Lecture 11

Object Oriented Programming

Burkay Genç, Ahmet Selman Bozkır, and Selma Dilek

24/05/2023

PREVIOUS LECTURE

- file IO
- how to read from a file on disk
- how to write to a file on disk

TODAY

- Object Oriented Programming
  - A different way of thinking about programming
- Classes
- Objects

OBJECTS

OBJECTS

- In Python, everything is an **object**
  - `3` is an object
  - `'3'` is an object
  - `"Three"` is an object
  - `3.141592` is an object
  - `[1,2,3]` is an object
  - `(1,2,3)` is an object
  - `{1:'a', 2:'b', 3:'c'}` is an object
- Each of these objects has a different type

TYPES OF OBJECTS

- Each of these objects has a different type

```
type(3)
```

```
## <class 'int'>
```

```
type('3')
```

```
## <class 'str'>
```

```
type("Three")
```

```
## <class 'str'>
```

```
type(3.141592)
```

```
## <class 'float'>
```

```
type([1,2,3])
```

```
## <class 'list'>
```

```
type((1,2,3))
```

```
## <class 'tuple'>
```

```
type({1:'a', 2:'b', 3:'c'})
```

```
## <class 'dict'>
```

CLASSES

- The type of an object determines its **class**
- Each object has a **class**
- The class declares
  - **attributes** and
  - **methods** of the object

CLASS EXAMPLES

- **Class Human**
  - *Attributes*:
    - Age
    - Gender
    - Height
    - Weight
    - Every **Human** has the above **attributes**
  - *Methods*:
    - Breathe
    - Eat
    - Drink
    - Sleep
    - Every **Human** has the above **methods** (can do that thing)

CLASS EXAMPLES

- **Class Car**
  - *Attributes*:
    - Power
    - Torque
    - Transmission Type
    - Capacity
  - *Methods*:
    - Start
    - Drive
    - Break
    - Hand Break
    - Steer Right
    - Steer Left
    - Stop

WHAT IS A CLASS?

- The class defines the common attributes and methods of an object
- Every object that belongs to that class, has those attributes and methods

INSTANCE

- An object is an **instance** of a class
  - It is created from that class
  - It has all the attributes and methods of that class
- Two objects of a class are **similar but different**
  - They have the **same attributes**, but the **attribute values are different**

INSTANCE EXAMPLE

- **Class Human**
  - *Attributes*:
    - Age
    - Gender
    - Height
    - Weight
  - *Methods*:
    - Breathe
    - Eat
    - Drink
    - Sleep

- *Ahmet* is a **Human**
  - Ahmet is an *object* of the Human **class**
  - Ahmet has Age, Gender, Height and Weight (because every Human has these)
    - Ahmet's Age is 23.
    - Ahmet's Gender is 'Male'.
    - Ahmet's Height is 170 cm.
    - Ahmet's Weight is 65 kg.
  - Ahmet can Breathe, Eat, Drink and Sleep (because every Human can do these)

INSTANCE EXAMPLE

- **Class Human**
  - *Attributes*:
    - Age
    - Gender
    - Height
    - Weight
  - *Methods*:
    - Breathe
    - Eat
    - Drink
    - Sleep

- *Selma* is a **Human**
  - Selma is an *object* of the Human **class**
  - Selma has Age, Gender, Height and Weight (because every Human has these)
    - Selma's Age is 21.
    - Selma's Gender is 'Female'.
    - Selma's Height is 175 cm.
    - Selma's Weight is 58 kg.
  - Selma can Breathe, Eat, Drink and Sleep (because every Human can do these)

INSTANCE

- Both Ahmet and Selma are Human
  - They are **instances** of the *Human class*
  - Because of this, they have the **same attributes and methods**
  - But their **attribute values** are different
- Classes **define attributes and methods**
  - But they **don't declare** the *values of the attributes*
  - Objects **declare** the values of the attributes
- When you **instantiate** an object of a class
  - You are cloning the class atributes and methods
  - You are declaring the **inital values of the attributes** of that intance

CLASSES IN PYTHON

- How do you create a class in Python?

```
class Human:
  def __init__(self, age, gender, height, weight):
    self.Age = age
    self.Gender = gender
    self.Height = height
    self.Weight = weight
```

- The `__init__(self)` function is a special function used to create an object instance from a class
  - It is always written as `__init__` and its first argument is always `self`
  - `self` means the **object's self**

HOW TO USE IT

- Let's instantiate an object from class Human:

```
Ahmet = Human(23, 'Male', 170, 65)
print(Ahmet)
```

```
## <__main__.Human object at 0x000002D5BBC98A10>
```

- Let's access an attribute of `Ahmet`:

```
print(Ahmet.Age)
```

```
## 23
```

```
print(Ahmet.Gender)
```

```
## Male
```

MORE EXAMPLES

- We can now use `Ahmet` as an argument to a function:

```
def isTeenager(human):
  if human.Age < 20:
    return True
  else:
    return False

isTeenager(Ahmet)
```

```
## False
```

- `isTeenager` function only works, if the provided argument has an `Age` attribute
- Otherwise, you will get an error

```
try:
  isTeenager(5)
except Exception as e:
  print("Error: ", e)
```

```
## Error:  'int' object has no attribute 'Age'
```

CLASS METHODS

- To avoid problems like the one in the previous slide, we can make the function **belong** to the class
- This way, we guarantee that we have the right attributes

```
class Human:
  def __init__(self, age, gender, height, weight):
    self.Age = age
    self.Gender = gender
    self.Height = height
    self.Weight = weight

  def isTeenager(self):      # We always send self to object methods
    if self.Age < 20:
      return True
    else:
      return False
```

- Now, instantiate Ahmet and it will have the `isTeenager()` method!

```
Ahmet = Human(23, 'Male', 170, 65)
Ahmet.isTeenager()
```

```
## False
```

EXAMPLE

- Define a **Rectangle** class with the following attributes and methods:
  - Attributes: `width`, `height`
  - Methods: `area()`, `circumference()`

EXAMPLE

- Define a **Rectangle** class with the following attributes and methods:
  - Attributes: `width`, `height`
  - Methods: `area()`, `circumference()`

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

  def area(self):
    return self.width * self.height

  def circumference(self):
    return 2 * (self.width + self.height)
```

```
r1 = Rectangle(3, 4)
r1.area()
```

```
## 12
```

```
r1.circumference()
```

```
## 14
```

EXAMPLE

```
r1 = Rectangle(3, 6)
r2 = Rectangle(4, 5)
r2.area() > r1.area()
```

```
## True
```

```
r3 = Rectangle(r1.width, r2.height)
r3.circumference()
```

```
## 16
```

SPECIAL METHODS

- Some special methods can also be defined within a class:
  - `__str__` is called by Python to print an object
  - It **returns** a string representation of the object

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

  def __str__(self):    # returns the string to print
    return ("<" + str(self.width) + "," + str(self.height) + ">")

r1 = Rectangle(3, 6)
r2 = Rectangle(4, 5)
print(r1)
```

```
## <3,6>
```

```
print(r2)
```

```
## <4,5>
```

SPECIAL METHODS

- There are other special methods you can **implement**:
  - `__add(self, other)__` : adds self and other
  - `__sub(self, other)__` : subtracts other from self
  - `__eq(self, other)__` : returns whether self == other
  - `__lt(self, other)__` : returns whether self < other
  - `__len(self)__` : returns the length of self
  - …and others
- Only implement a special method if it is meaningful
  - No point in implementing `__len__` for `Rectangle` class

EXAMPLE

- Implement `__eq__` for Rectangle class

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

  def __eq__(self, other):
    return self.width == other.width and self.height == other.height

r1 = Rectangle(3, 5)
r2 = Rectangle(4, 6)
r3 = Rectangle(3, 5)
r1 == r2
```

```
## False
```

```
r1 == r3
```

```
## True
```

EXAMPLE

- Implement a class of rational numbers. Your class must support addition, subtraction and multiplication of rational numbers. It should also support simplification. It should support equals check.

EXAMPLE

```
class Rational:
  def __init__(self, a, b):
    self.A = a
    self.B = b

  def __str__(self):
    return str(self.A) + "/" + str(self.B)

r1 = Rational(3, 5)
print(r1)
```

```
## 3/5
```

EXAMPLE

- Let us implement `simplify()` first:
  - The following function is normally placed under the class definition
  - We omit the class definition here due to limited slide space

```
  def simplify(self):
    # Find the GCD
    gcd = 1
    for i in range(2, min(self.A, self.B) + 1):
      if self.A % i == 0 and self.B % i == 0:
        gcd = i

    # Update self
    self.A = self.A // gcd   # Force integer division
    self.B = self.B // gcd   # Force integer division
```

```
r1 = Rational(12, 15)
print(r1)
```

```
## 12/15
```

```
r1.simplify()
print(r1)
```

```
## 4/5
```

EXAMPLE

- We can now update `__init__` to be smarter:

```
  def __init__(self, a, b):
    self.A = a
    self.B = b
    self.simplify()
```

```
r1 = Rational(12, 15)
print(r1)
```

```
## 4/5
```

EXAMPLE

- Time for equals check:

```
  def __eq__(self, other):
    self.simplify()        # To be on the safe side, simplify both rationals
    other.simplify()
    return self.A == other.A and self.B == other.B
```

```
r1 = Rational(12, 15)
r2 = Rational(8, 10)
r3 = Rational(9, 12)
r1 == r2
```

```
## True
```

```
r1 == r3
```

```
## False
```

```
r2 == r3
```

```
## False
```

EXAMPLE

- Time for addition:

```
  def __add__(self, other):
    # Find the LCM of self and other's denominators
    big, small = max(self.B, other.B), min(self.B, other.B)
    i = 1
    while not (big * i) % small == 0:
      i += 1
    lcm = big * i

    return Rational(self.A * (lcm // self.B) + other.A * (lcm // other.B), lcm)
```

```
r1 = Rational(1, 5)
r2 = Rational(3, 4)
r3 = Rational(12, 8)
print(r1 + r2)
```

```
## 19/20
```

```
print(r2 + r3)
```

```
## 9/4
```

```
print(r1 + r3)
```

```
## 17/10
```

EXAMPLE

- We will do multiplication next:

```
  def __mul__(self, other):
    if type(other) == int:
      return Rational(self.A * other, self.B)
    else:
      return Rational(self.A * other.A, self.B * other.B)

  def __rmul__(self, other):
    if type(other) == int:
      return Rational(self.A * other, self.B)
    else:
      return Rational(self.A * other.A, self.B * other.B)
```

```
r1 = Rational(2, 9)
r2 = Rational(3, 4)
print(2 * r1)
```

```
## 4/9
```

```
print(r1 * 2)
```

```
## 4/9
```

```
print(r1 * r2)
```

```
## 1/6
```

EXAMPLE

- Finally subtraction is trivial by function reuse:

```
  def __sub__(self, other):
    return self + (-1 * other)
```

```
r1 = Rational(2, 9)
r2 = Rational(3, 4)
print(r1 - r2)
```

```
## -19/36
```

```
print(r2 - r1)
```

```
## 19/36
```

HOME EXERCISE

- Notice that we have omitted **many** details in the Rational class
  - initializing with **negative denominators**
  - **numerator being 0**
  - denominator being 0 (**division by zero**!)
  - addition and subtraction with integers
  - addition, subtraction and multiplication with **floats**!
  - and possibly more…
- Try to implement/solve/handle as many of these situations as possible