Announcements

• Midterm 1 is on Monday 9/22 from 7pm to 9pm
Announcements

• Midterm 1 is on Monday 9/22 from 7pm to 9pm
  • 2 review sessions on Saturday 9/20 3pm–4:30pm and 4:30pm–6pm in 1 Pimentel
**Announcements**

- Midterm 1 is on Monday 9/22 from 7pm to 9pm
  - 2 review sessions on Saturday 9/20 3pm–4:30pm and 4:30pm–6pm in 1 Pimentel
  - HKN review session on Sunday 9/21 from 12pm to 3pm in 2060 Valley LSB
Announcements

• Midterm 1 is on Monday 9/22 from 7pm to 9pm
  ▪ 2 review sessions on Saturday 9/20 3pm–4:30pm and 4:30pm–6pm in 1 Pimentel
  ▪ HKN review session on Sunday 9/21 from 12pm to 3pm in 2060 Valley LSB
• No lecture on Monday
Announcements

• Midterm 1 is on Monday 9/22 from 7pm to 9pm
  ▪ 2 review sessions on Saturday 9/20 3pm–4:30pm and 4:30pm–6pm in 1 Pimentel
  ▪ HKN review session on Sunday 9/21 from 12pm to 3pm in 2060 Valley LSB
• No lecture on Monday
• No lab or office hours next week: Tuesday 9/23, or Wednesday 9/24
Announcements

• Midterm 1 is on Monday 9/22 from 7pm to 9pm
  ▪ 2 review sessions on Saturday 9/20 3pm–4:30pm and 4:30pm–6pm in 1 Pimentel
  ▪ HKN review session on Sunday 9/21 from 12pm to 3pm in 2060 Valley LSB
• No lecture on Monday
• No lab or office hours next week: Tuesday 9/23, or Wednesday 9/24
• Optional Hog strategy contest ends Wednesday 10/1 @ 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)
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

```
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
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.  
- Square has the intrinsic name `square`.  

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
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.

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
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.
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)
```
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
```
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
```

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.
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.
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

<table>
<thead>
<tr>
<th>From:</th>
<th>To:</th>
</tr>
</thead>
</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>
</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 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, 0</td>
<td>k, i, m</td>
</tr>
</tbody>
</table>

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

The type of value bound to the name is best documented in a function's docstring.

Function names typically convey their effect (print), their behavior (triple), or the value returned (abs).
Which Values Deserve a Name

Reasons to add a new name
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

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

Reasons to add a new name

Repeated compound expressions:

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

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

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

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

Reasons to add a new name

Repeated compound expressions:

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:

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

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

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

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

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

```python
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)
  ```
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

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:

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:

    average_age = average(age, students)

    is preferable to

    # Compute average age of students
    aa = avg(a, st)
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

```python
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.

  ```python
  n, k, i - Usually integers
  x, y, z - Usually real numbers
  f, g, h - Usually functions
  ```
Which Values Deserve a Name

Reasons to add a new name

Repeated compound expressions:

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

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

Meaningful parts of complex expressions:

```python
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.
  ```python
  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.

Bonus idea: 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.

Bonus idea: 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.

Bonus idea: Run your code interactively.

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

Interactive sessions can become doctests. Just copy and paste.
Test-Driven Development

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

A test will clarify the domain, range, & behavior of a function.

Tests can help identify tricky edge cases.

Develop incrementally and test each piece before moving on.

You can't depend upon code that hasn't been tested.

Run your old tests again after you make new changes.

Bonus idea: Run your code interactively.

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

Interactive sessions can become doctests. Just copy and paste.
Decorators
Function Decorators

(Demo)
Function Decorators

(Demo)

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

(Demo)

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

(Demo)

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

Function decorator

Decorated function
Function Decorators

(Demo)

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

is identical to
Function Decorators

(Demo)

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

is identical to

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

(Demo)

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

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from operator import add, mul</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>def square(x):</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>    return mul(x, x)</code></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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</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>Interactive Output</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)

print(5)
print(5)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>print(5)</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)
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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>
</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(print(5))
print(5)

This expression | Evaluates to | Interactive Output
-----------------|--------------|----------------------
5                | 5            | 5                    
print(5)         | None         | 5                    
print(print(5))  | 5            | 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)

print(print(5))
None

print(5)
5
None

print(print(5))
None
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</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.

```python
from operator import add, mul

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

print(print(5))  # None
5
5

This expression | Evaluates to | Interactive Output
--- | --- | ---
5 | 5 | 5
print(5) | None | 5
print(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)

print(print(5))  # None
print(5)         # None
print(print(5))  # None

This expression | Evaluates to | Interactive Output
-----------------|--------------|---------------------
5                | 5            | 5                   
print(5)         | None         | 5                   
print(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)

print(5)  # None
print(5)  # 5 5 None
print(print(5))  # None

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(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.

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

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5</td>
</tr>
</tbody>
</table>

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

def g():
    return arg

return g
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>delay(delay)()(6)()</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>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5 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.

```
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
```

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

```
def print(5):
    None
```

Names in nested def statements can refer to their enclosing scope

```
def print(print(5)):
    None
```

This expression | Evaluates to | Interactive Output
--- | --- | ---
5 | 5 | 5
print(5) | None | 5
print(print(5)) | None | 5 None
delay(delay)()(6)() | | None
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

def g():
    return arg
```

This expression | Evaluates to | Interactive Output
---|---|---
5 | 5 | 5
```
print(5)
```

None | None | 5 None
```
print(print(5))
```

None | None | 5 None
```
delay(delay)()(6)
```

None | None | 5 None
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

```
print(print(5))
```

This expression | Evaluates to | Interactive Output
--- | --- | ---
5 | 5 | 5
print(5) | None | 5
print(print(5)) | None | 5
```

This expression Evaluates to Interactive Output
--- --- ---
delay(delay)()(6)() | None | 5 None
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

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>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5 None</td>
</tr>
<tr>
<td>delay(delay)()(6)()</td>
<td>None</td>
<td>5</td>
</tr>
</tbody>
</table>

This expression Evaluates to Interactive Output

None

None

5

5

5 None
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
```

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

```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

```
print(5)
```

This expression | Evaluates to | Interactive Output
--- | --- | ---
5 | 5 | 5
print(5) | None | 5
print(print(5)) | None | 5 None
(delay(delay)()(6))() | | 
| | |
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
return

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5 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.

```
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

def g():
    return arg

This expression  | Evaluates to  | Interactive Output
5               | 5             | 5
print(5)        | None          | 5 None
print(print(5)) | None          | 5 None
delay(delay)(6)()   | None          | delayed delayed
```
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 g():
    return
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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>(delay(delay))()()()()</td>
<td>None</td>
<td>delayed delayed delayed</td>
</tr>
<tr>
<td>(delay(delay))()(6)()()</td>
<td></td>
<td>delayed delayed 6</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.

\[
\text{from operator import add, mul} \\
\text{def square(x):} \\
\quad \text{return mul}(x, x)
\]

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

\[
\text{def delay(arg):} \\
\quad \text{print('delayed')} \\
\quad \text{def g():} \\
\quad \quad \text{return arg} \\
\quad \text{return g}
\]

Names in nested def statements can refer to their enclosing scope

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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 None</td>
</tr>
<tr>
<td>print(print(5))</td>
<td>None</td>
<td>5 None</td>
</tr>
<tr>
<td>delay(delay)();(6)()()</td>
<td>6</td>
<td>delayed delayed 6</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 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>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5 ( None )</td>
</tr>
<tr>
<td>delay(delay)()()</td>
<td>6</td>
<td>delayed</td>
</tr>
<tr>
<td>delay(delay)()() print(delay(print)())(4))</td>
<td>6</td>
<td>6</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
```

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

```python
from operator import add, mul

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

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5 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>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)

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

def g():
    return
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 | Interactive Output
---|---|---
5 | 5 | 5
print(5) | None | 5
print(print(5)) | None | 5
\(\text{delay}()\)(6)() \(\text{delay}(\text{delay})()\)(()) | 6 | delayed
\(\text{delay}(\text{delay})()\)(6)() | None | 6
\(\text{delay}(\text{delay})()\)(())(4) | 5 | delayed
```

None
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')
    return arg
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 | Interactive Output
---|---|---
5 | 5 | 5
print(5) | None | 5
print(print(5)) | None | 5
None | None | None
delay(delay)()(6)() | 6 | delayed 6
print(delay(print)()(4)) | delayed 4 | None
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')
    return
return
```

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

```
def g():
    return arg
return g
```

Names in nested def statements can refer to their enclosing scope

```
print(print(5))
print(delay(print)(())(4))
print(delay(delay)(())(6)())
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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(print(5))</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>delayed</td>
<td>delayed</td>
</tr>
<tr>
<td>4</td>
<td>delayed</td>
<td>delayed</td>
</tr>
<tr>
<td>None</td>
<td>None</td>
<td>4</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
from operator import add, mul

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

This expression: `square(13)` Evaluates to: `169` Interactive Output: 169
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 | Interactive Output
--- | --- | ---

13

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.

```
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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</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.

<table>
<thead>
<tr>
<th>from operator import add, mul</th>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>def square(x):</td>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>return mul(x, x)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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.

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

A function that always returns the identity function

```
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 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.

```
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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add(pirate(3)(square)(4), 1)</code></td>
<td>13</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.
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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td>Matey 17</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.
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

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

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td>Matey</td>
</tr>
<tr>
<td></td>
<td></td>
<td>17</td>
</tr>
</tbody>
</table>

Interactive Output

Matey

17
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

A function that always returns the identity function
```

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td></td>
<td>Matey 17</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.
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.
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

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.
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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey 17</td>
</tr>
<tr>
<td>func square(x)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16</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.
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

# A function that always returns the identity function
pirate(pirate(pirate))(5)(7)
```

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

---

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey 17</td>
</tr>
</tbody>
</table>

```
func square(x)

16
```

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.

```
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.

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</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>
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
```

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.
```

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

<table>
<thead>
<tr>
<th>This expression</th>
<th>Evaluates to</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey 17</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 Matey Error</td>
</tr>
</tbody>
</table>

Identity function
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>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(pirate(3)(square)(4), 1)</td>
<td>17</td>
<td>Matey 17</td>
</tr>
<tr>
<td>func square(x)</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>pirate(pirate(pirate))(5)(7)</td>
<td>Identity function</td>
<td>Matey Matey Error</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.
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
```

A function that always returns the identity function

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

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

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

A function that always returns the identity function

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

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

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

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

mask = lambda horse: horse(2)

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

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

```
```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)

mask = lambda horse: horse(2)

horse(mask)