Announcements
Abstraction
Functional Abstractions

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

What does `sum_squares` need to know about `square`?

- Square takes one argument.  
  - Yes
- Square has the intrinsic name `square`.  
  - No
- Square computes the square of a number.  
  - Yes
- Square computes the square by calling `mul`.  
  - No

```python
def square(x):
    return pow(x, 2)
def square(x):
    return mul(x, x-1) + x
```

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

Names typically don’t matter for correctness

*but*

they matter a lot for composition

<table>
<thead>
<tr>
<th>From:</th>
<th>To:</th>
</tr>
</thead>
<tbody>
<tr>
<td>true_false</td>
<td>rolled_a_one</td>
</tr>
<tr>
<td>d</td>
<td>dice</td>
</tr>
<tr>
<td>helper</td>
<td>take_turn</td>
</tr>
<tr>
<td>my_int</td>
<td>num_rolls</td>
</tr>
<tr>
<td>l, I, 0</td>
<td>k, i, m</td>
</tr>
</tbody>
</table>

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:

\[
\text{if } \sqrt{\text{square}(a) + \text{square}(b)} > 1: \\
x = x + \sqrt{\text{square}(a) + \text{square}(b)}
\]

\[
\text{hypotenuse} = \sqrt{\text{square}(a) + \text{square}(b)} \\
\text{if } \text{hypotenuse} > 1: \\
x = x + \text{hypotenuse}
\]

Meaningful parts of complex expressions:

\[
x_1 = \frac{-b + \sqrt{\text{square}(b) - 4 \times a \times c}}{2 \times a}
\]

\[
\text{discriminant} = \text{square}(b) - 4 \times a \times c \\
x_1 = \frac{-b + \sqrt{\text{discriminant}}}{2 \times a}
\]

More Naming Tips

• Names can be long if they help document your code:

\[
\text{average}_\text{age} = \text{average}(\text{age}, \text{students})
\]

is preferable to

\[
\# \text{ Compute average age of students} \\
\text{aa} = \text{avg}(a, st)
\]

• Names can be short if they represent generic quantities: counts, arbitrary functions, arguments to mathematical operations, etc.

\[
n, k, i - \text{ Usually integers} \\
x, y, z - \text{ Usually real numbers} \\
f, g, h - \text{ Usually functions}
\]
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.
Currying
Function Currying

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

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

Curry: Transform a multi-argument function into a single-argument, higher-order function
Decorators
Function Decorators

(Demo)

@trace1
def triple(x):
    return 3 * x

is identical to

def triple(x):
    return 3 * x
triple = trace1(triple)

Why not just use this?
Review
What Would Python Display?

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)

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

# A function that takes any argument and returns a function that returns that arg

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

# Names in nested def statements can refer to their enclosing scope
```

### This expression | Evaluates to | Interactive Output
---|---|---
5 | 5 | 5
print(5) | None | 5 None
power(5) | None | 5 None
delay(delay)(6)() | 6 | delayed delayed 6
delay(print)()()4) | None | delayed 4 None
```python
def horse(mask):
    horse = mask
    return horse

mask = lambda horse: horse(2)
horse(mask)
```