Discussion 4

DSC 20, Spring 2024

Meme of the week

Agenda

- Practice Questions
- **Content**
    - lambda
    - iterators
    - map
    - filter

Content

Lambda Functions

- known as anonymous functions (their functions are so simple, they don't need a name)
- syntax: lambda (input): (some operation)
- within the scope of this course, lambda is used in conjunction with map and filter

```
In [4]: def add_2(x):
            return x+2
        add_2(1)

Out[4]: 3

In [5]: func = lambda x: x+2
        func(1)

Out[5]: 3
```

Checkpoint

Are the following 2 functions equivalent?

```
In [6]: def strip_caps(string):
            output = ''
            for char in string:
                if not char.isupper():
                    output+=char
            return output
```

```
In [7]: lambda_strip = lambda x: x if x.isupper() else ''
```

A. Yes

B. No

Checkpoint Solution

nope!

```
In [11]: example = 'MARINAlanglois'

In [12]: strip_caps(example)

Out[12]: 'langlois'

In [13]: ''.join(list(map(lambda_strip, example)))

Out[13]: 'MARINA'
```

Iterator

Iterator - **Syntax: iter(iterable)**, **next(iterator)**

- An iterator in Python is an object that can be iterated upon, meaning that you can traverse through all the values.
- Typically, an iterator is created from an iterable using the `iter()` function and the elements are accessed via the `next()` function.
- Iterators remember the state as you traverse through them. The next call to `next()` starts off where the previous one stopped.

In [74]:
```
with open('files/review.txt', 'r') as f:
    print(f.read())
```

```
DSC20 is so hard. It's probably the hardest class I've taken!
I have so many hard classes this quarter.
```

In [75]:
```
with open('files/review.txt', 'r') as f:
    f.readline()
    print(f.read())
```

```
I have so many hard classes this quarter.
```

In [78]:
```
with open('files/review.txt', 'r') as f:
    f.readlines()
    print(f.read())
```

Map

Map - **Syntax: map(function, iterable)**

- Map allows you to apply a function to all elements to an iterable input
- very common to use a lambda function as the function to apply
- returns a lazy iterator through the iterable object, applying the function as it traverses

```
In [12]: data = [1,2,3,4,5]
         list(map(lambda x:x+2, data))

Out[12]: [3, 4, 5, 6, 7]
```

Filter

Filter - **Syntax: filter(function, iterable)**

- Filter takes in a function that returns a boolean and only keeps elements that satisfy the function (i.e. return True).
- Very common to use a lambda function as the function to apply, but keep in mind the function **must return a boolean**.
- Returns a lazy iterator through the iterable object that only yields values that pass the function.

```
In [14]: data = [1,2,3,4,5]
         list(filter(lambda x:x%2==0,data))

Out[14]: [2, 4]
```

Checkpoint

Are the following 2 statements equivalent?

```
In [16]: data = list(range(0,101))
```

```
In [ ]: lambda_map = lambda x: x*2 if x%2==0 else 0
        sum(map(lambda_map, data))
```

```
In [ ]: lambda_filter = lambda x: x%2==0
        sum(map(lambda_map, filter(lambda_filter, data)))
```

A. Yes

B. No

Checkpoint Solution

yep!

```
In [19]: data = list(range(0,101))
```

```
In [20]: lambda_map = lambda x: x*2 if x%2==0 else 0
         sum(map(lambda_map, data))

Out[20]: 5100
```

```
In [21]: lambda_filter = lambda x: x%2==0
         sum(map(lambda_map, filter(lambda_filter, data)))

Out[21]: 5100
```

Aside: lambda complexity

Lambda functions can take on additional complexity in the form of nesting them or taking in multiple arguments

```
In [32]: temp = lambda x: lambda: lambda y: x + y
         temp(1)()(2)

Out[32]: 3
```

```
In [34]: a = [1,2,3]
         b = [4,5,6]
         temp = lambda x,y: x+y
         list(map(temp, a, b))

Out[34]: [5, 7, 9]
```

Aside: Why iterators?

Students tend to get confused about why use map/filter when they're functionally very similar to list comps or for loops. Consider the problem of scale: What happens if your data is too large to load all at once?

```
In [42]: os.listdir(broken_path)
```

```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
Cell In[42], line 1
----> 1 os.listdir('../../../../../Google Drive/My Drive/LANDSAT LC09 C02 T1_L2_calculations_2023-06-01_2023-12-01')

OSError: [Errno 89] Operation canceled: '../../../../../Google Drive/My Drive/LANDSAT LC09 C02 T1_L2_calculations_2023-06-01_2023-12-01'
```

```
In [54]: with os.scandir(broken_path) as entries:
             for _ in range(10):
                 print(next(entries).is_file())
```

```
True
True
True
```

```
True
True
True
True
True
True
True
```

Real Example

Students complained before that I should relate what we learn to real life data science (How I can do that with basic python is beyond me), but this is one of the few times where there's a direct application

In [39]:
```
import pandas as pd
data = pd.read_csv('data/reviews.csv')
```

In [40]:
```
data.shape
```

Out[40]:
```
(568454, 10)
```

In [41]:
```
data.head()
```

Out[41]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	H
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	H
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

Let's say I want to filter the entries down to entries from 2012. The Time column seems to be in unix time, which is not very human interpretable. Let's write an expression to transform it into something else.

```
In [42]: from datetime import datetime
```

```
In [43]: data['Time'] = data['Time'].apply(lambda x: datetime.utcfromtimestamp(x
         data.head()
```

Out[43]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	H
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

In [52]:
```python
data['Year'] = data['Time'].apply(lambda x: int(str(x)[:4]))
```

In [54]:
```python
data.head()
```

Out[54]:

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	H
0	1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	
1	2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	
2	3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	
3	4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	

	Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	H
4	5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	

```
In [56]: data[data['Year']==2012].shape
```

```
Out[56]: (198659, 11)
```

Thanks for coming!