CST0006 – Computer Programming Foundations

INTRODUCTION to WEEK10

In this lecture and lab section we are going to dive into object oriented programming and break apart the main concepts, theories and why it is so popular. This is a HUGE topic to cover so we're only going to go though some of the main features.

After this lecture and lab section, you will have a general understanding of object oriented programming and be able to write a program that uses objects.

Basic Understanding

Object Oriented Programming is a programming paradigm where we create new data types that mimic real world objects. Each object having its own attributes and behaviours. Another way of looking at it is we create objects (new data types) that can send and receive messages to allow them to interact with each other. It's important to understand that this is a programming paradigm, and it's available to you if you want to use it. There is no rule that says you have to, unless the company you work for demands this approach.

Classes and Objects

The popular way to describe the relationship between a class and an object is that a class is like a blueprint for building a house. The house is the object that gets created from the blueprint, and you can create hundreds or even thousands of houses from one blueprint. The same relationship exists between a class and an object. For example, the house class has all the details within it to allow you to build a house object and just like you cant live in a blueprint of a house, a class is only the description of what attributes and behaviours you need to build a house object. In this way it's the same as a cake recipe and a cake. You can't eat a recipe for a cake, but you can make many cakes from one cake recipe. The term for creating an object from a class is called instantiation.

Properties and Methods

Each class, (the blueprint of an object), will have associated attributes (data types) and behaviours (functionality) built into the class. These attributes and behaviours are referred to as properties and methods.

Properties are the variables that the class needs for it to represent the data members the class holds.

Methods are the functions within the class that represents the behaviours of the object.

When you create a class, you will decide what features are important for your class that you need to represent within that class. If you don't need to have a brand name of a shoe but the colour is important, then you will only need to make the colour property. Likewise, if all of your shoes won't be tied, then you don't need to make a "tie_up" method.

Example Properties vs Methods

Object type	Properties	Methods
Shoe	Colour: Blue Size: 11	Tie Untie
Die (Dice)	NumSides: 6	Roll (SetNumber) GetNumber
PlayingCards	NumCards: 52	Shuffle Deal

The terminology changes depending on which book you read or which tutorial you find online. Each resource will have their own way of describing the properties and methods of a class.

Properties:	**Methods:**
Attributes	Behaviours
Data Members	Functionality
Fields	Etc...

OOP Concepts

Object Oriented Programming considers code reuse a high priority and is built directly into the language.

There are four key concepts that are used to describe object oriented functionality. Unfortunately as with any terminology, there are many different descriptions and sometimes blurred lines by which these terms are described.

Abstraction:	hiding complexity or implementation and showing only interface (can also mean) choosing what's important to represent in a class, like the shoe example from above
Polymorphism:	(means many forms) Allows you to create a generic class, then create more specific types
Inheritance:	Inheriting properties and methods from a main class into a newer sub class
Encapsulation:	Data hiding, or limiting access to the data properties of the class

Again, these are the general concepts. Pick any web page or book and you'll get a different description. If you can remember the four key concepts and generally what they mean, you're fine.

Code Examples

Creating a class

person.py

```
class Person:
    '''simple class example'''

    name = "nonebody"

    # set name
    def set_name(self, name):
        self.name = name

    # get name
    def get_name(self):
        return "You be " + self.name + " yo!"
```

Instantiating a class

test.py

```
from person import Person

p1 = Person()
```

Using the methods in a class

test2.py

```
from person import Person

p1 = Person()

p1.set_name("Chuck")

print(p1.get_name())
```

Accessing the Data Members Directly

test3.py

```
from person import Person

p1 = Person()

p1.set_name("Chuck")

p1.name = "Charles the poohead"

print(p1.get_name())
```

OH NO! THAT'S BAD! Usually we will want to protect the data members of the class so that they can't be accessed from outside of the class directly. We've gone through the trouble of creating getter and setter methods so that the outside world can access the data members through our own methods. Poor Charles is now a poohead because we didn't protect the data members.

Access Specifiers

Some languages actually use public, protected and private as access specifiers, Python uses underscores to set the access level. One underscore sets the variable to protected, two underscores sets the variable to private, ...none and Charles is a poohead.

Variable Name	Access Level	Description
`name`	Public	Can changed inside and outside the class
`_name`	Protected	Can be changed from the class and a subclass
`__name`	Private	Can only be changed inside class

Making the Data Members Private

`person.py`

```
class Person:
    '''simple class example'''

    __name = "nonebody"

    # set name
    def set_name(self, name):
        self.__name = name

    # get name
    def get_name(self):
        return "You be " + self.__name + " yo!"
```

Accessing the Data Members Directly

`test4.py`

```
from person import Person

p1 = Person()

p1.set_name("Chuck")

p1.__name = "Charles the poohead"

print(p1.get_name())
```

Yay! We protected Charles' good name! :)