Python Programming

Object Oriented Programming

Mariusz Dzieńkowski

Institute of Computer Science
Lublin University of Technology
m.dzienkowski@pollub.pl

Classes

- A class defines the properties and behaviours for objects.
- Classes are a way of grupping together related data and functions which act upon that data.
- A class is a kind of data type, just like a string, integer or list.
- Class names start with capital letter.

Objects

- In Python everything is an object.
- Objects are created from classes.
- An object has a unique identity, state, and behaviour.
- When we create an object of that data type, we call it an instance of a class.
- The data values which are stored inside an objects are called attributes.
- The functions which are associated with the object are called methods.
- To find out the type of any object use the `type` function:

```
type(any_object)
```

Objects

- Python automatically assigns each object a unique id for identifying the object at runtime.
- An object's state (also known as its properties of attributes) is represented by variables, called data fields.
- Python uses methods to define an object's behaviour.
- The terms object and instance are often used interchangeably.

Defining classes

- The Python uses the following syntax to define a class:

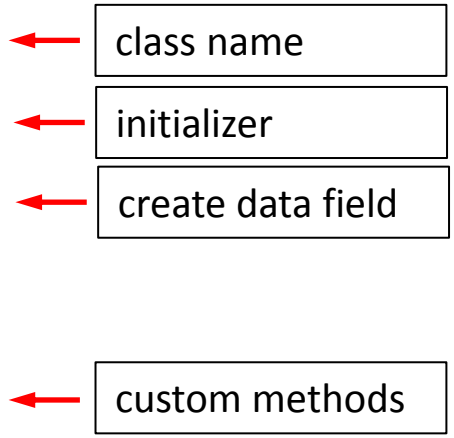
```
class ClassName:
    initializer
    methods
```

- A special __init__ method, known as an initializer, is invoked to initialize a new object's state when it is created.
- Initializers are designed to perform initializing actions, such as creating an object's data field with initial values.
- The init needs to be preceded and followed by two underscores.

Example 1a

- Listing: Circle.py

```
import math

class Circle:

    def __init__(self, radius = 1):
        self.radius = radius

    def perimeter(self):
        return 2 * self.radius * math.pi

    def area(self):
        return 2 * self.radius * math.pi

    def setRadius(self, radius):
        self.radius = radius

    def getRadius(self):
        return self.radius
```



Constructing objects

- All methods, including the initializer, have the first parameter self.
- This parameter refers to the object that invokes the method.
- The self parameter in the __init__ method is automatically set to reference the object that was just created.
- The self is used in the implementation of the method, but it is not used when the method is called.

Accessing members of objects

- Accessing to an object data fields and invoking its methods is possible by using the dot operator also known as the object member access operator .

```
objRefVar = ClassName(arguments)
objRefVar.datafield
objRefVar.method(args)
```

Example 1b

- Creating objects

```
c1 = Circle()
c1.setRadius(3)
print("Radius is",c1.getRadius())
print("Area is", c1.area())

c2 = Circle(5)
print("Area is", c2.area())

print("Area is", Circle(2).area())
```

creating the object and assign it to a variable

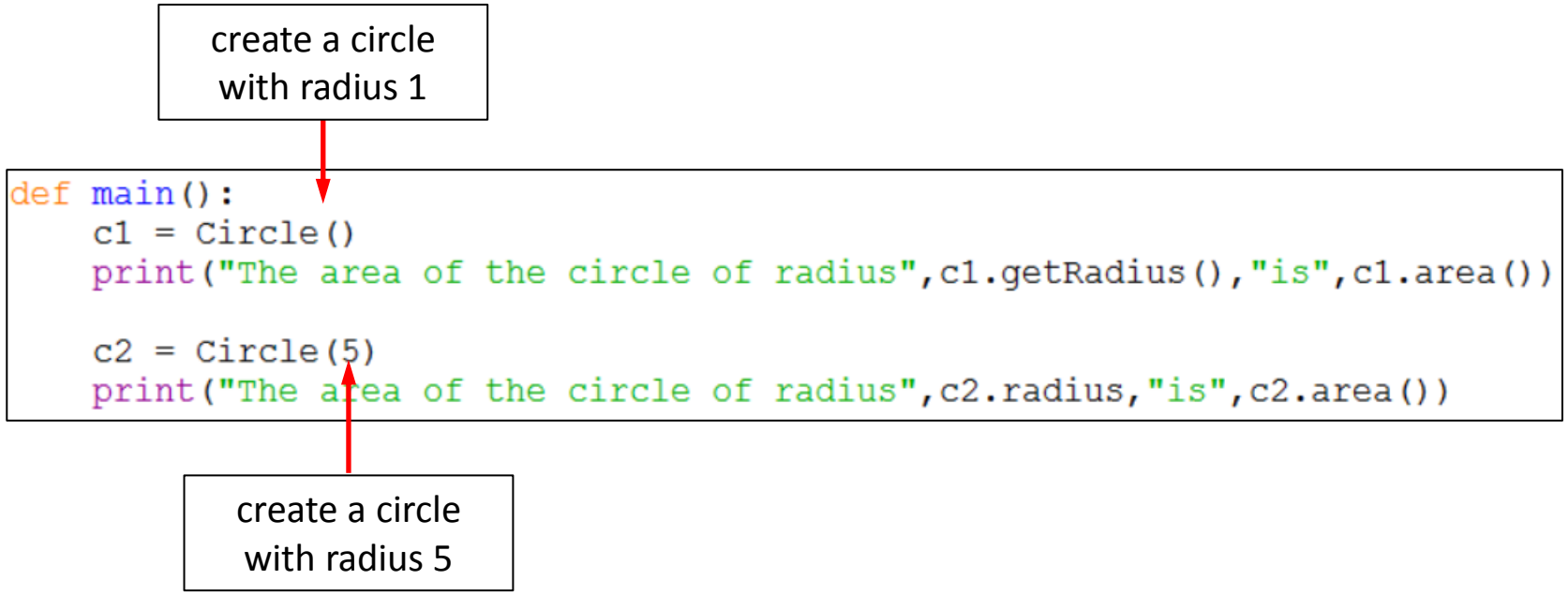
using the variable to reference the object

create data field

occasionally an object does not need to be referenced later

Example 1c

- Creating objects



Problem 1

Defining and using the Person class – a program that has a simple custom class which stores information about a person

- Guidelines:
  - Inside the class body define two object's method:
    - special __init__() method with parameters (name, surname, birthdate, telephone, and email) that are passed to the class object
    - a custom age() method which calculates the age of a person using the birthdate and the current date
  - The birthdate attribute is itself an object. The date class is defined in the datetime module.
  - Create an object of the Person class and display a person name, email and age.

Problem 1 - solution

- Source code

```
import datetime

class Person:
    def __init__(self, name, surname, birthdate, telephone, email):
        self.name = name
        self.surname = surname
        self.birthdate = birthdate
        self.telephone = telephone
        self.email = email

    def age(self):
        today = datetime.date.today()
        age = today.year - self.birthdate.year
        if today < datetime.date(today.year,
                                 self.birthdate.month,
                                 self.birthdate.day):
            age -= 1
        return age
```

Problem 1 - solution

- Source code

```
person = Person(
    "Jan",
    "Kowalski",
    datetime.date(1995,4,23), # year, month, day
    "123 444 0101",
    "jan.kowal@example.com"
    )

print(person.name)
print(person.email)
print(person.age())
```

Mutable class attribute

- Modifying a class attribute of a mutable type in-place – affects all objects of that class at the same time.

```
class Person:
    pets = []

    def add_pet(self, pet):
        self.pets.append(pet)

jane = Person()
bob = Person()

jane.add_pet("cat")
print(jane.pets)
jane.add_pet("dog")
print(bob.pets)
```

```
['cat']
['cat', 'dog']
```

```
class Person:

    def __init__(self):
        self.pets = []

    def add_pet(self, pet):
        self.pets.append(pet)

jane = Person()
bob = Person()

jane.add_pet("cat")
print(jane.pets)
jane.add_pet("dog")
print(bob.pets)
```

```
['cat']
[]
```

Example 2

Vector class (Overloading Operators) – an application that perform vector addition

- Create a Vector class to represent two-dimensional vectors
- Define the __add__ method in a Vector class to perform vector addition
- Define the __str__ method to represent an object as a string

```
class Vector:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def __str__(self):
        return 'Vector (%d, %d)' % (self.a, self.b)

    def __add__(self,other):
        return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(3,7)
v2 = Vector(4,-2)
print(v1+v2)
```

Destroying Objects

__del__ destructor – prints the class name of an instance that is about to be destroyed

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

    def __del__(self):
        class_name = self.__class__.__name__
        print(class_name, "destroyed")

pt1 = Point()
pt2 = pt1
pt3 = pt1
print(id(pt1), id(pt2), id(pt3))
del pt1
del pt2
del pt3
```

```
616990036376 616990036376 616990036376
Point destroyed
```

Properties

Properties – allow to add behaviour to data attributes

- Can replace direct attribute access with a property

```
class MyVector2(object):
    def __init__(self, x, y):
        self._x = x
        self._y = y

    def get_x(self):
        print("Returning x, which is {}".format(self._x))
        return self._x

    def set_x(self, x):
        print("Setting x to {}".format(x))
        self._x = x

    x = property(get_x, set_x)

v1 = MyVector2(1,2)
x = v1.x # uses the getter, which prints the value
v1.x = 4 # uses the setter, printing the value
```

Problem 2

The TV set simulation – develop a program in which TV set is an object with:

- states - which are represented by data fields:

  `channel` - current channel (1 to 120)

  `volumeLevel` - current volume level (1 to 7)

  `on` - power on/off – indicates whether TV is on/off

- behaviours - which are actions in each TV object implements with methods:

  `__init__()` - initializer and instance variables (default channel =1, default volume level=1, TV set is off)

  `turnOn(),turnOff()` - turn on/off TV

  `getChannel(), setChannel(channel)` - get/set a channel

  `getVolume(), setVolume(volumeLevel)` - get/set a volume level

  `channelUp(), channelDown()` - increase/decrease channel

  `volumeUp(), volumeDown()` - increase/decrease volume level

Problem 2

The TV set simulation

- Create two TV objects tv1 and tv2 and invokes the methods on the objects to perform actions for setting channels and volume levels and for increasing channels and volumes.
- The program should display the state of two TV objects.

Problem 2 - solution

- Source code

```
class TV:
    def __init__(self):
        self.channel = 1
        self.volumeLevel = 1
        self.on = False

    def turnOn(self):
        self.on = True

    def turnOff(self):
        self.on = False

    def getChannel(self):
        return self.channel

    def setChannel(self,channel):
        if self.on and 1 <= self.channel <= 120:
            self.channel = channel

    def getVolumeLevel(self):
        return self.volumeLevel

    def setVolumeLevel(self, volumeLevel):
        if self.on and 1 <= self.volumeLevel <= 7:
            self.volumeLevel = volumeLevel
```

Problem 2 - solution

- Source code

```
    def channelUp(self):
        if self.on and self.channel < 120:
            self.channel += 1

    def channelDown(self):
        if self.on and self.channel > 0:
            self.channel -= 1

    def volumeUp(self):
        if self.on and self.volumeLevel < 7:
            self.volumeLevel += 1

    def volumeDown(self):
        if self.on and self.volumeLevel > 1:
            self.volumeLevel -= 1
```

Problem 2 - solution

- Source code

```
def main():
    tv1 = TV()
    tv1.turnOn()
    tv1.setChannel(30)
    tv1.setVolumeLevel(3)

    tv2 = TV()
    tv2.turnOn()
    tv2.channelUp()
    tv2.channelUp()
    tv2.volumeUp()

    print("tv1's channel is", tv1.getChannel(),
          "and volume level is", tv1.getVolumeLevel())
    print("tv12's channel is", tv2.getChannel(),
          "and volume level is", tv2.getVolumeLevel())

main()
```

Problem 3

Compute BMI – a program with the object-oriented approach for computing body mass index

- Define a class with data fields as values of weight, height, a person's name and age.
- Define methods:

  `__init__(self, name, age, weight, height)` - initializer

  `getName()` – returns the name

  `getAge()`

  `getWeight()`

  `getHeight()`

  `getBMI()` – returns the BMI

  `getStatus ()` – returns the BMI status (Underweight, Normal, Overweight, Obese)
- Create a BMI object with the specified name, age (the default is 20), weight and height.

Problem 3 - solution

- Source code

```
class BMI:
    def __init__(self, name, age, weight, height):
        self.__name = name
        self.__age = age
        self.__weight = weight
        self.__height = height

    def getBMI(self):
        return self.__weight/(self.__height**2)

    def getStatus(self):
        bmi = self.getBMI()
        if bmi < 18.5:
            return "Underweight"
        elif bmi < 25:
            return "Normal"
        elif bmi < 30:
            return "Overweight"
        else:
            return "Obese"
```

Problem 3 - solution

- Source code

```
    def getName(self):
        return self.__name

    def getAge(self):
        return self.__age

    def getWeight(self):
        return self.__weight

    def getHeight(self):
        return self.__height

def main():
    bmi1 = BMI("Jan Kowalski", 21, 78, 1.81)
    print("The BMI for", bmi1.getName(), "is",
          format(bmi1.getBMI(),'0.2f'),'-', bmi1.getStatus())

main()
```