Worksheet 00

Name:Wilbert Limson UID:U11894403

Topics

- course overview
- python review

Course Overview

a) Why are you taking this course?

Based on my experience in Accenture and Tokopedia, I've been involved in managing data to build a data platform as a data engineer. By being in this class, it helps me with more touch on experience as a data scientist and what they do.

b) What are your academic and professional goals for this semester?

To be in business development technology side where I have the ability to use and manage data processing.

c) Do you have previous Data Science experience? If so, please expand.

Cleaning data then being processed by BigQuery, Uploading and setting up automation for data collecting, and simple SQL query.

d) Data Science is a combination of programming, math (linear algebra and calculus), and statistics. Which of these three do you struggle with the most (you may pick more than one)?

Statistics, knowing what to use and when to use certain function.

Python review

Lambda functions

Python supports the creation of anonymous functions (i.e. functions that are not bound to a name) at runtime, using a construct called `lambda`. Instead of writing a named function as such:

In [1]:
```
def f(x):
    return x**2
f(8)
```

Out[1]:
```
64
```

One can write an anonymous function as such:

In [2]:
```
(lambda x: x**2)(8)
```

Out[2]:
```
64
```

A `lambda` function can take multiple arguments:

In [3]:
```
(lambda x, y : x + y)(2, 3)
```

Out[3]:
```
5
```

The arguments can be `lambda` functions themselves:

In [4]:
```
(lambda x : x(3))(lambda y: 2 + y)
```

Out[4]:
```
5
```

a) write a `lambda` function that takes three arguments `x, y, z` and returns `True` only if `x < y < z`.

In [5]:
```
(lambda x, y, z: x < y < z)
```

Out[5]:
```
<function __main__.<lambda>(x, y, z)>
```

b) write a `lambda` function that takes a parameter `n` and returns a lambda function that will multiply any input it receives by `n`. For example, if we called this function `g`, then `g(n)(2) = 2n`

In [6]:
```
(lambda n: (lambda x: x * n))
```

Out[6]:
```
<function __main__.<lambda>(n)>
```

Map

`map(func, s)`

`func` is a function and `s` is a sequence (e.g., a list).

`map()` returns an object that will apply function `func` to each of the elements of `s`.

For example if you want to multiply every element in a list by 2 you can write the following:

In [7]:
```
mylist = [1, 2, 3, 4, 5]
mylist_mul_by_2 = map(lambda x : 2 * x, mylist)
print(list(mylist_mul_by_2))
```

```
[2, 4, 6, 8, 10]
```

`map` can also be applied to more than one list as long as they are the same size:

In [8]:
```
a = [1, 2, 3, 4, 5]
b = [5, 4, 3, 2, 1]

a_plus_b = map(lambda x, y: x + y, a, b)
list(a_plus_b)
```

Out[8]:
```
[6, 6, 6, 6, 6]
```

c) write a map that checks if elements are greater than zero

In [9]:
```
c = [-2, -1, 0, 1, 2]
gt_zero = map(lambda x: x > 0, c)
list(gt_zero)
```

Out[9]:
```
[False, False, False, True, True]
```

d) write a map that checks if elements are multiples of 3

In [10]:
```
d = [1, 3, 6, 11, 2]
mul_of3 = map(lambda x: x % 3 == 0, d)
list(mul_of3)
```

Out[10]:
```
[False, True, True, False, False]
```

Filter

`filter(function, list)` returns a new list containing all the elements of `list` for which `function()` evaluates to `True.`

e) write a filter that will only return even numbers in the list

In [11]:
```
e = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = filter(lambda x: x % 2 == 0, e)
list(evens)
```

Out[11]:
```
[2, 4, 6, 8, 10]
```

Reduce

`reduce(function, sequence[, initial])` returns the result of sequentially applying the function to the sequence (starting at an initial state). You can think of reduce as consuming the sequence via the function.

For example, let's say we want to add all elements in a list. We could write the following:

In [12]:
```
from functools import reduce

nums = [1, 2, 3, 4, 5]
sum_nums = reduce(lambda acc, x : acc + x, nums, 0)
print(sum_nums)
```

```
15
```

Let's walk through the steps of `reduce` above:

1) the value of `acc` is set to 0 (our initial value) 2) Apply the lambda function on `acc` and the first element of the list: `acc` = `acc` + 1 = 1 3) `acc` = `acc` + 2 = 3 4) `acc` = `acc` + 3 = 6 5) `acc` = `acc` + 4 = 10 6) `acc` = `acc` + 5 = 15 7) return `acc`

`acc` is short for `accumulator`.

f) `*challenging` Using `reduce` write a function that returns the factorial of a number. (recall: N! (N factorial) = N *(N - 1)* (N - 2) ... 2 * 1)

In [13]:
```
factorial = lambda x: reduce(lambda acc, y: acc * y, range(1, x + 1), 1)
factorial(10)
```

Out[13]:
```
3628800
```

g) `*challenging` Using `reduce` and `filter`, write a function that returns all the primes below a certain number

In [14]:
```
sieve = lambda x: reduce(lambda r, y: r + [y] if all(y % p != 0 for p in r) else r, ra
print(sieve(100))
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79,
83, 89, 97]
```

What is going on?

For each of the following code snippets, explain why the result may be unexpected and why the output is what it is:

In [15]:
```
class Bank:
  def __init__(self, balance):
    self.balance = balance

  def is_overdrawn(self):
    return self.balance < 0

myBank = Bank(100)
if myBank.is_overdrawn :
  print("OVERDRAWN")
else:
  print("ALL GOOD")
```

```
OVERDRAWN
```

Eventhough the balance result is positive it still output "OVERDAWN" this is mainly because of the statement when they check myBank.is_overdrawn the method project in Python are like TRUE so when the statement like None, False, 0 therefore the condition is passed as true and the condition is passed as true and printed as "OVERDRAWN"

In [16]:
```
for i in range(4):
    print(i)
    i = 10
```

```
0
1
2
3
```

The unexpected output is caused by printing only 0,1,2,3 despite i is being set to 10 within the loop, however the error is as it print in the range of 4, it was reassigned with each iteration of the loop that make the internal modication of i = 10 is irrelevant. The loop will always follow the sequence that is geenrated by range(4).

In [17]:
```
row = [""] * 3 # row i['', '', '']
board = [row] * 3
print(board) # [['', '', ''], ['', '', ''], ['', '', '']]
board[0][0] = "X"
print(board)
```

```
[['', '', ''], ['', '', ''], ['', '', '']]
[['X', '', ''], ['X', '', ''], ['X', '', '']]
```

When you set the first element of the first sublist in board to "X", it unexpectedly changes the first element of every sublist to "X". This happens because board = row * 3 actually creates a list containing three references to the same row object, not three separate row objects. So, when you modify one element in any of these sublists, it changes that element in all of them, as they are all referring to the same underlying list. To avoid this, you should create each sublist as a separate object, which can be done using a loop or list comprehension, ensuring each sublist is independent.

In [18]:
```
funcs = []
results = []
for x in range(3):
    def some_func():
        return x
    funcs.append(some_func)
    results.append(some_func())  # note the function call here

funcs_results = [func() for func in funcs]
print(results) # [0,1,2]
print(funcs_results)
```

```
[0, 1, 2]
[2, 2, 2]
```

In the results list correctly shows 0,1,2 however the funcs_results ends up as 2,2,2 . This happens because when some_func() is called within the loop to generate results, it uses the current loop value of x. However, the functions stored in funcs are closures that hold onto the x variable itself, not the value it had at each loop iteration. Due to this late binding of x, when these functions are finally called after the loop, they all see x as 2, the last value it held in the loop. So, each function in funcs ends up returning 2, leading to funcs_results being 2,2,2

In [19]:
```
f = open("./data.txt", "w+")
f.write("1,2,3,4,5")
f.close()

nums = []
```

```
with open("./data.txt", "w+") as f:
  lines = f.readlines()
  for line in lines:
    nums += [int(x) for x in line.split(",")]

print(sum(nums))
```

```
0
```

the sum of numbers unexpectedly is 0. This is because the file is opened in "w+" mode for the second time, which actually clears out all the data previously written to the file. So, when the program tries to read the file using readlines(), it finds the file empty and as a result, the nums list remains empty. To fix this, the file should be opened in "r" mode for reading after the initial write operation, ensuring that the data written to the file is actually read back instead of being erased.