OBJECT ORIENTED PROGRAMMING

LECTURES 3 & 4

BY: KEVIN HARVEY

ACCESS SPECIFIERS

As a manager, I would want data that I want no one to see as a manager, like:

- Credit Card information of customers
- IMEI of the devices sold and in the store
- Other sensitive info…

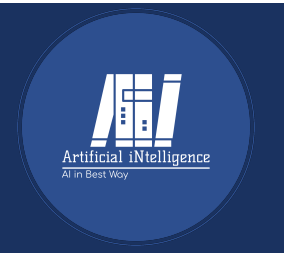
To protect this data, we use the dual underscore (__) to specify to the compiler know it's private and cannot be accessed **DIRECTLY** outside of the class

بنحط الاتنين
دول underscore
عشان نحدد لل
compiler ان ال
attribute دي private

```
class Apple:
    def __init__(self,specs,price,quantity, imei):
        self.specs = specs
        self.price = int(price)
        self.quantity = quantity
        self.__imei = imei
```

Public Attributes

EXAMPLE OF USING A PRIVATE ATTRIBUTE NATIVELY




```python
class Apple:
    def __init__(self,specs,price,quantity, imei):
        self.specs = specs
        self.price = int(price)
        self.quantity = quantity
        self.__imei = imei
iphone = Apple("iPhone 16 Pro Max",1200,1, "123456789")
print(iphone.__imei)
```

Output:

```
/usr/local/bin/python3 /Users/kevinharvey/scriptt.py
kevinharvey@Kevins-MacBook-Pro ~ % /usr/local/bin/python3 /Users/kevinharvey/scriptt.py
Traceback (most recent call last):
  File "/Users/kevinharvey/scriptt.py", line 8, in <module>
    print(iphone.__imei)
          ^^^^^^^^^^^^^
AttributeError: 'Apple' object has no attribute '__imei'
```

THE WORKAROUND SOLUTIONS

Getter (Accessor) & Setter (Mutator “Modifier”) Functions:

Using these functions allows us to access or modify private variables safely

Example Using Setter/Getter Functions:

```python
class Apple:
    def __init__(self, specs, price, quantity, imei):
        self.specs = specs
        self.price = int(price)
        self.quantity = quantity
        self.__imei = imei
    def set_imei(self, new_imei):
        self.__imei = new_imei
    def get_imei(self):
        return self.__imei
iphone = Apple("iPhone 16 Pro Max", 1200, 1, "123456789")
print("Original IMEI:", iphone.get_imei())
iphone.set_imei("987654321")
print("Updated IMEI:", iphone.get_imei())
```

احنا هنا استدعينا ال private variable بطريقة indirect
عشان اللى بيستدعيها ال function و استحالة تستدعيها
لوحدها من غير getter function

EXAMPLE

```python
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height
rect1 = Rectangle(10, 5)
rect2 = Rectangle(20, 5)
rect2.width = rect1.width
rect2.width = 30
print("rect1 width:", rect1.width)
print("rect2 width:", rect2.width)
```

```
kevinharvey@Kevins-MacBook-Pro ~ % /usr/local/bin/python3 /Users/kevinharvey/scriptt.py
rect1 width: 10
rect2 width: 30
```

DESTRUCTOR

- **It's a function that is automatically called when an object is destroyed to free memory, It's also a member function that is never declared with a return data type, but that may have arguments (same as the constructor)**

ال syntax بتاع ال destructor

```python
class Rectangle:
    def __init__(self, name, width, height):
        self.name = name
        self.width = width
        self.height = height
    def __del__(self):
        print(f"Rectangle {self.name} is deleted")
rect1 = Rectangle("rect1", 10, 5)
rect2 = Rectangle("rect2", 20, 5)
del rect1
```

OVERLOADING CONSTRUCTORS

- **A class may have one or more constructors but Python only recognizes the last constructor but we can workaround it by using *args**

```python
class Apple:
    def __init__(self, *args):
        if len(args) == 1:
            self.count = args[0]
        elif len(args) == 2:
            self.count = args[0]
            self.color = args[1]
        elif len(args) == 3:
            self.count = args[0]
            self.color = args[1]
            self.chip = args[2]
    def __str__(self):
        return f"Apple count: {self.count}, color: {self.color}, chip: {self.chip}"
iphone15pro= Apple(1, "black", "A15")
print(iphone15pro)
```

```
kevinharvey@Kevins-MacBook-Pro ~ % /usr/local/bin/python3 /Users/kevinharvey/scriptt.py
Apple count: 1, color: black, chip: A15
```

STATIC VARIABLE “CLASS VARIABLE”

- **It’s a variable that is shared for all objects and not specific for any instance**

Static Variable (It doesn’t contain “self” therefore it doesn’t deal with the object, it deals with all of them at once)

```
class Apple:
    count = 0
    def __init__(self, name, price):
        self.name = name
        self.price = price
        Apple.count += 1
iphone = Apple("iPhone", 1000)
ipad = Apple("iPad", 800)
print(Apple.count)
```

Output:

```
kevinharvey@Kevins-MacBook-Pro ~ % /usr/local/bin/python3 /Users/kevinharvey/scriptt.py
2
```

STATIC METHOD “CLASS METHOD”

- **Following the static variable, there’s also a static method that deals with the class itself not every object, they also can be called without creating an object**

☢️ ملحوظتين:

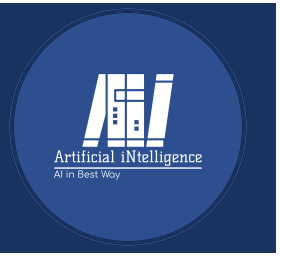
1. مابتلمسش ال object ذات نفسه
2. **مابتتعاملش مع ال self نهائيا**

هنا بنحدد اننا نتعامل مع

Output:

```
1   class Apple:
2       count = 0
3       def __init__(self,color,chip):
4           self.color = color
5           self.chip = chip
6           Apple.count += 1
7       @staticmethod
8       def get_count():
9           return Apple.count
10  iphone = Apple("black","A15")
11  ipad = Apple("white","M2")
12  print(Apple.get_count())
```

FRIEND FUNCTIONS



- It's a function that's not a member of a class but has access to private variables of the class.
- Python doesn't support it but we can work around it.

```
1   class Apple:
2       def __init__(self,count,color,chip):
3           self.count = count
4           self.color = color
5           self.__chip = chip
6       def friendfunction(self): # friend :)
7           print("chip:",self.__chip)
```

INHERITANCE الوراثة

- **It’s a way that allows a new class to inherit all of the features another class had**

فلنفترض ان الراجل ده لونه ابيض و شعره اسود

وقتها لما يخلف هيتنقل كل صفاته للطفل بتاعه

بس وقتها ممكن يزيد عليه حاجات من جينات الام

Is a

حيوان

فا في الطبيعي هيورث صفات الحيوان بس هيبقى ليه صفات زيادة

INHERITANCE الوراثة

- **It’s a way that allows a new class to inherit all of the features another class had**

```
1   class Apple:
2       def __init__(self,count,color,chip):
3           self.count = count
4           self.color = color
5           self.__chip = chip
```

بنوري لل compiler اننا بنورث من ال class اللى فوق

نفس الفكرة، كل حاجة
هيورثها من الاب بس
بحاجات زياده من الام

```
class AppleProducts(Apple):
    def __init__(self,count,color,chip,price):
        super().__init__(count,color,chip)
        self.price = price
```

PROTECTED VARIABLES

- **Protected variables are like private variables, BUT can be accessed from inherited classes**

```python
class Apple:
    def __init__(self,color,chip):
        self._color = color
        self._chip = chip
class Appleproducts(Apple):
    def __init__(self,color,chip,price):
        super().__init__(color,chip)
        self.price = price
    def __str__(self):
        return f"Apple products: {self._color} {self._chip} {self.price}"
iphone = Appleproducts("black","A15", "500$")
ipad = Appleproducts("white","M2" , "1000$")
print(ipad)
```

Protected Variables

هنا محتاجناش نعمل setter و getter

```
kevinharvey@Kevins-MacBook-Pro ~ % /usr/local/bin/python3 /Users/kevinharvey/scriptt.py
Apple products: white M2 1000$
```

TABLE

Thank you