SECTION 13: PYTHON GENERATORS - 17 minutes, 3 parts

3/3 Generators Homework Solutions

- **solutions for iterators and generators**
  - **generating the sum of the squares**
    - -> iterating through a range
    - -> and then using yield keyword to square its elements and return the fun of squares

Iterators and Generators Homework - Solution

Problem 1

Create a generator that generates the squares of numbers up to some number N.

```
In [1]: def gensquares(N):
            for i in range(N):
                yield i**2
```

```
In [2]: for x in gensquares(10):
            print(x)
```

```
0
1
4
9
16
25
36
49
64
```

  - **then creating a generator for random numbers**
    - -> create a generator for random numbers, between some low and high number
    - -> we are taking advantage of the random.randint function
    - -> we are generating the number and then yielding it out of the function

Problem 2

Create a generator that yields "n" random numbers between a low and high number (that are inputs).
Note: Use the random library. For example:

```
In [3]: import random

        random.randint(1,10)
```

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

```
In [4]: def rand_num(low,high,n):

            for i in range(n):
                yield random.randint(low, high)
```

```
In [5]: for num in rand_num(1,10,12):
            print(num)
```

```
3
9
```

  - **the iter() function**
    - -> using the iter() function to convert the string into an iterator
    - -> the function generates a random number and then yields it
    - -> you can concert a string from an iterable into an iterator
    - -> use the iter function

Problem 3

Use the iter() function to convert the string below into an iterator:

```
In [6]: s = 'hello'

        s = iter(s)

        print(next(s))
```

```
h
```

  - **building an iterator for another use case**
    - -> using a iter statement
    - -> if the output creates a large amount of memory

Problem 4

Explain a use case for a generator using a yield statement where you would not want to use a normal function with a return statement.

If the output has the potential of taking up a large amount of memory and you only intend to iterate through it, you would want to use a generator. (Multiple answers are acceptable here!)

  - **then gencomp**
    - -> generator comprehension is like list comprehension
    - -> we are going to generate it
    - -> iterating the generator

Extra Credit!

Can you explain what *gencomp* is in the code below? (Note: We never covered this in lecture!)

```
In :  my_list = [1,2,3,4,5]

      gencomp = (item for item in my_list if item > 3)

      for item in gencomp:
          print(item)
```

```
4
5
```

Hint: Google *generator comprehension*!