15: Classes

Alexandra Papoutsaki
she/her/hers
Lectures

Zilong Ye
he/him/his
Labs
Lecture 15: Classes

- Objects
- Classes
Objects

- Software bundles that maintain their states/data in variables and implement their behavior with methods.

- For example, a list is an object:

```python
>>> my_list = [1, 2, 3, 4]
>>> my_list.append(5)
>>> my_list
[1, 2, 3, 4, 5]
>>> my_list.reverse()
>>> my_list
[5, 4, 3, 2, 1]
```
Method types

- **Mutators**: change or mutate the state/data of the object.

- **Accessors**: do not change the state/data of the object, only ask questions about it.

You can look at all of the methods available for an object using `help`. e.g.,

```python
>>> help(my_list)
```

or by passing the type of the object:

```python
>>> help(list)
```
Constructors

- Every object has a special method called constructor.
- A constructor has the same name as the type of the object and can be called on its own to "construct" a new object.
- Many of the methods we've actually been using already are actually constructors which construct a new object of that type:

```python
>>> str(10)
'10'
>>> int("10")
10
>>> float(10)
10.0
```
Constructors

Some objects, like lists and tuples, have special ways of constructing them, but they, too, have constructors.

```python
>>> list()
[]
>>> tuple()
()
>>> list("abcde")
['a', 'b', 'c', 'd', 'e']
>>> tuple("abcde")
('a', 'b', 'c', 'd', 'e')
>>> tuple([1, 2, 3, 4])
(1, 2, 3, 4)
>>> list((1, 2, 3, 4))
[1, 2, 3, 4]
```
Lecture 15: Classes

- Administrative
- Objects
- Classes
Classes

- A class is the blueprint describing what data and methods an object will have.
- An object is an *instance* of a class.
- For example, we could define a class Person.
  - A person has certain attributes, like name, age, place of birth, etc.
  - A person has certain methods, e.g., says their name, moves residence.
  - When we define a particular person, it is an object, that is an instance of the class Person.
- Classes define types. In Python, since all things are objects, then they all represent instances of objects. Though in other languages, you could have a type that is not defined by a class.
Defining our own classes

- Syntax:
  
  ```python
  class NameOfClass:
      # methods in the class
  ```

- By convention, class names should start with a capital letter. If they have multiple words, capitalize each word but do NOT use underscores to separate.

- this is called “camel case”. 
Look at Person class in person.py

- 5 methods
- 2 “special” methods
  - methods that are surrounded by two underscores on each side are "special" methods. They are generally NOT called directly.
    - __init__ defines the constructor for the method.
    - __str__ defines what the object will be when it is used in a string context, e.g., when printed.
**self**

- `self` is a variable that allows us to store data and retrieve data in an object.
- `self` is a reference to the current object.
- It should be the first parameter to every method in a class (not totally true, but fine for this class)
- Though you do NOT include it when you call the methods (this is a little annoying, but you'll get used to it). This is also how we indicate that a function is a method
- We can access the "data" associated with a class by using `self`,
  - e.g. `self.x = 10`
  - Creates a new instance variable (variable associated with this object) called `x` and assigns it the value 10.
__init__ method in Person class

- Takes self and two parameters.
- All it does is save these parameters into an object.
- self.name creates a new instance variable called name and stores persons_name into it.
Other methods in Person class

- **get\_X methods:**
  - these are accessor methods, they give us information back.
  - Notice that they only take self as a parameter and use that to give back characteristics about the object.

- **\_str\_**
  - it doesn't take any parameters (except self) and must return a string.
Using classes

- Look into the main function:
  - It creates two instances of the Person class and stores them in separate variables.
  - Person(...) is a call to the constructor (__init__ method)
  - It goes through a week (days 1 to 7) and each day “prints out” the two people:
    - print(p1) will call the __str__ method on p1.
  - Uses the accessor methods (get_X) to ask questions about the two Person objects.
Practice Time

- Write a class called Rectangle:
  - Four instance variables: x1, y1, x2, y2.
  - Write the constructor `__init__`.
  - Write a method `area` that returns the area of the rectangle.
- Look into `rectangle.py`
Why classes?

- Encapsulation!
  - Look at the Rectangle class in rectangle2.py code.
    - We have the same constructor and area method, however, we have different internal representation.
      - We store the bottom left corner, width and height.
      - Notice that we can ask the exact same question about its area using this representation.
    - Anyone using the Rectangle class should NOT care which implementation we use
  - Both have the same set of methods and the same functionality.
  - This is the power of using classes, a general framework called object-oriented programming.
Why classes?

- Modularity
  - functions create single units that we can use to build up other functions.
  - In the same way, classes allow us to create functional units (in this cases a class of objects with a particular behavior).

- Data sharing
  - classes allow us to share data between methods/functions without having to have them as explicitly parameters.
  - this can be very useful (we'll see an example of this for the next assignment).

- Avoid naming conflicts
Resources

- Textbook: Chapter 17 and Chapter 18
- person.py
- rectangle.py
- rectangle2.py

Homework

- Assignment 7 (cont'd)