Python Programming

Lesson 4 – Web Scraping(Introduction)

Lesson 4 - Outline

- Four basic concepts of OOP
- HTTP request in Python

Four basic concepts of OOP

Four basic concepts of OOP

- Encapsulation
- Inheritance
- Abstraction
- Polymorphism

Encapsulation

What is Encapsulation in OOP

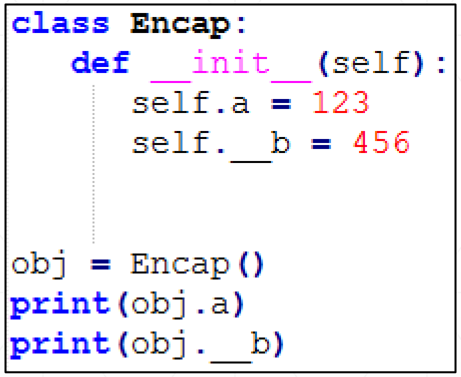
- **Background**
  An object variable should not always be directly accessible
- **Reason**
  Prevent accidental change
- **Solution**
  - An object variable can sometimes only be changed with an objects methods
  - Those type of variables are **private** variables
  - The methods can ensure the correct values are set. If an incorrect value is set, the method can return an error

What is Encapsulation in OOP

- **More Information**
  - Python does not have the **private keyword**, unlike some other object oriented languages (e.g. Java, C#), but encapsulation can be done
  - A class variable that should not directly be accessed should be **prefixed with two underscores**

What is Encapsulation in OOP

- **Example 1**

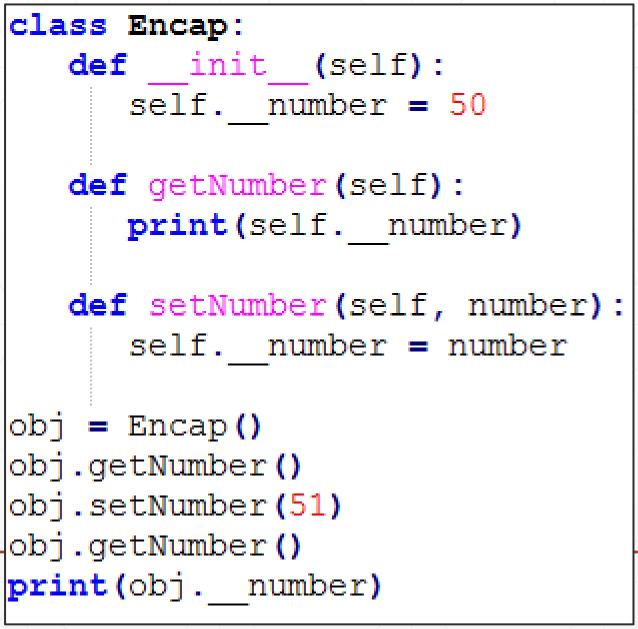


```
class Encap:
    def __init__(self):
        self.a = 123
        self.__b = 456

obj = Encap()
print(obj.a)
print(obj.__b)
```

- **Explanation**
  - **A double underscore:** Private variable, cannot be accessed directly

What is Encapsulation in OOP

- **Getter and setter**
  - Private variables are intended to be changed using getter and setter methods
- **Example 2**

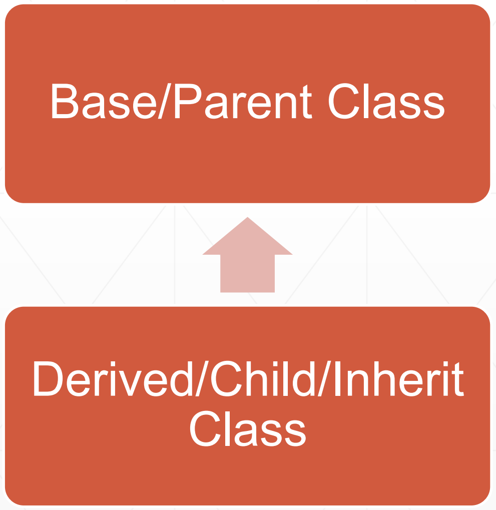


```python
class Encap:
    def __init__(self):
        self.__number = 50

    def getNumber(self):
        print(self.__number)

    def setNumber(self, number):
        self.__number = number

obj = Encap()
obj.getNumber()
obj.setNumber(51)
obj.getNumber()
print(obj.__number)
```

Inheritance

What is Inheritance in OOP

- **Background**
  Inheritance is the process of creating a class that can derive or inherit the **properties** and **methods** from another class(parent/base).
- **Graphical Explanation**

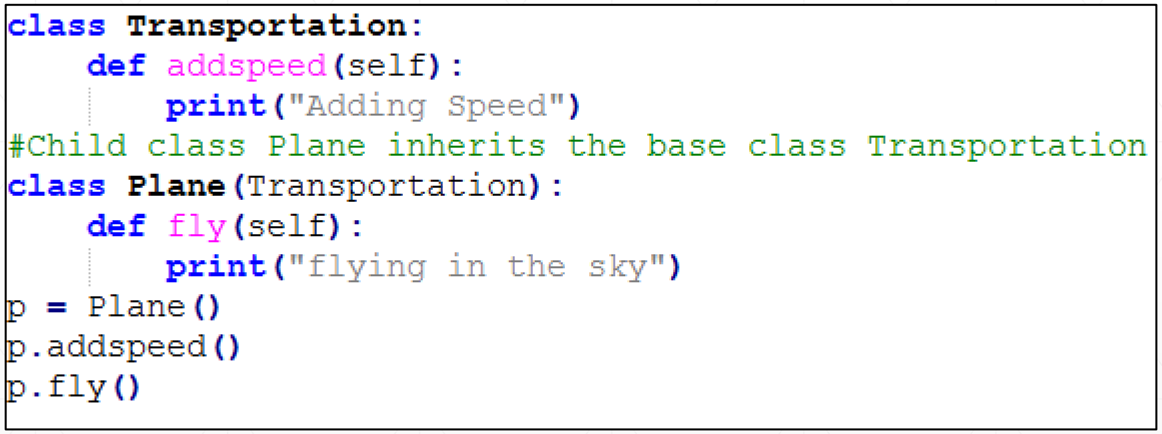


- **Extra Information**
  - Transitive nature, e.g. when class B inherited from class A, then the classes that inherited from class B will automatically be inherited from class A *(a.k.a. Multi-Level inheritance)*

What is Inheritance in OOP

- **Example 3**

  The child class can use the properties / methods of its parent class



```
class Transportation:
    def addspeed(self):
        print("Adding Speed")
#Child class Plane inherits the base class Transportation
class Plane(Transportation):
    def fly(self):
        print("flying in the sky")
p = Plane()
p.addspeed()
p.fly()
```

Abstraction

What is Abstraction in OOP

- **Background**

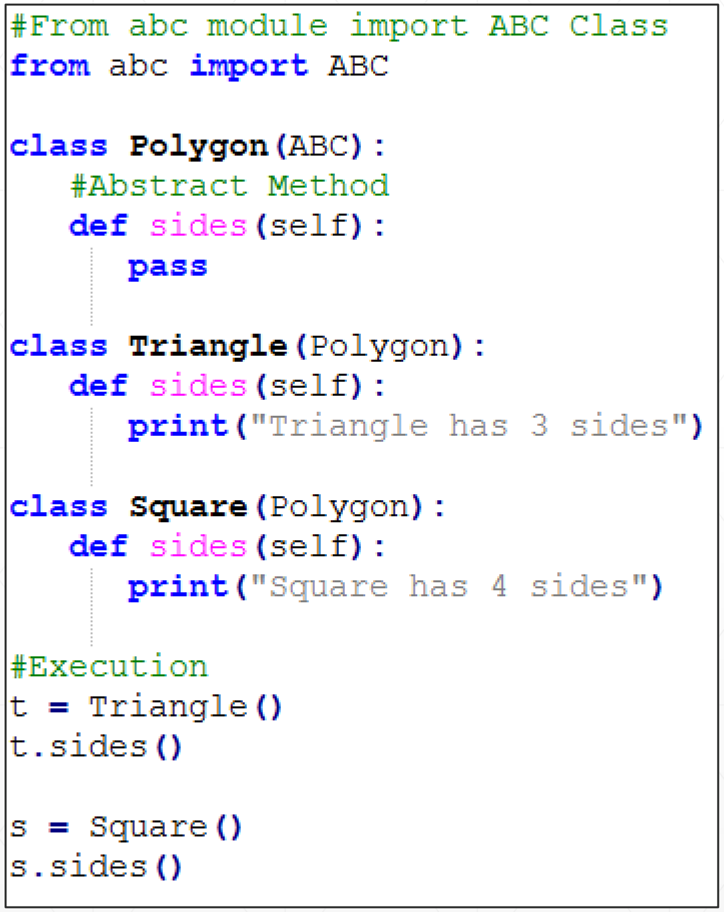
  Abstraction in OOP is a process of hiding the real **implementation** of the method by only showing a **method signature**

- **Solution**
  - `ABC` **(Abstract Base Class)** is a class from the `abc` module in Python
  - If we extend any class with `ABC`, and this class will have to implement those abstract methods

- **Extra Information**
  - Using `@abstractmethod` annotation, then the classes inherited from this class will have to mandatorily implement those abstract methods
  - Objects cannot be created for Abstract Class

What is Abstraction in OOP

- **<u>Example 4</u>**
  The child class can implement the methods in the abstract class



```python
#From abc module import ABC Class
from abc import ABC

class Polygon(ABC):
    #Abstract Method
    def sides(self):
        pass

class Triangle(Polygon):
    def sides(self):
        print("Triangle has 3 sides")

class Square(Polygon):
    def sides(self):
        print("Square has 4 sides")

#Execution
t = Triangle()
t.sides()

s = Square()
s.sides()
```

Polymorphism

What is Polymorphism in OOP

- **Background**

  Use a single entity, like a method or object, to represent different behaviors

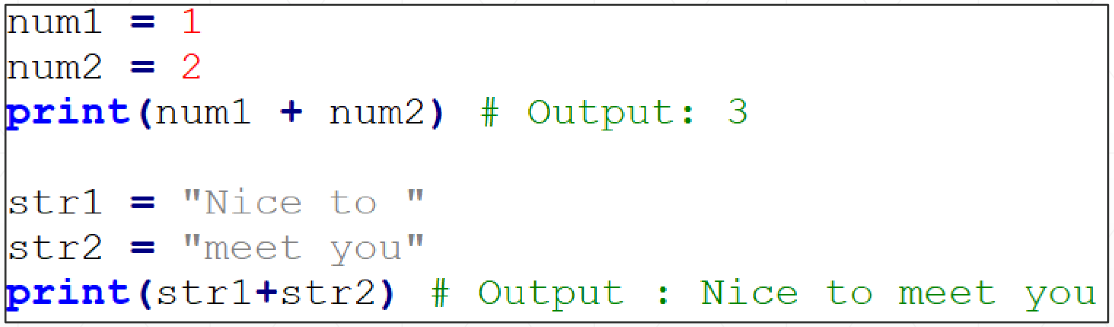
- **Category**
  - Operator Overloading
  - Method Overloading

- **Extra Information**
  - Method overriding is also a kind of polymorphism

What is Polymorphism in OOP

- **Example 5**

  Operator Overloading



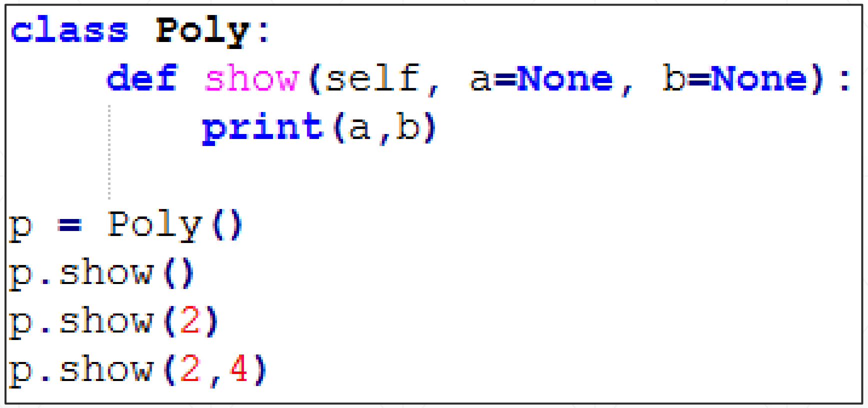
```
num1 = 1
num2 = 2
print(num1 + num2) # Output: 3

str1 = "Nice to "
str2 = "meet you"
print(str1+str2) # Output : Nice to meet you
```

- + operator act as an addition between two integers and concatenation between strings

What is Polymorphism in OOP

- **<u>Example 6</u>**

  Method Overloading



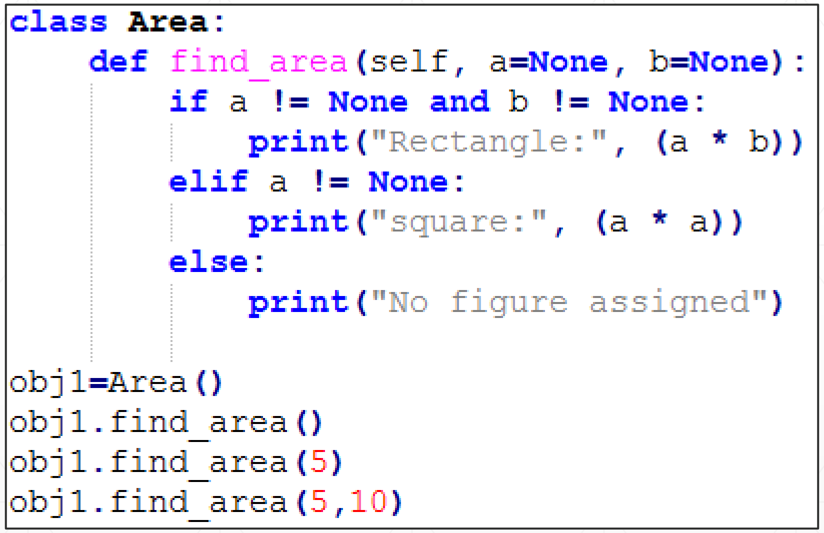
```
class Poly:
    def show(self, a=None, b=None):
        print(a,b)

p = Poly()
p.show()
p.show(2)
p.show(2,4)
```

- Accepting zero, one or two arguments for the same function for this example

What is Polymorphism in OOP

- **Example 7**

  Method Overloading



```
class Area:
    def find_area(self, a=None, b=None):
        if a != None and b != None:
            print("Rectangle:", (a * b))
        elif a != None:
            print("square:", (a * a))
        else:
            print("No figure assigned")

obj1=Area()
obj1.find_area()
obj1.find_area(5)
obj1.find_area(5,10)
```

- Accepting zero, one or two arguments with different handling ways using the same function for this example

HTTP request in Python

What is HTTP request

- **Definition**

  "An HTTP request is **made by a client**, to a named host, which is located on a server. The aim of the request is to access a **resource on the server**."

- **Examples of resource**
  - Webpage
  - Media – Image, Video, Audio, etc.
  - Data from Database

HTTP request in Python

- **Usage of requests module**
  - **Request:** The `requests` module enables to to send HTTP requests using Python
  - **Response:** The HTTP request returns a Response Object with all the response data *(content, encoding, status, etc.)*

- **Install of requests module**

```
pip install requests
```

HTTP request in Python

- **Syntax of using requests**

```
requests.methodname(params)
```

- **Methods of requests**

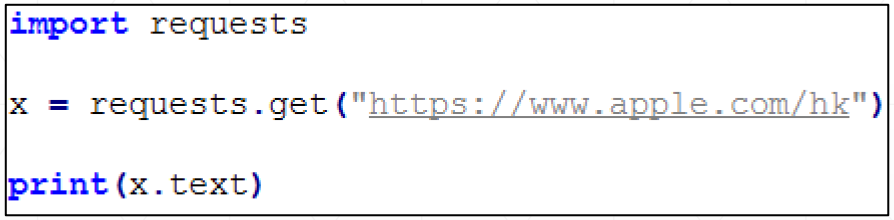
Method	**Description**
delete(*url*, *args*)	Sends a DELETE request to the specified url
get(*url, params, args*)	**Sends a GET request to the specified url**
head(*url*, *args*)	Sends a HEAD request to the specified url
patch(*url*, *data, args*)	Sends a PATCH request to the specified url
post(*url*, *data, json, args*)	**Sends a POST request to the specified url**
put(*url*, *data, args*)	Sends a PUT request to the specified url
request(*method*, *url*, *args*)	**Sends a request of the specified method to the specified url**

HTTP request in Python

- **Import requests module**

```
import requests
```

- **Example 8**



```
import requests

x = requests.get("https://www.apple.com/hk")

print(x.text)
```

- **Explanation**

  Create HTTP GET request to a web site

HTTP request in Python

- **Example 9**

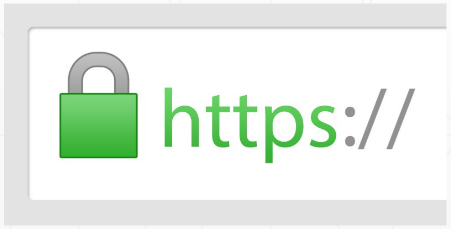
```
import requests

#payload = {'key1': 'value1', 'key2': 'value2'}
payload = {"keywordForQuickSearch": "programmer"}

x = requests.get("https://www.ctgoodjobs.hk/ctjob/listing/joblist.asp", params=payload)
#https://www.ctgoodjobs.hk/ctjob/listing/joblist.asp?keywordForQuickSearch=programmer

print(x.url)
print(x.text)
```

- **Explanation**

Create HTTP GET request with parameters to a web site

HTTP request in Python

- **More aspects to be considered**
- How to call HTTP POST request?
- How to handle a response in JSON format?



Thank you