Python 101

Fu Swee Tee, Brian Loh & Mark Tee

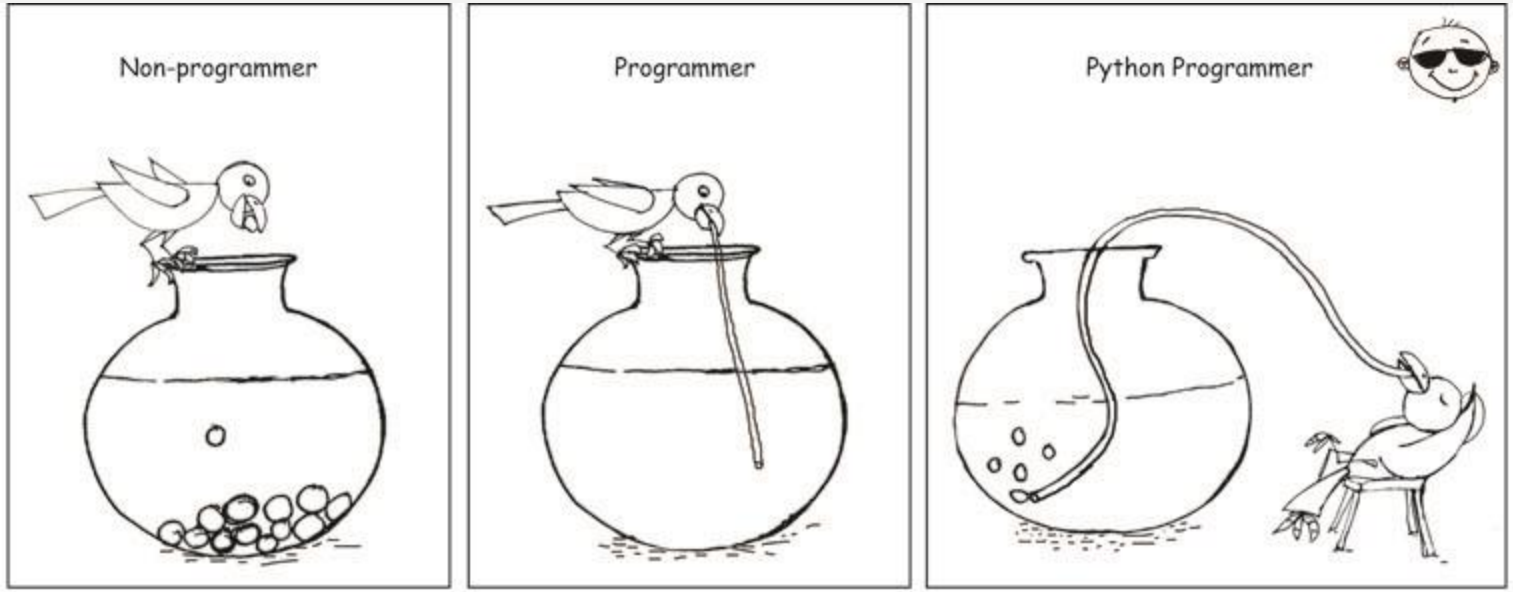
Vong Wan Tze

Yakub Sebastian

Hamada Rasheed Hassan Al-Absl

WHAT IS PYTHON?

- Python is a powerful, easy-to-read, high level programming language.
  - Commands read like English words, which makes it easy to learn.
  - It has simple easy-to-use syntax, making it a perfect language for someone trying to learn computer programming for the first time.



Top 10 Reasons to study Python

- Popularity & High Salary
- Simple & Easy to Learn
- Portable & extensible
- Web Development
- Artificial Intelligence
- Computer Graphics
- Testing Framework
- Big Data
- Machine Learning
- Data Science

CREATING A COMPUTER PROGRAM

Program Creation

- A person (programmer) writes a computer program (series of instructions).
- The program is written and saved using a text editor.
- The instructions in the programming language are high level (look much like a human language).

Translation

- A special computer program (translator) translates the program written by the programmer into the *only* form that the computer can understand (**machine language/binary**)

Execution

- The machine language instructions can now be directly executed by the computer.

HOW TO INSTALL AND BEGIN PROGRAMMING IN PYTHON?

- **Anaconda** is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analysis, games and etc.), that aim to simplify package management and deployment.
- Download Anaconda Distribution: https://www.anaconda.com/distribution/

ANACONDA®

THE JUPYTER NOTEBOOK

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Language of choice

Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala.

Interactive output

Your code can produce rich, interactive output: HTML, images, videos, LaTeX, and custom MIME types.

Share notebooks

Notebooks can be shared with others using email, Dropbox, GitHub and the Jupyter Notebook Viewer.

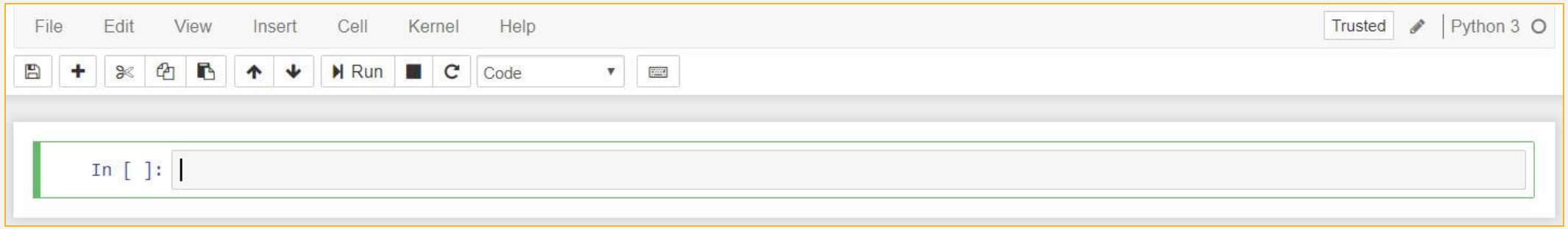
Big data integration

Leverage big data tools, such as Apache Spark, from Python, R and Scala. Explore that same data with pandas, scikit-learn, ggplot2, TensorFlow.

A LITTLE NOTE ABOUT JUPYTER NOTEBOOK…

Create a new notebook

The notebook interface



LET'S TRY THIS OUT!

```
In [1]: print("Hello World")

        Hello World
```

```
In [3]: print("Enter a number:")
        number = input()
        print("The number you entered is " + number)

        Enter a number:
        10
        The number you entered is 10
```

The input() function reads a line entered on a console by an input device such as a keyboard and convert it into a string and returns it.

1) Data Types

Lists

3) Functions

Function

2) Decision Making & Loop

If...Else, While Loop, For Loop

4) Modules & Files

Modules, Files, Directory

Data Types

DATA TYPES

Fundamental Data Types	Example
int	x = 50
float	x = 50.5
bool	x = True
str	x = "Hey"
list	x = ["Bing","Bang","Boom"]

ARITHMETIC OPERATORS

Operator	**Description**	**Example** (Let x = 12, y = 6)
+	Addition	x + y = 18
-	Subtraction	x - y = 6
*	Multiplication	x * y = 72
/	Division	x / y = 2
%	Modulus	x % y = 0

#PRACTICE 1: FUNDAMENTAL DATA TYPES

```
print(3 + 3)        #  6
print(3 - 3)        #  0
print(3 * 3)        #  9
print(3 / 3)        #  1.0
print(3 % 3)        #  0
print("good")       #  good
```

```
print(type(3 + 3))        #  <class 'int'>
print(type(3 - 3))        #  <class 'int'>
print(type(3 * 3))        #  <class 'int'>
print(type(3 / 3))        #  <class ‘float'>
print(type(3 % 3))        #  <class 'int'>
print(type("good"))       #  <class 'str'>
```

DATA TYPES

Fundamental Data Types	Example
int	x = 50
float	x = 50.5
bool	x = True
str	x = "Hey"
list	x = ["Bing", "Bang", "Boom"]

WHAT IS A LIST?

A list is an ordered data structure with elements enclosed within square brackets and separated by commas.

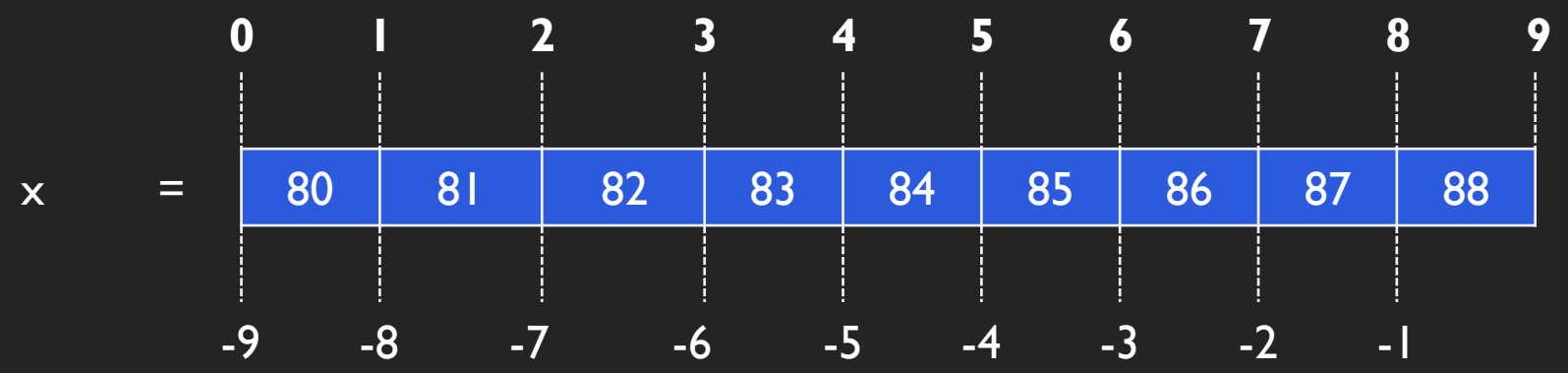
Single Data Type:

```
x = ["Bing","Bang","Boom"]

x = [80,88,89,98,99,100]
```

Mixed Data Types:

```
x = [80,"Bing",88,"Bang",True]
```

LIST INDEXING



Create a list

```
x = [80, 81, 82, 83, 84, 85, 86, 87, 88]        # [80, 81, 82, 83, 84, 85, 86, 87, 88]
print(x)
```

Indexing

```
print(x[0])        # 80
print(x[3])        # 83
print(x[-1])       # 88
print(x[-5])       # 84
print(x[9])        # IndexError: list index out of range
```

LIST SLICING

List Slicing is the method of splitting a list into a subset.

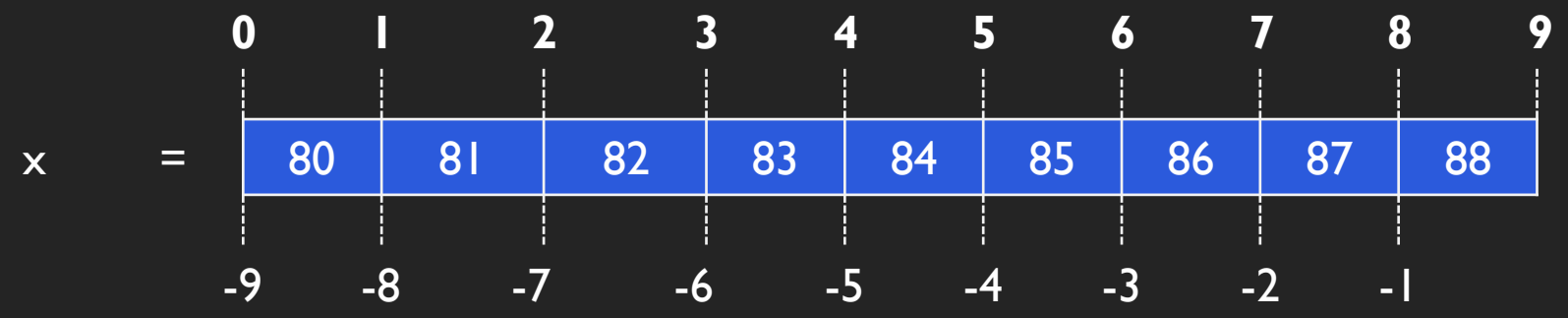
Syntax:

```
Listname[start:stop:steps]
```

Example:

```
x = [80, 81, 82, 83, 84, 85, 86, 87, 88]
print(x[1:3])      #[81, 82]
```

#PRACTICE 2: LIST SLICING



Slicing

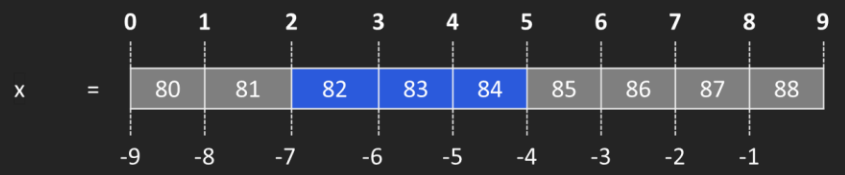
```
print(x[:])          # [80, 81, 82, 83, 84, 85, 86, 87, 88]

print(x[0:3])        # [80, 81, 82]
print(x[:3])         # [80, 81, 82]

print(x[2:])         # [82, 83, 84, 85, 86, 87, 88]
print(x[-7:-2])      # [82, 83, 84, 85, 86]
print(x[2:-5])       # [82, 83]
```

#PRACTICE 2: LIST SLICING

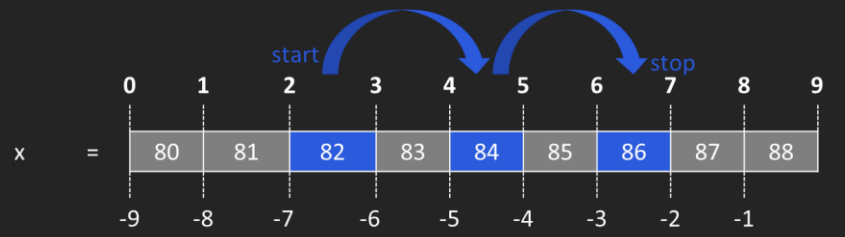
Slicing

```
print(x[2:5:1])      # [82, 83, 84]
print(x[2:5])        # [82, 83, 84]
```

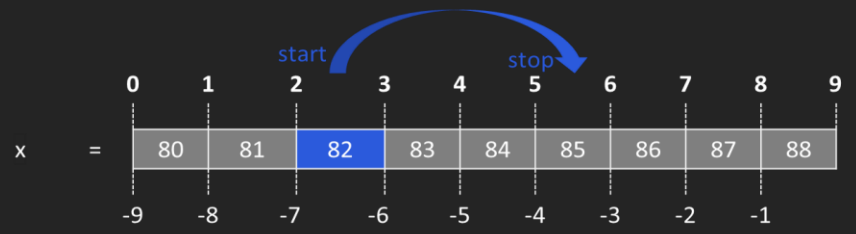
The <step> parameter is optional and by default 1.




```
print(x[2:7:2])      # [82, 84, 86]
```




```
print(x[2:5:3])      # [82]
```



#PRACTICE 2: LIST SLICING

Reverse a List	print(x[::-1])	# [88,87,86,85,84,83,82,81,80]
Modify List Values	x[2:4] = ["Bing","Bang"] print(x)	# [80,81,"Bing","Bang",84,85,86,87,88]
Insert multiple list items at the start	x[:0] = ["Durian","Cake"] print(x)	# ["Durian","Cake",80,81,"Bing","Bang",84,85,86,87,88]
Insert multiple list items at the end	x[len(x):] = ["Nice", "Yummy"] print(x)	# ["Durian","Cake",80,81,"Bing","Bang",84,85,86,87,88,"Nice","Yummy"]

#PRACTICE 2: LIST SLICING

Delete multiple list items

```
x[0:2] = []
print(x)
```

[80, 81, **"Bing", "Bang",** 84, 85, 86, 87, 88, **"Nice", "Yummy"**]

```
del x[2:4]
print(x)
```

[80, 81, 84, 85, 86, 87, 88, **"Nice", "Yummy"**]

Duplicate a list

```
print(x*2)
```

[80, 81, 84, 85, 86, 87, 88, "Nice", "Yummy", 80, 81, 84, 85, 86, 87, 88, "Nice", "Yummy"]

#EXERCISE 1: TYPE CONVERSION

Create a program to calculate age:

```
birthyear = input("What year were you born?")
age = 2019-birthyear
print("Your age is " + age)
```

```
What year were you born?1984

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-fe80d77d56ee> in <module>
      1 birthyear = input("What year were you born?")
----> 2 age = 2019-birthyear
      3 print("Your age is " + age)

TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

How to solve this TypeError?

#EXERCISE 1: TYPE CONVERSION

Create a program to calculate age:

```
birthyear = input("What year were you born?")
age = 2019 - int(birthyear)
print("Your age is " + str(age))
```

```
What year were you born?2000
Your age is 19
```

#EXERCISE 2: PASSWORD CHECKER

Create a program to check the length of passwords.

Sample of output:

```
What is your username? Melissa
What is your password? mel2019
Melissa, your password, *******, is 7 letters long
```

How to create this program?

#EXERCISE 2: PASSWORD CHECKER

Create a program to check the length of passwords.

```
username = input("What is your username? ")
password = input("What is your password? ")
password_length = len(password)
hidden_password = "*" * password_length

print(username + ", your password, " + hidden_password + ", is " + str(password_length) + " letters long")
```

Decision Making & Loop

If…Else, While Loop, For Loop

SIGNIFICANCE OF INDENTATION

Unlike JavaScript, indentation has a special significance in Python.

Indentation is used to define a block of code.

Contiguous statements that are indented to the same level are considered as part of the same block.

```
is_old = True

if is_old:
    print('You are old enough to drive!')
else:
    print('You are not of age!')
```

```
You are old enough to drive!
```

```
is_old = True

if is_old:
print('You are old enough to drive!')
else:
print('You are not of age!')
```

```
  File "<ipython-input-4-e87e4b241ca7>", line 4
    print('You are old enough to drive!')
        ^
IndentationError: expected an indented block
```

```
is_old = False

if is_old:
    print('You are old enough to drive!')
else:
    print('You are not of age!')
```

```
You are not of age!
```

IF…ELSE…ELIF STATEMENT

The statement is used to execute some statements only when some condition holds.

```
if condition:
    statement
    statement
        ...
elif condition:
    statement
    statement
        ...
else:
    statement
    statement
        ...
following_statement
```

First condition

This is executed if the first condition is true

New condition

A new condition to test if previous condition isn't true

False branch

This is executed if none of the conditions are true

COMPARISON OPERATORS

Operator	Description	Example
==	Equals	If x == y
!=	Not equals	If x != y
>	Greater than	If x > y
>=	Greater than or equal to	If x >= y
<	Less than	If x < y
<=	Less than or equal to	If x <= y

#PRACTICE 3: IF…ELIF…ELSE

In Python, any non-zero value or nonempty container is considered TRUE, whereas Zero, None, and empty container is considered FALSE

```
x, y = 10, 5
if (x + y):
    print('true')
else:
    print('false')
```

```
true
```

```
colour = ['red', 'blue']
if colour:
    print('true')
else:
    print('false')
```

```
true
```

```
x, y = 5, 5
if (x - y):
    print('true')
else:
    print('false')
```

```
false
```

```
colour = []
if colour:
    print('true')
else:
    print('false')
```

```
false
```

#PRACTICE 3: IF…ELIF…ELSE

Basic Example:

```
x, y = 5, 10
if x > y:
    print('x is greater')
elif x < y:
    print('y is greater')
else:
    print('x and y are equal')
```

```
y is greater
```

Substitute for **Switch Case:**

```
choice = 3

if choice == 1:
    print('apple')
elif choice == 2:
    print('orange')
elif choice == 3:
    print('durian')
else:
    print('I don eat fruits')
```

```
durian
```

#PRACTICE 3: IF…ELIF…ELSE

Multiple Conditions:

Use logical operators (and, or, not) to join two or more conditions into a single if statement.

```
x, y, z = 7, 4, 2
if x > y and x > z:
    print('x is greater')
```

```
x is greater
```

```
x, y, z = 7, 4, 9
if x > y or x > z:
    print('x is greater than y or z')
```

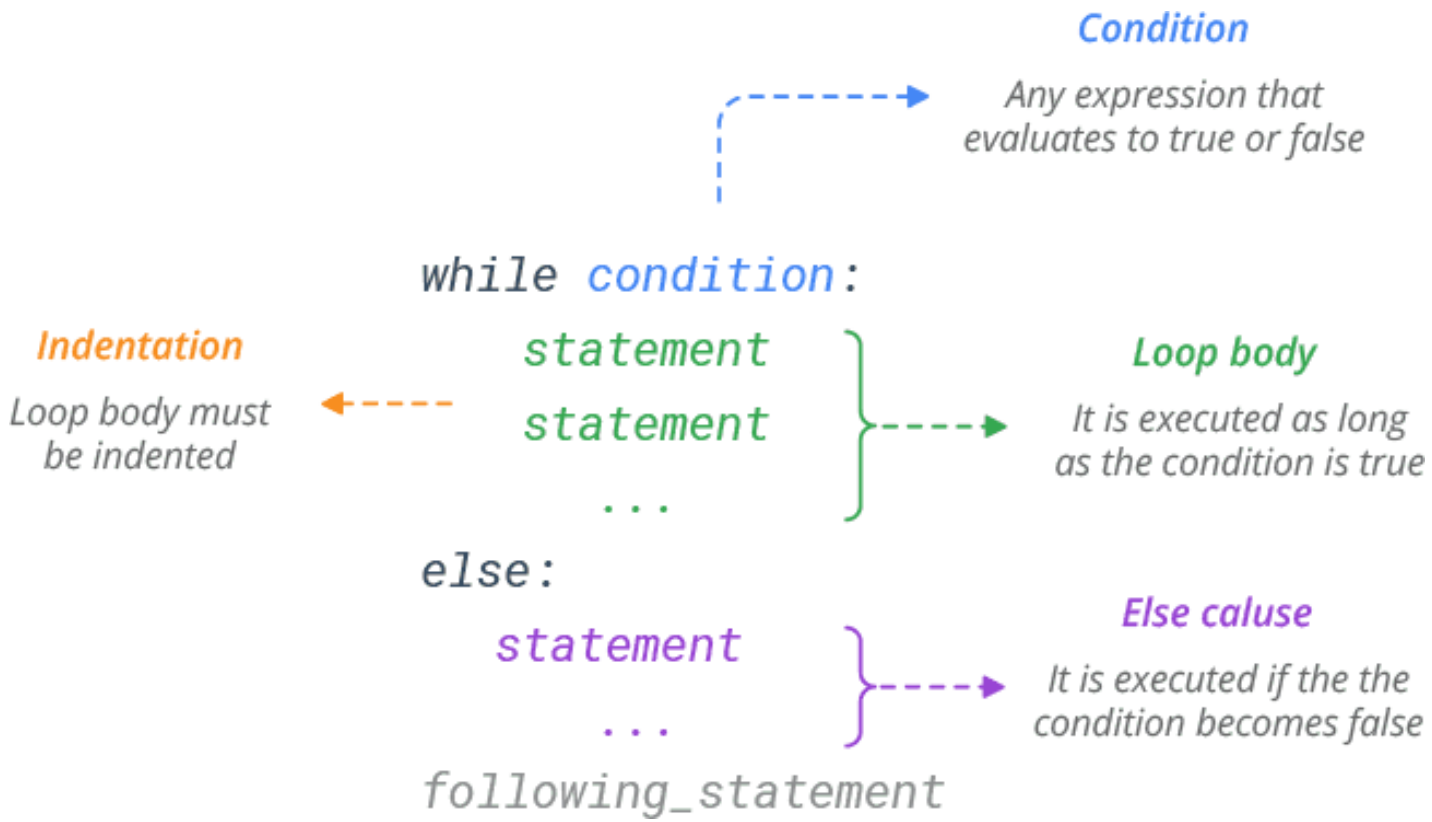
```
x is greater than y or z
```

```
x, y = 7, 5
if not x < y:
    print('x is greater')
```

```
x is greater
```

WHILE LOOP STATEMENT

The statement is used to perform a task indefinitely, until a particular condition is met.



#PRACTICE 4: WHILE LOOP

```python
x = 5
y = 2

while y <= x:
    print("Loop is easy")
    y += 1
else:
    print("Loop is terminated")
```

```
Loop is easy
Loop is easy
Loop is easy
Loop is easy
Loop is terminated
```

Iteration	Variable	y <= x	Body of Loop
1st	x = 5 **y = 2**	True	Loop is easy
2nd	x = 5 **y = 3**	True	Loop is easy
3rd	x = 5 **y = 4**	True	Loop is easy
4th	x = 5 **y = 5**	True	Loop is easy
5th	x = 5 **y = 6**	False	Loop is terminated

#PRACTICE 4: WHILE LOOP

```python
x = 20

while x:
    print(x)
    x -= 2
    if x == 10:
        break
```

```
20
18
16
14
12
10
```

Exit the loop

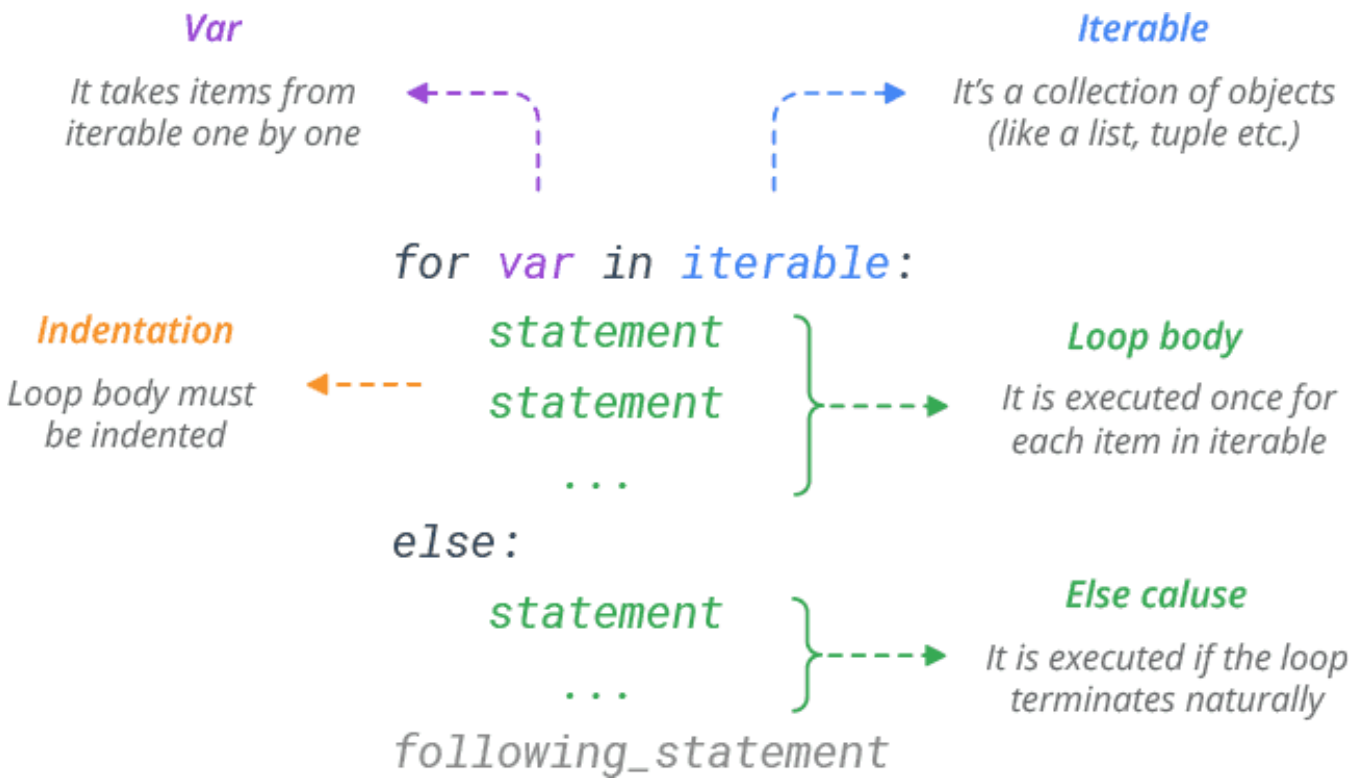
```python
x = 20

while x:
    x -= 2
    if x == 10:
        continue
    print(x)
print("Exit the Loop")
```

```
18
16
14
12
8
6
4
2
0
Exit the Loop
```

Skip the iteration of a loop and continue with the next iteration

FOR LOOP STATEMENT

The statement is used to iterate over the items of any iterable (list, tuple, dictionary, set or string).



#PRACTICE 5: FOR LOOP

```python
name = ["John", "Johny", "Jonathon"]

for x in name:
    print(x)
else:
    print("Done!")
```

```
John
Johny
Jonathon
Done!
```

Iteration	Variable	Body of Loop
1st	"John"	"John" is printed
2nd	"Johny"	"Johny" is printed
3rd	"Jonathon"	"Johnathon" is printed

#PRACTICE 5: FOR LOOP

```
for x in range (5, 10):
    print(x)
```

```
5
6
7
8
9
```

```
for x in range (5, 10, 2):
    print(x)
```

```
5
7
9
```

`range(start,stop,step)`

Parameter	Condition	Description
start	Optional	A number specifying start position. Default is 0.
stop	Required	A number specifying end position.
step	Optional	A number specifying the increment. Default is 1.

#PRACTICE 5: FOR LOOP

- Use zip(iterable) function to loop through multiple lists at once.

```
name = ["John", "Johny", "Jonathon"]
age = [25, 30, 35]
for x, y in zip(name, age):
    print(x, y)
```

```
John 25
Johny 30
Jonathon 35
```

#EXERCISE 3: DUPLICATE CHECKER

Create a program to check for duplicates in a list of fruits:

fruits = ["apple","orange","mango","banana","durian","apple","mango","banana","mango", "apple"]

Use For Loop and If to check for duplicates.

Output:

['apple','mango','banana']

#EXERCISE 3: DUPLICATE CHECKER

Create a program to check for duplicates in a list of fruits:

```
fruits = ["apple", "orange", "mango", "banana", "durian", "apple", "mango", "banana", "mango", "apple"]

duplicates = []

for value in fruits:
    if fruits.count(value) > 1:
        if value not in duplicates:
            duplicates.append(value)

print(duplicates)
```

```
['apple', 'mango', 'banana']
```

#EXERCISE 4: CHRISTMAS TREE

Create a program to display a Christmas Tree:

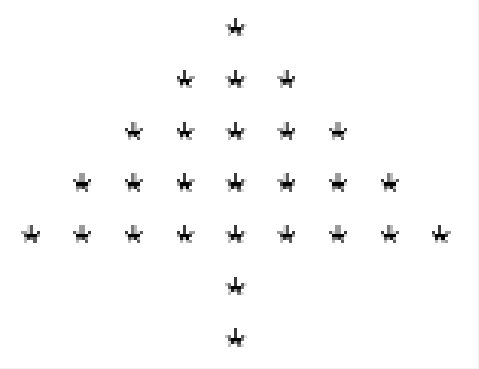
```
picture = [
    [0,0,0,1,0,0,0],
    [0,0,1,1,1,0,0],
    [0,1,1,1,1,1,0],
    [1,1,1,1,1,1,1],
    [0,0,0,1,0,0,0],
    [0,0,0,1,0,0,0],
    [0,0,0,1,0,0,0]
]
```

Iterate over the picture:

- **If 0 → print empty space ' '**
- **If 1 → print asterisk ***

```
print(objects,sep,end,file,flush)
```

end	Optional	A string to print at the end. Default is a newline '\n'.



#EXERCISE 4: CHRISTMAS TREE

Create a program to display a Christmas Tree:

```
picture = [
    [0,0,0,0,1,0,0,0,0],
    [0,0,0,1,1,1,0,0,0],
    [0,0,1,1,1,1,1,0,0],
    [0,1,1,1,1,1,1,1,0],
    [1,1,1,1,1,1,1,1,1],
    [0,0,0,0,1,0,0,0,0],
    [0,0,0,0,1,0,0,0,0]
]

for row in picture:
    for pixel in row:
        if (pixel == 1):
            print('*', end=' ')
        else:
            print(' ', end=' ')
    print(' ')
```

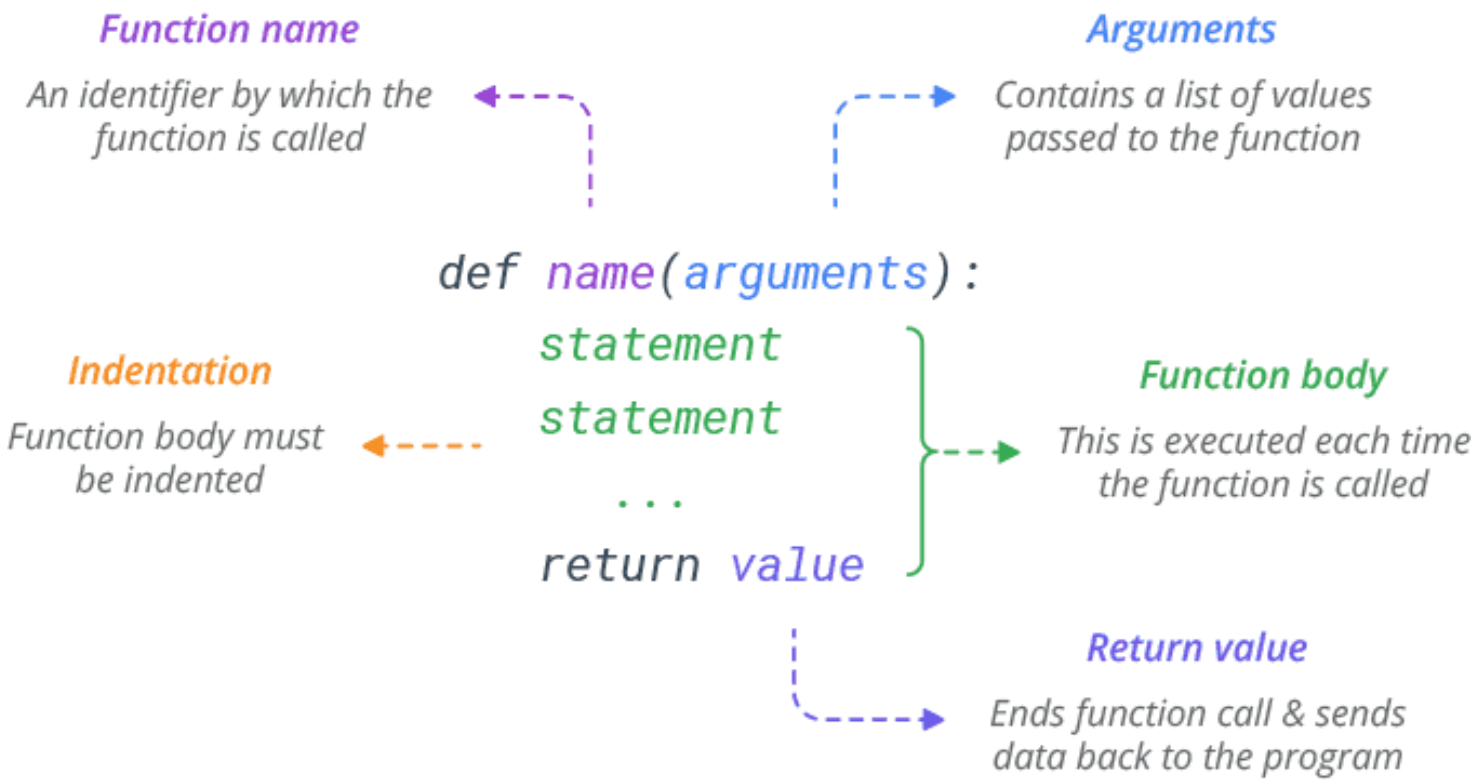
```
        *
      * * *
    * * * * *
  * * * * * * *
* * * * * * * * *
        *
        *
```

Function

WHAT IS A FUNCTION?

A function is a group of related statements that perform a specific task.

Syntax:



CREATE & CALL A FUNCTION

Create a Function:

```
def greet(name):
    print("Hello, " + name + "!")
    print("How are you doing today?")
```

Call a Function:

```
greet("Rex")
```

```
# Hello, Rex!
  How are you doing today?
```

PASS ARGUMENTS TO A FUNCTION

Pass single argument:

```
def greet(name):
    print("Hello, " + name + "!")

greet("Rex")
```

Hello, Rex!

Pass two arguments:

```
def employee(name,job):
    print(name, 'is a', job)

employee('Rex','Software Developer')
```

Rex is a Software Developer

DEFAULT ARGUMENTS

```
def employee(name, job = 'Software Developer'):
    print(name, 'is a', job)

employee('Rex’)

# Rex is a Software Developer
```

```
def employee(name, job = 'Software Developer'):
    print(name, 'is a', job)

employee('Rex', 'Software Engineer')

# Rex is a Software Engineer
```

VARIABLE LENGTH ARGUMENTS

To create functions that take unlimited number of arguments using *args.

```
def employee_age(*args):
    print(args)
employee_age(25, 27, 35, 28, 40)
# (25, 27, 35, 28, 40)

def employee_name(*args):
    print(args)
employee_name("Bing", "Bang", "Boo")
# ("Bing", "Bang", "Boo")
```

THE RETURN STATEMENT

Return Single Value:

```
def sum(x, y):
    return x + y

result = sum(5, 10)
print(result)                    # 15
```

Return Multiple Values:

```
def calculate(x, y):
    return x + y, x*y

result = calculate(5, 10)
print(result)                    # (15, 50)
```

#EXERCISE 5: TESLA

You just got employed by Tesla, and you need to solve a problem for their self driving car:

```
if int(age) < 18:
    print("Sorry, you are too young to drive this car. Powering off")
elif int(age) > 18:
    print("Powering On. Enjoy the ride!");
elif int(age) == 18:
    print("Congratulations on your first year of driving. Enjoy the ride!")
```

1. Wrap the above code in a function called checkDriverAge().
2. Make the checkDriverAge() function accept an argument of age, so that if you enter: checkDriverAge(50), it returns "Powering On. Enjoy the ride!"; whereas if no argument is given, set the default age to 0 .

#EXERCISE 5: TESLA

1. Wrap the above code in a function called checkDriverAge().

```
def checkDriverAge():
    age = input("What is your age?: ")
    if int(age) < 18:
        print("Sorry, you are too young to drive this car. Powering off")
    elif int(age) > 18:
        print("Powering On. Enjoy the ride!");
    elif int(age) == 18:
        print("Congratulations on your first year of driving. Enjoy the ride!")
checkDriverAge()
```

```
What is your age?: 50
Powering On. Enjoy the ride!
```

#EXERCISE 5: TESLA

2. Make the checkDriverAge() function to accept an argument of age, so that if you enter: checkDriverAge(50), it returns "Powering On. Enjoy the ride!"; whereas if no argument is given, set the default age to 0 .

```
def checkDriverAge(age=0):
    if int(age)<18:
        print("Sorry, you are too young to drive this car. Powering off")
    elif int(age)>18:
        print("Powering On. Enjoy the ride")
    elif int(age)==8:
        print("Congratulations on your first year od driving. Enjoy the ride!")

checkDriverAge()
```

```
Sorry, you are too young to drive this car. Powering off
```

```
checkDriverAge(65)
```

```
Powering On. Enjoy the ride!
```

Modules & Files

Modules, Files, Directory

WHAT IS A MODULE?

- A module is a file containing Python definitions and statements.
- Consider a module to be the same as a code library.
- It is a file containing a set of functions you want to include in your application.
- The file name is the module name with the suffix .py appended.

WHAT IS A MODULE?

Example:

Save the following code in a file named *mymodule.py*

mymodule.py

```
def greeting(name):
  print("Hello, " + name)

```

Import the module named *mymodule*, and call the greeting function:

Test.py

```
import mymodule

mymodule.greeting("Jonathan")

```

Hello, Jonathan

BUILT-IN MODULE

Example:

- Import and use the platform module:

Test.py

```
1 import platform
2
3 x = platform.system()
4 print(x)
```

- Output:

```
Windows
```

USING THE DIR() FUNCTION

There is a built-in function to list all the function names (or variable names) in a module. The **dir()** function:

```
import platform

x = dir(platform)
print(x)
```

```
['DEV_NULL', '_UNIXCONFDIR', '_WIN32_CLIENT_RELEASES', '_WIN32_SERVER_RELEASES',
'__builtins__', '__cached__', '__copyright__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', '__version__',
'_comparable_version', '_component_re', '_default_architecture',
'_dist_try_harder', '_follow_symlinks', '_ironpython26_sys_version_parser',
'_ironpython_sys_version_parser', '_java_getprop', '_libc_search',
'_linux_distribution', '_lsb_release_version', '_mac_ver_xml', '_node',
'_norm_version', '_parse_release_file', '_platform', '_platform_cache',
'_pypy_sys_version_parser', '_release_filename', '_release_version',
'_supported_dists', '_sys_version', '_sys_version_cache', '_sys_version_parser',
'_syscmd_file', '_syscmd_uname', '_syscmd_ver', '_uname_cache', '_ver_output',
'_ver_stages', 'architecture', 'collections', 'dist', 'java_ver', 'libc_ver',
'linux_distribution', 'mac_ver', 'machine', 'node', 'os', 'platform', 'popen',
'processor', 'python_branch', 'python_build', 'python_compiler',
'python_implementation', 'python_revision', 'python_version',
'python_version_tuple', 're', 'release', 'subprocess', 'sys', 'system',
'system_alias', 'uname', 'uname_result', 'version', 'warnings', 'win32_ver']
```

Output:

A list of all functions' names within the platform module.

#EXERCISE 6: ADDITION.PY

- Define a module called addition.py which has a function that adds two numbers and returns the result:

```
def add(a,b):
    result = a+b
    return result
```

In addition.py file

- Import the module and use it to get the sum of two numbers.

```
import addition

result = addition.add(43,33)
print(result)
```

```
76
```

76

FILES

PYTHON HAS SEVERAL FUNCTIONS FOR CREATING, READING, UPDATING, AND DELETING FILES.

FILE HANDLING IN PYTHON - OPEN()

The key function for working with files in Python is the open() function.

The open() function takes two parameters: **filename**, and **mode**.

There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

FILE HANDLING IN PYTHON – READ()

Create a text file called (workshop.txt) and type anything in the file.

You can open the file using:

```
f = open("workshop.txt","rt")
```

The open() function returns a file object, which has a read() method for reading the content of the file:

```
f = open("workshop.txt","rt")
print(f.read())
```

```
Welcome to the Python workshop!
Hope you all have fun!
```

FILE HANDLING IN PYTHON – READLINE()

Modify your text file, make sure it has multiple lines of text.

You can return one line by using the readline() method:

```
f=open("workshop.txt","rt")
print(f.readline())
```

This will return the first line. By calling readline() two times, you can read the two first lines.

You can also loop through the file line by line.

```
f = open("workshop.txt","rt")
for x in f:
  print(x)
```

```
Welcome to the Python workshop!
Hope you all have fun!
```

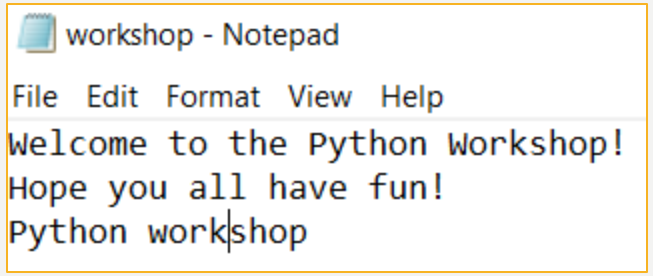
FILE HANDLING IN PYTHON – WRITE()

To write to an existing file, you must add a parameter to the open() function:

- "a" - Append - will append to the end of the file
- "w" - Write - will overwrite any existing content

Add a new line to your file

```
f = open("workshop.txt","a")
f.write("\nPython workshop")
f.close()
```



A good practice when dealing with files is to close at the end using close().

FILE HANDLING IN PYTHON – CREATE & REMOVE

To create a file, use the open() function with "w".

```
f1 = open("test.txt","w")
```

To delete a file, you must import the OS module, and run its os.remove() function:

```
import os
os.remove("workshop.txt")
```

DIRECTORY

A DIRECTORY OR FOLDER IS A COLLECTION OF FILES AND SUB DIRECTORIES.

PYTHON HAS THE OS MODULE, WHICH PROVIDES US WITH MANY USEFUL METHODS TO WORK WITH DIRECTORIES (AND FILES AS WELL).

WORKING DIRECTORY

getcwd() method:

– Is used to **get** the current working directory.

```
import os
```

```
os.getcwd()
```

```
'C:\\Users\\halabsi'
```

Test.py

```
import os
print(os.getcwd())
```

```
C:\Users\halabsi
```

chdir() method:

- Is used to **change** the current working directory.

```
os.chdir('C:\\PythonWorkshop')
```

```
os.getcwd()
```

```
'C:\\PythonWorkshop'
```

LIST DIRECTORIES AND FILES

listdir() method:

– Is used to list files and sub-directories inside a directory

```
os.listdir()
```

```
['Images',
 'Materials',
 'Program.docx',
 'Python For Beginners.pptx',
 'workshop.txt']
```

MAKING A NEW DIRECTORY

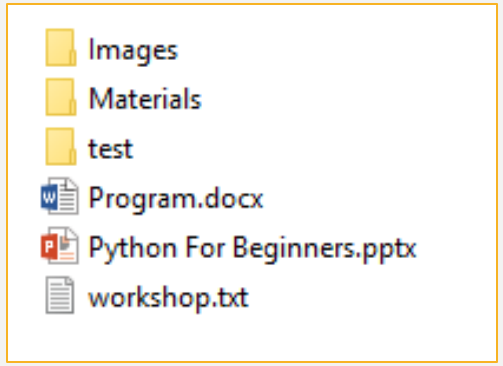
mkdir() method:

– Is used to **create** a new directory (folder)

```
os.mkdir('test')
```

```
os.listdir()
```

```
['Images',
 'Materials',
 'Program.docx',
 'Python For Beginners.pptx',
 'test',
 'workshop.txt']
```



RENAMING A DIRECTORY OR A FILE

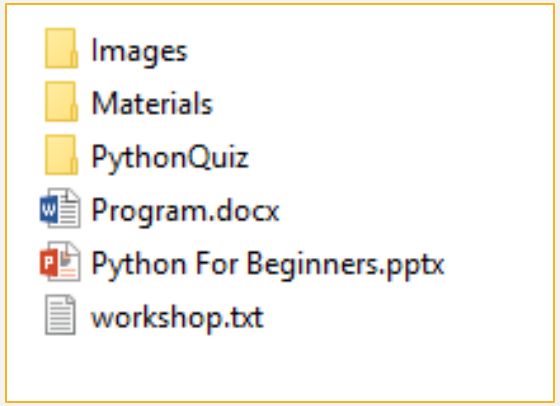
rename() method

– Is used to rename a file or a directory.

```
os.rename('test','PythonQuiz')
```

```
os.listdir()
```

```
['Images',
 'Materials',
 'Program.docx',
 'Python For Beginners.pptx',
 'PythonQuiz',
 'workshop.txt']
```



REMOVING DIRECTORY OR FILE

remove() method

– Is used to remove a file.

```
os.remove('Python For Beginners.pptx')
```

```
os.listdir()
```

```
['Images', 'Materials', 'Program.docx', 'PythonQuiz', 'workshop.txt']
```

rmdir() method

- Is used to remove an **empty** directory.

```
os.listdir()
```

```
['Images', 'Materials', 'Program.docx', 'PythonQuiz', 'workshop.txt']
```

```
os.rmdir('Images')
```

```
os.listdir()
```

```
['Materials', 'Program.docx', 'PythonQuiz', 'workshop.txt']
```

#EXERCISE 7

Write a python script that:

- Creates a directory called "MyDirectory".
- Creates a file inside "MyDirectory" named "MyInfo.txt" with the following info (use \n to add to different lines):

  Name:Your Name

  DOB:Your DOB

  Email

- Removes "MyInfo.txt" (optional).

```
import os

os.chdir("C:\\")
os.mkdir("MyDirectory")
os.chdir("C:\\MyDirectory")
file = open("MyInfo.txt","w")
file.write("Name: Sam Wong\n")
file.write("DOB: 31/12/2000\n")
file.write("email: sam@gmail.com\n")
file.close()
```

LET'S TRY OUT A FEW PYTHON PROGRAMS!

DICE ROLLING SIMULATOR

```
import random

while True:
    rolled_num = random.randint(1,6)
    print("The dice rolled and you got: ", rolled_num)
    input("Press any key to roll again.")
```

GUESSING GAME

```
import random
n = random.randint(1, 99)
guess = int(input("Enter an integer from 1 to 99: "))
while n != "guess":
    print
    if guess < n:
        print("guess is low")
        guess = int(input("Enter an integer from 1 to 99: "))
    elif guess > n:
        print("guess is high")
        guess = int(input("Enter an integer from 1 to 99: "))
    else:
        print("you guessed it!")
        break
    print
```

PYTHON HANGMAN GAME

```python
#importing the time module
import time

#welcoming the user
name = input("What is your name? ")

print ("Hello, " + name + "! Time to play hangman!")

print ("")

#wait for 1 second
time.sleep(1)

print("Start guessing...")
time.sleep(0.5)

#here we set the secret
word = "secret"

#creates an variable with an empty value
guesses = ""

#determine the number of turns
turns = 10

# Create a while loop

#check if the turns are more than zero
while turns > 0:

    # make a counter that starts with zero
    failed = 0

    # for every character in secret_word
    for char in word:

    # see if the character is in the players guess
        if char in guesses:

        # print then out the character
            print (char)

        else:

        # if not found, print a dash
            print ("_")

        # and increase the failed counter with one
            failed += 1

    # if failed is equal to zero

    # print You Won
    if failed == 0:
        print ("You won")

    # exit the script
        break

    print(" ")

    # ask the user go guess a character
    guess = input("guess a character:")

    # set the players guess to guesses
    guesses += guess

    # if the guess is not found in the secret word
    if guess not in word:

     # turns counter decreases with 1 (now 9)
        turns -= 1

    # print wrong
        print ("Wrong")

    # how many turns are left
        print ("You have " + str(turns) + " more guesses")

    # if the turns are equal to zero
        if turns == 0:

        # print "You Loose"
            print ("You Loose")
```

QUICK RECAP

- What is Python?
- Where and what it can be used for?
- How to install and begin programming in Python?
- Your first program: writing and reading
- Understand the building blocks of programs: Data Types
- Perform mathematical operations
- Control the program flow: Sequence, Decision and Iteration
- Using Functions, Modules, Files and Directories
- Importing Modules

IN CASE YOU MISSED IT

DON’T FORGET TO

=

GIVE FEEDBACK

+

COLLECT CERT

+

TAKE PICTURES

+

BONUS: EXPLORE OPEN DAY


python
101
ZERO TO HERO
in 3 hours

THANK YOU