Python

WEEK 4

Object Oriented Programming

AI Academy

COMPUTER PROGRAMMING WITH PYTHON

Instructor - James E. Robinson, III

Teaching Assistant - Travis Martin

AI Academy

NC STATE

LIGHTNING REVIEW

- Variables
- Input / Output
- Expressions
- Functions
- Conditional Control
- Looping
- Data Types
- Logging
- Functions
- Scope
- Decorators
- Recursion
- Dynamic Prg
- Exceptions

TOPICS COVERED

- Classes
- Objects
- Encapsulation
- Public v/s Private
- Dunder Methods
- Working with Instances
- Inheritance

OBJECT ORIENTED PROGRAMMING

FOCUS IS ON CREATING AND MANIPULATING OBJECTS

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or objects, rather than functions and logic.

In OOP we work with Objects as defined with Classes, Methods, and Attributes.

Procedural programming vs OOP
While procedural programming focuses to create functions that operate on the program's data, OOP leans towards encapsulating data and manipulating objects

OBJECT ORIENTED PROGRAMMING

ELEMENTS OF OBJECT ORIENTED PROGRAMMING

Class
It is a set of statements that define the methods and attributes of an object

Object
It is an *instance of a class* that is defined by its own attributes and methods

Attribute
It defines the data and values (variables) that the object holds

Method
It defines the procedures (functions) that an object can perform

CLASSES

CLASSES ARE THE BLUEPRINTS FOR INDIVIDUAL OBJECTS

CLASS

```
class Car:
    def __init__(self, year, model, color):
        self.year = year
        self.model = model
        self.color = color

    def changeColor(self,color):
        self.color = color

my_car = Car(2002,’X’,’black’)
```

ATTRIBUTES

METHOD

OBJECT

self Parameter

It is a parameter required in every method of a class to reference the current object

Note:

A class can have multiple methods

A class can have multiple instances

OBJECTS

AN OBJECT IS AN INSTANCE OF A CLASS

Creating object

my_car = Car(2002 , ‘X’ , ’black’)

Accessing Attributes:

year = my_car.year # 2002
model = my_car.model # ‘X’
color = my_car.color # ‘black’

Using Methods:

my_car.changeColor(‘green’)
# changes the color attribute value
# of my_car to green

Note:

For accessing/using any data/method related to an object, the dot(.) notation is used

Attribute values are initialized when creating an object, depending on the __init__ method defined in the class

Attributes can be added or updated using methods of the object

ENCAPSULATION

ALL IMPORTANT INFORMATION IS CONTAINED INSIDE AN OBJECT

Encapsulation is a principle of OOP that states that all data and code is encapsulated within an object and only select information is exposed

Data hiding

Object's attributes are hidden from code outside the object

- Access *restricted* to the object's methods
- Protects from accidental corruption
- Outside code does not need to know internal structure of the object

Object reusability

The same object can be used in different programs

- Example: the Logger class; reusable, flexible, and customizable

PUBLIC V/S PRIVATE

ATTRIBUTES AND METHODS OF A CLASS CAN BE PUBLIC OR PRIVATE

Public

These attributes / methods can be accessed by outside code through the object

Private

These attributes / methods can only be accessed by other methods of the same object and are not visible to outside code

Private information is defined by double underscores ‘ __ ’

Note:

Implicit methods like dunder methods are always private.

Example

```
class Car:
    def __init__(self, year, color):
        self.year = year
        self.color = color
        self.__engineNum = 101
    def changeColor(self,color):
        self.color = color
    def __newEngine(self):
        self.__engineNum = 202
    def engineUpgrade(self):
        print("Old Engine=", self.__engineNum)
        self.__newEngine()
        print("New Engine=", self.__engineNum)
```

Public Attributes

Private Attribute

Dunder Method

Public Method

Private Method

Public Method accessing private information

AI Academy

NC STATE

DUNDER METHODS

SPECIAL METHODS INVOKED INTERNALLY UNDER CERTAIN CIRCUMSTANCES

Dunder Methods also called Magic Methods are not defined by users but are predefined in the python language.
The most commonly used dunder methods in OOP are:

__init__()

- Initializes essential attributes of an object
- Invoked when object is created in outside code

__str__()

- Defines the state of the object to outside code
- Invoked when object is printed using print statement

DUNDER METHODS

EXAMPLE CODE

```
class Car:
    def __init__(self, year, model, color):
        self.year = year
        self.model = model
        self.color = color
    def __str__(self):
        return f"A {self.year}-{self.model} in {self.color}"

this_car = Car(2002,"BMW","black")
print(this_car)
```

OUTPUT:

```
A 2002-BMW in black
```

Note:

__str__ requires the code to return a string

__init__ defines the parameters required to create an object

WORKING WITH INSTANCES

EACH INSTANCE OF A CLASS IS DISTINCT AND HAS ITS OWN INFORMATION

```
class Car:
    def __init__(self, year, model, color):
        self.year = year
        self.model = model
        self.color = color
```

Mutator Method

```
    def changeColor(self,clr):
        self.color = clr
        print("---Changed Color--- \n")
```

Accessor Method

```
    def displayCar(self):
        print(f"Model : {self.year}")
        print(f"Color : {self.color} \n")

car1 = Car(2002,"BMW","black")
car2 = Car(1998,"Merc","red")
```

```
car1.displayCar()
car2.displayCar()

car1.changeColor("blue")
car1.displayCar()
car2.displayCar()

car2.changeColor("black")
car1.displayCar()
car2.displayCar()
```

OUTPUT:

```
Model : 2002
Color : black

Model : 1998
Color : red

---Changed Color---

Model : 2002
Color : blue

Model : 1998
Color : red

---Changed Color---

Model : 2002
Color : blue

Model : 1998
Color : black
```

INHERITANCE

CLASSES CAN REUSE CODE FROM OTHER CLASSES BASED ON RELATIONSHIPS

Super Class

```
class Vehicle:
    def __init__(self, make, model, price):
        self.make = make
        self.model = model
        self.price = price
    def get_make(self):
        return self.make
```

Sub Class

```
class Car(Vehicle):
    def get_price(self):
        tax = 1000
        return self.price + tax
```

```
my_car = Car(‘BMW’,’B’,5000)
```

```
my_car.get_make()
```

OUTPUT:

```
BMW
```

```
my_car.get_price()
```

OUTPUT:

```
6000
```

Note:

Inheritance is a “Is a” Relationship

A SubClass can only use the methods not alter

A SubClass can use any method from the Superclass but not vice versa

INHERITANCE - SUPER KEYWORD

ALLOWS US TO ACCESS METHODS OF THE BASE CLASS

```
class Vehicle:
    def __init__(self, make, model, price):
        self.make = make
        self.model = model
        self.price = price
    def get_price(self):
        return self.price

class Car(Vehicle):
    def __init__(self, make, model, price, color):
        super().__init__(make, model, price)
        self.color = color
    def get_color(self):
        return self.color
```

WEEK SUMMARY

- Learned the concept of Object Oriented Programing
- Learned about classes, objects, methods and attributes
- Learned the concept of encapsulation and data hiding
- Learned how to keep data private and make data public in classes
- Learned about dunder methods
- Learned the concept of inheritance

THANK YOU

FOR ADDITIONAL QUERIES OR DOUBTS CONTACT:
jerobins@ncsu.edu

AI Academy