1 Worksheet 00

Name: Jian Xie

UID: 75516303

1.0.1 Topics

- course overview
- python review

1.0.2 Course Overview

a) Why are you taking this course?

Because I want to gain some data science experience.

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

Grasp well in-class topics and real-world project skills.

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

Not exatly Data Science but some computer vision experience with a pa

d) Data Science is a combination of programming, math (linear algebra a
Which of these three do you struggle with the most (you may pick more

Maybe statistics.

1.0.3 Python review

1.0.3.1 Lambda functions

Python supports the creation of anonymous functions (i.e. functions tha
runtime, using a construct called `lambda` . Instead of writing a named fu

In [1]:

```python
def f(x):
    return x**2
f(8)
```

```
64
```

One can write an anonymous function as such:

In [2]:
```
(lambda x: x**2)(8)
```
```
64
```

A `lambda` function can take multiple arguments:

In [1]:
```
(lambda x, y : x + y)(2, 3)
```
```
5
```

The arguments can be `lambda` functions themselves:

In [4]:
```
(lambda x : x(3))(lambda y: 2 + y)
```
```
5
```

a) write a `lambda` function that takes three arguments `x, y, z` and retu

In [10]:
```
lambda_func_1 = lambda x, y, z : x < y < z
lambda_func_1(1,2,3)
```
```
True
```

b) write a `lambda` function that takes a parameter `n` and returns a lamb
any input it receives by `n`. For example, if we called this function `g`, the

In [12]:
```
lambda_func_2 = lambda n: lambda x: x * n
lambda_func_2(2)(3)
```
```
6
```

1.0.3.2 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 ele

For example if you want to multiply every element in a list by 2 you can

In [15]:
```python
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 sa

In [9]:
```python
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)
```

```
[6, 6, 6, 6, 6]
```

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

In [16]:
```python
c = [-2, -1, 0, 1, 2]
gt_zero = map(lambda x : x > 0, c)
list(gt_zero)
```

```
[False, False, False, True, True]
```

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

In [17]:
```python
d = [1, 3, 6, 11, 2]
mul_of3 = map(lambda x : x * 3, d)
list(mul_of3)
```

```
[3, 9, 18, 33, 6]
```

1.0.3.3 Filter

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

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

In [18]:

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

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

1.0.3.4 Reduce

`reduce(function, sequence[, initial])` returns the result of sequen
the sequence (starting at an initial state). You can think of reduce as con
function.

For example, let's say we want to add all elements in a list. We could wri

In [20]:

```
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 fun
element of the list: `acc` = `acc` + 1 = 1 3) `acc` = `acc` + 2 = 3 4) `acc` =
= 10 6) `acc` = `acc` + 5 = 15 7) return `acc`

`acc` is short for `accumulator`.

f) `*challenging` Using `reduce` write a function that returns the factoria
factorial) = N * (N - 1) * (N - 2) * ... * 2 * 1)

In [24]:

```
factorial = lambda n : reduce(lambda x, y: x * y, range(1, r
factorial(10)
```

```
3628800
```

g) `*challenging` Using `reduce` and `filter`, write a function that retur
certain number

In [26]:
```python
sieve = lambda n : reduce(lambda r, x: r - set(range(x**2, r
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,
```

1.0.4 What is going on?

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

In [25]:
```python
1  class Bank:
2    def __init__(self, balance):
3      self.balance = balance
4
5    def is_overdrawn(self):
6      return self.balance < 0
7
8  myBank = Bank(100)
9  if myBank.is_overdrawn :
10   print("OVERDRAWN")
11 else:
12   print("ALL GOOD")
```

```
OVERDRAWN
```

The result is unexpected since the account is all good.

The reason why the output is overdrawn exsits in line 9 'if myBank.is_ove
'is_overdrawn' method. Instead, it references the method object itself. In
object, including functions and methods. We have to use 'myBank.is_ove

In [2]:
```python
1  for i in range(4):
2      print(i)
3      i = 10
```

```
0
1
2
3
```

Line 3 'i = 10' may wanna change the value of variable i and output it. Bu

This is because after every time line 3 works, the for loop is assigning ne
is unnecessary.

In [4]:

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

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

The original idea is to change element 'board[0][0]' to "X", not the first e

The key point here is understanding what happens in the line board = [r
because each row of element 'board' is a copy of element 'row'. When th
changed, this change is reflected across all three rows. To create a board
we should create each row separately.

In [5]:

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

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

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

'results': results.append(some_func()). 'some_func()' is called immediately
'results' should store the current value of the loop variable (0, 1, 2), whic

'funcs_results' may be unexpected: funcs_results = [func() for func in fun
funcs iterately which refers to a closed function 'some_func'. As this func
x is 2 now, it will be [2, 2, 2].

In [15]:

```
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 'nums' is unexpected, which should be 15.

The reason why this is 0 is that when opening the file second time, we u
reading and writing and will erase the file first. This causes the file to be
for just reading.