ISTA 350

•••

Lab 1: Classes and RegEx
Spring 2020

Some things you should know...

- Pandas!
    - Series
    - DataFrames
        - Traversing DataFrames

Classes

Defining Classes

Classes allow us to construct our own objects. This gives us the ability to personalize and model our own data structures (objects or instances) into a logical grouping (logical connections between data and functions/methods).

To define a class, you need the **`class keyword`**, a **`name`** for the class, and a **colon** at the end.

```
class Car:
```

Note: Class names are typically CamelCased

The `init` method

After defining a class, you need to define the initializer, **`__init__`**, aka the constructor. This will be the first method you implement after the first class you create. It is automatically called by python each time you instantiate a new object.

```
class Car:
    def __init__(self, name, make, year, mileage,
                color = None):
```

Each car object will have a name, manufacturer, year, mileage, and color (if known).

Instance Variables

In the initializer we can define and initialize instance variables. Instance variables are variables that are defined in a class. They will always have the prefix of `self`.

```
class Car:
    def __init__(self, name, make, year, mileage,
             color = None):
        self.name = name
        self.price = input("How much did it cost? ")
        self.make = make
        self.yr = year
        self.miles = mileage
        if color == None:
             self.set_color()
```

Other Methods

When you create other methods, you make them the same way as `__init__` , except they don’t start or end with underscores and you can name them whatever you want. This is because `__init__` is a magic method (`__repr__` is another example of this).

Other Methods

```
Class Car:
    ...
    def set_color(self):
        while True:
             info = input("Do you know the EXACT color (Y/N)? ")
        if info.upper() == "Y":
                 self.color = input("What color is your car? ")
                 break
         elif info.upper() == "N":
                 print("Too bad :(")
                 self.color = None
                 break
         else:
                 print("I don't understand try again.")
    def get_mileage(self):
                 return self.miles
```

The `repr` method

The `__repr__` method (short for representation) is a method that tells python how to represent your class as a string.

The `repr` method only has self as a parameter. Calling the print function on a class object will implicitly call repr.

`repr` should always return a string.

The `repr` method

```
def __repr__(self):
        result = "\n"
        result += "NAME:".ljust(7) + self.name + "\n"
        result += "MAKE:".ljust(7) + self.make + "\n"
        result += "YEAR:".ljust(7) + str(self.yr) + "\n"
        result += "MILES:".ljust(7) + str(self.miles) + "\n"
        if self.color != None:
          result += "COLOR:".ljust(7) + str(self.color) + "\n"
        return result
```

Making Instances and Invoking Methods

Methods are invoked using the dot operator. When the method is invoked, object.method(), the argument corresponding to self is the object.

Instantiation!

```
def main():
    mustang = Car("Mustang GT", "Ford", 2014, 41000)
    print(mustang) # Same as print(mustang.__repr__())
    if mustang.color == None:
        mustang.set_color()
    print(mustang)
if __name__ == "__main__":
    main()
```

The `self` variable

Every method in a class takes a parameter called `self` which is an instance of the class. But when you invoke a method on an instance of a class, you don't pass self in as an argument.

Instead, self represents the specific object you are calling the method on.

The `self` variable

If you call the `get_mileage` method on mustang you don't pass in any parameters, even though the get_mileage method has one parameter which is self.

```
mustang.get_mileage()
```

The object we called `get_mileage` on, mustang, is assigned to the self variable. Self is the instance of the car that get_mileage is being called on. So `mustang.get_mileage()` is technically `Car.get_mileage(mustang).`

RegEx

What are Regular Expressions?

A regular expression is a pattern describing a certain amount of text. The name is usually abbreviated to “regex” or “regexp”.

It’s usually used for ‘find’ or ‘find and replace’ operations on strings, or input validation in files.

We specify a pattern to search for by creating a string, but some of the characters have special meanings that allow them to match multiple other characters.

The highlights

- `.` – matches any character except a newline
- `\d` – matches any digit
- `\s` – matches any whitespace character
- `[...]` – matches any character between the brackets
  - `[az9.]` matches any `'a'`, `'z'`, `'9'`, or `'.'` and doesn't take on the special meaning of `'.'`
  - `r'd[iou]g'` matches 'dig', 'dog', or 'dug'.
- `[^...]` – matches any character EXCEPT the characters within the brackets; ^ inside of a character set means 'complement', but has a different meaning outside of the character set.

Negative look-ahead assertions

Negative lookahead is indispensable if you want to match something not followed by something else.

Example: say we want to match a `q` not followed by a `u`

The appropriate regular expression would be: `q(?!u)`

The negative lookahead construct is the pair of parentheses, with the opening parenthesis followed by a question mark and an exclamation point. Inside the lookahead, we have the trivial regex `u`.

Complete lab1.py

Only spend a maximum of two hours on it! As long as it’s clear that you gave it a good shot, you will get credit for it at the beginning of lab next week.

Hopefully it won’t be too challenging, the goal of this lab is to give you some review that will help with the homework.

Homework 1 is due Thursday, 1/30