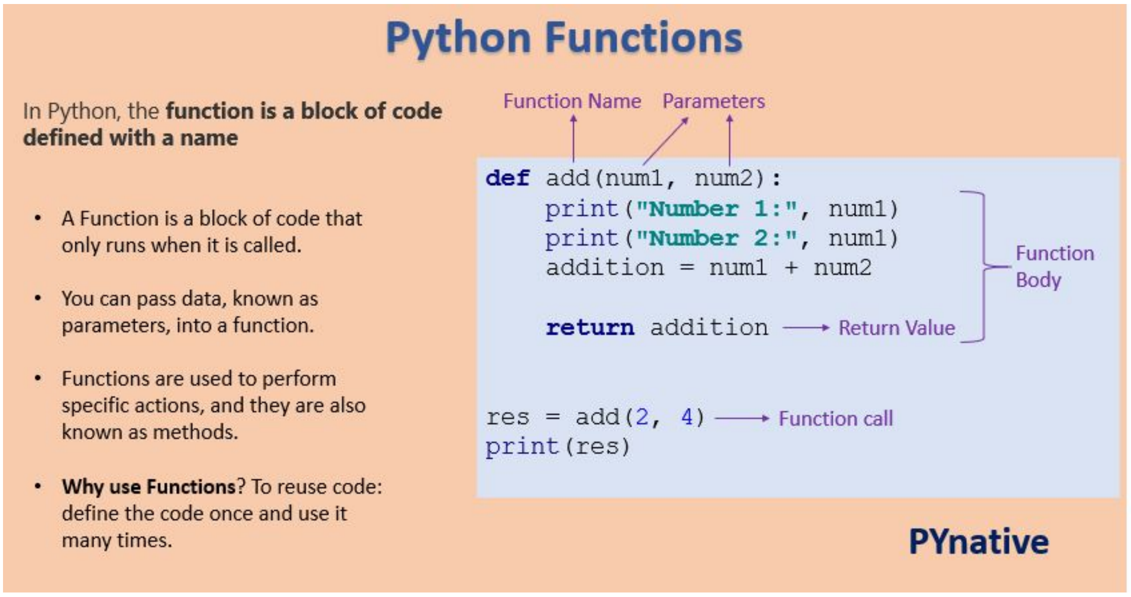
9.Functions in Python



In Python, the **function is a block of code defined with a name**

- A Function is a block of code that only runs when it is called.
- You can pass data, known as parameters, into a function.
- Functions are used to perform specific actions, and they are also known as methods.
- **Why use Functions?** To reuse code: define the code once and use it many times.

```
def add(num1, num2):
    print("Number 1:", num1)
    print("Number 2:", num1)
    addition = num1 + num2

    return addition

res = add(2, 4)
print(res)
```

Syntax:

```
'''
  def fun_name(parameter list):
     function_body
```

def:

- It is a keyword which is used to define function.

fun_name:

- It is the actual name of the function.
- It is also used at the time of calling the function.

Parameter List

- It is the place where we can pass a number of parameter/variable.
- the value of parameter is passed from the calling of function.

Body

- It is the place where the actual code is written to perform the specific task.

Benefits of using a Function

- Code Modularity
- Code Readibility
- Code Reusability

```
# Example definition
def add(): # here `add` is function name
  x= 10
  y=30;
  z= x+y
  print("Add=",z)
  #function calling
add()
```

```
Add= 40
```

```
# Example 2
def greet_user():
 """Display a simple greeting."""
 print("Hello!")
greet_user()

```

```
Hello!
```

Parameters Vs Arguments

Passing information to a function

- nformation that's passed to a function is called an argument;
- information that's received by a function is called a parameter.
- Arguments are included in parentheses after the function's name, and
- parameters are listed in parentheses in the function's definition.

```
# function with parameter
def fruits(name): # here name inside parentheses , is parameter
  print('I like',name)

# function calling
fruits('Mango')
fruits('Orange') # this are arguments inside the parentheses
fruits('Apple')
```

```
I like Mango
I like Orange
I like Apple
```

```
# passing a single argument
def greet_user(username):  # parameter
 """Display a simple greeting."""
 print("Hello, " + username + "!")
greet_user('jesse')
greet_user('diana')  # arguments
greet_user('brandon')

```

```
Hello, jesse!
Hello, diana!
Hello, brandon!
```

```
def fruits(name,fruit):
  print(name,"likes ",fruit)

# fuction calling
fruits('Anjali','Mango')
fruits('Nisha','Orange')
```

```
Anjali likes  Mango
Nisha likes  Orange
```

```

```

Types of Arguments

- Default Argument
- Positional Argument
- Keyword Argument

Function with default parameter value

- We can also assign a default value to a parameter.
- If the function is called without argument,it uses the default value.

```
# Example-1
def fruits(name='Rocky',fruit='Apple'):
  print(name,'likes',fruit)
```

```
# function calling without arguments
fruits()
fruits('salman',)
fruits('dates')
```

```
Rocky likes Apple
salman likes Apple
dates likes Apple
```

Function with return type

- We can also return a value from the function using return type.
- `return` keyword is used to return a value.
- A Funcion stops running when it reaches a return statement.

```
def add(x,y):
  return (x + y)
# function calling
add(5,6) # function return value without print() function
```

```
11
```

```
print("5+6=",add(5,6))
```

```
5+6= 11
```

Positional and keyword arguments

The two main kinds of arguments are positional and keyword arguments:

- When you use positional arguments Python matches the first argument in the function call with the first parameter in the function definition, and so forth.
- With keyword arguments, you specify which parameter each argument should be assigned to in the function call. When you use keyword arguments, the order of the arguments doesn't matter.

```
#Example-1
def power(a=1,b=1):
  return(a**b)
```

```

```

```
power()
power(3,)
```

```
3
```

```
#positional arguments
power(2,3)
```

```
8
```

```
# keyword arguments
power(b=3,a=2)
```

```
8
```

```
# Using positional arguments
def describe_pet(animal, name):
  """ Display information about a pet. """
  print('\nI have a ' + animal + '.')
  print("Its name is " + name + '.')

describe_pet('hamster','harry')
describe_pet('dog','Seru')
```

```

I have a hamster.
Its name is harry.

I have a dog.
Its name is Seru.
```

```
# Using keyword arguments
def describe_pet(animal, name):
  """ Display information about a pet. """
  print('\nI have a ' + animal + '.')
  print("Its name is " + name + '.')

describe_pet(animal='hamster',name='harry')
describe_pet(name='Seru',animal='dog')
```

```
I have a hamster.
Its name is harry.

I have a dog.
Its name is Seru.
```

Passing a List to a function as an Argument

- You can pass a list as an argument to a function, and the function can work with the values in the list

```
def fruits(fruitlist):
  for name in fruitlist:
    print("I like ",name)

mylist=['Mango','Orange','Apple']
# function calling
fruits(mylist)
```

```
I like  Mango
I like  Orange
I like  Apple
```

Passing an Arbitrary number of agrguments

*Sometimes you won't know how many arguments a function will need to accept.

- Python allows you to collect an arbitrary number of arguments into one parameter using the `* operator`.
- if the number of arguments is unknown add asterisk(*) symbol before the parameter name.

```
# *args
# allows us to pass a variable number of non-keyword arguments to a function.

def multiply(*args):
  product = 1

  for i in args:
    product = product * i

  print(args)
  return product
```

```
multiply(1,2,3,4)
```

```
(1, 2, 3, 4)
24
```

```
# **kwargs
# **kwargs allows us to pass any number of keyword arguments.
# Keyword arguments mean that they contain a key-value pair, like a Python dictionary.

def display(**kwargs):

  for (key,value) in kwargs.items():
    print(key,'->',value)
```

```
display(india='delhi',srilanka='colombo',nepal='kathmandu',pakistan='islamabad')
```

```
india -> delhi
srilanka -> colombo
nepal -> kathmandu
pakistan -> islamabad
```

```
def is_even(num):
  """
  This function returns if a given number is odd or even
  input - any valid integer
  output - odd/even
```

```
created on - 16th Nov 2022
"""
if type(num) == int:
  if num % 2 == 0:
    return 'even'
  else:
    return 'odd'
else:
  return 'pagal hai kya?'
```

```
print(is_even(20))
print(is_even(21))
print(is_even('hello'))
```

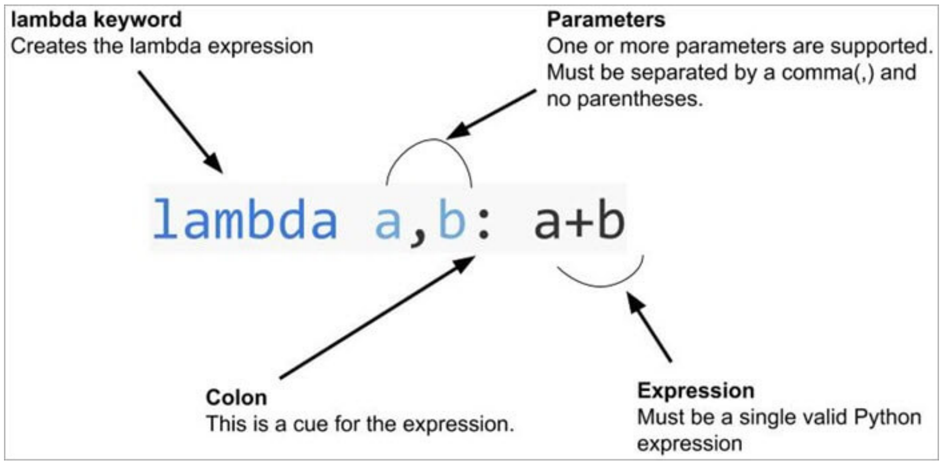
```
even
odd
pagal hai kya?
```

```

```

Lambda Functions in Python

- In Python, a lambda function is a small anonymous function without a name.
- A lambda function can take any number of arguments, but can only have one expression.
- It is defined using the lambda keyword and has the following syntax:

```
'''
lambda arguments: expression
```


lambda keyword
Creates the lambda expression
Parameters
One or more parameters are supported. Must be separated by a comma(,) and no parentheses.
lambda a,b: a+b
Colon
This is a cue for the expression.
Expression
Must be a single valid Python expression

Difference between lambda vs Normal Function

- No name lambda has no return value(infact,returns a function)
- lambda is written in 1 line
- not reusable
- Then why use lambda functions?
- They are commonly used as arguments to higher-order functions,such as `map`, `filter`,and `reduce`.

```
# function to double the input
def double(x):
  return x ** 2
double(3)
```

```
9
```

```
# Lambda function to double the input
double= lambda x: x*2
cube= lambda x: x*x*x
avg= lambda x,y,z: (x+y+z)/3
print(double(5))
print(cube(3))
print(avg(2,4,6))
```

```
10
27
```

```
4.0
```

```python
# function as a arguments inside function
def appl(fx, value):
    return 6 + fx(value)
print(appl(lambda x: x*x, 2))
```

```
10
```

```python
# x,y -> x+y
a = lambda x,y:x+y
a(5,2)
```

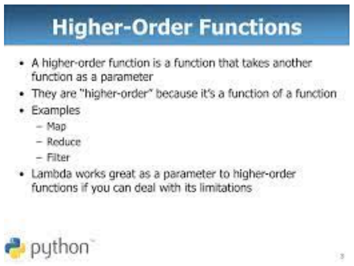
```
7
```

```python
# check if a string has 'a'
a = lambda s:'a' in s
a('hello')
```

```
False
```

```python
# odd or even
a = lambda x:'even' if x%2 == 0 else 'odd'
a(6)
```

```
'even'
```



Higher-Order Functions

- A higher-order function is a function that takes another function as a parameter
- They are "higher-order" because it's a function of a function
- Examples
  - Map
  - Reduce
  - Filter
- Lambda works great as a parameter to higher-order functions if you can deal with its limitations

python

map

- The map function applies a function to each element in a sequence and returns a new sequence containing the transformed elements.
- The map function has the following syntax:

  ```
  '''
  map(function,  iterable)
  ```

- the function task in function that is applied to each element in the iterable argument.
- the iterable argument can be a list, tuple, or any other iterable object.

```python
# normal way to find cube from list iterable using function
def cube(x):
  return x*x*x
l=[1,2,4,6,4,3]
newl=list()
for item in l:
 newl.append(cube(item))
 print(newl)
```

```
[1]
[1, 8]
[1, 8, 64]
[1, 8, 64, 216]
[1, 8, 64, 216, 64]
[1, 8, 64, 216, 64, 27]
```

```python
# HOF
def transform(f,L):
  output = []
  for i in L:
    output.append(f(i))

  print(output)
```

```

L = [1,2,3,4,5]

transform(lambda x:x**3,L)
```

```
[1, 8, 27, 64, 125]
```

```
# square the items of a list
list(map(lambda x:x**2,[1,2,3,4,5]))
```

```
[1, 4, 9, 16, 25]
```

```
# odd/even labelling of list items
L = [1,2,3,4,5]
list(map(lambda x:'even' if x%2 == 0 else 'odd',L))
```

```
['odd', 'even', 'odd', 'even', 'odd']
```

```
# fetch names from a list of dict

users = [
    {
        'name':'Rahul',
        'age':45,
        'gender':'male'
    },
    {
        'name':'Nitish',
        'age':33,
        'gender':'male'
    },
    {
        'name':'Ankita',
        'age':50,
        'gender':'female'
    }
]

list(map(lambda users:users['gender'],users))
```

```
['male', 'male', 'female']
```

filter

- The filter function filters a sequence of elements based on a given predicate (a function that returns a boolean value) and
- returns a new sequence containing only the elements that meet the predicate.
- the filter function has the following syntax:

  ```
  '''
  filter(predicate, iterable)
  ```

- the predicate argument is a function that returns a boolean value and is applied to each element in the iterable argument.
- The iterable argument can be a list, tuple, or any other iterable object.

```
# without lambda function
l=[ 1,2,3,4,6,4,3]
def filter_function(a):
  return a>2
newl=list(filter(filter_function,l))
print(newl)
```

```
[3, 4, 6, 4, 3]
```

```
# using lambda function
lam_list=list(filter(lambda a:a>2,l))
print(lam_list)
```

```
[3, 4, 6, 4, 3]
```

```
# numbers greater than 5
L = [3,4,5,6,7]

list(filter(lambda x:x>5,L))
```

```
[6, 7]
```

```python
# fetch fruits starting with 'a'
fruits =['apple','guava','cherry']
list(filter(lambda x:x.startswith('a'), fruits))
```

```
['apple']
```

```python
# list of numbers
numbers = [1,2,3,4,5]

# get only the even numbers using the filter function
evens= filter(lambda x: x%2 == 0, numbers)

# print the even numbers
print(list(evens))
```

```
[2, 4]
```

`reduce`

- The reduce function is a higher-order function that applies a function to a sequence and returns a single value.
- it is a part of the functlools module in Python and has the following syntax:

  ```
  '''

  reduce(function, iterable)
  ```

- the reduce function applies to the first two elements in the iterable and then applies the function to the result and the next element, and so on.
- the reduce function returns the final result.

```python
from functools import reduce

# list of numbers
numbers = [1,2,3,4,5]
```

```python
# calculate the sum of the numbers using the reduce function without lambda function
def mysum(x, y):
  return x + y

sum = reduce(mysum, numbers)

# print the sum
print(sum)
```

```
15
```

```python
# using lambda function
sum = reduce(lambda x, y: x + y, numbers)
print(sum)
```

```
15
```

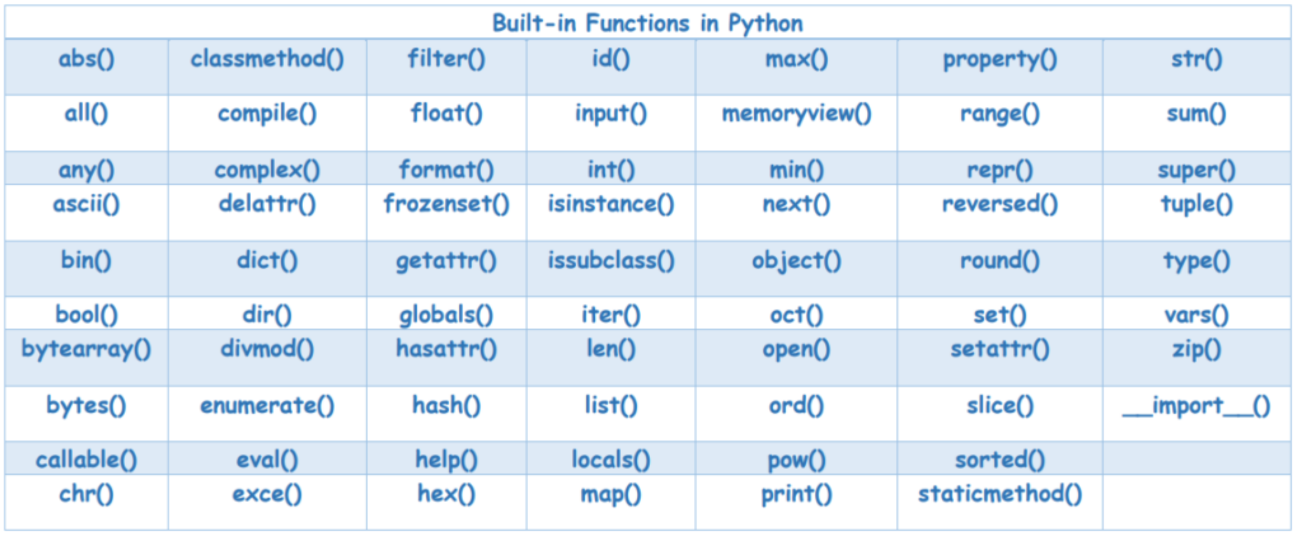
```python
# find max
reduce(lambda x,y:x if x>y else y,[23,11,45,10,1])
```

```
45
```

Built-in Functions

- The Python built-in functions are defined as the functions whose functionality is pre-defined in Python.

- There are several built-in Functions in Python which are listed below:



Built-in Functions in Python						
abs()	classmethod()	filter()	id()	max()	property()	str()
all()	compile()	float()	input()	memoryview()	range()	sum()
any()	complex()	format()	int()	min()	repr()	super()
ascii()	delattr()	frozenset()	isinstance()	next()	reversed()	tuple()
bin()	dict()	getattr()	issubclass()	object()	round()	type()
bool()	dir()	globals()	iter()	oct()	set()	vars()
bytearray()	divmod()	hasattr()	len()	open()	setattr()	zip()
bytes()	enumerate()	hash()	list()	ord()	slice()	__import__()
callable()	eval()	help()	locals()	pow()	sorted()	
chr()	exce()	hex()	map()	print()	staticmethod()	

```python
# python abs() Function Example
# integer number
integer = -20
print('Absolute value of -40 is:', abs(integer))
```

```
Absolute value of -40 is: 20
```

```python
# python all() function
# It returns true if all items in passed iterable are true.Otherwise, it returns False
val_list=[1,4,5,3]
print(all(val_list))
print()
# all values false
values=[0,False]
print(all(values))
```

```
True

False
```

```python
# Python bin() Function
# bin() function is used to return the binary representation of a specified integer
# a result always starts with the prefix 0b
x = 10
y = bin(x)
print(y)
```

```
0b1010
```

```python
# python bytes()
# python bytes() in python is used for returning a bytes object.
# It is an immutable version of the bytearray() function
string ="Hello World."
array = bytes(string,'utf-8')
print(array)
```

```
b'Hello World.'
```

```python
# Python sum() Function
# it is used to get the sum of numbers of an iterable,i.e, list
s = sum([1,2,4])
print(s)
```

```
7
```

```python

```

Modules in Python

- A module is a collection of statements in which we store functions, classes and variables.
- Python module acts as a library in which we can store set of functions and can use as per our requirements.

create Module

- now i am going to create a module with name mymodule.py that contain a function add() that prints the sum of two given number.

```
# creating function
def add(x,y):
  print("Sum=",x+y)
```

Include Module in our Python Code

There are two way of including module in our python code:

- import statement
- from import statement

import statement

```
syntax:
       '''
       import module1, module2,.......
```

- we can import multiple module in a single statements.
- The import statements includes all the functionality of one module into another.

from import statement

- this statement is used when there is a need to include only the specific attributes of a module.

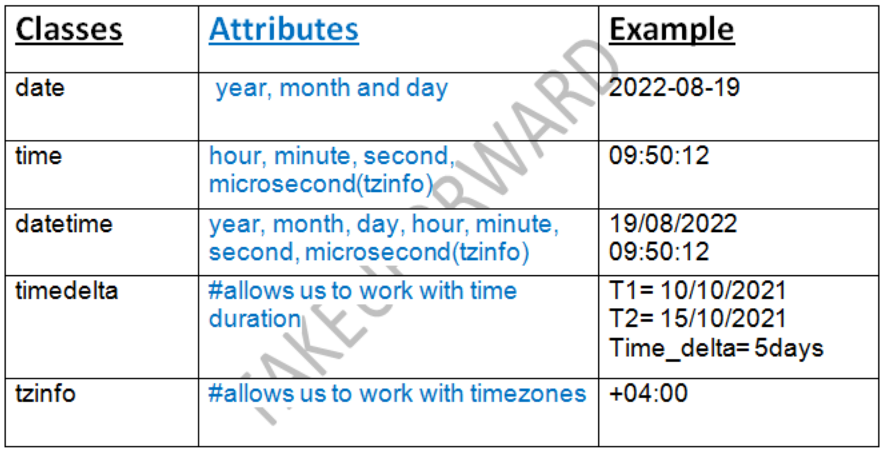
  ```
  syntax:
        '''
        from `name_of_module` import `name_of_specific_attribute`
  ```

important python modules

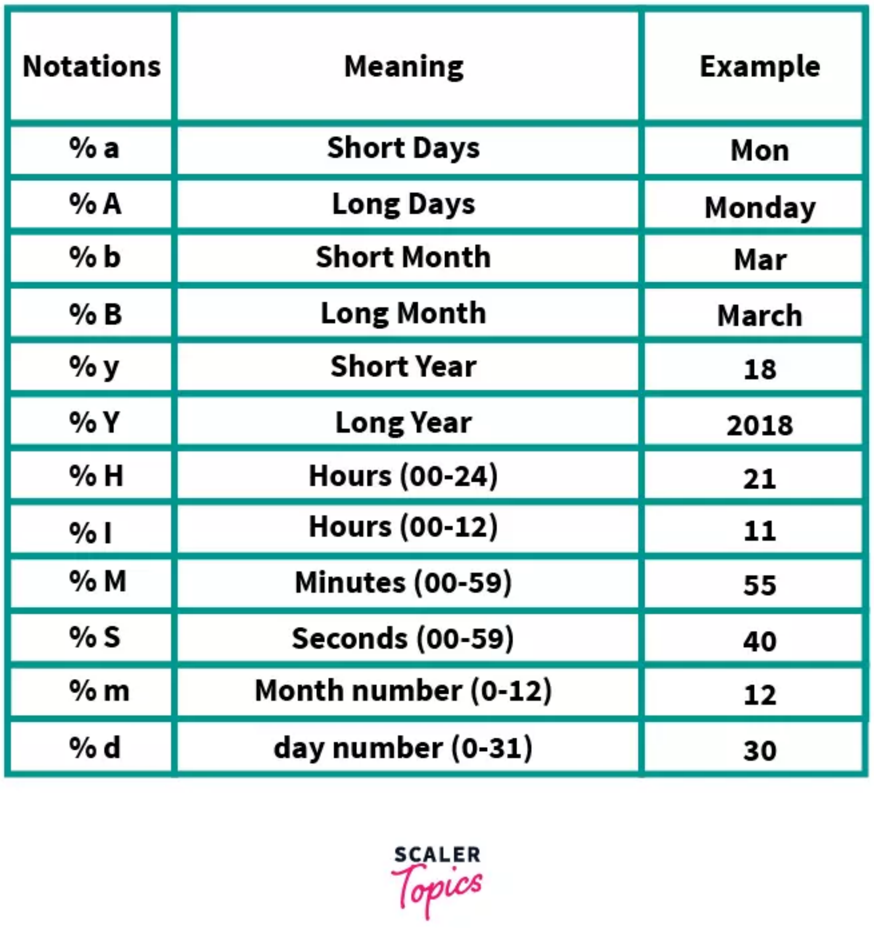
- Datetime https://www.programiz.com/python-programming/datetime
- math
- os
- Regular expression https://www.programiz.com/python-programming/regex or https://regexr.com/
- calendar etc.

Date and Time

- by using datetime, date and time module we can get current date and time in very simple way.
- there are many way to print data and time in python i am trying to explain in simple way



Classes	Attributes	Example
date	year, month and day	2022-08-19
time	hour, minute, second, microsecond(tzinfo)	09:50:12
datetime	year, month, day, hour, minute, second, microsecond(tzinfo)	19/08/2022 09:50:12
timedelta	#allows us to work with time duration	T1= 10/10/2021 T2= 15/10/2021 Time_delta= 5days
tzinfo	#allows us to work with timezones	+04:00



Notations	Meaning	Example
% a	Short Days	Mon
% A	Long Days	Monday
% b	Short Month	Mar
% B	Long Month	March
% y	Short Year	18
% Y	Long Year	2018
% H	Hours (00-24)	21
% I	Hours (00-12)	11
% M	Minutes (00-59)	55
% S	Seconds (00-59)	40
% m	Month number (0-12)	12
% d	day number (0-31)	30

Example- 1 Get current Date and Time

```
import datetime

# get the current date and time
now = datetime.datetime.now()

print(now)
```

```
2023-09-07 01:49:19.398418
```

```
# Example -2 get current date
import datetime

# get current date
current_date = datetime.date.today()
print(current_date)
```

```
2023-09-07
```

```python
# Example - 3 Date object to represent a date
import datetime
d = datetime.date(2022, 12, 25)
print(d)
```

```
2022-12-25
```

```python
# get current date using today()
from datetime import date
 # today() to get current date
todays_date = date.today()

print("Today's date =", todays_date)
```

```
Today's date = 2023-09-07
```

`timestamp`()

A Unix timestamp is the number of seconds between a particular date and January 1, 1970 at UTC. You can convert a timestamp to date using the fromtimestamp() method.

```python
from datetime import date

timestamp = date.fromtimestamp(132244366)

print("Date =", timestamp)
```

```
Date = 1974-03-11
```

```python
# Print today's year, month and day
from datetime import date

# date object of today's date
today = date.today()

print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```

```
Current year: 2023
Current month: 9
Current day: 7
```

Python datetime.time Class

A time object instantiated from the time class represents the local time.

```python
# Time object to represent time
from datetime import time

# time(hour = 0, minute = 0, second = 0)
a = time()
print(a)

# time(hour, minute and second)
b = time(11, 34, 56)
print(b)

# time(hour, minute and second)
c = time(hour = 11, minute = 34, second = 56)
print(c)

# time(hour, minute, second, microsecond)
d = time(11, 34, 56, 234566)
print(d)
```

```
00:00:00
11:34:56
11:34:56
11:34:56.234566
```

```python
#Print hour, minute, second and microsecond
from datetime import time

```

```
a = time(11, 34, 56)

print("Hour =", a.hour)
print("Minute =", a.minute)
print("Second =", a.second)
print("Microsecond =", a.microsecond)
```

```
Hour = 11
Minute = 34
Second = 56
Microsecond = 0
```

The datetime.datetime Class

The `datetime` module has a class named datetime that can contain information from both `date` and `time` objects.

```
# Print year, month, hour, minute and timestamp
from datetime import datetime

a = datetime(2022, 12, 28, 23, 55, 59, 342380)

print("Year =", a.year)
print("Month =", a.month)
print("Hour =", a.hour)
print("Minute =", a.minute)
print("Timestamp =", a.timestamp())
```

```
Year = 2022
Month = 12
Hour = 23
Minute = 55
Timestamp = 1672271759.34238
```

Python datetime.timedelta Class

A `timedelta` object represents the difference between two `dates` or `times`. For example,

```
from datetime import datetime, date

# using date()
t1 = date(year = 2018, month = 7, day = 12)
t2 = date(year = 2017, month = 12, day = 23)

t3 = t1 - t2

print("t3 =", t3)

# using datetime()
t4 = datetime(year = 2018, month = 7, day = 12, hour = 7, minute = 9, second = 33)
t5 = datetime(year = 2019, month = 6, day = 10, hour = 5, minute = 55, second = 13)
t6 = t4 - t5
print("t6 =", t6)

print("Type of t3 =", type(t3))
print("Type of t6 =", type(t6))
```

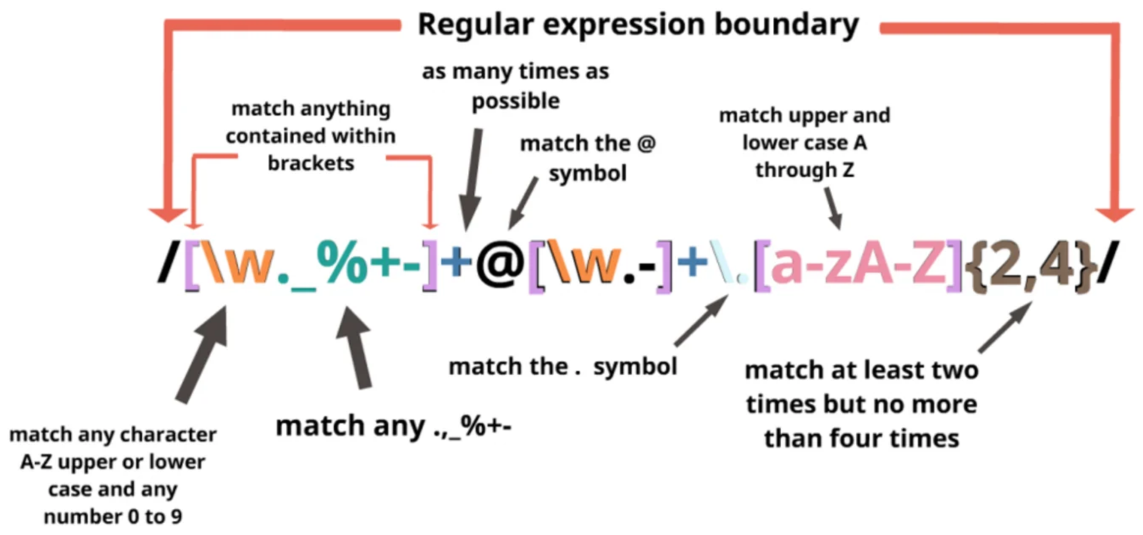
```
t3 = 201 days, 0:00:00
t6 = -333 days, 1:14:20
Type of t3 = <class 'datetime.timedelta'>
Type of t6 = <class 'datetime.timedelta'>
```

```
#Difference between two timedelta objects
from datetime import timedelta

t1 = timedelta(weeks = 2, days = 5, hours = 1, seconds = 33)
t2 = timedelta(days = 4, hours = 11, minutes = 4, seconds = 54)

t3 = t1 - t2

print("t3 =", t3)
```

```
t3 = 14 days, 13:55:39
```

Regular Expressions in Python

- Regular expression, or "regex" for short, are a powerful tool for working with strings and text data in Python.
- They allow you to match and manipulate strings based on patterns
- making it easy to perform complex string operations with just a few lines of code


Regular expression boundary
as many times as possible
match anything contained within brackets
match the @ symbol
match upper and lower case A through Z
/[\w._%+-]+@[\w.-]+\.[a-zA-Z]{2,4}/
match the . symbol
match at least two times but no more than four times
match any ._%+-
match any character A-Z upper or lower case and any number 0 to 9

Metacharacters in reagular expressions

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

```
from ast import pattern
# match the pattern against a string
import re
text = "Hello, World!"
pattern = r"Hello"

match = re.search(pattern, text)

if match:
  print("Match found")
```

```
else:
```

```
Match found
```

```python
import re
pattern = r"in"
text = "The cat is in the hat."

matches = re.findall(pattern, text)

print(matches)
```

```
['in']
```

```python
import re
pattern = r"[a-z]+at"
text = "The cat is in the hat."

matches = re.findall(pattern, text)

print(matches)
```

```
['cat', 'hat']
```

```python
import re

text = "Hello, this is a sample text with some words."
pattern = r"\bHell\b"

match = re.search(pattern, text)
if match:
    print("Word 'Hello' found!")
else:
    print("Word 'Hello' not found.")
```

```
Word 'Hello' not found.
```

```python
import re

text = "Contact us at support@example.com or info@domain.com"
pattern = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"

emails = re.findall(pattern, text)
print("Email addresses found:", emails)

```

```
Email addresses found: ['support@example.com', 'info@domain.com']
```

```python
import re

text = "The price of the product is $99.99 and the quantity is 25."
pattern = r"\d+\.\d+|\d+"

numbers = re.findall(pattern, text)
print("Numbers found:", numbers)
```

```
Numbers found: ['99.99', '25']
```

```python
import re

text = "Python is a popular programming language. I love python!"
pattern = r"\bpython\b"

replaced_text = re.sub(pattern, "Java", text, flags=re.IGNORECASE)
print(replaced_text)
```

```
Java is a popular programming language. I love Java!
```

```python
import re

text = "Apple,Orange,Mango,Banana"
pattern = r","

fruits = re.split(pattern, text)
print("Fruits:", fruits)
```

```
Fruits: ['Apple', 'Orange', 'Mango', 'Banana']
```

```
import re

text = "My phone number is (123) 456-7890."
pattern = r"\((\d{3})\)\s(\d{3})-(\d{4})"

match = re.search(pattern, text)
if match:
    area_code, first_part, second_part = match.groups()
    print(f"Area Code: {area_code}, First Part: {first_part}, Second Part: {second_part}")
```

```
Area Code: 123, First Part: 456, Second Part: 7890
```

Class and Object

Object

- Object is an instance of a class.
- Object is comprises both data members and methods.

Object is having states and behaviors in which

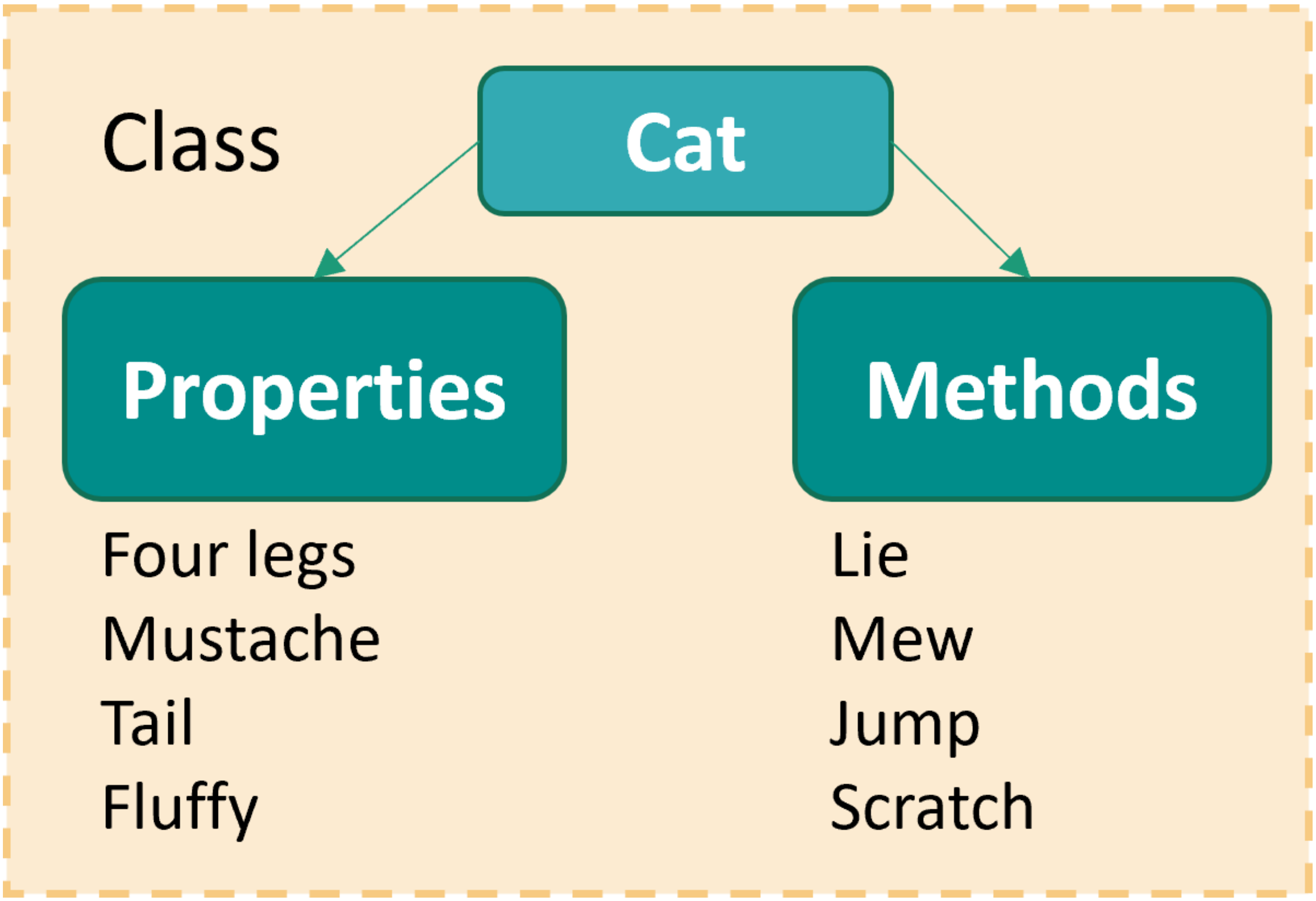
- state means what does it has,
- behavior means what does it do.

for example a Pen has

- States: ink, nib, needle ,cap etc.
- behaviors: Writing
- object is created from a class.

Class

- It is a collection of data members and member functions.
- Data members are the variable used inside class.
- Member functions are the function used inside class
- It is also called Userdefined data type
- class is keyword is used to create class



syntax:

```
'''
(keyword) -> Class ClassName:----->(user defined class name)
                       ( variables)
                   Data member 1

                   (methods)
                   Member function 1
```

Double-click (or enter) to edit

```python
# Creating a class
class Student:

  # defining the class variables
  name = "Rocky"
  roll = 205

 # creating object
s1 = Student()

 # accessing class variable
print('Name:', s1.name)
print('Rollno:', s1.roll)

```

```
Name: Rocky
Rollno: 205
```

```python
# Creating a class
class Student:
    # Defining class variables
    name = "Rocky"
    roll = 205
    marks = 85.6

    # Defining a function
    def ShowInfo(self):
        print('Name:', self.name)
        print('Rollno:', self.roll)
        print('Marks:', self.marks)

# Creating a Student object
s1 = Student()

# Calling the function of the class
s1.ShowInfo()

```

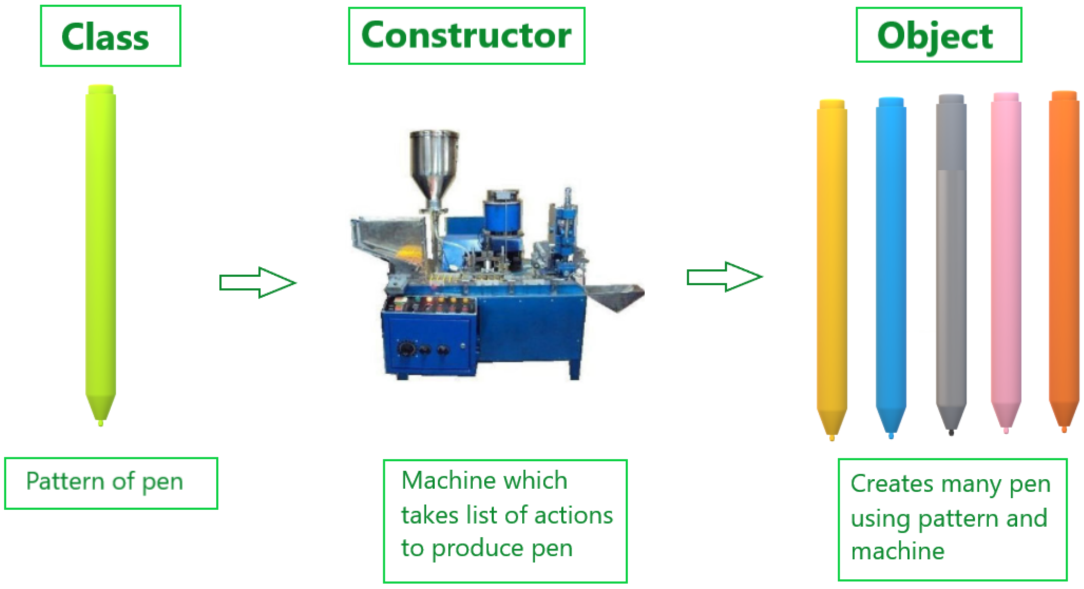
```
Name: Rocky
Rollno: 205
Marks: 85.6
```

```

```

Constructor

- It is a special member function of class that executes when we create the instance (object) of that class
- In other word we can say that there is no need to call a constructor.
- **init**() method is used as a constructor in Python.
- There are two types of constructor are used in Python:

  ```
  * Parameterized Constructor
  * Non-parameterized Constructor
  ```


Class
Constructor
Object
Pattern of pen
Machine which takes list of actions to produce pen
Creates many pen using pattern and machine

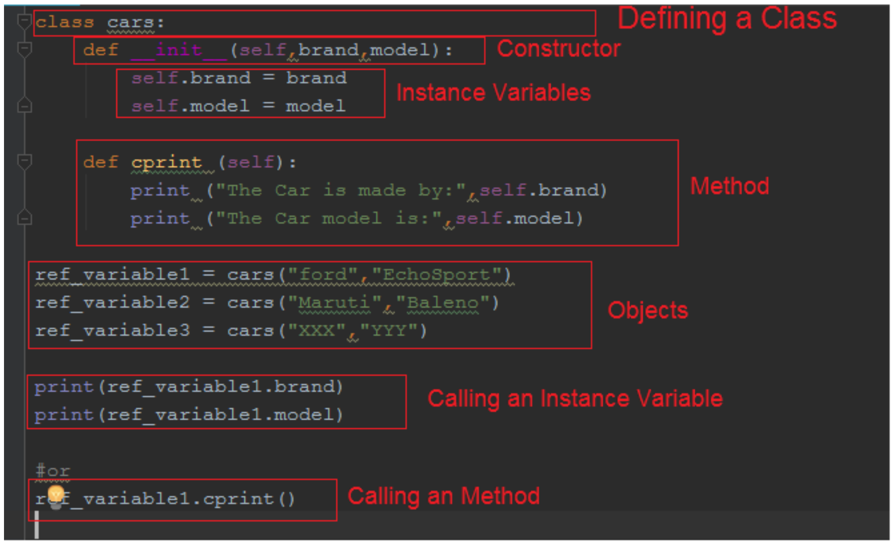
```python
# The constructor with parameter is called Parameterized Constructor
# Creating class
class Stuedent:

# creating parameterized constructor
    def __init__(self, name, roll):
       print("Name:", name)
       print("Rollno:", roll)

# creating object
s1 = Student("Aayushi", 305)

```

```
Name: Aayushi
Rollno: 305
```

```python
# Non-parameterized constructor'
# the constructor with no parameter is called Non-parameterized Constructor

#creating class
class Student:
    def __init__(self):
        print("Hi I am non-parameterized Constructor")
    def Info(slef):
        print("Name: Aayushi")
        print("Rollno: 305")

#creating object
s1 = Student()
s1.Info()
```

```
Hi I am non-parameterized Constructor
Name: Aayushi
Rollno: 305
```

```

```



```
class cars:
    def __init__(self,brand,model):
        self.brand = brand
        self.model = model

    def cprint (self):
        print ("The Car is made by:",self.brand)
        print ("The Car model is:",self.model)

ref_variable1 = cars("ford","EchoSport")
ref_variable2 = cars("Maruti","Baleno")
ref_variable3 = cars("XXX","YYY")

print(ref_variable1.brand)
print(ref_variable1.model)

#or
ref_variable1.cprint()
```

Self parameter

- The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self, you can call it whatever you like, but it has to be the first parameter of any function in the class.

Check the following example:

```
# Function init()
class Data:
 def __init__(self, euler_number, pi_number, golden_ratio):
   self.euler_number = euler_number
   self.pi_number = pi_number
   self.golden_ratio = golden_ratio
val = Data(2.718, 3.14, 1.618)
print(val.euler_number)
print(val.golden_ratio)
print(val.pi_number)

```

```
2.718
1.618
3.14
```

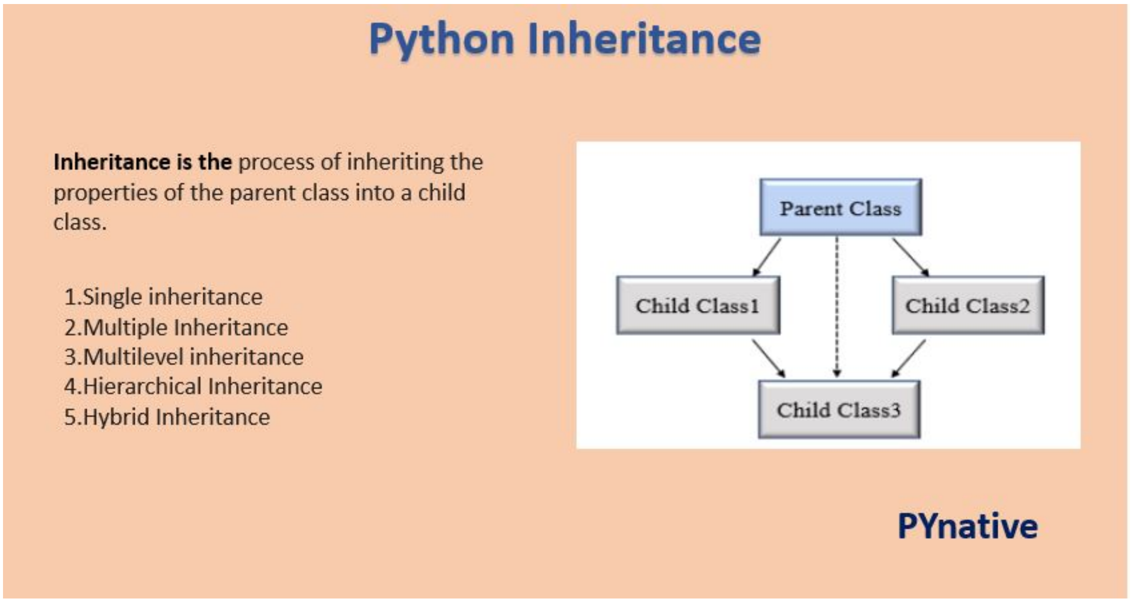
```
# Methods
class Data:
 def __init__(self, euler_number, pi_number, golden_ratio):
  self.euler_number = euler_number
  self.pi_number = pi_number
  self.golden_ratio = golden_ratio
 def msg_function(self):
  print("The euler number is", self.euler_number)
  print("The golden ratio is", self.golden_ratio)
  print("The pi number is", self.pi_number)
val = Data(2.718, 3.14, 1.618)
val.msg_function()

```

```
The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14
```

```
"""
The following codes are the same as the above codes under the title 'Methods'.
You see that the output is the same, but this codes contain 'classFirstParameter' instead of 'self'.
"""
class Data:
 def __init__(classFirstParameter, euler_number, pi_number, golden_ratio):
  classFirstParameter.euler_number = euler_number
  classFirstParameter.pi_number = pi_number
  classFirstParameter.golden_ratio = golden_ratio
 def msg_function(classFirstParameter):
  print("The euler number is", classFirstParameter.euler_number)
  print("The golden ratio is", classFirstParameter.golden_ratio)
  print("The pi number is", classFirstParameter.pi_number)
val = Data(2.718, 3.14, 1.618)
```

```
val.msg_function()
```

```
The euler number is 2.718
The golden ratio is 1.618
The pi number is 3.14
```



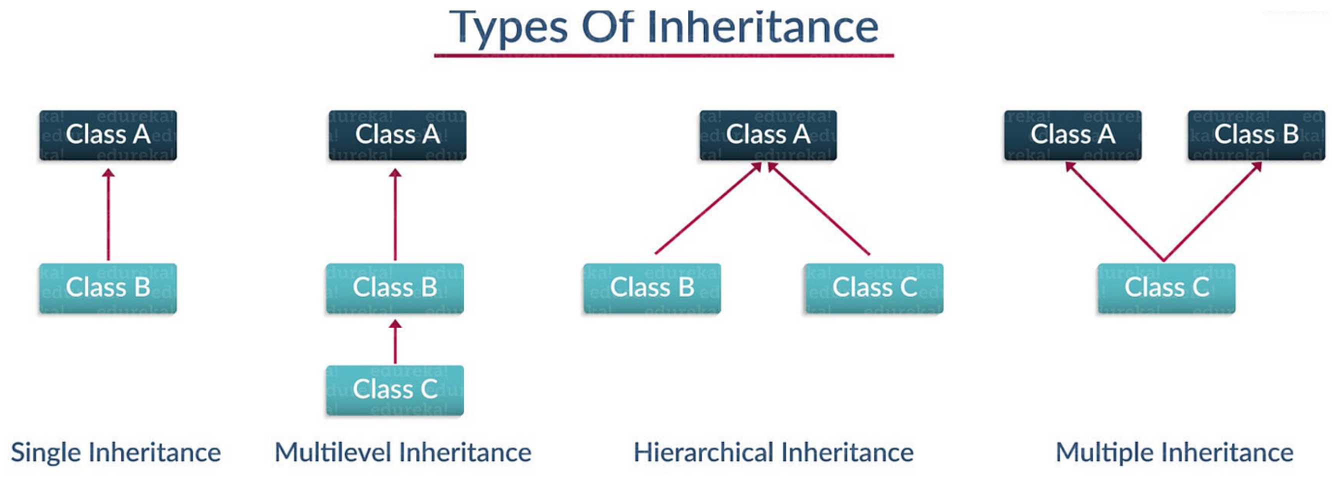
Python Inheritance

Inheritance is the process of inheriting the properties of the parent class into a child class.

1.Single inheritance
2.Multiple Inheritance
3.Multilevel inheritance
4.Hierarchical Inheritance
5.Hybrid Inheritance

Parent Class
Child Class1
Child Class2
Child Class3
PYnative

- The process of deriving a new class from an old class is called `inheritance`
- in which , the new class is called derived or child or sub-class and
- old class is called Base or Parent or Super class



OOPS task

1) Write a Rectangle class in Python language, allowing you to build a rectangle with length and width attributes.

```
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

    def is_square(self):
        return self.length == self.width

    def __str__(self):
        return f"Rectangle with length {self.length} and width {self.width}"


```

```python
# Example usage:
rect1 = Rectangle(5, 4)
print(rect1)
print(f"Area: {rect1.area()}")
print(f"Perimeter: {rect1.perimeter()}")
print(f"Is it a square? {rect1.is_square()}")
```

```
Rectangle with length 5 and width 4
Area: 20
Perimeter: 18
Is it a square? False
```

Double-click (or enter) to edit

```
1
```

syntax:

```
'''
class BaseClass:
      Body of base class
class DerivedClass(BaseClass):
      Body of derived class
```

Single Inheritance

In single inheritance, a child class inherits from a single-parent class. Here is one child class and one parent class.

```python
# Base class
class Vehicle:
   def Vehicle_info(self):
       print('Inside Vehicle class')

# Child class
class Car(Vehicle):
   def car_info(self):
       print("Inside Car class")

# Create object of Car
car = Car()

# access vehicle's info using car object
car.Vehicle_info()
car.car_info()
```

```
Inside Vehicle class
Inside Car class
```

```
1
```

Multiple Inheritance

In multiple inheritance, one child class can inherit from multiple parent classes. So here is one child class and multiple parent classes.

```python
# Parent class 1
class Person:
  def person_info(self, name, age):
    print("inside Person class")
    print("Name:", name, 'Age:',age)

# Parent class 2
class Company:
  def company_info(self, company_name, location):
    print('Inside Company class')
    print('Name:', company_name, 'location:', location)

# child class
class Employee(Person, Company):
  def Employee_info(self, salary, skill):
    print('Inside Employee class')
    print('Salary:', salary, 'Skill:', skill)

# create object of Employee
emp = Employee()

```

```
# access data
emp.person_info('Jessa', 28)
emp.company_info('Google', 'Atlanta')
emp.Employee_info(12000, 'Machine Learning')
```

```
inside Person class
Name: Jessa Age: 28
Inside Company class
Name: Google location: Atlanta
Inside Employee class
Salary: 12000 Skill: Machine Learning
```

Multilevel inheritance

In multilevel inheritance, a class inherits from a child class or derived class.

- Suppose three classes A, B, C. A is the superclass, B is the child class of A, C is the child class of B. In other words, we can say a chain of classes is called multilevel inheritance.

```
# Base class
class Vehicle:
  def Vehicle_info(self):
    print('Inside vehicle class')

# Child class
class Car(Vehicle):
  def car_info(self):
    print('Inside Car class')

# Child class
class SportsCar(Car):
  def sports_car_info(self):
    print('Inside SportsCar class')

# create object of SportsCar
s_car = SportsCar()

# acess Vehicles  and Car info using SportsCar object
s_car.Vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```

```
Inside vehicle class
Inside Car class
Inside SportsCar class
```

Hierarchical Inheritance

In Hierarchical inheritance, more than one child class is derived from a single parent class.

- In other words, we can say one parent class and multiple child classes

```
class Vehicle:
    def info(self):
        print("This is Vehicle")

class Car(Vehicle):
    def car_info(self, name):
        print("Car name is:", name)

class Truck(Vehicle):
    def truck_info(self, name):
        print("Truck name is:", name)

obj1 = Car()
obj1.info()
obj1.car_info('BMW')

obj2 = Truck()
obj2.info()
obj2.truck_info('Ford')
```

```
This is Vehicle
Car name is: BMW
```

```
This is Vehicle
Truck name is: Ford
```

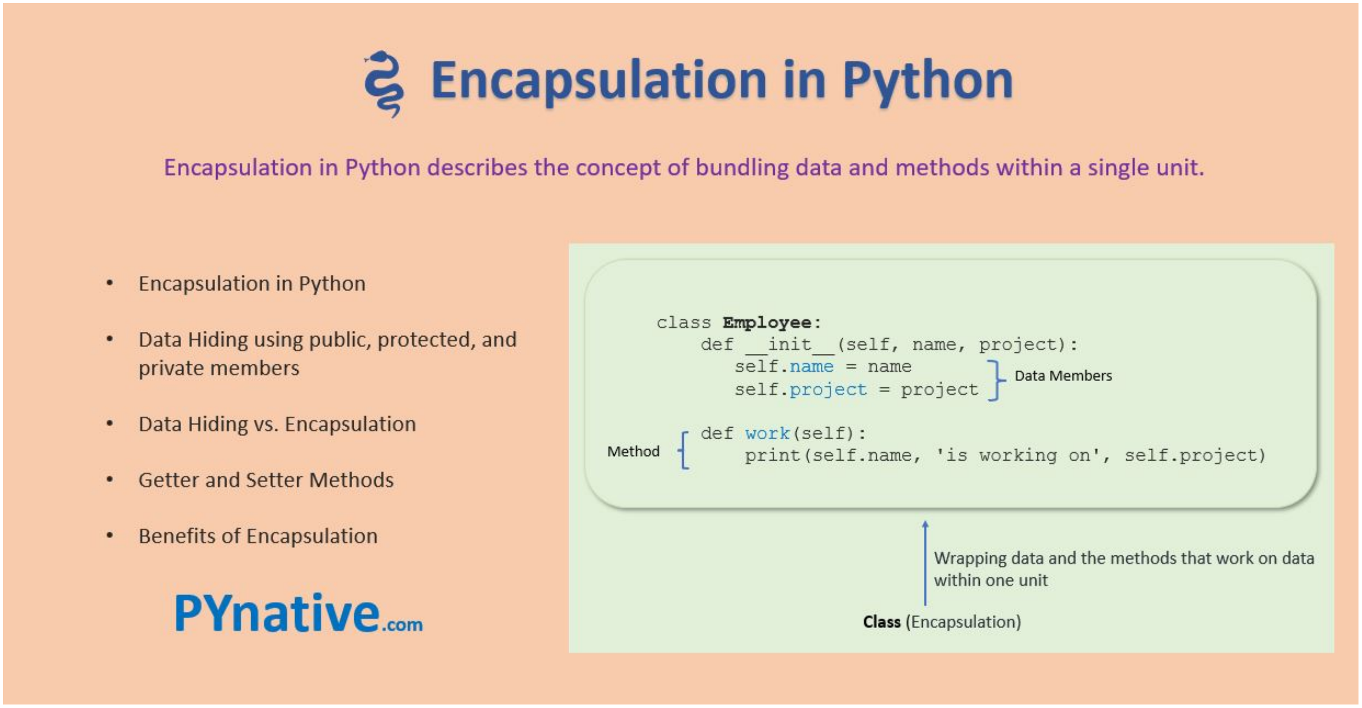
Hybrid Inheritance

When inheritance is consists of multiple types or a combination of different inheritance is called hybrid inheritance.

link -https://pynative.com/python-class-method/

```python
class Vehicle:
    def vehicle_info(self):
        print("Inside Vehicle class")

class Car(Vehicle):
    def car_info(self):
        print("Inside Car class")

class Truck(Vehicle):
    def truck_info(self):
        print("Inside Truck class")

# Sports Car can inherits properties of Vehicle and Car
class SportsCar(Car, Vehicle):
    def sports_car_info(self):
        print("Inside SportsCar class")

# create object
s_car = SportsCar()

s_car.vehicle_info()
s_car.car_info()
s_car.sports_car_info()
```

```
Inside Vehicle class
Inside Car class
Inside SportsCar class
```

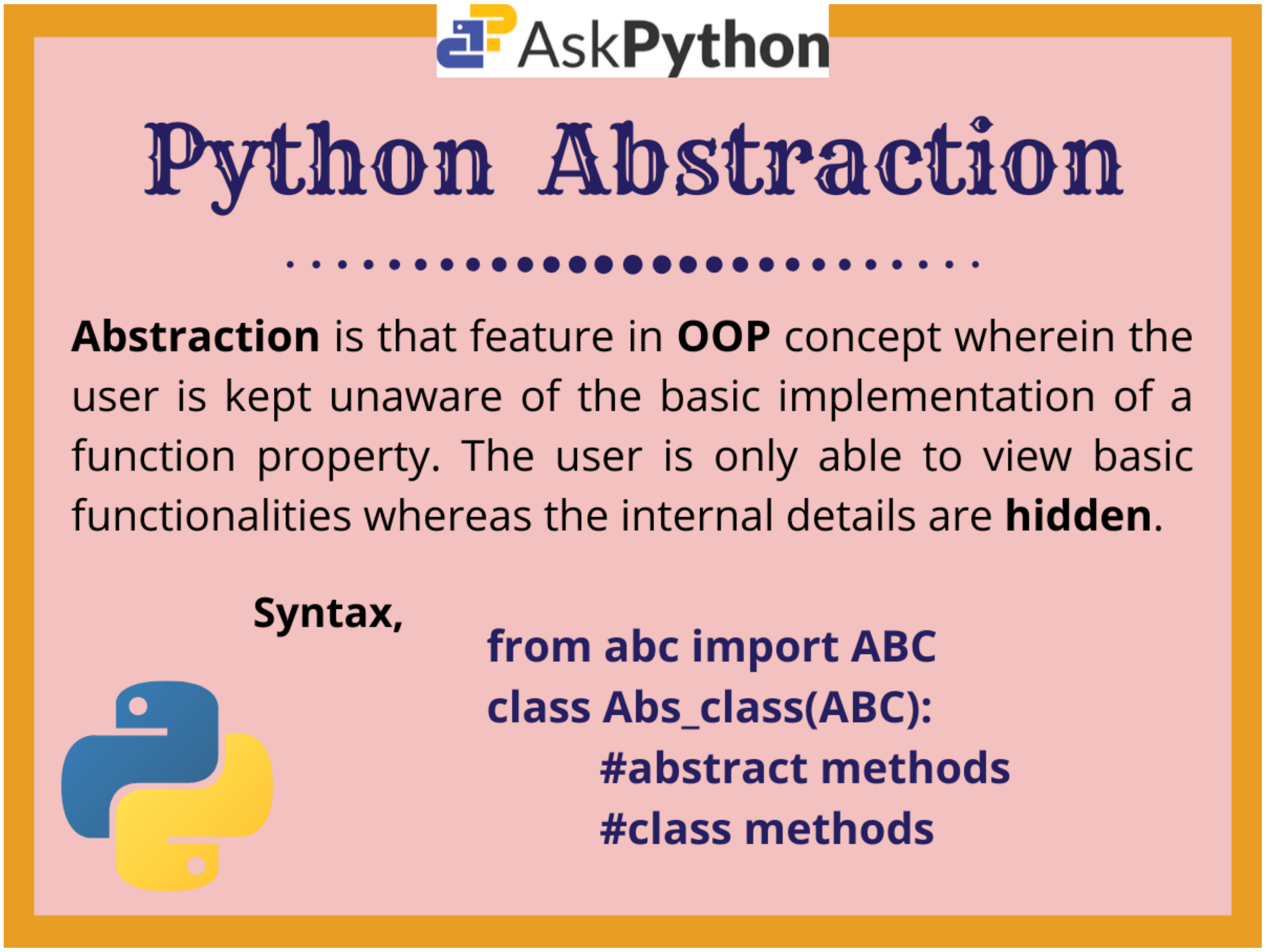

Encapsulation in Python
Encapsulation in Python describes the concept of bundling data and methods within a single unit.
Encapsulation in Python
Data Hiding using public, protected, and private members
Data Hiding vs. Encapsulation
Getter and Setter Methods
Benefits of Encapsulation
PYnative.com
Data Members
Method
Wrapping data and the methods that work on data within one unit
Class (Encapsulation)

```python
class Employee:
    def __init__(self, name, project):
        self.name = name
        self.project = project

    def work(self):
        print(self.name, 'is working on', self.project)
```

link-https://pynative.com/python-encapsulation/

Data Abstraction

- Abstraction means Data hiding , in other word we can say that in this type of programming essential data is shown to the user or outside class and unessential data is hidden.
- In python if we want to perform data hiding then it can be done by using double underscore(__) prefix with variables or functions then they can not be accessed outside that function.


AskPython

Python Abstraction

Abstraction is that feature in **OOP** concept wherein the user is kept unaware of the basic implementation of a function property. The user is only able to view basic functionalities whereas the internal details are **hidden**.

Syntax,

```
from abc import ABC
class Abs_class(ABC):
        #abstract methods
        #class methods
```

link for detail- https://pynative.com/python-class-method/

```
# Normal Example
class Test1:
  x=10
  y=20

  def myFun1(self):
    print("This is function1")

class Test2(Test1):
  def myFun2(self):
    print("This is function2")

    # calling base class function and variable
    print(self.x)
    self.myFun1()

# creating object
obj = Test2()
obj.myFun2()
```

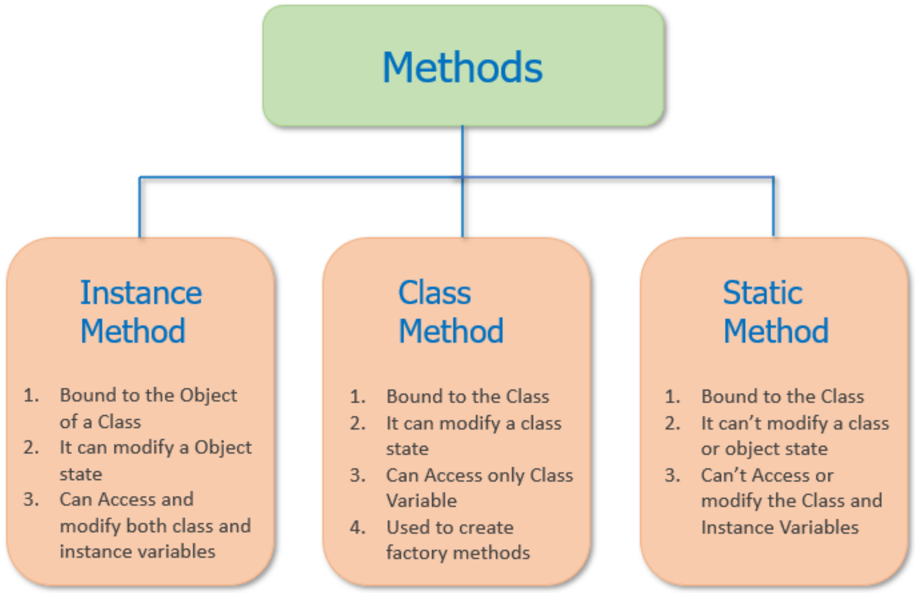
```
This is function2
10
This is function1
```

```
#Example with Data Hiding
class Test1:
  _x=10
  y=20

  def myFun1(self):
```

```
  print("This is function1")

class Test2(Test1):
  def myFun2(self):
    print("This is function2")

    # calling base class function and variable
    print(self.x)
    self.myFun1()

# creating object
obj = Test2()
obj.myFun2()
```

```
This is function2
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-3-7d69aaad6d0f> in <cell line: 19>()
     17 # creating object
     18 obj = Test2()
---> 19 obj.myFun2()

<ipython-input-3-7d69aaad6d0f> in myFun2(self)
     12 
     13     # calling base class function and variable
---> 14     print(self.x)
     15     self.myFun1()
     16 

AttributeError: 'Test2' object has no attribute 'x'
```

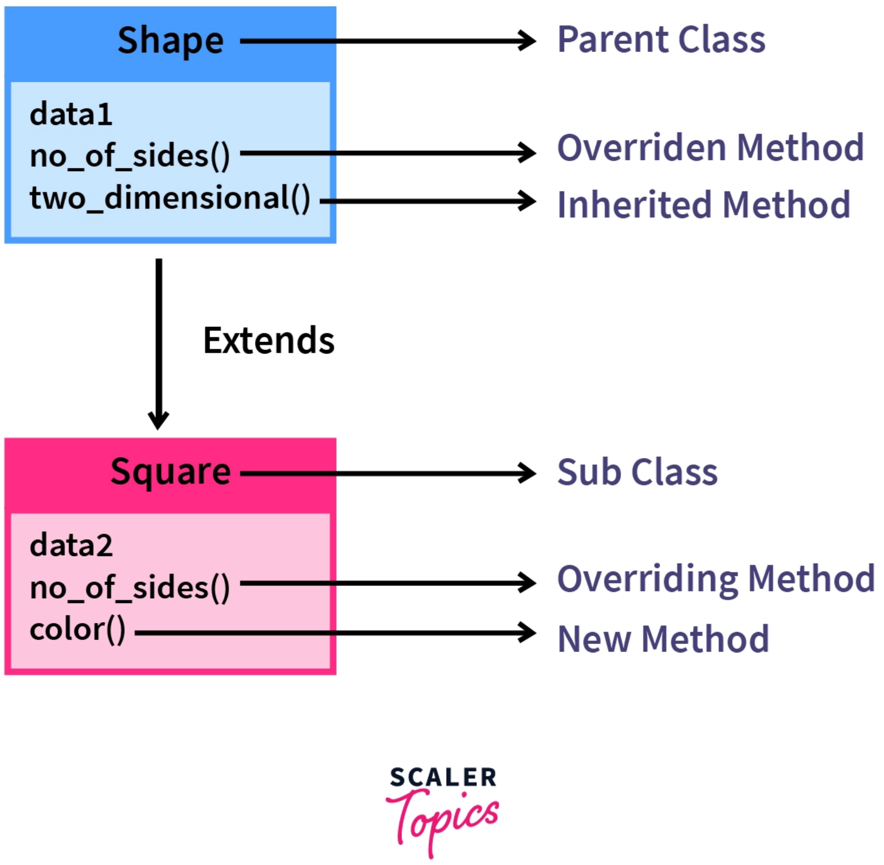
SEARCH STACK OVERFLOW


Methods
Instance Method
1. Bound to the Object of a Class
2. It can modify a Object state
3. Can Access and modify both class and instance variables
Class Method
1. Bound to the Class
2. It can modify a class state
3. Can Access only Class Variable
4. Used to create factory methods
Static Method
1. Bound to the Class
2. It can't modify a class or object state
3. Can't Access or modify the Class and Instance Variables

link-https://pynative.com/python-class-method-vs-static-method-vs-instance-method/

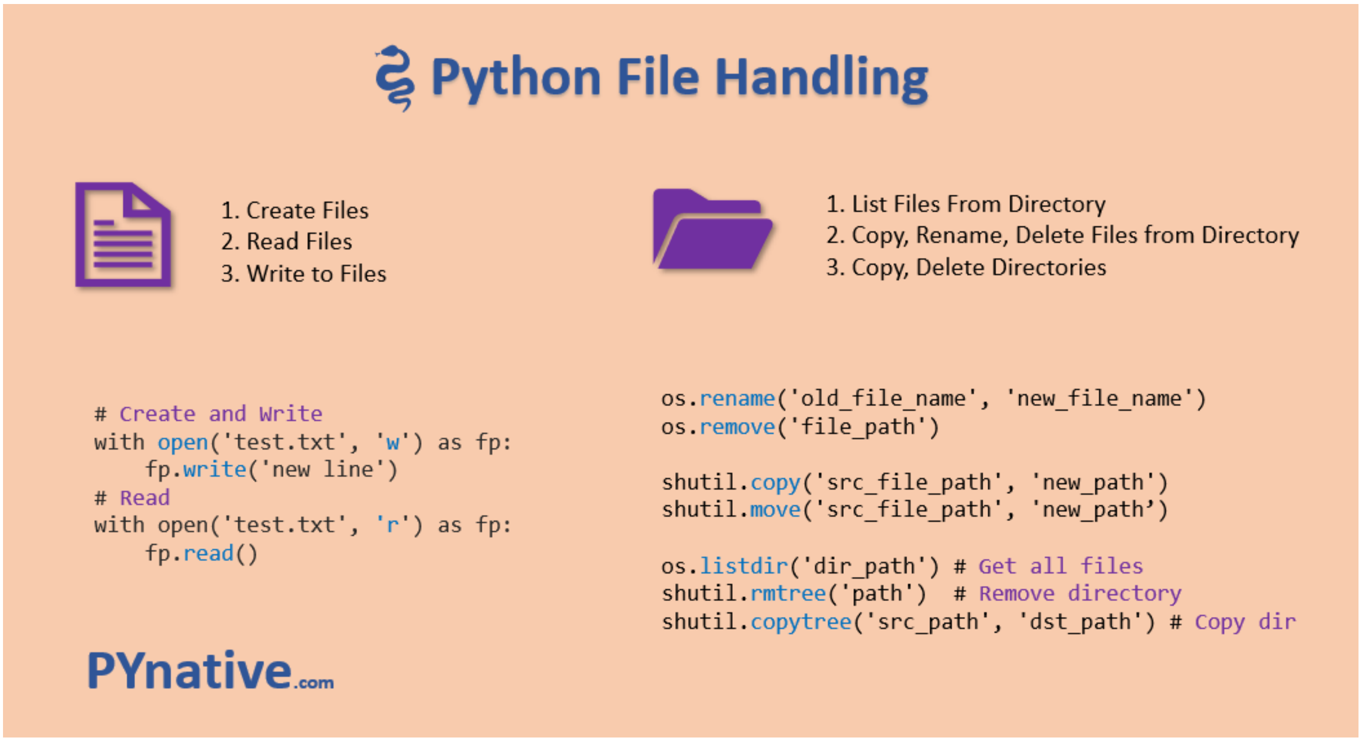
Function Overriding

- Function with same name and same parameters is called function overriding.
- It is not possible to make function with same name and same parameters in a single class so we use two class.



```
1
```

```
1
```



Python File Handling

1. Create Files
2. Read Files
3. Write to Files

1. List Files From Directory
2. Copy, Rename, Delete Files from Directory
3. Copy, Delete Directories

```
# Create and Write
with open('test.txt', 'w') as fp:
    fp.write('new line')
# Read
with open('test.txt', 'r') as fp:
    fp.read()
```

```
os.rename('old_file_name', 'new_file_name')
os.remove('file_path')

shutil.copy('src_file_path', 'new_path')
shutil.move('src_file_path', 'new_path')

os.listdir('dir_path') # Get all files
shutil.rmtree('path')  # Remove directory
shutil.copytree('src_path', 'dst_path') # Copy dir
```

PYnative.com

File Handling in Python

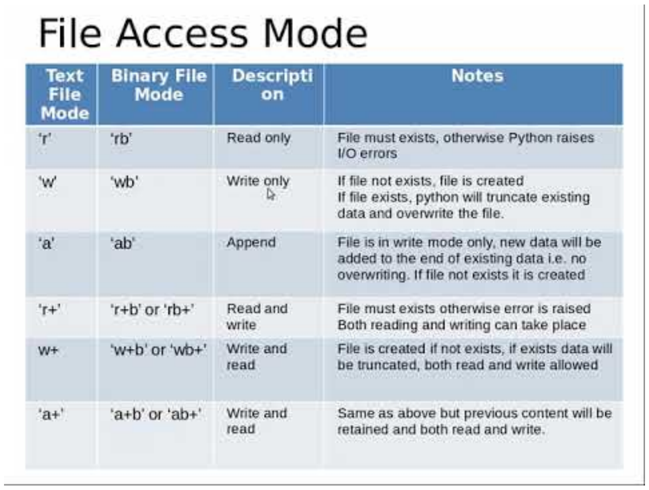
- File handling is a mechanism to store the data on the disk permanetly.
- There are several functions for :

```
- creating
-reading
-writing
-updating
-deleting
```

Operation on File

1.Opening of file

2. Writing into a file
3. Appending data into a file
4. Reading from a file
5. Closing of file



File Access Mode

Text File Mode	Binary File Mode	Description	Notes
'r'	'rb'	Read only	File must exists, otherwise Python raises I/O errors
'w'	'wb'	Write only	If file not exists, file is created If file exists, python will truncate existing data and overwrite the file.
'a'	'ab'	Append	File is in write mode only, new data will be added to the end of existing data i.e. no overwriting. If file not exists it is created
'r+'	'r+b' or 'rb+'	Read and write	File must exists otherwise error is raised Both reading and writing can take place
w+	'w+b' or 'wb+'	Write and read	File is created if not exists, if exists data will be truncated, both read and write allowed
'a+'	'a+b' or 'ab+'	Write and read	Same as above but previous content will be retained and both read and write.

```python
# case 1 - if the file is not present
f = open('sample.txt','w')
f.write('Hello world')
f.close()
```

```python
# write multiline strings
f = open('sample1.txt','w')
f.write('hello world')
f.write('\nhow are you?')
f.close()
```

```python
# case 2 - if the file is already present
f = open('sample.txt','w')
f.write('salman khan')
f.close()
```

```python
# introducing append mode
f = open('/content/sample1.txt','a')
f.write('\nI am fine')
f.close()
```

```python
# reading from files
# -> using read()
f = open('/content/sample.txt','r')
s = f.read()
print(s)
f.close()
```

```
salman khan
```

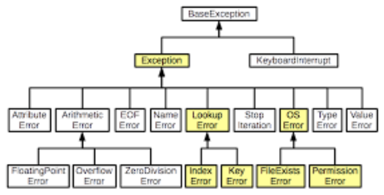
```python
# reading upto n chars
f = open('/content/sample.txt','r')
s = f.read(10)
print(s)
f.close()
```

```
salman kha
```

Exception handling

To understand the execution firstly we should know about the errors:

- Compile Time Errors:- errors caught during complied time is called Compile time errors.
- Run Time Errors:- The error that occurs during the run time of program is called run time error . They also known as Exceptions.



Syntax Error

- Something in the program is not written according to the program grammar.
- Error is raised by the interpreter/compiler
- You can solve it by rectifying the program
- Leaving symbols like colon,brackets
- Misspelling a keyword
- Incorrect indentation
- empty if/else/loops/class/functions

```
# Examples of syntax error
print 'hello world'
```

```
  File "<ipython-input-7-4655b84ba7b7>", line 2
    print 'hello world'
    ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(...)?
```

SEARCH STACK OVERFLOW

```
a = 5
if a==3
  print('hello')
```

```
  File "<ipython-input-8-efc58c10458d>", line 2
    if a==3
           ^
SyntaxError: expected ':'
```

SEARCH STACK OVERFLOW

```
a = 5
iff a==3:
  print('hello')
```

```
  File "<ipython-input-9-d1e6fae154d5>", line 2
    iff a==3:
        ^
SyntaxError: invalid syntax
```

SEARCH STACK OVERFLOW

```
a = 5
if a==3:
print('hello')
```

```
  File "<ipython-input-10-ccc702dc036c>", line 3
    print('hello')
    ^
IndentationError: expected an indented block after 'if' statement on line 2
```

SEARCH STACK OVERFLOW

```
# IndexError
# The IndexError is thrown when trying to access an item at an invalid index.
```

```python
L = [1,2,3]
L[100]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-11-c90668d2b194> in <cell line: 4>()
      2 # The IndexError is thrown when trying to access an item at an invalid index.
      3 L = [1,2,3]
----> 4 L[100]

IndexError: list index out of range
```

SEARCH STACK OVERFLOW

```python
# ModuleNotFoundError
# The ModuleNotFoundError is thrown when a module could not be found.
import mathi
math.floor(5.3)
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-12-cbdaf00191df> in <cell line: 3>()
      1 # ModuleNotFoundError
      2 # The ModuleNotFoundError is thrown when a module could not be found.
----> 3 import mathi
      4 math.floor(5.3)

ModuleNotFoundError: No module named 'mathi'

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES SEARCH STACK OVERFLOW

```python
# KeyError
# The KeyError is thrown when a key is not found

d = {'name':'nitish'}
d['age']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-13-453afa1c9765> in <cell line: 5>()
      3
      4 d = {'name':'nitish'}
----> 5 d['age']

KeyError: 'age'
```

SEARCH STACK OVERFLOW

```python
  # TypeError
  # The TypeError is thrown when an operation or function is applied to an object of an inappropriate type.
  1 + 'a'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-2a3eb3f5bb0a> in <cell line: 3>()
      1 # TypeError
      2 # The TypeError is thrown when an operation or function is applied to an object of an
inappropriate type.
----> 3 1 + 'a'

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

SEARCH STACK OVERFLOW

```python
  # ValueError
  # The ValueError is thrown when a function's argument is of an inappropriate type.
  int('a')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-e419d2a084b4> in <cell line: 3>()
      1 # ValueError
      2 # The ValueError is thrown when a function's argument is of an inappropriate type
```

```python
# NameError
# The NameError is thrown when an object could not be found.
print(k)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-16-e3e8aaa4ec45> in <cell line: 3>()
      1 # NameError
      2 # The NameError is thrown when an object could not be found.
----> 3 print(k)

NameError: name 'k' is not defined
```

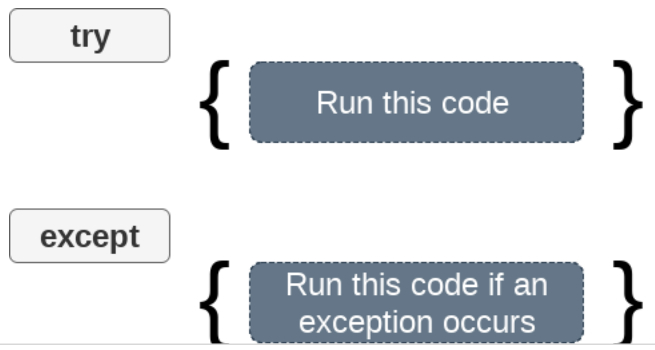
SEARCH STACK OVERFLOW

```python
# AttributeError
L = [1,2,3]
L.upper()

# Stacktrace
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-17-02532e56c452> in <cell line: 3>()
      1 # AttributeError
      2 L = [1,2,3]
----> 3 L.upper()
      4
      5 # Stacktrace

AttributeError: 'list' object has no attribute 'upper'
```

SEARCH STACK OVERFLOW

```

```



✓ 0s completed at 10:08 AM