Announcements

Functional Abstractions

```python
def square(x):
    return pow(x, 2)
def sum_squares(x, y):
    return square(x) + square(y)
```

If the name "square" were bound to a built-in function, `sum_squares` would still work identically.

Choosing Names

Names should convey the meaning or purpose of the values to which they are bound.

The type of value bound to the name is best documented in a function's docstring.

Function names typically convey their effect (print), their behavior (triple), or the value returned (abs).

Which Values Deserve a Name

Reasons to add a new name

- Repeated compound expressions:
  ```python
  if sqrt(square(a) + square(b)) > 1:
      x = x + sqrt(square(a) + square(b))
  ```

- Meaningful parts of complex expressions:
  ```python
  x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
  ```

More Naming Tips

- Names can be long if they help document your code:
  ```python
  average_age = average(age, students)
  ```

- Names can be short if they represent generic quantities: counts, arbitrary functions, arguments to mathematical operations, etc.
  ```python
  n, k, i - Usually integers
  x, y, z - Usually real numbers
  r, p, n - Usually functions
  ```

Test-Driven Development

Write the test of a function before you write the function.

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Bonus idea: Run your code interactively.

Don't be afraid to experiment with a function after you write it.

Interactive sessions can become doctests. Just copy and paste.

Test-Driven Development

Testing
**Currying**

Currying: Transform a multi-argument function into a single-argument, higher-order function.

```python
def make_adder(n):
    return lambda k: n + k

>>> make_adder(2)(3)
5
```

**Function Decorators**

```python
@trace1
def triple(x):
    return 3 * x
def triple(x):
    return 3 * x
triple = trace1(triple)
```

Why not just use this?

```python
def delay(arg):
    print('delayed')
def g():
    return arg
return g
```

What Would Python Print?

The print function returns None. It also displays its arguments (separated by spaces) when it is called.

```python
from operator import add, mul
def square(x):
    return mul(x, x)
```

Names in nested def statements can refer to their enclosing scope

Interactive Output

```
None
5
5
5
1
5
None
5
1
5
None
5
None
```