Python from Basics to OOP

Stay connected!!!

Facebook:

https://www.facebook.com/profile.php?id=100086216576997

YouTube:

https://www.youtube.com/channel/UCNHweLJ7GlfLWjfF1rZbU8Q

Output

- **print() function**

In [ ]:
```
print("Datasiasts")
print(2)
```

```
Datasiasts
2
```

In [ ]:
```
print("hello")
print("hello")
```

```
hello
hello
```

In [ ]:
```
print("hello",end="\n")
print("hello")
```

```
hello
hello
```

- **string formating**

In [ ]:
```
name="achraf"
age=20

#using %
print("Hello %s,you are %d years old"%(name,age))

#using .format()
print("Hello {},you are {} years old".format(name,age))

#using f-string
print(f"Hello {name},you are {age} years old")
```

```
Hello achraf,you are 20 years old
Hello achraf,you are 20 years old
Hello achraf,you are 20 years old
```

bonus 1:

In [ ]:
```
number1=157.3345667
print(f"number1 = {round(number1,4)}")
```

```
number1 = 157.3346
```

bonus 2:

In [ ]:
```
name = "achraf"
string1 = f"hello {name}"
string2 = "hello {} again".format(name)
string3 = "hello %s again again" %name
print(string1)
print(string2)
print(string3)
```

```
hello achraf
hello achraf again
hello achraf again again
```

Input

- **input() function**

In [ ]:
```
# string1=input()
# print(string1)

# print(type(string1))

# string2=input("Enter a string : ")
# print(string2)

# int1=int(input())
# print(int1)

int2=int(input("Enter an integer : "))
print(int2)

#print(type(s2))
```

```
Enter a string : 4
4
```

bonus 1:

In [ ]:
```
x,y,z =input("Enter three values : ").split()

print(f"x={x} , y={y} , z={z}")
```

```
Enter three values : 1 2 3
x=1 , y=2 , z=3
```

In [ ]:
```
x,y,z =input("Enter three values separated with ',' : ").split(",")

print(f"x={x} , y={y} , z={z}")
```

```
Enter three values separated with ',' : 1,2,3
x=1 , y=2 , z=3
```

In [ ]:
```
L=input("Enter multiple values : ").split()

print(L)
```

```
Enter multiple values : 1 3 5 6 7
['1', '3', '5', '6', '7']
```

Conditionals

In [ ]:
```
x=2
if x>=0 :
    print(f"{x} is positive")
else :
    print(f"{x} is negative")
```

```
2 is positive
```

In [ ]:
```
x=1.5
if x>=2 :
    print(f"{x} is greater than 2")
elif x>=1 :
    print(f"{x} is between 1 and 2")
elif x>=0 :
    print(f"{x} is between 0 and 1")
else :
    print(f"{x} is negative")
```

```
1.5 is between 1 and 2
```

In [ ]:
```
a=1
b=2
# if a==1 and b==2:
#     print("iwa mbrouk")
# if a==1 or b==3:
#     print("iwa mbrouk2")
if not a==2:
    print("iwa mbrouk3")
```

In [ ]:
```

```

Loops

- for loop

In [ ]:
```
for i in range(5):
    print(i,end=" ")
```

```
0 1 2 3 4
```

In [ ]:
```
List=["said","sabir", 2 ]
for i in List:
    print(i,end=" ")
```

```
said sabir 2
```

- while loop

In [ ]:
```
number=int(input("input a positive number : "))
while number<0 :
    number=int(input("input a positive number : "))
print(number)
```

```
input a positive number : -2
input a positive number : -3
input a positive number : -6
input a positive number : 2
2
```

- continue

In [ ]:
```
string1 ='Datasiast'
for i in string1 :
    if i == 'a':
        continue
    print(i)
```

```
D
t
s
i
s
t
```

- break

In [ ]:
```
string1 ='Datasiast'
for i in string1:
    if i == 't':
        break
    print(i)
```

```
D
a
```

Data types

- **Integer**

In [ ]:
```
int1=-1
print(int1)
# print(type(int1))
```

```
-1
<class 'int'>
```

- **Float**

In [ ]:
```
float1=22/7
print(float1)
print(type(float1))
```

```
3.142857142857143
<class 'float'>
```

- **Boolean**

In [ ]:
```
T=True
F=False
print(T)
print(F)
```

```
True
False
```

- **String**

In [ ]:
```
phrase = 'Data not found'
phrase2 = "Data not found"
print(phrase[0]) # returns the first character
```

```
print(phrase[-1]) # returns the first character from the end
print(phrase[-3]) # returns the sixth character from the end
```

```
D
d
u
```

In [ ]:
```
print(phrase.upper()) # returns an uppercase version of phrase
print(phrase.lower()) # returns an lowercase version of phrase
print(phrase.title()) # returns a new version of phrase with first letter of every word capitalized
print(phrase.replace('a', 'b')) # returns a new version of phrase where all 'a's are replaced with 'b's
print("----------")
print(phrase) #to check if the original string has been changed
print("----------")
print(phrase.find("a")) # returns the index of the first occurrence of "a"
```

```
DATA NOT FOUND
data not found
Data Not Found
Dbtb not found
----------
Data not found
----------
1
```

- **Lists**

In [ ]:
```
List = [50, 70, 30, 20, 90, 10, 40]
print(List)
List2 = [50, "sabir", 30, 5.8, 90, 10, 40]
print(List2)
```

```
[50, 70, 30, 20, 90, 10, 40]
[50, 'sabir', 30, 5.8, 90, 10, 40]
```

In [ ]:
```
print(List[3]) #access
List[3]=5 #modify
print(List) #check
```

```
20
[50, 70, 30, 5, 90, 10, 40]
```

bonus 1: check if an element is in the list

In [ ]:
```
x=50
if x in List :
    print(f"{x} is in the list ")
else:
    print(f"{x} is not in the list ")
```

```
50 is in the list
```

bonus 2: slicing

form : List_name[start : end : step]

In [ ]:
```
print(List[::])#taking copy of the list
print(List[::2])
print(List[4::])
print("----------")
print(List[::-1])#inverse a list
print(List[4:2:-1])
print("----------")
print(List)#to check if the original string has been changed
```

```
[50, 70, 30, 5, 90, 10, 40]
[50, 30, 90, 40]
[90, 10, 40]
----------
[40, 10, 90, 5, 30, 70, 50]
[90, 5]
----------
[50, 70, 30, 5, 90, 10, 40]
```

note : the slicing concept works also with strings

bonus 3: list comprehension

In [ ]:
```
List=[ 2*x for x in range(9)]
print(List)
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16]
```

In [ ]:
```
List=[ 2*x for x in range(9) if x%3==0]
print(List)
```

```
[0, 6, 12]
```

- **Sets**

A set is a collection of **unindexed** and **unorderd** elements ,not allowing **duplicates**

In [ ]:
```
#usage
set1=set([1,2,3])
print(set1)

set2={3,4,5}
print(set2)
```

```
{1, 2, 3}
{3, 4, 5}
```

In [ ]:
```
#unordered
set1={"hello","hi"}
for i in set1 :
    print(i)
```

```
hello
hi
```

In [ ]:
```
#unindexed
print(set1[0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1660\554774743.py in <module>
      1 #unindexed
----> 2 print(set1[0])

TypeError: 'set' object is not subscriptable
```

In [ ]:
```
#removing duplicates
listWithDuplicates=[1,2,8,7,1,2,5]
print("listWithDuplicates =",listWithDuplicates)

listToSet=set(listWithDuplicates)
print("listToSet =",listToSet)

cleanedList=list(listToSet)
print("clenedList =",cleanedList)
```

```
listWithDuplicates = [1, 2, 8, 7, 1, 2, 5]
listToSet = {1, 2, 5, 7, 8}
clenedList = [1, 2, 5, 7, 8]
```

In [ ]:
```
#basic functions
setA={2,4}
setB={2,3,5}

setA.add(5)
print(setA)

setA.remove(5)
print(setA)

print(setA.intersection(setB))
print(setA.union(setB))
print(setA.difference(setB))
```

```
{2, 4, 5}
{2, 4}
{2}
{2, 3, 4, 5}
{4}
```

- **Tuples**

A tuple is a collection which is **ordered** and **unchangeable**.

In [ ]:
```
#usage
Tuple1 = (50, 70, 30, 20, 90, 10, 40)
print(Tuple1)
Tuple2 = (50, "sabir", 30, 5.8, 90, 10, 40)
print(Tuple2)
```

```
(50, 70, 30, 20, 90, 10, 40)
(50, 'sabir', 30, 5.8, 90, 10, 40)
```

In [ ]:
```
#unchangeable
print(Tuple1[3]) #access
#Tuple1[3]=5 #testing modifying
# print(Tuple1) #check
```

```
20
```

In [ ]:
```
#ordered
Tuple1=(1,2,3)
for i in Tuple1 :
    print(i)
```

```
1
2
3
```

In [ ]:
```
#slicing
print(Tuple1[::2])
```

```
(1, 3)
```

note : the slicing concept works also with tuples

bonus :joining tuples

In [ ]:
```
#join
Tuple1=(1,2)
Tuple2=(3,2)
Tuple3=Tuple1+Tuple2
print(Tuple3)
print(Tuple1)
```

```
(1, 2, 3, 2)
(1, 2)
```

- **Dictionaries**

form : Dict_name{ key1 :value1 ,key2 :value2 }

In [ ]:
```
Dictionary1={"name":"sabir","age":6}
print(Dictionary1)
Dictionary2={"first name":"sabir","second name":"man3rf"}
print(Dictionary2)
Dictionary3={"name":"sabir",
        "age":6}
print(Dictionary3)
```

```
{'name': 'sabir', 'age': 6}
{'first name': 'sabir', 'second name': 'man3rf'}
{'name': 'sabir', 'age': 6}
```

In [ ]:
```
print(Dictionary1.keys())
print(Dictionary1.values())

for element in Dictionary1.values():
    print(element)
```

```
dict_keys(['name', 'age'])
dict_values(['sabir', 6])
sabir
6
```

- **Mutable vs Immutable**

Mutable : can be changed (Lists ,dictionaries ...)
Immutable : can not be changed (Numbers ,Strings,Tuples ...)

In [ ]:
```
#test1
list1=[5,6]
print(list1,":",id(list1))
list1.append(7)
print(list1,":",id(list1))
```

```
[5, 6] : 3100147990016
[5, 6, 7] : 3100147990016
```

In [ ]:
```
#test2
string1="hello"
print(string1,":",id(string1))
string1=string1+" Club Data"
print(string1,":",id(string1))
```

```
hello : 3100109817776
hello Club Data : 3100148531760
```

In [ ]:
```
#test3
int1=4
print(int1,":",id(int1))
int1=5
print(int1,":",id(int1))
```

```
4 : 3100027742608
5 : 3100027742640
```

Functions

In [ ]:
```
def welcoming():
    print("hello guys")
welcoming()
```

```
hello guys
```

In [ ]:
```
def showInfo(name,age,state="student"):#default value of state
    print(f"hello {name} , you are {age} years old , you are {state}")

showInfo("sabir",6)
showInfo("sabir",6,"software engineer")
```

```
hello sabir , you are 6 years old , you are student
hello sabir , you are 6 years old , you are software engineer
```

- **local vs global**

In [ ]:
```
#seuil is global
#mark is local

seuil=12
def acceptedOrNot(mark):
    if mark >seuil:
        print("accepted")
    else:
        print("not accepted")

acceptedOrNot(13)
print(mark)
```

```
accepted
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_1660\2422232500.py in <module>
     10
     11 acceptedOrNot(13)
---> 12 print(mark)

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

In [ ]:
```
seuil=10
def modifySeuil():
    global seuil
    seuil=12

modifySeuil()

print(seuil)
```

```
12
```

- **print vs return**

In [ ]:
```
def factorialReturned(num):
    p=1
    for i in range(1,num+1):
        p=p*i

    return p

x=factorialReturned(3)
print(x)
```

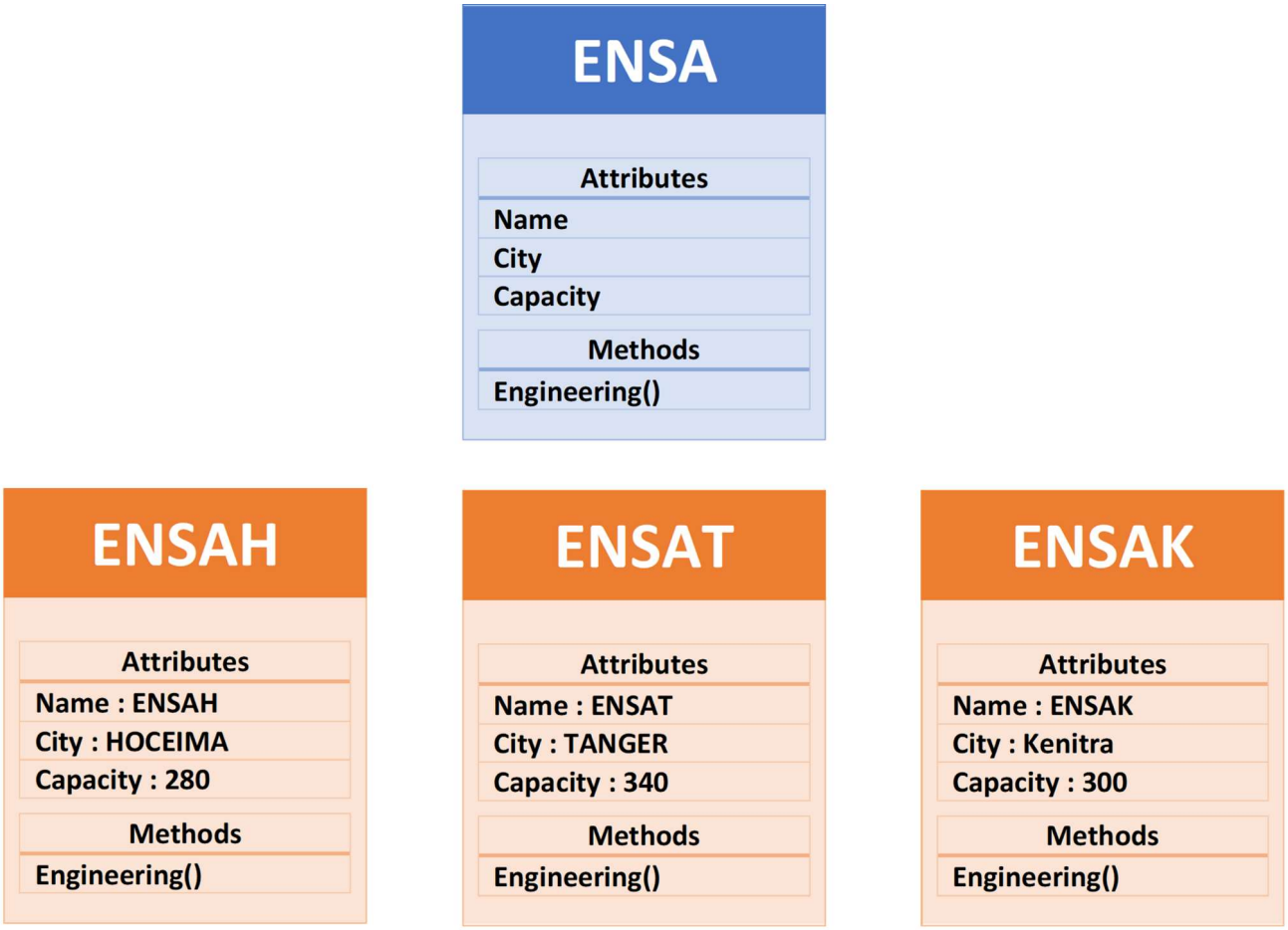
Out[ ]:
```
6
```

In [ ]:
```python
def factorialPrinted(num):
    p=1
    for i in range(1,num+1):
        p=p*i

    print(p)
    return p

x=factorialPrinted(3)
print(x)
```

```
6
6
```

OOP

- **Concept**



- **Class**

In [ ]:
```python
class Class_name:
    def __init__(self,attribute1,attribute2):
        self.attribute1=attribute1
        self.attribute2=attribute2

    def show_info(self,param):
        print(f"value of attribute1 is {self.attribute1} and value of attribute2 is {self.attribute2}")
        print(f"value of parameter is {param}")
```

- **Object**

In [ ]:
```
object1=Class_name("string",6)
print(object1.attribute1)
print(object1.attribute2)
object1.show_info("argument")
```

```
string
6
value of attribute1 is string and value of attribute2 is 6
value of parameter is argument
```

- **Constructor**

In [ ]:
```
class Class_name:
    def __init__(self,attribut1,attribut2):
        self.attribut1=attribut1
        self.attribut2=attribut2
        print("object created")

C=Class_name("string",6)
```

```
object created
```

Example

In [ ]:
```
class ENSA:
    def __init__(self,name,city,capacity):
        self.name=name
        self.city=city
        self.capacity=capacity

    def engineering(self):
        print(f"{self.capacity} student are joining {self.name} every year ")
```

In [ ]:
```
ENSAH=ENSA("ENSAH","Hoceima",280)
ENSAT=ENSA("ENSAT","Tanger",340)
ENSAK=ENSA("ENSAK","Kenitra",300)
```

In [ ]:
```
print(ENSAH.capacity)
print(ENSAK.city)
ENSAT.engineering()
ENSAH.engineering()
```

```
280
Kenitra
340 student are joining ENSAT every year
280 student are joining ENSAH every year
```

- **Static attributes**

Humans count example

In [ ]:
```
class Human:
    count=0 #class/static attribute

    def __init__(self,name):
        self.name=name #object attribute
        Human.count+=1

human1=Human("sabir")
print(Human.count)
human2=Human("amine")
```

```
print(Human.count)
print(human1.count)
```

```
1
2
2
```

In [ ]:
```
human1.count=5
print(human1.count)
print(Human.count)
```

```
5
2
```

- **Static methodes**

Student and administration example

In [ ]:
```
class Student:
    count=0 #class/static attribute
    def __init__(self,name):
        self.name=name #object attribute
        Student.count+=1

    @staticmethod
    def showInfo():# static methode
        print(f"number of students is {Student.count}")

Student.showInfo()
student1=Student("sabir")
Student.showInfo()
student1.showInfo()
```

```
number of students is 0
number of students is 1
number of students is 1
```

- **Operator overloading**

In [ ]:
```
class Assir:
    def __init__(self,tofah,banane):
        self.tofah=tofah
        self.banane=banane

    def __add__(self,another_one):
        return Assir( self.tofah + another_one.tofah , self.banane + another_one.banane)

assir1=Assir(2,9)
assir2=Assir(4,3)
assir3=assir1+assir2
print(f"number of tofah in assir3 = {assir3.tofah}")
print(f"number of banane in assir3 = {assir3.banane}")
```

```
number of tofah in assir3 = 6
number of banane in assir3 = 12
```

for more you can check : https://www.geeksforgeeks.org/operator-overloading-in-python/

- **Printing objects**

In [ ]:
```
class Assir:
    def __init__(self,tofah,banane):
        self.tofah=tofah
        self.banane=banane

    def __add__(self,another_one):
```

```
        return Assir(self.tofah+another_one.tofah,self.banane+another_one.banane)

    def __str__(self):
        return f"there is {self.tofah} tofahs and {self.banane} bananes"

assir=Assir(5,3)
print(assir)
```

```
there is 5 tofahs and 3 bananes
```

Inheritence

In [ ]:
```
class Parent:
    def __init__(self,attribute1,attribute2,name):
        self.attribute1=attribute1
        self.attribute2=attribute2
        self.name=name

    def parentFunction(self):
        print(f"I am {self.name} and this function is from the parent class")


class Child(Parent):
    def __init__(self,attribute1,attribute2,name,childAttribute):
        super().__init__(attribute1,attribute2,name)

        self.childAttribute=childAttribute

    def childFunction(self):
        print(f"I am {self.name} and this function is from the child class")


parent1=Parent(2,3,"father")
child1=Child(4,5,"child","extra atribute")

parent1.parentFunction()#executing a parent function from a parent object
child1.parentFunction()#executing a parent function from a child object
child1.childFunction()#executing a child fuction from a child object
```

```
I am father and this function is from the parent class
I am child and this function is from the parent class
I am child and this function is from the child class
```

exemple

In [ ]:
```
class Hayawan :
    def __init__(self,name,weight):
        self.name=name
        self.weight=weight

    def eat(self):
        print(f"{self.name} is eating")

class Cat(Hayawan):
    def __init__(self,name,weight,owner):
        super().__init__(name,weight)
        self.owner=owner

    def miaw(self):
        print("miaw miaw miaw")

class Shark(Hayawan):
    def __init__(self,name,weight):
        super().__init__(name,weight)
```

```
    def swim(self):
        print("swiiiiming")
```

In [ ]:
```
hayawan1=Hayawan("far",5)
hayawan1.eat()
```

```
far is eating
```

In [ ]:
```
cat1=Cat("lmech",10,"Anas")
cat1.eat()
cat1.miaw()
```

```
lmech is eating
miaw miaw miaw
```

In [ ]:
```
shark1=Shark("l9irch",50)
shark1.eat()
shark1.swim()
```

```
l9irch is eating
swiiiiming
```

Polymorphisme

In [ ]:
```
class Person:
    def __init__(self,nom):
        self.nom=nom

    def affiche(self):
        print("I am a person")

class Student(Person):
    def __init__(self,nom,cne):
        super().__init__(nom)
        self.cne=cne

    def affiche(self):
        print("I am a student")

class Teacher(Person):
    def __init__(self,nom,ppr):
        super().__init__(nom)
        self.ppr=ppr

    def affiche(self):
        print("I am a teacher")
```

In [ ]:
```
parent1=Person("sabir")
student1=Student("amine","p110144139")
teacher1=Teacher("mohamed",444444)
```

Let's notice the behavior of the **affiche function**

In [ ]:
```
parent1.affiche()
```

```
I am a person
```

In [ ]:
```
student1.affiche()
```

```
I am a student
```

In [ ]:
```
teacher1.affiche()
```

```
I am a teacher
```

In [ ]:

Encapsulation

This is encapsulation 👇

In [ ]:
```
class Class_name:
    def __init__(self,attribute1,attribute2):
        self.attribute1=attribute1
        self.attribute2=attribute2

    def function1(self):
        print("Doing something")

    def function2(self):
        print("Doing something else")
```

Access modifiers

- **public** : accessible from everywhere
- **_protected** : accessible from the class itself and it's children
- **__private** : accessible just from the class

In [ ]:
```
class Parent:
    def __init__(self):
        self.publicAttribute="this is public"
        self._protectedAttribute="this is protected"
        self.__privateAttribute="this is private"


    def showPublic(self):
        print(f"printing the public attribute : {self.publicAttribute}")

    def showProtected(self):
        print(f"printing the protected attribute : {self._protectedAttribute}")

    def showPrivate(self):
        print(f"printing the private attribute : {self.__privateAttribute}")


class Child(Parent):
    def __init__(self):
        super().__init__()

    def printPublic(self):
        print(f"this a public attribute called from child : {self.publicAttribute}")
    def printProtected(self):
        print(f"this a protected attribute called from child : {self._protectedAttribute}")
    def printPrivate(self):
        print(f"this a private attribute called from child : {self.__privateAttribute}")
```

In [ ]:
```
parent1=Parent()
child1=Child()
```

Accessing the **public** attributes

In [ ]:
```
print(parent1.publicAttribute)
parent1.showPublic()
child1.printPublic()
```

```
this is public
printing the public attribute : this is public
this a public attribute called from child : this is public
```

Accessing the **private** attributes

In [ ]:
```
parent1.showPrivate()
# print(parent1.__privateAttribute)
child1.printPrivate()
```

```
printing the private attribute : this is private
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9652\500738171.py in <module>
      1 parent1.showPrivate()
      2 # print(parent1.__privateAttribute)
----> 3 child1.printPrivate()

~\AppData\Local\Temp\ipykernel_9652\96072027.py in printPrivate(self)
     25         print(f"this a protected attribute called from child : {self._protectedAttribute}")
     26     def printPrivate(self):
---> 27         print(f"this a private attribute called from child : {self.__privateAttribute}")
     28

AttributeError: 'Child' object has no attribute '_Child__privateAttribute'
```

Accessing the **protected** attributes

Guess the output

In [ ]:
```
parent1.showProtected()
#print(parent1._protectedAttribute)
child1.printProtected()
```

```
printing the protected attribute : this is protected
this is protected
this a protected attribute called from child : this is protected
```

Let's do another test

In [ ]:
```
print(parent1._Parent__privateAttribute)
```

```
this is private
```

Getters && Setters

Getter : to control how the user gets the value

Setter : to control how the user sets the value

In [ ]:
```
class Parent:
    def __init__(self):
        self.__fullName=None

    def setFullName(self,name):
        self.__fullName=name.replace(" ","_")

    def getFullName(self):
        if self.__fullName==None:
            return "no name set"
        return self.__fullName.replace("_"," ")

    def printRealFullName(self):
        return self.__fullName
```

In [ ]:
```
parent1=Parent()
parent1.setFullName("yassine boonooooo")
```

```
print(parent1.getFullName())
print(parent1.printRealFullName())
```

```
yassine boonooooo
yassine_boonooooo
```

In [ ]: