Object Oriented Programming (Using Python)

UNIT- I (Part-2) cont'd.

Classes and Objects:

- Prototyping
- Referencing the variables in functions
- Inline, static and friend functions
- Memory allocation for classes and objects
- Arrays of objects

Prof. R. MADANA MOHANA
Professor, Artificial Intelligence & Data Science
http://rmadanamohana.com/

Prototyping

- Prototyping in Python classes involves creating an initial version of a class that can be used to test and refine its design before finalizing it.
- This approach can help developers identify potential issues and make necessary changes in a timely manner, improving the overall quality of the code.

Prototyping

Here is an example of how to prototype a Python class:

```
class Car:
    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.speed = 0
    def accelerate(self):
        self.speed += 10
    def brake(self):
        if self.speed >= 10:
            self.speed -= 10
```

Prototyping

Example of how to prototype a Python class: cont'd.

```
    def current_speed(self):
        return self.speed
car1 = Car("TATA", "Nexon", 2023)
# Prototype testing
print(car1.make)  # Output: TATA
print(car1.model)  # Output: Nexon
print(car1.year)  # Output: 2023
car1.accelerate()
print(car1.current_speed())  # Output: 10
car1.brake()
print(car1.current_speed())  # Output: 0
```

Prototyping

Example of how to prototype a Python class: cont'd.

```
Output:
TATA
Nexon
2023
10
0
```

Prototyping

- In this example, we define a '**Car**' class with some basic attributes and methods for acceleration, braking, and reporting the current speed.
- We create an instance of the class (**'car1'**) and use it to test the prototype by calling its methods and attributes.
- Once we have tested and refined the prototype, we can finalize the class and use it in our application.

Referencing the variables in functions

- To reference variables in functions of classes in Python, you need to use the **self** keyword.
- The **self** keyword is used to reference the current instance of the class, allowing you to access and modify its variables and methods.

Referencing the variables in functions

Example of how to reference variables in functions of a class in Python

```
class MyClass:
    def __init__(self, my_var):
        self.my_var = my_var
    def my_function(self):
        print(self.my_var)
my_object = MyClass("Hello World!")
my_object.my_function()
Output:
Hello World!
```

Referencing the variables in functions

- In this example, we define a **MyClass** class with a constructor that takes a parameter **my_var**. We set **my_var** to an instance variable of the class using **self.my_var = my_var**.
- We also define a function **my_function** that uses **self.my_var** to access the instance variable **my_var** and print its value.
- Finally, we create an instance of the class **my_object** and call its **my_function** method, which prints the value of **my_var** ("Hello World!").

Referencing the variables in functions

- Note that you must use '**self**' to reference instance variables in any function within a class. If you do not use '**self**', Python will look for a local variable with that name and raise an error if it does not exist.

Inline, static and friend functions

- In Python, there are no inline, static or friend functions in classes like there are in some other programming languages like C++.
- Inline functions are used in C++ to optimize the performance of code by copying the function's code to the location where it is called instead of executing a function call.
- Python does not have inline functions because it uses dynamic typing, which means that the function's code needs to be evaluated at runtime, making inline functions unnecessary.

Inline, static and friend functions

- Static functions in C++ are used to group functions together in a class that do not need to access any of the class's instance variables or methods.
- In Python, functions can be defined outside of a class and do not need to be part of a class to be called.
- Friend functions in C++ are used to grant access to private members of a class to a function that is not a member of the class.
- Python does not have the concept of private members like C++, so there is no need for friend functions.

Inline, static and friend functions

- However, in Python, you can define methods and properties within a class using the **@staticmethod** and **@property** decorators, respectively, to achieve similar functionality.
- The **@staticmethod** decorator allows you to define a method that does not access any instance variables or methods and can be called directly on the class itself.

Inline, static and friend functions

Here is an example:

```
class MyClass:
    @staticmethod
    def my_static_method():
        print("This is a static method")


MyClass.my_static_method()
```

Output:

```
This is a static method
```

Inline, static and friend functions

The @property decorator allows you to define a method that is accessed like an attribute instead of a method call. Here is an example:

```
class MyClass:
    def __init__(self):
        self._my_var = "Hello World!"
    @property
    def my_var(self):
        return self._my_var
my_object = MyClass()
print(my_object.my_var)  # Output: Hello World!
```

Inline, static and friend functions

In this example, we define a class **MyClass** with a private instance variable **_my_var**. We define a method **my_var** with the **@property** decorator that returns the value of **_my_var**.

When we create an instance of the class **my_object** and call **my_object.my_var**, we get the value of **_my_var** ("Hello World!") without needing to call a method.

Memory allocation for classes and objects

- In Python, memory is dynamically allocated as needed during runtime. When a new object is created, memory is allocated for that object. For classes, memory is allocated when the class is defined.
- When you create an instance of a class, memory is allocated for that instance. The amount of memory allocated depends on the size of the object and the amount of data it contains. For example, an object that contains a large amount of data will require more memory than an object that contains only a few data elements.

Memory allocation for classes and objects

- Python also has a garbage collector that automatically frees memory that is no longer being used by the program. This means that you don't need to worry about manually deallocating memory in Python.

Here is an example of creating a class and an instance of that class:

```
class MyClass:
    def __init__(self, x, y):
        self.x = x
        self.y = y
my_instance = MyClass(1, 2)
```

Memory allocation for classes and objects

In this example, memory is allocated for the **MyClass** class when it is defined. When an instance of the class is created with **my_instance = MyClass(1, 2)**, memory is allocated for **my_instance**, which contains the **x** and **y** attributes.

Memory allocation for classes and objects

- It's worth noting that Python uses a reference counting mechanism for memory management.
- When an object is no longer referenced by any part of the program, its memory is automatically freed. However, in some cases, circular references between objects can prevent the reference counting mechanism from freeing up memory. To handle this, Python uses a garbage collector that periodically checks for and cleans up circular references.

Arrays of objects

- In Python, you can create arrays of objects of a class by using a list.
- A list in Python is a collection of objects that are ordered and changeable.
- To create a list of objects of a class, you can instantiate the class multiple times and add each instance to the list.

Arrays of objects

Example:

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
person_list = []
person1 = Person("Ram", 25)
person2 = Person("Bob", 30)
person3 = Person("Charlie", 35)
person_list.append(person1)
person_list.append(person2)
person_list.append(person3)
```

Arrays of objects

Example:

In this example, we create a **Person** class with **name** and **age** attributes. We then create an empty list called **person_list**. We create three instances of the **Person** class and add them to the **person_list** using the **append()** method.

You can access the objects in the list using their index. For example, to access the first object in the **person_list**, you can use **person_list[0]**.

Arrays of objects

You can also iterate over the list of objects using a for loop.

Here is an example:

```
for person in person_list:
    print(person.name, person.age)
```

This will print out the name and age of each person in the **person_list**.

Arrays of objects

- Arrays of objects in Python can also be implemented using the NumPy library.
- NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions. It is commonly used for scientific computing and data analysis.