Lecture 03: More on Class and Objects

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

February 1, 2024

Point Class

```
class Point:
    def __init__(self, xCor = 0, yCor=0):
        self.x = xCor
        self.y = yCor
p1 = Point()
print(f"({p1.x},{p1.y})")
p2 = Point(50,50)
print(f"({p2.x},{p2.y})")
```

```
(0,0)
(50,50)
```

- For each object we are writing print commands.

Lets write a method!

```
class Point:
    def __init__(self, xCor = 0, yCor=0):
        self.x = xCor
        self.y = yCor
    def show(self):
        print(f"({self.x},{self.y})")
p1 = Point()
p1.show()
p2 = Point(50,50)
p2.show()
```

```
(0,0)
(50,50)
```

Altering Attributes

- We can write methods to alter the attributes.

```
class Point:
    def __init__(self, xCor = 0, yCor=0):
        self.x = xCor
        self.y = yCor
    def show(self):
        print(f"({self.x},{self.y})")
    def increaseX(self,val):
        self.x+=val
    def increaseY(self,val):
        self.y+=val
p1 = Point(50,50)
p1.increaseX(5)
p1.increaseY(-5)
p1.show()
```

Methods may return values

```
class Rectangle:
    def __init__(self,w,h):
        self.width=w
        self.height=h
    def getArea(self):
        return self.height*self.width

r = Rectangle(4,12)
print(r.getArea())
```

Other types of fields/attributes

```
class MyList:
    def __init__(self):
        self.list = []
    def add(self,a):
        self.list.append(a)
    def show(self):
        print(self.list)

m = MyList()
m.add("Bangladesh")
m.add(2)
m.add(23.7)
m.show()
```

```
['Bangladesh', 2, 23.7]
```

Objects as parameters

```
class Point:
    def __init__(self, xCor = 0, yCor=0):
        self.x = xCor
        self.y = yCor
    def show(self):
        print(f"({self.x},{self.y})")

def distance(p1,p2):
    return ((p1.x-p2.x)*(p1.x-p2.x)+
        (p1.y-p2.y)*(p1.y-p2.y))**0.5

print(distance(Point(4,3),Point()))
```

Objects are passed as references

```
class Point:
    def __init__(self, xCor = 0, yCor=0):
        self.x = xCor
        self.y = yCor
    def show(self):
        print(f"({self.x},{self.y})")
def change(q):
    q.x=15
    q.show()
p = Point()
p.show()
change(p)
p.show()
```

Objects as fields

```
class Point:
    def __init__(self, xCor = 0, yCor=0):
        self.x = xCor
        self.y = yCor
    def show(self):
        print(f"({self.x},{self.y})")
class Line:
    def __init__(self,p1,p2):
        self.start = p1
        self.end = p2
    def show(self):
        print("Start:",end="")
        self.start.show()
        print("End:",end="")
        self.end.show()
l= Line(Point(0,0),Point(4,5))
l.show()
```

__str__ method

- The __str__() dunder method returns a reader-friendly string representation of a class object.

```
class Rectangle:
    def __init__(self,w,h):
        self.width=w
        self.height=h
    def getArea(self):
        return self.height*self.width
    def __str__(self):
        return f'width: {self.width} height: {self.height}'
r = Rectangle(4,12)
print(r)
```

```
width: 4 height: 12
```

Destructor

```
class Rectangle:
    def __init__(self,w,h):
        self.width=w
        self.height=h
        print("I am a constructor")
    def getArea(self):
        return self.height*self.width
    def __str__(self):
        return f'width: {self.width} height: {self.height}'
    def __del__(self):
        print("I am a destructor")
r = Rectangle(4,12)
del r
```

```
I am a constructor
I am a destructor
```