Algorithm and Data Structures
Week 3

Endang Wahyu Pamungkas, P.hD.

Class

- In OOP, data and processes are wrapped into one (encapsulation), namely in an object. Data is stored in the form of properties, and processes are stored in the form of functions or methods. In Python, properties and methods are called attributes.
- For example Circle class, the radius is a property/attribut. The process of calculating the circumference and the process of calculating the area is a method.

Class

- Python was built as a procedural language
  OOP exists and works fine, but feels a bit more "tacked on"
  Java probably does classes better than Python (gasp)

  Declaring a class:

```
class names:
    statements
```

Fields

name = value

```
class Points:
    x = 0
    y = 0

# play
p1 = Points()
p1.x = 2
p1.y = -5
```

can be declared directly inside class (as shown here) or in constructors (more common)

Python does not really have encapsulation or private fields
and on caller to "be nice" and not mess with objects' contents

Using a Class

`import` **`class`**

- client programs must import the classes they use

point_main.py

```
 1 from Point import *
 2
 3 # main
 4 other = Point()
 5 other.x = 7
 6 other.y = -3
 7 ...
 8
 9 # Python objects are dynamic (can add fields any time!)
10 p1.name = "Tyler Durden"
```

Object Methods

```
def name(self, parameter, ..., parameter):
    statements
```

- self must be the first parameter to any object method
  represents the "implicit parameter" (this in Java)
- must access the object's fields through the self reference

.

```
class Point:
    def translate(self, dx, dy):
        self.x += dx
        self.y += dy
    ...
```

Implicit Parameters (self)

- Java: `this`, implicit

```
public void translate(int dx, int dy) {
    x += dx;       // this.x += dx;
    y += dy;       // this.y += dy;
}
```

- Python: `self`, explicit

```
def translate(self, dx, dy):
    self.x += dx
    self.y += dy
```

Exercise

- Write a class to calculate the distance between two points. This class has several methods including **set_location** (to set the point), **distance_from_origin** (to calculate the distance with origin point), and **distance** (to calculate the distance with other points).

Calling Method

- A client can call the methods of an object in two ways:
  - (the value of self can be an implicit or explicit parameter)

1) **object . method (parameters)**

or

2) **Class . method (object, parameters)**

- Example:

```
p = Point(3, -4)
p.set_location(1, 5)
Point.set_location(p, 1, 5)
```

-

Constructor

```
def __init__(self, parameter, ..., parameter):
    statements
```

- a constructor is a special method with the name `__init__`

  Example:

```
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    …
```

Operator Overloading

- **operator overloading**: You can define functions so that Python's built-in operators can be used with your class.
  - See also: http://docs.python.org/ref/customization.html

Operator	Class Method
-	`__neg__(self, other)`
+	`__pos__(self, other)`
*	`__mul__(self, other)`
/	`__truediv__(self, other)`
-	`__neg__(self)`
+	`__pos__(self)`

Unary Operators

Operator	Class Method
==	`__eq__(self, other)`
!=	`__ne__(self, other)`
<	`__lt__(self, other)`
>	`__gt__(self, other)`
<=	`__le__(self, other)`
>=	`__ge__(self, other)`

x

Generating Exception

`raise` **`ExceptionType`**`("`**`message`**`")`

- useful when the client uses your object improperly

  `types: ArithmeticError, AssertionError, IndexError, NameError, SyntaxError, TypeError, ValueError`

- Example:

```
class BankAccount:
    ...
    def deposit(self, amount):
        if amount < 0:
            raise ValueError("negative amount")
        ...
```

Inheritance

```
class name(superclass):
    statements
```

- Example:

```
class Point3D(Point):    # Point3D extends Point
    z = 0
    ...
```

- Python also supports multiple inheritance

```
class name(superclass, ..., superclass):
    statements

        ...
```

Calling Superclass Method

- methods: **class.method(object, parameters)**
- constructors: **class.__init__(parameters)**

```
class Point3D(Point):
    z = 0
    def __init__(self, x, y, z):
        Point.__init__(self, x, y)
        self.z = z

    def translate(self, dx, dy, dz):
        Point.translate(self, dx, dy)
        self.z += dz
```

Exercise

- Create class vehicle with five different attributes including name, color, year_production, capacity, and mileage and one method called set_capacity
- Create a child class truck which inherit all attributes and method of vehicle class.
- Try to call the class with a dummy value.