MY JOURNAL TO PYTHON

4110E210

MY DEAR GREAT TEACHER

AGENDA

- Python
- Input and Output : input() and print
- Data Types : numeric, string, list, dict.
- Operation ON data type:
- CONTROLS : IF- | IF –ELSE IF | -F-ELSE
- LOOP : FOR |WHILE | RANGE () |BREAK | CONTINUE
- FUNCTION

① PARAMETERS (ARGUMENTS)
② RECURSIVE FUNCTION
③ LAMBA FUNCTION

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.

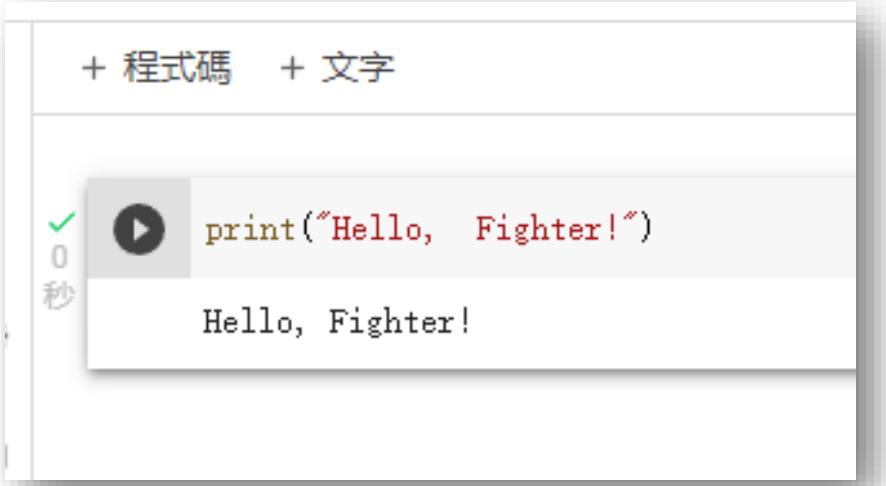
Good to know:

The most recent major version of Python is Python 3, which we shall be using in this tutorial.

However, Python 2, although not being updated with anything other than security updates, is still quite popular.
In this tutorial Python will be written in a text editor.

It is possible to write Python in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse which are particularly useful when managing larger collections of Python files.

Example



PYTHON SYNTAX

- Python syntax can be executed by writing directly in the Command Line:

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

Or by creating a python file on the server, using the .py file extension, and running it in the Command Line:

```
C:\Users\Your Name>python myfile.py
```

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.

Python uses indentation to indicate a block of code.

Example

```
if 6 > 3:
    print("Six is greater than three!")
```

```
Six is greater than three!
```

Python will give you an error if you skip the indentation:

Example

```
if 6 > 3:
print("Six is greater than three!")
```

```
  File "<ipython-input-10-34e4e31cc3cb>", line 2
    print("Six is greater than three!")
        ^
IndentationError: expected an indented block
```

SEARCH STACK OVERFLOW

The number of spaces is up to you as a programmer, the most common use is four, but it has to be at least one.

Example

```
if 6 > 3:
    print("Six is greater than three!")
if 6 > 3:
                print("Six is greater than three!")
```

```
Six is greater than three!
Six is greater than three!
```

You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

```
if 6 > 3:
    print("Six is greater than three!")
                print("Six is greater than three!")
```

```
  File "<ipython-input-12-c0cb13af0a71>", line 3
    print("Six is greater than three!")
    ^
IndentationError: unexpected indent
```

SEARCH STACK OVERFLOW

In Python, variables are created when you assign a value to it:

Example

:

```
x = 5
y = "Hello, Fighter!"
```

Python has no command for declaring a variable.

Comments

Python has commenting capability for the purpose of in-code documentation.
Comments start with a #, and Python will render the rest of the line as a comment:

Example

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

```
Hello, Fighter!
```

PYTHON VARIABLES

In Python, variables are created when you assign a value to it:

Example

Variables in Python:

```
x = 5
y = "Hello, Fighter!"
```

Python has no command for declaring a variable.

You will learn more about variables in the Python Variables chapter.

Comments

Python has commenting capability for the purpose of in-code documentation.

Comments start with a #, and Python will render the rest of the line as a comment:

Example

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

```
Hello, Fighter!
```

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:

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

```
Hello, Fihgter!
```

PYTHON COMMENTS

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:

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

```
Hello, Fighter!
```

Comments can be placed at the end of a line, and Python will ignore the rest of the line:

Example

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

```
Hello, Fighter!
```

A comment does not have to be text that explains the code, it can also be used to prevent Python from executing code:

```
#print("Hello, Fighter!")
print("Cheers, LOve!")
```

```
Cheers, LOve!
```

Multi Line Comments

Python does not really have a syntax for multi line comments.
To add a multiline comment you could insert a # for each line:

Example

```
#This is a comment
#written in
#more than just one line
print("Hello, Fighter!")
```

```
Hello, Fighter!
```

Or, not quite as intended, you can use a multiline string.

Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:

Example

As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

```
"""
This is a comment
written in
more than just one line
"""
print("Hello, Fighter!")
```

```
Hello, Fighter!
```

PYTHON VARIABLES

Variables

Variables are containers for storing data values.

Creating Variables

Python has no command for declaring a variable.

A variable is created the moment you first assign a value to it.

Example

```
x = 9
y = "Angel"
print(x)
print(y)
```

```
9
Angel
```

Variables do not need to be declared with any particular *type*, and can even change type after they have been set.

Example

```
x = 9        # x is of type int
x = "Angel" # x is now of type str
print(x)
```

```
Angel
```

Casting

If you want to specify the data type of a variable, this can be done with casting.

Example

```
x = str(9)     # x will be '9'
y = int(9)     # y will be 9
z = float(9)   # z will be 9.0
```

Python - Variable Names

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)

Example

```
myvar = "Angel"
my_var = "Angel"
_my_var = "Angel"
myVar = "Angel"
MYVAR = "Angel"
myvar2 = "Angel"
```

Example

Illegal variable names:

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

```
  File "<ipython-input-21-4f0bd1ea342e>", line 1
    2myvar = "Angel"
         ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

Multi Words Variable Names

Variable names with more than one word can be difficult to read.

There are several techniques you can use to make them more readable:

Camel Case

Each word, except the first, starts with a capital letter:

Each word starts with a capital letter:

```
myVariableName = "Angel"
```

Pascal Case

Each word starts with a capital letter:

```
MyVariableName = "Angel"
```

Snake Case

Each word is separated by an underscore character:

```
my_variable_name = "John"
```

Python Variables - Assign Multiple Values

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

Example

```
x, y, z = "Square", "Circle", "Triangle"
print(x)
print(y)
print(z)
```

One Value to Multiple Variables

And you can assign the *same* value to multiple variables in one line:

Example

```
x = y = z = "Square"
print(x)
print(y)
print(z)
```

```
Square
Square
Square
```

UNPACK A COLLECTION

If you have a collection of values in a list, tuple etc. Python allows you to extract the values into variables. This is called *unpacking*.

Example

```
shapes = ["square", "circle", "triangle"]
x, y, z = shapes
print(x)
print(y)
print(z)
```

```
square
circle
triangle
```

PYTHON - OUTPUT VARIABLES

Output Variables

The Python print() function is often used to output variables.

Example

```
x = "Python is challenging"
print(x)
```

```
Python is challenging
```

In the print() function, you output multiple variables, separated by a comma:

Example

```python
x = "Python"
y = "is"
z = "challenging"
print(x, y, z)
```

```
Python is challenging
```

You can also use the + operator to output multiple variables:

Example

```python
x = "Python "
y = "is "
z = "challenging"
print(x + y + z)
```

```
Python is challenging
```

For numbers, the + character works as a mathematical operator:

Example

```
x = 9
y = 18
print(x + y)
```

```
27
```

In the print() function, when you try to combine a string and a number with the + operator, Python will give you an error:

Example

```
x = 9
y = "Angel"
print(x + y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-36-8ad36b56f839> in <module>
      1 x = 9
      2 y = "Angel"
----> 3 print(x + y)

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

SEARCH STACK OVERFLOW

The best way to output multiple variables in the print() function is to separate them with commas, which even support different data types:

Example

```
x = 9
y = "Angel"
print(x, y)
```

```
9 Angel
```

Python - Global Variables

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

```
x = "challenging"

def myfunc():
  print("Python is " + x)

myfunc()
```

```
Python is challenging
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```
x = "challenging"

def myfunc():
  x = "enthusiastic"
  print("Python is " + x)

myfunc()

print("Python is " + x)
```

```
Python is enthusiastic
Python is challenging
```

The global Keyword

Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.

To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
  global x
  x = "enthusiastic"

myfunc()

print("Python is " + x)
```

```
Python is enthusiastic
```

Also, use the global keyword if you want to change a global variable inside a function.

Example

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

```
x = "challenging"

def myfunc():
  global x
  x = "enthusiastic"

myfunc()

print("Python is " + x)
```

```
Python is enthusiastic
```

PYTHON DATA TYPES

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`

Getting the Data Type

You can get the data type of any object by using the type() function:

Example

Print the data type of the variable x:

```
x = 8
print(type(x))
```

```
<class 'int'>
```

SETTING THE DATA TYPE

In Python, the data type is set when you assign a value to a variable:

Example	Data Type	Try it
`x = "Hello World"`	str	Try it »
`x = 20`	int	Try it »
`x = 20.5`	float	Try it »
`x = 1j`	complex	Try it »
`x = ["apple", "banana", "cherry"]`	list	Try it »
`x = ("apple", "banana", "cherry")`	tuple	Try it »
`x = range(6)`	range	Try it »
`x = {"name" : "John", "age" : 36}`	dict	Try it »
`x = {"apple", "banana", "cherry"}`	set	Try it »
`x = frozenset({"apple", "banana", "cherry"})`	frozenset	Try it »
`x = True`	bool	Try it »
`x = b"Hello"`	bytes	Try it »
`x = bytearray(5)`	bytearray	Try it »
`x = memoryview(bytes(5))`	memoryview	Try it »
`x = None`	NoneType	Try it »

SETTING THE SPECIFIC DATA TYPE

If you want to specify the data type, you can use the following constructor functions:

Example	Data Type	Try it
`x = str("Hello World")`	str	Try it »
`x = int(20)`	int	Try it »
`x = float(20.5)`	float	Try it »
`x = complex(1j)`	complex	Try it »
`x = list(("apple", "banana", "cherry"))`	list	Try it »
`x = tuple(("apple", "banana", "cherry"))`	tuple	Try it »
`x = range(6)`	range	Try it »
`x = dict(name="John", age=36)`	dict	Try it »
`x = set(("apple", "banana", "cherry"))`	set	Try it »
`x = frozenset(("apple", "banana", "cherry"))`	frozenset	Try it »
`x = bool(5)`	bool	Try it »
`x = bytes(5)`	bytes	Try it »
`x = bytearray(5)`	bytearray	Try it »
`x = memoryview(bytes(5))`	memoryview	Try it »

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:

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

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

Int

Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example

Integers

```
x = 3
y = 12192002121922
z = -54321

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

```
<class 'int'>
<class 'int'>
<class 'int'>
```

Float

Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example

```
x = 1.19
y = 1.9
z = -22.19

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

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Float can also be scientific numbers with an "e" to indicate the power of 10.

Example

Floats:

```
x = 19e2
y = 12E4
z = -97.7e129

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

```
<class 'float'>
<class 'float'>
<class 'float'>
```

Complex

Complex numbers are written with a "j" as the imaginary part:

Example

Complex:

```
x = 9+2j
y = 2j
z = -2j

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

```
<class 'complex'>
<class 'complex'>
<class 'complex'>
```

Type Conversion

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

Example

Convert from one type to another:

```python
x = 2    # int
y = 1.9  # float
z = 9j   # 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))
```

```
2.0
1
(2+0j)
<class 'float'>
<class 'int'>
<class 'complex'>
```

Random Number

Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example

Import the random module, and display a random number between 1 and 9:

```
import random

print(random.randrange(2, 19))
```

```
11
```

PYTHON CASTING

SPECIFY A VARIABLE TYPE

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

•int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

•float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

•str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example

Integers:

```
x = int(1)   # x will be 1
y = int(2.9) # y will be 2
z = int("3") # z will be 3
```

Floats:

```
x = float(1)     # x will be 1.0
y = float(2.9)   # y will be 2.9
z = float("3")   # z will be 3.0
w = float("4.1") # w will be 4.2
```

Strings:

```
x = str("s2") # x will be 's2'
y = str(3)    # y will be '3'
z = str(4.0)  # z will be '4.0'
```

PYTHON STRINGS

STRINGS

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example

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

```
Hello
Hello
```

ASSIGN STRING TO A VARIABLE

Assigning a string to a variable is done with the variable name followed by an equal sign and the string:

Example

```
a = "Hello"
print(a)
```

```
Hello
```

MULTILINE STRINGS

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

Example

You can use three double quotes:

```
a = """Dont just take but give,
Dont just see but feel,
Dont just dream but do."""
print(a)
```

```
Dont just take but give,
Dont just see but feel,
Dont just dream but do.
```

Or three single quotes:

Example

```
a = '''Dont just take but give,
Dont just see but feel,
Dont just dream but do.'''
print(a)
```

```
Dont just take but give,
Dont just see but feel,
Dont just dream but do.
```

STRINGS ARE ARRAYS

Like many other popular programming languages, strings in Python are arrays of bytes representing unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.

Square brackets can be used to access elements of the string.

Example

Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, Fighter!"
print(a[1])
```

```
e
```

LOOPING THROUGH A STRING

Since strings are arrays, we can loop through the characters in a string, with a for loop.

Example

Loop through the letters in the word “square":

```
for x in "square":
  print(x)
```

```
s
q
u
a
r
e
```

STRING LENGTH

To get the length of a string, use the len() function.

Example

The len() function returns the length of a string:

```
a = "Hello, Fighter!"
print(len(a))
```

```
15
```

CHECK STRING

To check if a certain phrase or character is present in a string, we can use the keyword in.

Example

Check if “love" is present in the following text:

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"
print("love" in txt)
```

```
True
```

Use it in an if statement:

Example

Print only if “love" is present:

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"
if "love" in txt:
  print("Yes, 'love' is present.")
```

```
Yes, 'love' is present.
```

CHECK IF NOT

To check if a certain phrase or character is **NOT** present in a string, we can not in.

Example

Check if "**friendship**" is NOT present in the following text:

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"
print("friendship" not in txt)
```

```
True
```

Use it in an **if** statement:

Example

print only if “friendship" is NOT present:

```
txt = "True love is like ghosts, which everyone talks about and few have seen.!"
if "friendship" not in txt:
  print("No, 'friendship' is NOT present.")
```

```
No, 'friendship' is NOT present.
```

PYTHON - SLICING STRINGS

Example

Get the characters from position 2 to position 5 (not included):

```
b = "Hello, fighter!"
print(b[2:5])
```

```
llo
```

SLICING

You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.

SLICE FROM THE START

By leaving out the start index, the range will start at the first character:

Example

Get the characters from the start to position 5 (not included):

```
b = "Hello, Fighter!"
print(b[:5])
```

```
Hello
```

SLICE TO THE END

By leaving out the *end* index, the range will go to the end:

Example

Get the characters from position 2, and all the way to the end:

```
b = "Hello, Fighter!"
print(b[2:])
```

```
llo, Fighter!
```

NEGATIVE INDEXING

Use negative indexes to start the slice from the end of the string:

Example

Get the characters:

From: “g" in “Fighter!" (position -5)

To, but not included: “r" in “Fighter!" (position -2)

```
b = "Hello, Fighter!"
print(b[-5:-2])
```

```
hte
```

PYTHON - MODIFY STRINGS

Python has a set of built-in methods that you can use on strings.

UPPER CASE

Example

The upper() method returns the string in upper case:

```
a = "Hello, Fighter!"
print(a.upper())
```

LOWER CASE

Example

The lower() method returns the string in lower case:

```
a = "Hello, Fighter!"
print(a.lower())
```

REMOVE WHITESPACE

Whitespace is the space before and/or after the actual text, and very often you want to remove this space.

Example

The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, Fighter! "
print(a.strip()) # returns "Hello, Fighter!"
```

```
Hello, Fighter!
```

REPLACE STRING

Example

The replace() method replaces a string with another string:

```
a = "Hello, Fighter!"
print(a.replace("H", "H"))
```

```
Hello, Fighter!
```

SPLIT STRING

The split() method returns a list where the text between the specified separator becomes the list items.

Example

The split() method splits the string into substrings if it finds instances of the separator:

```
a = "Hello, Fighter!"
print(a.split(",")) # returns ['Hello', ' Fighter!']
```

```
['Hello', ' Fighter!']
```

PYTHON - STRING CONCATENATION

String Concatenation

To concatenate, or combine, two strings you can use the + operator.

Example

Merge variable a with variable b into variable c:

```
a = "Hello"
b = "Fighter"
c = a + b
print(c)
```

```
HelloFighter
```

Example

To add a space between them, add a " ":

```
a = "Hello"
b = "Fighter"
c = a + " " + b
print(c)
```

```
Hello Fighter
```