CLASSES

Object-Oriented Programming

Classes and Objects (1 of 2)

- Programs grow in size, become more complex
- Dependencies and interrelationships throughout code increases
- Partial solution to this problem is **data encapsulation**
  - *Within a unit, as much implementation detail as possible is hidden*
- Programmer using an object is concerned only with
  - *Tasks that the object can perform*
  - *Parameters used by these tasks*

Classes and Objects (2 of 2)

- 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.

Built-in Classes (1 of 3)

Ex1_buildinClasses.py

- Program identifies types of objects
  - *Note use of word "class" instead of "type"*
  - *We refer to a specific literal from one of these classes as an instance of the class*
- Examples:

```
s = "Hello World!"
L = [1, 2, 3]
print(type(s))
print(type(L))
```

[Run]

```
<class 'str'>
<class 'list'>
```

```
1  s = "Hello World!"
2  #using built-in functions
3  print(type(s))
4  print(id(s))
5  print(len(s))
6  #using string methods
7  print(s.upper())
8  print(s.lower())
9  print(s.split())
10 print()
```

Built-in Classes (2 of 3)

- All **strings** are instances of the class ***`str`***
  - *Although each object holds its own string value*
  - *All strings have the same methods*
- All **integers** are instances of the class ***`int`***
- All **floating point numbers** are instances of the class ***`float`***

Built-in Classes (3 of 3)

- **List** is a collection which is ordered and changeable. Allows duplicate members.
  - *All lists are instances of the class* ***`list`***
- **Tuple** is a collection which is ordered and unchangeable. Allows duplicate members.
  - *All tuples are instances of the class* ***`tuple`***
- **Dictionary** is a collection which is ordered and changeable. No duplicate members.
  - *All dictionaries are instances of the class* ***`dict`***

User-defined Classes (1 of 7)

- Each class we define will
  - *Have a specified set of methods*
- Each object (instance) of the class
  - *Will have its own value(s)*
- Class definitions have the general form

```
class ClassName:
    indented list of methods for the class
```

User-defined Classes (2 of 7)

Ex2_student.py

- Methods defined much like ordinary functions
  - *Methods have `self` as their first parameter.*
  - *Each method's `self` parameter references the object*
  - *`self` must come first before the other parameters.*
  - *Every method call associated with a class automatically passes `self`, it gives the individual instance access to the attributes and method*

```
1   class ETStudent:
2       def __init__(self, name, id):
3           self.name = name
4           self.id = id
5
6       def display(self):
7           prefix = 'QCC-ET-'
8           print("Name:",self.name.title())
9           print(f"Id: {prefix}{self.id}")
10
11  s1 = ETStudent("john smith",123456)
12  s1.display()
```

```
Python - Ex2_student.py:11 ✓

Name: John Smith
Id: QCC-ET-123456
```

User-defined Classes (3 of 7)

Ex3_dog.py

- Classes are templates from which objects are created
  - *Specifies properties, methods that will be common to all objects, instances of that class*

dog.py

```
1  class Dog:
2      """A simple attempt to model a dog."""
3
4      def __init__(self, name, age):
5          """Initialize name and age attributes."""
6          self.name = name
7          self.age = age
8
9      def sit(self):
10         """Simulate a dog sitting in response to a command."""
11         print(f"{self.name} is now sitting.")
12
13     def roll_over(self):
14         """Simulate rolling over in response to a command."""
15         print(f"{self.name} rolled over!")
16
```

PE11_1

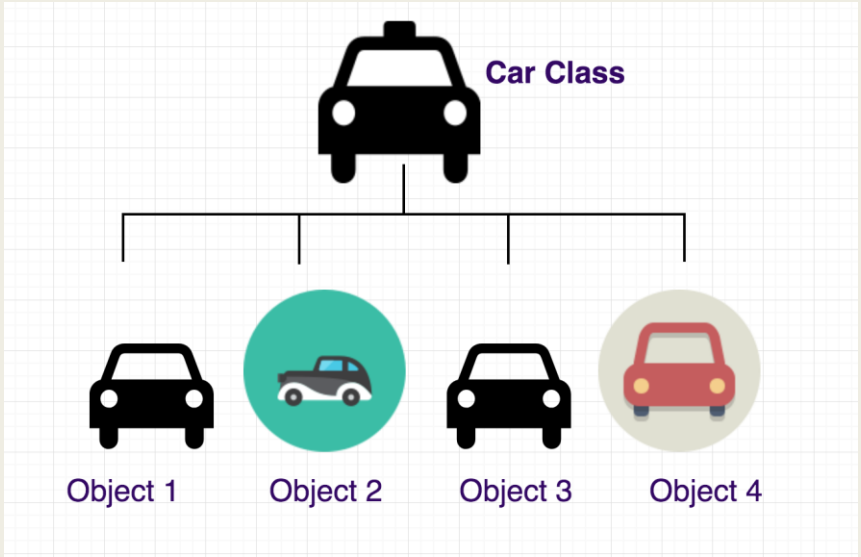
- Write your codes and run

User-defined Classes (4 of 7)

Ex4_car.py

- All classes have a method called `__init__()`, which is always executed when the class is being initiated.
- Use the `__init__()` method to assign instance values to object properties, or other operations that are necessary to do when the object is being created.

```
1  class Car:
2      """A simple attempt to represent a car."""
3
4      def __init__(self, make, model, year):
5          """Initialize attributes to describe a car."""
6          self.make = make
7          self.model = model
8          self.year = year
9          self.odometer_reading = 0
10
```



User-defined Classes (5 of 7)

- Methods in objects are functions that belong to the object.

dog.py

```python
class Dog:
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Initialize name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dog sitting in response to a command."""
        print(f"{self.name} is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(f"{self.name} rolled over!")

```

```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        """Initialize attributes to describe a car."""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = f"{self.year} {self.make} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print(f"This car has {self.odometer_reading} miles on it.")
```

User-defined Classes (6 of 7)

importing_classes\car.py & importing_classes\my_car.py

- Classes can be
  - *Typed directly into programs*
  - *Stored in modules and brought into programs with an import statement*

```
1 > class Dog:⋯

16

17  my_dog = Dog('Willie', 6)
18  your_dog = Dog('Lucy', 3)
19
20  print(f"My dog's name is {my_dog.name}.")
21  print(f"My dog is {my_dog.age} years old.")
22  my_dog.sit()
23
24  print(f"\nYour dog's name is {your_dog.name}.")
25  print(f"Your dog is {your_dog.age} years old.")
26  your_dog.sit()
```

car.py | *my_car.py*

```
1  from car import Car
2
3  my_new_car = Car('audi', 'a4', 2019)
4  print(my_new_car.get_descriptive_name())
5
6  my_new_car.odometer_reading = 23
7  my_new_car.read_odometer()
```

User-defined Classes (7 of 7)

Ex5_pass.py

- 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.
- ```
  class Person:
      Pass
  ```
  ```

PE11_2 & PE11_3

- Write your codes and run

Summary (1 of 2)

- An **object** is an entity that stores data (instance variables), and has **methods** that manipulate the data.
- A **class** is a template from which objects are created.
- The header of a class definition has the form.
  `class ClassName:`
- The first method is usually an **initializer** named `__init__` that is called automatically when an object is created. It initializes the attributes of the class.
- The **first** parameter of the initializer is name `self`.
- The parameter `self` is a variable that refers to the **object itself**.

Summary (2 of 2)

- When a method is applied to an object, the object itself is **implicitly** passed to the `self` parameter of the method definition.
- Data are stored in instance variables and accessed by methods are called **mutators** (change values of the instance variable) and **accessors** (retrieve values of instance variables).
- The statement `objectName = className (arg1, arg2, …)` is said to **instantiate** the object.
- A class variable is visible to all instances of a class and does not vary from instance to instance.
- The process of bundling together data and methods that operate on the data, while hiding the implementation of the methods, is called **encapsulation**.
- Python has many **built-in** classes such as *`int`, `float`, `str`, `list`, `tuple`, `bool`, `dict`, and `set`*.

Classes Terminologies

- ❑ 1 Attribute
- ❑ 2 Accessors
- ❑ 3 Built-in Classes
- ❑ 4 Encapsulation
- ❑ 5 _Init_
- ❑ 6 Initializer
- ❑ 7 Instance
- ❑ 8 Instance Variables
- ❑ 9 Instantiate
- ❑ 10 Method
- ❑ 11 Mutators
- ❑ 12 Mutators
- ❑ 13 Object
- ❑ 14 Object-Oriented Programming
- ❑ 15 Self
- ❑ 16 User-defined Classes

- ***Inheritance*** *allows a* ***new class*** *to be created from an* ***existing class*** *and to inherit its instance variables and methods*
- ***Existing Class:*** *Superclass / Parent Class / Base Class*
- ***New Class:*** *Sub Class / Child Class / Derived Class*
- *A method defined in a child class with the same name as a method in its parent class* ***overrides*** *the parent's method.*
- ***Polymorphism*** *is the ability to use same syntax for objects of different types. Every OOP language allows two classes to use the same method name but with different implementation.*

Quiz 11

- Quiz 11A has 10 questions in 15 minutes, 10 pts
  - *10 multiple choice/true or false questions, 1 pt. for each question*
  - *Quiz 11A has two attempt, the higher grade will be selected*
  - *Submit Quiz 11A (at least 1-minute) **before** the due time to Blackboard*

- Quiz 11B has 2 code questions, 15 pts
  - *Write the Python code based on the given question*
  - *Each question will be given during the last 10-minute of each session of week 11*
  - *Quiz 11B-1 on session A, and Quiz 11B-2 on session B*
  - *Quiz 11B has one attempt*

DB 11

- Instruction:
  1) Choose any **three terminologies** from the PowerPoint this week or last week. Please **avoid** selecting the exact same terms. Make sure to indicate the **terms** you're working on in the thread title as soon as you open your thread. Then you can **explain and edit your terms** (1.2 pt).
  2) There are **three** ways to **pass arguments** to parameters in a function. List them and give an example of each type (0.3 pt).
  3) Submit your posts before the due date. Let's learn from each other.