CDS 230
Modeling and Simulation I

Module 6

Classes and Object-Oriented Programming

Dr. Hamdi Kavak
http://www.hamdikavak.com
hkavak@gmu.edu

GEORGE MASON UNIVERSITY

Center for Social Complexity

Object Oriented Programming (OOP)

- A programming paradigm that considers everything as an object (vs. procedural programming).
- Enables more organized and easy-to-collaborate coding experience.
- Instead of functions, we now have **classes** that contain **functions** and **variables** in them.
- Our focus:
  - How to define our own classes to organize our code better
  - How to use classes that others provided

Defining a class

Begin the definition of a class.

User chooses the name of the class

Create a function inside of the class (a.k.a. method)

We will explain 'self' later.

There is one input to this function. ('self' is not a user input)

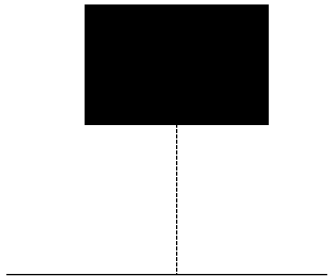
Define variables.

```
class FallingObject():
    def set_start_position(self, y):
        self.start_position = y
```

We haven't created a falling object – We just started creating the definition.

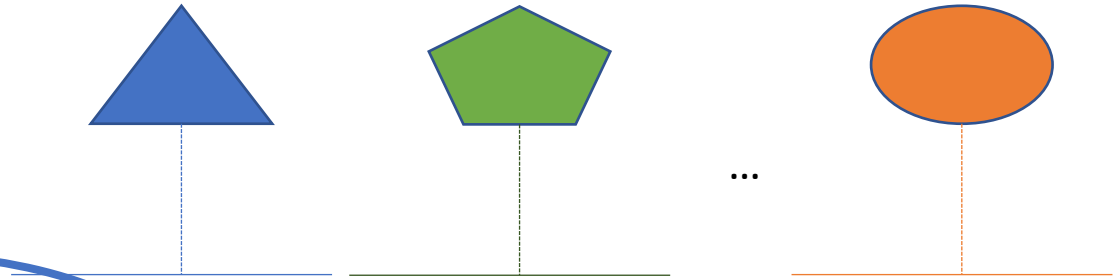
Creating an object

- Class is the definition. We use the term ***object*** to create an instance of the class. In short, one class; many objects.



Class

Creates an instance of `FallingObject` class



Objects

Function is called using . (dot) followed by the function name

```
class FallingObject():
    def set_start_position(self, y):
        self.start_position = y
```

```
triangle_object = FallingObject()
triangle_object.set_start_position(20)
print(triangle_object.start_position)
```

```
20
```

Variables within the object are accessed using . (dot) followed by the variable name

Creating many objects

- Creating more objects:

```
triangle_object = FallingObject()
triangle_object.set_start_position(20)

pentagon_object = FallingObject()
pentagon_object.set_start_position(30)

hexagon_object = FallingObject()
hexagon_object.set_start_position(40)
```

- Put objects into a list:

```
falling_objects = [triangle_object, pentagon_object, hexagon_object]
```

- Print starting positions:

```
for o in falling_objects:
    print(o.start_position)
```

```
20
30
40
```

self

- Allows us to access and define class level variables and functions

```
class FallingObject():
    start_position = 0
    def set_start_position(self, y):
        self.start_position = y
        local_variable = "hello"
```

Because of the `self` keyword, we can assign value to `start_position` variable.

No `self` keyword is used, the local variable cannot be used outside of this function.

__init__()

- Is a base function (two underscores before and two underscores after)
- We can redefine it to enable passing parameter values without calling an explicit function.

```
class FallingObject():
    start_position = 0
    def __init__(self, y):
        self.start_position = y
```

`__init__` replaces the `set_start_position` function, but why?

```
triangle_object = FallingObject(20)
print(triangle_object.start_position)
```

```
20
```

Because it allows us to pass value(s) while creating the object.