Lecture 23: Object-Oriented Programming

CS 51P

December 4, 2019

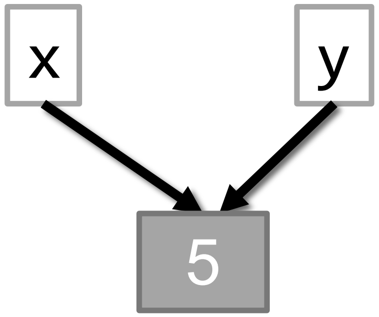
Review: Types in Python

Primitive Types

- int
- float
- bool
- string

```
x = 5
y = 5
```

```
>>> x == y
True
>>> x is y
True
```

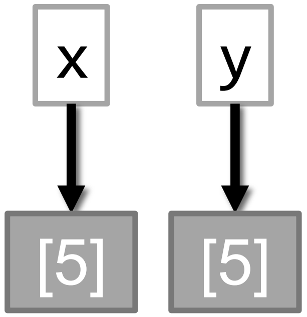


Objects

- tuple
- list
- dictionary
- Create your own…

```
x = [5]
y = [5]
```

```
>>> x == y
True
>>> x is y
False
```



Review: Classes

- Defining a type:
  - how would you describe it? what distinguishes one object of this type from another?
  - what can an object of this type do?

- Example: Classroom type
  - attributes: building, room number, capacity, accessible
  - methods:
    - find out building, room number, capacity
    - change capacity

```
room1 = Classroom("Edmunds", "114", 40)
room2 = Classroom("Edmunds", "101", 30)
print(room2)
print(room2.get_capacity())
room2.set_capacity(50)
print(room2.get_capacity())
```

Review: Classes

```
class Classroom:
    def __init__(self, building, room, capacity):
        self.building = building
        self.room_number = room
        self.capacity = capacity

    def get_building(self):
         return self.building

    def get_room_number(self):
         return self.room_number

    def get_capacity(self):
         return self.capacity

    def set_capacity(self, capacity):
         self.capacity = capacity

    def __str__(self):
        return(self.building + self.room_number +
               ", capacity " + self.capacity)
```

Review: Creating and Using Objects

```
room = Classroom("Edmunds", "114", 40)
print(room)

print(room.get_capacity())
room.set_capacity(50)
print(room.get_capacity())

enough_space([room, Classroom("Edmunds", "101", 30")], 30)
```

default parameters

```
class Classroom:
    def __init__(self, building, room, capacity, accessible=True):
        self.building = building
        self.room_number = room
        self.capacity = capacity
        self.accessible = accessible

mason22 = Classroom("mason", 22, 18, False)
edmunds114 = Classroom("edmunds", 114, 40)
```

- Can use default parameters in functions

```
class Thing:

    def __init__(self):
        self.a = 1
        self.b = 4

    def foo(self, param):
        self.a = self.a + param
        self.b = self.b + param
        return (self.a + self.b)

    def bar(self, param):
        a = self.a + param
        b = self.b + param
        return (a + b)

    def __str__(self):
        return ('a is ' + str(self.a) +
                    ', b is ' + str(self.b))

it = Thing()
print(it.foo(2))
print(it.bar(3))
print(it)
```

Programming as a way of thinking

- Decomposition
  - what does a problem remind you of
  - how can you reduce it to smaller, coherent pieces
- Testing
  - how do you know if something works
- Debugging
  - how to isolate where the problem is
- Communication
  - how to explain what you did

Design

- Say you want to simulate the following:
  - there are a group of people
  - every person has a closet full of clothes
  - they each choose clothes on any given day based on the temperature and their personal cold/hot comfort zone
  - when they all see each other something happens based on what each of them chose

Design

- Say you want to simulate the following:
  - there are 2 people
  - each person has a collection of 4 shirts: red, blue, green, yellow
  - every day for 5 days the two people randomly choose a shirt to wear
  - a special message is displayed on any day when both people wear the same color shirt

Sample run

```
---------- Day 1 ----------
Alice has a blue shirt
Bob has a green shirt
---------- Day 2 ----------
Alice has a red shirt
Bob has a blue shirt
---------- Day 3 ----------
Alice has a yellow shirt
Bob has a red shirt
---------- Day 4 ----------
Alice has a red shirt
Bob has a red shirt
Alice and Bob are wearing the same color shirt!
---------- Day 5 ----------
Alice has a red shirt
Bob has a blue shirt
```

Defining a class:

what attributes does it have?

what can you do with it?

Exercise

```
class Person:
    SHIRT_COLORS = ("red", "green", "blue", "yellow")

    def __init__(self, person_name, shirt_color = "blue"):
         pass

    def get_shirt_color(self):
         pass

    def get_name(self):
         pass

    def change_shirt(self):
         pass

    def __str__(self):
         pass
```

Abstraction

- abstraction is the idea of removing low-level details so you can focus on more important things (like getting your code working)
- fundamental concept in computer science

Exercise

- Assume you have a class Person with methods get_name, get_shirt_color, and change_shirt. Implement a program that will exhibit the following behavior:

```
---------- Day 1 ----------
Alice has a blue shirt
Bob has a green shirt
---------- Day 2 ----------
Alice has a red shirt
Bob has a blue shirt
---------- Day 3 ----------
Alice has a yellow shirt
Bob has a red shirt
---------- Day 4 ----------
Alice has a red shirt
Bob has a red shirt
Alice and Bob are wearing the same color shirt!
---------- Day 5 ----------
Alice has a red shirt
Bob has a blue shirt
```