Encapsulation

One of the key parts of object-oriented programming is encapsulation, which involves the packaging of variables and related functions in one simple-to-use object: the instance of a class.

A concept related to this is data hiding, which consists in hiding the implementation details of a class. In this way the user interface of this class is clean and more intuitive.

In other programming languages, data hiding is done by creating private methods and attributes, to which their external access to the function is blocked. In Python, however, the philosophy is slightly different “we are all consenting adults” and therefore there are no particular restrictions on access.

So there is nothing that can deprive an attribute or method in such a way as to make it inaccessible to the outside of the classroom.

In Python, we use double underscore (__) before the attributes name to make those attributes private.

We can use single underscore (_) before the attributes name to make those attributes protected.

Example	Output

class A:
  __a=10 #private
  _b=20 #protected
  d=40 #public
  def abc(self,c):
  print(self.__a+c)
  print(self._b+c)
obj=A()
obj.abc(10)
 	
#access public var
print(obj.d)
 	
#access protected var
print(obj._b)
 	
#access private var
print(obj.__a)
	20
	30
	40
	20
	Traceback (most recent call last):
	  File "main.py", line 12, in <module>
	  print(obj.__a)
	AttributeError: A instance has no attribute '__a'

We can access the value of hidden attribute by a tricky syntax as object._className__attrName.

Example	Output

class A:
  __a=10 #private
  _b=20 #protected
  d=40 #public
  def abc(self,c):
  print(self.__a+c)
  print(self._b+c)
obj=A()
obj.abc(10)
 	
#access public var
print(obj.d)
 	
#access protected var
print(obj._b)
 	
#access private var
print(obj._A__a)
	20
	30
	40
	20
	10