day_2_lecture

January 8, 2020

1 Day 2 Python (Advanced)

1.1 Dictionaries

Dictionaries can be seen as a list with a key. Each item has a key associated to it. The key within a dictionary is unique, each key can only exist once in each dictionary (`int float str`). The key has to be of some comparabe type, while the value an be any kind of object. The values don't even have to be the same, similar to list. We use the curly brackets to define a dictionary, with a key specifying its values with a colon

```
In [72]: x = {"color": "green",
              "size": "medium",
              False : 1,
              0:2}
         x

Out[72]: {False: 2, 'color': 'green', 'size': 'medium'}
```

We can acces the values of the dictionary with the according keys. Just like in lists.

```
In [73]: x['color']

Out[73]: 'green'

In [74]: x.keys()

Out[74]: dict_keys([False, 'size', 'color'])

In [75]: x.values()

Out[75]: dict_values([2, 'medium', 'green'])
```

We can use dictionaries for classification tasks. The are better suited than lists, because it doesn't matter where the value is stored.

```
In [76]: def classify(x):
             if x['color'] == 'green':
                 if x['size'] == 'big':
                     decision = 'watermelon'
```

```
            elif x['size'] == 'medium':
                decision = 'apple'
            else:
                decision = 'other'
        else:
            decision = 'other'
        return decision
```

```
In [77]: x_new = {'color': 'green', 'size': 'big'}
         classify(x_new)
```

```
Out[77]: 'watermelon'
```

```
In [78]: classify({'color': 'green', 'size': 'medium'})
```

```
Out[78]: 'apple'
```

```
In [79]: classify({'color': 'red', 'size': 'small'})
```

```
Out[79]: 'other'
```

1.2 Sets

Sets are really just lists with no repeating elements and no order. Therefore we can not use indexing

```
In [96]: x = {6,5,7,7,3, "hello", True}
         type(x)
```

```
Out[96]: set
```

```
In [97]: x
```

```
Out[97]: {3, 5, 6, 7, True, 'hello'}
```

```
In [98]: x[0] # no indexig
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-98-69d3f2be9341> in <module>
----> 1 x[0] # no indexig


TypeError: 'set' object does not support indexing
```

But we can still loop over it! Order is not defined though

```
In [99]: for y in x:
             print(y)
```

```
True
3
5
6
7
hello
```

2 Other useful built-in functions

Some other useful built-in functions that can make your life easier.

```
In [100]: x = [1,2,3,4]
          y = [2,3,4,5]
```

With `zip` we can combine two iterables of the same length together, and get an iterator of tuples containing each the elements of all lists

```
In [101]: for a,b in zip(x,y):
              print(str(a)+","+str(b))
```

```
1,2
2,3
3,4
4,5
```

```
In [102]: z = [8,9,10,11]

          # we can define as many as we want
          for a,b,c in zip(x,y,z):
              print(str(a)+","+str(b)+","+str(c))
```

```
1,2,8
2,3,9
3,4,10
4,5,11
```

Similar to `zip`, `enumerate` returns an iterator of tuples, but the first value is the index.

```
In [103]: for i,a in enumerate(z):
              print(str(i)+","+str(a))
```

```
0,8
1,9
2,10
3,11
```

`min()`, `max()`, `sum()` are also availabe!

```
In [104]: max(x)

Out[104]: 4

In [105]: min(y)

Out[105]: 2

In [106]: sum(z)

Out[106]: 38
```

`min()`, `max()` can also be used on strings (alphabetical order)

```
In [107]: h = ["string","stg","shshs"]
          max(h)

Out[107]: 'string'
```

We can also create most common container classes from built-in functions.

```
In [108]: x = list("fsfsfs")
          x

Out[108]: ['f', 's', 'f', 's', 'f', 's']

In [109]: y = dict([("color", 'green'), ("size", '7'),("brand", "nike")])
          y

Out[109]: {'brand': 'nike', 'color': 'green', 'size': '7'}
```

2.1 Iterators vs Iterables

Iterators are objects that contain a coutable number of values. An iterator can be iterated upon, meaning that you can traverse through all the values.

Classes like `list`, `dict`, `tuple` and `set` are iterable objects. They are *iterable* containers, from which you can get an iterator from with `iter(iterable)`.

The for loop actually calls upon the iterator of an iterator, to allow to iterate over all objects with in the list.

Each iterator has to implement the two `__iter__()` and `__next__()` (Iterator Protocol)

```
In [110]: x = ['brand','green', 'size']
          y = iter(x) # get the iterator of list x
```

```
In [ ]:

In [111]: y.__next__()

Out[111]: 'brand'

In [112]: type(y)

Out[112]: list_iterator

In [113]: setc = set(x)

In [114]: setc

Out[114]: {'brand', 'green', 'size'}
```

`range`, `enumerate`, `zip` are all iterator objects, while the in-built function of the same name give us back the iterator object. We can call the built-in `next` function.

```
In [115]: y = enumerate(x)

          next(y),next(y),next(y)

Out[115]: ((0, 'brand'), (1, 'green'), (2, 'size'))
```

Since the iterator is empty now, the `next` call will throw a error.

```
In [116]: next(y)


        ---------------------------------------------------------------------------

        StopIteration                             Traceback (most recent call last)

        <ipython-input-116-81b9d2f0f16a> in <module>
    ----> 1 next(y)


        StopIteration:
```

2.2 List Comprehension

List comprehension in python give concise and compact ways to create lists. Altogether there are 3 parts to list comprehension:

- Iteration
- Filter
- Transform

In the above example, we iterate over all elements x in data. We then filter all elements which do not fullfil the given condition, here it is the color green. If the condition is fullfilled, the item will be changed or transformed. We don't have to ,but always need iteration!

Only iteration, No filtering and no transformation

```
In [117]: [x for x in range(5)]

Out[117]: [0, 1, 2, 3, 4]
```

Transformation by adding 2 and iteration, no filtering

```
In [118]: [x + 2 for x in range(5)]

Out[118]: [2, 3, 4, 5, 6]
```

Transformation if condition is met, else use other transformation

```
In [119]: [x + 2 if x > 2 else x - 1 for x in range(5)]

Out[119]: [-1, 0, 1, 5, 6]
```

Until now, we have dont no filtering. The list size is always the same of the initial list. With the if statement at the end, we can filter elements out, which do not fullfil the condition.

Only consider numbers greater than 3, and add 2.

```
In [120]: [x + 2 for x in range(5) if x > 3]

Out[120]: [6]
```

All together:

```
In [121]: values = list(range(10))
          values

Out[121]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [122]: values.append("hello")
          values.append("world")

          # filter out all elements only of type int, and add 2 to x if x is greater than
          # five
          [x + 2 if x > 5 else x-5 for x in values if type(x) is int]

Out[122]: [-5, -4, -3, -2, -1, 0, 8, 9, 10, 11]
```

We can also make multiple predictions for muliple obsercation by using list comprehension!

```
In [123]: data = [
          {'color': 'green', 'size': 'big'},
          {'color': 'yellow', 'shape': 'round', 'size': 'big'},
          {'color': 'red', 'size': 'medium'},
          {'color': 'green', 'size': 'big'},
          {'color': 'red', 'size': 'small', 'taste': 'sour'},
          {'color': 'green', 'size': 'small'}
          ]
          type(data), type(data[0])

Out[123]: (list, dict)
```

We can create an iterator which goes over every element in an dictionary as well, similar to lists.

```
In [124]: results = list()
          for x in data:
              results.append(classify(x))
          print(results)

['watermelon', 'other', 'other', 'watermelon', 'other', 'other']
```

The above code can be even written shorter with python list comprehension.

```
In [125]: print([classify(x) for x in data])

['watermelon', 'other', 'other', 'watermelon', 'other', 'other']
```

We can also combine this with conditions:

```
In [126]: print([classify(x) for x in data if x['color'] == 'green'])

['watermelon', 'watermelon', 'other']
```

```
In [127]: result = [classify(x) for x in data]
```

If we want to count the number of watermelons in the data set, we would usually iterate over the data set, check for each element if it is a watermelon, and increase a counter variable if that is the case.

```
In [128]: count = 0
          for r in result:
              if r == 'watermelon':
                  count = count + 1
          print(count)

2
```

But now, we can also do this in a python way:

```
In [129]: sum([x=='watermelon' for x in result])

Out[129]: 2
```

this works too:

```
In [130]: len([x for x in result if x=='watermelon'])

Out[130]: 2
```

Whats the difference between the two list comprehensions? First transforms x into booleans, answering the question if they are watermelons. The list therefore only contain True and False. Then, the sum function counts all True values, because the booleans can be converted into 1s and 0s. The second method filters out all values which are not x and then gets the length of the list. As we can see, there are multiple ways of getting the right answer.

2.3 Random

We can use the module `random` to create pseudo-random numbers. We need to import the module with import.

```
In [4]: import random
```

Some useful functions are:

```
In [5]: x = random.randint(0,10) # create a random number between 0 and 10
        x

Out[5]: 8

In [8]: y = random.random() # get a random float between 0 and 1
        y

Out[8]: 0.1999057241575689

In [9]: x = list("hello world")
        random.choice(x) # select a random element from a iterable

Out[9]: 'l'
```

More can be found in the docs: https://docs.python.org/3/library/random.html

2.4 Reading Data from a File

Usually, when we have a lot of data, we need to read it from some file. For that, python gives us two statements, which will let us manipulate files. First one is open, which opens a file and returns a file object. The second argument tells us what kind of mode we want to enter:

"r" - Read - Default value. Opens a file for reading, error if the file does not exist

"a" - Append - Opens a file for appending, creates the file if it does not exist

"w" - Write - Opens a file for writing, creates the file if it does not exist

"x" - Create - Creates the specified file, returns an error if the file exist

The next one is the with statements, which handles the setup and closing of the file object. More under https://docs.python.org/2.5/whatsnew/pep-343.html. Together, we can access file values.

```
In [10]: with open('test_results.txt', 'r') as f:
             D = list()
             for line in f:
                 D.extend([float(x) for x in str.split(line[:-1], ',')])
         print(D)
```

```
[80.0, 55.0, 16.0, 26.0, 37.0, 62.0, 49.0, 13.0, 28.0, 56.0, 43.0, 45.0, 47.0, 63.0, 43.0, 65.0,
```

We can just as easy read *csv.* files (Comma Separated Values). They are just text files seperated, or delimited by a comma. CSV is a simple file format used to store tabular data, such as a spreadsheet or database. Files in the CSV format can be imported to and exported from programs that store data in tables, such as Microsoft Excel or OpenOffice Calc.

```
In [11]: with open('csv_example.csv', 'r') as f:
             D = list()
             for line in f: # split until -1 because we dont want charater return (\r)!
                 D.append([float(x) if x.isnumeric() else x for x in str.split(line[:-1], ',')])
         print(D)
```

```
[['Sally Whittaker', 2018.0, 'McCarren House', 312.0, '3.75'], ['Belinda Jameson', 2017.0, 'Cush
```

There are also other ways to read certain type of data files, but more to that later!

2.5 Classes

Python is an object oriented programming language, which means we can write classes to create our own objects or complex data types. A class can be seen as a blueprint for objects. The classes define all the functions and internal variables an object should have. Classes help us to bundle data and functionanility together, and helps to write better observable code.

```
In [12]: class MyFirstClass:

             def hello(self):
                 print("Hello World")
```

```
        x = MyFirstClass()
        x.hello()
```

```
Hello World
```

```
In [13]: a = "hello"
         x = -1
         if type("a" is a) is str:
             x = 0
         elif 2 in a:
             x = 1
         elif 6**(bool(0)):
             x = 2
         print(x)
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-13-56f4b310947a> in <module>
          3 if type("a" is a) is str:
          4     x = 0
    ----> 5 elif 2 in a:
          6     x = 1
          7 elif 6**(bool(0)):


        TypeError: 'in <string>' requires string as left operand, not int
```

```
In [14]: class Car:
             # constructor
             def __init__(self, color, brand, velocity,isOld=True):
                 self.color = color
                 self.brand = brand
                 self.velocity = velocity
                 self.__isOld = isOld #this is a private attribute

             def left_turn(self):
                 print("skkirrt")

             def right_turn(self):
                 self.__alarm()
                 print("brum")

             def distance(self,t, speed):
```

```
                return t*speed;

            def __alarm(self):
                print("alarm")
```

```
In [15]: car = Car()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-15-bbefef4a906c> in <module>
----> 1 car = Car()


TypeError: __init__() missing 3 required positional arguments: 'color', 'brand', and 've
```

```
In [16]: car = Car("green","mercedes",30,False)
```

```
In [17]: car.__isOld # since this is a private attribute, we can not access it
         # from outside the class
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)

<ipython-input-17-d493d87f2ac3> in <module>
----> 1 car.__isOld # since this is a private attribute, we can not access it
      2 # from outside the class


AttributeError: 'Car' object has no attribute '__isOld'
```

```
In [18]: car.right_turn()
```

```
alarm
brum
```

```
In [19]: car.__alarm() # we can not access this
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
```

```
        <ipython-input-19-2ed05704d43e> in <module>
    ----> 1 car.__alarm() # we can not access this


        AttributeError: 'Car' object has no attribute '__alarm'
```

In [ ]: