INFO5002: Intro to Python for Info Sys

Classes

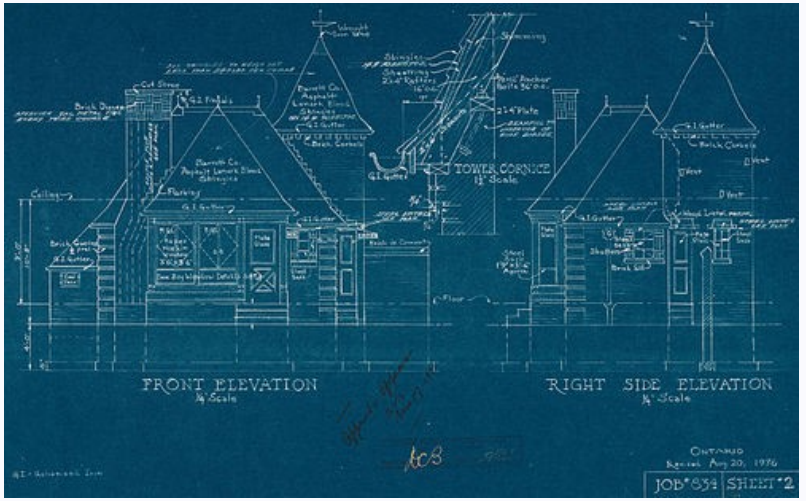
PCC 157-181

Northeastern University

Slides created by: Zachary Doucet

Classes to group functionality

- Functions group operations together and collections group data together.
- Classes allow you to collect functions and variables together.



Source: City of Toronto Archives

Creating classes

- Create a class with the class keyword.

```
class Car:
    ...
```

Creating objects

- Objects are created by “calling” a class.

```
car = Car()
```

Classes vs Objects

- Classes are like function declarations and objects are like function calls.
- Objects are instantiated classes.
- Objects have state which is the current value of all variables held by the object.

To avoid confusion, name properly!

- Variables and functions follow snake_case.
- For **classes** use UpperCamelCase.

```
class LightSaber:
    pass
```

```
class BuildingMaterial:
    pass
```

```
class PlanetaryVehicle:
    pass
```

Give your classes statefulness

- You can add attributes to your classes.

```
class myClass:
    x = 1
```

- You can turn your dictionaries (with finite keys) into classes.

```
x = {"model": "Kia Rio",
  "year": 2003, "mpg": 25.32}
```



```
class Car:
    model = "Kia Rio"
    year = 2003
    mpg = 25.32
x = Car()
```

Accessing and Mutating

```
x["model"]
```



```
x.model
```

```
x["model"] = "Kia Soul"
```



```
x.model = "Kia Soul"
```

Have different state with constructors

- Constructors allow passing in of data at object creation.
- This can allow for changing (and avoiding) default state.

```
class Car:
    model = "Kia Rio"
    year = 2003
    mpg = 25.32

    def __init__(self, model,
      year, mpg):
        self.model = model
        self.year = year
        self.mpg = mpg
x = Car("Kia Rio", 2003, 25.32)
```

```
class Car:
    def __init__(self, model
      year, mpg):
        self.model = model
        self.year = year
        self.mpg = mpg

x = Car("Kia Rio", 2003, 25.32)
```

No Defaults

We can still have defaults!

- Simply set constructor's arguments to optional.

```
class Car:
    def __init__(self,
      model = "Kia Rio",
      year = 2003,
      mpg = 25.32):
        self.model = model
        self.year = year
        self.mpg = mpg

...
```

```
...

a = Car()
b = Car(model="Kia Soul")
c = Car(year=1995)
d = Car(mpg=32.16)
e = Car(model="Kia Sportage",
        year=2025)
```

What is the state of objects a, b, c, d, and e after creation?

Every class has a constructor

- If you do not provide a constructor a default one is provided.

```
def __init__(self):
    pass
```

- Even without constructor you can still create objects which will call the default constructor.

```
class MyClass:
    pass
my_class = MyClass()
```

Printing looks weird

```
class Car:
    def __init__(self, model):
        self.model = model
x = Car("Honda")
print(x)
```

```
<__main__.Car object at 0x7fb8e5094ec0>
```

Create a **__str__** method.

```
class Car:
    def __init__(self, model):
        self.model = model
    def __str__(self):
        return f"Vehicle model:
            {self.model}"
x = Car("Honda")
print(x)
```

```
Vehicle model: Honda
```

Create methods to bind functionality

- Methods are functions written in a class.

```
class Car:
    def __init__(self, model):
        self.model = model

    def my_model(self):
        return self.model

    def car_sound():
        print("Vrooom")
```

```
x = Car("Ford")
print(x.my_model())
Car.car_sound()
print(Car.my_model(x))
```

When calling a method on an object, it passes itself as first argument.

self

- Variable that references the current instance of the class (the object) to get associated variables.
- Not necessary *except* for __init__. Those without are called **static**.
- Can be called anything but must always be the first argument.

Working with objects

- Mutate attribute by assignment.

```
x.brand = "Dodge"
```

- Delete attribute with del.

```
del x.brand
```

- Delete object with del.

```
del x
```

Let’s practice

- Create a class **Stats** which has a **constructor** that takes in a list of numbers. Create the following methods:
  - **average**: which returns the average of the list. Use **sum** and **len**.
  - **_median**: which returns the median of the list. Use **median** function that takes in a list of numbers.
  - **_min:** which returns the minimum of the list. Use **min** function.
  - **_max:** which returns the maximum of the list. Use **max** function.

Let’s practice (Continued)

- Create a class **Timer** with the following methods:
  - **start**: which starts the timer.
  - **stop**: which ends the timer.
  - **elapsed:** which returns the elapsed time.
  - **reset:** which resets the timer.
- You can get the current time with **time.time().**