COSC 2306
Data Programming

OOP and Class

Homework Assignment 01

- Due Date: **Sep. 29, 2025, 11:59 PM**
- Please submit your codes in **one .py file** and any **additional information** included as **comments at the top** of the file.
- Please name your file using the format: "**Lastname_Firstname_HW01.py**"
- All codes should be written and executable in **Python** (using **PyCharm IDE** is recommended and please make a note if you used a different IDE).
- Please **add comments** to help the grader understand -- this is important as you may still get partial credit if there is syntax error (if the output is wrong due to syntax error and you don't have comments to help TA to understand your logistics, you may not get partial credit).
- Using **AI is strictly forbidden** for all homework assignments and we will use AI detection tools to check. Once detected, you will lose all points of an assignment and may also face penalty of losing additional points.

Object-Oriented Framework

- Classes and objects are the two main aspects of object oriented programming
- A **class** creates a new *type* using the keyword class
- Where **objects** are *instances* of the class
- An analogy: we can have variables of type int -- variables that store integers are instances (objects) of the int class

Define a class:

```
class Dog:
    pass
```

Instantiate an object:

```
a = Dog()
b = Dog()
```

Output?

```
a == b

>>>False
```

Object-Oriented Framework

- Objects can store data using ordinary variables that *belong* to the object
- Variables that belong to an object or class are called **attributes**
- Objects can also have functionality by using functions that *belong* to the class (such functions are called **methods**)
- This terminology is important because it helps us to differentiate between 1) a function which is separate by itself, 2) a method which belongs to an object

Attributes

- Remember that attributes are of two types
  - they can belong to each instance (object) of the class
  - or they belong to the class itself
  - they are called instance attributes (variables) and class attributes (variables), respectively

```
class Student:
    # Class attributes
    grade = "Freshman"

    def __init__(self, name, ID):
        # Instance attributes
        self.name = name
        self.ID = ID
```

The attributes and methods of the class are listed in an indented block

Magic/special/dunder methods: predefined methods that have names starting and ending with **double underscores**.
They are **automatically called** by Python in response to certain operations on objects.

The self

Class methods have only one specific difference from ordinary functions

- they have an extra variable that has to be added to the beginning of the parameter list
- but we do **not** give a value for this parameter when we call the method
- this particular variable refers to the object itself
- and by convention, it is given the name self

The self

- Although, we can give any name for this parameter, it is *strongly recommended* that we use the name self
- Any other name is definitely frowned upon
- There are many advantages to using a standard name
  - any reader of our program will immediately recognize that it is the object variable (i.e. the self)
  - IDEs (Integrated Development Environments) can help us if we use this particular name

The self

- Python will automatically provide this value in the function parameter list.
- For example, if we have a class called **MyClass** and an instance (object) of this class called **MyObject**, then when we call a method of this object as MyObject.method(arg1, arg2), this is automatically converted to MyClass.method(MyObject, arg1, arg2).
- This is what the special self is all about.

The __init__ method

- The constructor of the class: instantiating an object
  - __init__ is the first method defined for the class -- it's the first piece of code executed in a newly created instance of the class
  - __init__ is called **instance attributes** (instance specific)

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

Class attributes

- Class attributes
  - the same value for all class instances
  - Outside of __init__
  - always need an initial value

```
class Student:
    # Class attribute
    grade = "Freshman"

    def __init__(self, name, ID):
        # Instance attributes
        self.name = name
        self.ID = ID
```

Creating a Class

```
class Person:
    pass #placeholder and to avoid error
p = Person()
print(p)

#<__main__.Person instance at 0x816a6cc>
```

__str__() and __repr__() methods

- Use the __str__() and __repr__() methods
  - essentially convert an object into a string
  - helpful for debugging via printing useful info
  - special methods (double underscore)
  - by default: return the instance address string
  - need "better" implementation
  - __str__() often for users (human-readable info)
  - __repr__() often for developers (information rich)

__str__() and __repr__() methods

```
class Ocean:
    def __init__(self, sea_creature_name, sea_creature_age):
        self.name = sea_creature_name
        self.age = sea_creature_age
    def __str__(self):
        return f"The creature type is {self.name} and the age is {self.age}"
    def __repr__(self):
        return f"Ocean(\'{self.name}\', {self.age})"

c = Ocean('Jellyfish', 5)
print(c.__str__())
print(str(c))
print(c.__repr__())
print(repr(c))
print(c)
```

```
The creature type is Jellyfish and the age is 5
The creature type is Jellyfish and the age is 5
Ocean('Jellyfish', 5)
Ocean('Jellyfish', 5)
The creature type is Jellyfish and the age is 5
```

Q1: what if “def __str__(self):” is missing?
Q2: what if “def __repr__(self):” is missing?

A1: str would default to repr
A2: repr would NOT default to str (print instance address instead)

__str__() and __repr__() methods

```
import datetime
mydate = datetime.datetime.now()

print("__str__() string: ", mydate.__str__())
print("str() string: ", str(mydate))
Print(mydate)

print("__repr__() string: ", mydate.__repr__())
print("repr() string: ", repr(mydate))
```

```
__str__() string: 2023-01-27 09:50:37.429078
str() string: 2023-01-27 09:50:37.429078
2023-01-27 09:50:37.429078

__repr__() string: datetime.datetime(2023, 1, 27, 9, 50, 37, 429078)
repr() string: datetime.datetime(2023, 1, 27, 9, 50, 37, 429078)
```

Object Methods

```
class Person:
    def sayHi(self):
        print('Hello, how are you?')
p = Person()
p.sayHi()
```

#Is this correct?

p = Person().sayHi()

Person().sayHi() returns None
i.e., print (p) -> None

```
>>> Hello, how are you?
```