DSC 20
Discussion Section 4

ARDA CANKAT BATI

Today's Plan

1. Going over 2 practice midterm questions
2. Helpful notes about some topics including:
   - Lamda Functions
   - Higher order functions
   - Scope
   - Generators
   - Dictionaries

Practice Midterm Review

Countizard

(a) Implement counter, which

- Takes a non-negative single-digit integer d.
- It **returns a function count** that takes a non-negative integer n and returns the number of times that d appears as a digit in n.

```
>>> counter(8)(8018)
2
>>> counter(0)(2016)
1
>>> counter(0)(0)
0
```

Practice Midterm Review

```
def counter(d):
    """Return a function of N that returns the number of times D appears in N.
    >>> counter(8)(8018)
    2
    """
    def count(n):
        k = 0
        #For example assume n = 8018 initially
        # With the below loop, we are going over each digit
        while n > 0:
            #Now n = 801, last = 8
            n, last = n // 10, n % 10
            # last == 8 so increase k by 1
            if last == d:
                k += 1
        # After going over all digits, return k
        return k
    # Return the function count
    return count
```

Practice Midterm Review

Caterepeat

(a) (4 pt) Implement repeat_sum, which takes:

- **a one-argument function f**
- **a value x**
- **a non-negative integer n.**

It returns the sum of n + 1 terms. Each term, indexed by k starting at 0, is the result of applying f to x repeatedly k times. You may assign to only one name in each of the three assignment statements.

Practice Midterm Review

```
def repeat_sum(f, x, n):

"""
Compute the following summation of N+1 terms, where the last term calls F N times:

x + f(x) + f(f(x)) + f(f(f(x))) + ... + f(f(...f(x)))

>>> repeat_sum(lambda x: x*x, 3, 0) # 3
3
>>> repeat_sum(lambda x: x*x, 3, 1) # 3 + 9
12
>>> repeat_sum(lambda x: x+2, 3, 4) # 3 + 5 + 7 + 9 + 11
35
"""
```

Practice Midterm Review

```
def repeat_sum(f, x, n):
"""
Omitted for shortness
"""

total, k = 0, 0 #Initializing variables
# total is the sum of the operations
# k is the counter of many times we call f

    while k <= n: # While we can still call f on itself
        total = total + x
        # Call the current x, x_prev
        x = f(x)  # Here the new x we get is x = f(x_prev)
        # When we continue the loop, we get x = f(f(...f(x_prev)))
        k = k + 1 # Counting how deep the f cal was

    return total
```

```
x + f(x) + f(f(x)) + f(f(f(x))) + ... + f(f(...f(x)))

>>> repeat_sum(lambda x: x*x, 3, 0) # 3
3
>>> repeat_sum(lambda x: x*x, 3, 1) # 3 + 9
12
>>> repeat_sum(lambda x: x+2, 3, 4) # 3 + 5 + 7 + 9 + 11
35
"""
```

Lamba Notes

```
z = 3

sum_func1 = lambda x,y : x + y + z

def sum_func2(x,y):
    return x + y + z

print(sum_func1(2,3))
print(sum_func2(2,3))
```

```
8
8
```

HOF & Lamba Notes

```
hof1 = lambda x: lambda y: x(y)

def hof2(x):

    def sub_func(y):
        return x(y)

    return sub_func

input_func = lambda x: x ** 2
y = 3

print(hof1(input_func)(y))
print(hof2(input_func)(y))
```

```
9
9
```

HOF & Lamba Notes

```
hof1 = lambda x: lambda y: x(y)

def hof2(x):

    def sub_func(y):
        return x(y)

    return sub_func

input_func = lambda x: x ** 2
y = 3

print(hof1(input_func)(y))
print(hof2(input_func)(y))
```

```
9
9
```

```
input_func = lambda x: x ** 3

#returns sub_func(y) which has x = x1
sub_func1 = hof1(input_func)
print(sub_func1(1))
print(sub_func1(2))
print(sub_func1(3))
```

```
1
8
27
```

HOF Notes

```
hof = lambda x: lambda y: x(y)
input_func = lambda x: x ** 3

numbers = [1,2,3,4,5,6]
map_func = hof(input_func)
squares = map(lambda number: map_func(number), numbers)
list(squares)
```

```
[1, 8, 27, 64, 125, 216]
```

HOF Notes

```
def hof(x):

    def sub_func(y):
        return x + y

    return sub_func

numbers = [1,2,3]
plus_one = map(lambda number: hof(1)(number), numbers)
list(plus_one)
```

```
[2, 3, 4]
```

HOF Notes

```
def hof(x):

    def sub_func(y):
        return x + y

    return sub_func

numbers = [1,2,3]
plus_one = map(lambda number: sub_func(number), numbers)
list(plus_one)
```

HOF Notes

```
def hof(x):

    def sub_func(y):
        return x + y

    return sub_func

numbers = [1,2,3]
plus_one = map(lambda number: sub_func(number), numbers)
list(plus_one)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-87-e6d08b2fad1e> in <module>
      8 numbers = [1,2,3]
      9 plus_one = map(lambda number: sub_func(number), numbers)
---> 10 list(plus_one)

<ipython-input-87-e6d08b2fad1e> in <lambda>(number)
      7
      8 numbers = [1,2,3]
----> 9 plus_one = map(lambda number: sub_func(number), numbers)
     10 list(plus_one)

NameError: name 'sub_func' is not defined
```

HOF Notes

```
def hof(x):

    def sub_func(y):
        return x + y

    return sub_func

sub_func = hof(1)

numbers = [1,2,3]
plus_one = map(lambda number: sub_func(number), numbers)
list(plus_one)
```

```
[2, 3, 4]
```

HOF & Scope Notes

```
def func1(x,y):
    def sub_func1(y):
        return x + y
    return sub_func1

def func2(x, y):
    return sub_func2(y)

def sub_func2(y):
    return x + y

print(func1(1,2))
print(func1(1,2)(3))
```

```
<function func1.<locals>.sub_func1 at 0x000002C2043C3D90>
4
```

Why is the output here 4 instead of 3?

HOF & Scope Notes

```
def func1(x,y):
    def sub_func1(y):
        return x + y
    return sub_func1

def func2(x, y):
    return sub_func2(y)

def sub_func2(y):
    return x + y

print(func1(1,2))
print(func1(1,2)(3))
```

```
<function func1.<locals>.sub_func1 at 0x000002C20
4
```

```
print(func2(1,2))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-c987b2566815> in <module>
----> 1 print(func2(1,2))

<ipython-input-45-b424467f685c> in func2(x, y)
      5
      6 def func2(x, y):
----> 7     return sub_func2(y)
      8
      9 def sub_func2(y):

<ipython-input-45-b424467f685c> in sub_func2(y)
      8
      9 def sub_func2(y):
---> 10     return x + y
     11
     12 print(func1(1,2))

NameError: name 'x' is not defined
```

Generators & Generator Functions

```
def gen_func():

    inital_val = 2
    while True:
        yield inital_val
        inital_val *= 2

a_func = gen_func #Notice no parentheses
print(type(a_func))
next(a_func)
```

```
<class 'function'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-55-070cd092c32c> in <module>
      8 a_func = gen_func
      9 print(type(a_func))
---> 10 next(a_func)

TypeError: 'function' object is not an iterator
```

Notice the lack of parentheses here!

It is a generator function, but not a generator yet.

Generators & Generator Functions

```
def gen_func():

    inital_val = 2
    while True:
        yield inital_val
        inital_val *= 2

a_func = gen_func #Notice no parentheses
print(type(a_func))
next(a_func)
```

a_func = gen_func
gives us a generator function.
The generator function is not an iterable so we can't call next() on it

```
a_gen = gen_func() #Notice we have parentheses
print(type(a_gen))
print(next(a_gen))
print(next(a_gen))
print(next(a_gen))
```

```
<class 'generator'>
2
4
8
```

a_func = gen_func()
'calls' the generator function which gives us the generator. The generator is an iterable so we can call next() on it

Dictionaries

```
# Below we are doing dictionary comprehension
squares_dict = {key: value for (key, value) in [(x,x**2) for x in range(10)]}
print(squares_dict,'\n')
print('dict_keys  ',list(squares_dict.keys()))
print('dict_values',list(squares_dict.values()))
print('dict_items  ',list(squares_dict.items()))
```

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}

dict_keys   [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
dict_values [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
dict_items   [(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
```

Notice the dict. comprehension:
num_dict = {key: value for (key,value) in iterator}

Dictionaries

```
d0 = {0: 999}
squares_dict.update(d0)
print(squares_dict.get(0))
print('dict_items  ',list(squares_dict.items()))
```

```
999
dict_items   [(0, 999), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25), (6, 36), (7, 49), (8, 64), (9, 81)]
```