Lab1
Python and PyCharm IDE Setup

Outline

- Installing python
- Installing PyCharm
- Choosing interpreter and installing packages
- Your first python program
- Debugger

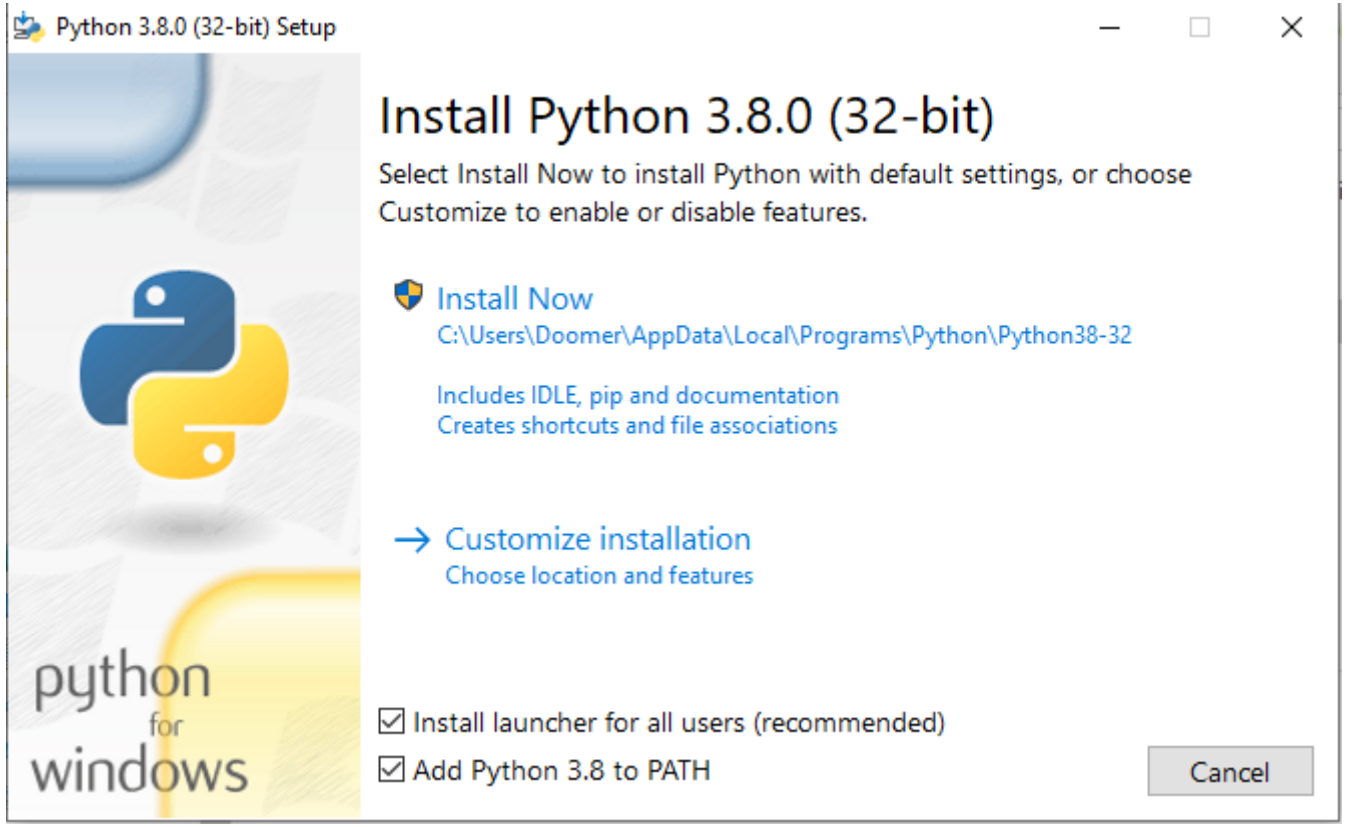
Python

- Python is a interpreted, high-level, general-purpose programming language.

Installing Python

- Go to https://www.python.org/downloads/, the Python organization website, and click on “Download Python x.x.x” button, which downloads the latest offered version.

- When the executable file is downloaded, run it.
- Make sure to enable “Add Python x.x to PATH” in this window, then click on “Install Now”.



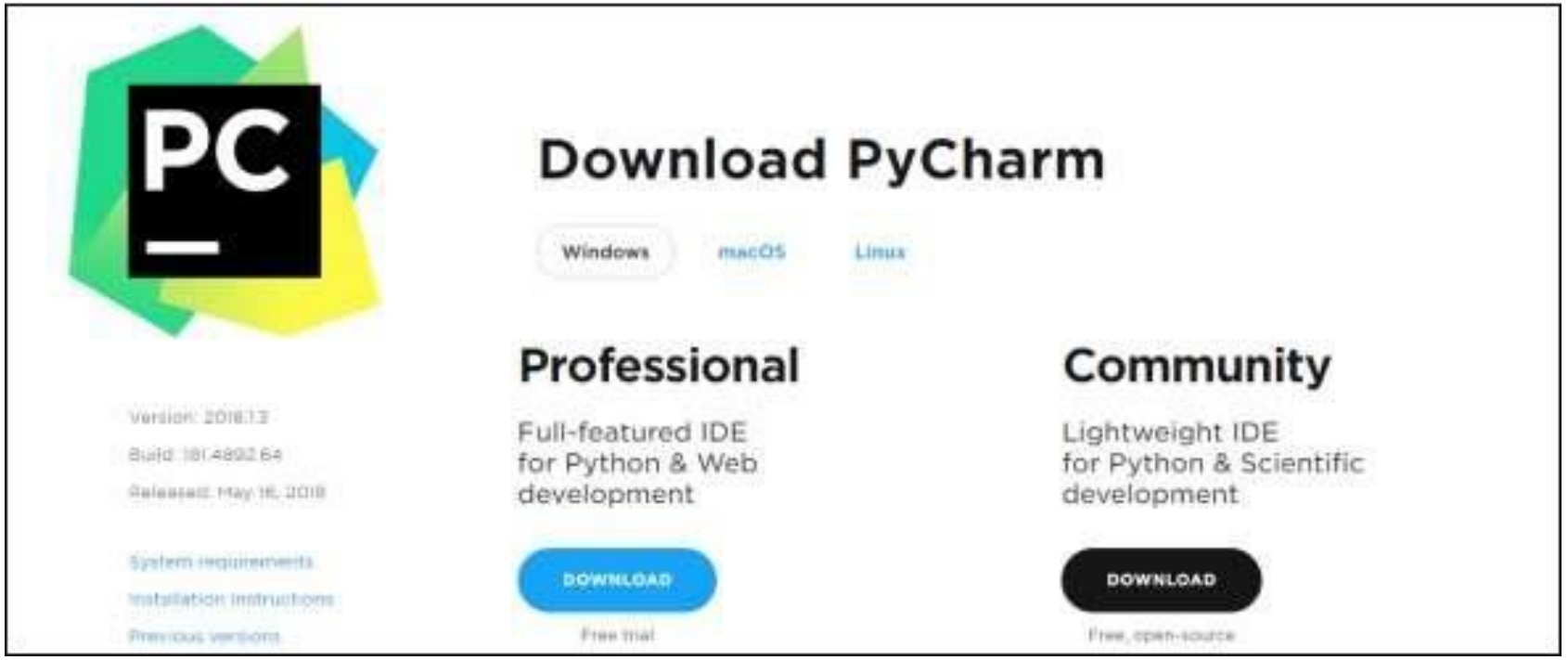
PyCharm

It is a integrated development environment (IDE) used in computer programming, specifically for Python.

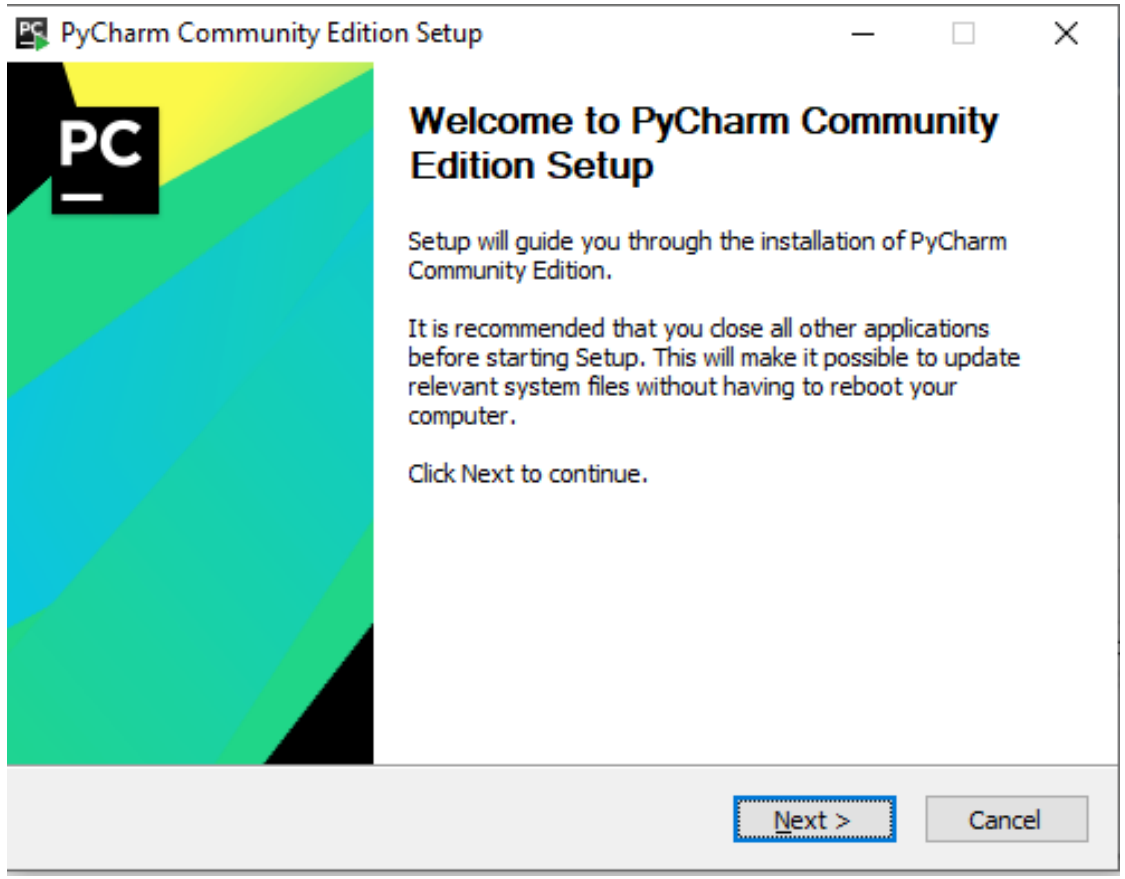
Installing PyCharm

- Go to https://www.jetbrains.com/pycharm/download/#section=windows

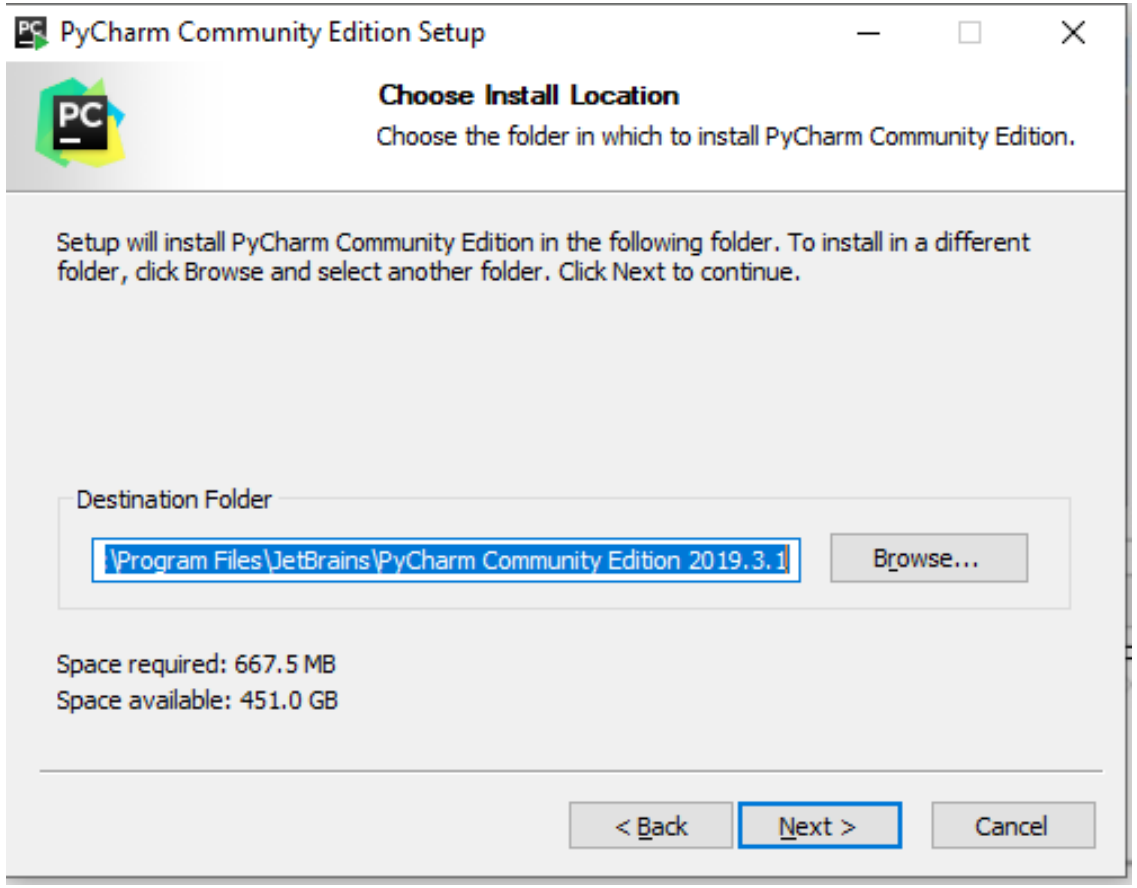
And download community version package



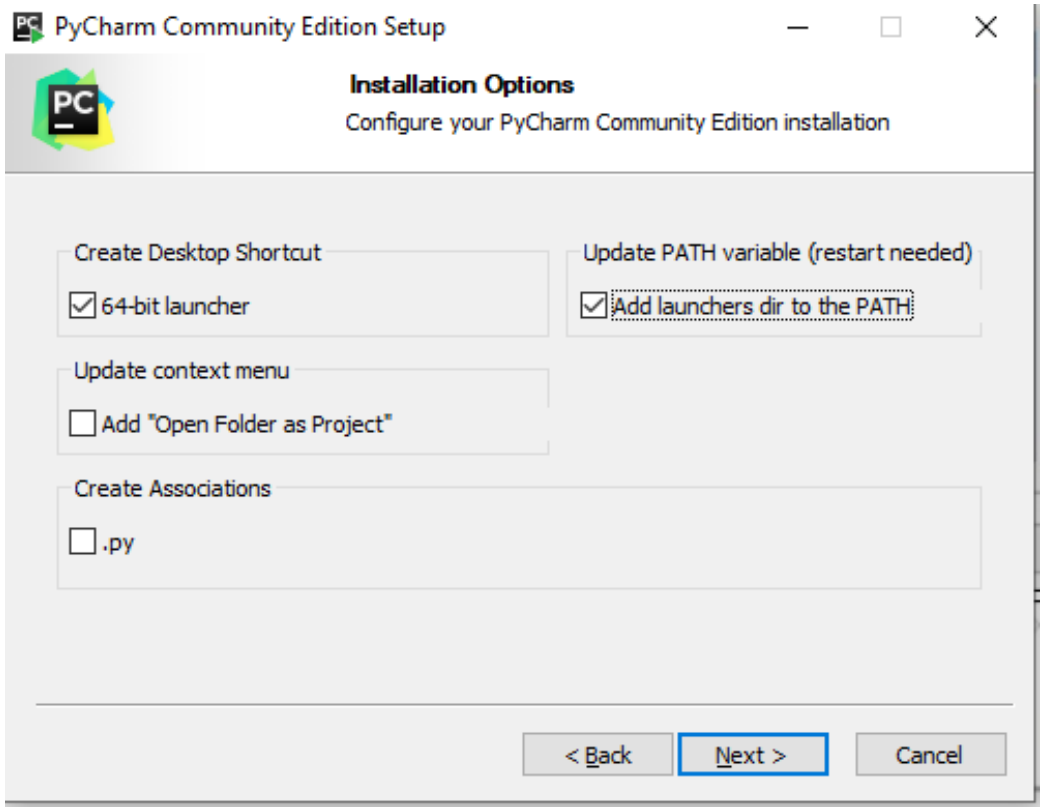
- Run the executable file

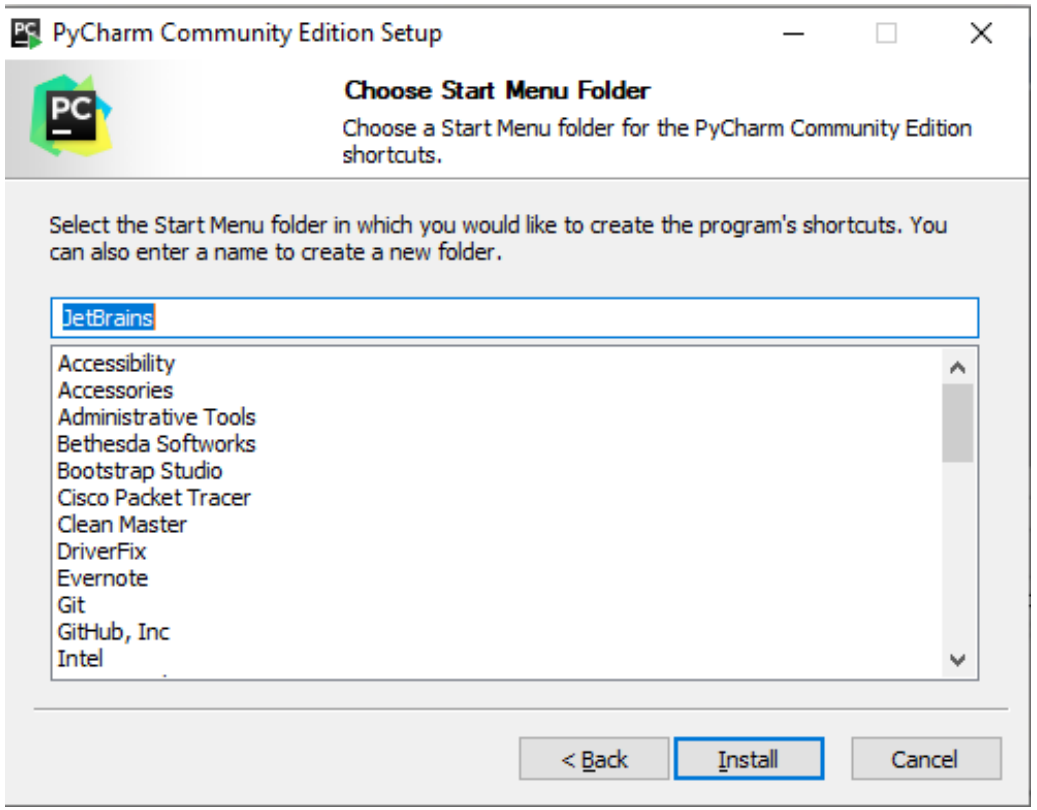


- Choose destination folder to install

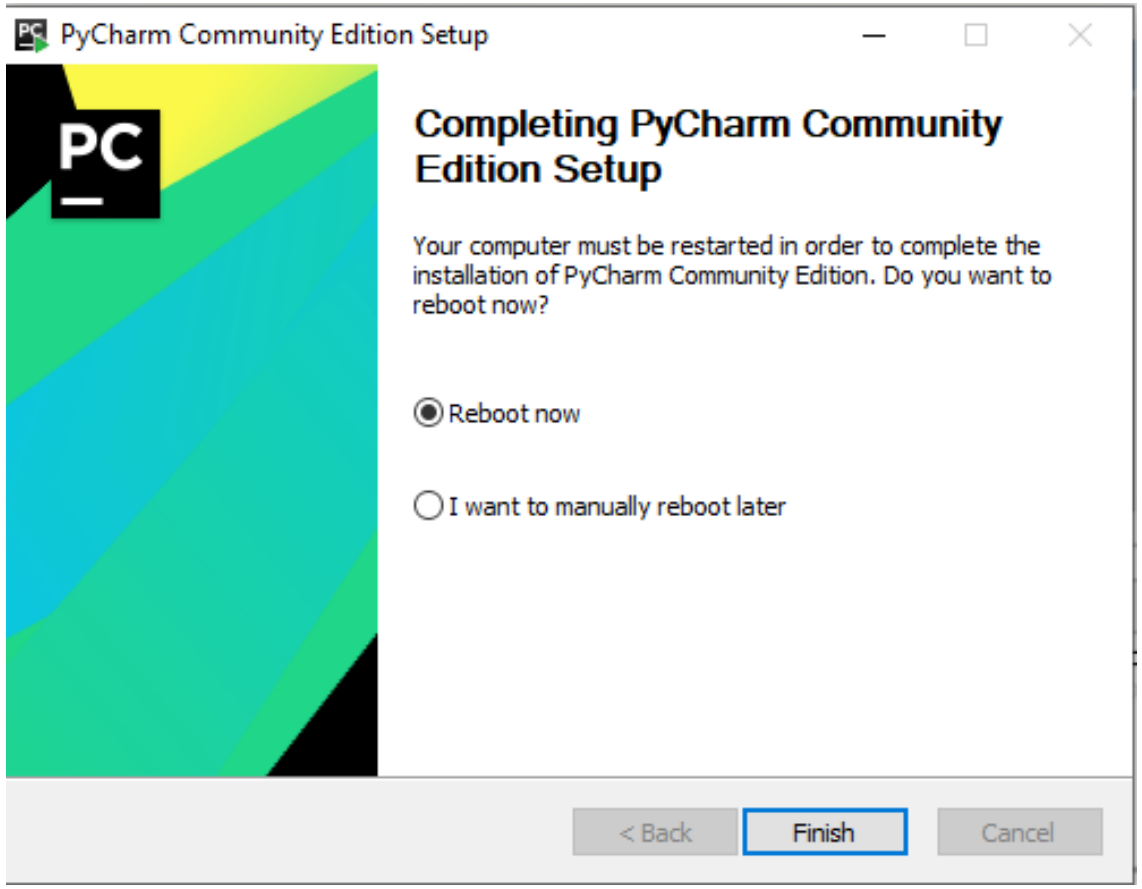


- When installing finishes, click next on these windows.



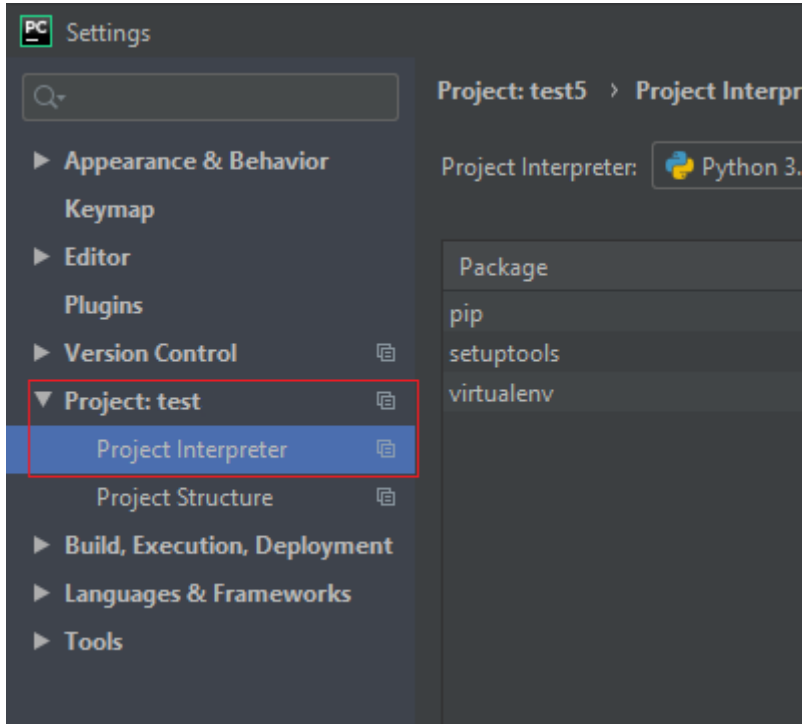


- Reboot your computer

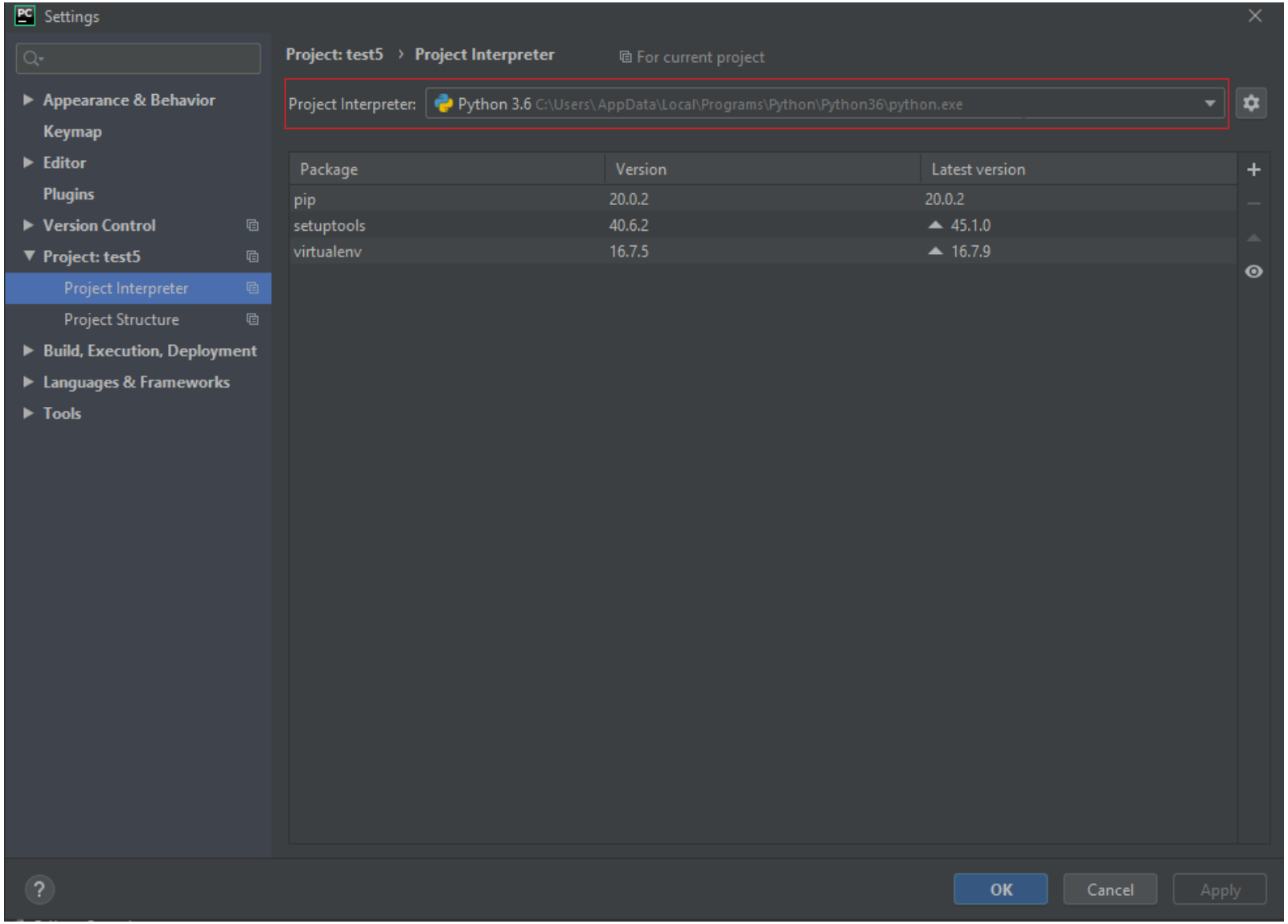


Choosing interpreter and installing packages

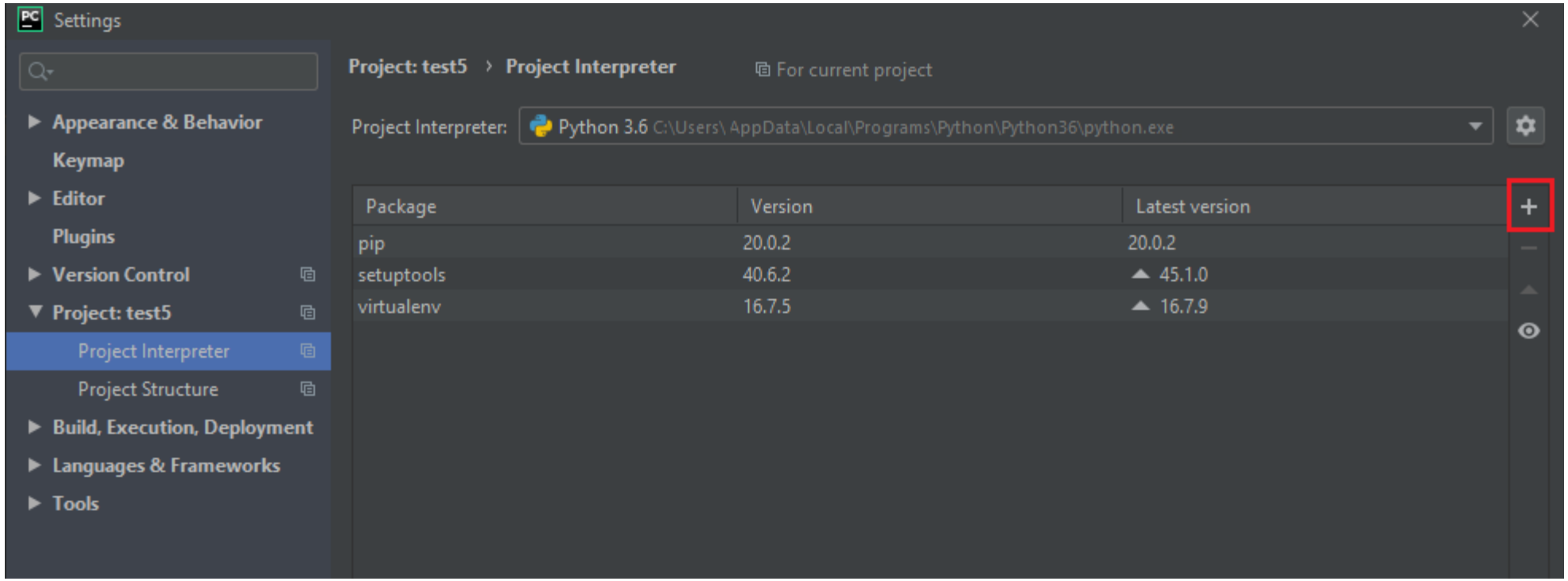
- Interpreter is a program that reads and executes code.
- To choose interpreter, open PyCharm and create a new project.
- File list -> Settings, then click on Project Interpreter under Project: ProjectName list



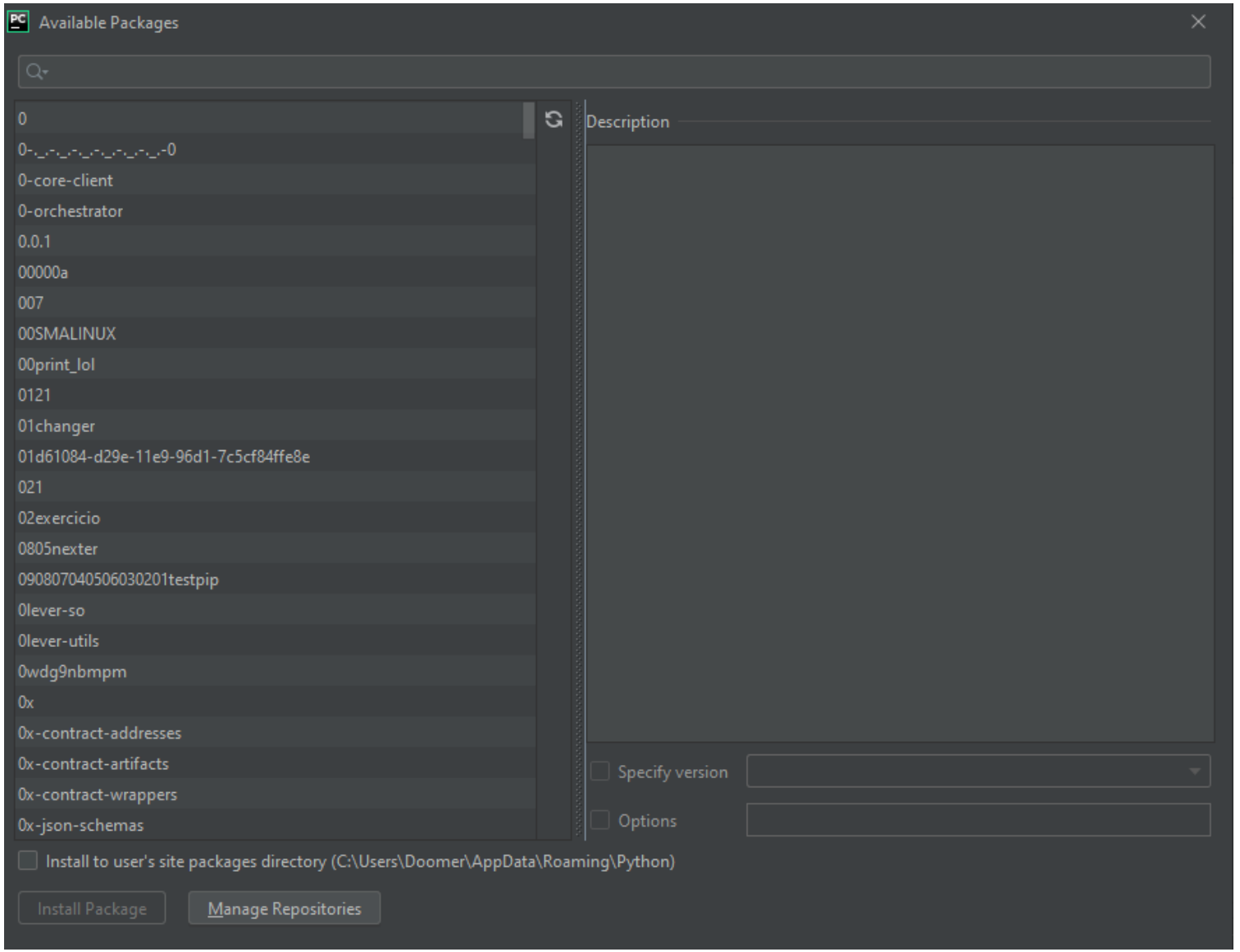
- Then choose an interpreter from the list in the top which represents the global python interpreter installed.



- To install packages to use in your project, go to the same Project Interpreter window as you did before.
- You can see the packages you have in a list. To add more, click on the + sign on the right side.

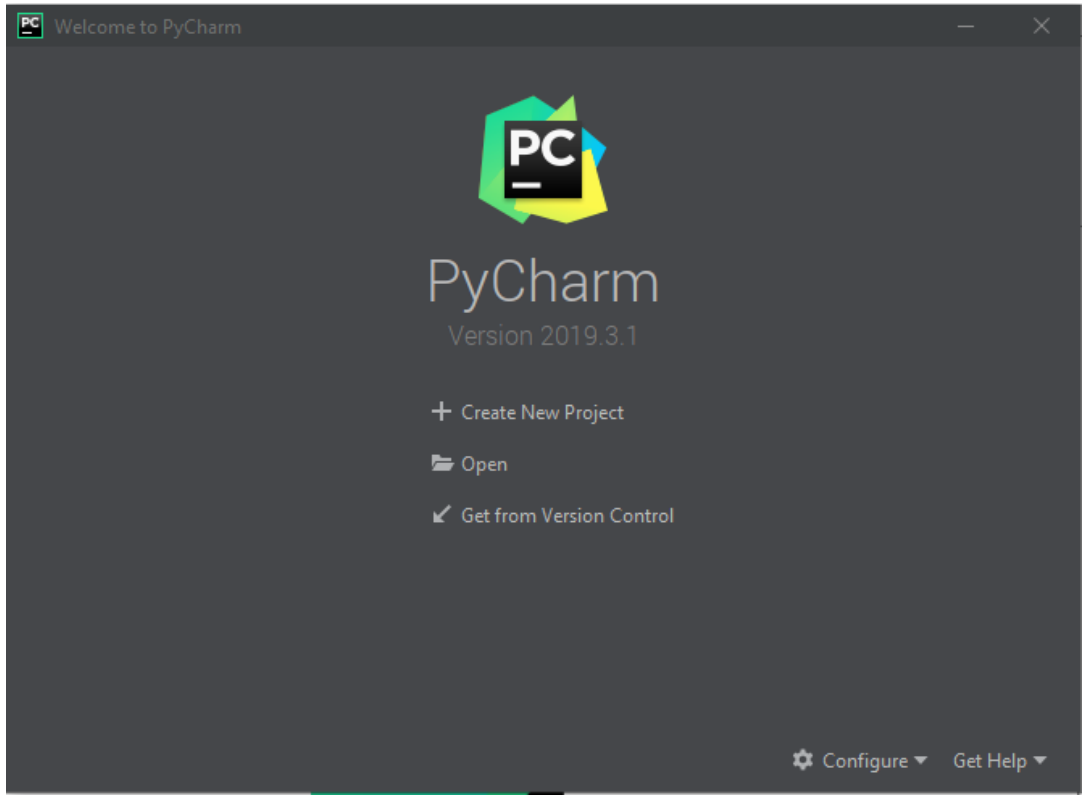


- Write the name of the package you want in the search bar, a list of packages appears, click on the one you want then click the Install Package button.

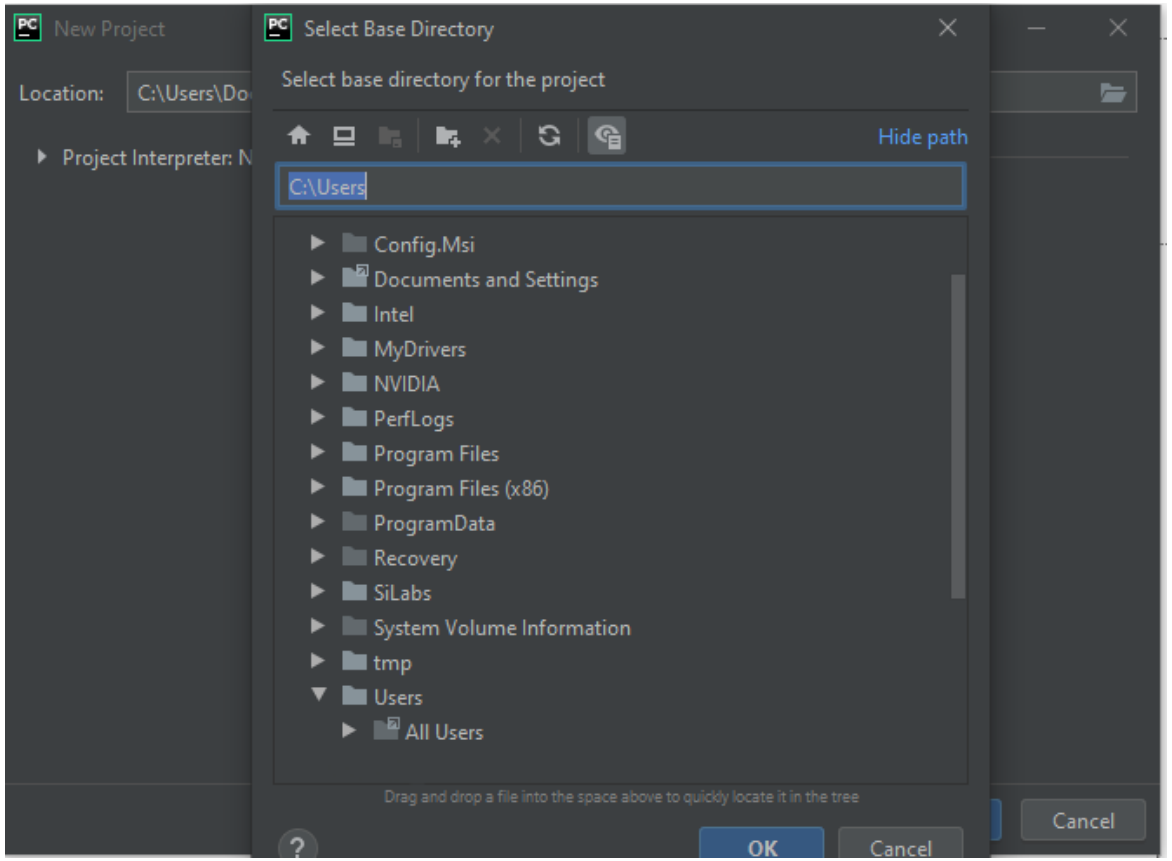


Your first python program

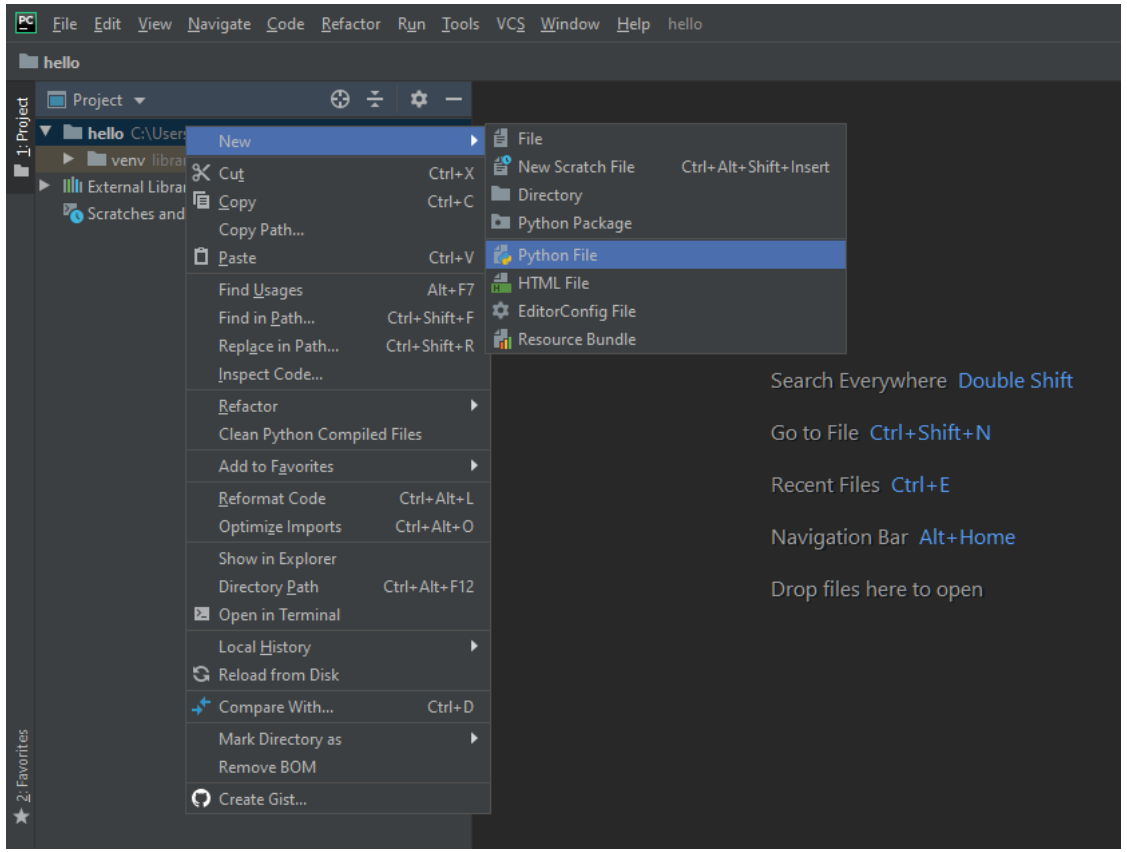
- Run the PyCharm application, then click on Create New Project



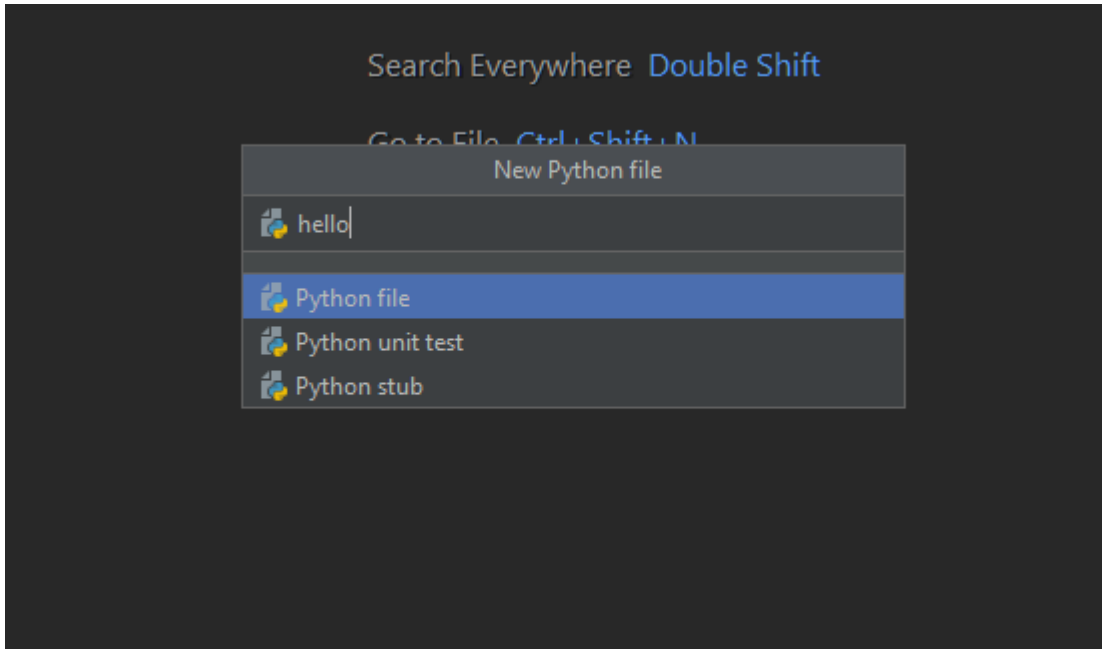
- Choose or create a folder



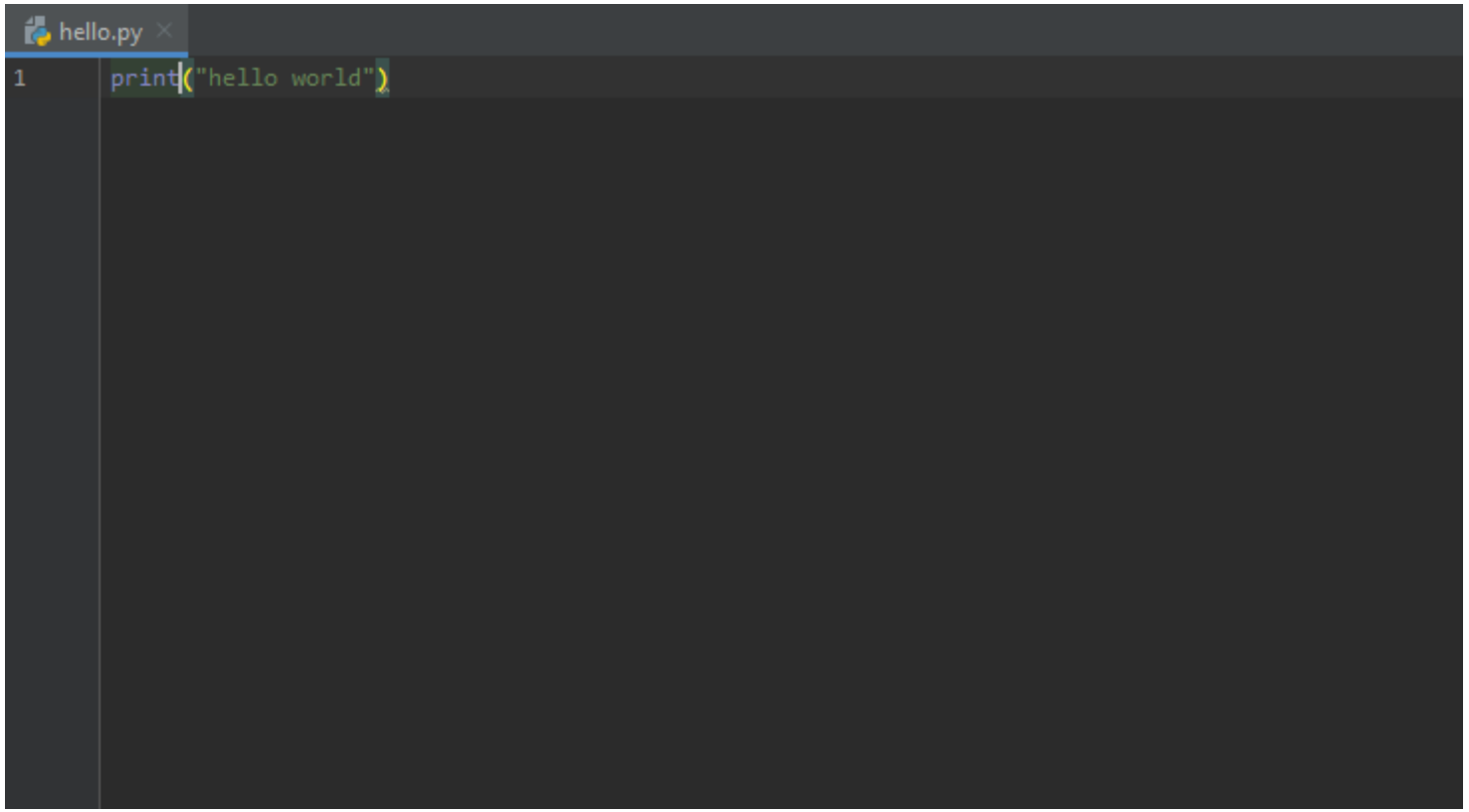
- Create a new python file by right clicking the name of the folder, choose New then choose Python File.



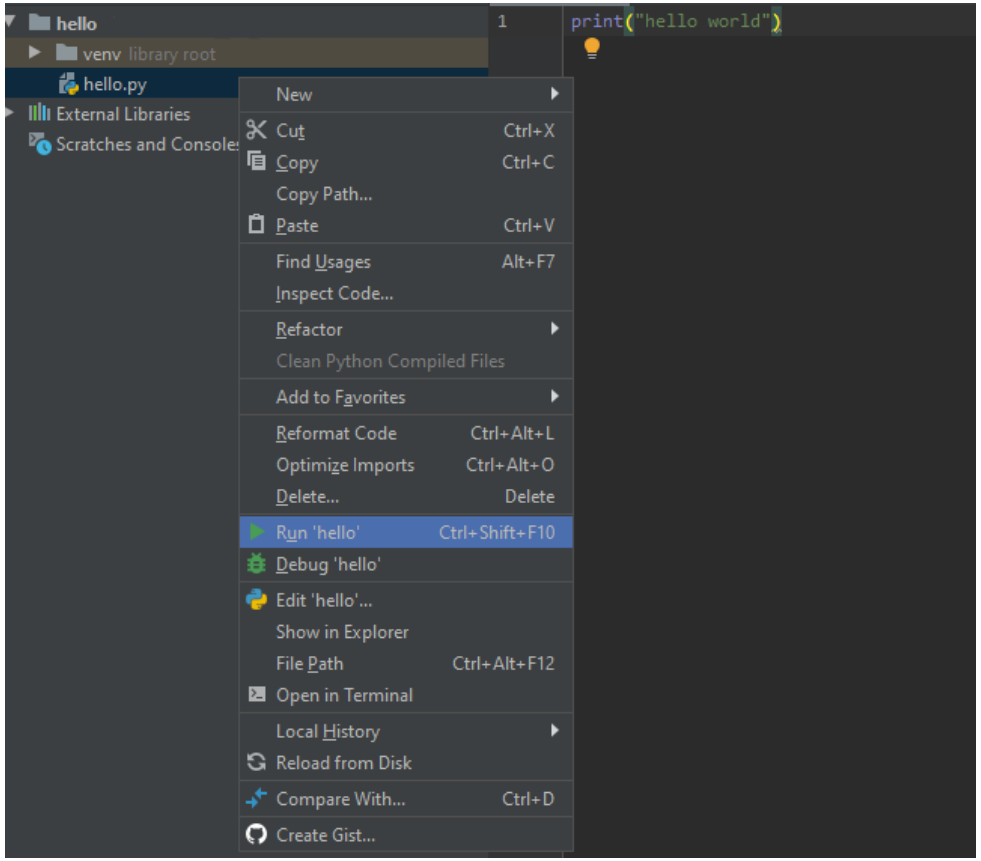
- Name your file



- A print() function prints a message to the screen



- To run the program, right click on the file name then choose Run ‘hello’.

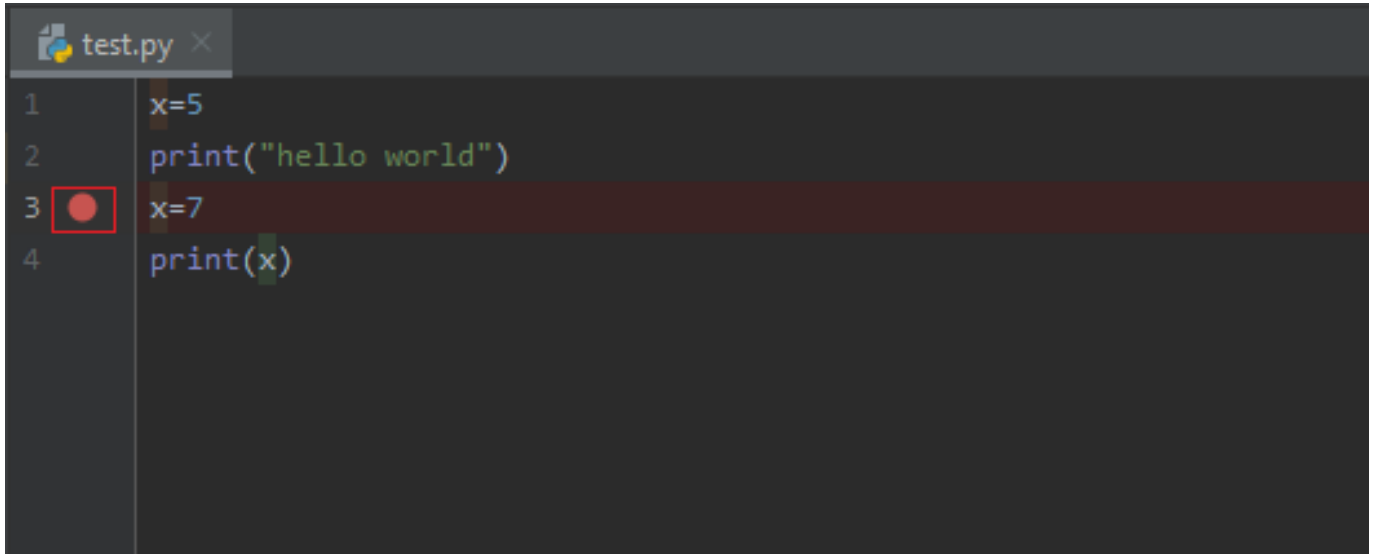


- The result will appear on the console.

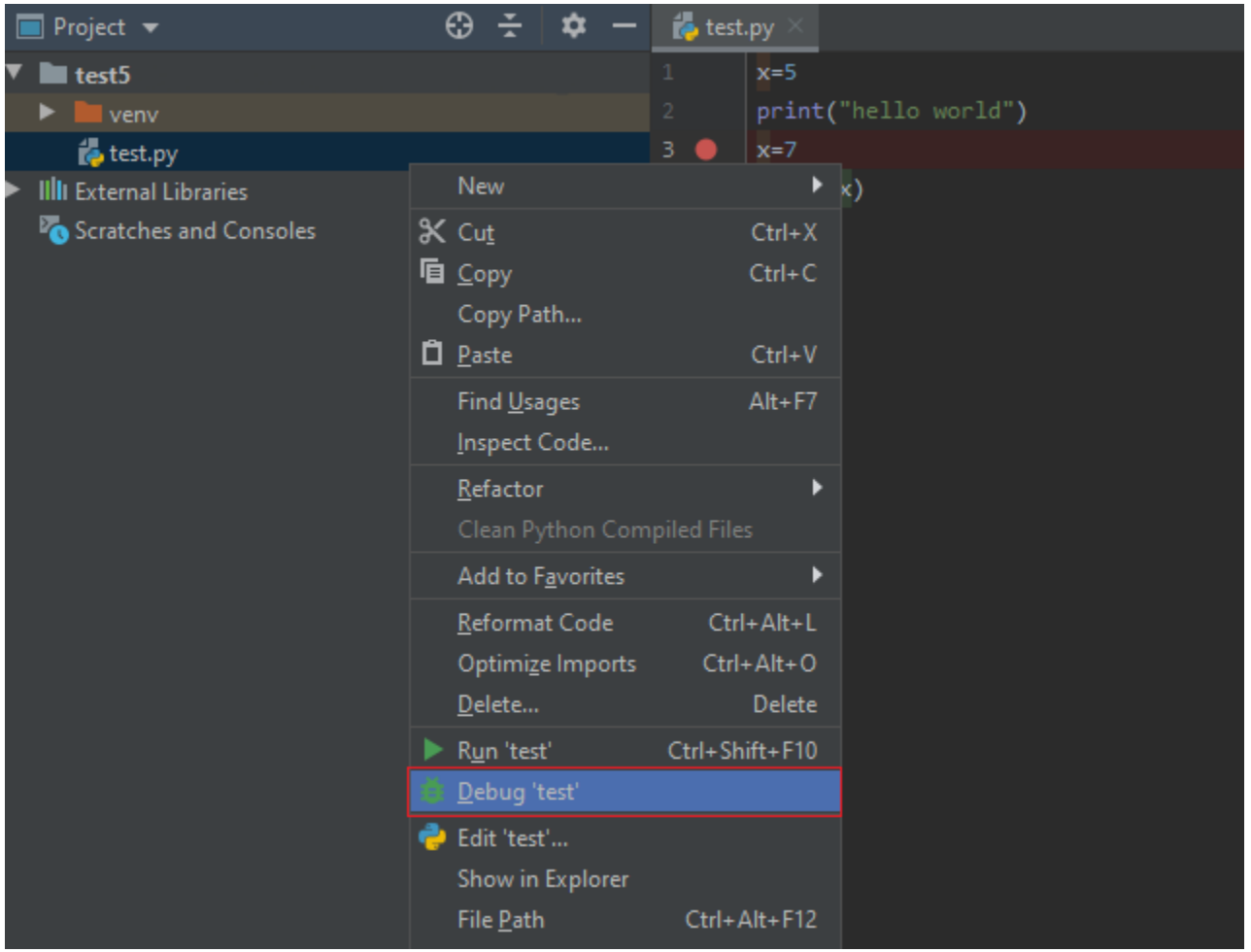
```
hello world

Process finished with exit code 0
```

Debugger

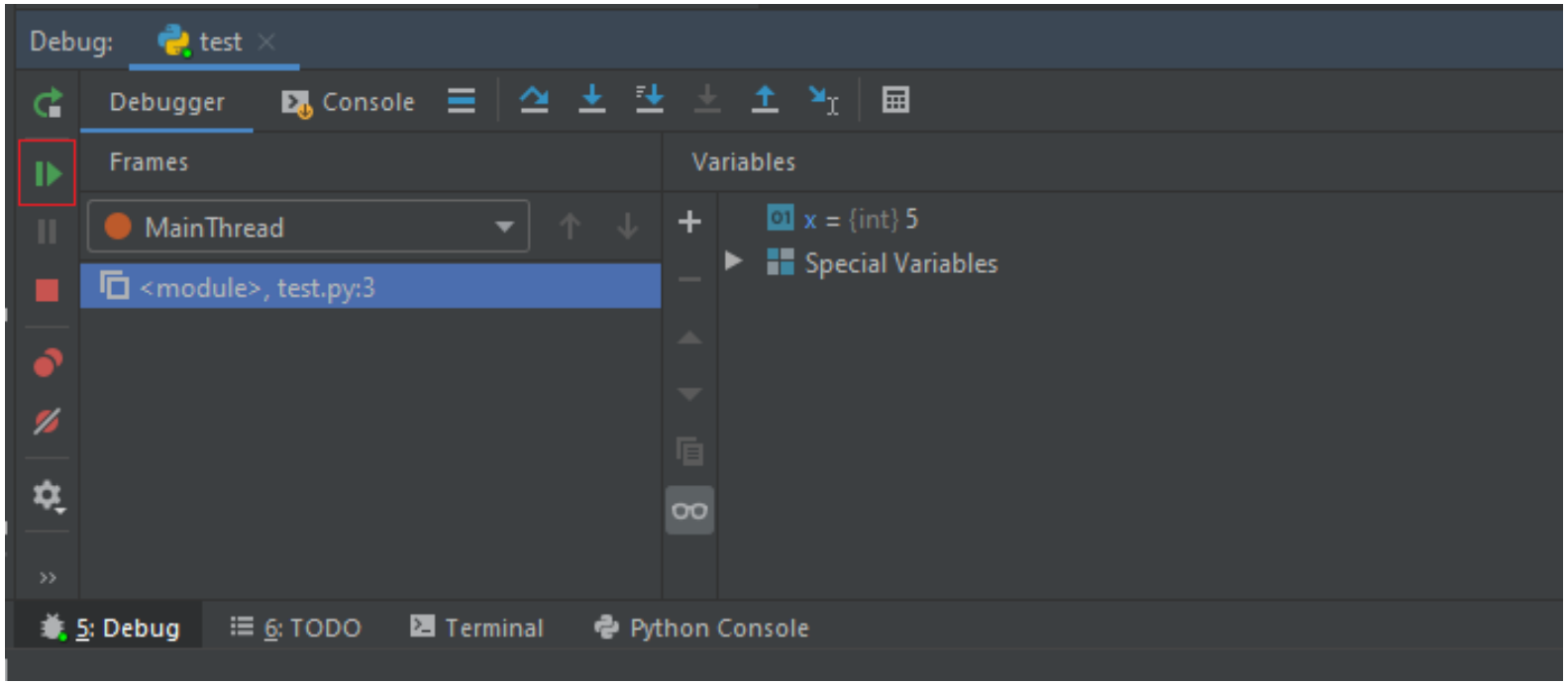
- A tool used to test and find bugs in programs.
- Place breakpoints, which work as stop signs, by clicking the space to the left of the line.



- To start the debugger, right click on the name of the file in browsing list, then click on debug ‘FileName’.



- A window appears in the lower side of the screen, click on the Resume program button to continue after the breakpoint where the program stopped.
- Note you can see the value of the variables and how it changes during the program.



Chapter 2
Data Types and Variables

Prepared by

Dr.Mohammad Abdel-Majeed, Eng.Abeer Awad and Ayah Alramahi

Outline

- Print Statement
- Variables
- Numeric Variables
- Boolean Variables
- Strings
- User Input

Hello World!

- Python files have the extension *.py*
- First program using **print()** function

  >>>**Print(“Hello World”)**

  Output: Hello World

- Variables: Used to hold a **value**

  >>>message = "Hello Python world!"
  >>>print(message)

  Output: Hello Python world!

Hello World in PyCharm

```
message = “Hello Python world!”
print(message)
```

```
Hello Python world!
```

Variables

- A Variable: is a named place in the **memory** where a programmer can store data and later retrieve the data using the variable "name"
- You can change the contents of a variable in a later statement

```
X = 5
Y = 6
print(X)
print(Y)
X = 10
print(X)
print(Y)
```

```
5
6
10
6
```

Naming and Using Variables

- Case Sensitive
- Variable names can contain only letters, numbers, and underscores.
- They can start with a letter or an underscore, but not with a number. For instance, you can call a variable *message_1* but not *1_message*.
- Spaces are not allowed in variable names
  - underscores can be used to separate words in variable names.
  - For example: ***greeting_message Vs greeting message***
- Avoid using Python keywords and function names as variable names
- Variable names should be short but descriptive.
  - *name* VS N
  - *student_name* VS *s_n*,
  - *name_length* Vs *length_of_persons_name*.

Reserved Words

and del for is raise
assert elif from lambda return
break else global not try
class except if or while
continue exec import pass yield
def finally in print

Naming errors

```
message = “Hello\nPython\tworld!”
print(mesage)
```

```
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    print(mesage)
NameError: name 'mesage' is not defined
```

Assignment Operator

- We assign a value to a variable using the assignment statement (=)
- An assignment statement consists of an expression on the right hand side and a variable to store the result

$$Y = X * 10 + 15$$
$$Lab\_name = "Computer\ Application\ Lab"$$

Variable Types

- Variable type is based on the data stored(assigned) to the variable

```
X = "5"
print(type(X))
X = 10
print(type(X))
X = 10.0
print(type(X))
X = True
print(type(X))
```

```
<class 'str'>
<class 'int'>
<class 'float'>
<class 'bool'>
```

Numeric Variables

- Hold Integer or Float types
- Numeric operators are applied
- Precedence rules (Highest to lowest)
  - Parenthesis
  - Exponentiation (raise to a power)
  - Multiplication, Division, and Remainder
  - Addition and Subtraction
  - Left to right

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Power
%	Remainder

Numeric Expressions

- When you perform an operation where one operand is an integer and the other operand is a floating point the result is a floating point
- The integer is converted to a floating point before the operation

```
x = 15
y = 4
y_float = 4.0
print(x + y)
print(x - y)
print(x % y)
print(x * y)
print(x ** y)
print(x / y_float)
print(x / y)
print(x // y)
```

```
19
11
3
60
50625
3.75
3.75
3
```

Built-in Numeric Tools

- Built-in mathematical functions
  - abs(x), pow(x,y)
- math module in python
  - Contains many mathematical functions listed in this link:

    https://docs.python.org/3/library/math.html

```
import math
print(math.pow(5,3))
print(math.sqrt(45))
```

```
125.0
6.708203932499369
```

Boolean Variables

- Store the value True or False

```
a = True
print(type(a))
```

```
<class 'bool'>
```

Integers and Floats as Booleans

- Zero is interpreted as False
- NonZero is interpreted as True

```
zero_int = 0
print(bool(zero_int))

nonzero_int =  5
print(bool(nonzero_int))

zero_float = 0.0
print(bool(zero_float))

nonzero_float = 5.0
print(bool(nonzero_float))
```

```
False
True
False
True
```

Boolean Arithmetic

- *Boolean arithmetic* is the arithmetic of true and false logic
- Boolean Operators
  - and
  - or
  - Not
  - Equal (==)
  - Not Equal (!=)

A	B	not A	not B	A == B	A =! B	A or B
T	F	F	T	F	T	T
F	T	T	F	F	T	T
T	T	F	F	T	F	T
F	F	T	T	T	F	F

Bitwise Operators

- Bitwise operators are used to compare integers in their binary formats.
- When performing a binary operations between 2 integers, there are first converted into binary numbers.

Bitwise Operator	Description
&	Bitwise and
|	Bitwise or
~	Bitwise not
>>	Bitwise shift right
<<	Bitwise shift left

```
A = 5
B = 6
print(A  & B)
print(A  | B)
print(~B)
print(A >>1)
print(A <<1)
```

```
4
7
-7
2
10
```

Comparison Operators

- compare the values of 2 objects and returns True or False
  - >, < , <=, <=, ==, !=

```
A = 5
B = 6
print(A  > B)
```

```
False
```

Operators Precedence

operators	descriptions
(), [], {}, ``	tuple, list, dictionnary, string
x.attr, x[], x[i:j], f()	attribute, index, slide, function call
+x, -x, ~x	unary negation, bitwise invert
**	exponent
*, /, %	multiplication, division, modulo
+, -	addition, substraction
<<, >>	bitwise shifts
&	bitwise and
^	bitwise xor
|	bitwise or
<, <=, >=, >	comparison operators
==, !=, is, is not, in,	comparison operators (continue)
not in	comparison operators (continue)
not	boolean NOT
and	boolean AND
or	boolean OR
lambda	lamnda expression

Strings

- *String:* a series of characters.
  - Anything inside quotes is considered a string in Python
  - You can use single or double quotes around strings
    - "This is a string.“
    - 'This is also a string.'
- Use quotes and apostrophes within your strings
  - 'I told my friend, "Python is my favorite language!"‘
  - "The language 'Python' is named after Monty Python, not the snake.“
  - "One of Python's strengths is its diverse and supportive community."

Strings—escape characters

- \n: new line
- \t: tab
- \\: prints \
- You can ignore escape characters by preceding the string quotes with **r**
  - **Ex:** Sample = r”This is \n a new string”

```
sample_string1 = r"Computer Application Lab
\n 0909331"
sample_string2= "Computer Application Lab \n
0909331"
print(sample_string1)
print(sample_string2)
```

```
Computer Application Lab \n 0909331
Computer Application Lab
 0909331
```

Strings Methods

- A *method* is an action that Python can perform on a piece of data
- title() method
  - The dot (.) after name in name.title() tells Python to make the title() method act on the variable name.

```
lab_name = "computer application lab"
print(lab_name)
print(lab_name.title())
```

```
computer application lab
Computer Application Lab
```

- **How to print on the same line?**

Strings Methods

- title() method
- upper() method
- lower() method

```
lab_name = "computer application lab"
print(lab_name)
print(lab_name.title())
print(lab_name.upper())
print(lab_name.lower())
```

```
computer application lab
Computer Application Lab
COMPUTER APPLICATION LAB
computer application lab
```

Strings Methods

- lower() :Ex: Variable_name.lower()
- upper() :Ex: Variable_name.upper()
- islower() :Checks if the string is in lower case
- isupper():Checks if the string is in upper case

```
sample = "Computer"
print(sample.lower())
print(sample.upper())
print(sample.islower())
print(sample.isupper())
```

```
computer
COMPUTER
False
False
```

String: in, not in, stratswith, endswith

- Check if a certain string appears inside another string
  - Ex: ‘Welcome’ **in** ‘Welcome to the computer applications lab ’
  - Ex: sentence = ‘Welcome to the computer applications lab ’

    ‘Welcome’ in sentence
- Check if certain string **starts** or **ends** with a certain string
  - Ex: sentence = ‘Welcome to the computer applications lab ’

    sentence.startswith(‘Welcome’)

    sentence.endswith(‘Lab’)

```
sample = "Computer Application Lab"
print("computer" in sample)
print("Computer" in sample)
print(sample.startswith("Computer"))
print(sample.startswith("Com"))
print(sample.endswith("Lab"))
```

```
False
True
True
True
True
```

Concatenating Strings

- Python uses the plus symbol (+) to combine strings
- Combining strings is called *concatenation*

```
lab_name = "computer application lab"
lab_no = "0907311"
lab_info = lab_name + " " +lab_no
Print(lab_info.title())
```

```
Computer Application Lab 0907311
```

Strings—split() and join()

- split(): splits a string using specified delimiter
  - Returns a list
- join(): takes a list of strings and join them

String indexing

- In Python, we can also index backward, from the end—positive indexes count from the left, and negative indexes count back from the right
- Indexing is done using square brackets

```
S = "Computer"
print(S)
print(S[0:len(S)])
print(S[0])
print(S[-1])
print(S[1:])
print(S[-3:])
print(S[0:-2])
print(S[:3])
print(S[3:])
```

```
Computer
Computer
C
r
omputer
ter
Comput
Com
puter
```

Strings Immutability

- Strings do not support item assignment

```
S = "Computer"
S[0] = "A"
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    S[0] = "A"
TypeError: 'str' object does not support item assignment
```

Stripping Whitespace

- strip()
- lstrip()
- rstrip()

```
lab_name = " computer application lab "
print(lab_name)
print(lab_name.lstrip())
print(lab_name.rstrip())
print(lab_name.strip())
```

```
 computer application lab
computer application lab
 computer application lab
computer application lab
```

Type Conversion

- str(): Covert a variable to string
- int(): Covert a variable to integer
- float(): Covert a variable to float

```
lab_name = " computer application lab "
x = 20
print( " The number of students in the" + lab_name + "is " + x)
```

```
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print( " The  number of students in the" + lab_name +  "is  " + x)
TypeError: can only concatenate str (not "int") to str
```

Type Conversion

```
lab_name = " computer application lab "
x = 20
print( " The number of students in the" + lab_name + "is " + str(x) )
```

```
The number of students in the computer application lab is 20
```

User input

- input() function is used to get the user input
- The input() function pauses your program and waits for the user to enter **some text**.
- Once Python receives the user's input, it stores it in a variable to make it convenient for you to work with

```
message = input("Enter your first name: ")
print(message)
```

```
Enter your first name: Mohammed
Mohammed
```

User input

```
message = "Welcome to the Computer Application Lab"
message += " \nPlease Enter Your First Name: "
first_name= input(message)
print("Hello " + first_name)
```

```
Welcome to the Computer Application Lab
Please Enter Your First Name: Mohammed
Hello Mohammed
```

Accept Numerical Input

- When you use the input() function, Python interprets everything the user enters as a string.

```
X = input("Enter a Number: ")
X = X +5
```

```
Enter a Number: 5
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    X = X +5
TypeError: can only concatenate str (not "int") to str
```

Accept Numerical Input

```
X = input("Enter a Number: ")
X = int(X) +5
print("X + 5 = " + X)
```

```
Enter a Number: 5
Traceback (most recent call last):
  File "main.py", line 3, in <module>
    print("X + 5 =  " + X)
TypeError: can only concatenate str (not "int") to str
```

Accept Numerical Input

```
X = input("Enter a Number: ")
X = int(X) +5
print("X + 5 = " + str(X))
```

```
Enter a Number: 6
X + 5 = 11
```

Lists, Tuples and Dictionaries

(Chapter 4)

Prepared by Dr.Mohammad Abdel-Majeed, Eng.Abeer Awad and Ayah Alramahi

List in Python

- A *list* is a collection of items in a particular order
  - Numbers, letters, strings etc.
  - mutable
- Defined using square brackets []

```
bicycles =['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
```

List in Python

- List can have elements with different types

```
Mixed_List = ['treck',5, True, 'A']
print(type(Mixed_List[0]))
print(type(Mixed_List[1]))
print(type(Mixed_List[2]))
print(type(Mixed_List[3]))
```

```
<class 'str'>
<class 'int'>
<class 'bool'>
<class 'str'>
```

Accessing elements in the list

- Index Positions Start at 0, Not 1

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])
```

```
trek
```

Accessing elements in the list

- By asking for the item at index -1, Python always returns the last item in the list:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

```
specialized
```

Modifying Elements in a List

- To change an element, use the name of the list followed by the index of the element you want to change, and then provide the new value you want that item to have.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles[0] = "BMX"
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['BMX', 'cannondale', 'redline', 'specialized']
```

Adding Elements to a List– append()

- Add element after the last element in the list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.append("BMX")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'redline', 'specialized', 'BMX']
```

Adding Elements to a List– insert()

- Use insert() method
  - The element will be inserted before the item located at the specified index.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.insert(2,"BMX")
print(bicycles)

bicycles.insert(-1,"Honda")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'BMX', 'redline', 'specialized']
['trek', 'cannondale', 'BMX', 'redline', 'Honda', 'specialized']
```

Removing Elements from a List

- You can remove an item according to its **position** in the list or according to its **value**

- **del** statement can be used to remove an element from the list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

del bicycles[1]
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'redline', 'specialized']
```

Removing Elements from a List—pop()

- **pop()** method removes and return the last element in the list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

element = bicycles.pop()
print(bicycles)
print(element)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'redline']
specialized
```

Removing Elements from a List—pop(index)

- **pop(index)** method removes and return the the element at the given index

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

element = bicycles.pop(2)
print(bicycles)
print(element)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'cannondale', 'specialized']
redline
```

Removing Elements from a List by Value

- remove(Value) method can be used to remove an element from a list

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.remove("cannondale")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['trek', 'redline', 'specialized']
```

Removing Elements from a List by Value

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)

bicycles.remove("trek")
print(bicycles)
```

```
['trek', 'cannondale', 'redline', 'specialized']
['cannondale', 'redline', 'specialized']
```

Removing Elements from a List by Value

- The remove() method deletes only the first occurrence of the value you specify

```
bicycles = ['trek', 'cannondale', `trek', 'specialized']
print(bicycles)

A = "trek"
bicycles.remove(A)
print(bicycles)
```

```
['trek', 'cannondale', 'trek', 'specialized']
['cannondale', 'trek', 'specialized']
```

Organizing a List-Sorting

- sort() method
  - Permanent sort

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

bicycles.sort()

print(bicycles)
```

```
['trek', 'cannondale', 'bmx', 'specialized']
['bmx', 'cannondale', 'specialized', 'trek']
```

Organizing a List-Sorting

- sort() method

```
bicycles = ['trek', 'cannondale', 2, 'specialized']
print(bicycles)

bicycles.sort()

print(bicycles)
```

```
['trek', 'cannondale', 2, 'specialized']
Traceback (most recent call last):
  File "main.py", line 4, in <module>
    bicycles.sort()
TypeError: '<' not supported between
instances of 'int' and 'str'
```

Organizing a List-Sorting

- sorted() function

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

s_bicycles = sorted(bicycles)
print(bicycles)
print(s_bicycles)
```

```
['trek', 'cannondale', 'bmx', 'specialized']
['trek', 'cannondale', 'bmx', 'specialized']
['bmx', 'cannondale', 'specialized', 'trek']
```

Printing the list in the reversed order

- reverse() method

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

s_bicycles = bicycles.reverse()
print(bicycles)
print(s_bicycles)
```

```
['trek', 'cannondale', 'bmx', 'specialized']
['specialized', 'bmx', 'cannondale', 'trek']
None
```

Length of a List

- Len() function returns the length of the list.

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']
print(bicycles)

print(len(bicycles))
```

```
4
```

Operation	Interpretation
`L * 3`	
`for x in L: print(x)`	Iteration, membership
`3 in L`	
`L.append(4)`	Methods: growing
`L.extend([5,6,7])`	
`L.insert(i, X)`	
`L.index(X)`	Methods: searching
`L.count(X)`	
`L.sort()`	Methods: sorting, reversing,
`L.reverse()`	copying (3.3+), clearing (3.3+)
`L.copy()`	
`L.clear()`	
`L.pop(i)`	Methods, statements: shrinking
`L.remove(X)`	
`del L[i]`	
`del L[i:j]`	
`L[i:j] = []`	
`L[i] = 3`	Index assignment, slice assignment
`L[i:j] = [4,5,6]`	
`L = [x**2 for x in range(5)]`	List comprehensions and maps (Chapter 4, Chapter 14, Chapter 20)
`list(map(ord, 'spam'))`	

Looping through the list

- *for loop*: pull an item from the bicycles list and place it in the variable bicycle
  - Indentation is important
  - We will cover looping and control structures later

```
bicycles = ['trek', 'cannondale', 'bmx', 'specialized']

for bicycle in bicycles:
    print(bicycle)
```

```
trek
cannondale
bmx
specialized
```

range() function

- range() function makes it easy to generate a series of numbers.
- To print the numbers from 1 to 4, you would use range(1,5)

```
for value in range(1,5):
    print(value)
```

```
1
2
3
4
```

list() and range() functions

- You can convert the results of range() directly into a list using the list() function

```
numbers = list(range(1,5))
print(numbers)
```

```
[1, 2, 3, 4]
```

range() with fixed interval

- The step between the numbers in the list can be changed
  - By default the step is 1
  - range(start, end, step)

```
even_numbers = list(range(2,11,2))
print(even_numbers)
```

```
[2, 4, 6, 8, 10]
```

Statistics with list of numbers

```
digits = [1,2,3,4,5,6,7,8,9,0]

print(min(digits))

print(max(digits))

print(sum(digits))
```

```
0
9
45
```

List Comprehensions

- Shortcut to create a list of numbers

```
squares = [value**2 for value in range(1,11)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Slicing a list

- To make a slice, you specify the index of the **first** and **last** elements you want to work with
  - List[first:last]→Last not included
  - fist is empty→from the beginning
  - Last is empty→till the end of the list

```
players = ['charles', 'martina', 'michel', 'florence', 'eli']
print(players[0:3])
print(players[0:])
print(players[:3])
print(players[:])
print(players[-3:])
```

```
['charles', 'martina', 'michel']
['charles', 'martina', 'michel', 'florence', 'eli']
['charles', 'martina', 'michel']
['charles', 'martina', 'michel', 'florence', 'eli']
['michel', 'florence', 'eli']
```

Looping Through a Slice

```
players = ['charles', 'martina', 'michel', 'florence', 'eli']
print('Here are the first three players on my team: ')
for player in players[:3]:
    print(player.title())
```

```
Here are the first three players on my team:
Charles
Martina
Michel
```

Copying a list

- Using assignment operator or copy() method do not create a separate copy

```
players = ['charles', 'martina', 'michel', 'florence', 'eli']
players_copy = players

print(players)
print(players_copy)

players[0] = "Mike"

print(players)
print(players_copy)
```

```
['charles', 'martina', 'michel', 'florence', 'eli']
['charles', 'martina', 'michel', 'florence', 'eli']
['Mike', 'martina', 'michel', 'florence', 'eli']
['Mike', 'martina', 'michel', 'florence', 'eli']
```

Copying a list

- Slicing can be used to create a separate copy

```
players = ['charles', 'martina', 'michel', 'florence', 'eli']
players_copy = players[:]

print(players)
print(players_copy)

players[0] = "Mike"

print(players)
print(players_copy)
```

```
['charles', 'martina', 'michel', 'florence', 'eli']
['charles', 'martina', 'michel', 'florence', 'eli']
['Mike', 'martina', 'michel', 'florence', 'eli']
['charles', 'martina', 'michel', 'florence', 'eli']
```

Multi-Dimensional Lists

- Multi-dimensional lists are the lists within lists.

```
a = [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]
for record in a:
    print(record)
```

```
[2, 4, 6, 8, 10]
[3, 6, 9, 12, 15]
[4, 8, 12, 16, 20]
```

```
a = [[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [4, 8, 12, 16, 20]]
for record in a:
print(record)
```

```
[[2, 4, 6, 8, 10], [3, 6, 9, 12, 15], [20, 16, 12, 8, 4]]
```

Tuples

- Tuple is an immutable list.
  - Its values cannot be changed

```
dimensions = (200,50)
print(dimensions[0])
print(dimensions[1])
```

```
200
50
```

Tuples

```
dimensions = (200,50)
dimensions[0] = 30
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    dimensions[0] = 30
TypeError: 'tuple' object does not support item assignment
```

Looping Through All Values in a Tuple

- Similar to the lists

```
dimensions = (200,50)
for dimension in dimensions:
    print(dimension)
```

```
200
50
```

```
dimensions = (200,50)
print(dimensions)

dimensions = (400,500)
print(dimensions)
```

```
(200, 50)
(400, 500)
```

Dictionaries

- A dictionary in Python is a collection of key-value pairs
- You can use a key to access the value associated with that key
- A key’s value can be a number, a string, a list, or even another dictionary
- A dictionary is wrapped in braces, {}
- Every key is connected to its value by a colon
- Individual key-value pairs are separated by commas

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
```

Dictionaries

- storing different kinds of information about one object

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
```

- You can also use a dictionary to store one kind of information about many objects

```
Age = {'Mohammad': 22, 'Ahmad': 40, 'Ayman': 30}
```

Accessing Values in a Dictionary

- To get the value associated with a key, give the name of the dictionary and then place the key inside a set of square brackets

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
print(student['name'])
```

```
Mohammad
```

Adding New Key-Value Pair

- Would give the name of the dictionary followed by the new key in square brackets along with the new value.

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
print(student['Age'])

student['city'] = 'Amman'
print(student)
```

```
22
{'city': 'Amman', 'name': 'Mohammad', 'Gender': 'M', 'Age': '22'}
```

```
student ={}
print(student)

student['Age'] = 22
student['name'] = 'Mohammad'
student['Gender'] = 'M'
student['city'] = 'Amman'
print(student)
```

```
{}
{'Age': 22, 'name': 'Mohammad', 'Gender': 'M', 'city': 'Amman'}
```

Modifying Values in a Dictionary

- give the name of the dictionary with the key in square brackets and then the new value you want associated with that key.

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

student['Age'] = 25
print(student)
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
{'name': 'Mohammad', 'Gender': 'M', 'Age': 25}
```

Removing Key-Value Pairs

- Use the del statement to completely remove a key-value pair.
- All del needs is the name of the dictionary and the key that you want to remove.

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

del student['Gender']
print(student)
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
{'name': 'Mohammad', 'Age': 22}
```

Removing Key-Value Pairs—pop()

- Use pop() method to remove key from a dictionary.
- pop() returns the value of the corresponding key()

```
my_dict = {'C1': 30, 'C2':25, 'C3':33}

my_dict.pop('C1')
print(my_dict)
```

```
{'C2': 25, 'C3': 33}
```

Looping Through a Dictionary- Key-Value Pairs

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

for key,value in student.items():
    print('the key is: ' + key +" and its value is: "+ str(value))
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the key is: name and its value is: Mohammad
the key is: Gender and its value is: M
the key is: Age and its value is: 22
```

Looping Through a Dictionary- Keys

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

for key in student.keys():
    print('the key is: ' + key )
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the key is: name
the key is: Gender
the key is: Age
```

Looping Through a Dictionary- Sorted Keys

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)
for key in sorted(student.keys()):
   print('the key is: ' + key )
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the key is: Age
the key is: Gender
the key is: name
```

Looping Through a Dictionary- Values

```
student = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
print(student)

for value in student.values():
    print('the value is: ' + str(value) )
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
the value is: Mohammad
the value is: M
the value is: 22
```

List of Dictionaries

```
student_1 = {'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
student_2 = {'name': 'Ahmad', 'Gender': 'M', 'Age': 25}
student_3 = {'name': 'Lina', 'Gender': 'F', 'Age': 18}
students = [student_1, student_2, student_3]

for student in students:
    print(student)

print(students[0]['name'])
```

```
{'name': 'Mohammad', 'Gender': 'M', 'Age': 22}
{'name': 'Ahmad', 'Gender': 'M', 'Age': 25}
{'name': 'Lina', 'Gender': 'F', 'Age': 18}
Mohammad
```

List in a Dictionary

```
pizza = {
'crust': 'thick',
'toppings':['mushrooms','extra cheese']}

print('You ordered a '+ pizza['crust']+ '-crust pizza '+ 'with
the following toppings')
for topping in pizza['toppings']:
    print("\t" + topping)
```

```
You ordered a thick-crust pizza with the following toppings
    mushrooms
    extra cheese
```

List in Dictionaries

```
favorite_language = {
'jen':['python','rupy'],
'sara': ['c'],
'edward':['rupy','go'],
}
for name, languages in favorite_language.items():
    print("\n" + name.title() + "'s favorite language are:")
    for language in languages:
        print("\t" + language.title())
```

```
Jen's favorite language are:
    Python
    Rupy

Sara's favorite language are:
    C

Edward's favorite language are:
    Rupy
    Go
```

Dictionary in a Dictionary

```
users = {
'aeinstein':{
'first': 'albert',
'last': 'einstein',
'location': 'princeton',
},
'mcurie':{
'first': 'marie',
'last': 'curie',
'location': 'paris',
},
}
for username, user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info['first']+ " "
    +user_info['last']
    location = user_info['location']

    print("\tFull name: "+
full_name.title())
    print("\tLocation: " + location.title())
```

```
Username: aeinstein
    Full name: Albert Einstein
    Location: Princeton

Username: mcurie
    Full name: Marie Curie
    Location: Paris
```

Concatenate Dictionaries--Update

- The update() method updates the dictionary with the elements from the another dictionary object or from an iterable of key/value pairs.

```
d = {1: "one", 2: "three"}
d1 = {2: "two"}

#update the value of key 2
d.update(d1)
print(d)

d1={3: "three"}
# adds element with key 3
d.update(d1)
print(d)

d= {'x':2}
d.update(y = 3, z = 0)
print(d)
```

```
{1: 'one', 2: 'two'}
{1: 'one', 2: 'two', 3: 'three'}
{'x': 2, 'y': 3, 'z': 0}
```

Control Flow and Error Exception
(CH5 and CH7)

Prepared by Dr.Mohammad Abdel-Majeed, Eng.Abeer Awad and Ayah Alramahi

Outline

- Conditional statements: *if*, *elif*, and *else*.
- *for* loops.
- *while* loops.
- Errors and exceptions.

Conditional statements: *if*, *elif*, and *else*.

- The **simplest** kind of if statement has one test and one action

if *conditional_test:*
    do something

Notes:

1. You can put any conditional test in the first line and just about any action in the indented block following the test. If the conditional test evaluates to *True*, Python executes the code following the *if* statement. If the test evaluates to *False*, Python ignores the code following the *if* statement.

```
age = 19
if age >= 18:
    print("you are old enough to vote")
```

```
you are old enough to vote
```

2. The print statement is supposed to be indented (one tab or four spaces). indented lines will be ignored if the test does not pass.

```
age = 17
if age >= 18:
    print("you are old enough to vote")
    print("Have you registered to vote yet?")


print("This statement will be always executed nevertheless the result of if statement")
```

```
This statement will be always executed nevertheless the result of if statement
```

❖*if-else Statements*

if *conditional_test:*

do something

else:

do something else

```
age = 17
if age >= 18:
    print("you are old enough to vote")
    print("Have you registered to vote yet?")
else:
    print("Sorry, you are too young to vote.")
    print("Please register to vote as soon as you turn 18!")
```

```
Sorry, you are too young to vote.
Please register to vote as soon as you turn 18!
```

❖*The if-elif-else Chain*

if *conditional_test 1:*
    do something
elif conditional_test 2:
    do something else
else:
    do another thing

```
age = 12
if age <= 4:
    price = 0
elif age < 18:
    price = 5
else:
    price = 10
print("Your admission cost is :$" + str(price) + ".")
```

```
Your admission cost is :$5.
```

Note: you can use multiple *elif* statements and you can omit the *else* block

❖Testing Multiple Conditions

- As soon as Python finds one test that passes, it skips the rest of the tests. This behavior is beneficial, because it's efficient and allows you to test for one specific condition. sometimes it's important to check all of the conditions of interest. In this case, you should use a series of simple independent *if* statements with no *elif* or *else* blocks.

```
requested_toppings = ['mushrooms', 'extra cheese']
if 'mushrooms' in requested_toppings:
    print("Adding mushrooms.")
if 'pepperoni' in requested_toppings:
    print("Adding pepperoni.")
if 'extra cheese' in requested_toppings:
    print("Adding extra cheese.")

print("\nFinished making your pizza!")
```

```
Adding mushrooms.
Adding extra cheese.

Finished making your pizza!
```

- We can use if statement to check if a list is not empty

```
requested_toppings =[]

if requested_toppings:
    print("some topping are requested")
else:
    print("you request nothing")
```

```
you request nothing
```

for loops

- Loops in Python are a way to repeatedly execute some code statement. We specify the variable we want to use (N), the sequence we want to loop over (iterator), and use the *in* operator to link them in an intuitive and readable way.

for *N* in *iterator*:
    do something

```
for N in [2, 3, 5, 6]:
    print(N, end=" ")
```

```
2 3 5 6
```

- One of the most commonly used iterator in Python is the *range* object.

  for N in range(10): # N=0,1,2,…,9

  for N in range(4,30,20): #N=4,6,8,…,28

- If the iterator is a list, N will be the contents of the list.

```
students = ['Ali', 'Ahmad', 'Yazan']
for N in students:
    print(N)
```

```
Ali
Ahmad
Yazan
```

Note: keep in mind when writing your own for loops that you can choose any name you want for the temporary variable that holds each value in the list. However, it's helpful to choose a meaningful name that represents a single item from the list.

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print(student)
```

- Avoid Indentation Errors

-Forgetting to Indent

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
print(student)
```

```
 print(student)
 ^
IndentationError: expected an
indented block
```

-Forgetting to Indent Additional Lines

(logical error)

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print("hello " ,student)
print("It's nice to meet you ", student)
```

```
hello  Ali
hello  Ahmad
hello  Yazan
It's nice to meet you  Yazan
```

- Indenting Unnecessarily

```
students = ['Ali', 'Ahmad', 'Yazan']
   print("hello world!")
```

```
 print("hello world!")
 ^
IndentationError: unexpected indent
```

- Indenting Unnecessarily after the *for* loop

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print("hello ", student)
    print('You are the best three
students\n')
```

```
hello  Ali
You are the best three students

hello  Ahmad
You are the best three students

hello  Yazan
You are the best three students
```

```
students = ['Ali', 'Ahmad', 'Yazan']
for student in students:
    print("hello ", student)
print('You are the best three
students\n')
```

```
hello  Ali
hello  Ahmad
hello  Yazan
You are the best three students
```

- *For* loops with dictionaries

```
student1 = {'name':'Ali','grade':'95'}
for student in student1:
    print( student)
```

```
name
grade
```

- *student* is the keys in the dictionary

```
student1 = {'name':'Ali','grade':'95'}
student2 = {'name':'Ahmad','grade':'84'}
student3 = {'name':'Yazan','grade':'98'}
students = [student1, student2, student3]
for student in students:
    print( student)
```

```
{'name': 'Ali', 'grade': '95'}
{'name': 'Ahmad', 'grade': '84'}
{'name': 'Yazan', 'grade': '98'}
```

- *student* is the elements in the list

```
student1 = {'name':'Ali','grade':'95'}
for key, value in student1.items():
    print( key, value)
```

```
name Ali
grade 95
```

- Use *.items()* to retrieve the (key, *value*) pairs.

```
student1 = {'name': 'Ali', 'grade':['95', '83']}
for N in student1['grade']:
    print(N)
```

```
95
83
```

- Specify which key to take data from.

while loops.

❖Introducing while Loops

- The *for* loop takes a collection of items and executes a block of code once for each item in the collection. In contrast, the while loop runs as long as, or *while*, a certain condition is true.

```
current_number = 1
while current_number <= 5:
    print(current_number)
    current_number += 1
```

```
1
2
3
4
5
```

Notes:

- *The used variable must be defined before the loop.*
- *The loop keeps testing if the boolean condition is true, it keeps executing the loop statements.*
- *The used variable must be changed during the loop, or infinite loop will occur*

- Letting the User Choose When to Quit

```
prompt = "\nTell me something, and I will repeat it back to
you:"
prompt += "\nEnter 'quit' to end the program. "
message = ""
while message != 'quit':
    message = input(prompt)
    print(message)
```

```
Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. hello
hello

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. welcome to python
welcome to python

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. quit
quit
```

- Using a Flag

```
prompt = "\nTell me something, and I will repeat it back to you:"
prompt += "\nEnter 'quit' to end the program. "
active = True
while active:
    message = input(prompt)

    if message == 'quit':
        active = False
    else:
        print(message)
```

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. *hello*
hello

Tell me something, and I will repeat it back to you:
Enter 'quit' to end the program. *quit*

- Using break to Exit a Loop

```
prompt = "\nPlease enter the name of a city you have visited:"
prompt += "\n(Enter 'quit' when you are finished.) "
while True:
    city = input(prompt)

    if city == 'quit':
        break
    else:
        print("I'd love to go to " + city.title() + "!")
```

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) *amman*
I'd love to go to Amman!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) *jerusalem*
I'd love to go to Jerusalem!

Please enter the name of a city you have visited:
(Enter 'quit' when you are finished.) *quit*

- Using continue in a Loop

```
current_number = 0
while current_number < 10:
    current_number += 1
    if current_number % 2 == 0:
        continue

    print(current_number)
```

```
1
3
5
7
9
```

❖Using a while Loop with Lists and Dictionaries

- Moving Items from One List to Another

```
unconfirmed_users = ['alice', 'brian', 'candace']
confirmed_users = []

while unconfirmed_users:
    current_user = unconfirmed_users.pop()

    print("Verifying user: " + current_user.title())
    confirmed_users.append(current_user)
print("\nThe following users have been confirmed:")
for confirmed_user in confirmed_users:
    print(confirmed_user.title())
```

```
Verifying user: Candace
Verifying user: Brian
Verifying user: Alice

The following users have been confirmed:
Candace
Brian
Alice
```

- Removing All Instances of Specific Values from a List

```
pets = ['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']

print(pets)

while 'cat' in pets:
    pets.remove('cat')

print(pets)
```

```
['dog', 'cat', 'dog', 'goldfish', 'cat', 'rabbit', 'cat']
['dog', 'dog', 'goldfish', 'rabbit']
```

- Filling a Dictionary with User Input

```
responses = {}
# Set a flag to indicate that polling is active.
polling_active = True
while polling_active:
    # Prompt for the person's name and response.
    name = input("\nWhat is your name? ")
    response = input("Which mountain would you like to
climb   someday? ")

    # Store the response in the dictionary:
    responses[name] = response

    # Find out if anyone else is going to take the poll.
    repeat = input("Would you like to let another person
respond? (yes/ no) ")
    if repeat == 'no':
        polling_active = False

# Polling is complete. Show the results.
print("\n--- Poll Results ---")
for name, response in responses.items():
    print(name + " would like to climb " + response + ".")
```

```
What is your name? Ali
Which mountain would you like to climb
someday? Everest
Would you like to let another person
respond? (yes/ no) yes

What is your name? Yazan
Which mountain would you like to climb
someday? Elbrus
Would you like to let another person
respond? (yes/ no) no

--- Poll Results ---
Ali would like to climb Everest.
Yazan would like to climb Elbrus.
```

Errors and exceptions.

❖Errors

No matter your skill as a programmer, you will eventually make a coding mistake. Such mistakes come in three basic flavors:

1. *Syntax errors*

   Errors where the code is not valid Python (generally easy to fix)

2. *Runtime errors*

   Errors where syntactically valid code fails to execute, perhaps due to invalid user input (sometimes easy to fix)

3. *Semantic errors*

   Errors in logic: code executes without a problem, but the result

   is not what you expect (often very difficult to identify and fix)

Run time errors

```
print(Q)
```

```
Traceback (most recent call last):
  File "main.py", line 1, in
<module>
    print(Q)
NameError: name 'Q' is not defined
```

```
print(1 + 'abc')
```

```
Traceback (most recent call last):
  File "main.py", line 1, in <module>
    print(1 + 'abc')
TypeError: unsupported operand type(s)
for +: 'int' and 'str'
```

```
print(3/0)
```

```
Traceback (most recent call last):
  File "main.py", line 1, in
<module>
    print(3/0)
ZeroDivisionError: division by zero
```

```
L= [1, 2, 3]
print(L[100])
```

```
Traceback (most recent call last):
  File "main.py", line 2, in <module>
    print(L[100])
IndexError: list index out of range
```

❖Catching Exceptions: try and except

- The main tool Python gives you for handling runtime exceptions is the *try…except* clause. Its basic structure is this:

```
try:
        print("this gets executed first")
except:
        print("this gets executed only if there is an error")
```

```
x = int(input('enter X: '))
y = int(input('enter Y: '))
try:
   print(x/y)
except:
   print('something bad happened!')
```

```
enter X: 10
enter Y: 5
2.0
```

```
enter X: 10
enter Y: 0
something bad happened!
```

- *try…except…else…finally*

```
try:
        print("try something here")
except:
        print("this happens only if it fails")
else:
        print("this happens only if it succeeds")
finally:
        print("this happens no matter what")
```

```
x = int(input('enter X: '))
y = int(input('enter Y: '))
try:
   Z = x/y
except:
   print('something bad happened!')
else:
   print(Z)
finally:
   print('be careful with dividing numbers')
```

```
enter X: 10
enter Y: 5
2.0
be careful with dividing numbers
```

```
enter X: 10
enter Y: 0
something bad happened!
be careful with dividing numbers
```

Functions & Files

Prepared by Dr.Mohammad Abdel-Majeed, Eng.Abeer Awad and Ayah Alramahi

Defining a function

- **Definition**

  def function_name(function parameters):

  function_definition

- **Function Call**

  function_name(arguments)

Defining a function-Example

```
def greet_user():
    """Display a simple greeting."""
    print("Hello!")

greet_user()
```

```
Hello!
```

Function with Parameters

```
def greet_user(name):
    """Display a simple greeting."""
    print("\nHello " + name + "!" )

greet_user("Ahmad" )
```

```
Hello Ahmad!
```

Passing Arguments

- Python must match each argument in the function call with a parameter in the function definition
- **Positional Arguments**: Pairing the arguments with parameters based on the order of the arguments provided
  - Order Matters

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")

describe_pet('hamster', 'harry')
```

```
I have a hamster.
My hamster's name is Harry.
```

Passing Arguments

- Keyword Arguments: pass the **name-value** pair to the function

```
def describe_pet(animal_type, pet_name):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")

#Positional Argument
describe_pet('hamster', 'harry')

#Keyword Arguments
describe_pet(animal_type='hamster', pet_name= 'harry')
```

```
I have a hamster.
My hamster's name is Harry.

I have a hamster.
My hamster's name is Harry.
```

Parameter's Default Values

- If an argument for a parameter is provided in the function call, Python uses the argument value. If not, it uses the parameter's default value
  - Default paramters should be listed at the end of the parameters list

```
def describe_pet(pet_name = "harry", animal_type ="dog" ):
    """Display information about a pet."""
    print("\nI have a " + animal_type + ".")
    print("My " + animal_type + "'s name is " +
pet_name.title()+".")

#Positional Argument
describe_pet( 'harry')

#Keyword Arguments
describe_pet(pet_name= 'harry')
```

```
I have a dog.
My dog's name is Harry.

I have a dog.
My dog's name is Harry.
```

Return Values

- The value the function returns is called a *return value*
- The *return* statement takes a value from inside a function and sends it back to the line that called the function
- Returned value can be a list, dictionarly, Boolean, string etc.

```
def get_fullname(first_name, last_name):
    """Return a full name, nestly formatted."""
    full_name = first_name + ' ' + last_name
    return full_name.title()

FullName = get_fullname('Mohammad', 'Ahmad')
print(FullName)
```

```
Mohammad Ahmad
```

Passing a List

```
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = "Hello, " + name.title() + "!"
        print(msg)

username = ['hannah', 'ty', 'margot']
greet_users(username)
```

```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

Modifying a List in a Function

- When you pass a list to a function, the function can modify the list.
- Any changes made to the list inside the function's body are permanent

```
def add_user(names, new_user):
    """Print a simple greeting to each user in the list."""
    names.append(new_user)

usernames = ['hannah', 'ty', 'margot']
add_user(usernames, 'Tylor')

print(usernames)
```

```
['hannah', 'ty', 'margot', 'Tylor']
```

Preventing a Function from Modifying a List

- Pass to the function a copy of the list
  - Use Slicing

```
def users(names):
    """Print a simple greeting to each user in the list."""
    names[0] = 'Tylor'

usernames = ['hannah', 'ty', 'margot']
users(usernames[:])

print(usernames)
```

```
['hannah', 'ty', 'margot']
```

Passing Arbitrary Number of Arguments

- Python allows a function to collect an arbitrary number of arguments from the calling statement.

```
def make_pizza(*toppings):
  """Print the list of toppings that have been requested."""
  print(toppings)

make_pizza('pepperoni')
make_pizza('mushrooms', 'green peppers', 'extra cheese')
```

```
('pepperoni',)
('mushrooms', 'green peppers', 'extra cheese')
```

Mixed Arguments

```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) +
    "-inch pizza with the following toppings:")
    for topping in toppings:
       print("- " + topping)

make_pizza(16, 'pepperoni')
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

```
Making a 16-inch pizza with the following toppings:
- pepperoni

Making a 12-inch pizza with the following toppings:
- mushrooms
- green peppers
- extra cheese
```

Using Arbitrary Keyword Arguments

```
def build_profile(first, last, **user_info):
    """Build a dictionary containing everything we know
about a user."""
    profile = {}
    profile['first_name'] = first
    profile['last_name'] = last
    for key, value in user_info.items():
        profile[key] = value
    return profile
user_profile = build_profile('albert', 'einstein',
 location='princeton', field='physics')
print(user_profile)
```

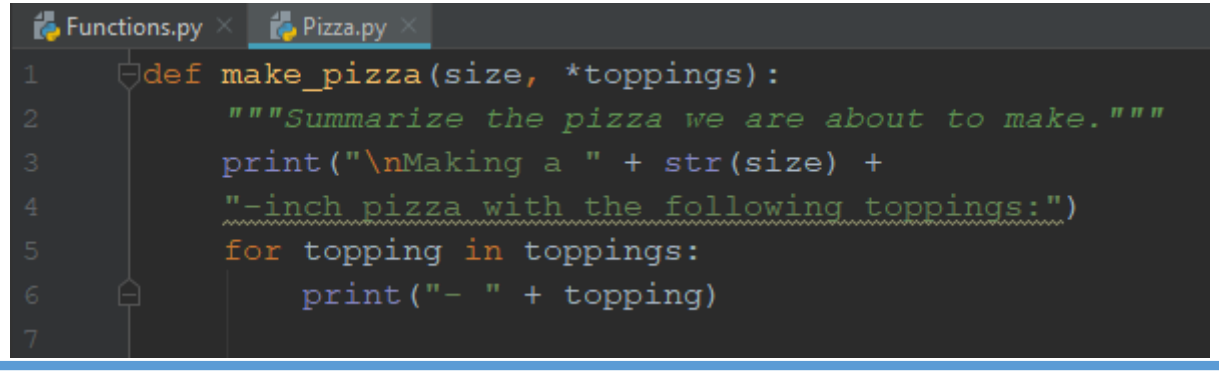
```
{'first_name': 'albert', 'last_name': 'einstein', 'location':
'princeton', 'field': 'physics'}
```

Storing Your Function in Modules

- storing your functions in a separate file called a *module* and then *importing* that module into your main program.
- An *import* statement tells Python to make the code in a module available in the currently running program file

Importing an Entire Module

- A *module* is a file ending in *.py* that contains the code you want to import into your Functions program.
- We created a file called *Pizza.py* and placed the function *make_pizza()* inside it.




```
def make_pizza(size, *toppings):
    """Summarize the pizza we are about to make."""
    print("\nMaking a " + str(size) + "-inch pizza with the
following toppings:")
    for topping in toppings:
       print("- " + topping)
```

Importing an Entire Module

- A *module* is a file ending in *.py* that contains the code you want to import into your Functions program.
- We created a file called *Pizza.py* and placed the function *make_pizza()* inside it.
- To be able to use this function we *imported* the *Pizza.py* module
- We call the function using the ***module_name.function_name()***

Functions.py | Pizza.py

```
1  import Pizza
2  Pizza.make_pizza(12,'mashrooms')
3
```

```
Making a 12-inch pizza with the following toppings:
- mashrooms

Process finished with exit code 0
```

```
import Pizza
Pizza.make_pizza(12,"mashroom")
```

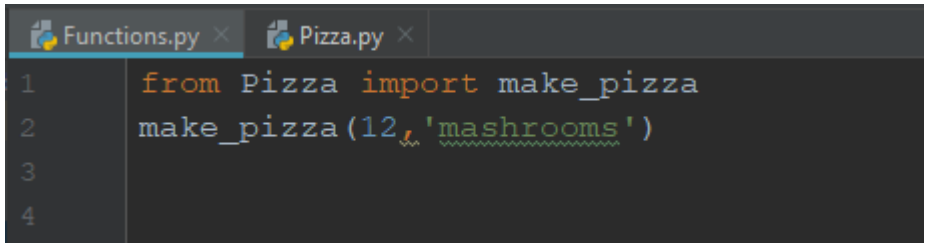
```
Making a 12-inch pizza with the following toppings:
- mashroom
```

Importing Specific Function

- You can also import a specifc function from a module not the entire module

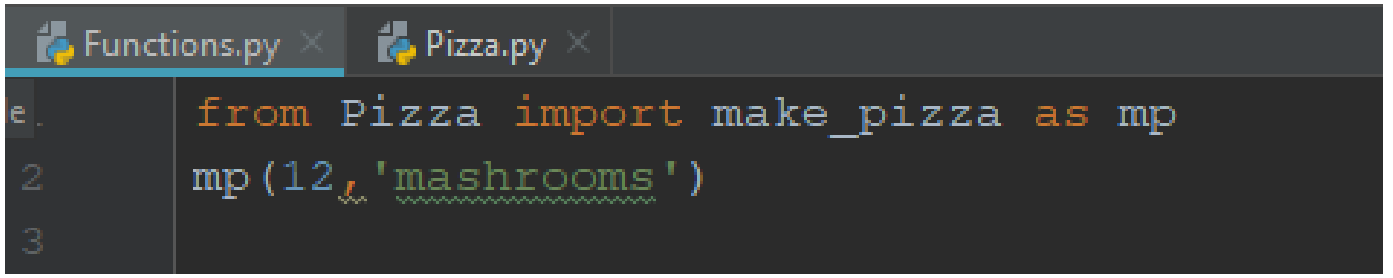
from module_name import function_nam

from module_name import function_0, function_1, function_2



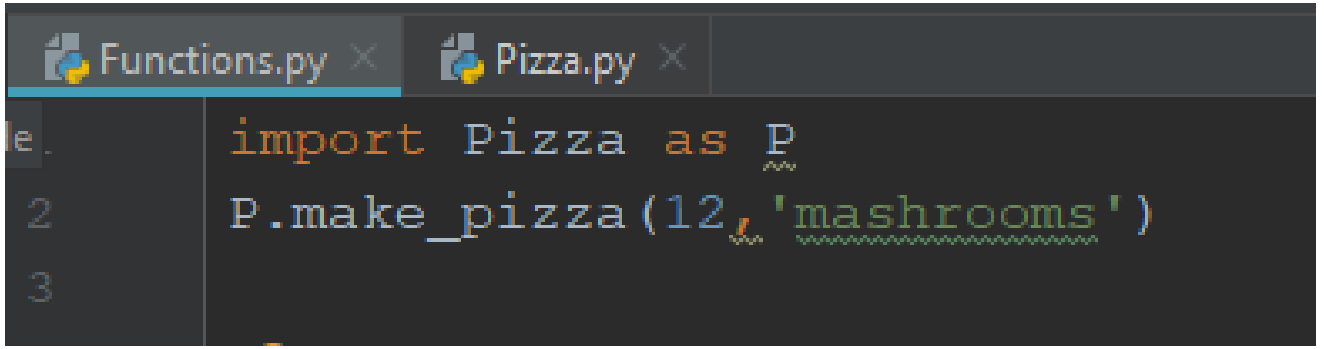

```
from Pizza import make_pizza
make_pizza(12,"mashroom")
```

Using “as” to Give a Function an Alias



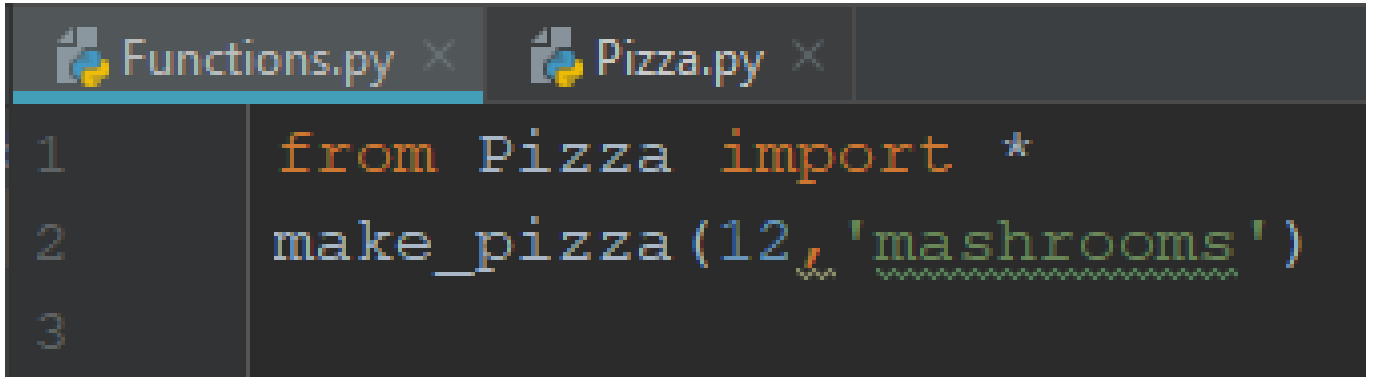

```
from Pizza import make_pizza as mp
mp(12,"mashroom")
```

Using “as” to Give a Module an Alias




```
import Pizza as P
P.make_pizza(12,"mashroom")
```

Importing All Functions in a Module




```
from Pizza import *
make_pizza(12,"mashroom")
```

Lambda function

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.

lambda arguments : expression

```
x = lambda a : a*3
print(x(5))
```

```
15
```

Lambda function-Example

- A lambda function that multiplies argument a with argument b and print the result:

```
x = lambda a, b: a * b
print(x(5,4))
```

```
20
```

Files

Reading from a File

- When you want to work with the information in a text file, the first step is to read the file into memory. You can read the entire contents of a file, or you can work through the file one line at a time.
- You should open the file before accessing it.
- open(filename) function looks for the file in the current directory
- close() function closes the file: improperly closed files can cause **data loss**

Files.py | data.txt

```
1  file_obj = open('data.txt')
2  print(file_obj.read())
3  file_obj.close()
4
```

```
Random Data

Process finished with exit code 0
```

```
file_obj = open('data.txt')
print(file_obj.read())
file_obj.close()
```

```
Random data
```

Reading from a File—with Keyword

- *with* keyword closes the file once access to it is no longer needed

Files.py | data.txt

```
1  with open('data.txt') as file_object:
2      contents = file_object.read()
3      print(contents)
4
```

```
Random Data

Process finished with exit code 0
```

```
with open('data.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

```
Random data
```

File Path

- If the file is not in the same folder of the currently running python program then you need to provide the file path (use forward slash in windows if backslash does not work)

Files.py | data.txt

```
with open('C:/Users/mohammad/data.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

```
Random Data

Process finished with exit code 0
```

Files.py | data.txt

```
filepath = "C:/Users/mohammad/data.txt"
with open(filepath) as file_object:
    contents = file_object.read()
    print(contents)
```

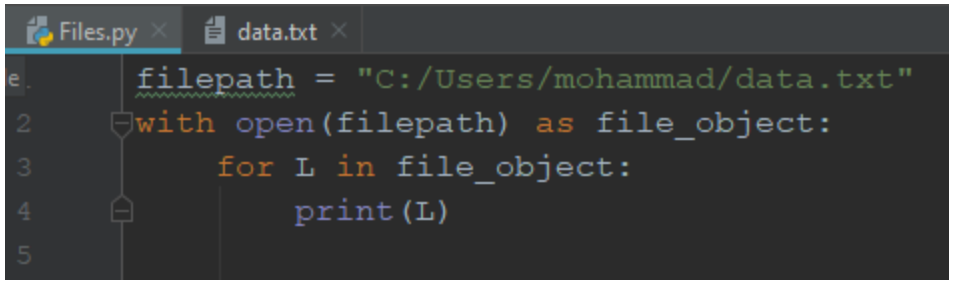
```
with open('C:/Users/mohammad/data.txt')
as file_object:
    contents = file_object.read()
    print(contents)
```

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    print(contents)
```

```
Random data
```

Reading Line by Line

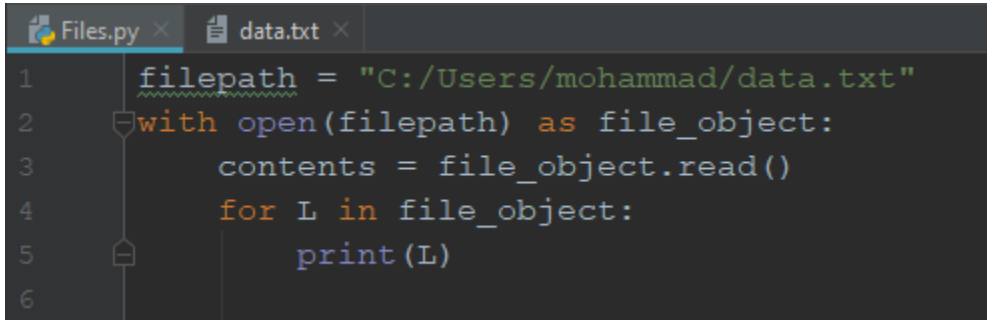
- You can use a for loop on the ***file object*** to examine each line from a file one at a time



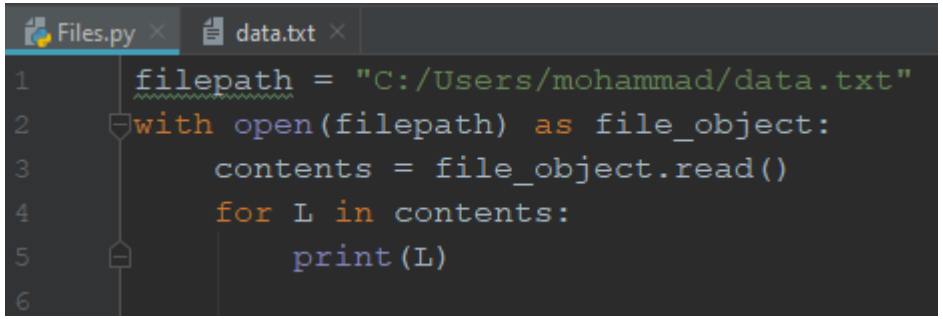

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    for L in file_object:
        print(L)
```

Reading Line by Line

- What is the output for each code?




```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    for L in file_object:
        print(L)
```




```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.read()
    for L in contents:
        print(L)
```

Making a List of Lines from a File

- If you want to retain access to a file's contents outside the *with* block, you can store the file's lines in a list inside the block and then work with that list

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.readlines()

for L in contents:
    print(L)
```

```
Random Data Line0
Random Data Line1
Random Data Line2
Random Data Line3
Random Data Line4
```

Strip Functions

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
    contents = file_object.readlines()

for L in contents:
    print(L.strip())
```

```
Random Data Line0
Random Data Line1
Random Data Line2
Random Data Line3
Random Data Line4
```

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
      contents = file_object.readlines()

One_Line= ' '
for L in contents:
    One_Line +=  '    ' + L.strip()

print(One_Line)
```

```
Random Data Line0 Random Data Line1 Random Data Line2 Random Data Line3 Random Data Line4
```

Writing to a File

- To write text to a file, you need to call open() with a second argument telling Python that you want to write to the file.
- Set the second argument of the open() function to ‘w’ to open the file in the write mode
- ‘a’ → Opens the file in append mode
- ‘r’ → Opens the file in read mode

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath, 'w ') as file_object:
    file_object.write('Random Data Line 5 ')

with open(filepath) as file_object:
    print(file_object.read())
```

```
Random Data Line 5
```

Writing to a File

- The open() function automatically creates the file you're writing to if it doesn't already exist. However, be careful opening a file in write mode ('w') **because if the file does exist, Python will erase the file before returning the file object.**

```
filepath = 'C:/Users/mohammad/data.txt'
with open(filepath) as file_object:
   print('The Content of the file before opening it for write:\n '
+file_object.read())

with open(filepath, 'w') as file_object:
   print('File opened for write:\n '

with open(filepath) as file_object:
   print('The Content of the file before opening it for write:\n ')
+file_object.read())
```

```
The Content of the file before opening it for write:
Random Data Line 0
File opened for write:

'The Content of the file before opening it for write:
```

Writing to a File

```
filepath = 'C:/Users/mohammad/data.txt'

with open(filepath, 'w') as file_object:
    file_object.write(' Random Data Line 0')
    file_object.write(' Random Data Line 1')

with open(filepath) as file_object:
 print('The Content of the file after writing:\n ')
+file_object.read())
```

```
The Content of the file after writing:
  Random Data Line 0  Random Data Line 1
```

Writing to a file--append

```
filepath = 'C:/Users/mohammad/data.txt'

with open(filepath) as file_object:
    print('The Content of the file before appending:\n ')
+file_object.read())


with open(filepath, 'a') as file_object:
   file_object.write('\nRandom Data Line 2\n ')
   file_object.write('Random Data Line 3')

with open(filepath) as file_object:
    print('The Content of the file after appending:\n ')
+file_object.read())
```

```
The Content of the file after appending:
Random Data Line 0
Random Data Line 1
The Content of the file after appending:
Random Data Line 0
Random Data Line 1
Random Data Line 2
Random Data Line 3
```

Exceptions Handling Using try-except Blocks

- You tell Python to try running some code, and you tell it what to do if the code results in a particular kind of exception.

```
try:
        #Code goes here
except:
        #what to do in case the code inside the try block created an exception
```

Handling the FileNotFoundError Exception

- One common issue when working with files is handling missing files
  - Different location
  - Filename misspelled
  - File may not exist at all

```
filepath = 'C:/Users/mohammad/data1.txt'

with open(filepath) as file_object:
 print('The Content of the file before appending:\n ')
+file_object.read())
```

```
with open(filepath) as file_object:
 FileNotFOundError:[Errno 2] No such file or
directory : 'C:/Users/mohammad/data1.txt'
```

Handling the FileNotFoundError Exception

```
filepath = 'C:/Users/mohammad/data1.txt'

try:
    with open(filepath) as file_object:
       print('The Content of the file before appending:\n ')
+file_object.read())


Except FileNotFOundError:
 print('The file ' + filepath  + ' is not found'
```

```
The file C:/Users/mohammad/data1.txt is not found
```

Handling the FileNotFoundError Exception

```
filepath = 'C:/Users/mohammad/data.txt'

try:
      with open(filepath) as file_object:
         content = file_object.read()


except FileNotFoundError:
  print('The file ' + filepath  + ' is not found '

else:
  print(content)
```

```
Random Data Line 0
Random Data Line 1
Random Data Line 2
Random Data Line 3
```

Failing Silently Using “pass”

```
filepath = 'C:/Users/mohammad/data.txt'

try:
      with open(filepath) as file_object:
         content = file_object.read()


except FileNotFoundError:
  pass

else:
  print(content)

print('done')
```

```
done
```

Storing Data

- Sometimes you need to store the information users provide in data structures such as lists and dictionaries.
- When users close a program, you'll almost always want to save the information they entered.
- A simple way to do this involves storing your data using the *json* module

json.dump() and json.load()

```
import json
numbers = [2, 3, 5, 7, 11, 13]
filepath =
'C:/Users/mohammad/numbers.json'
with open(filepath, 'w') as file_object:
    json.dump(numbers, file_object)

with open(filepath, 'r') as file_object:
    print(json.load(file_object))
```

```
[2, 3, 5, 7, 11, 13]
```

```
import json
numbers = [2, 3, 5, 7, 11, 13]
filepath =
'C:/Users/mohammad/numbers.json'
with open(filepath, 'w') as file_object:
    json.dump(numbers, file_object)

with open(filepath, 'r') as file_object:
    x = json.load(file_object )
    print(x[3])
```

```
7
```

Numpy
(Numerical Python)

Prepared by Dr. Mohammad Abdel-majeed

Outline

- Create ndarrays
- Indexing and slicing
  - Integer
  - Boolean
- Mathematical Operations
- Useful Functions
- Save and load numpy arrays
- Linalg and Scipy

Introduction

- Numpy
  - Numeric Python
  - Fast computation with *n*-dimensional arrays
  - Used for datascience

Numpy

- Based around one data structure

  ndarray
- n-dimensional array
- Import with `import numpy as np`
- Usage is `np.command(xxx)`

ndarrays

- One dimensional araay: 5,67,43,76,2,21

```
a=np.array([5,67,43,76,2,21])
```

- Two dimensional array:

```
a=np.array([[4,5,8,4],[6,3,2,1],[8,6,4,3]])
```

Create ndarray-Example

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

Create ndarray-Example

```
import numpy as np

a = np.zeros((2,2))    # Create an array of all zeros
print(a)               # Prints "[[ 0.  0.]
                       #          [ 0.  0.]]"
b = np.ones((1,2))     # Create an array of all ones
print(b)               # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"
d = np.eye(2)          # Create a 2x2 identity matrix
print(d)               # Prints "[[ 1.  0.]
                       #          [ 0.  1.]]"
e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

Create ndarray-Example

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1)
print (arr1)                    #[ 6.   7.5  8.   0.   1. ]


data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
print(arr2)     #[[1 2 3 4]
                 #[5 6 7 8]]
print(arr2.ndim)     #2
print (arr2.shape)   #(2,4)
print (arr2.shape[0])   #2
print (arr2.shape[1])   #4
```

NumPy data types 1

bool_	Boolean (True or False) stored as a byte
int_	Default integer type
intc	Identical to C int e.g int32 in64
intp	Integer used for indexing
int8	Byte (-128 to 127)
int16	Integer (-32768 to 32767)
int32	Integer (-2147483648 to 2147483647)
int64	Integer (-9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float16	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
complex64	Complex number, represented by two 32-bit floats (real and imaginary components)
complex128	Complex number, represented by two 64-bit floats (real and imaginary components)

Create ndarray-Example

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 = np.array(data1,dtype = np.bool)
print (arr1)       #[ True True True False True]


data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2,np.float32)
print(arr2)     #[[1. 2. 3. 4.]
                 #[5. 6. 7. 8.]]
print(arr2.ndim)     #2
print (arr2.shape)   #(2,4)
print (arr2.shapep[0])   #2
print (arr2.shape[1])   #4
print(arr2.dtype)     #float32
```

Change array data type

```
import numpy as np

arr = np.array([3.7, -1.2, -2.6])
print(arr) #[ 3.7 -1.2 -2.6]
print (arr.astype(np.int32)) #[ 3 -1 -2]
```

Indexing and Slicing

- Indexing can be done using the indexes of the element that want to be accessed or using slicing
- Similar to Python lists, numpy arrays can be sliced.

```
import numpy as np
arr = np.arange(10)
print (arr) #[0 1 2 3 4 5 6 7 8 9]
print (arr[5])#5
print (arr[5:8]) #[5 6 7]
arr[5:8] = 12
print (arr) #[ 0  1  2  3  4 12 12 12  8  9]
```

Indexing and Slicing

- Since arrays may be multidimensional, you must specify a slice for each dimension of the array:
- A slice of an array is a view into the same data, so modifying it will modify the original array

```
import numpy as np
arr = np.arange(10)
arr_slice = arr[5:8]
arr_slice[1] = 12345
print (arr_slice) #[5 12345 7]
print (arr) #[0 1 2 3 4 5 12345 7 8 9]
arr_slice[:] = 64 #[0 1 2 3 4 64 64 64 8 9]
print (arr)
```

2d array

- The data of each row should be between two square brackets.

```
import numpy as np
arr2d = np.array ([[1, 2, 3],
[4, 5, 6],  [7, 8, 9]])
print(arr2d[2]) #[7 8 9]
print (arr2d[0][2]) #3
print (arr2d[0, 2]) #3
```

Indexing with slices – 2D Array (Examples)

```
import numpy as np
arr2d = np.array([[1, 2, 3],  [4, 5, 6],  [7, 8, 9]])
print (arr2d) #[[1 2 3]
              # [4 5 6]
              # [7 8 9]]
print (arr2d[:2]) #[[1 2 3]
                  # [4 5 6]]
print (arr2d[:2, 1:])#[[2 3]
                     #[5 6]]
```

Indexing with slices – 2D Array (Examples)

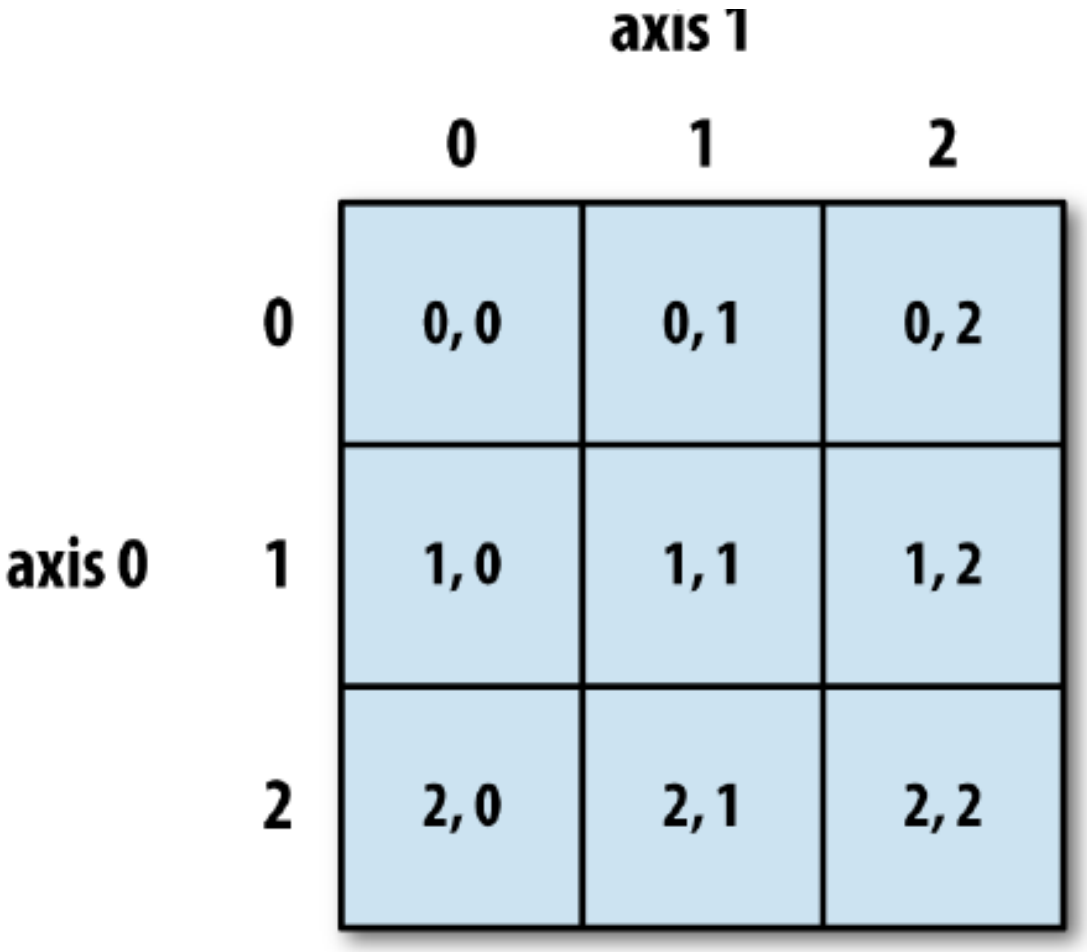
```
import numpy as np
arr2d = np.array(
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print (arr2d[1, :2])
print (arr2d[2, :1])
print (arr2d[:, :1])
```

Indexing with slices – 2D Array (Examples)

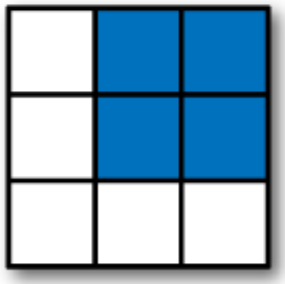
```
import numpy as np
arr2d = np.array(
[[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
print (arr2d[1, :2])
print (arr2d[2, :1])
print (arr2d[:, :1])
```

```
[4 5]

[7]

[[1]
 [4]
 [7]]
```

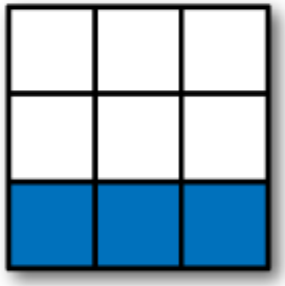
Indexing elements in a numpy array



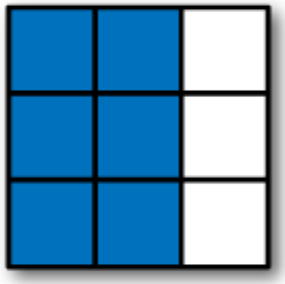
Two-dimensional array slicing



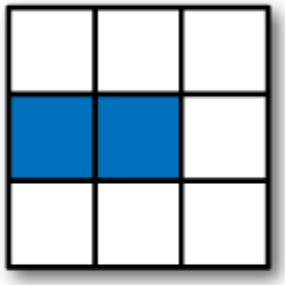
Expression	Shape
`arr[:2, 1:]`	`(2, 2)`



Expression	Shape
`arr[2]`	`(3,)`
`arr[2, :]`	`(3,)`
`arr[2:, :]`	`(1, 3)`



Expression	Shape
`arr[:, :2]`	`(3, 2)`



Expression	Shape
`arr[1, :2]`	`(2,)`
`arr[1:2, :2]`	`(1, 2)`

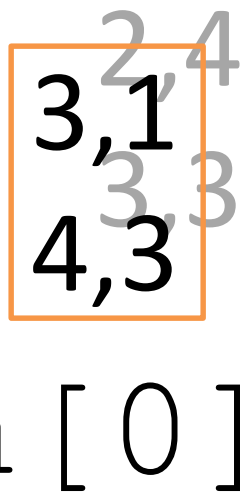
3d 2x2x2

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```

2,4
3,1
3,3
4,3

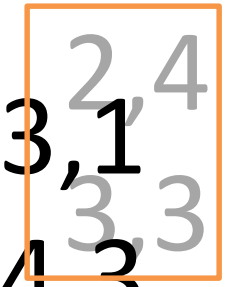
Indexing

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



Indexing

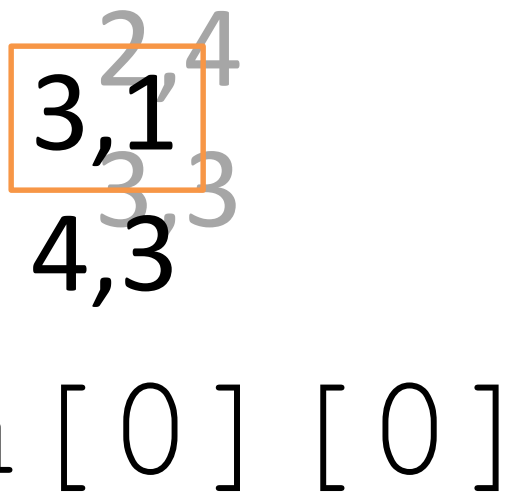
```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



a[1]

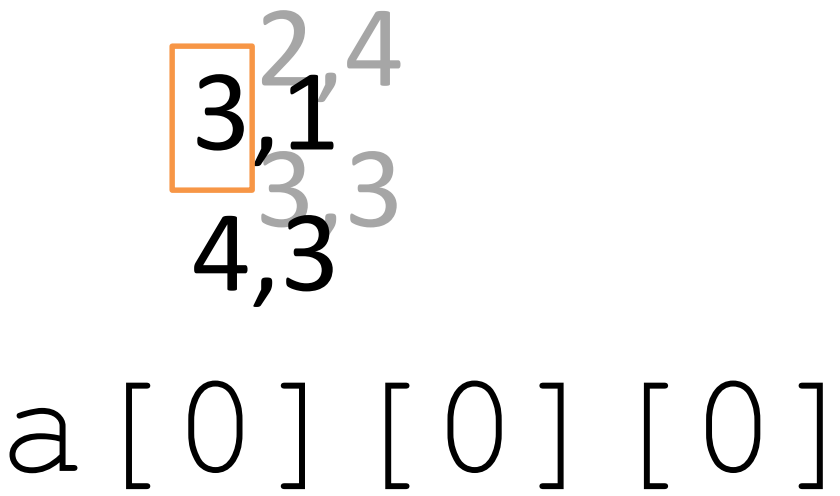
Indexing

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



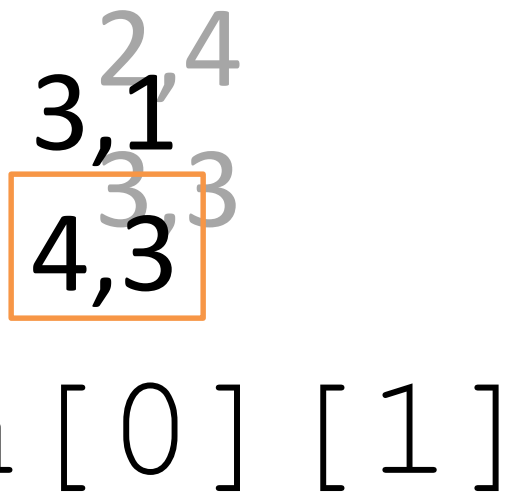
Indexing

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



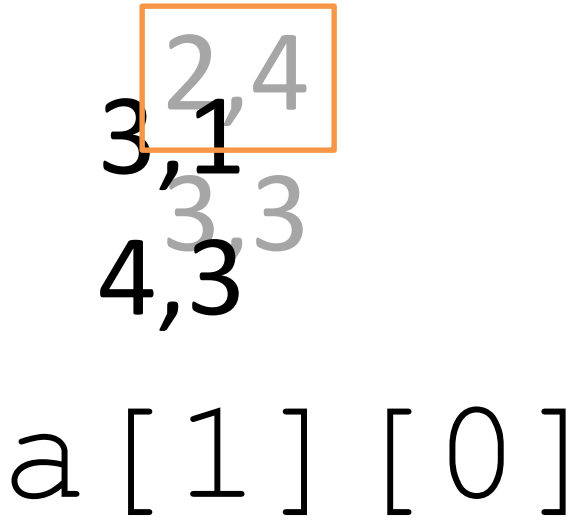
Indexing

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



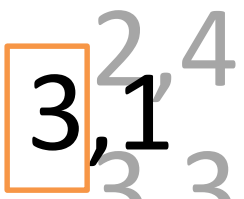
Indexing

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



Slicing

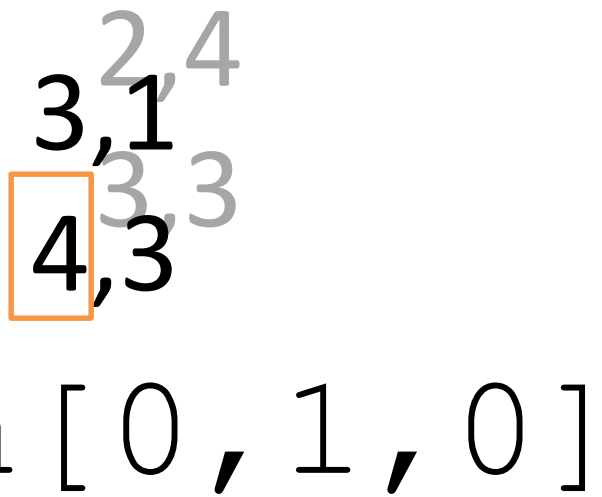
```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```




```
a[0,0,0]
```

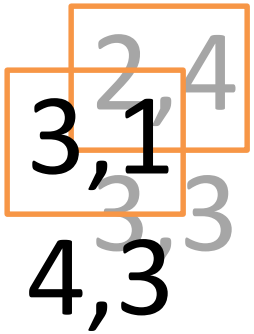
Slicing

```
a=np.array([
        [
                [3, 1],[4, 3]
        ],
        [
                [2, 4],[3, 3]
        ]
])
```



Slicing

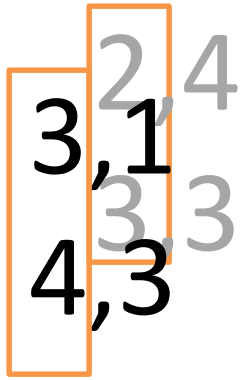
```
a=np.array([
        [
            [3, 1],[4, 3]
        ],
        [
            [2, 4],[3, 3]
        ]
])
```



`a[:,0]`

Slicing

```
a=np.array([
        [
            [3, 1],[4, 3]
        ],
        [
            [2, 4],[3, 3]
        ]
])
```



`a[:,:,0]`

Both slices, both rows, column 0

Integer Array Indexing

- Integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],[10,11,12]])
print(a)
print (a[[1, 3, 0]])
print (a[[-3, -1, -2]])
```

Integer Array Indexing

- Integer array indexing allows you to construct arbitrary arrays using the data from another array. Here is an example:

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 
print(a)
print (a[[1, 3, 0]])
print (a[[-3, -1, -2]])
```

```
#[[ 1  2  3]
# [ 4  5  5]
# [ 7  8  9]
# [10 11 12]]

#[[ 4  5  5]
# [10 11 12]
# [ 1  2  3]]

#[[ 4  5  5]
# [10 11 12]
# [ 7  8  9]]
```

Integer Array Indexing

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],[10,11,12]])

print(a[[0, 1, 2], [0, 1, 0]])
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))

print(a[[0, 0], [1, 1]])
print(np.array([a[0, 1], a[0, 1]]))
```

Integer Array Indexing

```
import numpy as np
a = np.array([[1,2,3], [4,5, 5], [7,8, 9],[10,11,12]])

print(a[[0, 1, 2], [0, 1, 0]])                    #[1  5  7]
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))      #[1  5  7]

print(a[[0, 0], [1, 1]])                          #[2  2]
print(np.array([a[0, 1], a[0, 1]]))               #[2  2]
```

Boolean Array Indexing

```
import numpy as np
a_index= np.array([True, True,False,False,True])
a = np.array([5,12,50,33,12])
print(a[a_index]) # [ 15  12  12]]
```

Stacking

- Create an array by stacking numpy arrays.

```
x = np.arange(0,10,2) # x=([0,2,4,6,8])
y = np.arange(5) # y=([0,1,2,3,4])
m = np.vstack([x,y]) # m=([[0,2,4,6,8],
# [0,1,2,3,4]])
xy = np.hstack([x,y]) # xy =([0,2,4,6,8,0,1,2,3,4])
```

Stacking

- Create an array by stacking numpy arrays.
  - vstack: Stack along first axis.
  - hstack: Stack along second axis.
  - dstackL Stack along the third axis.

```
x = np.arange(0,10,2) # x=([0,2,4,6,8])
y = np.arange(5) # y=([0,1,2,3,4])
m = np.vstack([x,y]) # m=([[0,2,4,6,8],
# [0,1,2,3,4]])
xy = np.hstack([x,y]) # xy =([0,2,4,6,8,0,1,2,3,4])
```

Broadcasting/Mathematical operations

- Basic mathematical functions operate elementwise on arrays,
  - Available both as operator overloads and as functions in the numpy module
  - Numpy operations are usually done on pairs of arrays on an element-by-element basis
  - Between **arrays** and **scalar value and array**

For more details and examples
https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

Elementwise opwerations/Mathematical (Between arrays)

```
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (arr*arr)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

```
[[ 0  1  4]
 [ 9 16 25]
 [36 49 64]]
```

Elementwise operations/mathematical

```
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (np.sqrt(arr))
print (np.exp(arr))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[[0.         1.         1.41421356]
 [1.73205081 2.         2.23606798]
 [2.44948974 2.64575131 2.82842712]]

[[1.00000000e+00 2.71828183e+00 7.38905610e+00]
 [2.00855369e+01 5.45981500e+01 1.48413159e+02]
 [4.03428793e+02 1.09663316e+03 2.98095799e+03]]
```

Elementwise Operations (scalar)

```
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (arr * 3)
print (arr + 4)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[[ 0  3  6]
 [ 9 12 15]
 [18 21 24]]

[[ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

Elementwise Operations (scalar)

```
import numpy as np
data = np.arange(5,dtype=np.int32)
print (data)#[0 1 2 3 4]
print (data * 10)#[ 0 10 20 30 40]
print (data + data)#[0 2 4 6 8]
print (1/((data+1)))#[1.  0.5  0.33333333 0.25  0.2 ]
```

Elementwise Operations?? Statistics(max,min ,sum)

```
import numpy as np
x = np.random.randn(4)
y = np.random.randn(4)
y = np.round(y,1)
print(x)
print(y)
print(np.maximum(x, y))
print(np.add(x,y)) #Equivalent to x + y
print(np.exp(x))
print(np.round(y))
```

```
[ 0.10762685 -0.10194233 -0.31373994  0.86389688]
[-0.2 -0.9 -0.4  0.3]
[ 0.10762685 -0.10194233 -0.31373994  0.86389688]
[-0.09237315 -1.00194233 -0.71373994  1.16389688]
0.5558414572185999
[1.11363211 0.90308163 0.73070903 2.37238762]
[-0.2 -0.9 -0.4  0.3]
[-0. -1. -0.  0.]
```

Inner Product

```
import numpy as np
import numpy as np
arr = np.arange(9).reshape((3, 3))
print (arr)
print (arr.T)
print (np.dot(arr, arr.T))
print(np.matmul(arr,arr.T))
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]

[[0 3 6]
 [1 4 7]
 [2 5 8]]

[[  5  14  23]
 [ 14  50  86]
 [ 23  86 149]]

[[  5  14  23]
 [ 14  50  86]
 [ 23  86 149]]
```

Axis operations

- Instead of applying the mathematical operations on the entire array they can be done **per-row or per-column**
  - **You should specify the axis**
  - **axis=0→ applied on each column**
  - **axis=1→ applied on each row**

Axis operations

```
import numpy as np
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print (arr) #[[0 1 2]
            # [3 4 5]
            #[6 7 8]]
print (arr.mean(axis=0)) #[3. 4. 5.]
print (arr.mean(axis=1)) #[1. 4. 7.]
```

Axis operations/sum

```
import numpy as np
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print (arr) #[[0 1 2]
            #[3 4 5]
            #[6 7 8]]

print (arr.sum(0)) #[ 9 12 15]

print (arr.sum(1))#[ 3 12 21]

print (arr.cumsum(0))#[[ 0  1  2]
                     #[ 3  5  7]
                     #[ 9 12 15]]

print (arr.cumprod(axis=1))#[[  0   0   0]
                           #[  3  12  60]
                           #[  6  42 336]]
```

Relational operators and numpy

```
import numpy as np
x = np.arange(1,9)
y = x>5
print(y)
```

```
[False False False False False  True  True  True]
```

Where()

- Where(Condition, if True, If False)
  - Returns numpy.ndarray array
  - Use the name of the array to keep the values the same

```
import numpy as np
arr = (np.random.random(16)).reshape(4,4)
print(arr)
print (np.where(arr > 0.5, 2, -2))
```

```
[[0.11305372 0.41972489 0.71758276 0.7024291 ]   [[-2 -2  2  2]
 [0.28989595 0.71371535 0.58332619 0.69298548]    [-2  2  2  2]
 [0.48567377 0.00463536 0.57581238 0.27679739]    [-2 -2  2 -2]
 [0.36887073 0.35191625 0.77679602 0.40723983]]   [-2 -2  2 -2]]
```

Where()

- Returns elements chosen from *x* or *y* depending on the *condition*.

```
import numpy as np
x = [1,2,3]
y = [10, 20, 30]
condition = [True, False, True]
np.where(condition,x,y) #Output is [ 1 20  3]
```

Relational operators and numpy (with where)

```
import numpy as np
x = np.arange(1,9)
y = x>5
print(y)
print(np.where(y))
print(np.where(y)[0])
#use np.nonzero(y)
```

```
[False False False False False  True  True  True]
(array([5, 6, 7], dtype=int64),)
[5 6 7]
```

Boolean Arrays

```
import numpy as np
arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
print ((arr > 4).sum()) #4

arr = np.array(
[False, False, True, False])
print (arr.any()) #True
print (arr.all()) #False
```

Sorting

```
import numpy as np
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
arr.sort()
print (arr)#[[0 2 5]
           #[1 4 6]
           #[3 6 7]]
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
arr.sort(0)
print (arr) )#[[0 4 1]
            # [6 5 2]
            # [6 7 3]]
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
arr.sort(1)
print (arr)  #[[0 2 5]
            # [1 4 6]
            # [3 6 7]]
```

Unique

```
import numpy as np
names = np.array(['Bob', 'Joe', 'Will', 'Bob', 'Will', 'Joe'])
print (np.unique(names)) #['Bob' 'Joe' 'Will']
print (sorted(set(names)))#['Bob' 'Joe' 'Will']
```

numpy.in1d

- Test whether each element of a 1-D array is also present in a second array.
  - Returns a boolean array the same length as *ar1* that is True where an element of *ar1* is in *ar2* and False otherwise.

```
print(np.in1d(ar1,ar2))
```

```
import numpy as np
arr = np.array([[0, 5, 2], [6, 4, 1], [6, 7, 3]])
print(np.in1d([1,5],arr)) #[ True  True]
```

Get index (argsort(),argwhere(),argmax())

- Used when interested in the index of the elements rather than value
- np.argmax()→ Returns the indices of the maximum values along an axis.
- np.argsort()→ Returns the indices that would sort an array.
- Np.argwhere()→Find the indices of array elements that are non-zero, grouped by element.

Get index (argsort(),argwhere(),argmax())

```
import numpy as np
x = np.array([3,6,12,5,3,77,67,43,23,50,77,11,24])
print(x)
print(np.argmax(x))
print(np.argsort(x))
print(np.argwhere(x>50))
```

```
[ 3  6 12  5  3 77 67 43 23 50 77 11 24]
5
[ 0  4  3  1 11  2  8 12  7  9  6  5 10]
[[ 5]
 [ 6]
 [10]]
```

Save numpy array

- numpy.save(): Saves an array to a binary file in numpy *.npy* format
  - Parameters: file name and numpy array

```
import numpy as np
arr = np.arange(10)
print (arr)#[0 1 2 3 4 5 6 7 8 9]
np.save('some_array', arr)
arr1 = np.load('some_array.npy')
print (arr1)#[0 1 2 3 4 5 6 7 8 9]
```

Saving multiple numpy arrays

```
import numpy as np
arr3 = np.arange(3)
arr5 = np.arange(5)
np.savez('array_archive.npz', a=arr3, b=arr5)
arch = np.load('array_archive.npz')
print (type(arch))#<class 'numpy.lib.npyio.NpzFile'>
print (arch['a'])#[0 1 2]
print (arch['b'])#[0 1 2 3 4]
print (dict(arch))
#{'a': array([0, 1, 2]), 'b': array([0, 1, 2, 3, 4])}
```

Loading text data into numpy array

```
import numpy as np
arr1 = np.loadtxt('array_ex.txt', delimiter=',')
print (arr1)#[[ 1.  2.  3.  4.]
            #[12. 13. 14. 15.]]
print (type(arr1)) #<class 'numpy.ndarray'>
```

Loading text data into numpy array

- Using **genfromtxt**: gives you some options like the parameters **missing_values**, **filling_values** that can help you dealing with an incomplete data

```
fill_values = (111, 222, 333, 444, 555) # one for each column
np.genfromtxt(filename,delimiter=',',filling_values=fill_values)
```

```
1,2,,,5
6,,8,,
11,,,,
```

```
array([[   1.,    2.,  333.,  444.,    5.],
       [   6.,  222.,    8.,  444.,  555.],
       [  11.,  222.,  333.,  444.,  555.]])
```

Scipy

- SciPy is a library that uses NumPy for more mathematical functions.
- SciPy uses NumPy arrays as the basic data structure,
- used tasks in scientific programming, including linear algebra, integration (calculus), ordinary differential equation solving, and signal processing.
- For a quick start on the functions check this

https://www.edureka.co/blog/scipy-tutorial/#numpyvsscipy

Scipy

```
from scipy import special
print(special.exp10(5))
```

Linear Algebra
Numpy.linalg

- Available in scipy and numpy
- Scipy version is more comprehensive and faster
  - Matrix and vector products
  - Decompositions
  - Matrix eigenvalues
  - Norms and other numbers
  - Solving equations and inverting matrices
  - Exceptions
  - Linear algebra on several matrices at once

For more details:

https://docs.scipy.org/doc/numpy-1.11.0/numpy-user-1.11.0.pdf

References

- Numpy Documentation, http://Scipy.org
- Python for Data Analysis by Katia Oleinik

Pandas

Prepared by Dr. Mohammad Abdel-majeed

Outline

- Series and Dataframes
- Reading the data
- Exploring the data
- Indexing
- Selection
- Data Analysis
- Grouping
- Applying functions
- Sorting
- Missing values
- Combining

Pandas

- Adds data structures and tools designed to work with table-like data
- Provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- Allows handling missing data

Pandas Data Structures

- Series: one dimensional data structure that can store values — and for every value it holds a unique index, too.

- DataFrame: two (or more) dimensional data structure – basically a table with rows and columns. The columns have names and the rows have indexes.

Series

```
S = pd.Series([15,20,13,55,67,34,23,1])
print(S)
```

```
0    15
1    20
2    13
3    55
4    67
5    34
6    23
7     1
```

Dataframes

- Index by default is an integer and starts from 0

```
df = pd.DataFrame({'Name':['Mohammad', 'Ahmad','Haneen','Leen'],
'Age':[12,25,40,17]})
print (df)
```

	Age	Name
0	12	Mohammad
1	25	Ahmad
2	40	Haneen
3	17	Leen

DataFrames with Index

```
df = pd.DataFrame({'Name':['Mohammad',
'Ahmad','Haneen','Leen'],
                   'Age':[12,25,40,17]},
                  index = ['s1','s2','s3','s4'])
```

	Age	Name
s1	12	Mohammad
s2	25	Ahmad
s3	40	Haneen
s4	17	Leen

Reading Data Files

- Several files types can be accessed and their content will be stored in Series or DataFrame

```
import numpy as np
import pandas as pd
filename =r'C:\Users\mohammad\Desktop\movies.xls'
movies = pd.read_excel(filename)#reads the first
sheet, xlrd
print(movies.shape)#(1604,19)
```

Reading Data Files

- Several files types can be accessed and their content will be stored in Series or DataFrame

```
import numpy as np
import pandas as pd
filename =r'C:\Users\mohammad\Desktop\movies.xls'
movies = pd.read_excel(filename,sheet_name=1)#reads
the second sheet, xlrd
print(movies.shape)#(1604,19)
```

Other read_*

- https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql

Exploring the Data

- Data types

```
print(movies.dtypes)
print(movies.Country.dtype)
movies.Duration.astype('int32')#make sure that you do not have NA values
```

```
Title                          object
Year                          float64
Genres                         object
Language                       object
Country                        object
Duration                      float64
Budget                        float64
Gross Earnings                float64
Director                       object
Actor 1                        object
Actor 2                        object
Facebook Likes - Actor 1      float64
Facebook Likes - Actor 2      float64
Facebook likes - Movie          int64
Facenumber in posters         float64
User Votes                      int64
Reviews by Users              float64
Reviews by Crtiics            float64
IMDB Score                    float64
```

Exploring the Data

```
filename =r'C:\Users\mohammad\Desktop\movies.xls'
movies = pd.read_excel(filename)#reads the first sheet, xlrd
print(movies.shape)
print(movies.head(4))#movies.tail()
```

```
(1604, 19)
                          Title     Year  ... Reviews by Crtiics IMDB Score
0                   127 Hours   2010.0  ...              450.0         7.6
1                3 Backyards    2010.0  ...               20.0         5.2
2                           3   2010.0  ...               76.0         6.8
3  8: The Mormon Proposition    2010.0  ...               28.0         7.1
```

Exploring the Data

```
import numpy as np
import pandas as pd
filename =r'C:\Users\mohammad\Desktop\movies.xls'
movies = pd.read_excel(filename)#reads the first sheet, xlrd
print(movies.columns)
```

```
Index(['Title', 'Year', 'Genres', 'Language', 'Country', 'Duration', 'Budget',
       'Gross Earnings', 'Director', 'Actor 1', 'Actor 2',
       'Facebook Likes - Actor 1', 'Facebook Likes - Actor 2',
       'Facebook likes - Movie', 'Facenumber in posters', 'User Votes',
       'Reviews by Users', 'Reviews by Crtiics', 'IMDB Score'],
      dtype='object')
```

Exploring the Data/ Data Frames attributes

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Add Columns Titles

- By default the first row is considered the columns headers.
- In case there is no columns headers then you read the data as follows:

```
movies = pd.read_excel(filename,header=None)
```

- create a list of the columns headers and assign it to the columns variable of the datafreme

```
movies.columns= list(range(19))
movies.columns = ['col1','col2'…]
```

Add Columns Titles

- By default the headers will be the integer values starting from 0.
- To add column headers you have to create a list of the columns headers names and assign it to the columns variable of the datafreme

```
movies.columns = [‘col1’,’col2’…]
```

Data Access/Column Access

- To access the data you can use the column header as follows:

```
print(movies['Title'])
Print(movies.Title)
```

```
0                                        127 Hours
1                                    3 Backyards
2                                              3
3                        8: The Mormon Proposition
4              A Turtle's Tale: Sammy's Adventures
                              ...
1602                 Wuthering Heights
1603          Yu-Gi-Oh! Duel Monsters
Name: Title, Length: 1604, dtype: object
```

Data Access/Column Access

- To access the data you can use the column header as follows:

```
print(movies[['Year','IMDB Score',]])
```

```
 Year  IMDB Score
0      2010.0          7.6
1      2010.0          5.2
2      2010.0          6.8
3      2010.0          7.1
4      2010.0          6.1
...       ...          ...
1600      NaN          7.3
1601      NaN          7.1
1602      NaN          7.7
1603      NaN          7.0
```

Renaming Columns

- To access columns you have to avoid spaces in column name

```
df2 = pd.DataFrame([[1, 1, 1, 1],
                    [2, 2, 2, 2]],
                   columns=['A 1','B 1','C 1','D 1'])
print(df2)
df2.columns = [c.replace(' ', '_') for c in df2.columns]
print(df2)
```

```
   A 1  B 1  C 1  D 1
0    1    1    1    1
1    2    2    2    2
   A_1  B_1  C_1  D_1
0    1    1    1    1
1    2    2    2    2
```

Indexing

- Works just like they do in the rest of the Python ecosystem.
- Pandas has its own access operators
  - ***loc*** : label based selection
  - **iloc:** index based selection

Index Based Selection (iloc)

```
print(movies.iloc[5])#returns row 5
print(movies.iloc[:5])#returns row 0,1,2,3,4
print(movies.iloc[:5,0])#returns titles (column 0) of the
first 5 moviess
print(movies.iloc[[0,1,2,3,4],0])#returns titles (column0)
of the first 5 movies
print(movies.iloc[[0,1,2,3,4],18])#returns IMDB scores
(column 18) of the first 5 movies
```

Label Based Selection (loc)

- When using loc the indexing is **inclusive**
  - The start and end are included.

```
movies = pd.read_excel(filename)#reads the first
sheet, xlrd
print(movies.loc[5])#returns row 5
print(movies.loc[:5])#returns row 0,1,2,3,4,5
print(movies.loc[:5,'Title'])#returns titles of the
first 6 movies
print(movies.loc[[0,1,2,3,4],'IMDB Score'])#returns
IMDB Scores of the first 5 movies
print(movies.loc[-5:,'IMDB Score'])#returns IMDB
Scores of the last 5 movies
print(movies.loc[-5:,['IMDB Score','Title']])#returns
IMDB Scores and titles of the last 5 movies
```

set_index()

- An Index column will be added to the dataframe by default
  - The range is from 0→#of rows -1
- set_index() can be used to set any of the columns values to be used as a row index
  - Duplicates are allowed

set_index()

```
df = pd.DataFrame({'Name':['Mohammad',
'Mohammad','Haneen','Leen'],
'Age':[12,25,40,17],
'Hobby':['Soccer','Singing','Reading','Reading']})

X = df.set_index('Age')
df.set_index('Name',inplace=True)
print(X)
print(df)
print(df.loc['Mohammad'])
```

Age	Hobby	Name
12	Soccer	Mohammad
25	Singing	Mohammad
40	Reading	Haneen
17	Reading	Leen

Name	Age	Hobby
Mohammad	12	Soccer
Mohammad	25	Singing
Haneen	40	Reading
Leen	17	Reading

Name	Age	Hobby
Mohammad	12	Soccer
Mohammad	25	Singing

Selection

- Several Techniques can be used to select certain elements
  - Relational Operators >,<,>=….
  - isin()
  - notnull(), isnull()

Selection/Examples

```
print(movies.loc[movies.Country== 'Spain'])
print(movies[movies.Country== 'Spain'])

print(movies[(movies['Country']== 'Spain') &
(movies['Reviews by Users']>400)])

print(movies[(movies['Year']== 2012) |
(movies['Year']==2011)])
print(movies.loc[movies.Year.isin([2011,2012])])

print(movies.loc[movies.Budget.notnull()])
print(movies.loc[movies.Budget.isnull()])
```

Assigning Data

- Assignment operator is used
  - Broadcasting is supported

```
movies.loc[3,'Budget'] = 1500
movies['Title']= 'New Title'
movies.loc[:5,'Title'] = 'New'
movies.head(5).Year= 2000#!!!!
print(movies.head(10))
```

Data Analysis
describe()

- Generates a high-level summary of the attributes of the given column.
  - It is type-aware, meaning that its output changes based on the data type of the input
  - For numeric data, the result's index will include **count, mean, std, min, max and 25, 50 and 75 percentiles**
  - For object data (e.g. strings or timestamps), the result's index will include **count, unique, top, and freq**

Data Analysis
describe()

- For mixed data types provided via a DataFrame, the default is to return only an analysis of **numeric** columns.

```
print(movies.describe())
```

	Year	Duration	...	Reviews by Crtiics	IMDB Score	
count	1497.000000	1594.000000	...	1571.000000	1604.000000	
mean	2012.773547	103.328733	...	187.586887	6.337718	
std	1.868725	27.429001	...	165.281572	1.169382	
min	2010.000000	7.000000	...	1.000000	1.600000	
25%	2011.000000	92.000000	...	38.000000	5.700000	
50%	2013.000000	102.000000	...	159.000000	6.400000	
75%	2014.000000	114.750000	...	288.000000	7.100000	
max	2016.000000	511.000000	...	813.000000	9.500000	

Data Analysis
describe()

- Numerical Fields

```
print(movies.Duration.describe())
```

```
count    1594.000000
mean      103.328733
std        27.429001
min         7.000000
25%        92.000000
50%       102.000000
75%       114.750000
max       511.000000
Name: Duration, dtype: float64
```

Data Analysis/Summary describe()

- String Fields

```
print(movies.Title.describe())
```

```
count       1604
unique      1551
top         Home
freq           3
Name: Title, dtype: object
```

Data Analysis/Basic Statistics

```
print(movies.describe().loc['max','Budget'])
print(movies.Budget.max())
print(movies.Budget.mean())

print(movies.Duration.mean().round())
movies.Duration += 15#Broadcasting
print(movies.Duration.mean().round())
```

```
600000000.0
600000000.0
40563243.28888889

103
115
```

Data Analysis/Summary Aggregation

- agg() method are useful when multiple statistics are computed per column:

```
print(movies[['Budget','IMDB Score']].agg([len,min,max]))
```

```
            Budget  IMDB Score
len         1604.0      1604.0
min         1400.0         1.6
max   600000000.0         9.5
```

Data Analysis/Summary describe()

```
print(movies.Year.unique())

print(movies.Year.value_counts())

print((movies.Year.value_counts()))

X = (movies.Year.value_counts())
print(X[2012])
```

```
[2010. 2011. 2012…...
2016.   nan]

2014.0    252
2013.0    237
.
.
2012.0    221
2016.0    106
Name: Year, dtype: int64
2014.0    252
2013.0    237
2010.0    230
.
.
2016.0    106
Name: Year, dtype: int64

221
```

Grouping

- Groupby() method is used to group the rows in the dataframe based on certain column(s) values.
  - movies.groupby(['Country'])→ groups the rows based on the Country
    - #of groups will be equal to the number of countries
  - We can perform operations on each group

```
grouped = movies.groupby('Country')
print(grouped.groups)
print(grouped.get_group('Jordan'))
```

Grouping

- Example: Find the budget spent by each country on movies production

```
print(movies.groupby(['Country']).Budget.sum())
```

- Example: Find the number of movies produced by each country

```
print(movies.groupby(['Country']).count())
```

- Note that the grouping category will be the new index for the generated Dataframe.

Grouping and Aggregation

- Use agg() to display more than one function per group
  - Results generated per group

```
print(movies.groupby(['Country']).Budget.agg([len,min,max]))
```

Country	len	min	max
Australia	18.0	2500000.0	150000000.0
Bahamas	1.0	5000000.0	5000000.0
Belgium	4.0	15000000.0	34000000.0

Grouping

- Notice that the result has new index and in this case multi-index.

```
print(movies.groupby(['Country','Language']).Budget.agg([len
,min,max]))
```

```
Brazil                English                1.0    3000000.0     3000000.0
                      Portuguese             2.0    4000000.0     4000000.0
...                                          ...          ...           ...
USA                    English            1174.0       1400.0  263700000.0
                      Hebrew                 1.0          NaN           NaN
                      None                   1.0    4000000.0     4000000.0
                      Spanish                3.0    1200000.0     6000000.0
United Arab Emirates  Arabic                 1.0     125000.0      125000.0
```

DataFrameGroupBy.filter()

- Return a copy of a DataFrame excluding elements from groups that do not satisfy the boolean criterion specified by func.

```
df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar',
                          'foo', 'bar'],
                   'B' : [1, 2, 3, 4, 5, 6],
                   'C' : [2.0, 5., 8., 1., 2., 9.]})
grouped = df.groupby('A')
print(grouped.filter(lambda x: x['B'].mean() > 3.))
```

```
     A  B    C
1  bar  2  5.0
3  bar  4  1.0
5  bar  6  9.0
```

DataFrameGroupBy.apply()

- Apply certain function on the group elements

```
grouped = df.groupby('A')
print(grouped.apply(lambda x:
x.describe()))
```

		B	C
A			
bar	count	3.0	3.000000
	mean	4.0	5.000000
	std	2.0	4.000000
	min	2.0	1.000000
	25%	3.0	3.000000
	50%	4.0	5.000000
	75%	5.0	7.000000
	max	6.0	9.000000
foo	count	3.0	3.000000
	mean	3.0	4.000000
	std	2.0	3.464102
	min	1.0	2.000000
	25%	2.0	2.000000
	50%	3.0	2.000000
	75%	4.0	5.000000
	max	5.0	8.000000

DataFrameGroupBy.apply()

```
def f(group):
    return pd.DataFrame({'original': group,
                         'demeaned': group - group.mean()})
grouped = df.groupby('A')
print(grouped['C'].apply(f))
```

	demeaned	original
0	-2.0	2.0
1	0.0	5.0
2	4.0	8.0
3	-4.0	1.0
4	-2.0	2.0
5	4.0	9.0

Groupby()

- reset_index() can be used to reset the index to decimalvalues starting from 0

```
print(movies.groupby(['Country','Language']).Budget.agg([len
,min,max]).reset_index())
```

```
3                 Brazil      English      1.0   3000000.0    3000000.0
4                 Brazil   Portuguese      2.0   4000000.0    4000000.0
..                   ...          ...      ...         ...          ...
78                   USA      English   1174.0      1400.0  263700000.0
79                   USA       Hebrew      1.0         NaN          NaN
80                   USA         None      1.0   4000000.0    4000000.0
```

Sorting

- sort_values() function/method can be used to sort dataframes according to certain column values.

```
print(movies.sort_values('Country').iloc[:,:5])
```

```
                      Title          Year  ... Language      Country
1138          The Water Diviner   2014.0   ...   English    Australia
859            The Great Gatsby   2013.0   ...   English    Australia
860            The Great Gatsby   2013.0   ...   English    Australia
.
.
```

Sorting

```
print(movies.groupby(['Country'])['IMDB
Score'].max().sort_values(0,ascending=False))

print(movies.groupby(['Country'])['IMDB
Score'].agg([max]).sort_values('max',ascending=False))
```

Sorting

- sort_values() works on Dataframes or Series objects

```
print(type(movies.groupby(['Country'])))
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>

print(type(movies.groupby(['Country'])['IMDB Score']))
<class 'pandas.core.groupby.generic.SeriesGroupBy'>

print(type(movies.groupby(['Country'])['IMDB Score'].max()))
<class 'pandas.core.series.Series'>
```

Sorting

- sort_values() can sort by more than one column.
- sort_index() is used to sort elements by index.

```
print(movies.sort_values(['Language','Country']).iloc[:,:5])
```

```
                                      Title  ...                 Country
884                                The Square  ...                   Egypt
845                     The Brain That Sings  ...  United Arab Emirates
308  In the Land of Blood and Honey           ...                     USA
1026                        Kung Fu Killer    ...                   China
1164                                Z Storm   ...                Hong K
```

Missing Data

- Several Methods are available to deal with missing data

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Missing Data

- To select NaN entries you can use pd.isnull() (or its companion pd.notnull())

```
print(movies[pd.isnull(movies.Country)])
```

	Title	Year	...	Reviews by Crtiics	IMDB Score
963	Dawn Patrol	2014.0	...	9.0	4.8
1497	10,000 B.C.	NaN	...	NaN	7.2
1529	Gone, Baby, Gone	NaN	...	NaN	6.6
1551	Preacher	NaN	...	18.0	8.3

Missing Data

- To select NaN entries you can use pd.isnull() (or its companion pd.notnull())

```
movies[movies.isnull().any(axis=1)].head()
```

Replacing Missing Values

- Replacing missing values is a common operation.
- fillna() provides a few different strategies for mitigating such data

```
movies.Country = movies.Country.fillna("X")
print(movies.iloc[963])

movies.Country.fillna("X",inplace = True)
print(movies.iloc[963])
```

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html

Removing Records with Missing Values

- dropna() can be used to remove all the records with ‘na’ values.

```
print(movies.shape) #(1604, 19)
movies.dropna(inplace=True)
print(movies.shape)#(1044, 19)
```

fillna()/Examples

```
df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns=list('ABCD'))

print(df)
df1 = df.fillna(method='ffill')
print(df1)

values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
df2 = df.fillna(value=values)
print(df2)
```

```
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4

     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  3.0  4.0 NaN  5
3  3.0  3.0 NaN  4

     A    B    C  D
0  0.0  2.0  2.0  0
1  3.0  4.0  2.0  1
2  0.0  1.0  2.0  5
3  0.0  3.0  2.0  4
```

Renaming

- lets you change index names and/or column names
- Change column name

```
movies.rename(columns={'IMDB Score':'IMDB_Score'},inplace=True)
```

- Change index
  - Rarely used; set_index() can be used instead

```
movies.rename(index = {0:'m0',1:'m1'})
```

Combining

- Dataframes can be combined into one Dataframe
  - concat(), join() and merge() are useful methods for this purpose.

Combining/Example

```
df1 = pd.DataFrame([[1, 2, 5, 0],
                    [3, 4, 6, 1]],
                   columns=list('ABCD'))

df2 = pd.DataFrame([[1, 1, 1, 1],
                    [2, 2, 2, 2]],
                   columns=list('ABCD'))

df4 = pd.DataFrame([[1, 1, 1, 1],
                    [2, 2, 2, 2]],
                   columns=list('ABCF'))
```

Combining/concat()

- Concatenate along an axis

```
df3 = pd.concat([df1,df2],ignore_index=True)
print(df3)

df3 = pd.concat([df1,df2],axis=1)
print(df3)
```

```
   A  B  C  D
0  1  2  5  0
1  3  4  6  1
2  1  1  1  1
3  2  2  2  2

   A  B  C  D  A  B  C  D
0  1  2  5  0  1  1  1  1
1  3  4  6  1  2  2  2  2
```

Combining/concat()

- Concatenate with different columns labels

```
df3 = pd.concat([df1,df4],sort=False,ignore_index=True)
print(df3)
```

```
   A  B  C    D    F
0  1  2  5  0.0  NaN
1  3  4  6  1.0  NaN
2  1  1  1  NaN  1.0
3  2  2  2  NaN  2.0
```

Combining/join()()

- Concatenate with different columns labels

```
df3 = df1.join(df4,lsuffix="_X",rsuffix="_Y")
print(df3)
```

```
   A_X  B_X  C_X  D  A_Y  B_Y  C_Y  F
0    1    2    5  0    1    1    1  1
1    3    4    6  1    2    2    2  2
```

References

- https://pandas.pydata.org/docs/
- https://pandas.pydata.org/pandas docs/stable/user_guide/groupby.html

Data Visualization using Matplotlib

Prepared by Dr. Mohammad Abdel-majeed

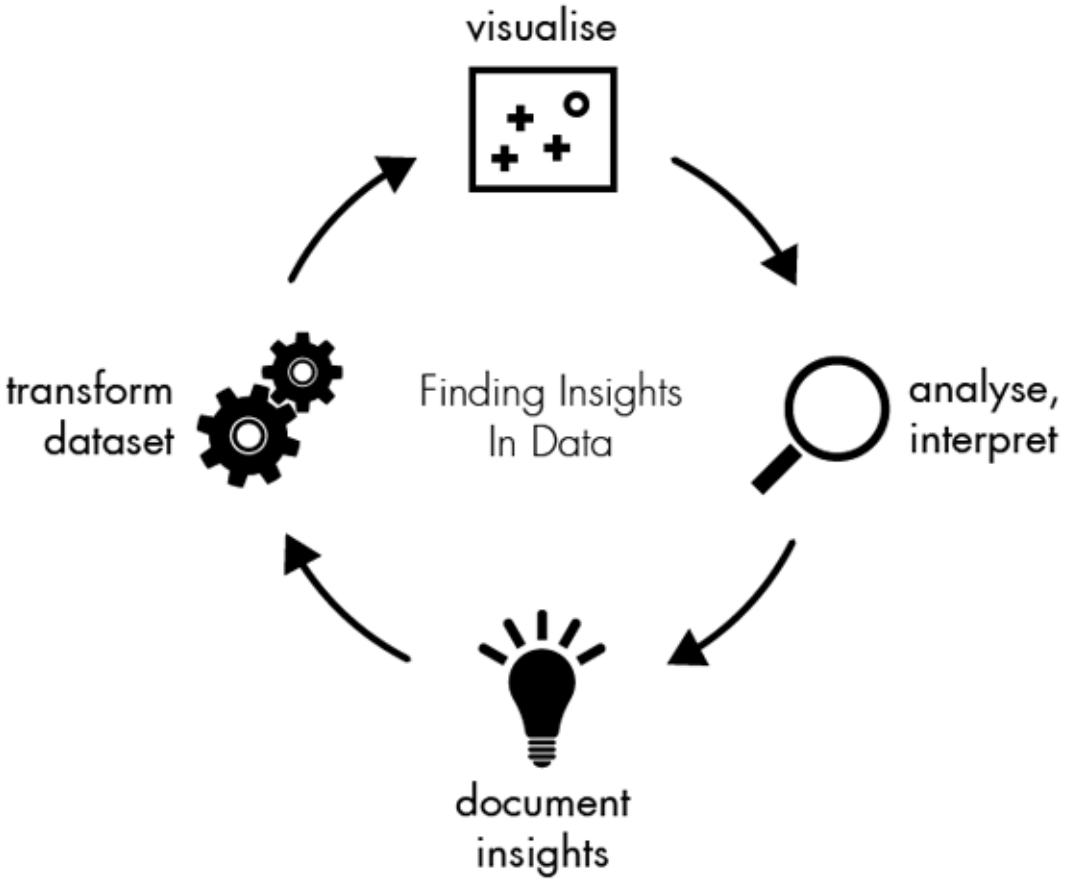
Outline

- Data Visualization
- Matplotlib.pyplot
  - Line plot
  - Bar Plot
  - Scatter Plot
  - Histogram Plot
  - Pie Plot
  - SubPlot
  - Annotation
- Seaborn

Data Visualization

- Human brain Process information faster when it is in graphical form
- Accessible way to see and understand trends, outliers, and patterns in data.
- Data visualization tools are essential to analyze and interpret massive amounts of information to make data-driven decisions.

Data Visualization

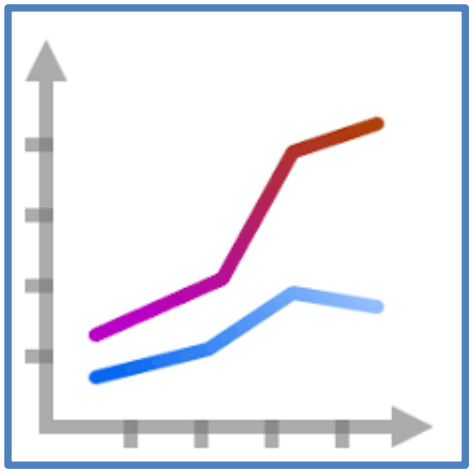


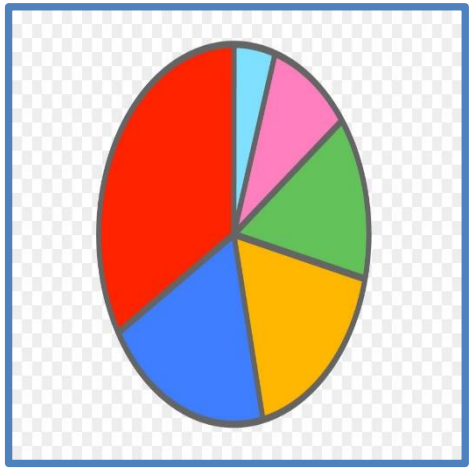
https://datajournalism.com/read/handbook/one/understanding-data/using-data-visualization-to-find-insights-in-data

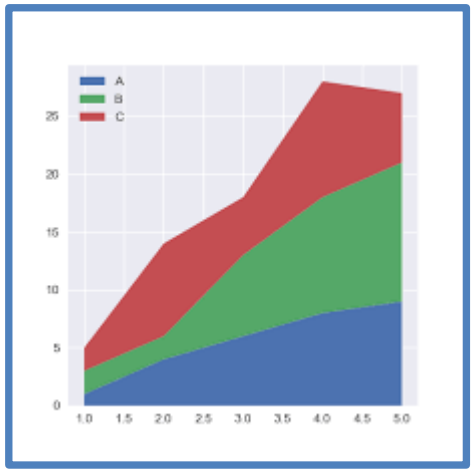
Matplotlib.pyplot

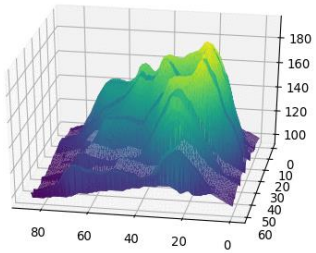
- Collection of command style functions that make matplotlib work like MATLAB.
- Various states are preserved across function calls, so that it keeps track of things like:
  - The current figure and plotting area
  - The plotting functions are directed to the current axes

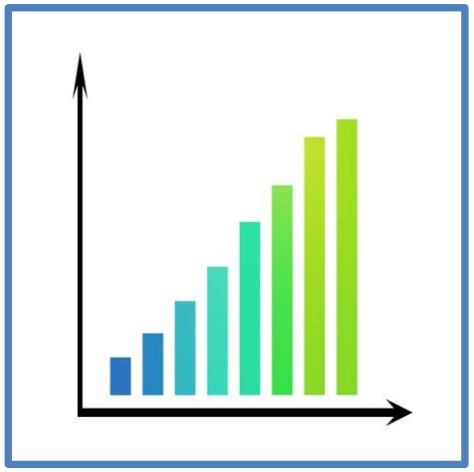
Examples






A
B
C
25
20
15
10
5
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


180
160
140
120
100
0
10
20
30
40
50
60
80
60
40
20
0

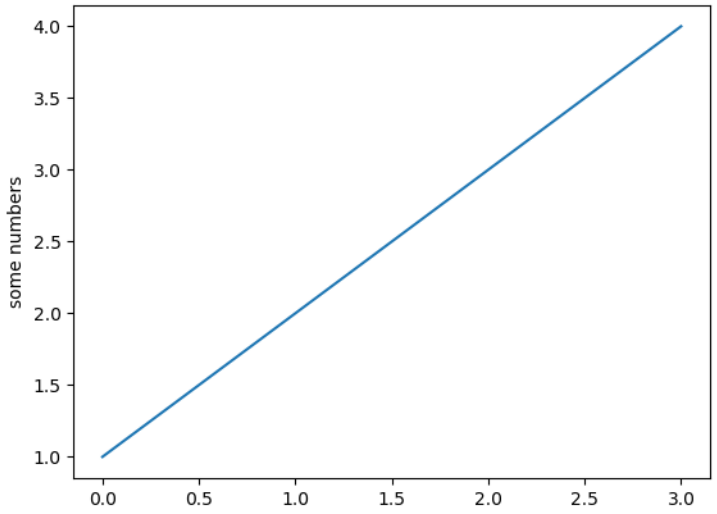


Examples

- https://matplotlib.org/tutorials/introductory/sample_plots.html
- https://matplotlib.org/3.2.1/gallery/index.html
- https://www.oreilly.com/library/view/python-data-science/9781491912126/ch04.html (Detailed Examples)

Line Plot

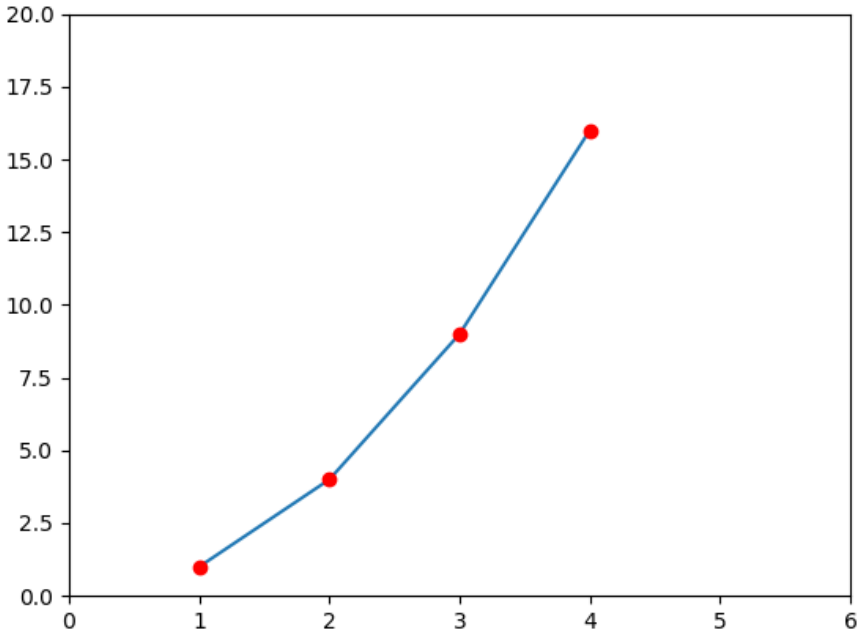
```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```



```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ko')
```

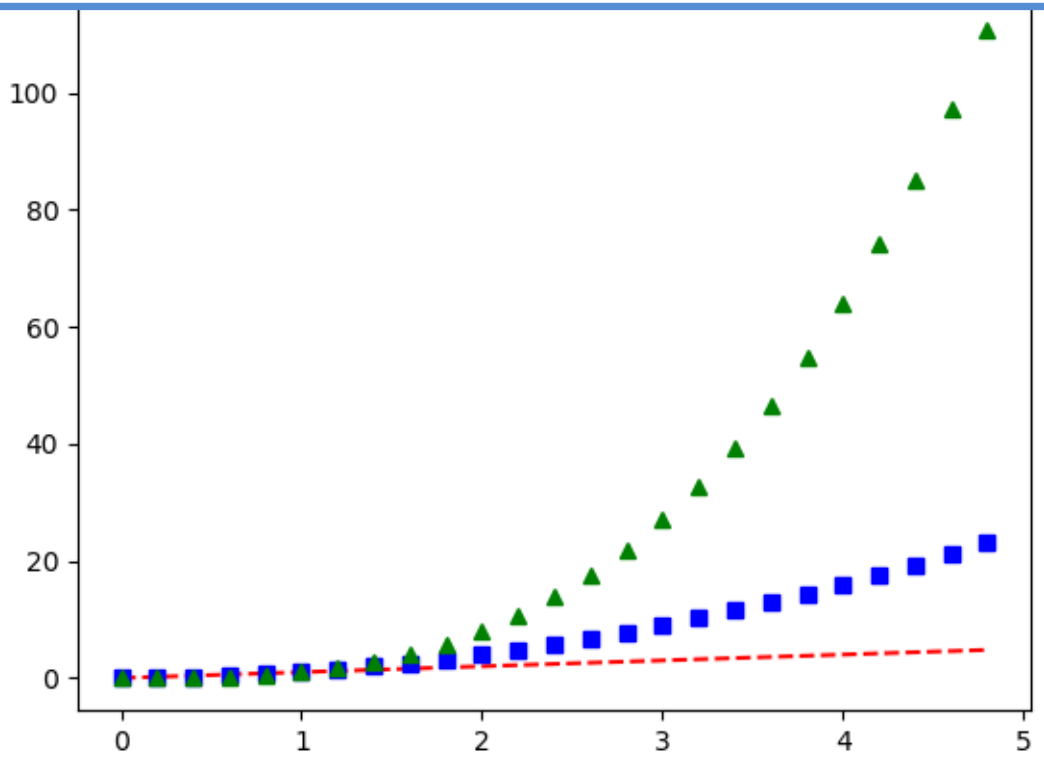
Line plot

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.plot([1, 2, 3, 4], [1, 4, 9, 16)
plt.axis([0, 6, 0, 20])
plt.show()
```



Line Plot

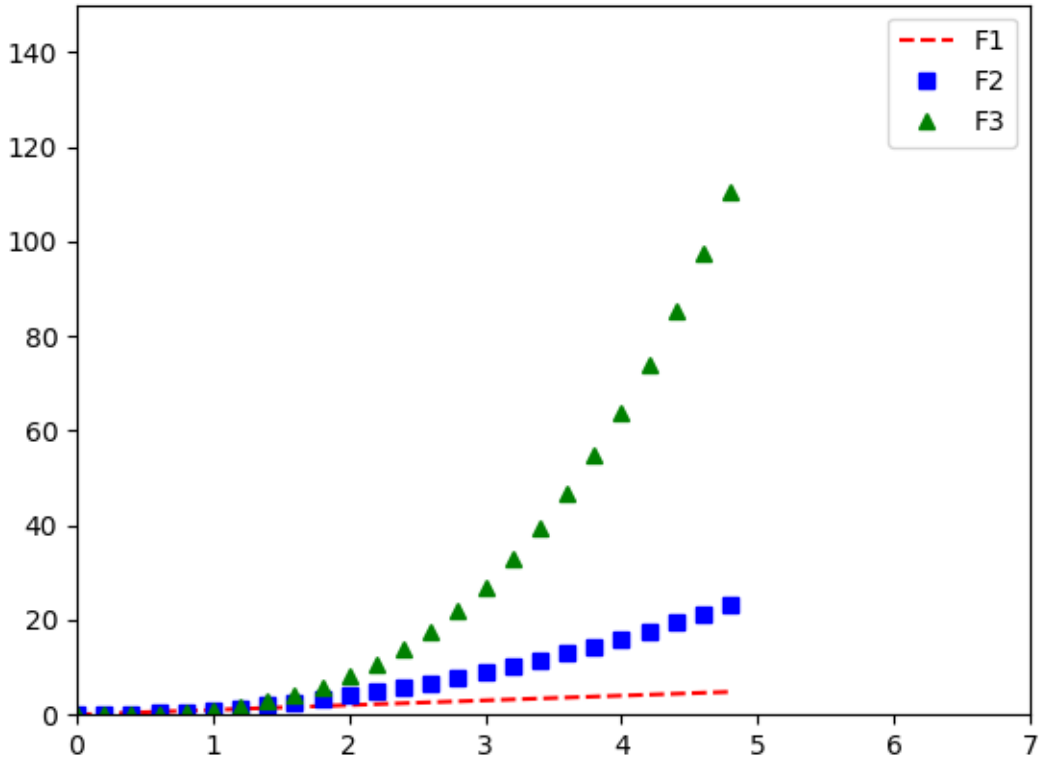
```
# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



Line Plot

```
import numpy as np

# evenly sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--',label='F1')
plt.plot(t, t**2,'bs',label='F2')
plt.plot(t, t**3, 'g^',label='F3')
plt.axis([0,7,0,150])
plt.legend()
plt.show()
```

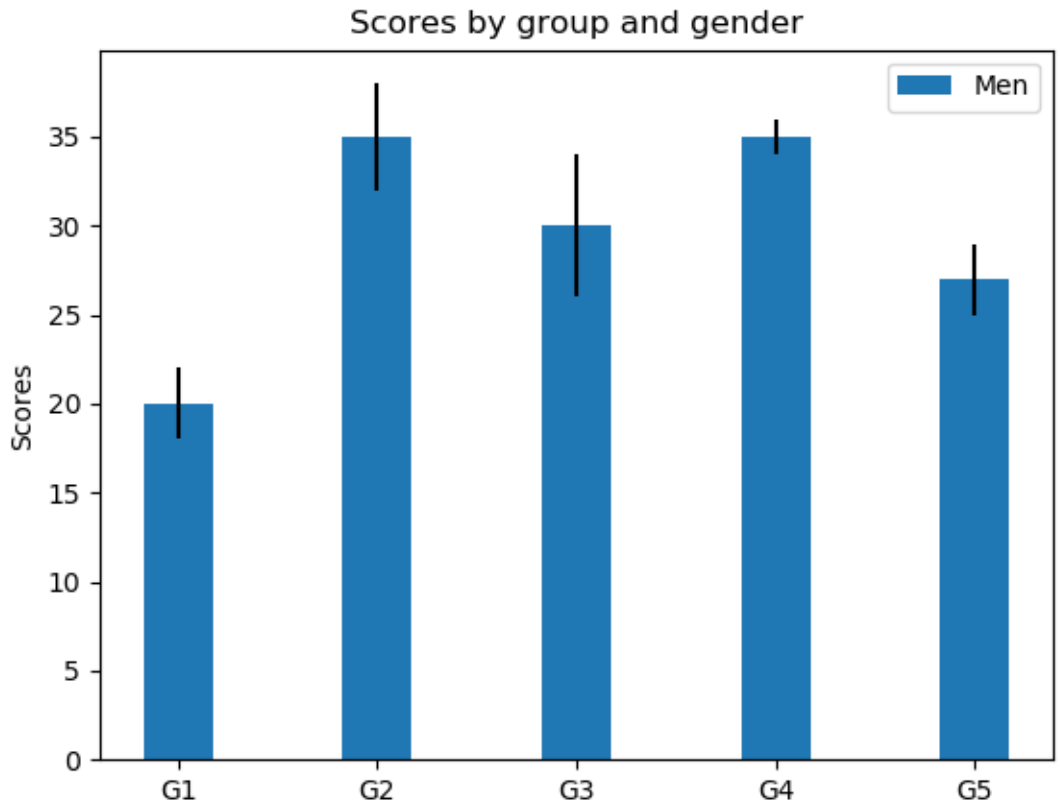
Line Plot



Bar plots

```
labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 35, 30, 35, 27]
men_std = [2, 3, 4, 1, 2]
width = 0.35       # the width of the bars: can also be len(x)
sequence
fig, ax = plt.subplots()
ax.bar(labels, men_means, width, yerr=men_std, label='Men')
ax.set_ylabel('Scores')
ax.set_title('Men Scores')
plt.show()
```

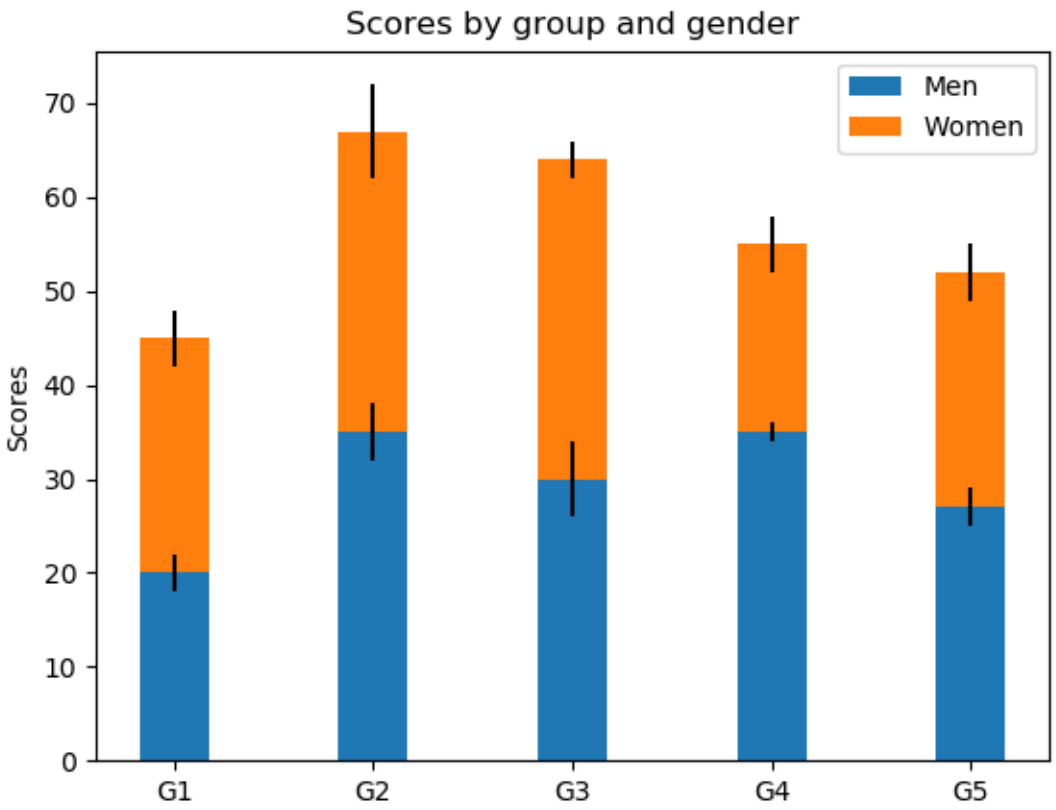
Bar plots


Scores by group and gender
Men
Scores
0
5
10
15
20
25
30
35
G1
G2
G3
G4
G5

Bar plots

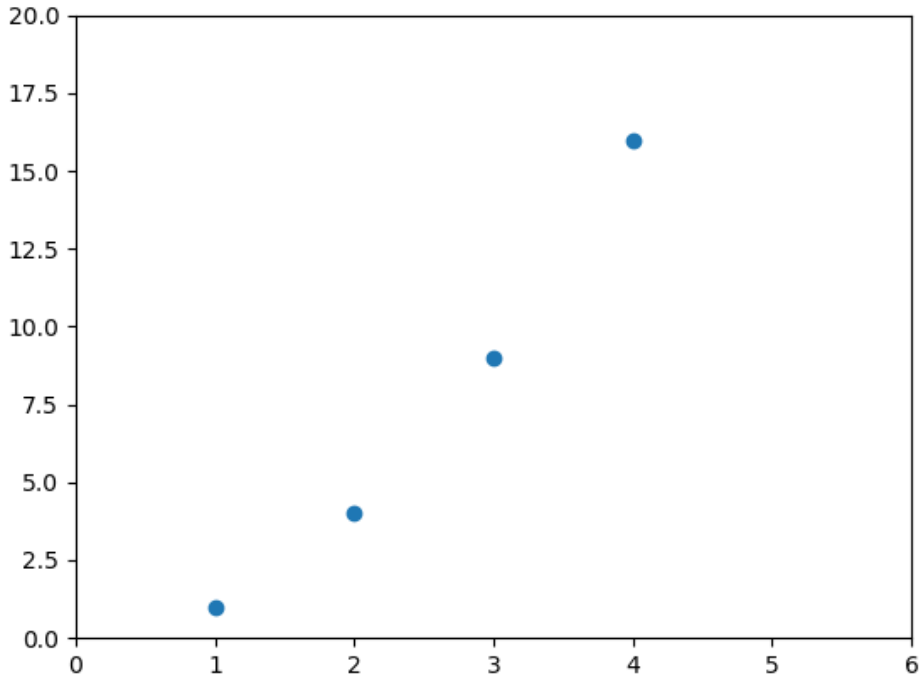
```
labels = ['G1', 'G2', 'G3', 'G4', 'G5']
men_means = [20, 35, 30, 35, 27]
women_means = [25, 32, 34, 20, 25]
men_std = [2, 3, 4, 1, 2]
women_std = [3, 5, 2, 3, 3]
width = 0.35       # the width of the bars: can also be len(x)
sequence

fig, ax = plt.subplots()

ax.bar(labels, men_means, width, yerr=men_std, label='Men')
ax.bar(labels, women_means, width, yerr=women_std,
bottom=men_means, label='Women')

ax.set_ylabel('Scores')
ax.set_title('Scores by group and gender')
ax.legend()
plt.show()
```

Bar Plots


Scores by group and gender
Men
Women
Scores
0
10
20
30
40
50
60
70
G1
G2
G3
G4
G5

Scatter plot

```
plt.scatter([1, 2, 3, 4], [1, 4, 9, 16])
plt.axis([0, 6, 0, 20])
plt.show()
```



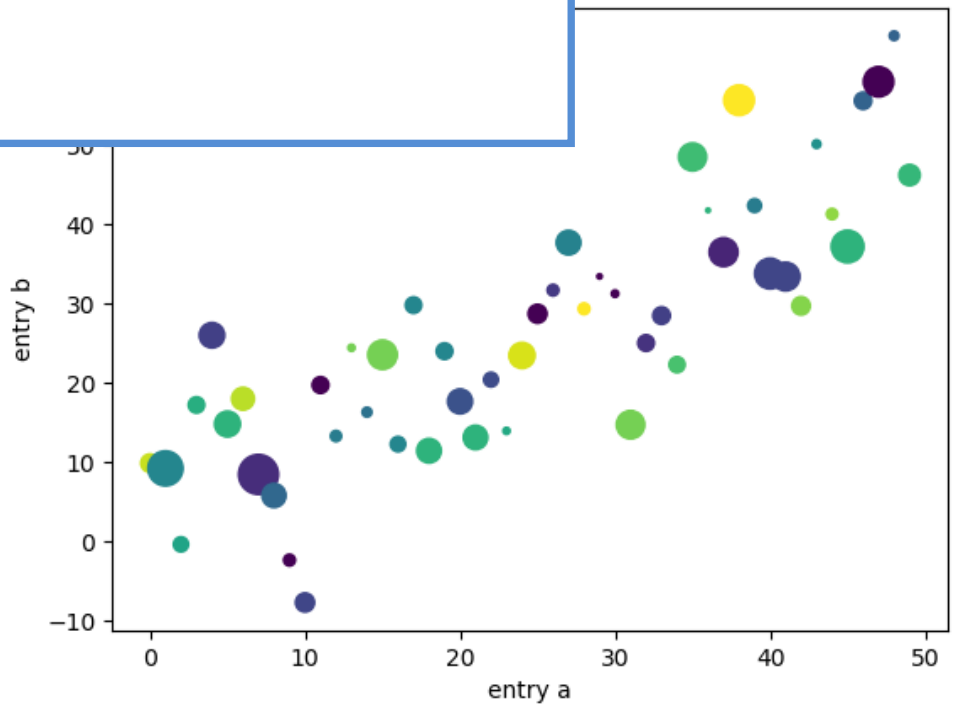
Scatter Plot

- We can vary different parameters:
  - Dot size → s
  - Color →c
  - Dot shape → marker

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
plt.savefig(r'C:\Users\mohammad\Desktop\scatter.png')
```

Scatter Plot

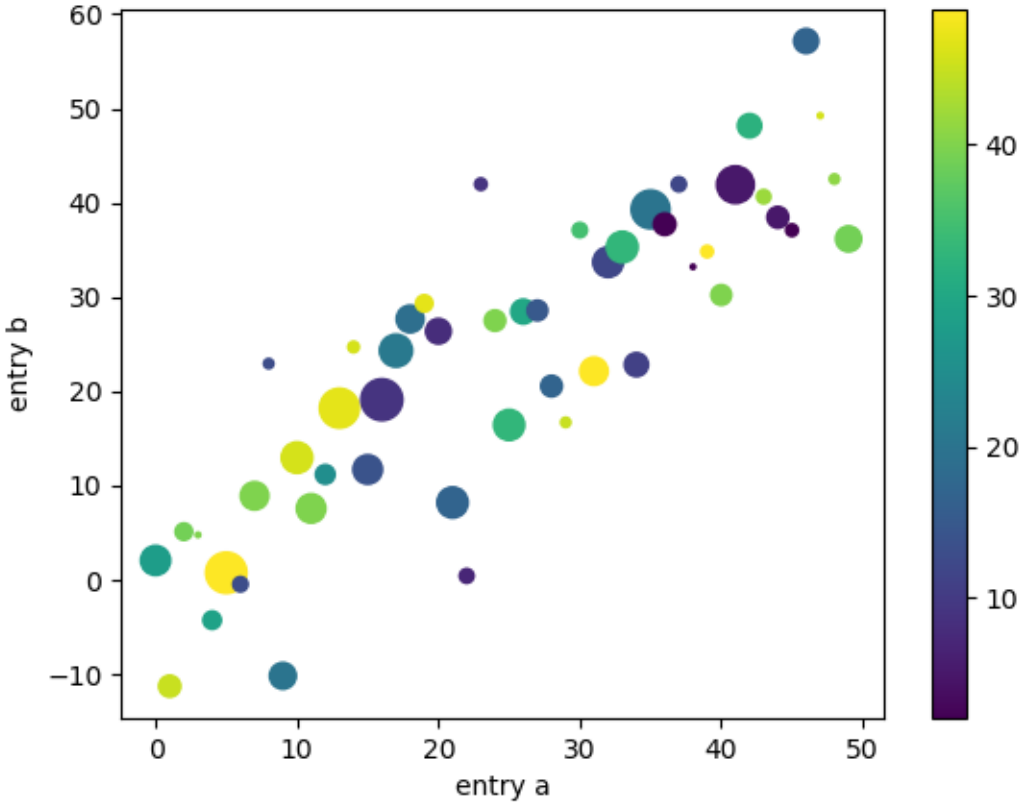
```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100
plt.scatter('a', 'b', c='c', s='d', data=data)
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.show()
```



Scatter Plot

```
data = {'a': np.arange(50),
        'c': np.random.randint(0, 50, 50),
        'd': np.random.randn(50)}
data['b'] = data['a'] + 10 * np.random.randn(50)
data['d'] = np.abs(data['d']) * 100

plt.scatter('a', 'b', c ='c' ,s='d',
data=data,cmap='viridis')
plt.xlabel('entry a')
plt.ylabel('entry b')
plt.colorbar();  # show color scale
plt.show()
```

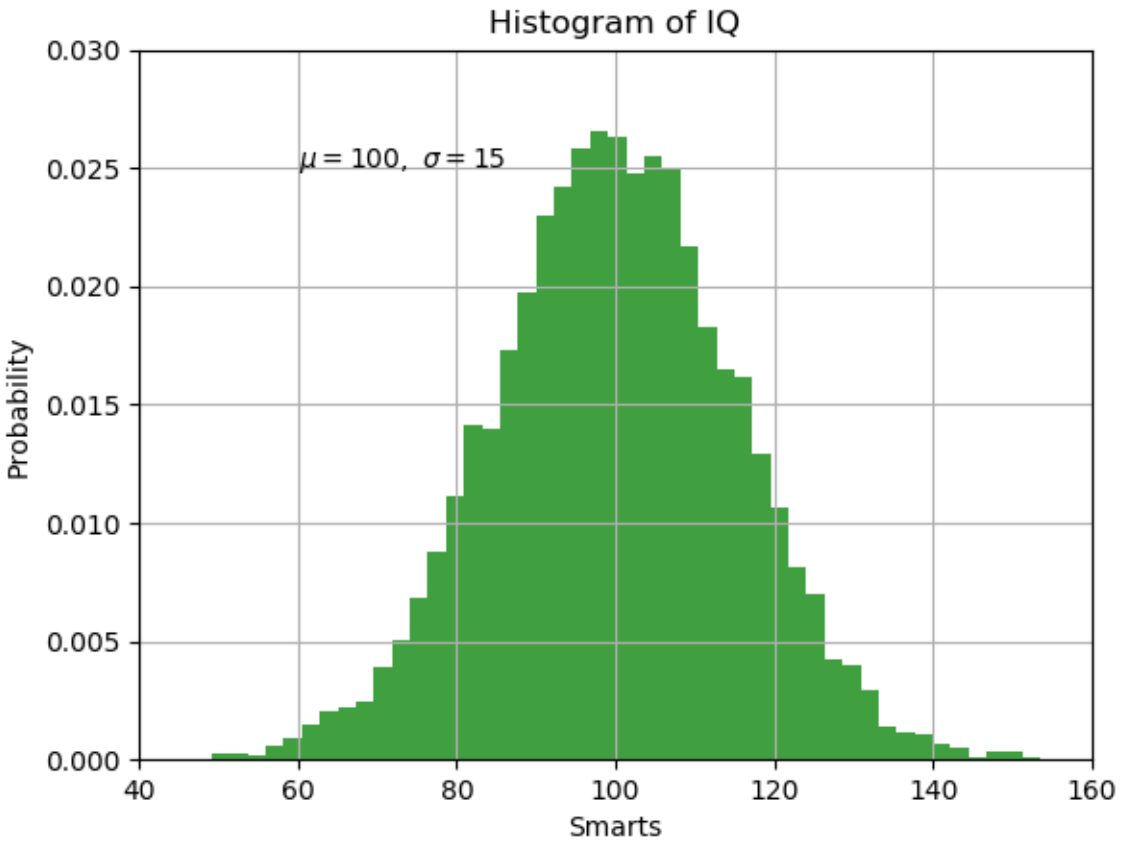
Scatter Plot



Histogram

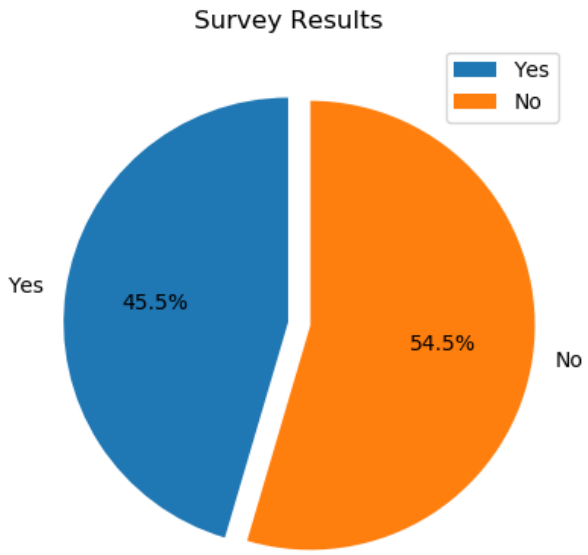
```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

# the histogram of the data
n, bins, patches = plt.hist(x, 50, density=1,
facecolor='g', alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60, .025, r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
#counts, bin_edges = np.histogram(x, bins=5)
```

Histogram


Histogram of IQ
$\mu = 100,\ \sigma = 15$
Probability
0.030
0.025
0.020
0.015
0.010
0.005
0.000
40
60
80
100
120
140
160
Smarts

Pie

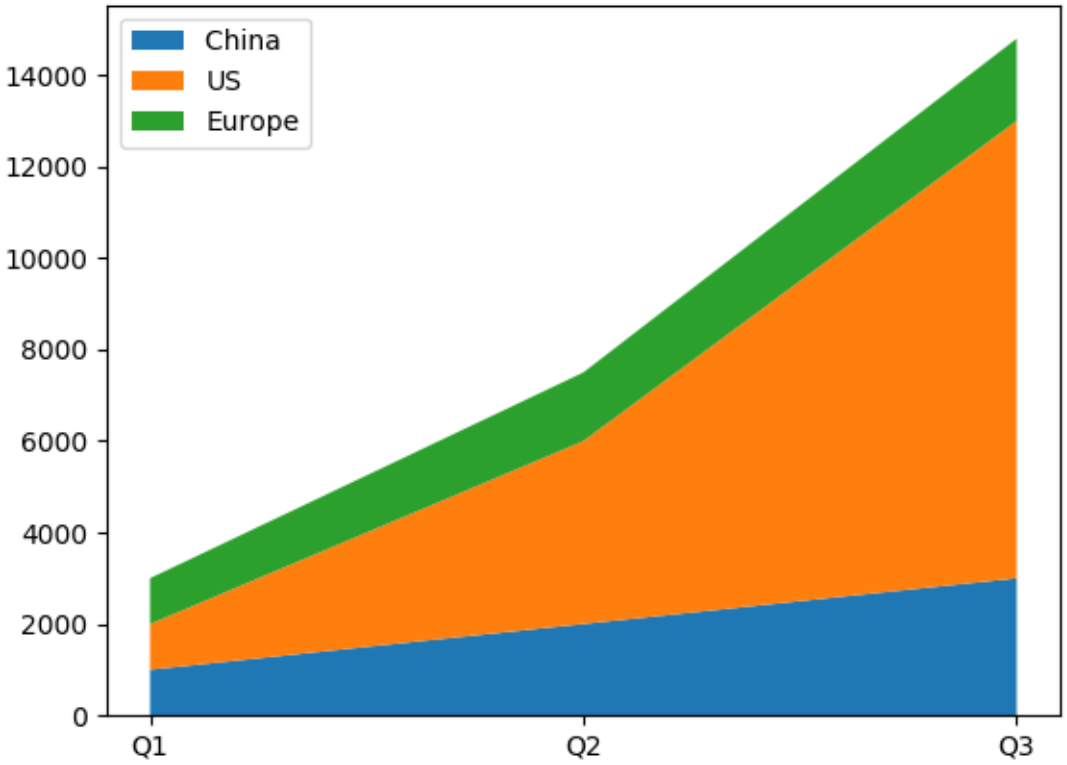
```
values  = [25000,30000]
Ans= ['Yes','No']
plt.pie(values,labels = Ans,autopct =
'%1.1f%%',startangle=90,explode =(0,.1))
plt.legend()
plt.title('Survey Results')
plt.show()
```



Stackplot

```
x = ['Q1', 'Q2', 'Q3']
y1 = [1000, 2000, 3000]
y2 = [1000,4000,10000]
y3 = [1000,1500,1800]

y = np.vstack([y1, y2, y3])

labels = ["China ", "US", "Europe"]

fig, ax = plt.subplots()
ax.stackplot(x, y1, y2, y3, labels=labels)
ax.legend(loc='upper left')
plt.show()
```

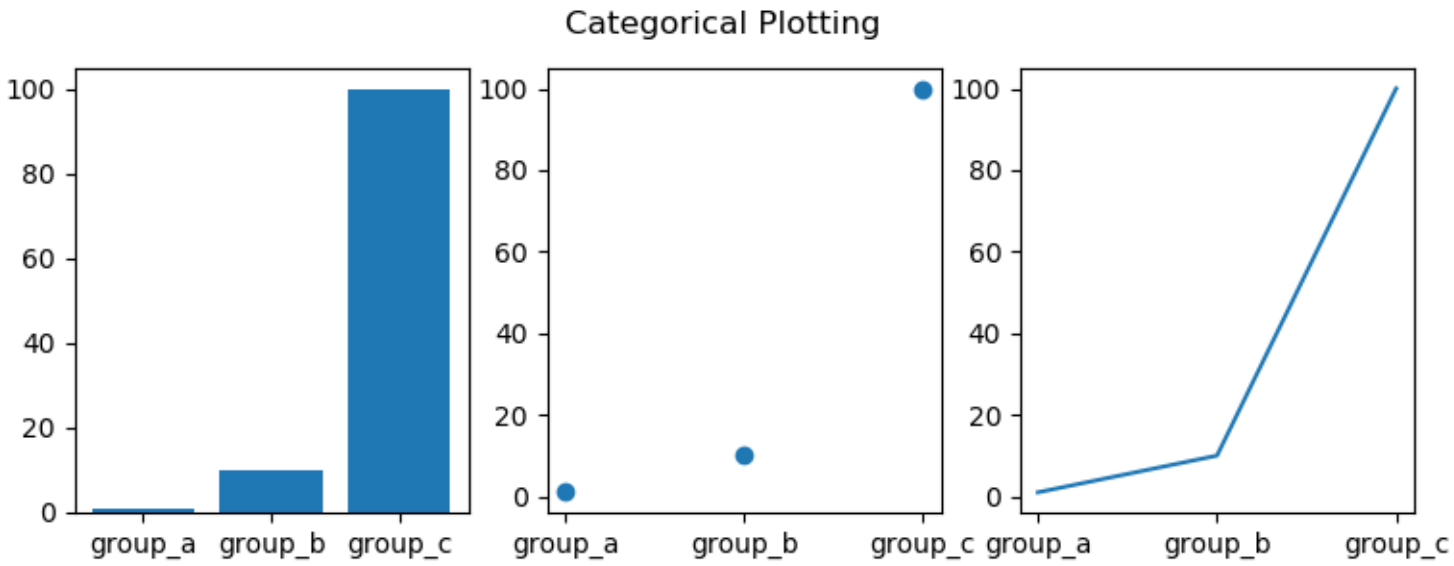
Stackplot


China
US
Europe
14000
12000
10000
8000
6000
4000
2000
0
Q1
Q2
Q3

Subplots

```
names = ['group_a', 'group_b', 'group_c']
values = [1, 10, 100]

plt.figure(figsize=(9, 3))

plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names, values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```

Subplots



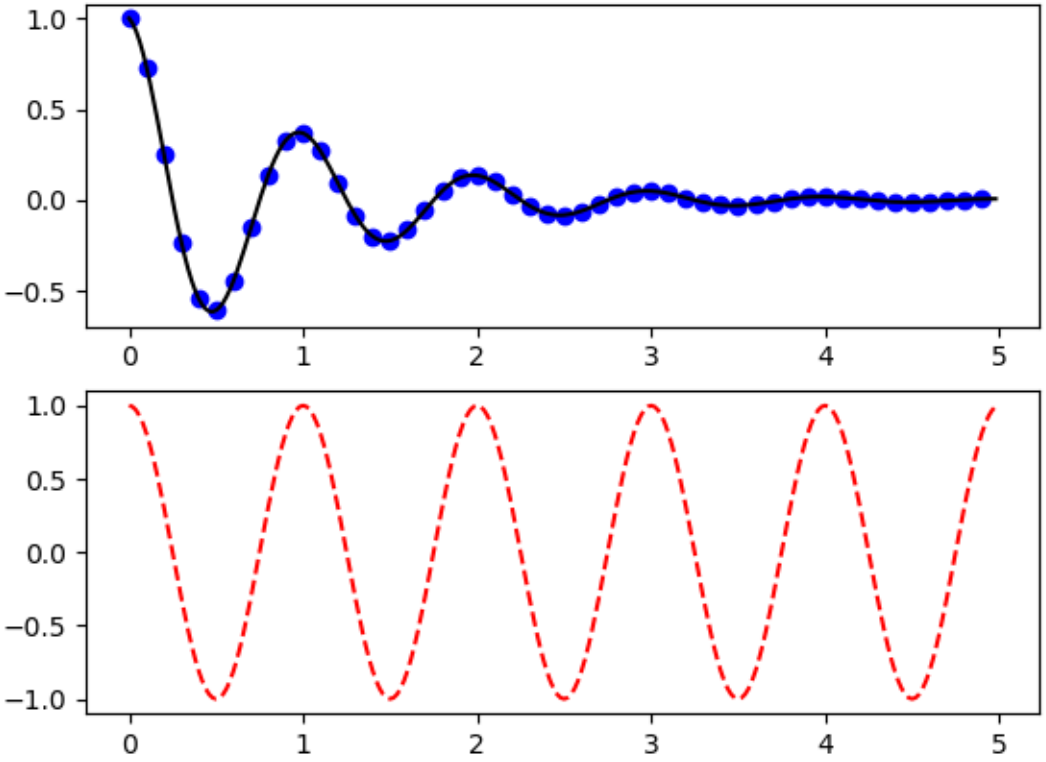
Subplots

```
def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'ob', t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

Subplots

```
fig, ax = plt.subplots(2)
ax[0].plot(t1, f(t1), 'ob', t2, f(t2), 'k')

ax[1].plot(t2, np.cos(2*np.pi*t2), 'r--')
plt.show()
```

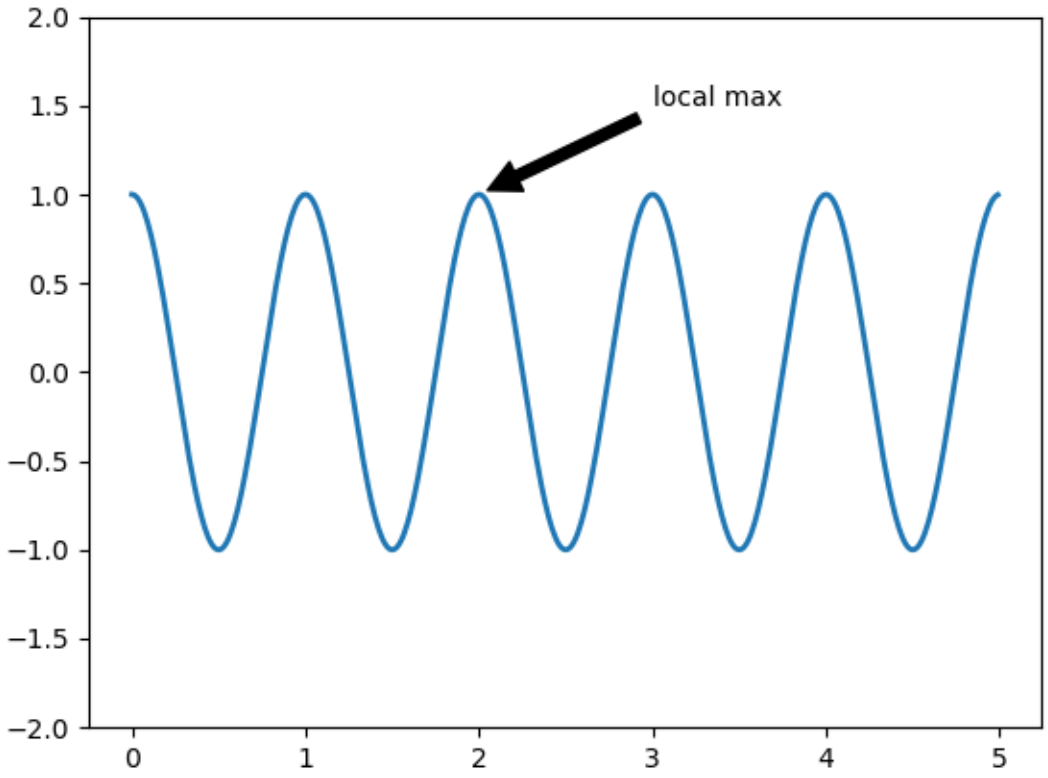
Subplots



Annotate

```
ax = plt.subplot(111)

t = np.arange(0.0, 5.0, 0.01)
s = np.cos(2*np.pi*t)
line, = plt.plot(t, s, lw=2)

plt.annotate('local max', xy=(2, 1), xytext=(3, 1.5),
             arrowprops=dict(facecolor='black',
shrink=0.05),
             )

plt.ylim(-2, 2)
plt.show()
```

Annotate



Seaborn

- if Matplotlib “tries to make easy things easy and hard things possible”, Seaborn tries to make a well-de fined set of hard things easy too.”
- Seaborn helps resolve the two major problems faced by Matplotlib
  - Default Matplotlib parameters
  - Working with data frames
- Starting point: https://www.tutorialspoint.com/seaborn/seaborn_tutorial.pdf