Python VI

List Comprehensions

Iterators

- Iterators repeat a similar calculation over and over again
- What really goes on when looping through a list: the iterator and next is used

```
In [1]: for i in range(10):
            print(i, end=' ')

0 1 2 3 4 5 6 7 8 9


In [2]: for value in [2, 4, 6, 8, 10]:
            # do some operation
            print(value + 1, end=' ')

3 5 7 9 11


In [3]:  iter([2, 4, 6, 8, 10])

Out [3]: <list_iterator at 0x104722400>


In [4]: I = iter([2, 4, 6, 8, 10])

In [5]: print(next(I))

2

In [6]: print(next(I))

4

In [7]: print(next(I))

6
```

Range

- Range is not a list
- It exposes an iterator so it can be looped through
- It lazy loads by using `next`

```
In [8]:  range(10)

Out [8]: range(0, 10)
```

range, like a list, exposes an iterator:

```
In [9]:  iter(range(10))

Out [9]: <range_iterator at 0x1045a1810>
```

So Python knows to treat it *as if* it's a list:

```
In [10]: for i in range(10):
             print(i, end=' ')

0 1 2 3 4 5 6 7 8 9
```

```
In [11]: N = 10 ** 12
         for i in range(N):
             if i >= 10: break
             print(i, end=', ')

0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Useful iterators

- `enumerate` can be used to go through a list and get indices and values
- `zip` allows one to enumerate through two lists
- `map` and `filter` are also iterators

```
In [13]: L = [2, 4, 6, 8, 10]

In [14]: for i, val in enumerate(L):
             print(i, val)

0 2
1 4
2 6
3 8
4 10
```

```
In [15]: L = [2, 4, 6, 8, 10]
         R = [3, 6, 9, 12, 15]
         for lval, rval in zip(L, R):
             print(lval, rval)

2 3
4 6
6 9
8 12
10 15
```

Basic list comprehensions

- Consider these two equivalent pieces of code
- The second, more “pythonic” way is a list comprehension

```
In [2]: L = []
        for n in range(12):
            L.append(n ** 2)
        L

Out [2]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]

In [3]: [n ** 2 for n in range(12)]

Out [3]: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

Multiple iteration, conditionals

- Building a value from two lists
- A list with conditionals
- The loop equivalent

```
In [4]:  [(i, j) for i in range(2) for j in range(3)]

Out [4]: [(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2)]
```

```
In [5]:  [val for val in range(20) if val % 3 > 0]

Out [5]: [1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19]
```

```
In [6]:  L = []
         for val in range(20):
             if val % 3:
                 L.append(val)
         L

Out [6]: [1, 2, 4, 5, 7, 8, 10, 11, 13, 14, 16, 17, 19]
```

More conditionals

- Assign a value depending on a condition
- Using a conditional assignment in a list comprehension
- Break up long list comprehensions with a line break

```
In [7]:  val = -10
         val if val >= 0 else -val

Out [7]: 10
```

```
In [8]:  [val if val % 2 else -val
          for val in range(20) if val % 3]

Out [8]: [1, -2, -4, 5, 7, -8, -10, 11, 13, -14, -16, 17, 19]
```

Other types of comprehension

- Set comprehension
- Dictionary comprehension

```
In [9]:  {n**2 for n in range(12)}

Out [9]: {0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121}
```

```
In [11]:  {n:n**2 for n in range(6)}

Out [11]: {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```