List Comprehensions

List comprehensions are declarative ways of defining lists in python. The following ways of building a list are equivalent:

```
l = [x**2 for x in range(1, 10)]

l = []
for x in range(1, 10):
    l.append(x**2)
```

```
m = [x for x in range(1, 10) if x % 2 == 1]

m = []
for x in range(1, 10):
    if x % 2 == 1:
        m.append(x)
```

```
n = [(x, y) for x in range(1, 10) for y in range(1, 10) if x > y]

n = []
for x in range(1, 10):
    for y in range(1, 10):
        if x > y:
            n.append((x, y))
```

The general form is:

```
[expression for x in y condition]
```

Dictionary Comprehensions

Dictionary comprehensions are the analogous construct for dictionaries:

```
o = {x: x**3 for x in range(1, 100) if math.sqrt(x).is_integer()}

o = {}
for x in range(1, 100):
    if math.sqrt(x).is_integer():
        o[x] = x**3
```

I have never used a dictionary comprehension outside of exercises. I don't know how useful they are, and I'm struggling to think of good examples.

Generators

A generator is a function that yields values rather than returning them. For example:

```
def squares():
    i = 0
    while True
```

```
        yield i**2
        i += 1
```

The built in *range* function is similar to a generator. Generators are iterable:

```
for square in squares():
    print(square)
    if square > 10000:
        break
```

But only once!

Generators can take arguments:

```
def exponents(exp):
    i = 0
    while True
        yield i**exp
        i += 1
```

Generator expressions are similar to list comprehensions:

```
squares = (i**2 for i in range(10000))
```

Generators are lazy, values are calculated when needed, meaning they can represent infinite sequences without requiring infinite memory.