Introduction to Data Science – Week 10

NETA LIVNEH

COURSE 55793

2019-2020

Last Week Recap

Regular Expressions

Reading from an archive file

Midterm takeaways

This Week

Classes and basic Object Oriented Programing (OOP)

List comprehension

Map, Filter and Lambda

Object-Oriented-Programming (OOP)

We mentioned that ***almost everything is an object in python***

A **Class** is like an object constructor, or a "blueprint" for creating objects

An object is an instance of a class, equipped with the general behavior that was defined by the class

Classes really just apply and extend the ideas we've already covered (functions, modules, etc.). roughly, they are packages of functions that use and process built-in object types.

How To Define a Class in Python

All classes create objects, and all objects contain characteristics called attributes.

```
class Cat(object):
    """A Cat class. Cats have the following properties:

    Attributes:
        name: A string representing the cat's name.
        age: A float representing the cat's age
    """
    def __init__(self, name, age=0):
        """Return a Cat object whose name is *name*, and age is *age*,
        default to 0.
        This method initialize the instance attributes
        """
        self.name = name
        self.age = age
```

Just because we've *defined* an object Cat doesn't mean we've *created* one.

Creating an Object

All classes have a built-in function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function (method) to assign values to object properties (attributes), or other operations that are necessary to do when the object is being created.

```
gizmo = Cat('Gizmo', 7.5)
cat = Cat('mitzi')
another_cat = Cat('mitzi')


type(gizmo)

__main__.Cat

cat == another_cat
False
```


Object Methods

A function defined in a class is called a "method".

Methods have access to all the data contained on the instance of the object; they can access and modify anything previously set on self.

Because they use self, they require an instance of the class in order to be used. For this reason, they're often referred to as "instance methods".

```
    # instance method
    def speak(self, sound):
        return "{} says {}".format(self.name, sound)
```

```
In [4]: gizmo.speak('Maow')
Out[4]: 'Gizmo says Maow'
```


The self Parameter

The *self* parameter is a reference to the instance itself, and is used to access variables that belongs to the class

It does not have to be named self , but it has to be the first parameter of any function in the class

When defining a method/function in the class, the first parameter should always be self, to reference the instance

- Self can be call whatever you like though the always named self…
- When calling your own class methods from within the class, you must include the ‘self’ argument
- When calling class methods from the outside, you should *not* specify it
- When calling methods or accessing attributes from within the class, self.attribute or self.method should be supplied

Inheritance

You don’t always have to start from scratch when writing a class. If the class you’re writing is a specialized version of another class you wrote, you can use inheritance

When one class inherits from another, it automatically takes on all the attributes and methods of the first class

The original class is called the parent class, and the new class is the child class..

The child class is free to define new attributes and methods of its own

Example

```
class Snake(Pet):

    """A Snake class which is a subclass of Pet but has
    some snake attributes and methods"""

    legs = 2  # class attribute

    def __init__(self, name, age):
        # inheretance of the super class attributes
        super().__init__(name, age)

    def do_tricks(self):
        return(f'{self.name} scared my friend')
```


List Comprehension

List comprehension is an elegant way to define and create lists in Python.

Essentially, it is Python's way of implementing a well-known notation for sets as used by mathematicians. In mathematics the square numbers of the natural numbers are for example created by $\{ x^2 \mid x \in \mathbb{N} \}$.

```
[expression(x) for x in seq if condition]
```

```
In [112]: [x**2 for x in range(10)]
Out[112]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
In [113]: [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
Out[113]: [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

and it's equivalent to:

```
In [114]: combs = []
In [115]: for x in [1,2,3]:
     ...:     for y in [3,1,4]:
     ...:         if x!=y:
     ...:             combs.append((x, y)
```


Map, Filter and Lambda

map

Calls function (f) for each element in the list or tuple

Returns a list - function value of each element

`map(function, sequence)`

```
In [98]: def cube(x): return x*x*x

In [99]: list(map(cube, range(10)))
Out[99]: [0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Lambda

Creates small anonymous functions that calculates an arithmetic expression.

`lambda argument_list: expression`

```
In [100]: cube = lambda x: x**3
```

filter

Calls function (f) for each element in the list

Returns a list of elements for which f returned true

`filter(function, sequence)`

```
In [101]: def fun(x): return (x%3 == 0) or (x%5 == 0)

In [102]: tuple(filter(fun, range(5, 25)))
Out[102]: (5, 6, 9, 10, 12, 15, 18, 20, 21, 24)
```


Practice List Comprehension

1. Write a list comprehension that would return the length of each element in a sequence.
2. Write the same task as a lambda function.
3. A Pythagorean triple consists of three positive integers a, b, and c, such that $a^2 + b^2 = c^2$. Such a triple is commonly written (a, b, c), and the best known example is (3, 4, 5). Write a list comprehension that creates a Pythagorean triples.