COMP1531

8.3 Python Generators

Important information

Project

- The end of the project is approaching....
- **Please make sure to read the spec where it outlines individual expectations**
- My recent analysis suggests there are a significant number who have or will not meet all the listed criteria

The Global Keyword

- Global is only necessary if you intend to *assign* to a global variable

```
1  message = ["Hello", "I'm", "stored", "in", "a", "global", "variable"]
2
3  def example1():
4      # Only reading from the variable; don't need global
5      print(message)
6
7  def example2():
8      # Modifying the list stored in the variable; don't need global.
9      message[0] = "G'day"
10
11 def example3():
12     # Calling a method on the object stored in the variable; don't need global
13     message.append("mate")
14
15 def example4():
16     # Assigning a new value to a variable; need global
17     global message
18     message = ["Good", "day", "sir", "I", "am", "a", "variable", "most", "global"]
```

Iterators

- Let us represent countable sets of values
- The for loop in python works for any iterator
- Example iterators:

```
1  class Squares:
2      def __init__(self):
3          self.i = 0
4
5      def __iter__(self):
6          return self
7
8      def __next__(self):
9          self.i += 1
10         return self.i*self.i
```

```
1  class Fibonacci():
2      def __init__(self):
3          self.a = 0
4          self.b = 1
5
6      def __iter__(self):
7          return self
8
9      def __next__(self):
10         c = self.a + self.b
11         self.a = self.b
12         self.b = c
13         return self.a
```

Generators

- A different way of writing iterators
- Defined via generator functions instead of classes
- Example generator

```
1 def simple_generator():
2     print("Hello")
3     yield 1
4     print("Nice to meet you")
5     yield 2
6     print("I am a generator")
```

Generators

- Intuitively, you can think of a generator as a suspendable computation
- Calling next() on a generator executes it until it reaches a yield, at which point it is suspended (frozen) until the subsequent call to next()

Generators

- More useful examples

```
1 def squares():
2     i = 0
3     while True:
4         i += 1
5         yield i*i
```

```
1 def fib():
2     a = 0
3     b = 1
4     while True:
5         c = a + b
6         a = b
7         b = c
8         yield a
```