Announcements
Announcements

*Midterm 1 is on Monday 9/23 from 7pm to 9pm*
Announcements

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
Announcements

• Midterm 1 is on Monday 9/23 from 7pm to 9pm
  • 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  • HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
**Announcements**

- Midterm 1 is on Monday 9/23 from 7pm to 9pm
  - 2 review sessions on Saturday 9/21 2pm-4pm and 4pm-6pm in 1 Pimentel
  - HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  - Extra weekend office hours announced on Piazza
Announcements

• Midterm 1 is on Monday 9/23 from 7pm to 9pm
  • 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  • HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  • Extra weekend office hours announced on Piazza
  • Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!
Announcements

• Midterm 1 is on Monday 9/23 from 7pm to 9pm
  • 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  • HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  • Extra weekend office hours announced on Piazza
  • Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!
• No lab next week: Monday 9/23, Tuesday 9/24, or Wednesday 9/25
Announcements

• Midterm 1 is on Monday 9/23 from 7pm to 9pm
  • 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  • HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  • Extra weekend office hours announced on Piazza
  • Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!

• No lab next week: Monday 9/23, Tuesday 9/24, or Wednesday 9/25

• Homework 3 due Tuesday 10/1 @ 11:59pm
Announcements

• Midterm 1 is on Monday 9/23 from 7pm to 9pm
  • 2 review sessions on Saturday 9/21 2pm–4pm and 4pm–6pm in 1 Pimentel
  • HKN review session on Sunday 9/22 from 4pm to 7pm in 2050 Valley LSB
  • Extra weekend office hours announced on Piazza
  • Cannot attend? Fill out the conflict form by Friday 9/20 @ 11:59pm!
• No lab next week: Monday 9/23, Tuesday 9/24, or Wednesday 9/25
• Homework 3 due Tuesday 10/1 @ 11:59pm
• Optional Hog strategy contest ends Thursday 10/3 @ 11:59pm
Abstraction
Functional Abstractions
Functional Abstractions

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

def sum_squares(x, y):
    return square(x) + square(y)
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`?
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.
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
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.
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
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.
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
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`.  

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
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
Functional Abstractions

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

def square(x):
    return pow(x, 2)

def square(x):
    return mul(x, x-1) + x
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
Choosing Names

Names typically *don’t* matter for correctness

*but*

they matter a lot for composition
Choosing Names

Names typically don’t matter for correctness

but

they matter a lot for composition

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

Names typically don’t matter for correctness

but

they matter a lot for composition

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.
Choosing Names

Names typically don’t matter for correctness

*but*

they matter a lot for composition

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).
Choosing Names

Names typically *don’t* matter for correctness

**but**

they matter a lot for composition

---

From:  

To:

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).
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>
</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).
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>
</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).
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>play_helper</td>
<td>take_turn</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).
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>play_helper</td>
<td>take_turn</td>
</tr>
<tr>
<td>my_int</td>
<td>num_rolls</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).
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>play_helper</td>
<td>take_turn</td>
</tr>
<tr>
<td>my_int</td>
<td>num_rolls</td>
</tr>
<tr>
<td>l, I, O</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
Which Values Deserve a Name

Repeated compound expressions:
Which Values Deserve a Name

Repeated compound expressions:

```python
if sqrt(square(a) + square(b)) > 1:
    x = x + sqrt(square(a) + square(b))
```
Which Values Deserve a Name

Repeated compound expressions:

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

hypotenuse = sqrt(square(a) + square(b))
if hypotenuse > 1:
    x = x + hypotenuse
```
Which Values Deserve a 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
hypotenuse = sqrt(square(a) + square(b))
if hypotenuse > 1:
    x = x + hypotenuse
```
Which Values Deserve a Name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

\[
x = \left(-b + \sqrt{\text{square}(b) - 4 \times a \times c}\right) / (2 \times a)
\]
Which Values Deserve a Name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

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

discriminant = sqrt(square(b) - 4 * a * c)
```
x = (-b + discriminant) / (2 * a)
```
Which Values Deserve a 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
x = (-b + sqrt(square(b) - 4 * a * c)) / (2 * a)
```

```python
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + discriminant) / (2 * a)
```
Which Values Deserve a Name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

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

discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + 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)
```
Repeated compound expressions:

\[
\begin{align*}
    \text{if } \sqrt{\text{square}(a) + \text{square}(b)} > 1: \\
    & x = x + \sqrt{\text{square}(a) + \text{square}(b)} \\
\end{align*}
\]

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

Meaningful parts of complex expressions:

\[
\begin{align*}
    x &= (-b + \sqrt{\text{square}(b) - 4 \times a \times c}) / (2 \times a) \\
\end{align*}
\]

\[
\begin{align*}
    \text{discriminant} &= \sqrt{\text{square}(b) - 4 \times a \times c} \\
    x &= (-b + \text{discriminant}) / (2 \times a)
\end{align*}
\]

More Naming Tips

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

\[
\begin{align*}
    \text{average}_.\text{age} &= \text{average}(\text{age, students}) \\
\end{align*}
\]

is preferable to

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

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

\[
\begin{align*}
    n, k, i - \text{Usually integers} \\
    x, y, z - \text{Usually real numbers} \\
    f, g, h - \text{Usually functions}
\end{align*}
\]
Which Values Deserve a Name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

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

```python
discriminant = sqrt(square(b) - 4 * a * c)
x = (-b + 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.

  - `n, k, i` — Usually integers
  - `x, y, z` — Usually real numbers
  - `f, g, h` — Usually functions
Testing
Test-Driven Development
Test-Driven Development

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

Run your code interactively.
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.

Run your code interactively.

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

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

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.

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.  (Demo)
Decorators
Function Decorators

(demo)
Function Decorators

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

(demo)
Function Decorators

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

Function Decorators

(de示)

Function decorator

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

Decorated function
Function Decorators

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

is identical to
Function Decorators

(demo)

Function decorator

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

Decorated function

is identical to

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

(demo)

Function decorator

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

Decorated function

is identical to

Why not just use this?

def triple(x):
    return 3 * x
triple = trace1(triple)
Review
What Would Python Print?
What Would Python Print?

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

This expression evaluates to and prints
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)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
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)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

print(5)
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)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>None</td>
</tr>
<tr>
<td>print(5)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>5</code></td>
<td><code>5</code></td>
<td></td>
</tr>
<tr>
<td><code>print(5)</code></td>
<td><code>None</code></td>
<td><code>5</code></td>
</tr>
</tbody>
</table>
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)

print(add(3, 4), print(5))
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Print?

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

```
from operator import add, mul

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

print(add(3, 4), print(5)) # 7
print(5) # 5
print(add(3, 4), print(5)) # 7
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>
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)

print(add(3, 4), print(5))

5
None
7

print(5)

5
None
5

print(add(3, 4), print(5))

7
None
5
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>7 None</td>
<td>5</td>
</tr>
</tbody>
</table>
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)

print(add(3, 4), print(5))
7 None

print(5)
5

print(add(3, 4), print(5))
7 None
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>7 None</td>
<td>5</td>
</tr>
</tbody>
</table>
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)

print(add(3, 4), print(5))
7 None

print(5)
5 None

print(add(3, 4), print(5))
7 None

print(5)
5 None
```

This expression | Evaluates to | And prints
--- | --- | ---
5 | 5 | 5
print(5) | None | 5
print(add(3, 4), print(5)) | None | 5
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)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 None</td>
</tr>
</tbody>
</table>
What Would Python Print?

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

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

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>None</td>
<td>7</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
What Would Python Print?

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

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

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

This expression | Evaluates to | And prints
---|---|---
5 | 5 | None
print(5) | None | 5
print(add(3, 4), print(5)) | None | 5 None
7 | 7 | None
delay(delay)()(6)() | 7 None | 7 None
```

This expression

Evaluates to

And prints
What Would Python Print?

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

```
from operator import add, mul

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

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

def delay(delay)()(6)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 7 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Names in nested def statements can refer to their enclosing scope.
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)

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

delay(delay)(6)()
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>None 5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>delay(delay)()()</td>
<td>7 None</td>
<td>7 None</td>
</tr>
</tbody>
</table>
**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)
```

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

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

Names in nested def statements can refer to their enclosing scope

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>delay(delay)()</td>
<td>None</td>
<td>7 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 7 None</td>
</tr>
<tr>
<td>delay(delay)()()6()</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>
``
What Would Python Print?

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

```
from operator import add, mul

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

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
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 7 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**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)

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

def g():
    return 4

def f():
    return delay(g())

print(f())
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>delay(delay)()()</td>
<td>7</td>
<td>7 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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)

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

def g():
    return arg

This expression                          | Evaluates to | And prints
---|---|---
5 | 5 |
print(5) | None | 5
print(add(3, 4), print(5)) | None | 5
    | 7 | None
(delay(delay)()(6)() | delayed

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

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

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

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

Names in nested def statements can refer to their enclosing scope

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 7 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td>delayed</td>
<td></td>
</tr>
</tbody>
</table>
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)

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

def g():
    return delay(delay)(6)()
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>None</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 7 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td>6</td>
<td>delayed</td>
</tr>
</tbody>
</table>
What Would Python Print?

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

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

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   And prints
5                                      5
print(5)                                None         5
print(add(3, 4), print(5))             None         5
                                           7 None
(delay(delay)())(6)()                   6             delayed delayed
print(delay(print)())(4))              None         5
                                           7 None
```

This expression Evaluates to And prints
5 5
print(5) None 5
print(add(3, 4), print(5)) None 5
                                       7 None
(delay(delay)())(6)() 6 delayed delayed
print(delay(print)())(4)) None 5
                                       7 None
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)

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

def g():
    return delay(6)

def print(x):
    print(x)

debug = delay(delay)(g)

print(delay(print)(4))
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>None</td>
</tr>
<tr>
<td>print(5)</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5 7 None</td>
</tr>
<tr>
<td>delay(delay)(g)()()</td>
<td>6</td>
<td>delayed</td>
</tr>
<tr>
<td>print(delay(print)(4))</td>
<td></td>
<td>delayed</td>
</tr>
</tbody>
</table>
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)

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

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>None</td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>None</td>
</tr>
<tr>
<td>delay(delay)(6)()()</td>
<td>6</td>
<td>delayed</td>
</tr>
<tr>
<td>print(delay(print)()(4))</td>
<td>delayed</td>
<td>4</td>
</tr>
</tbody>
</table>
What Would Python Print?

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

```
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

Names in nested def statements can refer to their enclosing scope

This expression | Evaluates to | And prints
---|---|---
5 | 5 | delayed
print(5) | None | 5
print(add(3, 4), print(5)) | None | 5 7 None
(delay(delay)()()()()) | 6 | delayed delayed
print(delay(print)()()()) | delayed | 4
None | None | None
### 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)

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

def g():
    return delay(delay)()(6)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>print(5)</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>print(add(3, 4), print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>7 None</td>
<td>7 None</td>
<td></td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td>6</td>
<td>delayed</td>
</tr>
<tr>
<td>print(delay(print)()())(4))</td>
<td>None</td>
<td>delayed</td>
</tr>
<tr>
<td>4 None</td>
<td>4 None</td>
<td></td>
</tr>
</tbody>
</table>

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

This expression Evaluates to And prints

Example: http://goo.gl/NdrVqr
### 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)

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print('matey')</code></td>
<td><code>matey</code></td>
<td></td>
</tr>
</tbody>
</table>

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
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)
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>13</td>
<td></td>
</tr>
</tbody>
</table>
What Would Python Print?

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

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

def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
    return plunder
```

```
This expression

add(pirate(3)(square)(4), 1)
```

Evaluates to
And prints

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
What Would Python Print?

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

```
from operator import add, mul
def square(x):
    return mul(x, x)
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A function that always returns the identity function

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
What Would Python Print?

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

```
from operator import add, mul

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

def pirate(arggg):
    print('matey')
    def plunder(arggg):
        return arggg
    return plunder
```

This expression  Evaluates to  And prints

```
add(pirate(3)(square)(4), 1)
```

A function that always returns the identity function

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr
What Would Python Print?

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

```
from operator import add, mul
def square(x):
    return mul(x, x)
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
    return plunder
```

This expression | Evaluates to | And prints
---|---|---
```
add(pirate(3)(square)(4), 1)
```
Matey

A function that always returns the identity function

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
### 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)

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder
```

A function that always returns the identity function

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>Matey</td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

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

def pirate(arggg):
    print('matey')
    return plunder

def plunder(arggg):
    return arggg

A function that always returns the identity function

add(pirate(3)(square)(4), 1)
```

This expression Evaluates to And prints

```
func square(x)
```

Matey

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
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)
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
return plunder
```

A function that always returns the identity function

```
add(pirate(3)(square)(4), 1)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>func square(x)</code></td>
<td>Matey</td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr
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)
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
return plunder

A function that always returns the identity function

This expression  |  Evaluates to  |  And prints
-----------------|----------------|-----------------------
add(pirate(3)(square)(4), 1)  |  16  |  Matey
```

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: http://goo.gl/NdrVqr
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)

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder

A function that always returns the identity function
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td><code>func square(x)</code></td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
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)
def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(pirate(3)(square)(4), 1)</code></td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td><code>func square(x)</code></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td><code>pirate(pirate(pirate))(5)(7)</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A function that always returns the identity function

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
What Would Python Print?

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

```
from operator import add, mul

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

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td>func square(x)</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A function that always returns the identity function

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

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

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder

A function that always returns the identity function

```

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
What Would Python Print?

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

```
from operator import add, mul

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

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg

# A function that always returns the identity function

A function that always returns the identity function
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td>func square(x)</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td></td>
<td>Matey</td>
</tr>
<tr>
<td>Identity function</td>
<td></td>
<td>Matey</td>
</tr>
</tbody>
</table>

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

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.
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)

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(pirate(3)(square)(4), 1)</code></td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td><code>func square(x)</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td><code>pirate(pirate(pirate))(5)(7)</code></td>
<td></td>
<td>Matey</td>
</tr>
</tbody>
</table>

A function that always returns the identity function:

```
def pirate(arggg):
    print('matey')
```

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdrVqr](http://goo.gl/NdrVqr)
What Would Python Print?

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

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

def pirate(arggg):
    print('matey')
def plunder(arggg):
    return arggg
    return plunder
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td>func square(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td></td>
<td>Matey Matey</td>
</tr>
<tr>
<td>Identity function</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

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

def pirate(arggg):
    print('matey')

def plunder(arggg):
    return arggg

return plunder
```

A function that always returns the identity function

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>And prints</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey</td>
</tr>
<tr>
<td>func square(x)</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td>Error</td>
<td>Matey</td>
</tr>
<tr>
<td>Identity function</td>
<td>5</td>
<td>Matey</td>
</tr>
</tbody>
</table>

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Example: [http://goo.gl/NdVqr](http://goo.gl/NdVqr)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```python
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

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

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

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

mask = lambda horse: horse(2)
horse(mask)
```python
def horse(mask):
    horse = mask
    return horse

def mask(horse):
    return horse(mask)

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

Return Value

Return Value

Return Value

Return Value
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

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

def mask(horse):
    return horse

return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
def horse(mask):
    horse = mask
    return horse

def mask(horse):
    return horse(mask)

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

def mask(horse):
    return horse

return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return mask(horse)

mask = lambda horse: horse(2)

horse(mask)
def horse(mask):
    horse = mask
def mask(horse):
    return horse
return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return mask(horse)

def horse(mask):
    horse = mask
    return horse

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse

def horse(mask):
    return horse(mask)

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

def mask(horse):
    return horse

return horse(mask)

mask = lambda horse: horse(2)

horse(mask)
```python
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse

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

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse

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

```
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse

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

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

def mask(horse):
    return horse

return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    return horse

def mask(horse):
    return horse(mask)

mask = lambda horse: horse(2)
horse(mask)
def horse(mask):
    horse = mask
    def mask(horse):
        return horse
    return horse(mask)

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