Higher-Order Functions
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

Finding common structure allows for shared implementation

(Demo)
Generalizing Over Computational Processes

The common structure among functions may be a computational process, rather than a number.

\[
\sum_{k=1}^{5} k = 1 + 2 + 3 + 4 + 5 = 15
\]

\[
\sum_{k=1}^{5} k^3 = 1^3 + 2^3 + 3^3 + 4^3 + 5^3 = 225
\]

\[
\sum_{k=1}^{5} \frac{8}{(4k-3)(4k-1)} = \frac{8}{3} + \frac{8}{35} + \frac{8}{99} + \frac{8}{195} + \frac{8}{323} = 3.04
\]

(Demo)
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence.\n>>> summation(5, cube)\n225\n"""
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

0 + 1 + 8 + 27 + 64 + 125
Functions as Return Values

(Demo)
Locally Defined Functions

Functions defined within other function bodies are bound to names in a local frame

A function that returns a function

```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."
    def adder(k):
        return k + n
    return adder
```

The name add_three is bound to a function

A def statement within another def statement

Can refer to names in the enclosing function

```python
>>> add_three = make_adder(3)
>>> add_three(4)
7
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

Operator

Operand

An expression that evaluates to its argument

func make_adder(n):
def adder(k):
    return k + n
return adder

func adder(k)

make_adder(1)

3

( 2 )

make_adder(1)

func adder(k)

make_adder(1)

func make_adder(n)
Function Example: Sounds
WAV Files

The Waveform Audio File Format encodes a sampled sound wave.

A triangle wave is the simple wave form with the most pleasing sound.

https://en.wikipedia.org/wiki/Triangle_wave

https://en.wikipedia.org/wiki/Sampling_(signal_processing)
Function Composition

(Demo)
1. `def square(x):`  
   `return x * x`  
2. `def make_adder(n):`  
   `def adder(k):`  
   `return k + n`  
   `return adder`  
3. `def compose1(f, g):`  
   `def h(x):`  
   `return f(g(x))`  
   `return h`  
4. `compose1(square, make_adder(2))(3)`  

Return value of make_adder is an argument to compose1
Abstraction
Functional Abstractions

- 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**

What does sum_squares need to know about square?

```python
def square(x):
    return mul(x, x)

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 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:

```python
if sqrt(square(a) + square(b)) > 1:
x = x + sqrt(square(a) + square(b))
```

```python
hypotenuse = sqrt(square(a) + square(b))
if hypotenuse > 1:
x = x + hypotenuse
```

Meaningful parts of complex expressions:

```python
x1 = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```python
discriminant = square(b) - 4 * a * c
x1 = (-b + sqrt(discriminant)) / (2 * a)
```

More Naming Tips

- Names can be long if they help document your code:
  ```python
  average_age = average(age, students)
  ```
  is preferable to
  ```python
  # Compute average age of students
  aa = avg(a, st)
  ```

- 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
  f, g, h - Usually functions
  ```