Lecture 22: Object-Oriented Programming

CS 51P

December 2, 2019

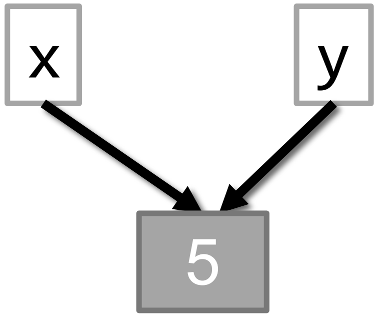
Types in Python

Primitive Types

- int
- float
- bool
- string

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

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

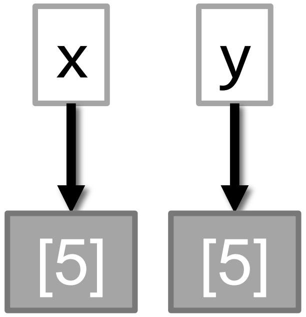


Objects

- tuple
- list
- dictionary
- Create your own…

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

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



class: programmer-defined type

- 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())
```

Class Syntax

```
class Classroom:
    # method definitions go here
```

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

Special methods

special methods have double underscores in name

- __init__
  - constructor
  - called when you create an object

self refers to this instance. all methods have self as the first parameter.

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

self.variable_name refers to instance attributes (i.e., variables)

- __str__
  - called when you print an object

all methods have self as the first parameter even if they have no other parameters

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

Example Class

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

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

Additional Methods

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

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

methods that return the current value in an attribute are called **getter** or **accessor** methods

methods that modify the current value in an attribute are called **setter** or **mutator** methods

Exercise

- What gets printed by the following code?

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

Exercise

Write a function `enough_space` that takes two parameters: `rooms` (a list of `Classrooms`) and `num_people` (`int`). The function should print the classrooms that have capacity greater than or equal to `num_people`.

Write a main function that creates a list of two classrooms and then calls `enough_space` with that list

Exercise

- Modify the class Classroom to add a Boolean instance variable that stores whether the classroom is accessible

```
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
```

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
- Example: what is the default parameter in function input

Exercise

- Define a class Rectangle with attributes width and height and methods __init__, get_width, set_width, get_height, set_height, and area

style

```
class Classroom:
    '''
    Class representing a classroom with a location, a capacity,
    and whether it is accessible
        [ ... as classes get more complex want to specify
          instance attributes, methods ... ]
    '''

    def __init__(self, building, room, cap, accessible=True):
        '''
        Create a new Classroom with given location, capacity, and
           accessibility
        param building (str): building name
        param room (str): room number
        param cap (int): capacity
        param accessible (bool): if room is accessible (default True)
        '''
```