Announcements
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!

  • Check your submission on ok.cs61a.org and submit again if it's not right
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on ok.cs61a.org and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on ok.cs61a.org and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  ▪ Open-computer, open notes, closed friends
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on [ok.cs61a.org](http://ok.cs61a.org) and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  ▪ Open-computer, open notes, closed friends
  ▪ Content Covered: Lectures through Monday 1/26 (same topics as Homework 1)
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on ok.cs61a.org and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  ▪ Open-computer, open notes, closed friends
  ▪ Content Covered: Lectures through Monday 1/26 (same topics as Homework 1)
  ▪ If you receive 0/3, talk to your TA (or me) about how to approach the course
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on ok.cs61a.org and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  ▪ Open-computer, open notes, closed friends
  ▪ Content Covered: Lectures through Monday 1/26 (same topics as Homework 1)
  ▪ If you receive 0/3, talk to your TA (or me) about how to approach the course
• Extra lectures: Earn 1 unit (pass/no pass) by learning about optional additional topics
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on ok.cs61a.org and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  ▪ Open-computer, open notes, closed friends
  ▪ Content Covered: Lectures through Monday 1/26 (same topics as Homework 1)
  ▪ If you receive 0/3, talk to your TA (or me) about how to approach the course
• Extra lectures: Earn 1 unit (pass/no pass) by learning about optional additional topics
  ▪ First extra lecture: Thursday 1/29 5–6:30pm in 2050 VLSB (Come there to learn more)
Announcements

• Homework 1 due Wednesday 1/28 at 11:59pm. Late homework is not accepted!
  ▪ Check your submission on ok.cs61a.org and submit again if it's not right
• Take-home quiz 1 released Wednesday 1/28, due Thursday 1/29 at 11:59pm
  ▪ Open-computer, open notes, closed friends
  ▪ Content Covered: Lectures through Monday 1/26 (same topics as Homework 1)
  ▪ If you receive 0/3, talk to your TA (or me) about how to approach the course
• Extra lectures: Earn 1 unit (pass/no pass) by learning about optional additional topics
  ▪ First extra lecture: Thursday 1/29 5–6:30pm in 2050 VLSB (Come there to learn more)
• Project 1 due Thursday 2/5 at 11:59pm
Iteration Example
The Fibonacci Sequence
The Fibonacci Sequence
The Fibonacci Sequence

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

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

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

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

```python
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1  # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

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

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

def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1  # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

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

```python
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1               # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr
```

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

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

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

The next Fibonacci number is the sum of the current one and its predecessor
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1  # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

The next Fibonacci number is the sum of the current one and its predecessor
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1               # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

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

The Fibonacci Sequence

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

```python
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1              # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

The next Fibonacci number is the sum of the current one and its predecessor
```
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    pred, curr = 0, 1  # Zeroth and first Fibonacci numbers
    k = 1  # curr is the kth Fibonacci number
    while k < n:
        pred, curr = curr, pred + curr
        k = k + 1
    return curr

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

The Fibonacci Sequence
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987
Discussion Question 1

What does pyramid compute?
Discussion Question 1

What does pyramid compute?

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
```
Discussion Question 1

What does pyramid compute?

def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
Discussion Question 1

What does pyramid compute?

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
```
Discussion Question 1

What does pyramid compute?

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
```

- $n^2$
- $(n+1)^2$
- $2 \cdot (n+1)$
- $n^2 + 1$
- $n \cdot (n + 1)$
Discussion Question 1

What does pyramid compute?

def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
Discussion Question 1

What does pyramid compute?

```
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
```
Discussion Question 1

What does pyramid compute?

def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
What does pyramid compute?

\[
\begin{align*}
&n^2 \\
&(n+1)^2 \\
&2 \cdot (n+1) \\
&n^2 + 1 \\
&n \cdot (n+1)
\end{align*}
\]

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
```
What does pyramid compute?

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
```
Discussion Question 1

What does pyramid compute?

def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total
Discussion Question 1

What does pyramid compute?

def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a+1, b-1
        total = total + a + b
    return total

\[
\begin{align*}
    n^2 & \\
    (n + 1)^2 & \\
    2 \cdot (n + 1) & \\
    n^2 + 1 & \\
    n \cdot (n + 1) &
\end{align*}
\]
Discussion Question 1

What does pyramid compute?

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a += 1
        b -= 1
        total = total + a + b
    return total
```

1. $n^2$
2. $(n + 1)^2$
3. $2 \cdot (n + 1)$
4. $n^2 + 1$
5. $n \cdot (n + 1)$
Discussion Question 1

What does pyramid compute?

```python
def pyramid(n):
    a, b, total = 0, n, 0
    while b:
        a, b = a + 1, b - 1
        total = total + a + b
    return total
```

$I'm$ still here
Designing 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.
Characteristics of Functions

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

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.
Characteristics of Functions

<table>
<thead>
<tr>
<th>def square(x):</th>
<th>def fib(n):</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;&quot;&quot;Return X * X.&quot;&quot;&quot;&quot;</td>
<td>&quot;&quot;&quot;&quot;Compute the nth Fibonacci number, for N &gt;= 1.&quot;&quot;&quot;&quot;</td>
</tr>
</tbody>
</table>

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.
Characteristics of Functions

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

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

```

```python
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""

x is a real number
```

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.
Characteristics of Functions

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

def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
```

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

- \( x \) is a real number

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

- Returns a non-negative real number

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

\begin{tabular}{|l|}
\hline
\textbf{def square(x):} & \textbf{def fib(n):} \\
\text{""""Return X * X.""""} & \text{""""Compute the nth Fibonacci number, for N >= 1.""""} \\
\hline
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. \\
\textit{x is a real number} & \textit{returns a non-negative real number} \\
\hline
A pure function's behavior is the relationship it creates between input and output. & \\
\textit{return value is the square of the input} & \\
\hline
\end{tabular}
Characteristics of Functions

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

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

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

\[ \text{returns a non-negative real number} \]

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

\[ \text{return value is the square of the input} \]

---

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

```
def fib(n):
    """Compute the nth Fibonacci number, for N >= 1."""
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

```
def x is a real number
```

\[ n \text{ is an integer greater than or equal to 1} \]
**Characteristics of Functions**

<table>
<thead>
<tr>
<th><strong>def square(x):</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;&quot;&quot;&quot;Return $x \times x$.&quot;&quot;&quot;&quot;</td>
</tr>
<tr>
<td><strong>def fib(n):</strong></td>
</tr>
<tr>
<td>&quot;&quot;&quot;&quot;Compute the nth Fibonacci number, for $n \geq 1$.&quot;&quot;&quot;&quot;</td>
</tr>
</tbody>
</table>

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

- $x$ is a real number
- $n$ is an integer greater than or equal to 1

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

- Returns a non-negative real number
- Returns a Fibonacci number

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

- Return value is the square of the input
Characteristics of Functions

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

- \( x \) is a real number

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

- Returns a non-negative real number

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

- Return value is the square of the input
- Return value is the \( n \)th Fibonacci number

---

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

    x is a real number

    Returns a non-negative real number

    Return value is the square of the input

---

def fib(n):
    """Compute the \( n \)th Fibonacci number, for \( N \geq 1 \)."""

    n is an integer greater than or equal to 1

    Returns a Fibonacci number

    Return value is the \( n \)th Fibonacci number
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:
Regular geometric shapes relate length and area.

Shape:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:
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:  
\[ r \]

Area:  
\[ r^2 \]
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

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

Area:  
- Square: $r^2$
- Circle: $\pi \cdot r^2$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

- Square: $r^2$
- 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:

\[ r \cdot r^2 \]

\[ \pi \cdot r^2 \]

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

Regular geometric shapes relate length and area.

Shape:

Area:

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

Regular geometric shapes relate length and area.

Shape:

Area: $\begin{align*}
\text{1}\cdot r^2
\end{align*}$

$\begin{align*}
\pi\cdot r^2
\end{align*}$

$\begin{align*}
\frac{3\sqrt{3}}{2}\cdot r^2
\end{align*}$
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

- Square: $r^2$
- Circle: $\pi r^2$
- Hexagon: $\frac{3\sqrt{3}}{2} r^2$

Area:
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

Finding common structure allows for shared implementation
Generalizing Patterns with Arguments

Regular geometric shapes relate length and area.

Shape:

Area:

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)
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
"""
```

# 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.
    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

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

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
"""
```
## 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
'""'
```

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

Function of a single argument (not called "term")

A formal parameter that will be bound to a function

The function bound to term gets called here
def cube(k):
    return pow(k, 3)

def summation(n, term):
    """Sum the first n terms of a sequence.\n    >>> summation(5, cube)
    225
    """
    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.\n    >>> add_three = make_adder(3)
    >>> add_three(4)
    7\n    """
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    """Return a function that composes f and g.\n    f, g −− functions of a single argument\n    """
    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
0 + 1 + 8 + 27 + 64 + 125
```

- 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
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

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
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.
    return lambda k: k + n

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

def adder(k):
    return k + n

return adder
```
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."
    def adder(k):
        return k + n
    return adder
```

A def statement within another def statement

The name add_three is bound to a function

A function that returns a function

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

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

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

The name `add_three` is bound to a function.

A def statement within another def statement.

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

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

\[ \text{make_adder}(1) \ (\ 2 \ ) \]
Call Expressions as Operator Expressions

\[
\begin{array}{c}
\text{Operator} \\
\text{make_adder}(1) \\
\end{array}
\begin{array}{c}
\text{Operand} \\
(2) \\
\end{array}
\]
Call Expressions as Operator Expressions

An expression that evaluates to a function

Operator

Operand

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

An expression that evaluates to a function

Operator

An expression that evaluates to its argument

Operand

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

- An expression that evaluates to a function
- An expression that evaluates to its argument

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

An expression that evaluates to a function

Operator

An expression that evaluates to its argument

Operand

make_adder(1) (2)

make_adder(1)
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

make_adder(1) ( 2 )

make_adder(1)

func make_adder(n)
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

make_adder(1) ( 2 )

func make_adder(n)

1
Call Expressions as Operator Expressions

An expression that evaluates to a function

make_adder( n )

An expression that evaluates to its argument

Operator

make_adder(1)     (         2         )

Operand

make_adder(1)

func make_adder(n)

make_adder( n ):

1
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

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

make_adder(1)
```

```
make_adder(1) ( 2 )
```

```
make_adder(1)
```

```
func make_adder(n)
```
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

\texttt{make_adder}(1) \quad ( \quad 2 \quad )

\texttt{func make_adder(n)}

\texttt{def adder(k):}
    \texttt{return k + n}
\texttt{return adder}
\texttt{func adder(k)}
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

operator
operand

make_adder(1) (2)

func adder(k)
make_adder(1)

func make_adder(n)

1

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

func adder(k)
Call Expressions as Operator Expressions

An expression that evaluates to a function

Operand

Operator

make_adder(1)     (         2         )

An expression that evaluates to its argument

func make_adder(n)

func adder(k)

make_adder(1) (2) 2

def adder(k):
    return k + n

func adder(k)
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

make_adder(1) (2)

func adder(k)

make_adder(1)

func make_adder(n)

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

→ func adder(k)
Call Expressions as Operator Expressions

An expression that evaluates to a function

An expression that evaluates to its argument

Operator

Operand

func make_adder(n)

func adder(k)

make_adder(1)

func adder(k):
  return k + n

def adder(k):
  return adder

1

3

2

2

16
Environments for Higher-Order Functions
Names can be Bound to Functional Arguments

```python
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```

Interactive Diagram
Names can be Bound to Functional Arguments

```
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```
Names can be Bound to Functional Arguments

1 def apply_twice(f, x):
2    return f(f(x))
3
4 def square(x):
5    return x * x
6
7 result = apply_twice(square, 2)

Applying a user-defined function:
• Create a new frame
• Bind formal parameters (f & x) to arguments
• Execute the body: return f(f(x))
Names can be Bound to Functional Arguments

```python
1 def apply_twice(f, x):
   return f(f(x))
2
3 def square(x):
   return x * x
4
5 result = apply_twice(square, 2)
```

**Interactive Diagram**

Applying a user-defined function:
- Create a new frame
- Bind formal parameters \((f \ & \ x)\) to arguments
- Execute the body:
  ```
  return f(f(x))
  ```
Names can be Bound to Functional Arguments

```python
1 def apply_twice(f, x):
2     return f(f(x))

4 def square(x):
5     return x * x

7 result = apply_twice(square, 2)
```

Applying a user-defined function:
- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))
Names can be Bound to Functional Arguments

```python
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

Interactive Diagram

Applying a user-defined function:
- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))
Discussion Question

What is the value of the final expression below? (Demo)
Discussion Question

What is the value of the final expression below? (Demo)

def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
Discussion Question

What is the value of the final expression below? (Demo)

```python
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

Interactive Diagram
Discussion Question

What is the value of the final expression below? (Demo)

```python
def repeat(f, x):
    while f(x) != x:
        x = f(x)
    return x

def g(y):
    return (y + 5) // 3

result = repeat(g, 5)
```

If you think there's an error