Common Mistakes

Print vs. Return

```
def bigger_than_three(num):
    """ Returns True if num is bigger
    than three, returns False otherwise

    Parameters: num (an integer or float)
    Returns: boolean
    """
    pass
```

Print vs. Return

```
def try1_bigger_than_three(num):
    return num > 3

def try2_bigger_than_three(num):
    print num > 3
```

Print vs. Return

```
def try1_bigger_than_three(num):
    return num > 3

def try2_bigger_than_three(num):
    print num > 3

x = 10
if try1_bigger_than_three(x):
    print "try1: %i is bigger than three" % x
if try2_bigger_than_three(x):
    print "try2: %i is bigger than three" % x
```

Print vs. Return

The vast majority of functions you write should not print anything.

If you print stuff inside of a function, that should be part of the spec and function name, i.e.

print_scrabble_board()
print_histogram()
output_debug_score_list()

Expressions

What's the difference between `{}` and `x = {}`?

Expressions

```
# Append an empty dictionary to y
d = {}
y.append(d)

# Simplified
y.append({})
```

Expressions

Using `[]` on a list or dictionary looks something up.

```
x = [1, 2, 3, 4]
y = x[1]

# Same thing as the above!
y = [1, 2, 3, 4][1]
```

Expressions

What is the value of this?

```
{3: { 2: [1, 2, 3, 4] } }[3][2][1]
```

Expressions

What is the value of this?

```
({3: { 2: [1, 2, 3, 4] } }[3])[2][1]
```

First, look up key 3 in the outer dictionary:

```
({ 2: [1, 2, 3, 4] })[2][1]
```

Expressions

What is the value of this, continued:

```
({ 2: [1, 2, 3, 4] }[2])[1]
```

Then, look up key 2 in the next dictionary:

```
([1, 2, 3, 4])[1]
```

Expressions

What is the value of this, continued:

```
([1, 2, 3, 4][1])
```

Finally, look up index 1 in the inner list:

```
(2)
```

Double for loops (what do they mean?)

```
for x in range(3):
    for y in range(3):
        print x, y
```

Double for loops (what do they mean?)

Remember that a for loop is like assigning a variable and then running a block of code multiple times.

```
for x in range(3):
    value = do_something(x)
    print value
```

Double for loops (manually expanded)

```
x = 0
value = do_something(x)
print value
x = 1
value = do_something(x)
print value
x = 2
value = do_something(x)
print value
```

Double for loops (back to original loop)

```
for x in range(3):
    for y in range(3):
        print x, y
```

How can this be expanded?

Double for loops (expand one loop at a time)

```
x = 0
for y in range(3):
    print x, y
x = 1
for y in range(3):
    print x, y
x = 2
for y in range(3):
    print x, y
```

Double for loops (...and now the other loop)

```
x = 0
y = 0
print x, y
y = 1
print x, y
y = 2
print x, y

# continued up ->
```

```
x = 1
y = 0
print x, y
y = 1
print x, y
y = 2
print x, y

# ...and for y=2
```