COMP1531

9.1 Python Generators

Iterators

- In Python, iterators are objects *containing* a countable number of elements
- For example, we can get an iterator for a list:

```
1 animals = ["dog", "cat", "chicken", "sheep"]
2
3 animal_iterator = iter(animals)
```

Iterators

- Any object with the methods __iter__() and __next__() is an iterator
- Duck typing ^^^
- Simple example (squares)

```
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
```

For loops

- Python for loops use iterators behind the scenes
- This is valid code:

```
squares = Squares()

for i in squares: # Loops forever
    print(i)
```

Iterator vs Iterable

- Intuitively:
  - An iterator stores the state of the iteration (i.e. where it's up to).
  - Something is iterable if it can be iterated over.
- Concretely:
  - An iterator has __iter__() and __next()__ methods.
  - Iterables have __iter__() methods
- Thus, all iterators are iterable, but not all iterables are iterators
- For example, lists are iterable, but they are not iterators
- For loops only need to be given something *iterable*

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 = 1
3     b = 1
4     while True:
5         yield a
6         a, b = b, a+b
```