Abdallah Mahmoud
Facebook: https://www.facebook.com/abdallahriig
LinkedIn: https://www.linkedin.com/in/abdallahmahmud/

# Chapter 2: Python in Data Science

## Python

- Variables
- Comments
- Receiving Input
- Strings
- Arithmetic Operations
- If Statements
- Comparison operators
- While loops
- For loops
- Lists
- Tuples
- Dictionaries
- Functions
- Exceptions
- Classes
- Inheritance
- Modules
- Packages
- Python Standard Library
- Pypi

## What is Python?

Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. #### It is used for: - web development (server-side), - software development, - mathematics, - system scripting.

### *What can Python do?*

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

### *Why Python?*

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).

- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

### *Python Install*

Many PCs and Macs will have python already installed.

To check if you have python installed on a Windows PC, search in the start bar for Python or run the following on the Command Line (cmd.exe):

```
C:\Users\Your Name>python --version
```

To check if you have python installed on a Linux or Mac, then on linux open the command line or on Mac open the Terminal and type:

```
python --version
    or
python3 --version
```

### Write Hello World Program

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

### *Python Indentation*

Indentation refers to the spaces at the beginning of a code line.

Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.

Python uses **indentation to indicate a block of code**.

### *Example*

```
if 5 > 2:
 print("Five is greater than two!")
```

### *Error*

Python will give you an error if you skip the indentation

```
if 5 > 2:
print("Five is greater than two!")
```

### *Comments*

- Comments are ignored by compiler or interpreter.

- Comments can be used to explain Python code.
- Comments can be used to make the code more readable.
- Comments can be used to prevent execution when testing code.

### Creating a Comment

Comments starts with a #, and Python will ignore them:

*Example*

```
#This is a comment
print("Hello, World!")
```

*Variables*

Variables are **containers for storing data values**.

### Creating Variables

- Python has no command for declaring a variable.
- Python is loosey typed programming language so is not needed to specify data variable data type explicitly.
- A variable is *created the moment you first assign a value to it*.

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

### Good to know:

- We can specify variable data type explicity using data types.

```
x:int = 5
y:str = "Abdallah"
print(x)
print(y)
```

*Variable Names*

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). #### Rules for Python variables: - A variable name must start with a letter or the underscore character - A variable name cannot start with a number - A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _ ) - Variable names are case-sensitive (age, Age and AGE are three different variables) - A variable name cannot be any of the Python keywords. #### Example ###### Legal variable names:

```
myvar = "Abdallah"
my_var = "Abdallah"
_my_var = "Abdallah"
myVar = "Abdallah"
```

```
MYVAR = "Abdallah"
myvar2 = "Abdallah"
```

Illegal variable names:

```
2myvar = "Abdallah"
my-var = "Abdallah"
my var = "Abdallah"
```

### *Built-in Data Types*

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

Python has the following data types built-in by default, in these categories:

- Text Type: *str*
- Numeric Types: *int, float, complex*
- Sequence Types: *list, tuple, range*
- Mapping Type: *dict*
- Set Types: *set, frozenset*
- Boolean Type: *bool*
- Binary Types: *bytes, bytearray, memoryview*
- None Type: *NoneType*

### *Python Numbers*

There are three numeric types in Python:

- int
- float
- complex Variables of numeric types are created when you assign a value to them:

Example

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
```

To verify the type of any object in Python, use the type() function:

Example

```
print(type(x))
print(type(y))
print(type(z))
```

### *Type Conversion*

You can convert from one type to another with the int(), float(), and complex() methods:

*Example*

Convert from one type to another:

```
x = 1    # int
y = 2.8  # float
z = 1j   # complex
#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

## Variables

We use variables to temporarily store data in computer's memory.

```
price = 10
rating = 4.
course_name = 'Python for Beginners'
is_published = True
```

In the above example,

- **price** is an *integer* (a whole number without a decimal point)
- **rating** is a *float* (a number with a decimal point)
- **course_name** is a *string* (a sequence of characters)
- **is_published** is a *boolean*. Boolean values can be True or False.

## Comments

We use comments to add notes to our code. Good comments explain the hows and whys, not what the code does. That should be reflected in the code itself. Use comments to add reminders to yourself or other developers, or also explain your assumptions and the reasons you've written code in a certain way.

Abdallah Mahmoud
Facebook: https://www.facebook.com/abdallahriig
LinkedIn: https://www.linkedin.com/in/abdallahmahmud/

```
#This is a comment and it won't get executed.

#Our comments can be multiple lines.
```

## Receiving Input

We can receive input from the user by calling the **input()** function.

```
birth_year = int(input('Birth year: '))
```

The **input()** function always returns data as a string. So, we're converting the result into an integer by calling the built-in **int()** function.

## Strings

We can define strings using single *(' ')* or double *(" ")* quotes. To define a multi-line string, we surround our string with tripe quotes (""""). **'hello'** is the same as **"hello".** You can display a string literal with the print() function: #### Example

```
print("Hello")
print('Hello')
```

We can get individual characters in a string using square brackets **[]**.

```
course = 'Python for Beginners'
course[ 0 ] # returns the first character
course[ 1 ] # returns the second character
course[-1] # returns the first character from the end
course[-2] # returns the second character from the end
```

We can slice a string using a similar notation:

The above expression returns all the characters starting from the index position of 1 to 5 (but excluding 5). The result will be **ytho** If we leave out the start index, 0 will be assumed. If we leave out the end index, the length of the string will be assumed.

We can use formatted strings to dynamically insert values into our strings:

```
name = 'Abdallah'
message = f'Hi, my name is {name}' # this called **f-String** formatting

message.**upper() #** to convert to uppercase
message.**lower() #** to convert to lowercase
message.**title() #** to capitalize the first letter of every word
message.**find(** 'p' **) #** returns the index of the first occurrence of p
(or -1 if not found)
message **.replace** ('p', 'q')
```

To check if a string contains a character (or a sequence of characters), we use the **in** operator:

```
contains = ‘Python’ in course
```

*Multiline Strings*

You can assign a multiline string to a variable by using three quotes:

Example

You can use three double quotes:

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

## Arithmetic Operations

- +
- -
- *
- / # returns a float
- // # returns an int
- % # returns the remainder of division
- ** # exponentiation - x ** y = x to the power of y

*Augmented assignment operator:*

```
x = x + 10
x += 10
```

Operator precedence:

1. parenthesis
2. exponentiation
3. multiplication / division
4. addition / subtraction

## If Statements

```
if is_hot:
    print(“hot day”)
elif is_cold:
```

```
    print(“cold day”)
else:
    print(“beautiful day”)
```

## Logical operators:

if has_high_income **and** has_good_credit: if has_high_income **or** has_good_credit:

```
is_day = True
is_night = **not** is_day
```

## Comparison operators

```
a > b
a >= b (greater than or equal to)
a < b
a <= b
a == b (equals)
a != b (not equals)
```

## While loops

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

## For loops

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

- **range(5):** generates 0, 1, 2, 3, 4
- **range(1, 5):** generates 1, 2, 3, 4
- **range(1, 5, 2):** generates 1, 3

## Functions

We use functions to break up our code into small chunks. These chunks are
easier
to read, understand and maintain. If there are bugs, it’s easier to find bugs
in a
small chunk than the entire program. We can also re-use these chunks.
```c
def greet_user(name):
print(f”Hi {name}”)
```

greet_user(“Abdallah”) **Parameters** are placeholders for the data we can pass to functions. **Arguments** are the actual values we pass. We have two types of arguments:

- Positional arguments: their position (order) matters
```

- Keyword arguments: position doesn't matter - we prefix them with the parameter name.

*Two positional arguments*

greet_user("Abdallah", "Mahmoud")

*Keyword arguments*

calculate_total(order= 50 , shipping= 5 , tax=0.1) Our functions can return values. If we don't use the return statement, by default **None** is returned. None is an object that represents the absence of a value.

```
def square(number):
return number * number

result = square(2)
print(result) # prints 4
```

### Simple Project: Guess game

```
secret_number = 7
count = 0
while True:
  guess = int(input("Guess a number between 1 and 10:"))
  print(f"(attemp {count + 1} of 3)")
  count += 1
  if guess == secret_number:
    print("You got it!")
    break
  if count == 3:
    print("You ran out of guesses")
    break
  else:
    print("Sorry, that's not it.")
```

## Data Structure in Python

### Lists

```
numbers = [1, 2, 3, 4, 5]
numbers[ 0 ] # returns the first item
numbers[ 1 ] # returns the second item
numbers[-1] # returns the first item from the end
numbers[-2] # returns the second item from the end

numbers.append( 6 ) # adds 6 to the end
numbers.insert(0, 6) # adds 6 at index position of 0
numbers.remove( 6 ) # removes 6
numbers.pop() # removes the last item
```

```
numbers.clear() # removes all the items
numbers.index( 8 ) # returns the index of first occurrence of 8
numbers.sort() # sorts the list
numbers.reverse() # reverses the list
numbers.copy() # returns a copy of the list
```

### Tuples

They are like read-only lists. We use them to store a list of items. But once we define a tuple, **we cannot add or remove items or change the existing items**.

```
coordinates = ( 1 , 2 , 3 )
```

We can unpack a list or a tuple into separate variables:

```
x, y, z = coordinates # Unpacking tuples
```

### Dictionaries

We use dictionaries to store key/value pairs.

```
customer = {
“name”: “Abdallah Mahmoud”,
“age”: 25 ,
“is_verified”: True
}
```

We can use strings or numbers to define keys. They should be unique. We can use any types for the values.

```
customer[“name”] # returns “John Smith”
customer[“type”] # throws an error
customer.get(“type”, “silver”) # returns “silver”
customer[“name”] = “new name”
```

### Set

Sets are used to store multiple items in a single variable. A set is a collection which is **unordered, unchangeable**, and **unindexed**.

```
myset = {"apple", "banana", "cherry"}
print(myset)
```

#### *Duplicates Not Allowed*

Sets cannot have two items with the same value.

```
thisset = {"apple", "banana", "cherry", "apple"}
print(thisset)  # apple repeated twice but will not printed
```

Abdallah Mahmoud
Facebook: https://www.facebook.com/abdallahriig
LinkedIn: https://www.linkedin.com/in/abdallahmahmud/

## Exceptions

Exceptions are errors that crash our programs. They often happen because of bad input or programming errors. It's our job to anticipate and handle these exceptions to prevent our programs from cashing.

```
try:
    age = int(input('Age: '))
    income = 20000
    risk = income / age
    print(age)
except ValueError:
    print('Not a valid number')
except ZeroDivisionError:
    print('Age cannot be 0')
```

## OOPs in Python

### *What Is OOP in Python?*

Object-oriented programming is a programming paradigm that provides a means of structuring programs so that properties and behaviors are bundled into individual objects. ### Encapsulation ## Classes - Class is blue print or build block of OOPs. - We use classes to define new types.

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
        def move(self):
            print("move")
```

- When a **function is part of a class**, we refer to it as a **method**.
- *Classes define templates or blueprints for creating objects*.
- An **object** is an instance of a class.
- Every time we create a new instance, that instance follows the structure

we define using the class.

```
point1 = Point(10, 5)
point2 = Point(2, 4)
```

**init** is a special method called **constructor**. It gets called at the time of creating new objects. We use it to **initialize** our objects.

Abdallah Mahmoud
Facebook: https://www.facebook.com/abdallahriig
LinkedIn: https://www.linkedin.com/in/abdallahmahmud/

## Inheritance

Inheritance is a technique to remove code duplication. We can create a *base class* to define the common methods and then have other classes inherit these methods.

```
class Person:
    def walk(self):
        print("walk")

class Teacher(Person):
    def teach(self):
        print("teach")

teach = Teacher()
teach.walk() # inherited from Person
teach.teach() # defined in Teacher
```

## Polymorphism

The word **"polymorphism"** means *"many forms"*, and in programming it refers to methods/functions/operators with the same name that can be executed on many objects or classes.

Function Polymorphism An example of a Python function that can be used on different objects is the len() function.

```
    name = 'Abdallah'
    print(len(name)) # here len() function is used with string

    my_list = [1, 2, 3, 4]
    print(len(my_list)) # here len() function is used with list
```

**#Note: This means that len() function is behaved in two different way so we can say len is polymorphism methon/function.**

*Create Overloading Method/Function*

```
    def pow(number, power=1):
        return number ** power

    print(pow(2)) # We passed only one argument and will take 1 as default.
    print(pow(2,3)) # We passed two args and will calculated print 8 as
result.
```

**#Note: So now we created our own overloaded method.**

*Create Overriding Method/Function*

When child class re-implements parent class method is called **method overriding**.

```
   class Shape:
    def area(self):
      """Use pass keyword to satisfy the compiler"""
      pass

class Circle(Shape):
  PI = 3.14
  def __init__(self, radius) -> None:
    self.radius = radius
  def area(self):
    """Overriding the area method"""
    return Circle.PI * self.radius ** 2


circle = Circle(2)
print(circle.area())
```

## Abstraction

Abstraction in python is defined as a process of handling complexity by hiding unnecessary information from the user. Python support three types of access modifiers - public (no symbol): we can access anywhere - protected(_) single underscore: we can access within the class and child classes - private(__) double underscode: we can access only within the class

*Example:*

Public:

```
class Employee:

    def __init__(self, name, salary):
        """These are the attributes of the class are public"""
        self.name = name
        self.salary = salary

    def display(self):
        print(self.name, self.salary)


emp = Employee("Abdallah", 10000)
#we can access name and salary
```

```
print(emp.name)
print(emp.salary)
```

Protected:

```
class Employee:

    def __init__(self, name, salary):
        """These are the attributes of the class are protected"""
        self._name = name
        self._salary = salary

    def display(self):
        print(self._name, self._salary)


class HR(Employee):
  pass

emp = HR("Abdallah", 10000)
#we can access name and salary because HR is a child class of Employee
print(emp._name)
print(emp._salary)
```

Private:

```
class Employee:

    def __init__(self, name, salary):
        """These are the attributes of the class are protected"""
        self.__name = name
        self.__salary = salary

    def display(self):
        print(self.__name, self.__salary)


emp = Employee("Abdallah", 10000)
#we can't access name and salary because they are private
print(emp.__name)
print(emp.__salary)

#AttributeError: 'Employee' object has no attribute '__name'
```

Abdallah Mahmoud
Facebook: https://www.facebook.com/abdallahriig
LinkedIn: https://www.linkedin.com/in/abdallahmahmud/

### Abstract class in Python:

## Modules

A module is a file with some Python code. We use modules to break up our program into multiple files. This way, our code will be better organized. We won't have one gigantic file with a million lines of code in it! There are 2 ways to import modules: we can import the entire module, or specific objects in a module. #### Create a file named with utils.py Write following two functions

```
def factorial(number:int)->int:
    if number == 0:
        return 1
    return number * factorial(number - 1)

def fibonacci(number: int) -> int:
  if number == 0:
    return 0
  if number == 1:
    return 1
  return fibonacci(number - 1) + fibonacci(number - 2)
```

```
#importing the entire utils module
import utils
result = utils.factorial(5)
print(result)
# importing one function in the utils module
from uitls import factorial
reuslt =factorial(5)
print(result)
```

## Packages

A package is a directory with **init.py** in it. It can contain one or more modules.

```
#importing the entire sales module
from ecommerce import sales
sales.calc_shipping()
# importing one function in the sales module
from ecommerce.sales import calc_shipping
calc_shipping()
```

## Python Standard Library

Python comes with a huge library of modules for performing common tasks such as sending emails, working with date/time, generating random values, etc.

Random Module

```
import random

random.random() # returns a float between 0 to 1
random.randint(1, 6) # returns an int between 1 to 6

members = [‘Abdallah’, ‘Mahmoud’, ‘Ahmed’]
leader = random.choice(members) # randomly picks an item
```

## Pypi

Python Package Index (pypi.org) is a directory of Python packages published by Python developers around the world. We use **pip** to install or uninstall these packages. pip install pandas pip uninstall numpy