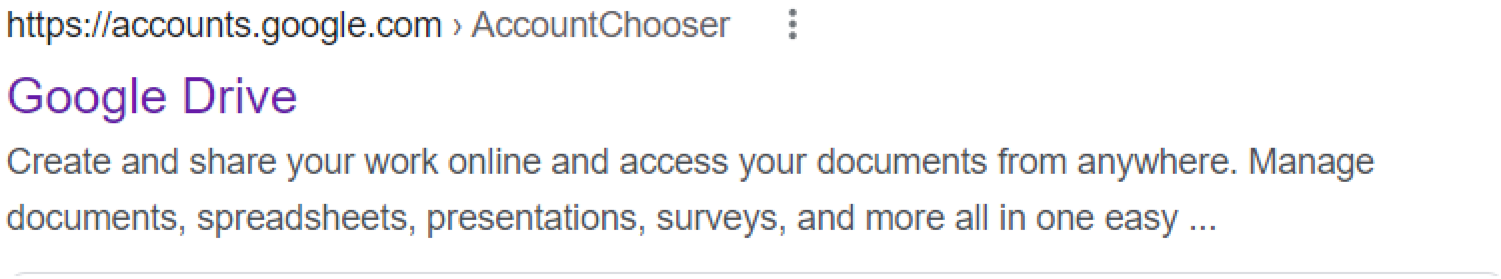
https://www.python.org/downloads/



Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, macOS, Other

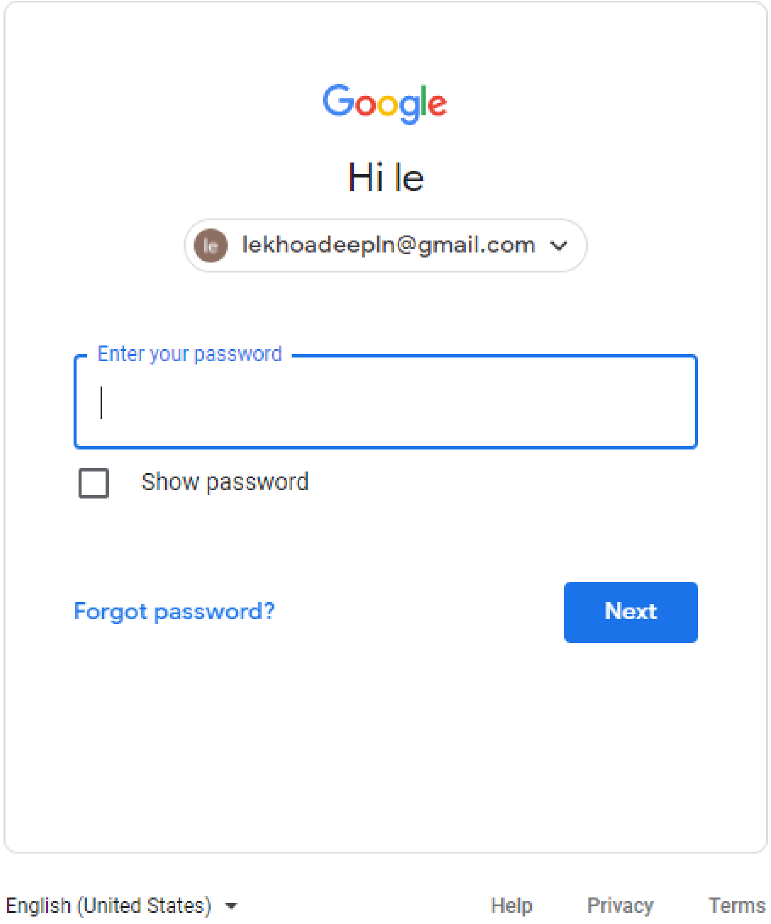
Want to help test development versions of Python? Prereleases, Docker images



https://accounts.google.com › AccountChooser

Google Drive

Create and share your work online and access your documents from anywhere. Manage documents, spreadsheets, presentations, surveys, and more all in one easy ...



Google

Hi le

lekhoadeepln@gmail.com

Enter your password

□ Show password

Forgot password?

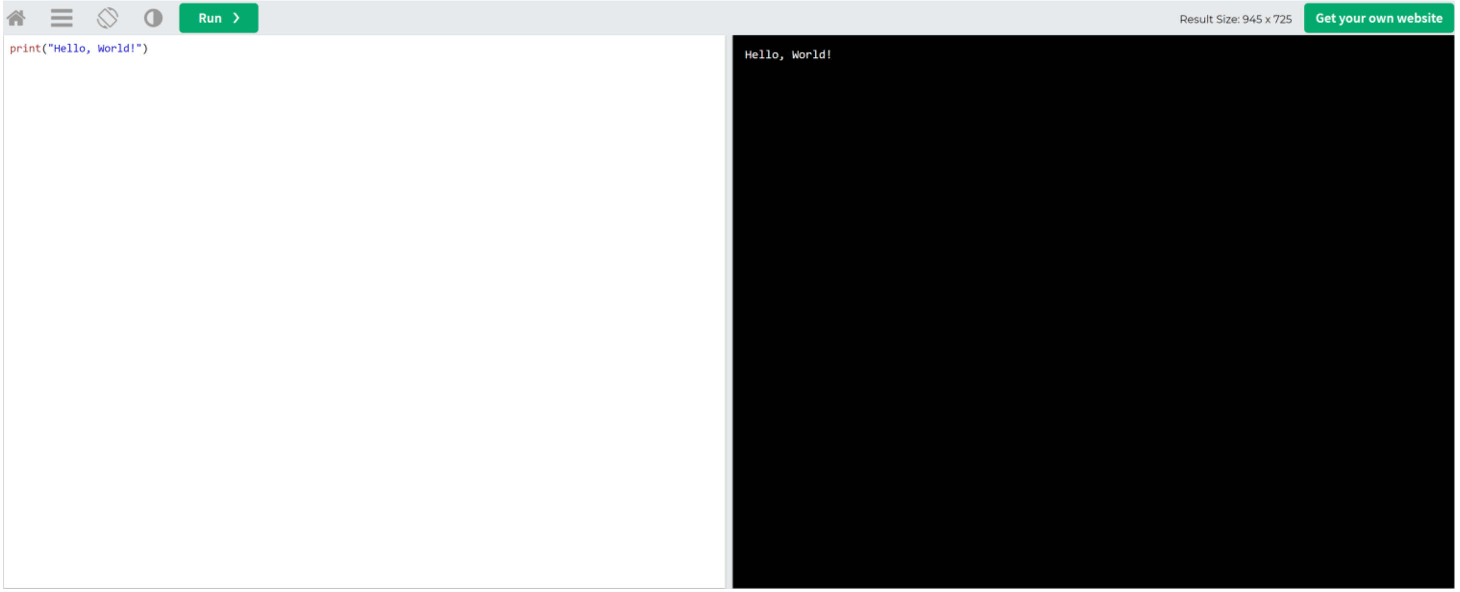
Next

Google Colab

Installing

Use online:

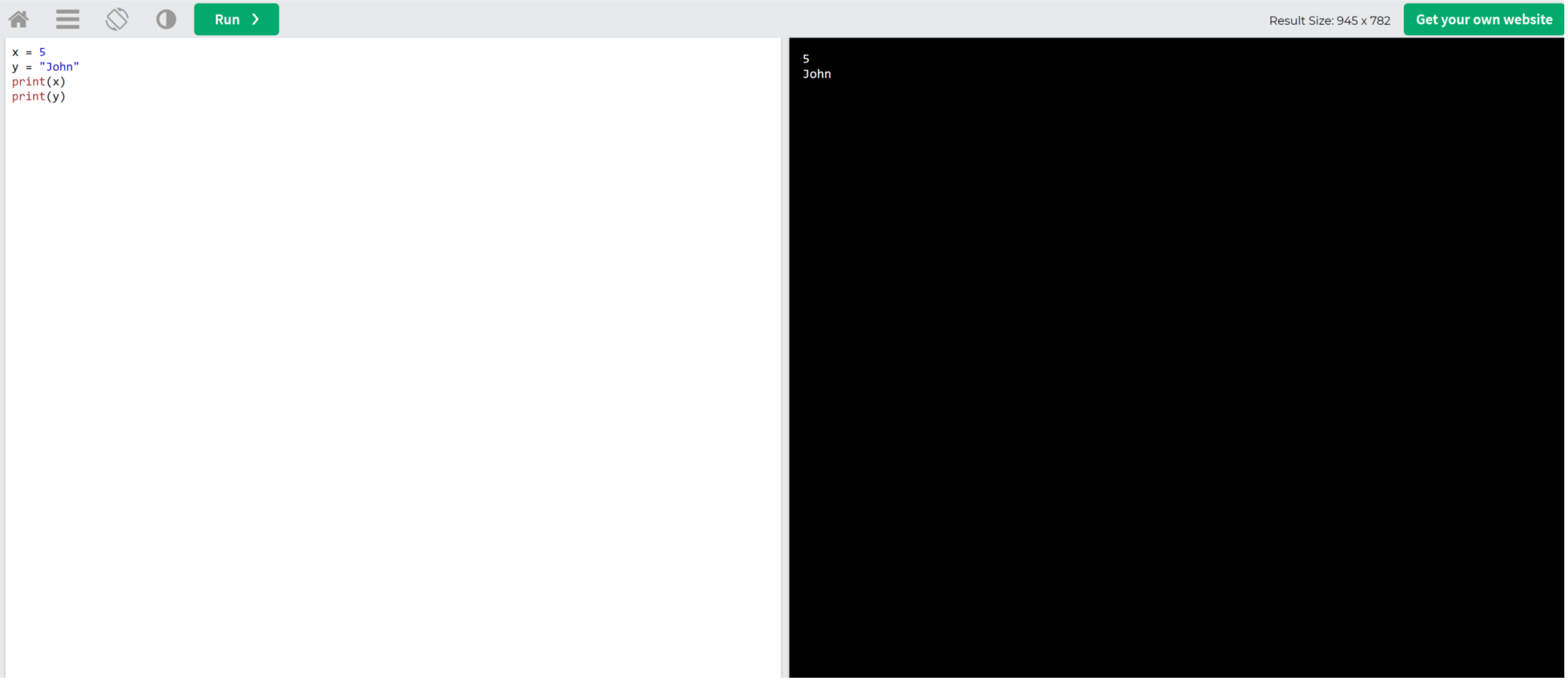
https://www.w3schools.com/python/trypython.asp?filename=demo_default



```
print("Hello, World!")
```

```
Hello, World!
```

Your first Python Program



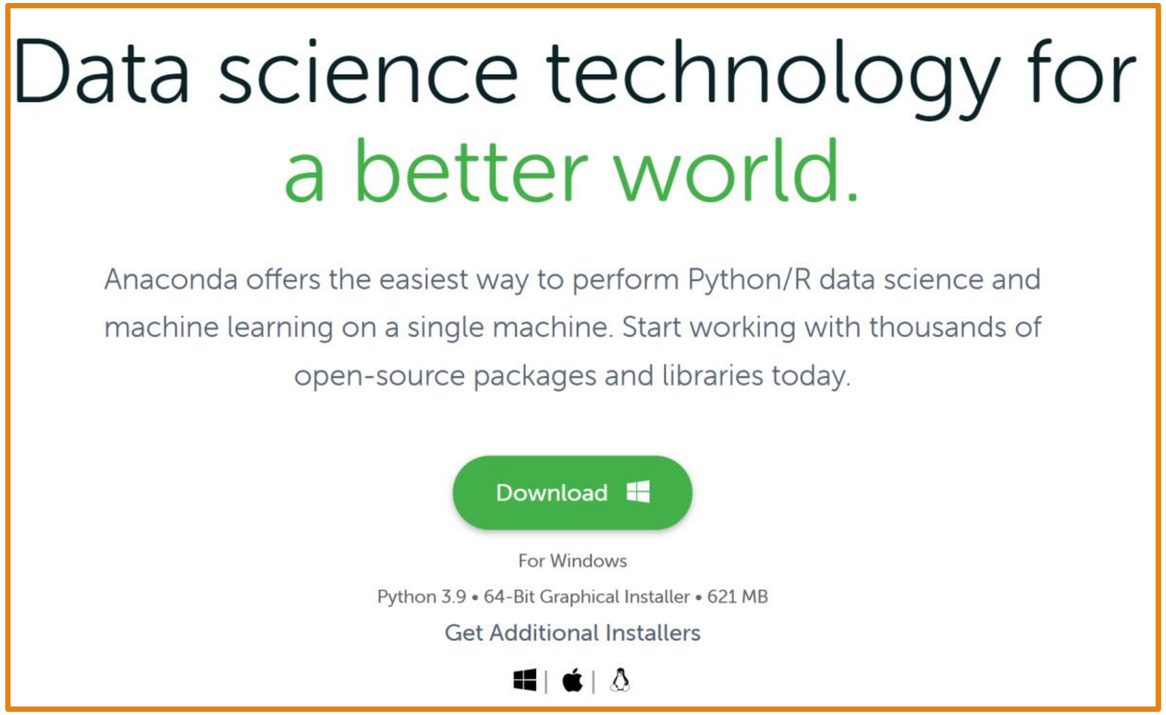
```
x = 5
y = "John"
print(x)
print(y)
```

```
5
John
```

Installing

Anaconda

https://www.anaconda.com/



Data science technology for a better world.

Anaconda offers the easiest way to perform Python/R data science and machine learning on a single machine. Start working with thousands of open-source packages and libraries today.

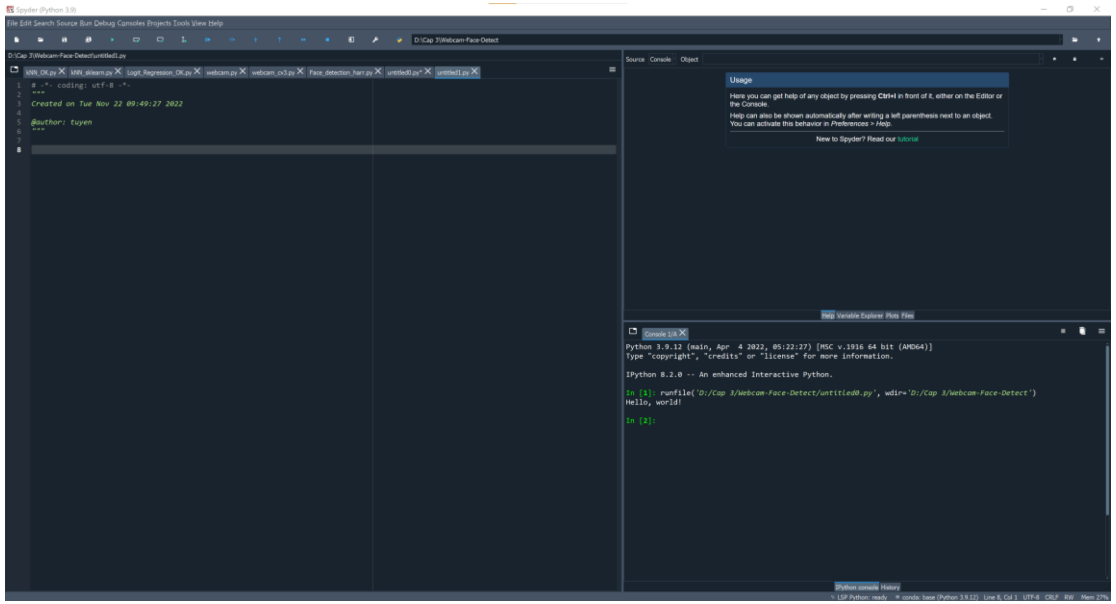
For Windows

Python 3.9 • 64-Bit Graphical Installer • 621 MB

How to Get Started With Python

Use Spyder.

Spyder is an open-source cross-platform integrated development environment for scientific programming in the Python language.



```
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 22 09:49:27 2022

@author: tuyen
"""
```

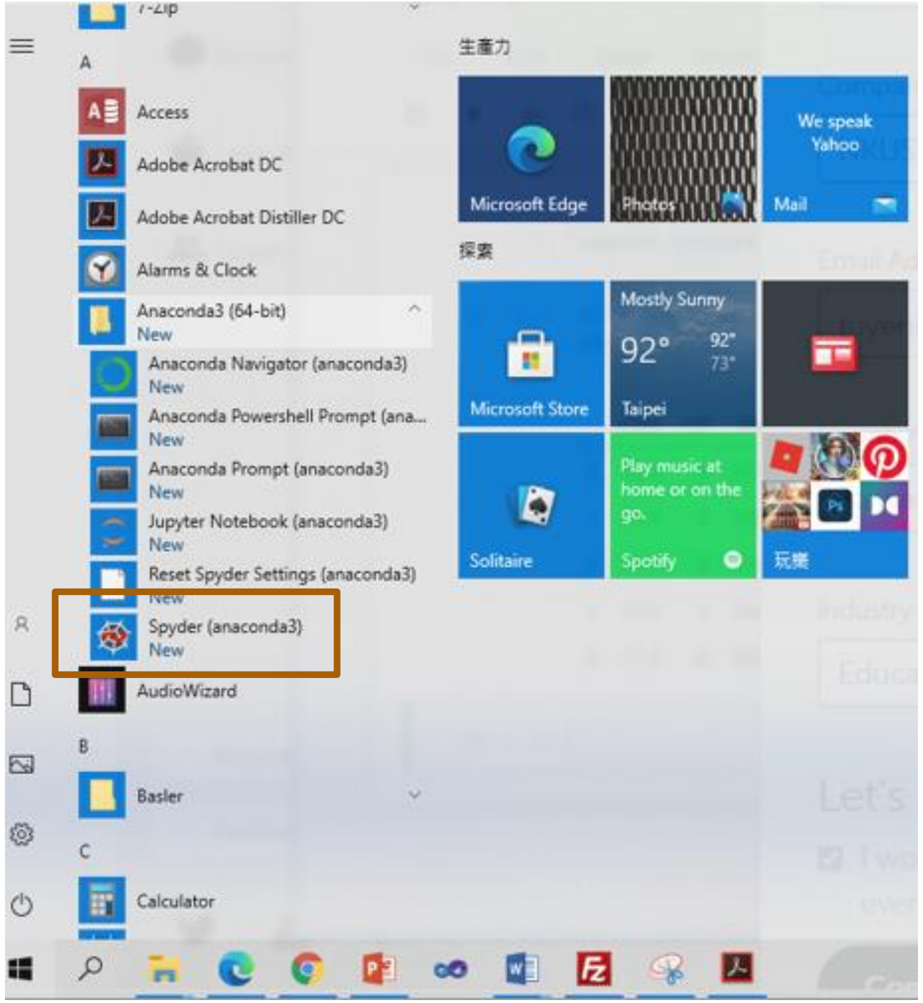
Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

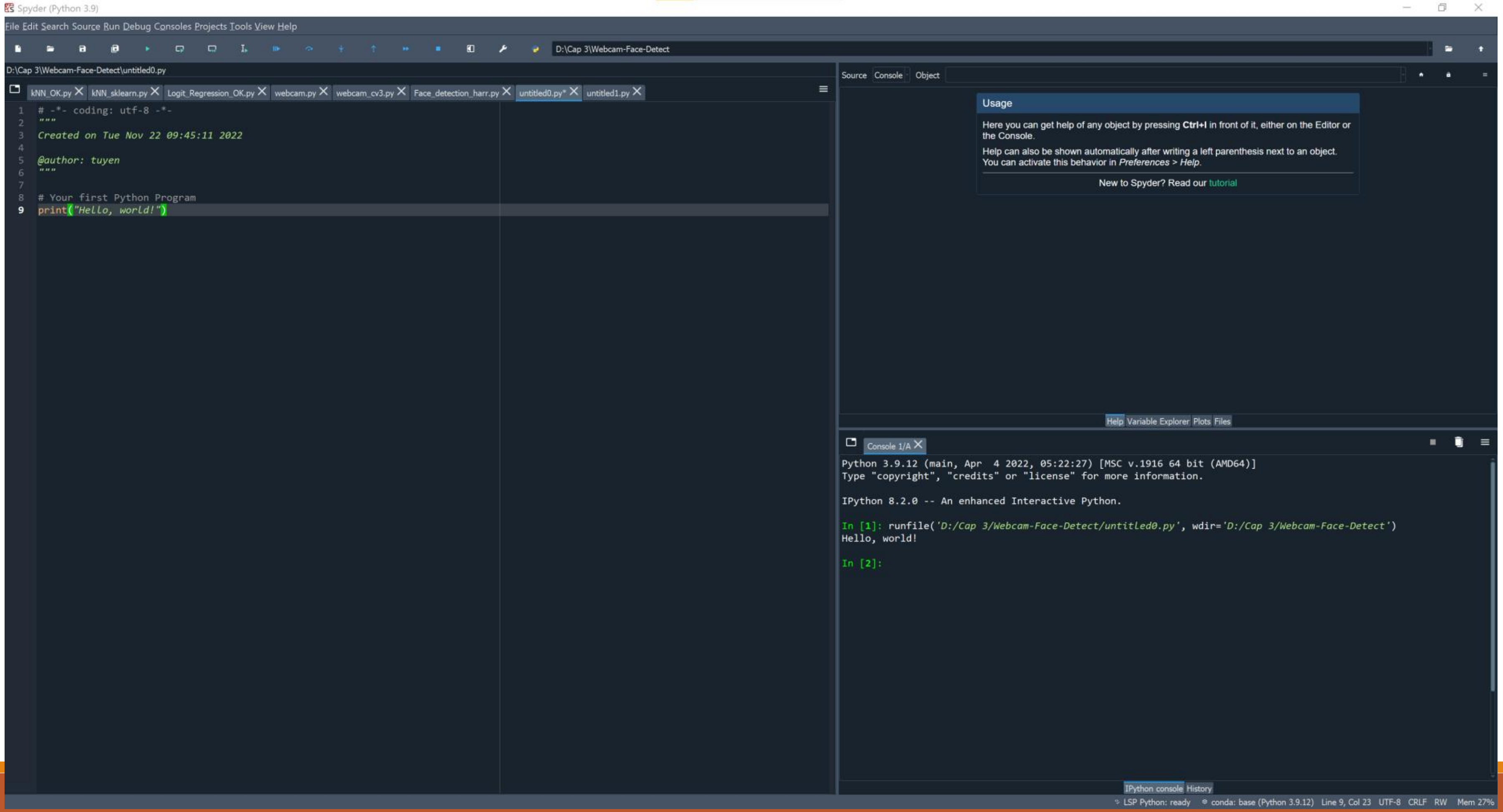
New to Spyder? Read our tutorial

```
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.2.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Cap 3/Webcam-Face-Detect/untitled0.py', wdir='D:/Cap 3/Webcam-Face-Detect')
Hello, world!

In [2]:
```



How to Get Started With Python

Use Spyder. Your first Python Program



```python
# -*- coding: utf-8 -*-
"""
Created on Tue Nov 22 09:45:11 2022

@author: tuyen
"""

# Your first Python Program
print("Hello, world!")
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

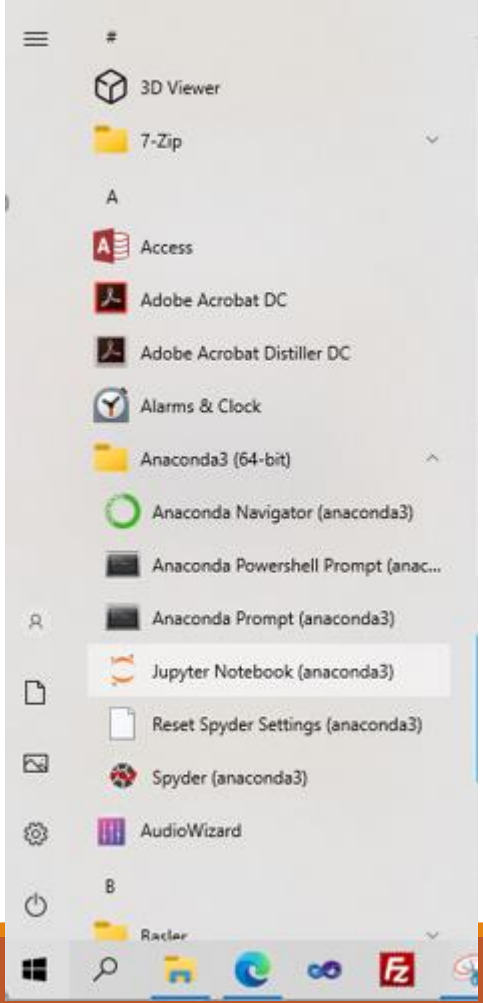
```
Python 3.9.12 (main, Apr  4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.2.0 -- An enhanced Interactive Python.

In [1]: runfile('D:/Cap 3/Webcam-Face-Detect/untitled0.py', wdir='D:/Cap 3/Webcam-Face-Detect')
Hello, world!

In [2]:
```

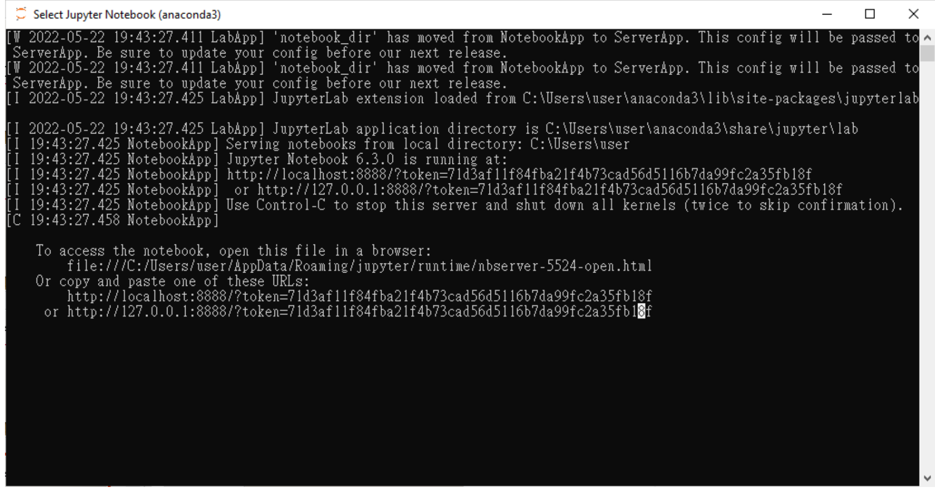
How to Get Started With Python

Jupyter Notebook

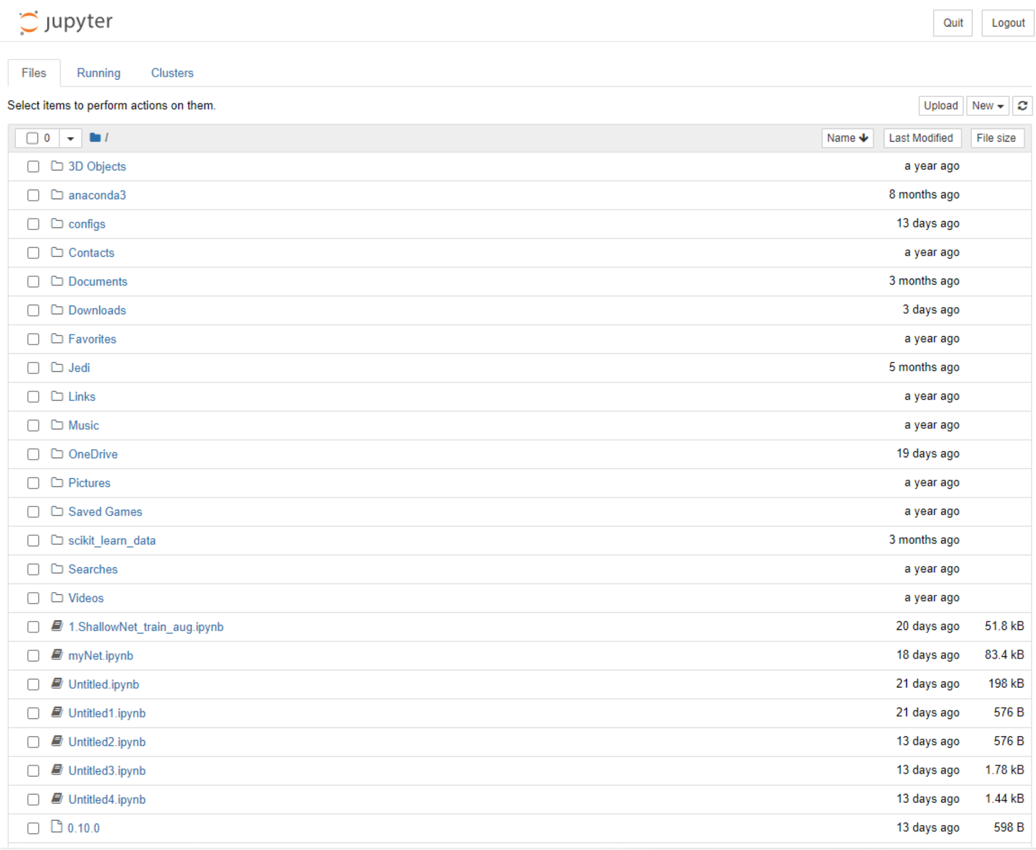
- The Jupyter Notebook is an open source web application that you can use to create and share documents that contain live code, equations, visualizations, and text.
- The name, Jupyter, comes from the core supported programming languages that it supports: Julia, Python, and R. Jupyter ships with the IPython kernel, which allows you to write your programs in Python, but there are currently over 100 other kernels that you can also use.

Using Jupyter Notebook on anaconda


#
3D Viewer
7-Zip
A
Access
Adobe Acrobat DC
Adobe Acrobat Distiller DC
Alarms & Clock
Anaconda3 (64-bit)
Anaconda Navigator (anaconda3)
Anaconda Powershell Prompt (anac...
Anaconda Prompt (anaconda3)
Jupyter Notebook (anaconda3)
Reset Spyder Settings (anaconda3)
Spyder (anaconda3)
AudioWizard
B
Basler


Select Jupyter Notebook (anaconda3)

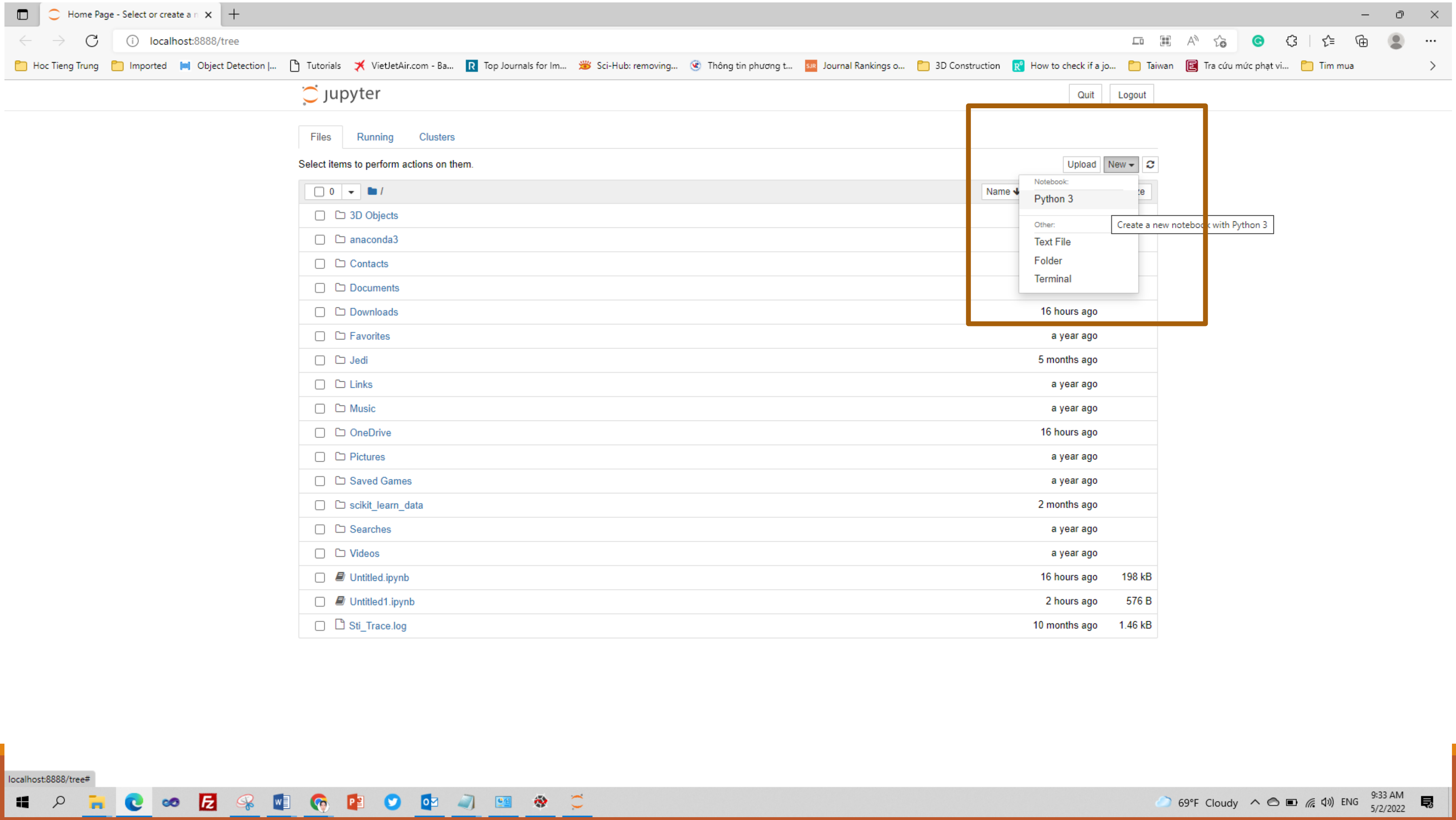
```
[W 2022-05-22 19:43:27.411 LabApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to
 ServerApp. Be sure to update your config before our next release.
[W 2022-05-22 19:43:27.411 LabApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to
 ServerApp. Be sure to update your config before our next release.
[I 2022-05-22 19:43:27.425 LabApp] JupyterLab extension loaded from C:\Users\user\anaconda3\lib\site-packages\jupyterlab

[I 2022-05-22 19:43:27.425 LabApp] JupyterLab application directory is C:\Users\user\anaconda3\share\jupyter\lab
[I 19:43:27.425 NotebookApp] Serving notebooks from local directory: C:\Users\user
[I 19:43:27.425 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 19:43:27.425 NotebookApp] http://localhost:8888/?token=71d3af11f84fba21f4b73cad56d5116b7da99fc2a35fb18f
[I 19:43:27.425 NotebookApp]  or http://127.0.0.1:8888/?token=71d3af11f84fba21f4b73cad56d5116b7da99fc2a35fb18f
[I 19:43:27.425 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 19:43:27.458 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/user/AppData/Roaming/jupyter/runtime/nbserver-5524-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=71d3af11f84fba21f4b73cad56d5116b7da99fc2a35fb18f
     or http://127.0.0.1:8888/?token=71d3af11f84fba21f4b73cad56d5116b7da99fc2a35fb18f
```



Select items to perform actions on them.

□ 0 /	Name	Last Modified	File size
□	3D Objects	a year ago	
□	anaconda3	8 months ago	
□	configs	13 days ago	
□	Contacts	a year ago	
□	Documents	3 months ago	
□	Downloads	3 days ago	
□	Favorites	a year ago	
□	Jedi	5 months ago	
□	Links	a year ago	
□	Music	a year ago	
□	OneDrive	19 days ago	
□	Pictures	a year ago	
□	Saved Games	a year ago	
□	scikit_learn_data	3 months ago	
□	Searches	a year ago	
□	Videos	a year ago	
□	1.ShallowNet_train_aug.ipynb	20 days ago	51.8 kB
□	myNet.ipynb	18 days ago	83.4 kB
□	Untitled.ipynb	21 days ago	198 kB
□	Untitled1.ipynb	21 days ago	576 B
□	Untitled2.ipynb	13 days ago	576 B
□	Untitled3.ipynb	13 days ago	1.78 kB
□	Untitled4.ipynb	13 days ago	1.44 kB
□	0.10.0	13 days ago	598 B

Using Jupyter Notebook on anaconda

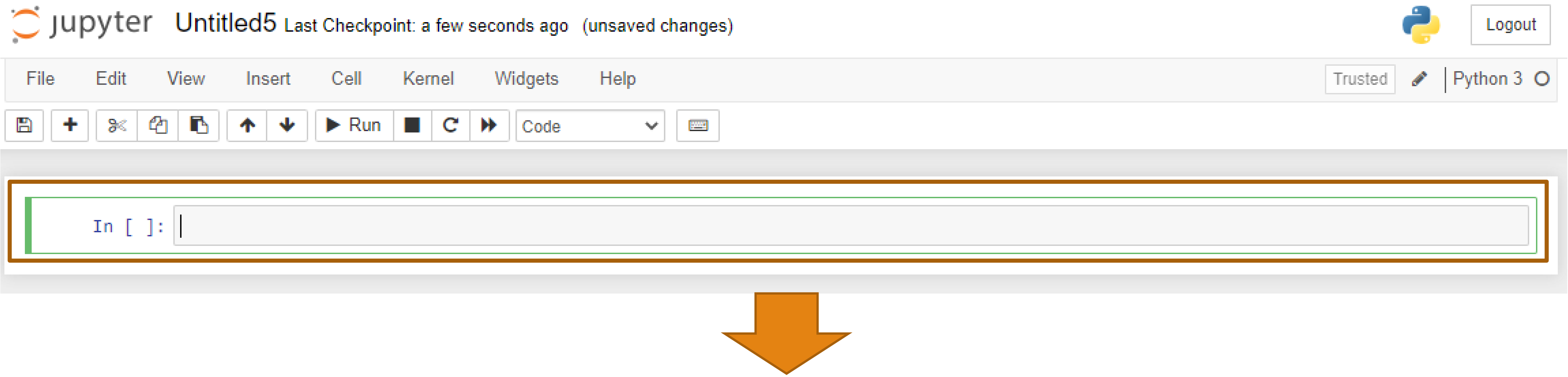


Select items to perform actions on them.

Create a new notebook with Python 3

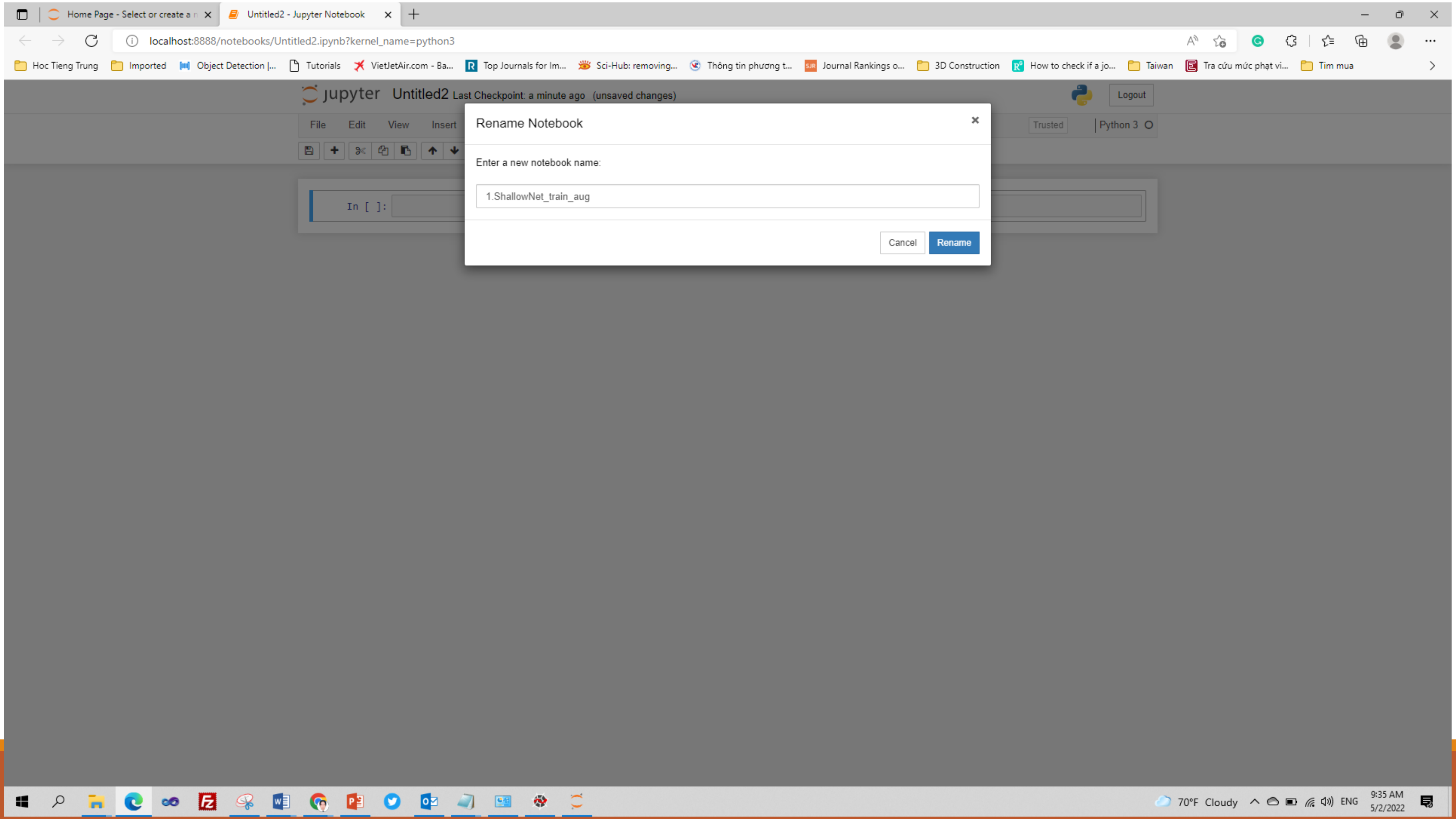
Name ↓		
3D Objects		
anaconda3		
Contacts		
Documents		
Downloads	16 hours ago	
Favorites	a year ago	
Jedi	5 months ago	
Links	a year ago	
Music	a year ago	
OneDrive	16 hours ago	
Pictures	a year ago	
Saved Games	a year ago	
scikit_learn_data	2 months ago	
Searches	a year ago	
Videos	a year ago	
Untitled.ipynb	16 hours ago	198 kB
Untitled1.ipynb	2 hours ago	576 B
Sti_Trace.log	10 months ago	1.46 kB

Using Jupyter Notebook on anaconda



Cells

Using Jupyter Notebook on anaconda



localhost:8888/notebooks/Untitled2.ipynb?kernel_name=python3

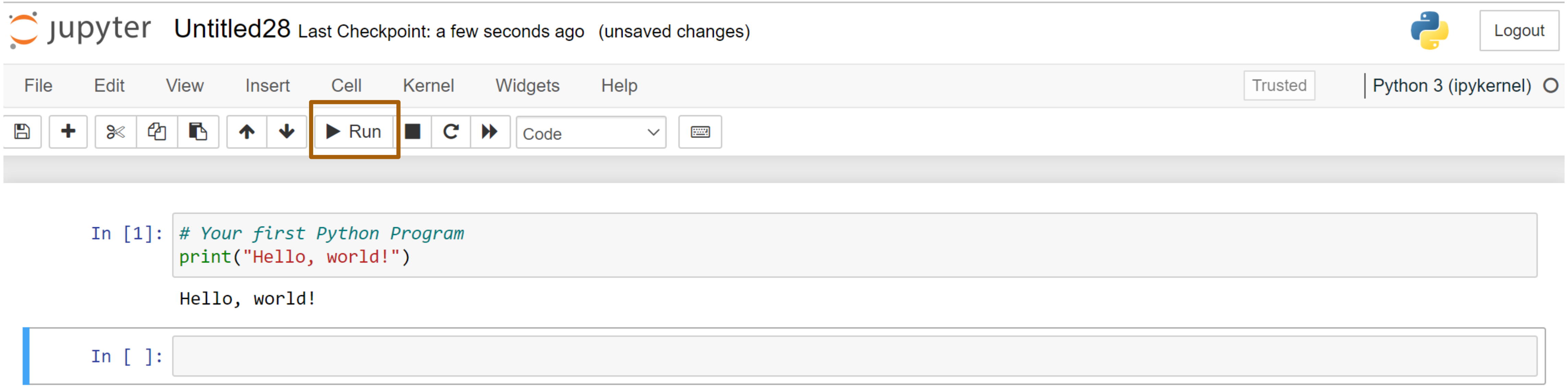
Untitled2 Last Checkpoint: a minute ago (unsaved changes)

Rename Notebook

Enter a new notebook name:

1.ShallowNet_train_aug

Your first Python Program



```
In [1]: # Your first Python Program
        print("Hello, world!")

        Hello, world!

In [ ]:
```

Keywords

- ✓ Keywords are predefined, reserved words used in Python programming that have special meanings to the compiler.
- ✓ We cannot use a keyword as a variable name, function name, or any other identifier (**classes**, **methods**, etc). They are used to define the syntax and structure of the Python language.
- ✓ All the keywords except True, False and None are in **lowercase** and they must be written as they are.

Keywords

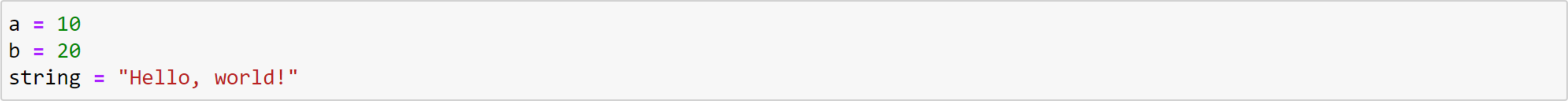
Keywords in Python programming language

False
None
True
and
as
assert
async
await
break
class
continue
def
del
elif
else
except
finally
for
from
global
if
import
in
is
lambda
nonlocal
not
or
pass
raise
return
try
while
with
yield

Identifiers

Identifiers are the name given to **variables**, **classes**, **methods**, etc.

Example:



```
a = 10
b = 20
string = "Hello, world!"
```

Here, a, b, string are variables (identifiers) that hold the value 10, 20, ‘Hello, world’.

Identifiers

Rules for Naming an Identifier

- ✓ Identifiers cannot be a keyword.
- ✓ Identifiers are case-sensitive.
- ✓ It can have a sequence of letters and digits. However, it must begin with a letter or _. The first letter of an identifier cannot be a digit.
- ✓ It's a convention to start an identifier with a letter rather _.
- ✓ Whitespaces are not allowed. Multiple words can be separated using an underscore, like this_is_a_long_variable.
- ✓ We cannot use special symbols like !, @, #, $, and so on.

Identifiers

Some Valid and Invalid Identifiers in Python

Valid Identifiers	Invalid Identifiers
score	@core
return_value	return
highest_score	highest score
na245me1	1name
convert_to_string	convert to_string

Comments

In computer programming, comments are hints that we use to make our code **easier to understand.**

single-line comment (use hash: #)

```
# create a variable
school_name = "明志科技大學" # school_name is a string

# print the value
print(school_name)
```

NOTE

- ✓ use single-quotes and double-quotes for string literals
- ✓ use triple-quotes for comment

multi-line comment

```
# This is a long comment
# and it extends
# to multiple lines
```

```
''' This is also a
perfect example of
multi-line comments lie
between open and
closed triple quotes '''
```

```
""" other multi-line
comments example lie
between open and
closed triple quotes"""
```

Basic concept

Outline

Variables

Data Types (Numeric Data type)

math Module

Operators

Conditional Statements

Variables

In programming, a variable is a **container** (storage area) to hold **data**.

Creating Variables: Python has no command for declaring a variable. A variable is created the moment you first assign a value to it.

variable_name = variable_value

```
# create a variable
# school_name is a variable storing the value 明志科技大學
school_name = "明志科技大學"
# print the value
print(school_name)
```

```
# Assigning multiple values to multiple variables
a, A, c = 10, 1.2, 'Hello'
print(a)  # prints 10
print(A)  # prints 1.2
print(c)  # prints Hello
```

Variables

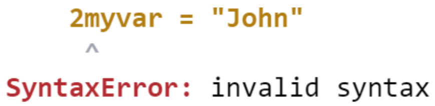
variable_name = variable_value

Rules for Variables' name

- ✓ **must start with a letter or the underscore character**
- ✓ **cannot start with a number, special characters (!, @, #, $, %, &, *)**
- ✓ **can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)**
- ✓ **case-sensitive (age, Age and AGE are three different variables)**

```
#Legal variable names:
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

```
#Illegal variable names:
2myvar = "John"
my-var = "John"
my var = "John“
*var = "John"
```



```
    2myvar = "John"
     ^
SyntaxError: invalid syntax
```

How to check the validity of Variables' name? `isidentifier()`

```
print("xyz".isidentifier())  # True
print("88x".isidentifier())  # False
print("_".isidentifier())    # True
print("while".isidentifier())# True
```

Variables

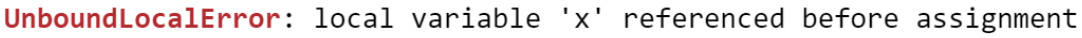
Global Variables

- **is a variable that is created <u>outside</u> of a function.**
- **can be used by everyone, both <u>inside</u> of functions and <u>outside</u>.**

```
x = "awesome" # x is global variable
def myfun():
    print("Python is ", x) # global variable use it inside the function
myfun()
```

```
x = "awesome" # x is global variable
def myfun():
    x = "fantastic"
    print("Python is ", x) # x is local variable
myfun()
print("Python is ", x) # x is global variable
```

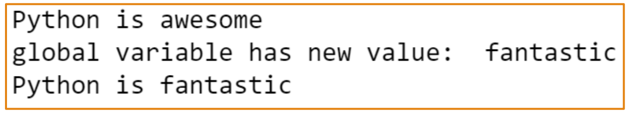
```
x = "awesome" # x is global variable
def myfun():
    print("Python is ", x) # x is local variable
    x = "fantastic"
myfun()
```



```
UnboundLocalError: local variable 'x' referenced before assignment
```

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "awesome"
def myfunc():
    global x
    x = "fantastic"
    print("global variable has new value: ", x)
print("Python is " + x)
myfunc()
print("Python is " + x)
```



```
Python is awesome
global variable has new value:  fantastic
Python is fantastic
```

Variables

Output Variables: print() function

Syntax:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Parameters:

- **objects - object to the printed. * indicates that there may be more than one object**
- **sep - objects are separated by sep. Default value: ‘ ‘ (space)**
- **end - end is printed at last**
- **file - must be an object with write(string) method. If omitted, sys.stdout will be used which prints objects on the screen.**
- **flush - If True, the stream is forcibly flushed. Default value: False**

```
print("1: Hello", "how are you?", sep="---")
print("2: Hello", "how are you?", sep="---", end = "\b")
print("3: Hello", "how are you?", sep="---", end = "\f")
print("4: Hello", "how are you?", sep="---", end = "\n")
print("5: Hello", "how are you?", sep="---", end = "\r")
print("6: Hello", "how are you?", sep="---", end = "\t")
print("7: Hello", "how are you?", sep="---", end = "\v")
print("8: Hello", "how are you?", sep="---", end = "\\")
print("9: Hello", "how are you?", sep="---", end = "\?")
```

Escape sequence	Character represented
\b	Backspace character
\f	Formfeed character
\n	Newline character
\r	Carriage return character
\t	Tab character
\v	Vertical Tab character
\\	Backslash character
\?	Question mark character

Data Types

- During the execution of the program, a lot of information needs to be calculated and stored
- The information is stored in the memory space
- The required format vary with different types of data
- Data type is designed for different types of data

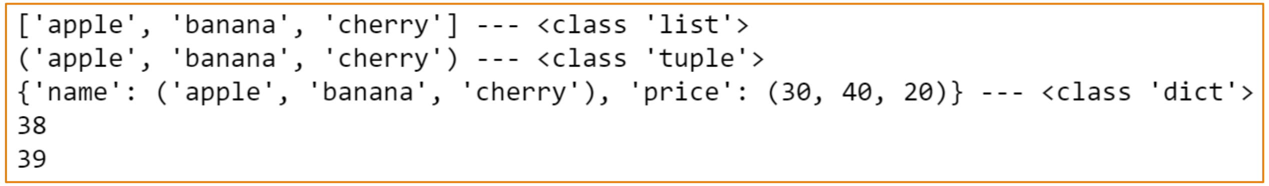
Data Types	**Classes**	**Description**	**Examples**
Numeric	int, float, complex	holds numeric values	a = int(2), b= float(2.1), c = complex(2,-3)
String	str	holds sequence of characters	x = “She is a student”
Sequence	list, tuple, range	holds collection of items	x = ("apple", "banana", "cherry")
Mapping	dict	holds data in key-value pair form	{'name': ('apple', 'banana',), 'price': ('50', '40')}
Boolean	bool	holds either True or False	x = True, y = False

https://www.programiz.com/python-programming/variables-datatypes

Data Types

We can use the type(name_of_variable) function to know which class a variable or a value belongs to.

```
str1, str2, str3 = "apple", "banana", "cherry"
price1, price2, price3 = 30, 40, 20
x = (str1, str2, str3 )
y = (price1, price2, price3)
a = list(x)
print(a,"---", type(a))
b = tuple(x)
print(b,"---", type(b))
c = dict(name = x, price = y)
print(c,"---", type(c))
for i in range (price1+8, price2):
   print(i)
```

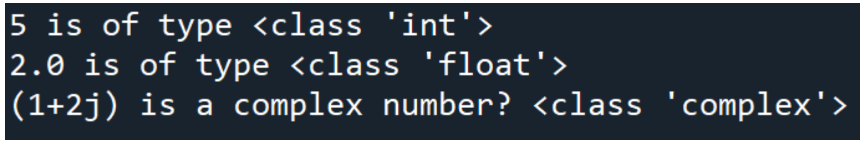


```
['apple', 'banana', 'cherry'] --- <class 'list'>
('apple', 'banana', 'cherry') --- <class 'tuple'>
{'name': ('apple', 'banana', 'cherry'), 'price': (30, 40, 20)} --- <class 'dict'>
38
39
```

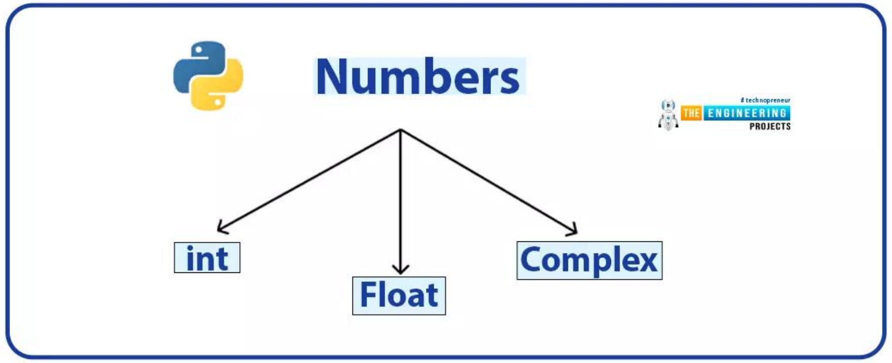
Numeric Data type

- ✓ Numeric data type is used to hold numeric values.
- ✓ Integers, floating-point numbers and complex numbers fall under Python numbers category. They are defined as int, float and complex classes in Python

```
num1 = 5
print(num1, 'is of type', type(num1))

num2 = 2.0
print(num2, 'is of type', type(num2))

num3 = 1+2j
print(num3, 'is a complex number?', type(num3))
```

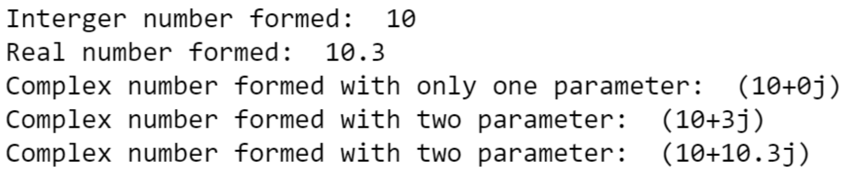


```
5 is of type <class 'int'>
2.0 is of type <class 'float'>
(1+2j) is a complex number? <class 'complex'>
```

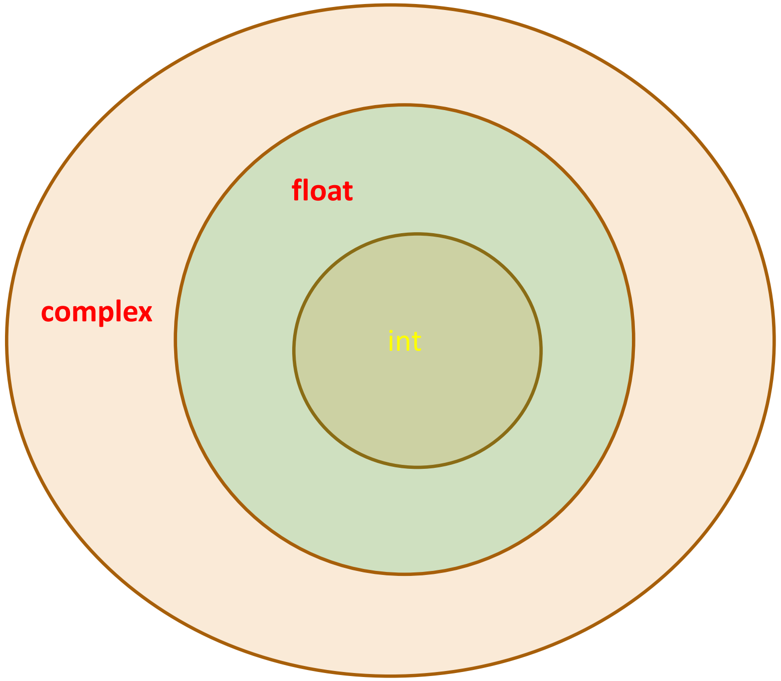


Numeric Data type

```
int_number = int(10)
real_number = float(10.3)
real = 10.0
imaginary = 3
complex_number1 = complex(real)
complex_number2 = complex(real, imaginary)
complex_number3 = complex(int_number, real_number)
print("Interger number formed: ", int_number)
print("Real number formed: ", real_number)
print("Complex number formed with only one parameter: ", complex_number1)
print("Complex number formed with two parameter: ", complex_number2)
print("Complex number formed with two parameter: ", complex_number3)
```



```
Interger number formed:  10
Real number formed:  10.3
Complex number formed with only one parameter:  (10+0j)
Complex number formed with two parameter:  (10+3j)
Complex number formed with two parameter:  (10+10.3j)
```

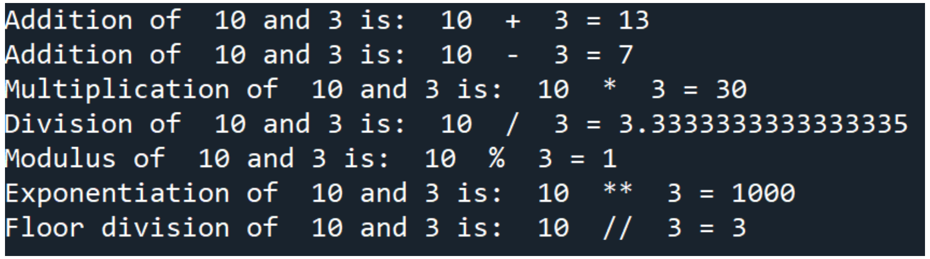


Numeric Data type

Arithmetic Operators

Operator	Name	Example
+ (plus)	Addition	x + y
- (minus)	Subtraction	x - y
* (asterisk)	Multiplication	x * y
/ (forward slash)	Division	x / y
% (percent)	Modulus	x % y
** (double asterisks)	Exponentiation	x ** y
// (double forward slashes)	Floor division	x // y

```
num1 = 10
num2 = 3
# Addition
print("Addition of ", num1, "and", num2, "is: ", num1," + ", num2, "=", num1 + num2)
# Subtraction
print("Addition of ", num1, "and", num2, "is: ", num1," - ", num2, "=", num1 - num2)
# Multiplication
print("Multiplication of ", num1, "and", num2, "is: ", num1," * ", num2, "=", num1 * num2)
# Division
print("Division of ", num1, "and", num2, "is: ", num1," / ", num2, "=", num1 / num2)
# Modulus
print("Modulus of ", num1, "and", num2, "is: ", num1," % ", num2, "=", num1 % num2)
# Exponentiation
print("Exponentiation of ", num1, "and", num2, "is: ", num1," ** ", num2, "=", num1 ** num2)
# Floor division
print("Floor division of ", num1, "and", num2, "is: ", num1," // ", num2, "=", num1 // num2)
```



```
Addition of  10 and 3 is:  10  +  3 = 13
Addition of  10 and 3 is:  10  -  3 = 7
Multiplication of  10 and 3 is:  10  *  3 = 30
Division of  10 and 3 is:  10  /  3 = 3.3333333333333335
Modulus of  10 and 3 is:  10  %  3 = 1
Exponentiation of  10 and 3 is:  10  **  3 = 1000
Floor division of  10 and 3 is:  10  //  3 = 3
```

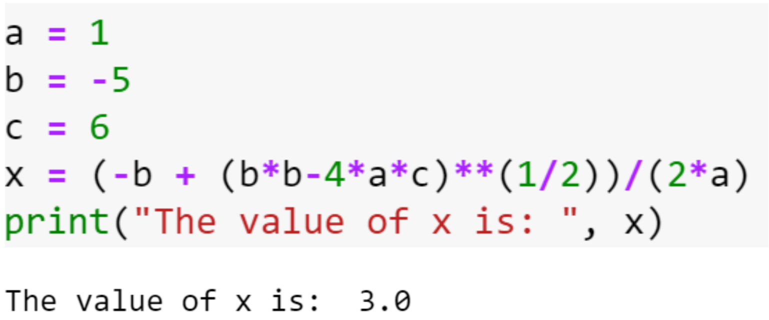
Numeric Data type

Exercise:

Using Python code write a program to calculate the following algebraic expression

$$x = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

when a = 1, b= -5, c = 6



```
a = 1
b = -5
c = 6
x = (-b + (b*b-4*a*c)**(1/2))/(2*a)
print("The value of x is: ", x)
```

```
The value of x is:  3.0
```

How to import a library in Python?

✓ Modules are Python .py files that consist of Python code. Any Python file can be referenced as a module.

✓ Modules can define functions, classes, and variables that you can reference in other Python .py files or via the Python command line interpreter.

✓ To make use of the functions in a module, you’ll need to import the module with an import statement.

✓ An import statement is made up of the import keyword along with the name of the module.

```
import name_of_module
# use methods in module
name_of_module.name_of_method()
```

```
import name_of_module as nm
# use methods in module
nm.name_of_method()
```

```
# use methods in module
from name_of_module import name_of_method
name_of_method()
```

math Module

- ✓ Python has a built-in module that you can use for mathematical tasks: **math**.

- ✓ The math module has a set of methods (functions) and constants.

import math module in Python

```
PI = 3.14159265359
import math
# use cos method in math
a= math.cos(PI)
print(a)
```

```
PI = 3.14159265359
from math import cos
# use cos method in math
a = cos(PI)
print(a)
```

Math Methods

Method	Description
math.acos()	Returns the arc cosine of a number
math.acosh()	Returns the inverse hyperbolic cosine of a number
math.asin()	Returns the arc sine of a number
math.asinh()	Returns the inverse hyperbolic sine of a number
math.atan()	Returns the arc tangent of a number in radians
math.atan2()	Returns the arc tangent of y/x in radians
math.atanh()	Returns the inverse hyperbolic tangent of a number
math.ceil()	Rounds a number up to the nearest integer
math.comb()	Returns the number of ways to choose k items from n items without repetition and order
math.copysign()	Returns a float consisting of the value of the first parameter and the sign of the second parameter
math.cos()	Returns the cosine of a number
math.cosh()	Returns the hyperbolic cosine of a number
math.degrees()	Converts an angle from radians to degrees
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf()	Returns the error function of a number
math.erfc()	Returns the complementary error function of a number

Math Methods

Method	Description
math.exp()	Returns E raised to the power of x
math.expm1()	Returns E^x - 1
math.fabs()	Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor()	Rounds a number down to the nearest integer
math.fmod()	Returns the remainder of x/y
math.frexp()	Returns the mantissa and the exponent, of a specified number
math.fsum()	Returns the sum of all items in any iterable (tuples, arrays, lists, etc.)
math.gamma()	Returns the gamma function at x
math.gcd()	Returns the greatest common divisor of two integers
math.hypot()	Returns the Euclidean norm
math.isclose()	Checks whether two values are close to each other, or not
math.isfinite()	Checks whether a number is finite or not
math.isinf()	Checks whether a number is infinite or not
math.isnan()	Checks whether a value is NaN (not a number) or not
math.isqrt()	Rounds a square root number downwards to the nearest integer

Math Methods

Method	Description
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10()	Returns the base-10 logarithm of x
math.log1p()	Returns the natural logarithm of 1+x
math.log2()	Returns the base-2 logarithm of x
math.perm()	Returns the number of ways to choose k items from n items with order and without repetition
math.pow()	Returns the value of x to the power of y
math.prod()	Returns the product of all the elements in an iterable
math.radians()	Converts a degree value into radians
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
math.sin()	Returns the sine of a number
math.sinh()	Returns the hyperbolic sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh()	Returns the hyperbolic tangent of a number
math.trunc()	Returns the truncated integer parts of a number

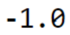
Math Constants

Constant	Description
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

math Module: examples

Syntax: *math.cos(x)*
Parameter: ***x,*** *value to be passed to cos(), x* should be in radians.
Returns: *Returns the cosine of value passed as argument*.

```
import math
PI = 3.14159265359
a = math.cos(PI)
print(“”a)
```



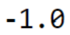
```
-1.0
```

Syntax: *round(number, number of digits)*
Parameter:

- ***number :*** *number to be rounded*
- ***number of digits (Optional) :*** *number of digits up to which the given number is to be rounded*

Returns:
if only an integer is given

```
from math import cos, pi
# use cos method and pi constant in math
a = cos(pi)
print(a)
```



```
-1.0
```

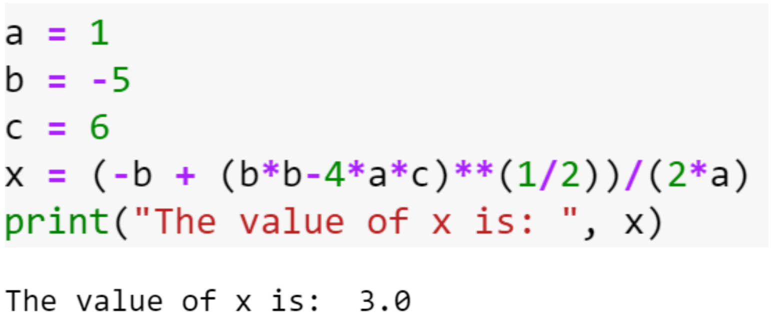
math Module: examples

Exercise:

Using Python code write a program to calculate the following algebraic expression

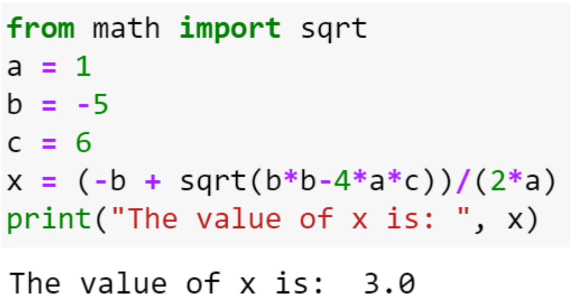
$$x = \frac{-b+\sqrt{b^2-4ac}}{2a}$$

when a = 1, b= -5, c = 6



```
a = 1
b = -5
c = 6
x = (-b + (b*b-4*a*c)**(1/2))/(2*a)
print("The value of x is: ", x)
```

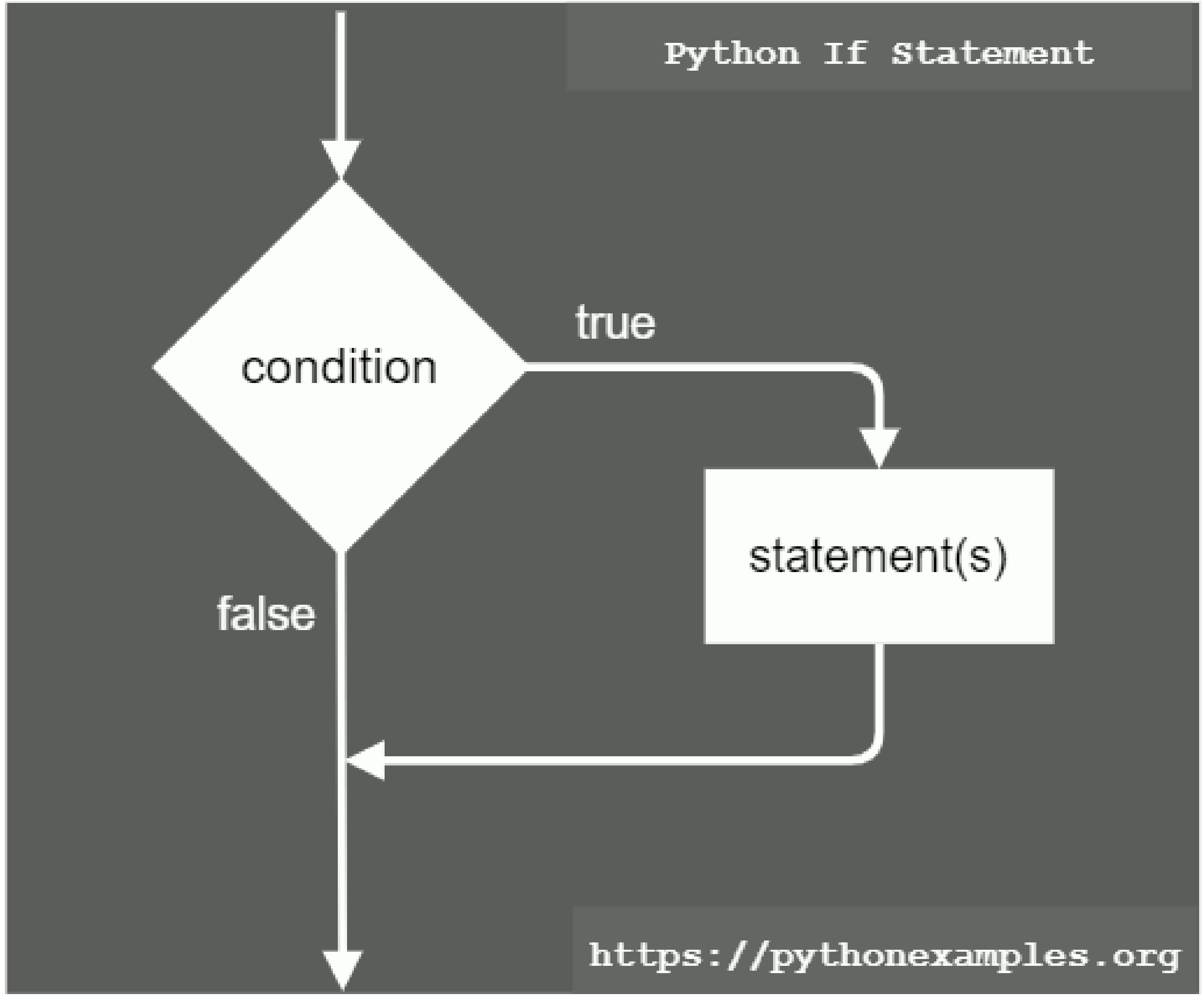
```
The value of x is:  3.0
```



```
from math import sqrt
a = 1
b = -5
c = 6
x = (-b + sqrt(b*b-4*a*c))/(2*a)
print("The value of x is: ", x)
```

```
The value of x is:  3.0
```

Conditional Statements

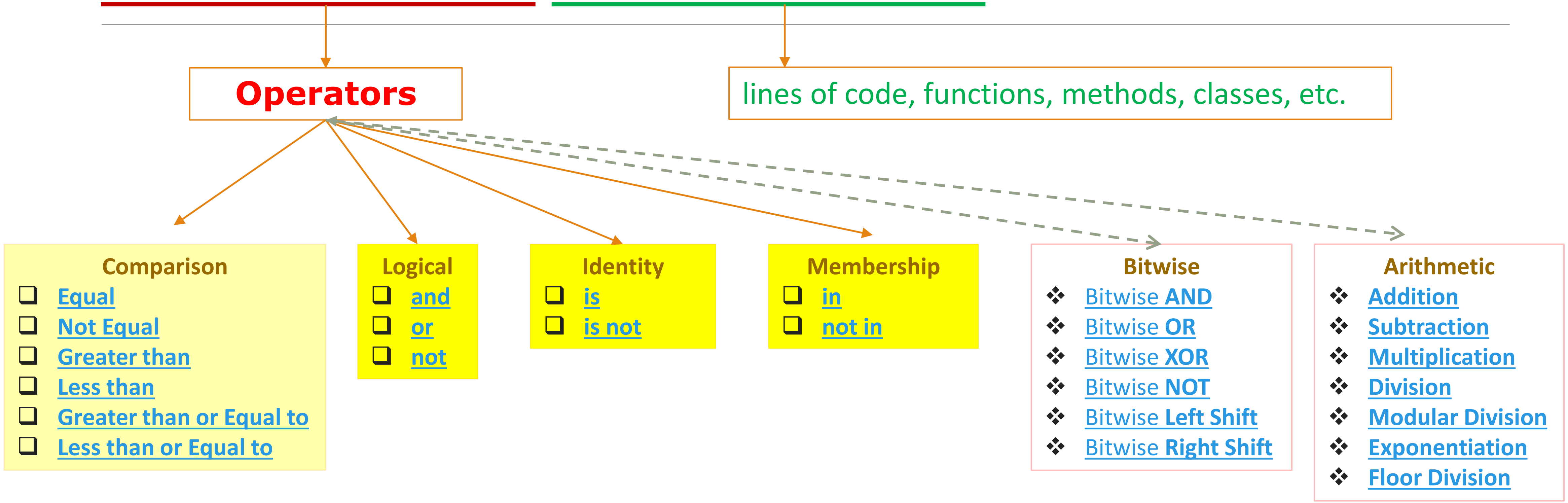


- **If... statement can allow to execute a block of statements based on given condition.**

- **If...Else statement can execute either if-block or else-block statements, based on the result of given condition.**

- **Elif... ladder of "if else-if else" statements with a condition at if block, and each elif block.**

Conditional Statements



Operators

lines of code, functions, methods, classes, etc.

Comparison

- ❑ Equal
- ❑ Not Equal
- ❑ Greater than
- ❑ Less than
- ❑ Greater than or Equal to
- ❑ Less than or Equal to

Logical

- ❑ and
- ❑ or
- ❑ not

Identity

- ❑ is
- ❑ is not

Membership

- ❑ in
- ❑ not in

Bitwise

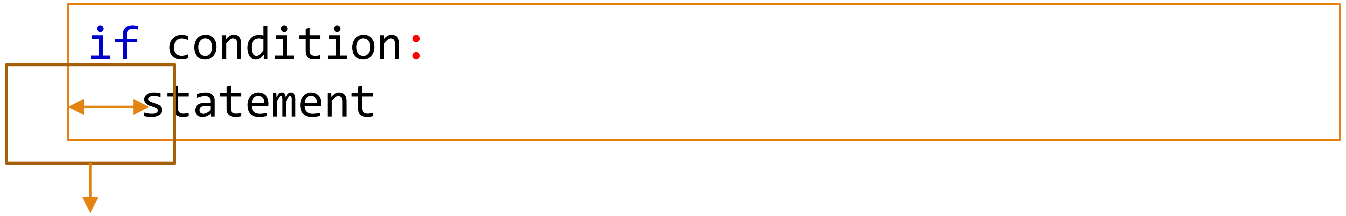
- ❖ Bitwise AND
- ❖ Bitwise OR
- ❖ Bitwise XOR
- ❖ Bitwise NOT
- ❖ Bitwise Left Shift
- ❖ Bitwise Right Shift

Arithmetic

- ❖ Addition
- ❖ Subtraction
- ❖ Multiplication
- ❖ Division
- ❖ Modular Division
- ❖ Exponentiation
- ❖ Floor Division

Conditional Statements: if...

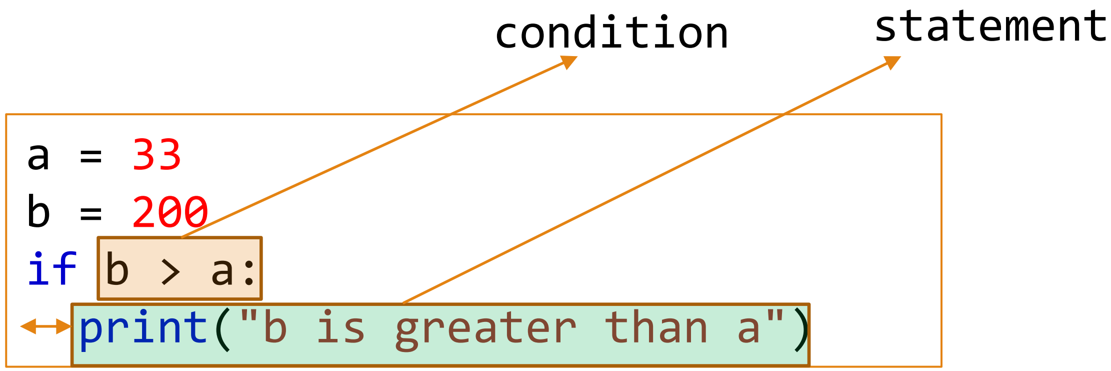
An "if statement" is written by using the `if` keyword.



```
if condition:
    statement
```

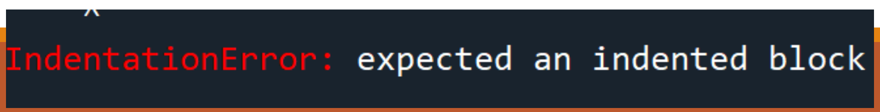
Indentation (1 tab)

Python relies on indentation (whitespace at the beginning of a line) to define scope in the code.



```
a = 33
b = 200
if b > a:
    print("b is greater than a")
```

```
a = 33
b = 200
if b > a:
print("b is greater than a")
```



```
IndentationError: expected an indented block
```

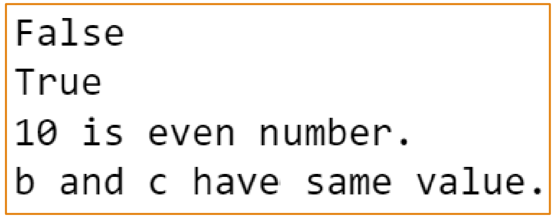
Comparison Operators

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

return value

True or False

```
a = 10
b = 12
c = 12
print(a == b)
print(b == c)
if a%2 == 0 :
   print(a, "is even number.")
if b == c :
   print("b and c have same value.")
```

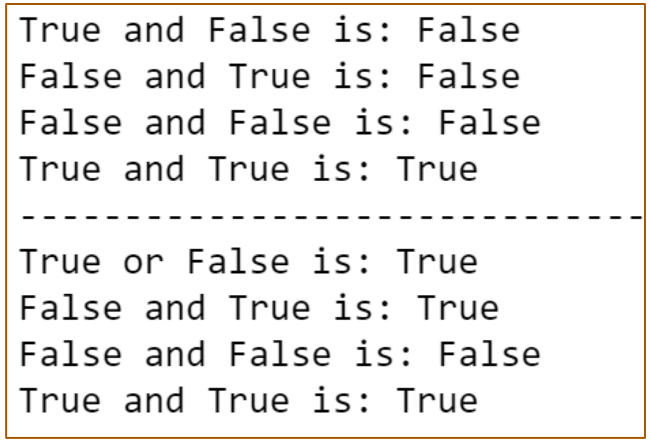


```
False
True
10 is even number.
b and c have same value.
```

Logical Operators

```
a = True
b = False

print(a,'and',b,'is:',a and b)
print(b,'and',a,'is:',b and a)
print(not a,'and',b,'is:',not a and b)
print(a,'and',not b,'is:',a and not b)
print('-----------------------------')
print(a,'or',b,'is:',a or b)
print(b,'and',a,'is:',b or a)
print(not a,'and',b,'is:',not a or b)
print(a,'and',not b,'is:',a or not b)
```

A	B	and	or	not A	not B
True	True	True	True	False	False
False	True	False	True	True	False
True	False	False	True	False	True
False	False	False	False	True	True



```
True and False is: False
False and True is: False
False and False is: False
True and True is: True
-----------------------------
True or False is: True
False and True is: True
False and False is: False
True and True is: True
```

Identity Operators: is and is not

is keyword is used to check if the memory reference of two Python objects are same or not. **is** operator takes two operands and returns **True** if both the objects have same memory reference and **False** if not.

```
a = [5, 8]
b = [5, 8]
c = a
if a is b:
        print('a is b')
if a is c:
        print('a is c')
```

```
a is c
```

is not operation is simply the opposite of **is**. **is not** checks if the two operands refer to the same memory reference. If they do not have the same memory reference, **is not** returns **True**, else it returns **False**.

```
a = [5, 8]
b = [5, 8]
c = a
if a is not b:
        print('a is not b')
if a is not c:
        print('a is not c')
```

```
a is not b
```

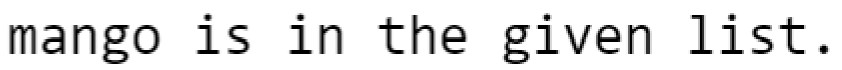
Membership Operators: in and not in

in is used to check if specific value is present in given collection.

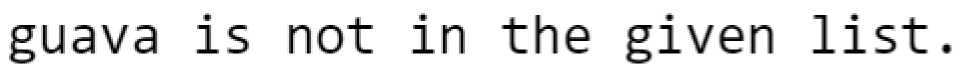
not in is used to check if specific value is not present in given collection.

```
fruits = ['apple', 'banana', 'cherry', 'mango']
x = 'mango'
if x in fruits:
   print('x is in the given list.')
```

```
fruits = ['apple', 'banana', 'cherry', 'mango']
x = 'guava'
if x not in fruits:
   print(f'{x} is not in the given list.')
```



```
mango is in the given list.
```



```
guava is not in the given list.
```

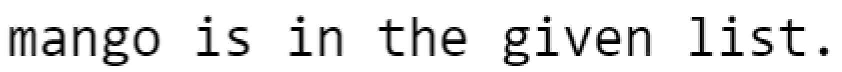
Membership Operators: in and not in

in is used to check if specific value is present in given collection.

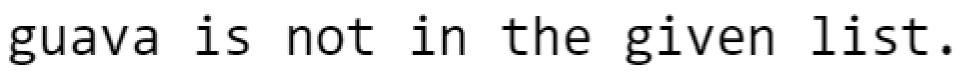
not in is used to check if specific value is not present in given collection.

```
fruits = ['apple', 'banana', 'cherry', 'mango']
x = 'mango'
if x in fruits:
   print('x is in the given list.')
```

```
fruits = ['apple', 'banana', 'cherry', 'mango']
x = 'guava'
if x not in fruits:
   print(f'{x} is not in the given list.')
```

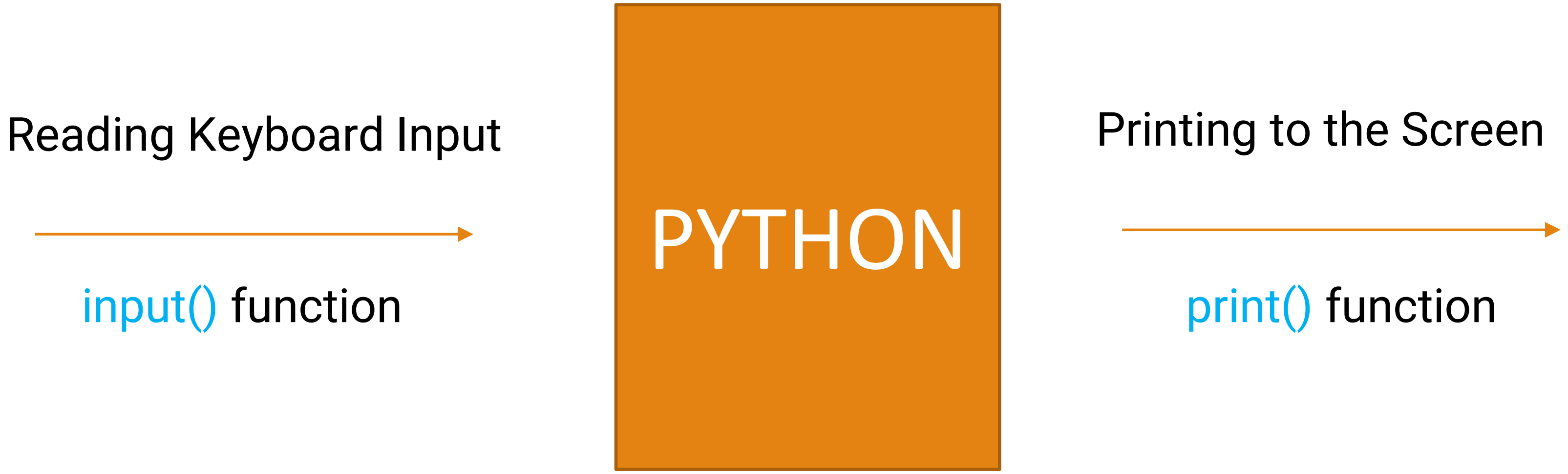


```
mango is in the given list.
```



```
guava is not in the given list.
```

Input/Output



Reading Keyboard Input

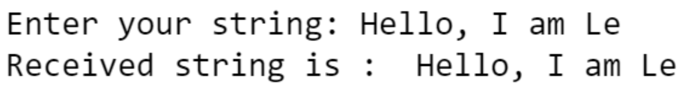
input() function

PYTHON

Printing to the Screen

print() function

```
str = input("Enter your string: ")
print ("Received string is: ", str)
```

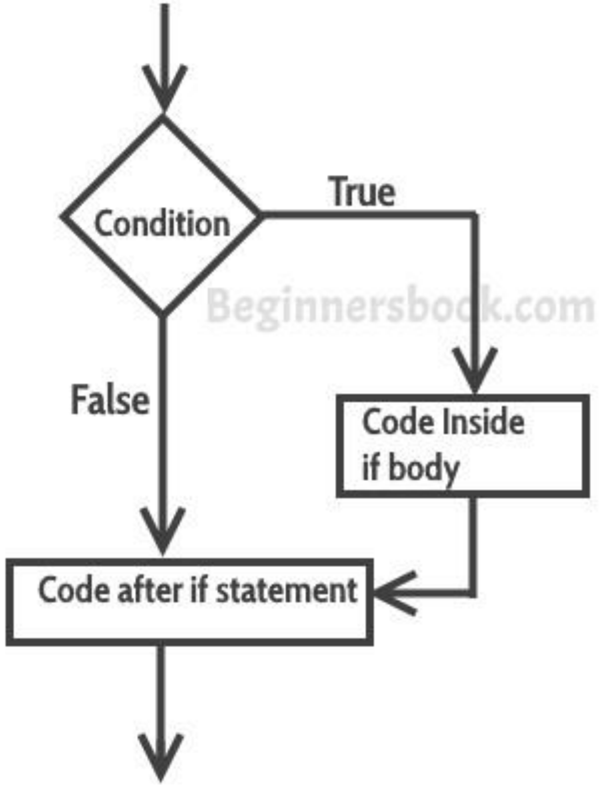


```
Enter your string: Hello, I am Le
Received string is :  Hello, I am Le
```

Conditional Statements: if…else

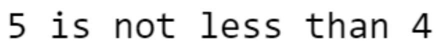
```
if condition:
  statement(s)1
else:
  statement(s)2
```

If condition == True, **statement(s)1** are executed and if condition == False, **statement(s)2** are executed

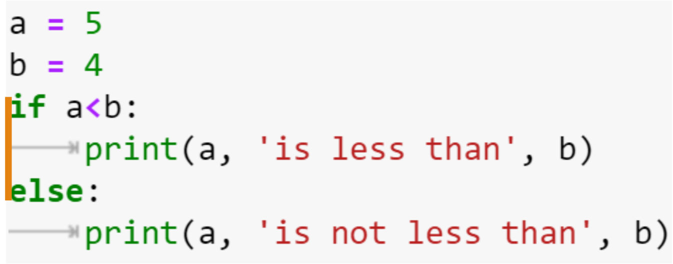


Conditional Statements: if…else

```
a = 5
b = 4
if a<b:
	print(a, 'is less than', b)
else:
	print(a, 'is not less than', b)
```

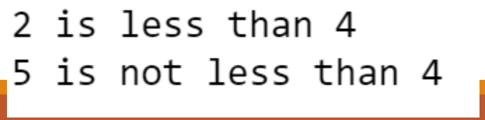


```
5 is not less than 4
```

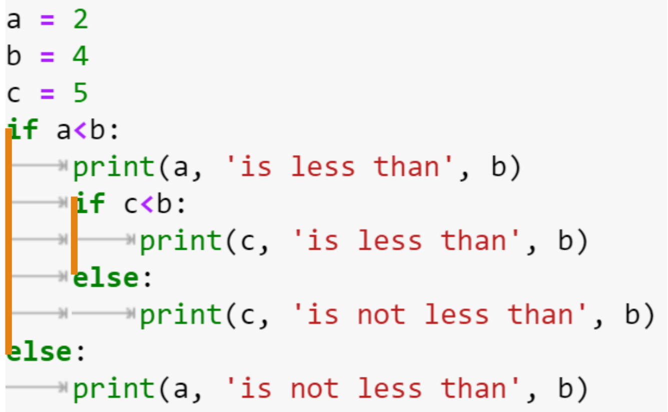


```
a = 5
b = 4
if a<b:
    print(a, 'is less than', b)
else:
    print(a, 'is not less than', b)
```

```
a = 5
b = 4
if a<b:
	print(a, 'is less than', b)
else:
	print(a, 'is not less than', b)
```



```
2 is less than 4
5 is not less than 4
```



```
a = 2
b = 4
c = 5
if a<b:
    print(a, 'is less than', b)
    if c<b:
        print(c, 'is less than', b)
    else:
        print(c, 'is not less than', b)
else:
    print(a, 'is not less than', b)
```

Conditional Statements: if…else

Exercise:

Using Python code write a program to calculate the following algebraic expression

$$x = \frac{-b+\sqrt{b^2-4ac}}{2a},$$

with *a, b, c* input from keyboard. Print to the screen whether the value of x is real (positive, zero, or negative) or complex.

```
from math import sqrt
a = input("Enter the first coefficient a = ")
a = float(a)
b = input("Enter the first coefficient b = ")
b = float(b)
c = input("Enter the first coefficient c = ")
c = float(c)
delta = b*b-4*a*c
if delta >= 0:
    x = (-b+sqrt(delta))/(2*a)
    if x>0:
        print("The value of real x is: ", x, "> 0")
    else:
        if x == 0:
            print("The value of real x is: ", x, "= 0")
        else:
            print("The value of real x is: ", x, "< 0")
else:
    real = -b/(2*a)
    image = sqrt(abs(delta)/(2*a))
    print("The value of x is a complex number: x = ", real, "+", image, "j")
```

Conditional Statements: elif

```
if boolean_expression_1:
        statement(s)
elif boolean_expression_2:
        statement(s)
elif boolean_expression_3:
        statement(s)
else:
        statement(s)
```

Python elif (short for else if) is used to execute a continuous chain of conditional logic ladder.

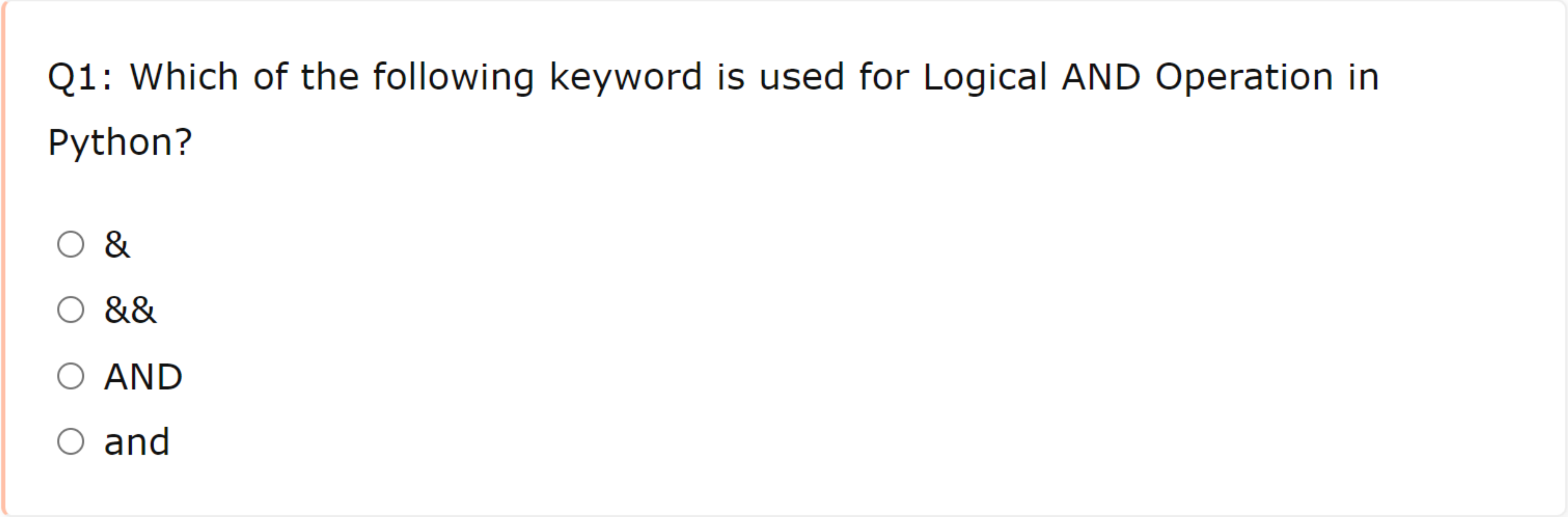
In elif, there are multiple conditions and the corresponding statement(s) as a ladder. Only one of the blocks gets executed when the corresponding boolean expression evaluates to true.

Conditional Statements: elif

```
a = input("Enter the first coefficient a = ")
a = float(a)
b = input("Enter the first coefficient b = ")
b = float(b)
if a<b:
        print(a, 'is less than', b)
elif a>b:
        print(a, 'is greater than', b)
else:
        print(a, 'equals', b)
```

```
a = input("Enter the first coefficient a = ")
a = float(a)
if a<0:
        print(a, 'is negative')
elif a==0:
        print('its a 0')
elif a>0 and a<10:
        print(a, 'is in (0,5)')
else:
        print(a, 'equals or greater than 5')
```

Exercises



Q1: Which of the following keyword is used for Logical AND Operation in Python?

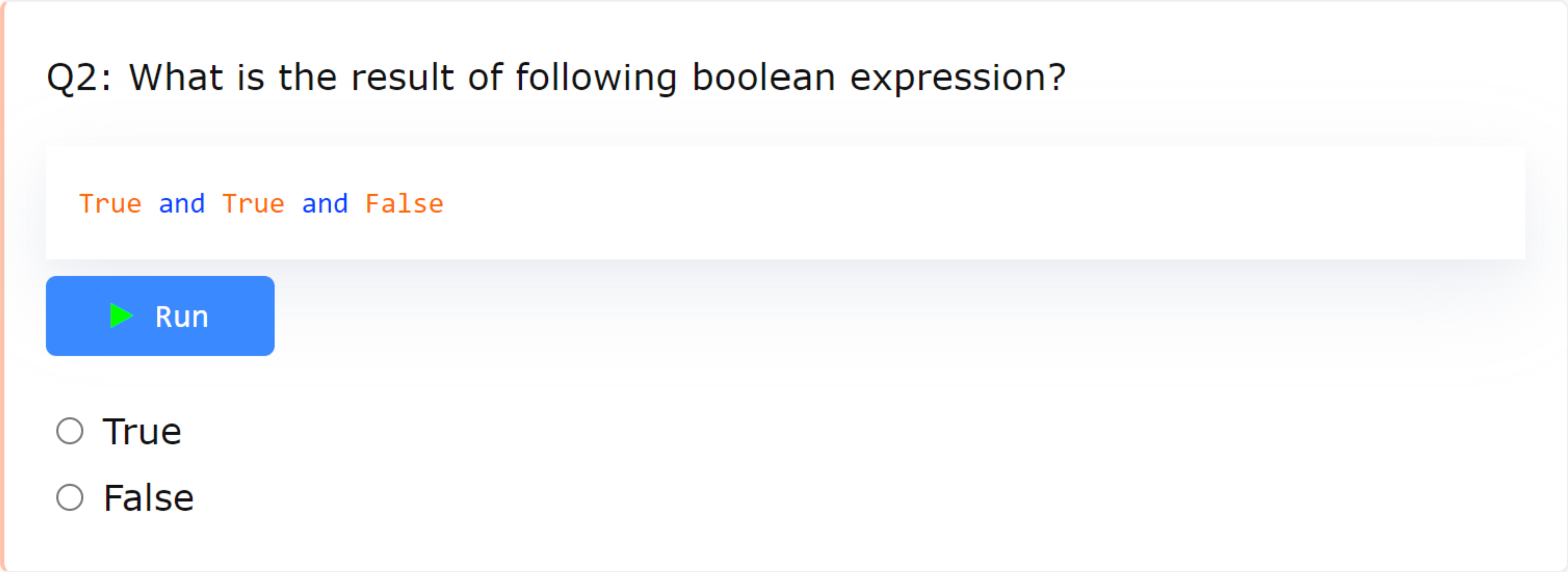
○ &

○ &&

○ AND

○ and

Exercises



Q2: What is the result of following boolean expression?

```
True and True and False
```

○ True

○ False

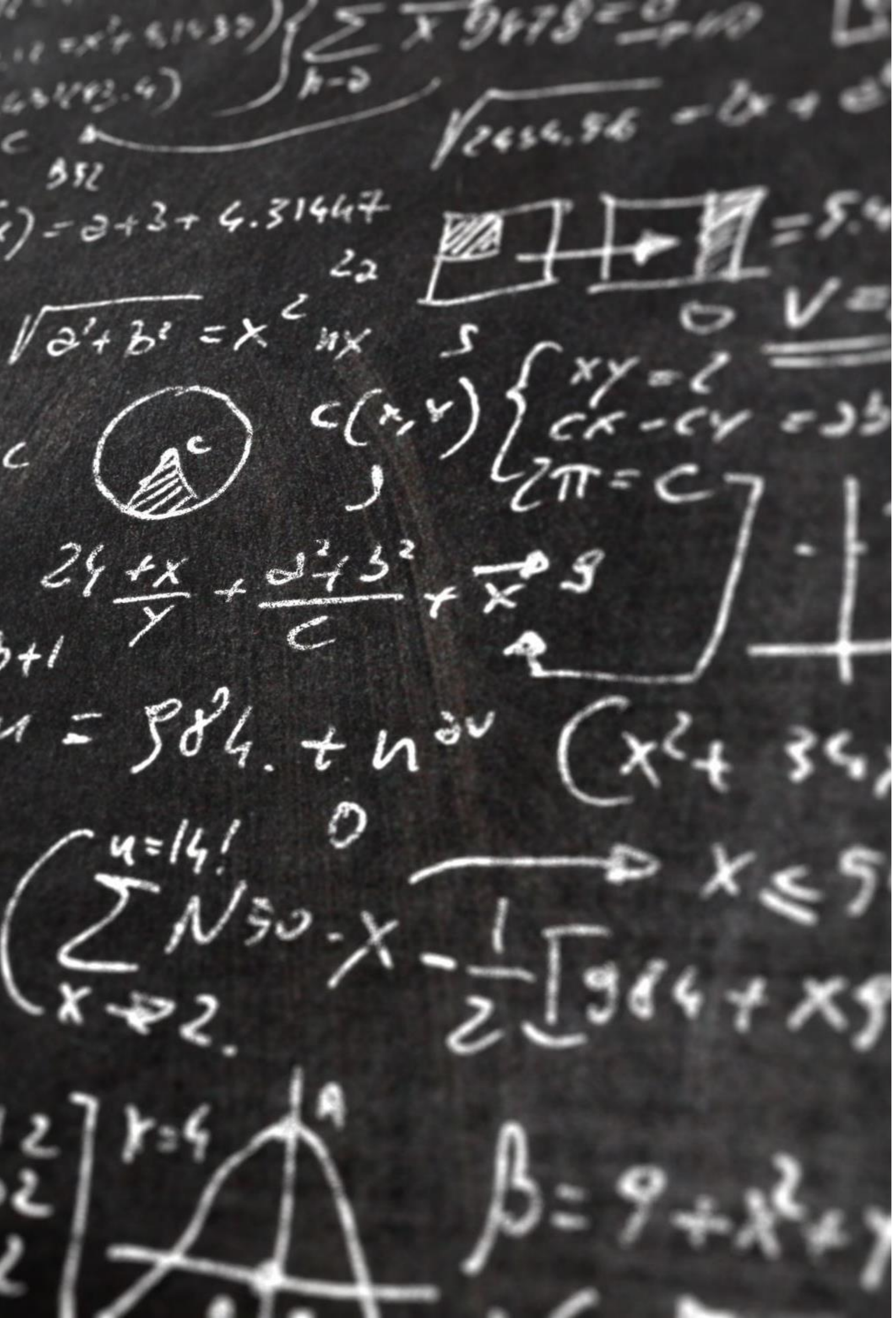
Exercises

Q3: Write a program to find solutions of the quadratic equation:

$$ax^2 + bx + c = 0$$

with the coefficients *a, b, c* are entered from the keyboard.

Q4: Write a program to input the password from the keyboard. If entered correctly, print the greeting on the screen: "Welcome to my application." If entered incorrectly, print the screen asking: "Re-enter the password." If more than three times incorrect input, the program stops and prints a small prompt: "Your program is temporarily locked. Please come back in 1 hour."



Functions and Loop Statements

Functions

A function is a logical block of code that does a specific task. Optionally, a function takes zero, one, or more arguments and can return (optional) a value.

Syntax:

```
def functionName(parameters):
    statement(s)
    return value
```

def	Python keyword to define a function.
functionName	Name given to the function, using which we can call it in the program.
parameters	[Optional] Input to the function (zero, one, or more arguments).
statement(s)	Python code.
return	[Optional] statement to return something from function.

Call the Function: To call the function, use the function name, followed by parenthesis, and pass arguments if the function accepts any.

Functions

Examples

```
# Function with No Parameters
def printHelloWorld():
    print('Hello World')
# Call the Function
printHelloWorld()
```

```
# Function with one parameters
# Compute the square root of one number
def square_root(x):
    if x >= 0:
        value = x**(1/2)
    else:
        value = complex(0,abs(x)**(1/2))
    return value
# Call the Function
a = square_root(-6)
print(a)
```

Functions

Examples

```
# Function with three parameters
def sum(x, y, z):
   value = x + y + z
   return value

#calling the function
a = sum(1, 2 ,3)
print('results: ', a)
```

```
# Default Value for Parameters
def sum(x, y, z=0):
   value = x + y + z
   return value

#calling the function
a = sum(1, 2)
b = sum(1, 2, 3)
print('results with default value: ', a)
print('results with full parameters value: ', b)
```

Functions

*args parameter in a function definition allows the function to accept multiple arguments without knowing how many arguments. In other words it lets the function accept a variable number of arguments.

```
# Function with Arbitrary Arguments
def sum(*args):
    value = 0
    for x in args:
        value += x
    return value

print('Sum :', sum(1, 3))
print('Sum :', sum(1, 3, 7))
print('Sum :', sum(1, 3, 7, 5))
```

```
# Function with Arbitrary Arguments
def calculator(operation, *numbers):
    if operation == "add":
        value = 0
        for num in numbers:
            value += num
        return value

    if operation == "product":
        value = 1
        for num in numbers:
            value *= num
        return value
x = calculator("add", 2, 5, 1, 9)
print(x)
y = calculator("product", 3, 5, 2)
print(y)
```

Loop Statements

Loop statements help us to execute a block of code repeatedly in a loop until the given condition fails, or execute a block of code for each element in a given collection.

For Loop

While Loop

Break Statement

Continue Statement

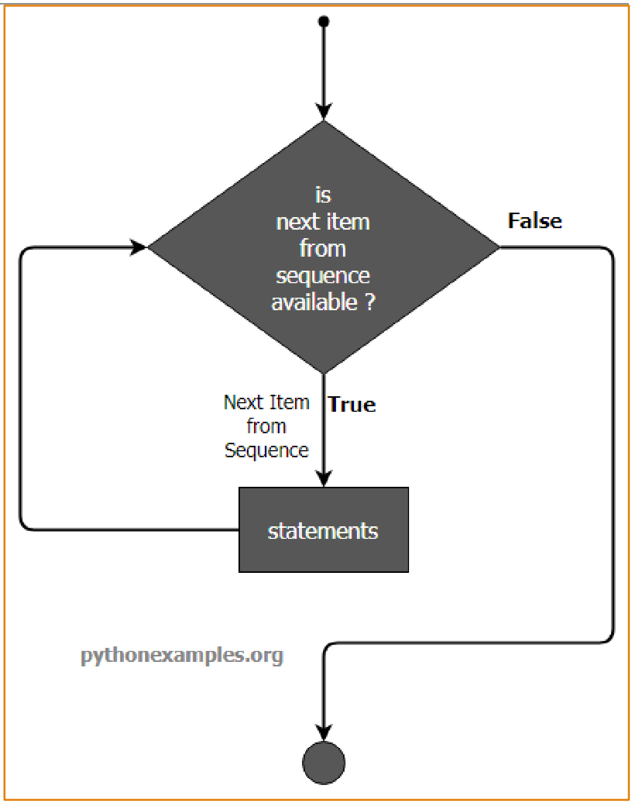
for loop

Syntax

```
for item in iterable:
    statement(s)
```

Range
List
Tuple
Dictionary
Set or a String

Python For Loop can be used to iterate a set of statements once for each item of a sequence or collection.



Flow Diagram – Python For Loop

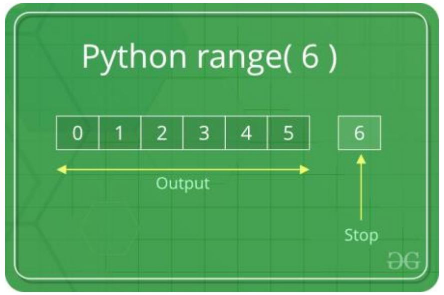
range() Function

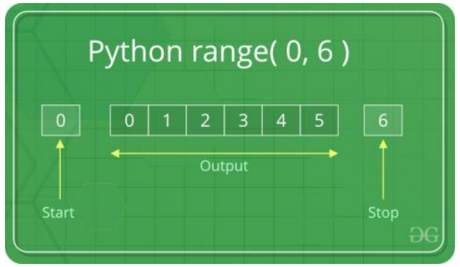
□ **Syntax:** range(*start*, *stop*, *step*) → returns a sequence of numbers

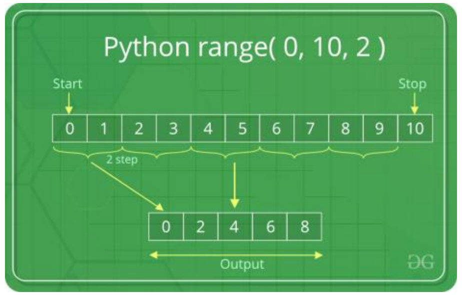
□ **Parameter Values:**

□ Python range() function takes can be initialized in 3 ways.
✓ range (stop) takes one argument → start = 0, step = 1
✓ range (start, stop) takes two arguments → step = 1
✓ range (start, stop, step) takes three arguments → step = step

NOTE:
✓ range() function only works with the integers, i.e. whole numbers.
✓ **start**, **stop** and **step** must be integers, can be positive or negative.
✓ The **step** value must not be zero.
✓ Users can access items in a range() by index, just as users do with a list. Ex: x=range()

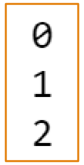






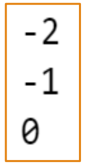
for i in range()

```
# range (stop)→ start = 0, step = 1
for x in range(3):
    print(x)
```



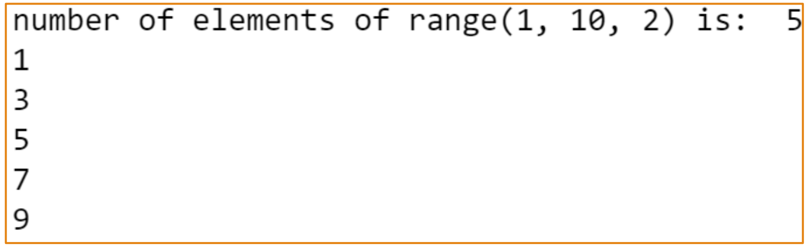
```
0
1
2
```

```
# range (,start, stop) step = 1
for x in range(-2, 1):
    print(x)
```



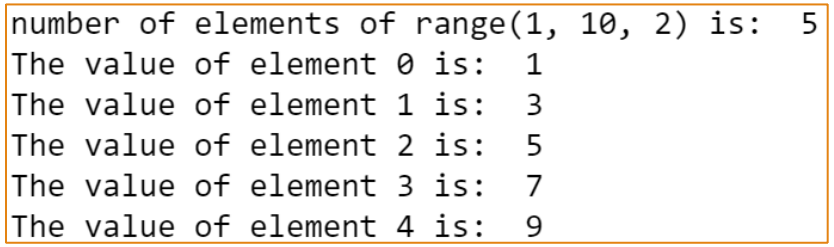
```
-2
-1
0
```

```
# range (start, stop, step)
# start = 1, stop = 10, step = 2
x = range(1, 10, 2)
print("number of elements of", x, "is: ", len(x))
for i in x:
    print(i)
```



```
number of elements of range(1, 10, 2) is:  5
1
3
5
7
9
```

```
# range (start, stop, step)
# start = 1, stop = 10, step = 2
x = range(1, 10, 2)
print("number of elements of", x, "is: ", len(x))
for i in range(len(x)):
    print("The value of element", i, "is: ", x[i])
```

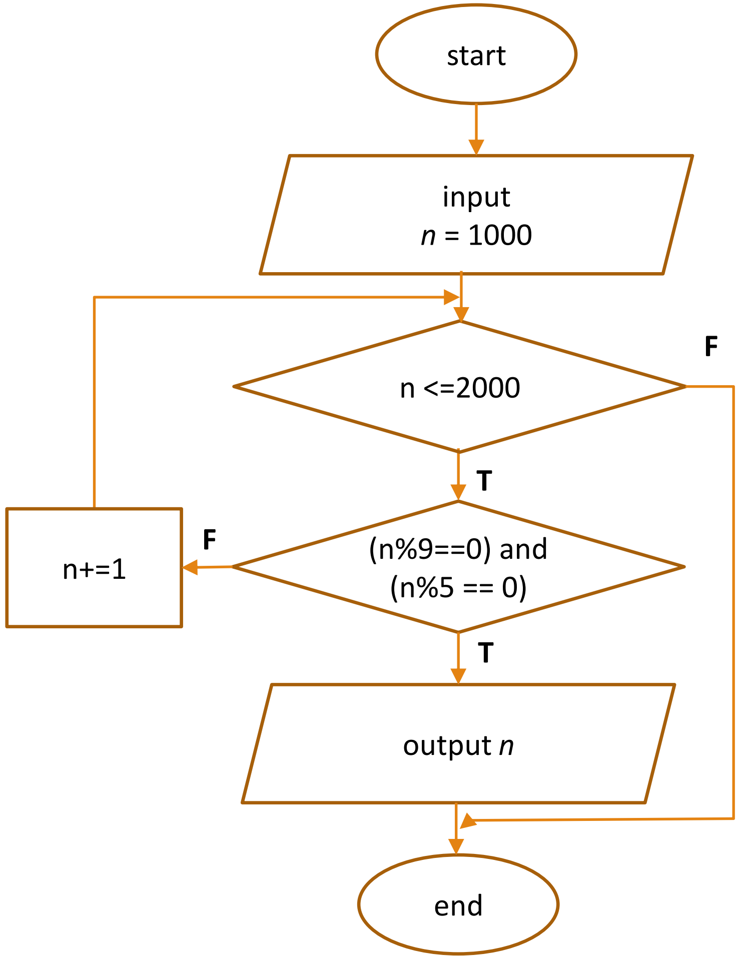


```
number of elements of range(1, 10, 2) is:  5
The value of element 0 is:  1
The value of element 1 is:  3
The value of element 2 is:  5
The value of element 3 is:  7
The value of element 4 is:  9
```

for i in range()

Problem: Find all numbers satisfies: 1/ between 1000 and 2000; 2/ divisible by 9 and multiple of 5.

```
count = 0
for n in range(1000, 2000, 1):
    if ((n%9==0) and (n%5==0)):
        print(n)
        count+=1
print("There are ", count, "number numbers between 1000 and 2000 divisible by 9 and multiple by 5")
```

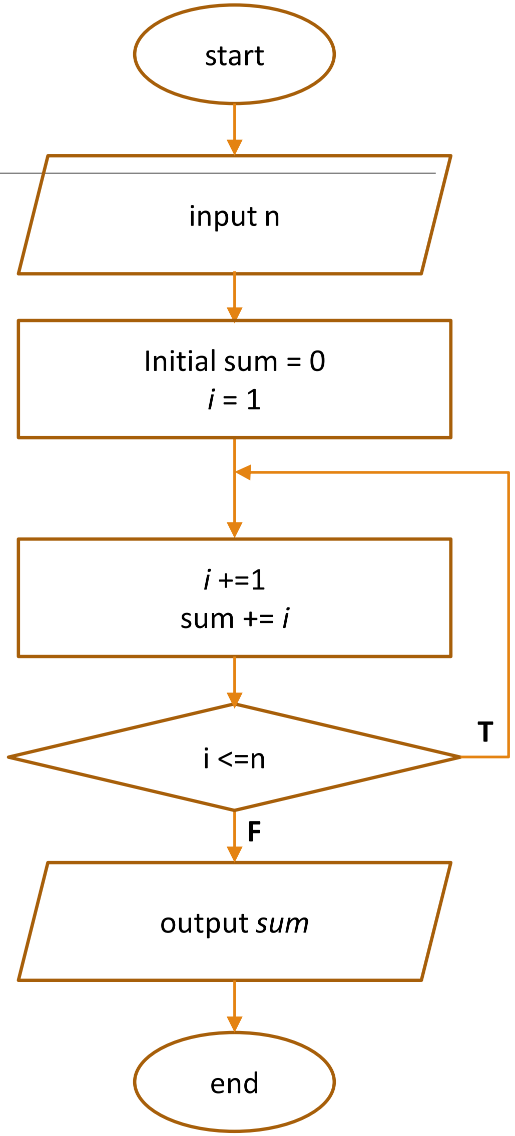


for i in range()

Problem: Write a program in Python to display the first n natural numbers and their sum.

$$\boldsymbol{S} = \mathbf{1} + \mathbf{2} + \cdots + \boldsymbol{n} = \sum_{i=1}^{n} i$$

```
sum = 0
n = int (input("Input the natural number: "))
print("sum = ", end = "")
for i in range(1,n + 1):
    sum += i
    if i<n:
        print(i, "+", end = " ")
    else:
        print(i, end = " ")
print("=", sum)
```

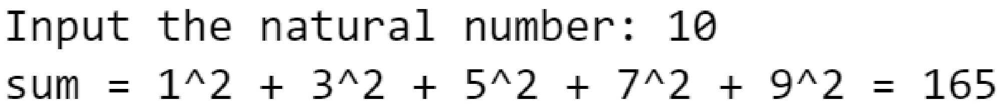


for i in range()

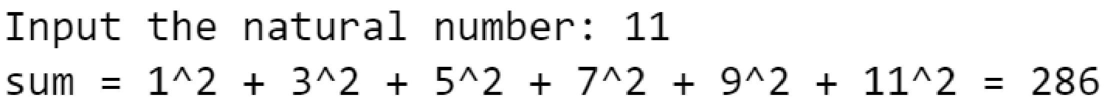
Problem: Write a program in Python to display the odd natural number smaller or equal inputted natural number n and their square sum.

$$\boldsymbol{S} = \mathbf{1}^2 + \mathbf{3}^2 + \cdots + \boldsymbol{n}^2$$ (n is odd number)

$$\boldsymbol{S} = \mathbf{1}^2 + \mathbf{3}^2 + \cdots + (\boldsymbol{n} - \mathbf{1})^2$$ (n is even number)

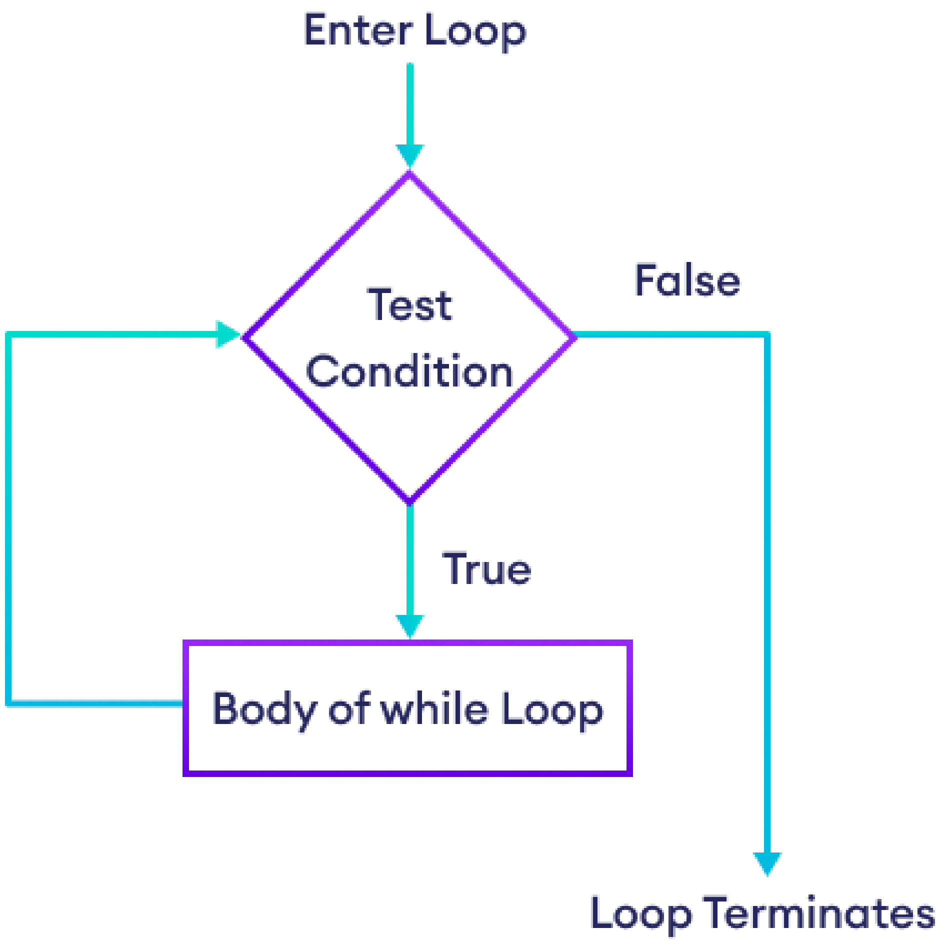


```
Input the natural number: 10
sum = 1^2 + 3^2 + 5^2 + 7^2 + 9^2 = 165
```



```
Input the natural number: 11
sum = 1^2 + 3^2 + 5^2 + 7^2 + 9^2 + 11^2 = 286
```

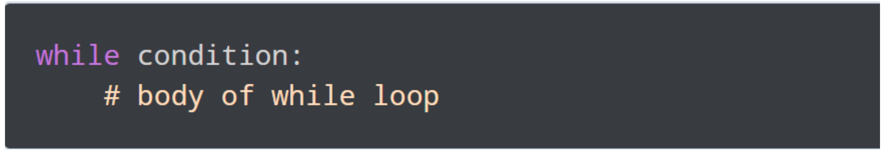
```
def square (i):
    return i**2

sum = 0
n = int (input("Input the natural number: "))
print("sum = ", end = "")
for i in range(1,n + 1):
    if i % 2 != 0:
        sum += square(i)
        if n%2 != 0:
            if i < n:
                print(i, "\b^2 +", end = " ")
            else:
                print(i, "\b^2", end = " ")
        else:
            if i < n-1:
                print(i, "\b^2 +", end = " ")
            else:
                print(i, "\b^2", end = " ")
print("=", sum)
```


Enter Loop
Test Condition
False
True
Body of while Loop
Loop Terminates

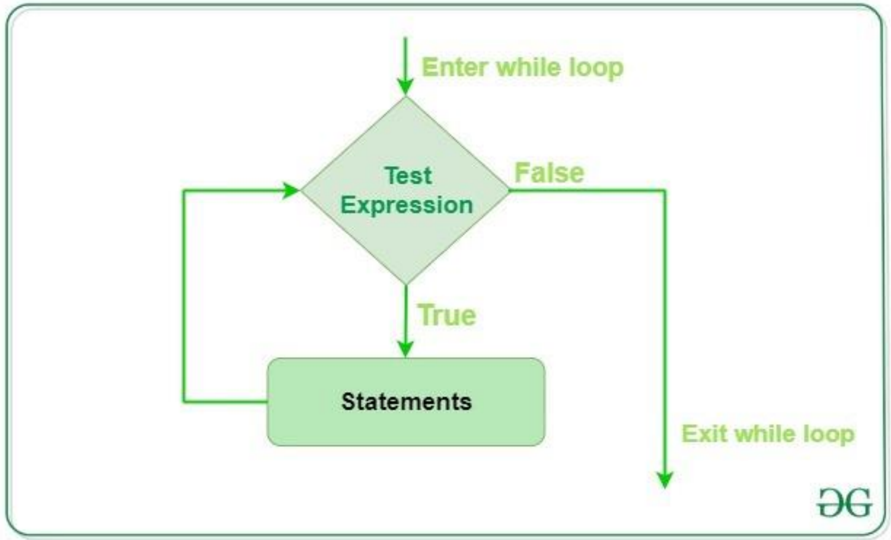
while Loop

while Loop

while loop is used to run a specific code until a certain condition is met.



```
while condition:
    # body of while loop
```



Here,

1. A while loop evaluates the condition
2. If the condition evaluates to True, the code inside the while loop is executed.
3. condition is evaluated again.
4. This process continues until the condition is False.
5. When condition evaluates to False, the loop stops.

while Loop

```
# range (stop)→ start = 0, step = 1
for x in range(3):
    print(x)
```

```
# range (,start, stop) step = 1
for x in range(-2, 1):
    print(x)
```

```
x = 0
while x<3:
    print(x)
    x+=1
```

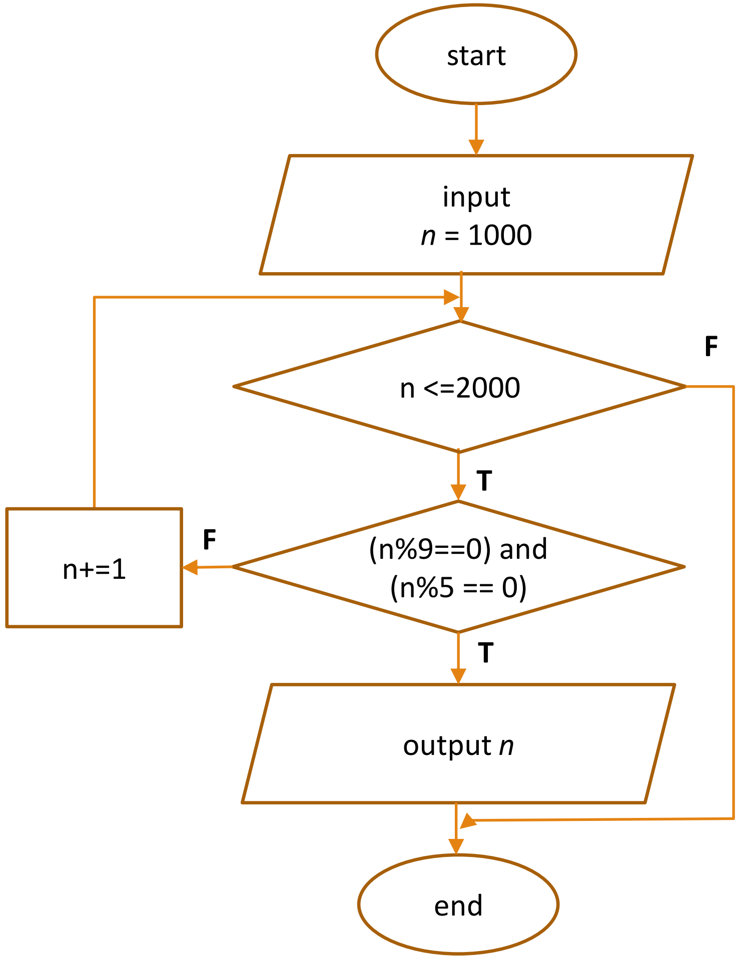
```
x = -2
while x<1:
    print(x)
    x+=1
```

while Loop

Problem: Find all numbers satisfies: 1/ between 1000 and 2000; 2/ divisible by 9 and multiple of 5.

```
count = 0
for n in range(1000, 2000, 1):
   if ((n%9==0) and (n%5==0)):
      print(n)
      count+=1
print("There are ", count, "number numbers between 1000 and 2000 divisible by
9 and multiple by 5")
```

```
count = 0
n = 1000
while (n<= 2000):
   if ((n%9==0) and (n%5==0)):
      print(n)
      count+=1
   n+= 1
print("There are ", count, "number numbers between 1000 and 2000 divisible by
9 and multiple by 5")
```



while Loop

Problem: Write a program in Python to display the first n natural numbers and their sum.

$$\boldsymbol{S} = \mathbf{1} + \mathbf{2} + \cdots + \boldsymbol{n} = \sum_{i=1}^{n} i$$

```
sum = 0
n = int (input("Input the natural number: "))
print("sum = ", end = "")
for i in range(1,n + 1):
    sum += i
    if i<n:
        print(i, "+", end = " ")
    else:
        print(i, end = " ")
print("=", sum)
```

```
sum = 0
n = int (input("Input the natural number: "))
print("sum = ", end = "")
i = 1
while i<=n:
#for i in range(1,n + 1):
    sum += i
    if i<n:
        print(i, "+", end = " ")
    else:
        print(i, end = " ")
    i+=1
print("=", sum)
```

while Loop

Problem: Write a program in Python to display the odd natural number smaller or equal inputted natural number n and their square sum.

$\boldsymbol{S = 1^2 + 3^2 + \cdots + n^2}$ (n is odd number)

$\boldsymbol{S = 1^2 + 3^2 + \cdots + (n-1)^2}$ (n is even number)

```
def square (i):
    return i**2

sum = 0
n = int (input("Input the natural number: "))
print("sum = ", end = "")
for i in range(1,n + 1):
    if i % 2 != 0:
        sum += square(i)
        if n%2 != 0:
            if i < n:
                print(i, "\b^2 +", end = " ")
            else:
                print(i, "\b^2", end = " ")
        else:
            if i < n-1:
                print(i, "\b^2 +", end = " ")
            else:
                print(i, "\b^2", end = " ")
print("=", sum)
```

```
def square (i):
    return i**2

sum = 0
n = int (input("Input the natural number: "))
print("sum = ", end = "")
i = 0
while i<=n:
#for i in range(1,n + 1):
    if i % 2 != 0:
        sum += square(i)
        if n%2 != 0:
            if i < n:
                print(i, "\b^2 +", end = " ")
            else:
                print(i, "\b^2", end = " ")
        else:
            if i < n-1:
                print(i, "\b^2 +", end = " ")
            else:
                print(i, "\b^2", end = " ")
    i+=1
print("=", sum)
```

Loop with condition in the middle

```
# An example of infinite loop
# press Ctrl + c to exit from the loop
password = "Taishan@"
while True: # infinite loop
    user_pass = input("Enter user pasword: ")
    # condition in the middle
    if user_pass == password:
        print("Well come Taishan High School!")
        break
    else:
        print("Wrong password. Try again!")
```

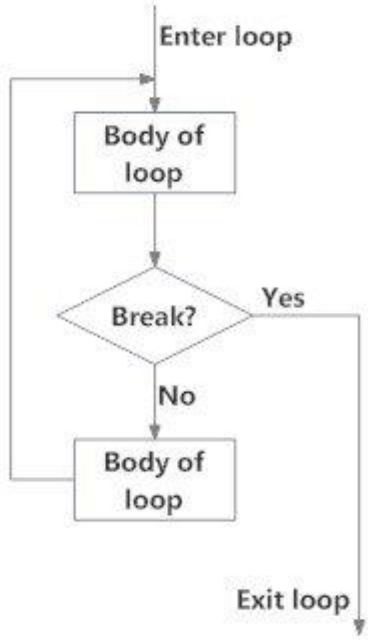
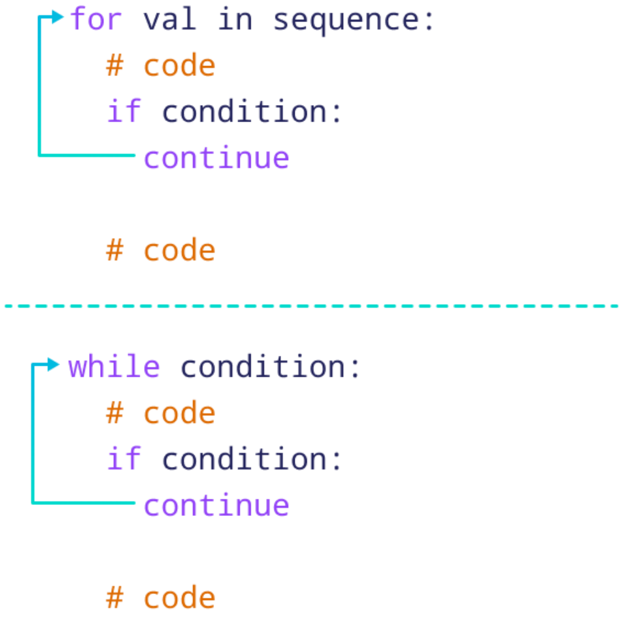


Fig: loop with condition in middle

Loop with condition in the middle

```
count = 0
password = "Taishan@"
while True: # infinite loop
    user_pass = input("Enter user pasword: ")
    count += 1
    # condition in the middle
    if user_pass == password:
        print("Wellcome Taishan High School!")
        break
    else:
        print("Wrong password. Try again!")
        #count += 1
    if count < 3:
        continue
    else:
        print("Your account is locked. Try again after 3 hours!")
        break
```



```
for val in sequence:
    # code
    if condition:
        continue

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```

Loop with condition in the middle

A Robot moves in the plane starting from the first point (0,0). The robot can move in the direction of UP, DOWN, LEFT, and RIGHT with specific steps. The robot movement marks are displayed as follows:

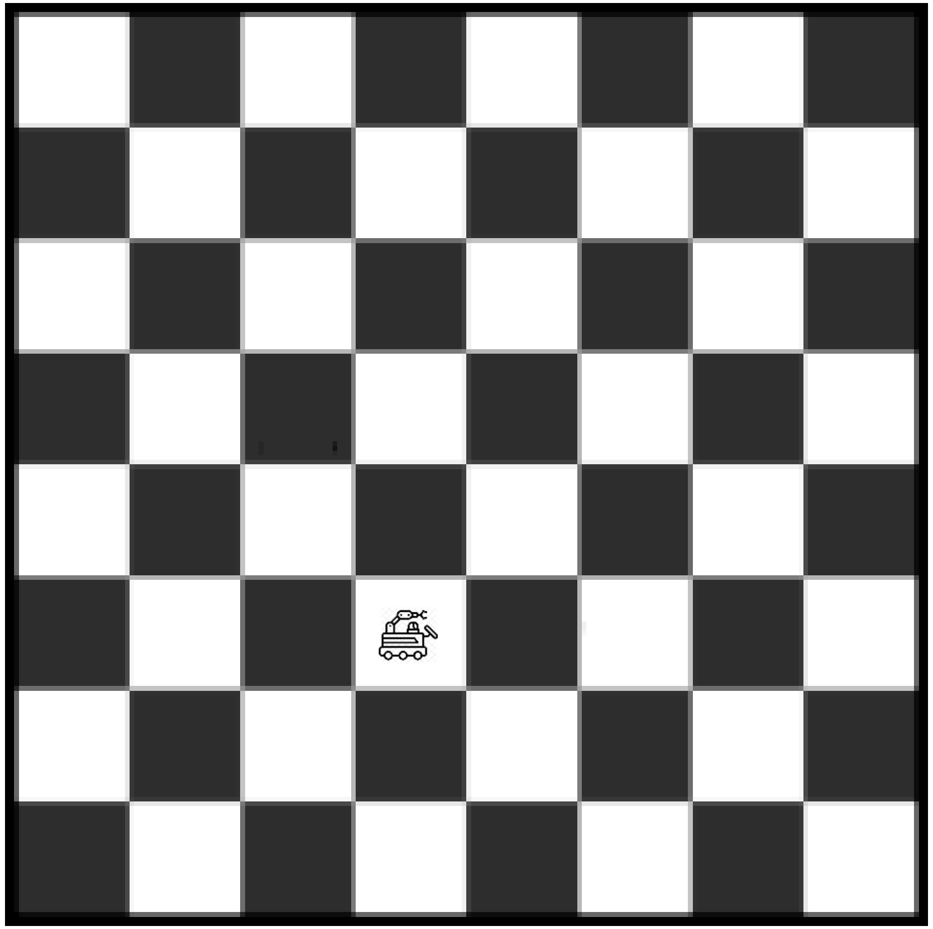
UP 5
DOWN 3
LEFT 3
RIGHT 3

The numbers behind the movement direction are the number of steps. The End of movement is confirmed with 'OK'. Write a program to calculate the distance from the current position to the last position after the robot has moved. If the distance is a decimal, print the nearest integer.

Example: If the following tuple is the input of the program:

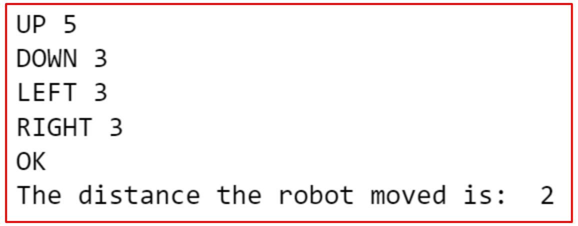
UP 5
DOWN 3
LEFT 3
RIGHT 2
OK

then the output will be 2.

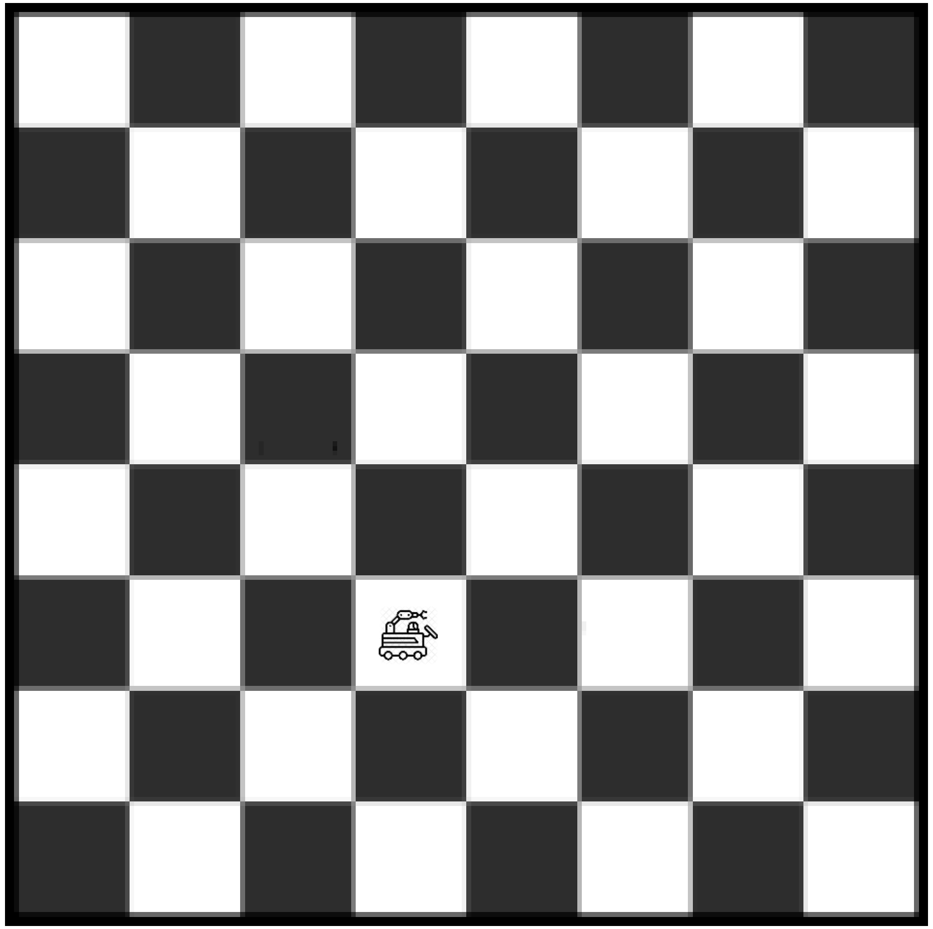


Loop with condition in the middle

```
import math
pos = [0,0]
while True:
    s = input()
    if s == 'OK':
        break
    movement = s.split(" ")
    direction = movement[0]
    steps = int(movement[1])
    if direction=="UP":
        pos[0]+=steps
    elif direction=="DOWN":
        pos[0]-=steps
    elif direction=="LEFT":
        pos[1]-=steps
    elif direction=="RIGHT":
        pos[1]+=steps
print ("The distance the robot moved is: ", int(round(math.sqrt(pos[1]**2+pos[0]**2))))
```



```
UP 5
DOWN 3
LEFT 3
RIGHT 3
OK
The distance the robot moved is:  2
```



References

- ❑ https://realpython.com/what-can-i-do-with-python/

- ❑ https://www.w3schools.com/python/python_getstarted.asp

- ❑ https://www.programiz.com/python-programming/first-program

- ❑ https://www.geeksforgeeks.org/

Thank you for listening!