Lecture 07: Abstract Class and Others

Swakkhar Shatabda

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

February 21, 2024

Abstract Class and Methods

- Abstract Class is a class that is not intended to be instantiated directly, but only to be used as a base class by one or more subclasses.
- Abstract Method is a method that must be overridden in every subclass .
- Python does not have a keyword to designate a class or method as abstract. However, the Python Standard Library contains the abc module, short for abstract base class, which is designed to help developers build abstract base classes and methods.

```
from abc import ABC, abstractmethod
class <classWeWantToDesignateAsAbstract>(ABC):
```

- We then must use the special decorator @abstractmethod before any methods that must be overwritten by all subclasses.

First Abstract Class

```
from abc import ABC,abstractmethod

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

- An abstract class can not be instantiated.
- There can be multiple abstract methods in an abstract class.

```
s = Shape()
```

```
TypeError: Can't instantiate abstract class Shape
with abstract method area
```

Extending Abstract Class

- Any subclass that extends abstract class must override / implement the abstract method, otherwise it becomes an abstract class itself.

```
class Rectangle(Shape):
    def __init__(self,w=0,h=0):
        self.width=w
        self.height=h
    def area(self):
        return self.width * self.height

r = Rectangle(10,4)
print(r.area())
```

First Class Object

- A first-class object in Python is an entity with all the properties of a Python variable, like integers, strings and other default and custom Python classes.

```
def multiply(x,y):
    return x*y

v = multiply

print(v(3,4))
```

Functions are objects

- Functions can be passed as arguments to other functions.

```
def myFunc():
    print("Hello My Func")

def runFunc(x):
    x()

x = myFunc
runFunc(x)
```

Returning Functions

- Functions can return another function.

```
def customFunc(x):
    def add(x,y):
        return x+y
    def sub(x,y):
        return x-y
    if x==1:
        return add
    else:
        return sub


f = customFunc(1)
print(f(1,3))
```

List of Objects

```
class Age:
    def __init__(self,y=0,m=0):
        self.year=y
        self.month=m

a1 = Age(12,3)
a2 = Age(5,3)
a3 = Age(40,3)

listAges =[]
listAges.append(a1)
listAges.append(a2)
listAges.append(a3)
print(listAges)
```

```
[<__main__.Age object at 0x104e06b60>,
<__main__.Age object at 0x104e06d10>,
<__main__.Age object at 0x104e06cb0>]
```

List of Objects

```
class Age:
    def __init__(self,y=0,m=0):
        self.year=y
        self.month=m
    def __repr__(self):
        return f"{self.year} years {self.month}"

a1 = Age(12,3)
a2 = Age(5,3)
a3 = Age(40,3)

listAges =[]
listAges.append(a1)
listAges.append(a2)
listAges.append(a3)
print(listAges)
```

```
[12 years 3, 5 years 3, 40 years 3]
```

List of Objects

```
class Age:
    def __init__(self,y=0,m=0):
        self.year=y
        self.month=m
    def __repr__(self):
        return f"{self.year} years {self.month}"
a1 = Age(12,3)
a2 = Age(5,3)
a3 = Age(40,3)

listAges =[]
listAges.append(a1)
listAges.append(a2)
listAges.append(a3)
print(listAges)
listAges.remove(Age(12,3))
print(listAges)
```

```
ValueError: list.remove(x): x not in list
```

List of Objects

```
class Age:
    def __init__(self,y=0,m=0):
        self.year=y
        self.month=m
    def __repr__(self):
        return f"{self.year} years {self.month}"
    def __eq__(self,other):
        return self.year==other.year and self.month==other.month
a1 = Age(12,3)
a2 = Age(5,3)
a3 = Age(40,3)

listAges =[]
listAges.append(a1)
listAges.append(a2)
listAges.append(a3)
print(listAges)
listAges.remove(Age(12,3))
print(listAges)
```

Dictionary of Objects

```
class Age:
    def __init__(self,y=0,m=0):
        self.year=y
        self.month=m
    def __repr__(self):
        return f"{self.year} years {self.month}"
a1 = Age(12,3)
a2 = Age(5,3)
a3 = Age(40,3)

dictAges = {a1:"Eligible",a2:"Not Eligible",a3:"Eligible"}
print(dictAges)
print(dictAges[Age(12,3)])
```

Dictionary of Objects

```
class Age:
    def __init__(self,y=0,m=0):
        self.year=y
        self.month=m
    def __repr__(self):
        return f"{self.year} years {self.month}"
    def __hash__(self):
        return hash((self.year,self.month))
    def __eq__(self,other):
        return self.year==other.year and self.month==other.month

a1 = Age(12,3)
a2 = Age(5,3)
a3 = Age(40,3)

dictAges = {a1:"Eligible",a2:"Not Eligible",a3:"Eligible"}
print(dictAges)
print(dictAges[Age(12,3)])
```

Problem Solving

List of Rectangles

Write a code for a class Rectangle with two attributes: height and width. Create a list of Rectangles and add Rectangle objects into it. Write another function to find the Rectangle from that list with max area.

```
class Rectangle:
    def __init__(self,w,h):
        self.w=w
        self.h=h
    def area(self):
        return self.h*self.w
    def __gt__(self,other):
        return self.area() > other.area()
    def __repr__(self):
        return f"width:{self.w} height:{self.h}"

myList = [Rectangle(4,3),Rectangle(12,3),Rectangle(40,3)]
print(max(myList))
```