كلية العلوم
قسم الانظمة الطبية الذكية
محاضرة مشتركة مع استاذ زائر

Lecture: (5)

Objects, Classes, and Relationships

Subject: Object oriented programming I

Class: Second

Dr. Maytham N. Meqdad , Lecturer Amal Khalil Awad

Python Classes and Objects

Python Classes/Objects

Python is an object oriented programming language.

Almost everything in Python is an object, with its properties and methods.

A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
```

==========

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

The __init__() Function

The examples above are classes and objects in their simplest form, and are not really useful in real life applications.

To understand the meaning of classes we have to understand the built-in __init__() function.

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example

Create a class named Person, use the __init__() function to assign values for name and age:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

The __str__() Function

The __str__() function controls what should be returned when the class object is represented as a string.

If the __str__() function is not set, the string representation of the object is returned:

Example

The string representation of an object WITHOUT the __str__() function:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("John", 36)

print(p1)
```

Example

The string representation of an object WITH the __str__() function:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def __str__(self):
    return f"{self.name}({self.age})"

p1 = Person("John", 36)

print(p1)
```

Object Methods

Objects can also contain methods. Methods in objects are functions that belong to the object.

Let us create a method in the Person class:

Example

Insert a function that prints a greeting, and execute it on the p1 object:

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("Hello my name is " + self.name)

p1 = Person("John", 36)
p1.myfunc()
```

Note: The `self` parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

The self Parameter

The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

It does not have to be named `self` , you can call it whatever you like, but it has to be the first parameter of any function in the class:

Example

Use the words *mysillyobject* and *abc* instead of *self*:

```
class Person:
  def __init__(mysillyobject, name, age):
    mysillyobject.name = name
    mysillyobject.age = age

  def myfunc(abc):
    print("Hello my name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

You can modify properties on objects like this:

Example

Set the age of p1 to 40:

```
p1.age = 40
```

Delete Object Properties

You can delete properties on objects by using the `del` keyword:

Example

Delete the age property from the p1 object:

```
del p1.age
```

Delete Objects

You can delete objects by using the `del` keyword:

Example

Delete the p1 object:

```
del p1
```

The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

Example

```
class Person:
  pass
```

Examples (python classes and objects)

1. Creating a Simple Class and Object:

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def bark(self):
        print(f"{self.name} says Woof!")

# Create objects of the Dog class
dog1 = Dog("Buddy", 2)
dog2 = Dog("Molly", 4)

# Access object attributes and methods
print(f"{dog1.name} is {dog1.age} years old.")
dog2.bark()
```

2. Bank Account Class:

```
def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
        else:
            print("Insufficient funds.")

    def get_balance(self):
        return self.balance

# Create a bank account object and perform transactions
account = BankAccount("12345", 1000)
account.deposit(500)
account.withdraw(200)
print(f"Account balance: ${account.get_balance()}")
```

3. Car Class with Inheritance:

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

    def display_info(self):
        print(f"Make: {self.make}, Model: {self.model}")

class Car(Vehicle):
    def __init__(self, make, model, year):
        super().__init__(make, model)
        self.year = year

    def display_info(self):
        print(f"Make: {self.make}, Model: {self.model}, Year: {self.year}")

# Create car objects and call methods
car1 = Car("Toyota", "Camry", 2022)
car2 = Car("Honda", "Civic", 2021)
car1.display_info()
car2.display_info()
```

4. Student Class with Multiple Objects:

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def display_info(self):
        print(f"Name: {self.name}, Age: {self.age}")

# Create a list of student objects and display their information
students = [Student("Alice", 20), Student("Bob", 22), Student("Charlie", 19)]

for student in students:
    student.display_info()
```

5. Rectangle Class:

```
class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

# Create rectangle objects and calculate area and perimeter
rect1 = Rectangle(5, 3)
rect2 = Rectangle(8, 4)

print("Rectangle 1 - Area:", rect1.area(), "Perimeter:", rect1.perimeter())
print("Rectangle 2 - Area:", rect2.area(), "Perimeter:", rect2.perimeter())
```

6. Bank Customer Class with Account Management:

```
class BankCustomer:
    def __init__(self, name):
        self.name = name
        self.accounts = {}

    def add_account(self, account_name, balance):
        self.accounts[account_name] = balance

    def display_accounts(self):
        print(f"Accounts for {self.name}:")
        for account, balance in self.accounts.items():
            print(f"{account}: ${balance:.2f}")

# Create a bank customer, add accounts, and display account information
customer = BankCustomer("Alice")
customer.add_account("Savings", 1000)
customer.add_account("Checking", 500)
customer.display_accounts()
```

7. Circle Class with Method Overriding:

```
import math

class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius ** 2

    def perimeter(self):
        return 2 * math.pi * self.radius

    def display_info(self):
        print(f"Circle - Radius: {self.radius:.2f}, Area: {self.area():.2f},
Perimeter: {self.perimeter():.2f}")

class ColoredCircle(Circle):
    def __init__(self, radius, color):
        super().__init__(radius)
        self.color = color

    def display_info(self):
        print(f"Colored Circle - Radius: {self.radius:.2f}, Area:
{self.area():.2f}, Perimeter: {self.perimeter():.2f}, Color: {self.color}")

# Create circle objects and call methods
circle1 = Circle(5)
circle2 = ColoredCircle(3, "Red")

circle1.display_info()
circle2.display_info()
```