HELLO WORLD, READY TO LEARN BASIC PYTHON?

LAST CHAPTER BASIC PYTHON "LIBRARY, TXT, AND FUNCTION"

LET'S KNOW US "MATPLOTLIB GROUP"

- Tobias Mikha Sulistiyo
- Daud Ibadurahman
- Fitri Alfaqrina
- Putri Reghina Hilmi Prasasti
- Sari Yuliastuti

LIST CONTENT

- Function
- Class, Object, Static Method
- Pydoc
- Io, Txt File, Os Path
- Unit Test
- Python Module

FUNCTION

Definition, Call and Return Function

FUNCTION

Function

- function is a process to connecting input and output.
- in python, function can organize codes to re-use.
- function better when just have 1 specific role but can re-use.

Define Function

- function can be define with keyword 'def' followed by function name and parameters in paranthesis '()'.
- on operational, we can add some docstring to describe function context.
- code block in the function start with ":" and with indentation
- funtion will be stop when there is statement return [expression] who give back the [expression] to called word.
- we can make a function cannot return with return 'None'.

DEFINE FUNCTION

This function can provide list, for loop, etc. function reuseable

```
def devide_by_2(x):
    devide = x/2
    return devide

devide_by_2(20)
```

```
10.0
```

DEFINE FUNCTION

if the function doesn't have a 'return'. it just to showing the results and just single-use

```
def devide_by_2(x):
  devide = x/2
  print(devide)
```

```
devide_by_2(20)
```

```
10.0
```

DEFINE FUNCTION

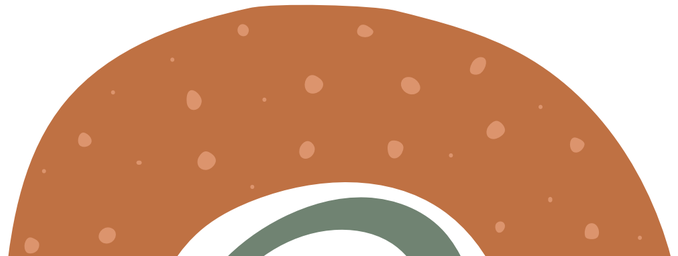
if the function just have a 'return' but didn't have definition for return, then the results didn't show

```
[1]  def devide_by_2(x):
       devide = x/2
       return

     devide_by_2(20)
```

STRING FUNCTION

```
[ ]  def name(x):
       greets = "Hello {}".format(x)
       return greets

[ ]  name("World")

     'Hello World'
```

You can use string to make function, not just for numbers



RETURN

```
def minus(x,y):
  min = x - y
  print("The results is {}".format(min))
  return min

minus(10,5)
```

```
The results is 5
5
```

'Return' can make execution out of function program and can bring back the value

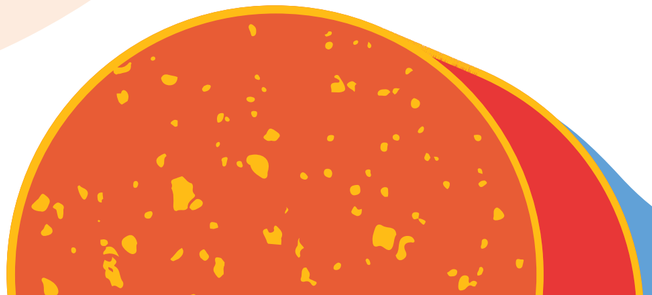
- "the results is 5" is showing results of your function
- "5" is showing results of return.

RETURN

```
[ ]  output = minus(10,5)
     print("The return results is {}".format(output))
```

```
The results is 5
The return results is 5
```

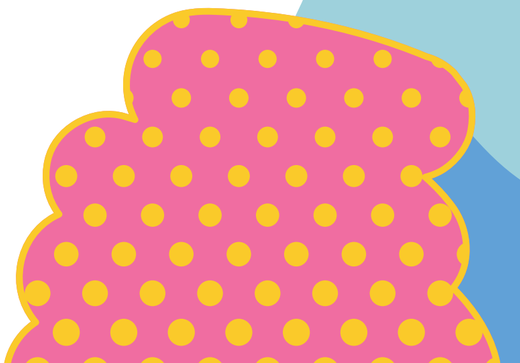
you can add string to clear up the return results



RETURN

- return value from a function can be saved in the variable
- that's the diffrent between a function return value and function didnt return value

```
[ ]  def devide_by_2(x):
       devide = x/2
       return devide

[ ]  number = 12
     result = devide_by_2(number)

[ ]  print(result)

     6.0
```



RETURN

```
number = int(input().strip())
result = devide_by_2(number)
print("The result is {}".format(result))
```

```
12
The result is 6.0
```

Can be like this

Or this

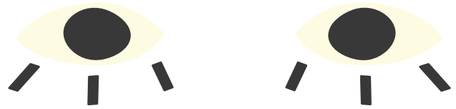
```
[ ]  print("{} devided by 2 is {}".format(number,result))
```

```
12 devided by 2 is 6.0
```

PASS BY REFRENCE VS VALUE

Inside def function called LOCAL Function

```
[ ]  def change(a_list):
       print("Value of the function is {}".format(a_list))

[ ]  a_list = [1,2]

[ ]  change(a_list)

     Value of the function is [1, 2]
```



```
[ ]  a_list.append([3,4])
     change(a_list)

     Value of the function is [1, 2, [3, 4]]
```

Outside def function (or append) called GLOBAL Function (list in list)

FUNCTION

Argument and Parameter

FUNCTION - ARGUMENT AND PARAMETER

Argument that can be sent to the function

There is 2 argument to call out the function:

1. Keyword Argument
2. Positional Argument

KEYWORD ARGUMENT

The argument who had an identifier or name of parameter explicitly mentioned. including with dictionary started by (**)

```
def package(date, month, year):
  arrive = "Your package will be arrive at {} {} {}".format(date,month,year)
  return arrive
```

```
[ ] package(date=16, month='June', year=2023)
```

```
'Your package will be arrive at 16 June 2023'
```

```
[ ] package(**{'date':13, 'month':'June', 'year':2023})
```

```
'Your package will be arrive at 13 June 2023'
```

POSITIONAL ARGUMENT

when we use an iterable variable, we use positional argument started by (*)

```
def package(date, month, year):
  arrive = "Your package will be arrive at {} {} {}".format(date,month,year)
  return arrive
```

```
[2] package(2,'feb',2022)
```

```
'Your package will be arrive at 2 feb 2022'
```

```
[3] package(*(2,'feb',2022))
```

```
'Your package will be arrive at 2 feb 2022'
```

PARAMETER FUNCTION ORDER

There are another possibility parameter function order according to python:

1. Position-or-keyword : we can type the argument as keyword argument or positional argument.

2. Positional-only : we can set the argument for certain position. Determine with declaration position.

PARAMETER FUNCTION ORDER

example:

```
def trial(*argument, **key_dict):
  for i in argument:
    print("position argument {}".format(i))
  for key, value in key_dict.items():
    print("key argument {}:{}".format(key,value))
```

```
[5]  trial(4)
```

```
position argument 4
```

```
[6]  trial(4, key=5, value=6)
```

```
position argument 4
key argument key:5
key argument value:6
```

ANONYMOUS FUNCTION

1. Anonymous function different than another function who use keyword 'def', but with 'lambda' keyword.
2. A lambda function can take an argument with any amount, but can give one expression value.
3. Lambda function cannot take another command. for example, lambda can't run 'print' argument.
4. Lambda function is a independent function.

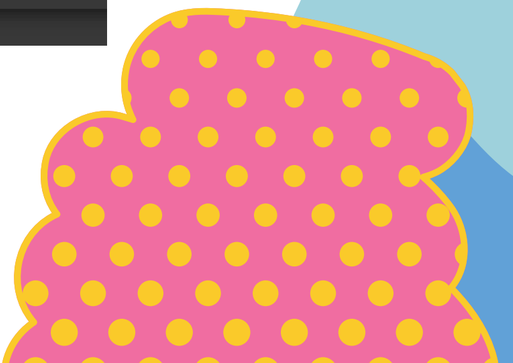
ANONNYMOUS FUNCTION (LAMBDA)

example:

```
multiply = lambda value1, value2: value1*value2
```

```
multiply(3,6)
```

```
18
```



METHOD

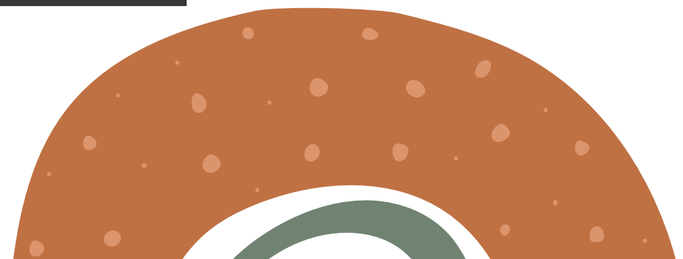
Class, Object and Static Method

CLASS

We make a simple calculator programing project, we use class method like this

```
class Calculator:

  def __init__(self, value1,value2):
    self.value1 = value1
    self.value2 = value2

  def plus(self):
    self.plus = self.value1 + self.value2
    return self.plus

  def minus(self):
    self.min = self.value1 - self.value2
    return self.min
```

```
k = Calculator("value1", "value2")

[8] k = Calculator(7,4)

[9] k.plus()

    11
```

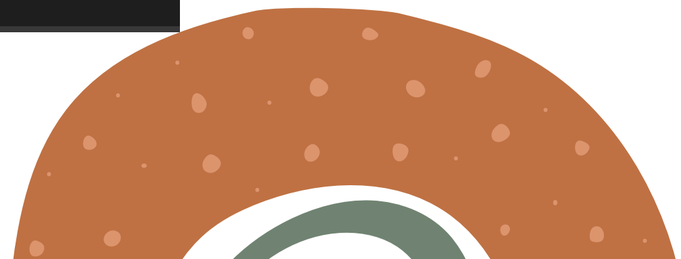


CLASS METHOD

This is how you make a calculator with class method

```
class Calculator:

  def __init__(self, value1,value2):
    self.value1 = value1
    self.value2 = value2

  @classmethod
  def plus_value(cls,value1,value2):
    cls.plus_val = value1 + value2
    return cls.plus_val

  @classmethod
  def minus_value(cls,value1,value2):
    cls.min_val = value1 - value2
    return cls.min_val
```

```
[16] x = Calculator("value1", "value2")

x.plus_value(7,4)

11

[18] x.minus_value(7,4)

3
```

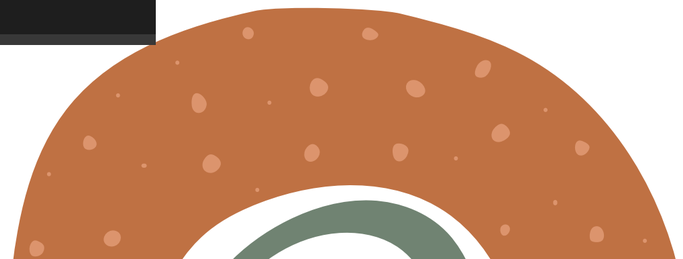


STATIC METHOD

This is how you make a calculator with static method

```
class Calculator:

  def __init__(self, value1,value2):
    self.value1 = value1
    self.value2 = value2

  @staticmethod
  def plus(value1,value2):
    plus_ = value1 + value2
    return plus_

  @staticmethod
  def minus(value1,value2):
    min = value1 - value2
    return min
```

```
[22] y = Calculator.plus(7,4)
     print(y)

     11

     y = Calculator.minus(7,4)
     print(y)

     3
```



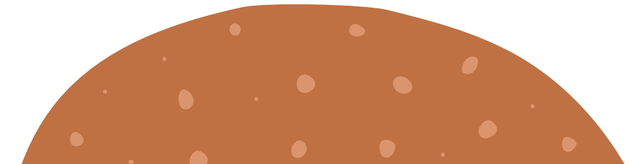
PYDOC

Module in Python

PYDOC

```
#command prompt (Anaconda Prompt)
#c:\users\admin>python -m pydoc pass
#c:\users\admin>python /?

#command line (Python)
#import pydoc
#pydoc.help()
#enter the name of the module, datatype, function, classes, etc.
#help>modules
```

The pydoc module automatically generates documentation from Python modules, comes packaged along with Python, which means you don’t have to download and install it separately.



TXT FILE

Open file in Python

OPEN TXT FILE

The open() function has many parameters but you'll be focusing on the first two {open(path_to_file, mode)}.
The open() function returns a file object which you will use to read text from a text file.

```
#Open a Data Science.txt file for reading by using the open() function
#Mode 'r', Description Open for Data Science.txt file for reading text
workFile = open('Data Science.txt', 'r')
```

TXT FILE

Read file in Python

READ TXT FILE

Read text from the text file using the file read(), readline(), or readlines() method of the file object

```
#Use the read() method to read all the contents of the Data Science.txt file
workFileContents = workFile.read()
print(workFileContents)
```

```
Data Science Track B
- Matplotlib
- Numpy
- Seaborn
- Null
- Code
- Sintaks
- Comment
- Whitespace
- SlcTerm
- Pandas
```

READ TXT FILE

```
#Use the readline() method to read the text file line by line
workFile = open('Data Science.txt', 'r')
workFileLine = workFile.readline()
print(workFileLine)
```

```
Data Science Track B
```

```
#Use the readlines() method to read all the lines of the text file
workFile = open('Data Science.txt', 'r')
workFileLine = workFile.readlines()
print(workFileLine[1])
```

```
- Matplotlib
```

TXT FILE

Close file in Python

CLOSE TXT FILE

The close() method used to close the file that is no longer in use. Always close a file after completing reading it using the close() method or the with statement.

```
#Call the close() method to close the Data Science.txt file
workFile.close()

#To close the file automatically without calling the close() method, use the with statement
with open('Data Science.txt') as workFile:
    workFileContents = workFile.readlines()
```

IO

Input Output Module

IO

Python io module allows us to manage the file-related input and output operations.
This module is quite useful when you want to perform file-related I/O operations (eg. file reading/writing)

```
# importing io module
import io
```

10

```
#Open a My Profile.txt file for writing by using the open() function and with statement
#Mode 'w', Describe Open a text file for writing text
with open('My Profile.txt','w') as writeFile:
  tolog = input('What do you want to write?')
  tolog1 = input('Whats your name?')
  tolog2 = input('Whats your hobby?')
  tolog3 = input('Whats your goals?')
  writeFile.write(tolog)
  writeFile.write('\n' + tolog1 + '\n' + tolog2 + '\n')
  writeFile.write(tolog3)
```

```
What do you want to write?This my profile
Whats your name?My name is Fitri Alfaqrina
Whats your hobby?My hobby is listening music
Whats your goals?My goals is to be a Data Scientist
```

This module gives a lot more flexibility regarding these operations.

OS.PATH

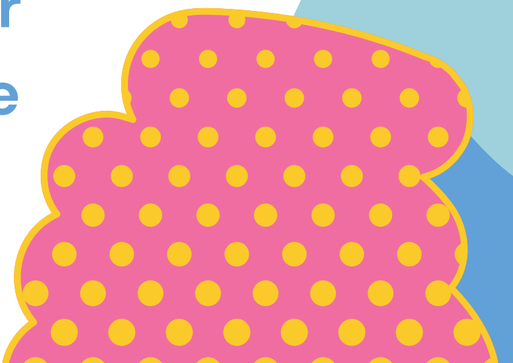
Module in Python

os

os module in Python provides functions for interacting with the operating system.

```
# importing os module
import os
```

The path parameters are either strings or bytes. The result is an object of the same type, if a path or file name is returned.



OS.PATH

```
#os.path.isfile() used to check whether the My Profile.txt is an existing regular file or not.
if os.path.isfile('My Profile.txt'):
#Open a My Profile.txt file for reading by using the open() function
#Mode 'a', Description Open a text file for appending text
  writeFile = open('My Profile.txt', 'a')
else:
#Mode 'w', Description Open a text file for writing text
  writeFile = open('My Profile.txt', 'w')

tolog = input('Whats your hobby?')
tolog1 = input('What do you want to write?')
writeFile.write('\n' + tolog + '\n' + '\n')
writeFile.write(tolog1)
writeFile.close()
```

```
Whats your hobby?Hobby : Listening Music
What do you want to write?See you again and thank you
```

OS.PATH

```
#Open a My Profile.txt file to read all the contents
with open('My Profile.txt') as workFile:
    workFileContents = workFile.read()
    print(workFileContents)
```

```
Full name : Fitri Alfaqrina
Occupation : QC Inspector
Education : Bachelor of Mathematics
Residence : Bojonggede, Bogor
Hobby : Listening Music

See you again and thank you
```

os.path module is sub module of OS module in Python used for common path name manipulation.

TXT FILE

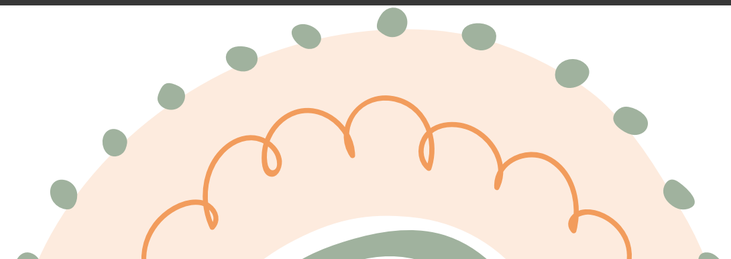
Delete file in Python

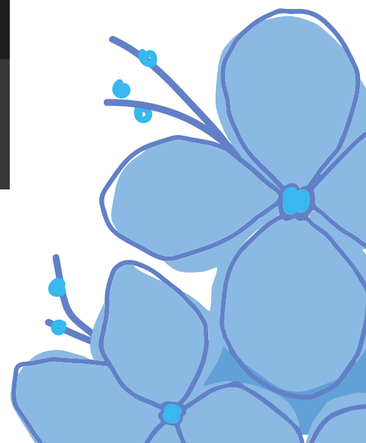
DELETE TXT FILE

os.remove() method in Python is used to remove or delete a file
You must import the OS module, and run its os.remove() function

```
# importing os module
import os

if os.path.isfile('My Profile.txt'):
# Remove the My Profile.txt file
  os.remove('My Profile.txt')
  print('The My Profile file has been removed')
else:
  print('There was no My Profile file to remove')
```

```
There was no My Profile file to remove
```





UNIT TEST

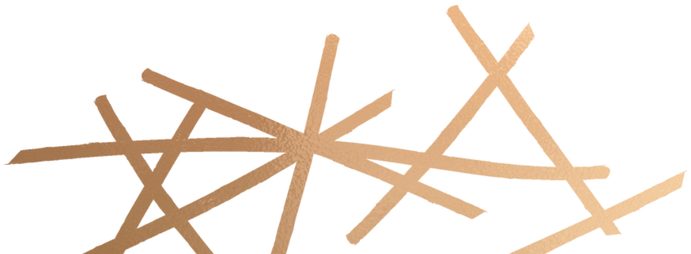
Module in Python

UNIT TEST

```
# importing unittest module
import unittest


#Testing String Methods
Pathway = 'Data Science Track B'
Mentor = 'Kak Suwarti'

# unittest will test all the methods whose name starts with 'test'
class SampleTest(unittest.TestCase):
   # return True or False
   def test(self):
      self.assertTrue(True)
# running the test
unittest.main()
```

```
E
======================================================================
```

The unittest module provides a rich set of tools for constructing and running tests.
The unittest unit testing framework was originally inspired by unit and has a similar flavor as major unit testing frameworks in other languages.
A testcase is created by subclassing unittest.TestCase.



PYTHON OS MODULE

To Create the Directory

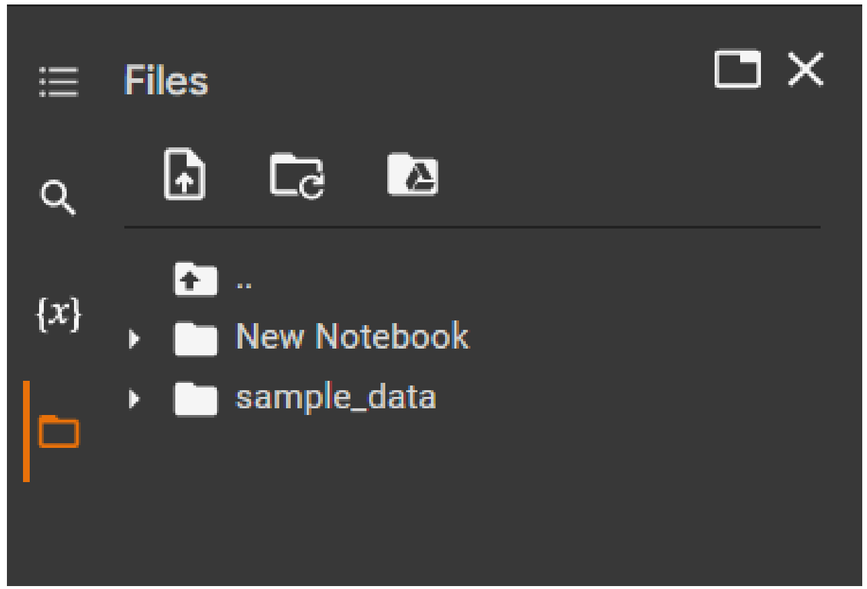
The OS module in Python provides functions for interacting with the operating system.

```
import os
dirName = input("Enter the name of the folder you want to create : ")
os.mkdir(dirName)
print("Directory Created")
```

```
Enter the name of the folder you want to create : New Notebook
Directory Created
```

Python OS Module, include many functions to interact with the file system, one of them is to create a directory in the python by using os.mkdir() method.

The directory that was successfully created will appear in the Files section as follows :



This method raise FileExistsError if the directory to be created already exists

```
import os
dirName2 = input("Enter the name of the folder you want to create : ")
os.mkdir(dirName2)
print("Directory Created")
```

```
Enter the name of the folder you want to create : New Notebook
---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
<ipython-input-6-bd20366791b7> in <module>()
      1 import os
      2 dirName2 = input("Enter the name of the folder you want to create : ")
----> 3 os.mkdir(dirName2)
      4 print("Directory Created")

FileExistsError: [Errno 17] File exists: 'New Notebook'
```

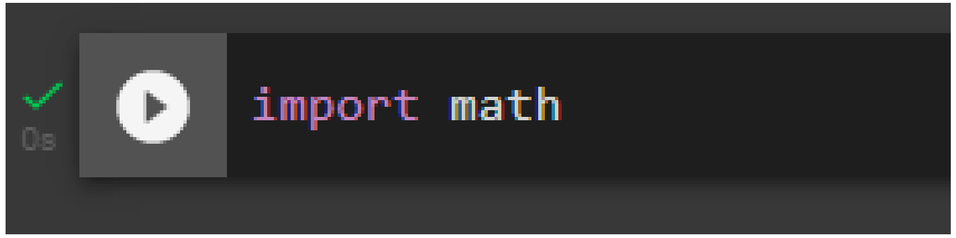
PYTHON MATH MODULE

Built-in Module in Python

Python math module is a standard module in Python that used to deal with mathematical operations. Math module provides the value of various constants and functions to deal with both basic operations such as addition(+), subtraction(-), multiplication(*), division(/) and advance operations like trigonometric, logarithmic, exponential functions.

How to use it ?

To use mathematical functions under this module, you have to import the module using import math.



What we can do with Math Module ?

Call the value of the mathematical constant

Suppose we want to call the value of the mathematical constant pi

```
[ ]  import math
     pi = math.pi
     print(pi)
```

```
3.141592653589793
```

Find the logarithm by using the math module with math.log()

```
[ ]  # returning the log2 of 64
     import math
     print ("The value of log2 of 64 is : ", math.log2(64))
```

```
The value of log2 of 64 is :  6.0
```

Find Ceil Value and Floor Value

```
[ ]  import math

[ ]  #returning the ceil value of pi
     upperBound = math.ceil(pi)
     print(upperBound)

     4

[ ]  #returning the floor value of pi
     lowerBound = math.floor(pi)
     print(lowerBound)

     3
```

Find the square root by using math module with math.sqrt()

```
[ ]  import math
     #returning the square root of 256
     print(math.sqrt(256))

     16.0
```

And doing many other mathematical operations (we can see with dir(math))

```
'acos',
'acosh',
'asin',
'asinh',
'atan',
'atan2',
'atanh',
'ceil',
'copysign',
'cos',
'cosh',
'degrees',
'e',
```

```
'erf',
'erfc',
'exp',
'expm1',
'fabs',
'factorial',
'floor',
'fmod',
'frexp',
'fsum',
'gamma',
'gcd',
```

```
'hypot',
'inf',
'isclose',
'isfinite',
'isinf',
'isnan',
'ldexp',
'lgamma',
'log',
'log10',
'log1p',
'log2',
'modf',
```

```
'nan',
'pi',
'pow',
'radians',
'remainder',
'sin',
'sinh',
'sqrt',
'tan',
'tanh',
'tau',
'trunc']
```

PYTHON DATETIME MODULE

Built-in Module in Python

The datetime module provides classes for manipulating dates and times. These classes provide a number of functions to deal with dates, times and time intervals.

Datetime Class

Datetime is a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo

```
import datetime
todaywithTime = datetime.datetime.today()
print(todaywithTime)
```

```
2022-04-09 16:25:02.141649
```

Date Class

Date class returns the year, month, and date.

```
import datetime
date = datetime.date.today()
print(date)
```

```
2022-04-09
```

```
import datetime
indonesianIndependenceDay = datetime.date(1945, 8, 17)
print(indonesianIndependenceDay)
```

```
1945-08-17
```

There are many other functions and classes in datetime module, such as time class, tzinfo, timezone, timedelta, etc.

PYTHON RANDOM MODULE

Built-in Module in Python

Python Random module is a built-in module of Python which is used to generate random numbers

We can generate random integers between the given range by using randint().

```
from random import randint
randNum = randint(10, 100)
print("Random number between 10 and 100 is % s" %(randNum))
```

```
Random number between 10 and 100 is 68
```

```
#returning the looping of random number between 1 and 199
from random import randint
for i in range(8):
  print(f"Random num {i} is {randint(1,199)}")
```

```
Random num 0 is 189
Random num 1 is 84
Random num 2 is 74
Random num 3 is 6
Random num 4 is 47
Random num 5 is 109
Random num 6 is 140
Random num 7 is 183
```

Note

There are many other functions that can be used in the random module, such as generating random float numbers, selecting random elements, suffling list, etc

PYTHON SYS MODULE

Built-in Module in Python

One of the functions in the sys module that is often used to handling exceptions is exc_info. This function returns a tuple of three values that give information about the exception that is currently being handled

```
import sys
a = float(input("Enter a number "))
b = float(input("Enter a number to divide by "))

try:
  print(f"The answer is {a/b}")
except:
  print(sys.exc_info()[0])
  print("Dind't work. Don't try to divide by zero or something ")
else:
  print("You've successuflly used the division feature in Python")
finally:
  print("Thank you")
```

```
Enter a number 8
Enter a number to divide by 0
<class 'ZeroDivisionError'>
Dind't work. Don't try to divide by zero or something
Thank you
```

The sys module in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment.

It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter

THANKS!!!