Announcements

• Homework 1 due Tuesday 9/10 at 5pm; Late homework is not accepted!

• Quiz on Wednesday 9/11 released at 1pm, due Thursday 9/12 at 11:59pm
  
  • *Open-computer:* You can use the Python interpreter, watch course videos, and read the online text ([http://composingprograms.com](http://composingprograms.com)).

  • *No external resources:* Please don't search for answers, talk to your classmates, etc.

  • *Content Covered:* Lectures through last Friday 9/6; Same topics as Homework 1.

• Project 1 due next Thursday 9/19 at 11:59pm
Iteration Example
The Fibonacci Sequence

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

```python
def fib(n):
  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
```

Example: [link to Fibonacci sequence](http://goo.gl/vfymhd)
The Fibonacci Sequence

def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current

Example: [Link](http://goo.gl/vfymhd)
def fib(n):
    # Compute the nth Fibonacci number, for n >= 2.
    predecessor, current = 0, 1    # First two Fibonacci numbers
    k = 2      # Tracks which Fibonacci number is called current
    while k < n:
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, predecessor + current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)
        k = k + 1
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, predecessor + current
        k = k + 1
    return current

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987

Example: http://goo.gl/vfymhd
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)
        k = k + 1
    return current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)
        k = k + 1
    return current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1    # First two Fibonacci numbers
    k = 2                        # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, predecessor + current
        k = k + 1
    return current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

The next Fibonacci number is the sum of the current one and its predecessor.

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)

def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, predecessor + current
        k = k + 1
    return current
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)
        k = k + 1
    return current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)
        k = k + 1
    return current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
The Fibonacci Sequence

```python
def fib(n):
    """Compute the nth Fibonacci number, for n >= 2."""
    predecessor, current = 0, 1  # First two Fibonacci numbers
    k = 2  # Tracks which Fibonacci number is called current
    while k < n:
        predecessor, current = current, (predecessor + current)
        k = k + 1
    return current
```

Example: [http://goo.gl/vfymhd](http://goo.gl/vfymhd)
Discussion Question

Complete the following definition by placing an expression in _________________.

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in _________________.

```python
def choose(total, selection):
```

Example: [http://goo.gl/38ch30](http://goo.gl/38ch30)
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.
```
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    \[
    \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 2 \cdot 1}
    \]
```

Example: http://goo.gl/38ch3o
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10

Example: http://goo.gl/38ch3o
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

Discussion Question

Complete the following definition by placing an expression in ________________.

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ____________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
```
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
# Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    
    choose(n, k) is typically defined in math as:  $\frac{n!}{(n-k)! / k!}$

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * ________________, total - 1
    return ways
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
Discussion Question

Complete the following definition by placing an expression in _________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    """

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    """

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * __________, total - 1
    return ways
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: http://goo.gl/38ch3o
Discussion Question

Complete the following definition by placing an expression in ________________.

```python
def choose(total, selection):
    """Return the number of ways to choose SELECTION items from TOTAL.

    choose(n, k) is typically defined in math as:  n! / (n-k)! / k!
    
    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """

    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

Example: [http://goo.gl/38ch3o](http://goo.gl/38ch3o)
Default Arguments

(Demo)
Designing Functions
Characteristics of Functions
Characteristics of Functions

A function's *domain* is the set of all inputs it might possibly take as arguments.
Characteristics of Functions

A function's **domain** is the set of all inputs it might possibly take as arguments.

A function's **range** is the set of output values it might possibly return.
Characteristics of Functions

A function's *domain* is the set of all inputs it might possibly take as arguments.

A function's *range* is the set of output values it might possibly return.

A pure function's *behavior* is the relationship it creates between input and output.
A function's **domain** is the set of all inputs it might possibly take as arguments.

A function's **range** is the set of output values it might possibly return.

A pure function's **behavior** is the relationship it creates between input and output.
A function's **domain** is the set of all inputs it might possibly take as arguments.

A function's **range** is the set of output values it might possibly return.

A pure function's **behavior** is the relationship it creates between input and output.

```python
def square(x):
    """Return X * X."""

def choose(n, d):
    """Return the number of ways to choose D of N items."""
```
A function's **domain** is the set of all inputs it might possibly take as arguments.

A function's **range** is the set of output values it might possibly return.

A pure function's **behavior** is the relationship it creates between input and output.
A function's **domain** is the set of all inputs it might possibly take as arguments.

```python
def square(x):
    """Return X * X."""
```

A function's **range** is the set of output values it might possibly return.

```python
def choose(n, d):
    """Return the number of ways to choose D of N items."""
```

A pure function's **behavior** is the relationship it creates between input and output.

$x$ is a number

$n$ and $d$ are positive integers with $n$ greater than or equal to $d.$
### Characteristics of Functions

A function's **domain** is the set of all inputs it might possibly take as arguments.

**def square(x):**

"""Return X * X."""

**x** is a number

A function's **range** is the set of output values it might possibly return.

**def choose(n, d):**

"""Return the number of ways to choose D of N items."""

**n and d** are positive integers with **n** greater than or equal to **d**.

A pure function's **behavior** is the relationship it creates between input and output.
Characteristics of Functions

**def square(x):**

"""Return X * X."""

A function's **domain** is the set of all inputs it might possibly take as arguments.

\[ x \text{ is a number} \]

**def choose(n, d):**

"""Return the number of ways to choose D of N items."""

A function's **range** is the set of output values it might possibly return.

\[ n \text{ and } d \text{ are positive integers with } n \text{ greater than or equal to } d. \]

**return value is a positive integer**

A pure function's **behavior** is the relationship it creates between input and output.
Characteristics of Functions

A function's **domain** is the set of all inputs it might possibly take as arguments.

- **x is a number**

A function's **range** is the set of output values it might possibly return.

- **return value is a positive number**

A pure function's **behavior** is the relationship it creates between input and output.

- **return value is the square of the input**

---

```python
def square(x):
    """Return X * X."""

def choose(n, d):
    """Return the number of ways to choose D of N items."""
```

- **n and d are positive integers with n greater than or equal to d.**

- **return value is a positive integer**
A function's **domain** is the set of all inputs it might possibly take as arguments.

- **x** is a number

A function's **range** is the set of output values it might possibly return.

- return value is a positive number

A pure function's **behavior** is the relationship it creates between input and output.

- return value is the square of the input

---

```python
def square(x):
    """Return X * X."""

def choose(n, d):
    """Return the number of ways to choose D of N items."""
```

- `n` and `d` are positive integers with `n` greater than or equal to `d`.
- return value is a positive integer

- return value is the number of ways to choose `d` of `n` items.
A Guide to Designing Function
A Guide to Designing Function

Give each function exactly one job.
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.

Define functions generally.
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.

Define functions generally.
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.

Define functions generally.
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.

Define functions generally.
A Guide to Designing Function

Give each function exactly one job.

Don’t repeat yourself (DRY). Implement a process just once, but execute it many times.

Define functions generally.
Generalization
Generalizing Patterns with Arguments
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

**Shape:**

- Square
- Circle
- Hexagon
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape: 

Area: $r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape: 

Area: 

\[ r^2 \] 

\[ \pi \cdot r^2 \]
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

\[ r^2 \quad \pi \cdot r^2 \quad \frac{3\sqrt{3}}{2} \cdot r^2 \]
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

- Square: $r$
- Circle: $r$
- Hexagon: $r$

Area:

- Square: $1 \cdot r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

- $1 \cdot r^2$
- $\pi \cdot r^2$
- $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
- Square: $r$
- Circle: $r$
- Hexagon: $r$

Area:
- Square: $1 \cdot r^2$
- Circle: $\pi \cdot r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

\[ \frac{3\sqrt{3}}{2} \cdot r^2 \]
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
\[ r \]

Area:
\[ \frac{1}{2} \cdot r^2 \] \[ \pi \cdot r^2 \] \[ \frac{3\sqrt{3}}{2} \cdot r^2 \]

Finding common structure allows for shared implementation
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

\[
\text{Square: } r \quad \text{Circle: } r \quad \text{Hexagon: } r
\]

Area:

\[
\begin{align*}
\text{Square: } & \quad \frac{1}{1} \cdot r^2 \\
\text{Circle: } & \quad \pi \cdot r^2 \\
\text{Hexagon: } & \quad \frac{3\sqrt{3}}{2} \cdot r^2
\end{align*}
\]

Finding common structure allows for shared implementation

(Demo)
Higher-Order Functions
Generalizing Over Computational Processes
Generalizing Over Computational Processes

The common structure among functions may be a computational process, rather than a number.
**Generalizing Over Computational Processes**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

>>> summation(5, cube)
225
"""
Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

Function of a single argument (not called term)
Summation Example

```python
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total
```

```
>>> summation(5, cube)
225
"""
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence."

    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

# Local function definitions; returning functions

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

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

compose1(f, g):
    """Return a function that composes f and g.

    f, g -- functions of a single argument"

    def h(x):
        return f(g(x))
    return h

@main
def run():
    interact()
Summation Example

```python
def cube(k):
    return pow(k, 3)

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

```python
>>> summation(5, cube)
225
""
```

```python
# Local function definitions; returning functions
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."
    
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    """Return a function that composes f and g."
    
    def h(x):
        return f(g(x))
    return h
```

```python
@main
def run():
    interact()
```

- Function of a single argument (not called term)
- A formal parameter that will be bound to a function
- The cube function is passed as an argument value
- The function bound to term gets called here
Summation Example

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

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

>>> summation(5, cube)
225

"""
Functions as Return Values

(Demo)
Locally Defined Functions
Locally Defined Functions

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

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

```python
def make_adder(n):
    """Return a function that takes one argument k and returns k + n."
    
    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return k + n
    return adder
```
Functions defined \textit{within other function bodies} are bound to names in a \textit{local frame}.
Locally Defined Functions

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

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

```
>>> add_three = make_adder(3)
>>> add_three(4)
7
```

A local def statement

Can refer to names in the enclosing function
Call Expressions as Operator Expressions

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```
Call Expressions as Operator Expressions

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

make_adder(1)(2)
```
Call Expressions as Operator Expressions

Operator

make_adder(1) ( 2 )

def make_adder(n):
    def adder(k):
        return k + n
    return adder
### Call Expressions as Operator Expressions

<table>
<thead>
<tr>
<th>Operator</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>make_adder(1)</code></td>
<td><code>(2)</code></td>
</tr>
</tbody>
</table>

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

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

```
def make_adder(1):
    ( 2  )
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

Operator

An expression that evaluates to any value

Operand

make_adder(1) ( 2 )

```python
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

Operator

An expression that evaluates to any value

Operand

make_adder(1)     (         2         )

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

Call Expressions as Operator Expressions

<table>
<thead>
<tr>
<th>An expression that evaluates to a function</th>
<th>An expression that evaluates to any value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator</strong></td>
<td><strong>Operand</strong></td>
</tr>
<tr>
<td>make_adder(1)</td>
<td>(2)</td>
</tr>
</tbody>
</table>

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

```python
def make_adder(1):
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

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

make_adder(1) (2)

An expression that evaluates to any value

func make_adder(n)
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to any value

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

make_adder(1)
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to any value

Operator

Operand

make_adder(1) ( 2 )

func adder(k)
make_adder(1)

func make_adder(n)
1

def make_adder(n):
def adder(k):
    return k + n
return adder
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to any value

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to any value

Operator

make_adder(1)

Operand

func adder(k)

make_adder(1)

func make_adder(n)

make_adder(1) ( 2 )

3

2

2

def make_adder(n):
    def adder(k):
        return k + n
    return adder
The Purpose of Higher-Order Functions
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.
The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value.
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
The Purpose of Higher-Order Functions

Functions are first-class: Functions can be manipulated as values in our programming language.

Higher-order function: A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
The Purpose of Higher-Order Functions

**Functions are first-class:** Functions can be manipulated as values in our programming language.

**Higher-order function:** A function that takes a function as an argument value or returns a function as a return value

Higher-order functions:

- Express general methods of computation
- Remove repetition from programs
- Separate concerns among functions
The Game of Hog

(Demo)