Function, OOP, and Library in Python

NULL SCIENCE

- FAHMI HAMZAH
- ADITYA HERMAWAN
- FARA RIZKHI KARUNIA
- SALSABILLA ATASYA PUTRI SETYAWAN
- NAFIATUL RISA

Python Functions

A FUNCTION IS A BLOCK OF CODE WHICH ONLY RUNS WHEN IT IS CALLED.
YOU CAN PASS DATA, KNOWN AS PARAMETERS, INTO A FUNCTION.
A FUNCTION CAN RETURN DATA AS A RESULT.

Creating a Function

```
[2]  def my_function():
         print("Hello Word")

     my_function()
```

```
Hello Word
```

Arguments

INFORMATION CAN BE PASSED INTO FUNCTIONS AS ARGUMENTS. ARGUMENTS ARE SPECIFIED AFTER THE FUNCTION NAME, INSIDE THE PARENTHESES. YOU CAN ADD AS MANY ARGUMENTS AS YOU WANT, JUST SEPARATE THEM WITH A COMMA.

THE FOLLOWING EXAMPLE HAS A FUNCTION WITH ONE ARGUMENT (FNAME). WHEN THE FUNCTION IS CALLED, WE PASS ALONG A FIRST NAME, WHICH IS USED INSIDE THE FUNCTION TO PRINT THE FULL NAME:

```
[5]  def my_function(fname):
       print("Data " + fname)

     my_function("Science")
     my_function("Analysist")
     my_function("Engineering")
```

```
Data Science
Data Analysist
Data Engineering
```

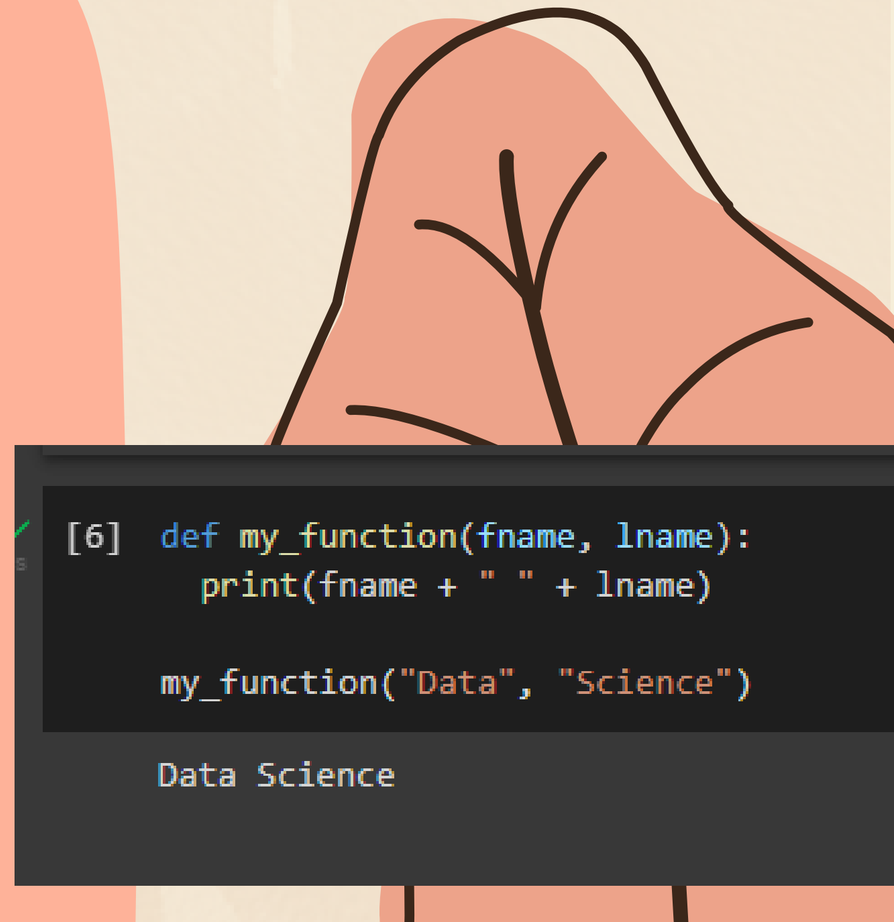
Parameters or Arguments

FROM A FUNCTION'S PERSPECTIVE:
A PARAMETER IS THE VARIABLE LISTED INSIDE THE PARENTHESES IN THE FUNCTION DEFINITION.
AN ARGUMENT IS THE VALUE THAT IS SENT TO THE FUNCTION WHEN IT IS CALLED.

Number of Arguments

BY DEFAULT, A FUNCTION MUST BE CALLED WITH THE CORRECT NUMBER OF ARGUMENTS. MEANING THAT IF YOUR FUNCTION EXPECTS 2 ARGUMENTS, YOU HAVE TO CALL THE FUNCTION WITH 2 ARGUMENTS, NOT MORE, AND NOT LESS.

EXAMPLE THIS FUNCTION EXPECTS 2 ARGUMENTS, AND GETS 2 ARGUMENTS:



```
[6]  def my_function(fname, lname):
       print(fname + " " + lname)

     my_function("Data", "Science")
```

```
Data Science
```

Arbitrary Arguments, *args

IF YOU DO NOT KNOW HOW MANY ARGUMENTS THAT WILL BE PASSED INTO YOUR FUNCTION, ADD A * BEFORE THE PARAMETER NAME IN THE FUNCTION DEFINITION.
THIS WAY THE FUNCTION WILL RECEIVE A TUPLE OF ARGUMENTS, AND CAN ACCESS THE ITEMS ACCORDINGLY:

```
def my_function(*name):
  print("Data " + name[2])

my_function("Science", "Analysist", "Engineering")
```

```
Data Engineering
```

Keyword Arguments

YOU CAN ALSO SEND ARGUMENTS WITH THE KEY = VALUE SYNTAX.
THIS WAY THE ORDER OF THE ARGUMENTS DOES NOT MATTER.

```
[10] def my_function(ciri1, ciri2, ciri3):
       print("Kampus Merdeka " + ciri2)

     my_function(ciri1 = "Magang", ciri2 = "independent", ciri3 = "Mengajar")
```

```
Kampus Merdeka independent
```

Arbitrary Keyword Arguments, **kwargs

IF YOU DO NOT KNOW HOW MANY KEYWORD ARGUMENTS THAT WILL BE PASSED INTO YOUR FUNCTION, ADD TWO ASTERISK: ** BEFORE THE PARAMETER NAME IN THE FUNCTION DEFINITION.
THIS WAY THE FUNCTION WILL RECEIVE A DICTIONARY OF ARGUMENTS, AND CAN ACCESS THE ITEMS ACCORDINGLY:

```
[11] def my_function(**name):
       print("Study Independent " + name["fname"])

     my_function(fname = "Data Science Track", lname = "Visual Vidio")
```

```
Study Independent Data Science Track
```

Default Parameter Value

THE FOLLOWING EXAMPLE SHOWS HOW TO USE A DEFAULT PARAMETER VALUE.
IF WE CALL THE FUNCTION WITHOUT ARGUMENT, IT USES THE DEFAULT VALUE:

```
[13] def my_function(country = "Indonesia"):
       print("I am from " + country)

     my_function("Jakarta")
     my_function("Bandung")
     my_function()
     my_function("Jogja")
```

```
I am from Jakarta
I am from Bandung
I am from Indonesia
I am from Jogja
```

Passing a List as an Argument

YOU CAN SEND ANY DATA TYPES OF ARGUMENT TO A FUNCTION (STRING, NUMBER, LIST, DICTIONARY ETC.), AND IT WILL BE TREATED AS THE SAME DATA TYPE INSIDE THE FUNCTION.
E.G. IF YOU SEND A LIST AS AN ARGUMENT, IT WILL STILL BE A LIST WHEN IT REACHES THE FUNCTION:

```
[14] def my_function(food):
       for x in food:
         print(x)

     fruits = ["apple", "banana", "cherry"]

     my_function(fruits)
```

```
apple
banana
cherry
```

Return Values

TO LET A FUNCTION RETURN A VALUE, USE THE RETURN STATEMENT:

```
[16] def my_function(x):
       return 2 * x

     print(my_function(3))
     print(my_function(5))
     print(my_function(9))
```

```
6
10
18
```

The pass Statement

FUNCTION DEFINITIONS CANNOT BE EMPTY, BUT IF YOU FOR SOME REASON HAVE A FUNCTION DEFINITION WITH NO CONTENT, PUT IN THE PASS STATEMENT TO AVOID GETTING AN ERROR.

```
[17] def myfunction():
         pass
```

Object-Oriented Programming (OOP)

OOP

Object-Oriented Programming (OOP) is a programming paradigm that uses objects and classes in programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

a little bit of OOP :

Class

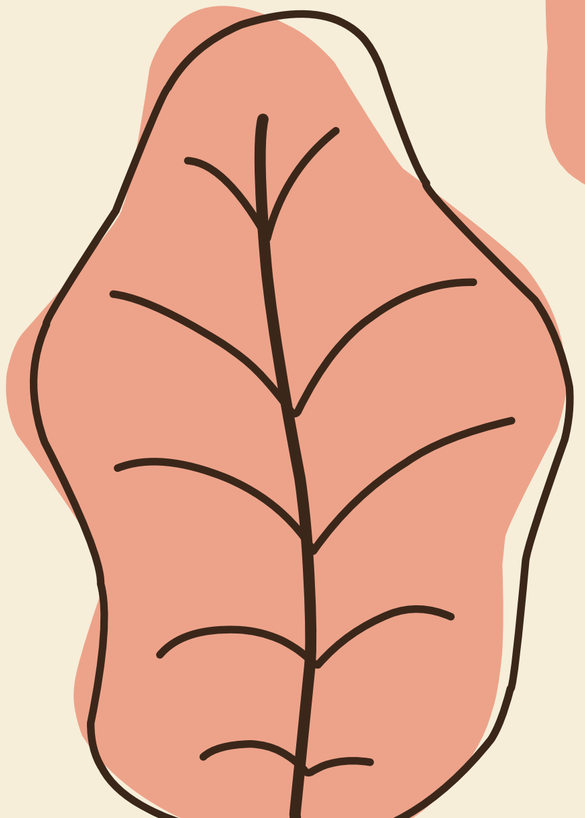
Object Method

Class Method

Static Method

Class

A CLASS IS A *COLLECTION OF OBJECTS*. A CLASS CONTAINS THE BLUEPRINTS OR THE PROTOTYPE FROM WHICH THE OBJECTS ARE BEING CREATED. IT IS A LOGICAL ENTITY THAT CONTAINS SOME ATTRIBUTES AND METHODS.



Define a class :

```
class ClassName:
    # Statement-1
    .
    # Statement-N
```

Object Method

------- we will set a class and make an Object to using the method of Class. we should use *self* as a parameter

```
[ ]  class Calculator: #Class
       def __init__(self, num1, num2):
         self.num1 = num1
         self.num2 = num2

       def addition(self):
         self.add = self.num1 + self.num2
         return self.add
```

__init__() functions means it always executed when the class is being initiated.

self is a reference to the current instance of the class, and is used to access variables that belongs to the class

```
[3]  cal = Calculator(3,2)
```

cal is an object

```
[4]  cal.addition()
```

```
[5]  Calculator.addition(cal)
```

another way to call a method from class

Class Method

------- similar with method before, but in class method we're using @classmethod before the method

```
[6]  class Calculator_cm:

       def __init__(self, num1, num2):
         self.num1 = num1
         self.num2 = num2

       @classmethod
       def addition(cls, num1, num2):
         cls.addNum = num1 + num2
         return cls.addNum
```

using **@classmethod**

in parameter we should using cls as a first parameter if we want to use this class method.

```
[7]  k = Calculator_cm

     k.addition(4,5)
```

Static Method

------- similar with method before, it's simple method that we can use. Don't ned self or cls to parameter

```
[8]  class Calculator_sm:

       def __init__(self, num1, num2):
         self.num1 = num1
         self.num2 = num2

       def addition(num1, num2):
         addNum = num1 + num2
         return addNum
```

using general parameter

return as a variable

```
[9]  cal = Calculator_sm

     print("Addition",cal.addition(7,9))
```

Unit Test

a **software testing** process that ensures **every unit/function of the program is tested.**

Simple Unittest

```
[ ]  import unittest

     a = 3
     b = 5

     class Test__53a_UnitTest(unittest.TestCase):

         def test_equal(self):
             self.assertEqual(3 + 4, a + b)

         def test_greater(self):
             self.assertTrue(a + b > 3 + 4)

         if __name__ == '__main__':
             unittest.main()
```

used to check if the result obtained is equal to the expected result..

used to verify if a given statement is true.

output

```
E
======================================================================
ERROR: /root/ (unittest.loader._FailedTest)
----------------------------------------------------------------------
AttributeError: module '__main__' has no attribute '/root/'

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (errors=1)
An exception has occurred, use %tb to see the full traceback.

SystemExit: True
```

Python Math Module

PYTHON HAS ALSO A BUILT-IN MODULE CALLED MATH, WHICH EXTENDS THE LIST OF MATHEMATICAL FUNCTIONS.

TO USE IT, YOU MUST IMPORT THE MATH MODULE:

```
[ ]  import math
```

WHEN YOU HAVE IMPORTED THE MATH MODULE, YOU CAN START USING METHODS AND CONSTANTS OF THE MODULE.

THE MATH.SQRT() METHOD FOR EXAMPLE, RETURNS THE SQUARE ROOT OF A NUMBER:

```
[2]  #Import math library
     import math

     x=math.sqrt(25)

     print(x)

     5.0
```

THE MATH.CEIL() METHOD ROUNDS A NUMBER UPWARDS TO ITS NEAREST INTEGER, AND THE MATH.FLOOR() METHOD ROUNDS A NUMBER DOWNWARDS TO ITS NEAREST INTEGER, AND RETURNS THE RESULT:

```
[3]  #Import math library
     import math

     #Round a number upward to its nearest integer
     x = math.ceil(2.4)

     #Round a number downward to its nearest integer
     y = math.floor(2.4)

     print(x)
     print(y)
```

```
3
2
```

THE MATH.PI CONSTANT, RETURNS THE VALUE OF PI (3.14...). SIMPLE CALCULATION OF THE CIRCUMFERENCE AND AREA OF A CIRCLE :

```
import math

r = float(input("Jari-Jari : "))

keliling = 2*math.pi*r
luas = math.pi*(r*r)

print("Keliling : ",keliling)
print("Luas : ",luas)
```

```
Jari-Jari : 7
Keliling :  43.982297150257104
Luas :  153.93804002589985
```

Python Random

PYTHON HAS A BUILT-IN MODULE THAT YOU CAN USE TO MAKE RANDOM NUMBER

```
[1]  import random
     from random import randint
     for i in range(10):
       print(f"Random num {i} is {randint(1,10)}")
```

```
Random num 0 is 6
Random num 1 is 2
Random num 2 is 4
Random num 3 is 7
Random num 4 is 9
Random num 5 is 10
Random num 6 is 3
Random num 7 is 10
Random num 8 is 6
Random num 9 is 1
```

TXT File, OS, and SYS

TXT File - How to Open, Read, and Write Files in Python

Opening Text Files – Read Mode

To open a text file use the open command. Make sure the file name and path are correct. If the mode parameter is not defined, the default is read mode.

```
[1]  # if in the same directory path can be ignored
     myfile = open("example.txt", "r")

[2]  myfiletest = myfile.read()

[3]  print(myfiletest)

     Hello World!
```

Reading Text File Contents

```
[4]  myfile = open("example.txt", "r")

[5]  myfilefirstline = myfile.readline()

[6]  print(myfilefirstline)

     Hello World!
```

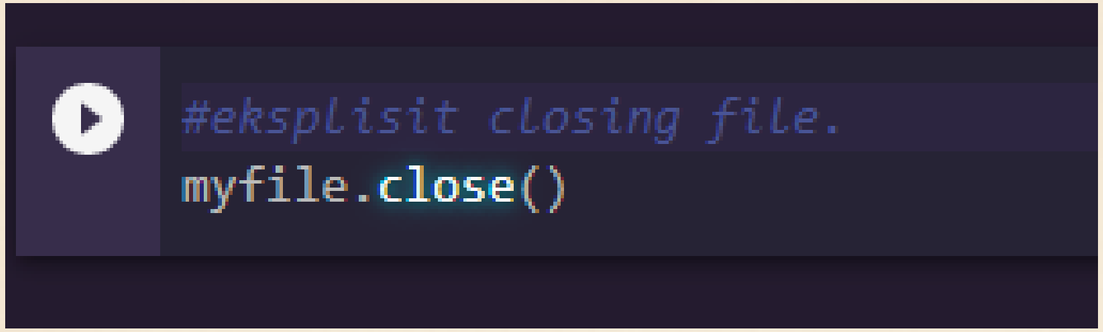
```
[9]  myfilesecondline = myfile.readline()

     print(myfilesecondline)
```

To read the contents of a text file, use the read() method. Python uses the cursor system. When a text file is opened, the cursor will be at the beginning of the file. After executing the read command, the cursor will be at the eof (end of file).

If the command read again is repeated, then the return obtained is an empty string, because the cursor is in eof. Therefore, to manipulate the contents of the file, we need to store it in a variable first.

Closing Files

Another important thing is that after opening the file, don't forget to close it. By using the close() command.



OS

The os module is a part of the standard library, or stdlib, within Python. This means that it comes with your Python installation, but you still must import it.

Sample code using os :

```
[1]  import os
```

All of the following code assumes you have os imported. Because it is not a built-in function, you must always import it. It is a part of the standard library, however, so you will not need to download or install it separately from your Python installation.

```
[3]  curDir = os.getcwd()
     print(curDir)

     /content
```

The above code will get your current working directory, hence "cwd."

To make a new directory :

```
[2]  dirName = input("Enter Name Folder : ")
     os.mkdir(dirName)
     print("Directory Created")

     Enter Name Folder : Data Science
     Directory Created
```

```
[4]  os.mkdir('newDir')
```

To change the name of, or rename, a directory :

```
[5]  os.rename('newDir','newDir2')
```

To remove a directory :

```
[6]  os.rmdir('newDir2')
```

```
[7]  os.rmdir('Data Science')
```

SYS

The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. You will learn some of the important features of this module here.

Sample code using sys :

```
[1]  import sys
```

```
[2]  try:
       s=float(input("Type a number = "))
       y=float(input("Type a number = "))
       z=s*y
       print(z)
     except (ValueError,TypeError):
         print("Please check your entered number again!")
     else:
         print("Done")
```

```
Type a number = 4
Type a number = 2
8.0
Done
```

Datetime Modulation

Datetime Modulation

Datetime modulation is a modul in Python works with date and time. Because datetime is a modul, we must import the datetime modul first with this syntax.

```
[2]  import datetime
```

Current Date and Time

To get current date and time, you can run this syntax :

```
[14] Current = datetime.datetime.now()
     print("The current date and time is :", Current)
```

Output :

```
The current date and time is : 2022-04-10 03:50:53.740534
```

Current Date

To get current date, you can run this syntax

```
[15] Today = datetime.date.today()
     print("The current date is :", Today)
```

Output :

```
The current date is : 2022-04-10
```

Date Object to Represent a Date

Input :

```
date = datetime.date(2022,10,4)
print(date)
```

Output :

```
2022-10-04
```

Format date using strftime()

The strftime() method returns a string representing date and time using date, time or datetime object.

Example :

```
#Current Date and Time
a = datetime.datetime.now()

#Current date using strftime()
date = a.strftime("%m/%d/%y")
print(date)

#Current time using strftime()
time = a.strftime("%H:%M")
print(time)
```

Format date using strftime()

Output :

```
04/10/22
04:03
```

Description :

- %Y - year [yyyy]
- %m - month [mm]
- %d - day [dd]
- %H - hour [HH]
- %M - minute [MM]
- %S - second [SS]

Thank You