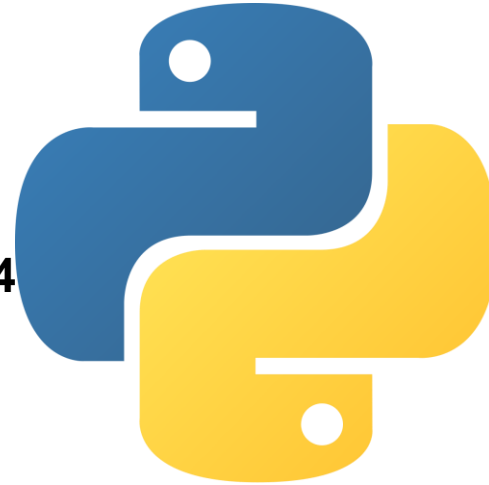


Python

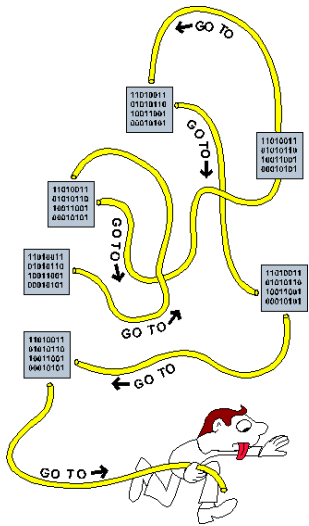
Day04

Programming for everyone



Object Oriented Python





Spaghetti Level

+ Functions

Procedural
Level

+ Modules

Modular Level

Object Oriented
Level

height = 175 cm
weight = 76 kg

Properties

velocity = 200 m/s
brakes = 2

walk()
speak()

Methods

stop()
move()



Man Object

ride(BikeObj)



Bike Object



OOP Keywords



A **class** is a template definition of an object's properties and methods.

```
class Human:  
    pass
```

Human Class



Object

An Object is an instance on a Class.

```
class Human:  
    pass  
  
man = Human()
```

Human Class



Man Object



Constructor

Constructor is a method called at the moment an object is instantiated.

```
class Human:  
  
    def __init__(self):  
        print("Hi there")  
  
man = Human()
```

Output:

```
Hi there
```

Human Class

`__init__()`



Man Object



Instance Variable

Instance Variable is an object characteristic, such as name.

```
class Human:

    def __init__(self, name):
        self.name = name

man = Human("Ahmed")
```

Human Class

name

__init__()



Name is Ahmed

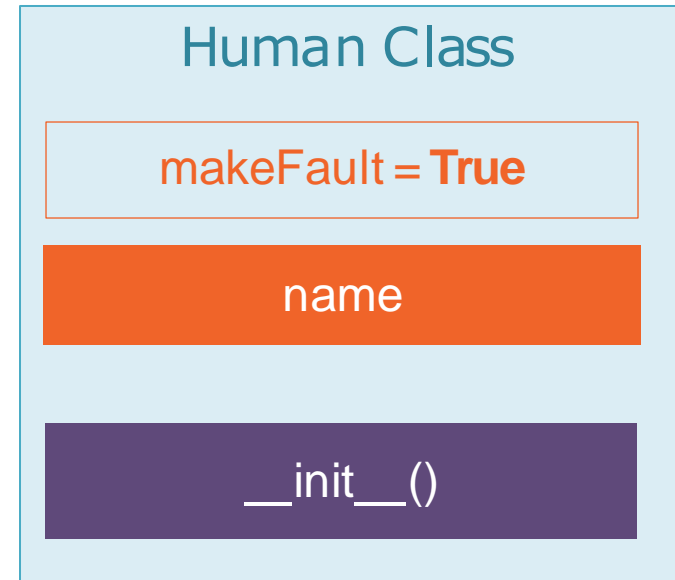
Man Object



Class Variable

Class Variable is the variable that shared by all instances.

```
class Human:  
  
    makeFault = True  
  
    def __init__(self, name):  
        self.name = name  
  
man = Human("Ahmed")  
man2 = Human("Mohamed")
```



Name is Ahmed

He makes faults



Name is Mohamed

He makes faults



Class Variable

```
class Human:
    faults = 0
    def __init__(self, name):
        self.name = name

man = Human("Ahmed")
man2 = Human("Mohamed")

man.faults = 1
print("Man :", man.faults)
print("Man 2:", man2.faults)
print("Human:", Human.faults)

Human.faults = 2
print("Man 2:", man2.faults)
print("Human:", Human.faults)
print("Man :", man.faults)
```

Output:

```
Man : 1

Man2 : 0

Human : 0

Man2 : 2

Human : 2

Man : 1
```



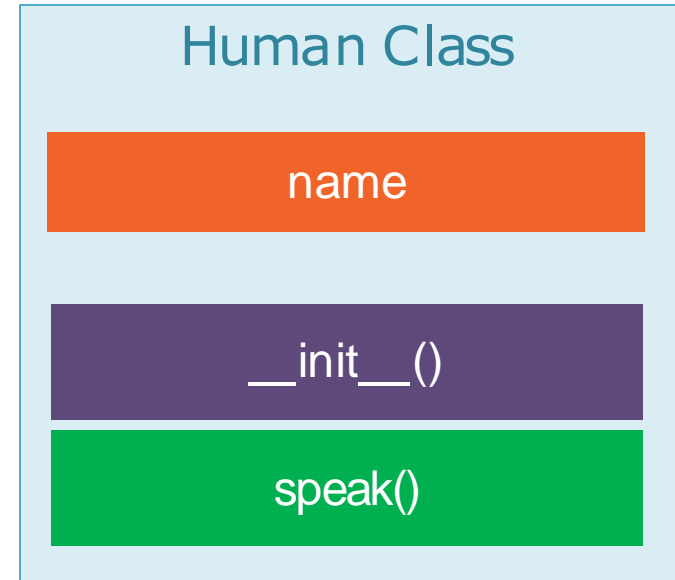
Instance Method

Instance Method is an object capability, such as walk.

```
class Human:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My Name is "+self.name)

man = Human("Ahmed")
man.speak()
```



My Name is Ahmed



Class Method

Class Method is a method that shared by all instances of the Class

```
class Human:

    faults=0

    def __init__(self, name):

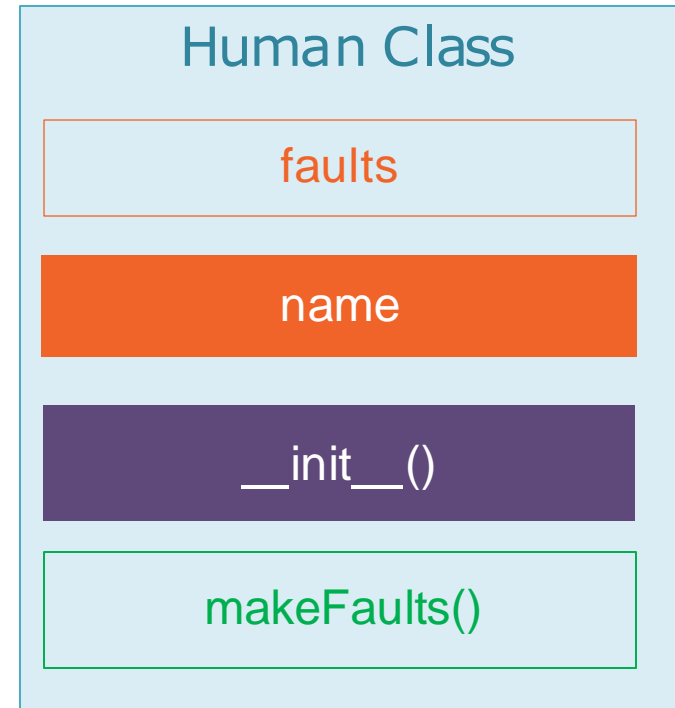
        self.name = name

    @classmethod
    def makeFaults(cls):

        cls.faults +=1

        print(cls.faults)

Human.makeFaults() #1
man = Human("Ahmed")
man.makeFaults() #2
```



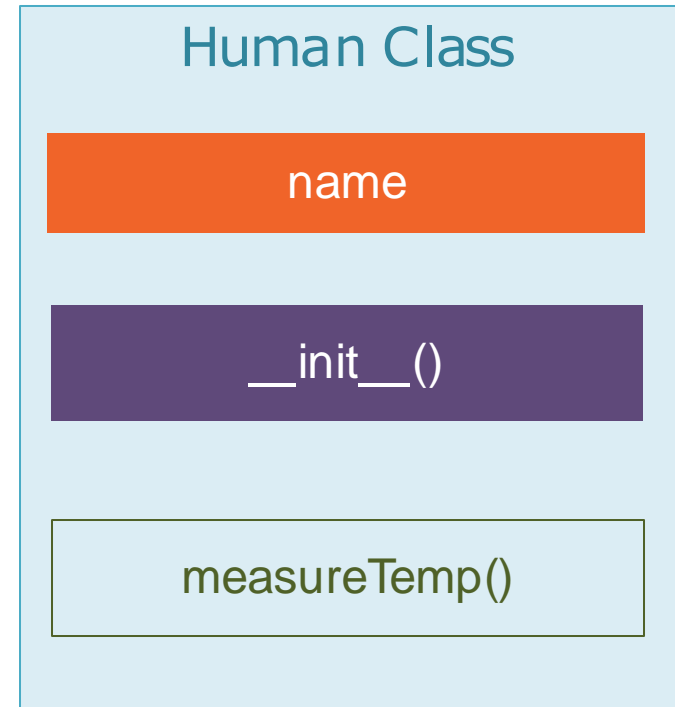
Static Method

Static Method is a normal function that have logic that related to the Class

```
class Human:
    def __init__(self, name):
        self.name = name

    @staticmethod
    def measureTemp(temp):
        if (temp == 37):
            return "Normal"
        return "Not Normal"

Human.measureTemp(38) # Not Normal
```



Static vs Class Methods

Class Method

```
# cls(Class) is implicitly passed to class method like self(instance) in instance method.
```

```
# Class Method is related to the class itself.
```

```
class Human:
    @classmethod
    def walk(cls):
        print("Walk ...")
```

```
Human.walk()
```

Static Method

```
# Static Method is like a normal function but we put it in the class because it has logic that is related to the class.
```

```
# We call it Helper Method
```

```
class Human:
    @staticmethod
    def sleep():
        print("whoa")
```

```
Human.sleep()
```

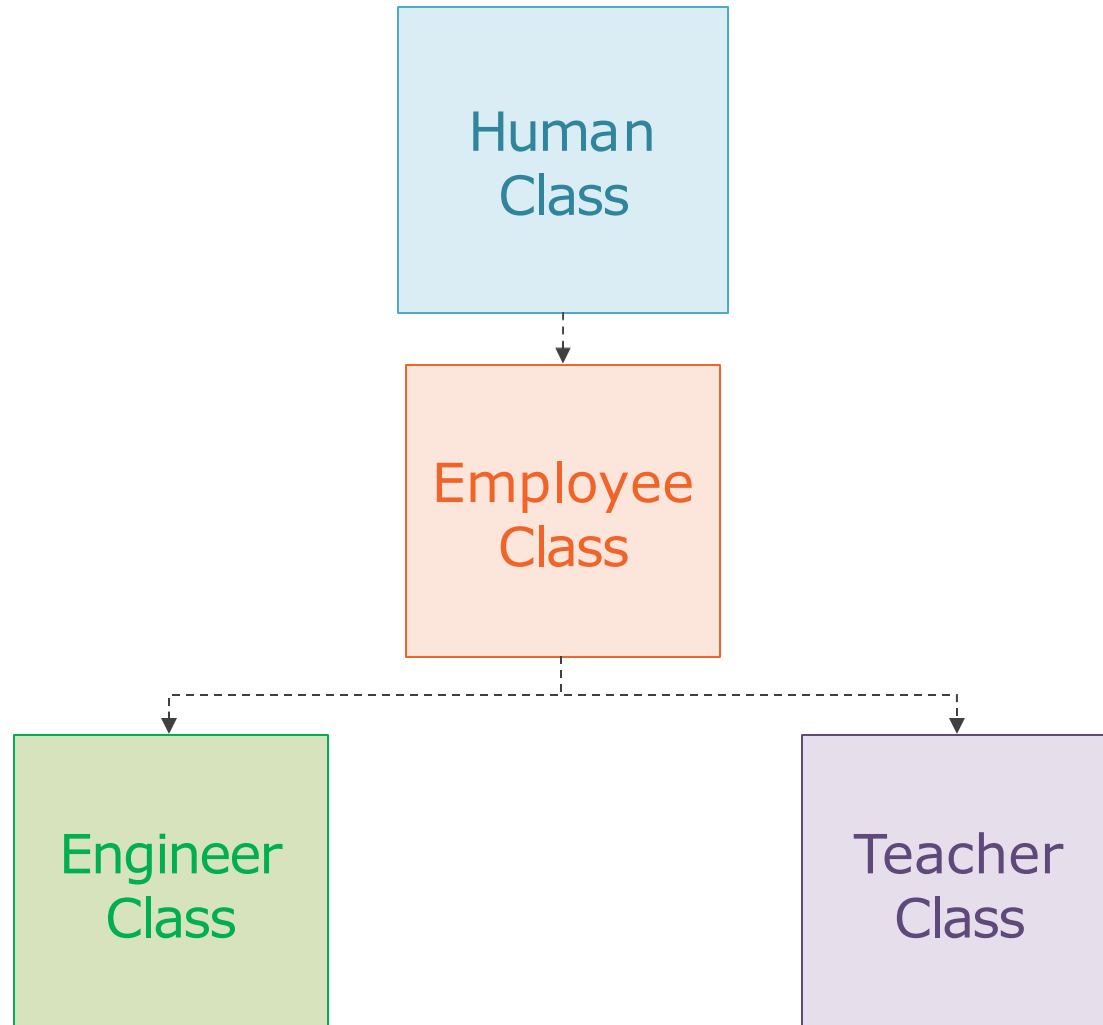


OOP Concepts



Inheritance





Example

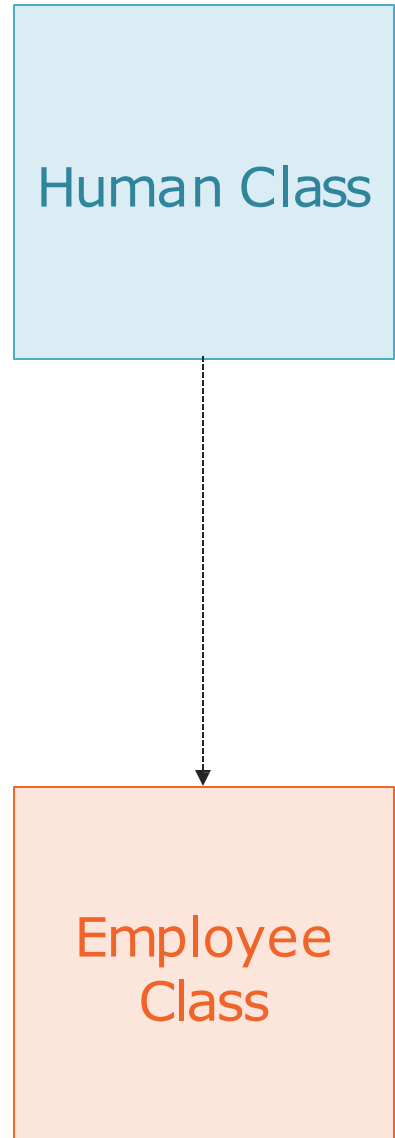
```
class Human:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My Name is "+self.name);

class Employee(Human):
    def __init__(self, name, salary):
        super(Employee, self).__init__(name)
        self.salary = salary

    def work(self):
        print("I'm working now");

emp = Employee("Ahmed", 500)
emp.speak()
emp.work()
```



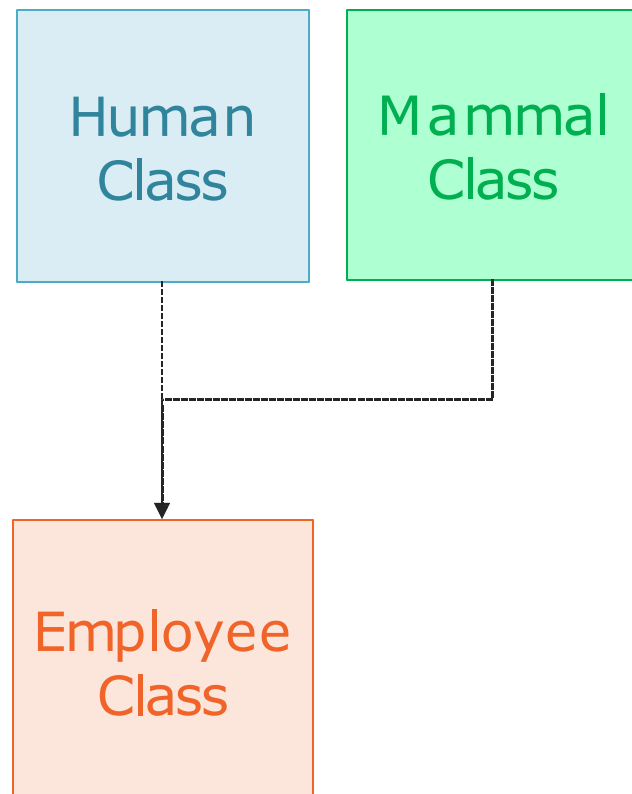
Multiple Inheritance

Python supports **Multiple** Inheritance

Report**:

- 1- How **super** Function handle **Multiple** Inheritance.
- 2- If **Human** and **Mammal** Have the same method like **eat** but with different Implementation. When Child [**Employee**] calls eat method how **python** handle this case.

******Prove your opinion with examples.

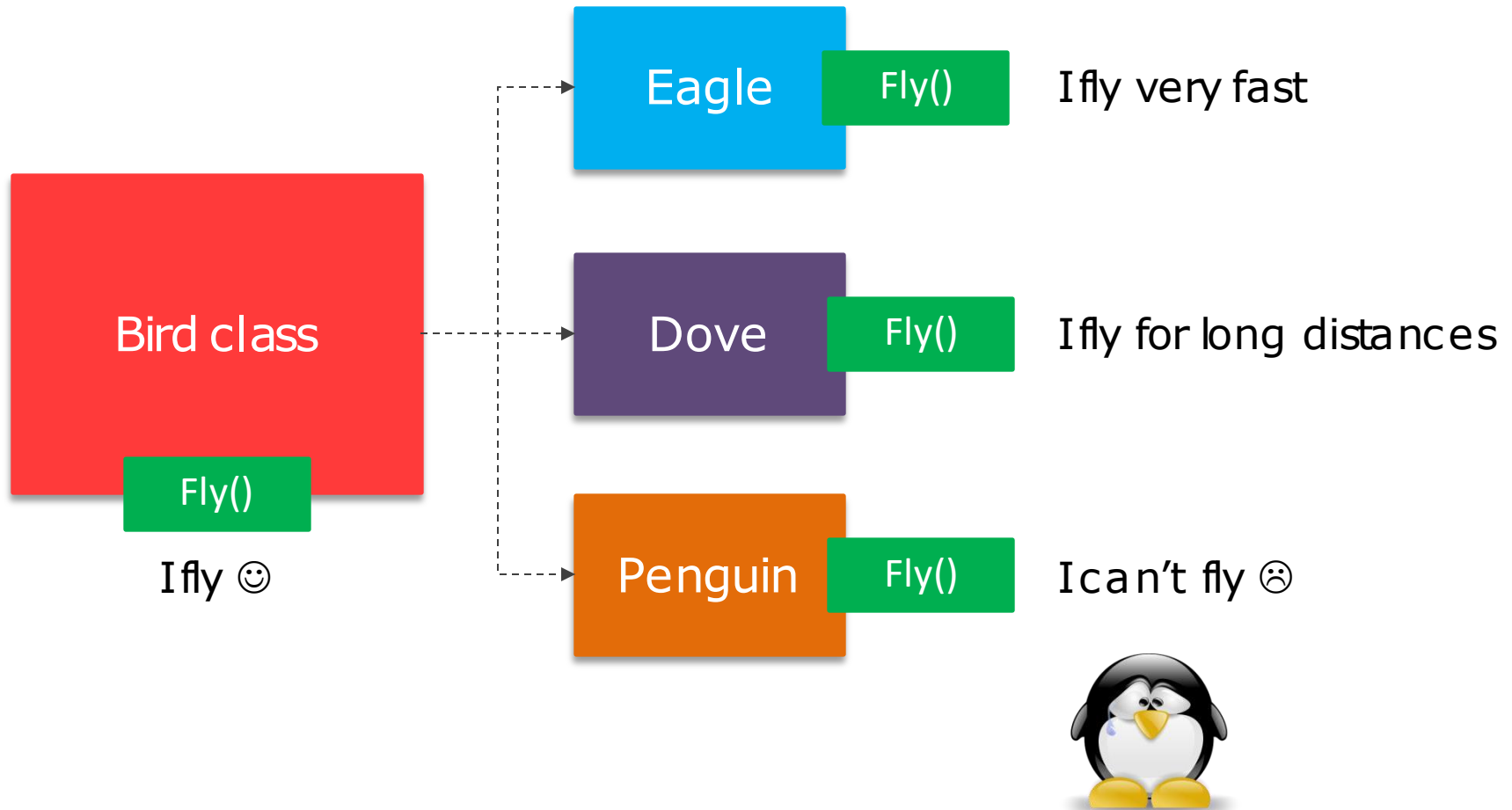


Polymorphism



Intro

Poly means "*many*" and **morphism** means "*forms*". Different classes might define the same method or property.



Method Overriding

```
class Human:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print("My Name is "+self.name);

class Employee(Human):
    def __init__(self, name, salary):
        super(Employee, self).__init__(name)
        self.salary = salary

    def speak(self):
        print("My salary is "+self.salary);

emp = Employee("Ahmed", 500)
emp.speak() #My Salary is 500
```

Human Class

Employee
Class



Method Overloading

Report**:

Can we do overloading in Python ?

If **Yes**, Tell me **How??**

If **No**, Tell me **Why??**



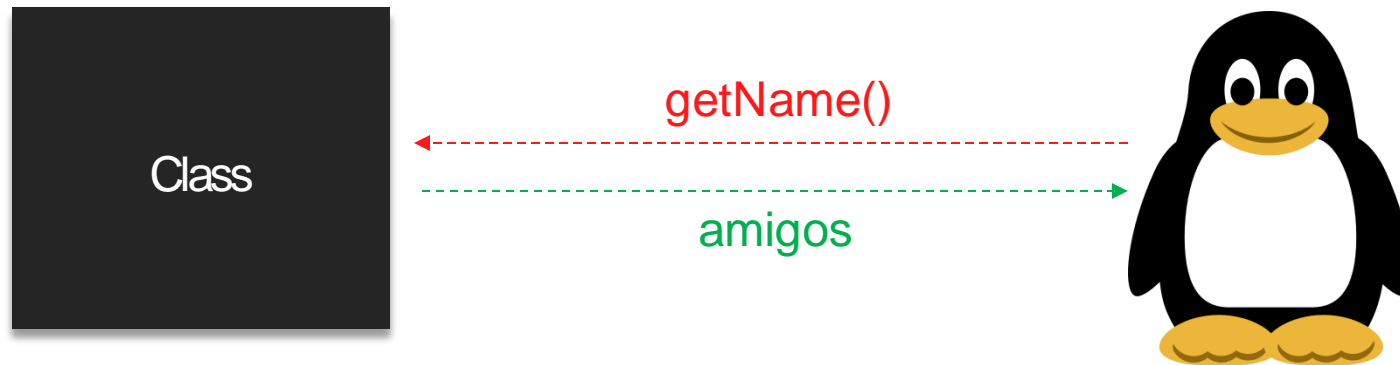
****** Support Your Answer by Examples.



Encapsulation



Encapsulation is the packing of data and functions into one component (for example, a class) and then controlling access to that component .



Example

```
class Human:
    def __init__(self, name):
        self.__name = name

    def getName(self):
        return self.__name
```

```
man = Human("Mahmoud")
```

```
print(man.__name)
```

```
AttributeError: 'Human' object has no attribute '__name'
```

```
print(man.getName())
```

```
#output: Mahmoud
```



@property

```
class Human:
    def __init__(self, age):
        self.age = age

    @property
    def age(self):
        return self.__age

    @age.setter
    def age(self, age):
        if age > 0:
            self.__age = age
        if age <= 0:
            self.__age = 0

man = Human(23)
print(man.age)    # 23
man.age = -25
print(man.age)    # 0
```



Special Methods



Special Method that controls how Object treats as printable

```
class Human:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return "Hi, I'm Human and my name is " + self.name

man = Human("Ahmed")

print(man)

#output: <_main_.Human object at 0x000000FD81804400>

print(man)

#output: Hi, I'm Human and my name is Ahmed
```



Special Method that controls how Object can show as callable

```
class Human:
    def __init__(self, name):
        self.name = name

    def __call__(self):
        print("You called me !")

man = Human("Ahmed")

man()

#output: TypeError: 'Employee' object is not callable

man()

#output: You called me !
```



__len__

Special Method that controls when measure the Object length

```
class Animal:
    def __init__(self, legs):
        self.legs = legs

    def __len__(self):
        return self.legs
```

```
dog = Animal(4)
```

```
len(dog)
```

```
#output: TypeError: 'Employee' object has no len()
```

```
len(dog)
```

```
#output: 4
```



Tips and Tricks



Lambda Expressions

Lambda Expressions are used to make anonymous functions

lambda input:output

Example

```
lmdaFn = lambda x:x+4  
  
lmdaFn(3)    #7  
  
def sumFn(n):  
    return lambda x:x+n  
  
sumFn(5)    #<function ...>  
  
sumFn(5)(4)    #9
```



Iterators (iter and next)

iter is used to generate an iterator from iterable.

next is used to return the next iteration from iterators.

Example

```
l = ["JavaScript", "Python", "Java"]           # iterable
it = iter(l)                                   # convert iterable to iterator
next(it)
#output: "JavaScript"
next(it)                                       #output: "Python"
next(it)                                       #output: "Java"
```



Generators

It is used to generate iterators

Example

```
def nonGenFn():  
    for i in range(5):  
        return i  
  
ng = nonGenFn()
```

```
next(ng)
```

```
TypeError: 'int' object is  
not an iterator
```

```
def genFn():  
    for i in range(5):  
        yield i  
  
g = genFn()
```

```
next(g)      #output: 0
```

```
next(g)      #output: 1
```



map Function

```
map(function, sequence)
```

Map function are used to make iterables from apply the given function on every item in the given sequence

Example

```
it = map(lambda x:x+4, [1,3,5])  
  
for i in it:  
    print(i)  
  
# 4  
# 7  
# 9
```



filter Function

filter(*condfunction*, *sequence*)

Filter function are used to make iterables from filter each item in the given sequence by the given function

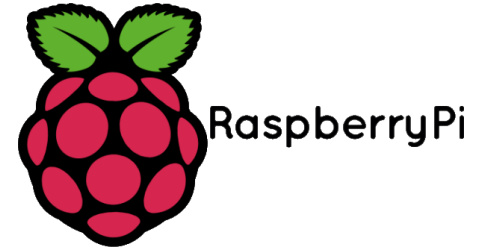
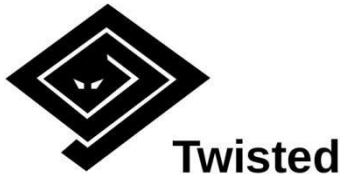
Example

```
it= filter(lambda x:x%5==0, [-15, -8, -5, 3, 5, 9, 25])
for i in it:
    print(i, end=" ", " ")
# -15, -5, 5, 25
```



What's Next ?





django



Thank You