61A Lecture 3

Friday, September 6

Announcements

Announcements

- Homework 1 is due next Tuesday at 5pm (no email when you submit).
  - Homework is graded for effort.

Announcements

- Homework 1 is due next Tuesday at 5pm (no email when you submit).
  - Homework is graded for effort.
- Take-home quiz released next Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  - 3 points, graded for correctness.
  - Similar in format to a homework assignment.
  - If you receive 0/3, you will need to talk to the course staff or be dropped.

Announcements

- Homework 1 is due next Tuesday at 5pm (no email when you submit).
  - Homework is graded for effort.
- Take-home quiz released next Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  - 3 points, graded for correctness.
  - Similar in format to a homework assignment.
  - If you receive 0/3, you will need to talk to the course staff or be dropped.
  - *Open-computer*: You can use the Python interpreter, watch course videos, and read the online text (http://composingprograms.com).
  - *No external resources*: Please don't search for answers, talk to your classmates, etc.

Announcements

- Homework 1 is due next Tuesday at 5pm (no email when you submit).
  - Homework is graded for effort.
- Take-home quiz released next Wednesday 9/11 at 1pm, due Thursday 9/12 at 11:59pm.
  - 3 points, graded for correctness.
  - Similar in format to a homework assignment.
  - If you receive 0/3, you will need to talk to the course staff or be dropped.
  - *Open-computer*: You can use the Python interpreter, watch course videos, and read the online text (http://composingprograms.com).
  - *No external resources*: Please don't search for answers, talk to your classmates, etc.
- Project 1 posted this Friday, due Thursday 9/19 at 11:59pm.
  - Demo during next lecture

Multiple Environments

Life Cycle of a User-Defined Function

What happens?

Def statement:

Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:

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

Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:

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

Def statement

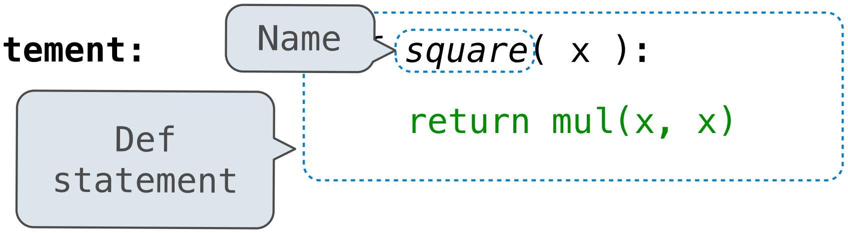
Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



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

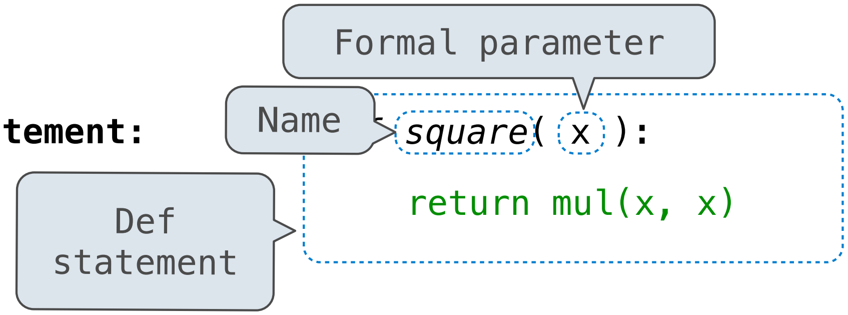
Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



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

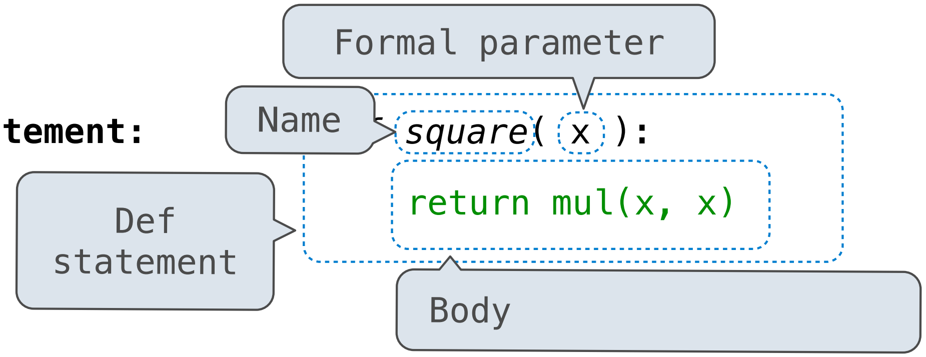
Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



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

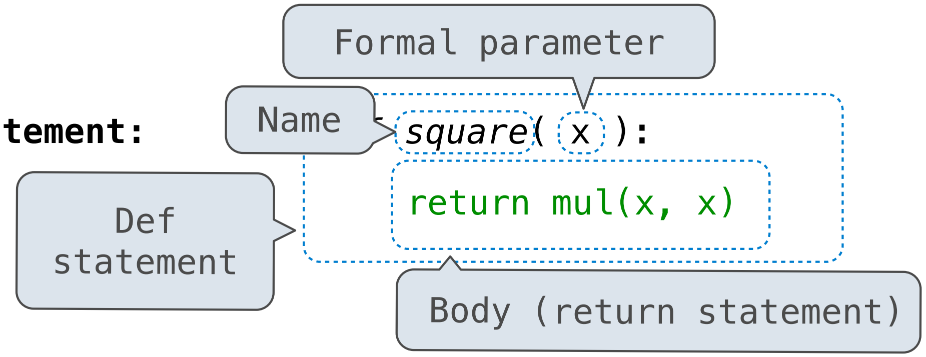
Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



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

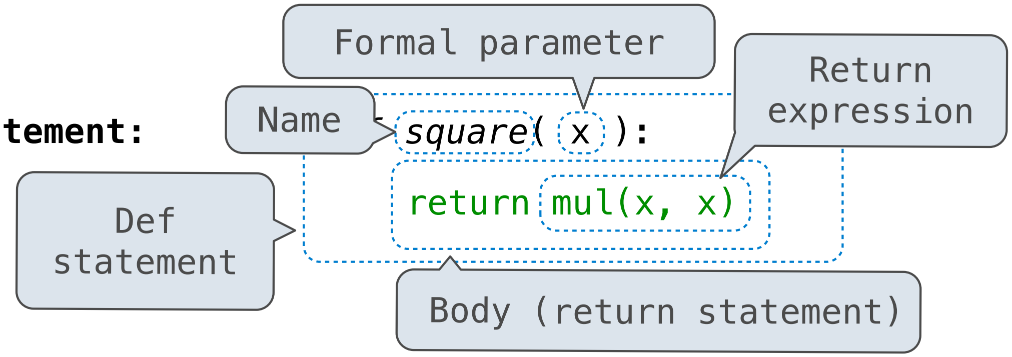
Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



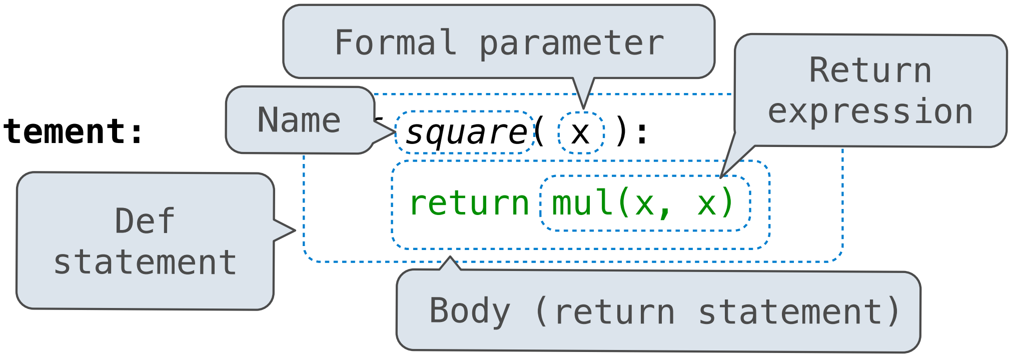
Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



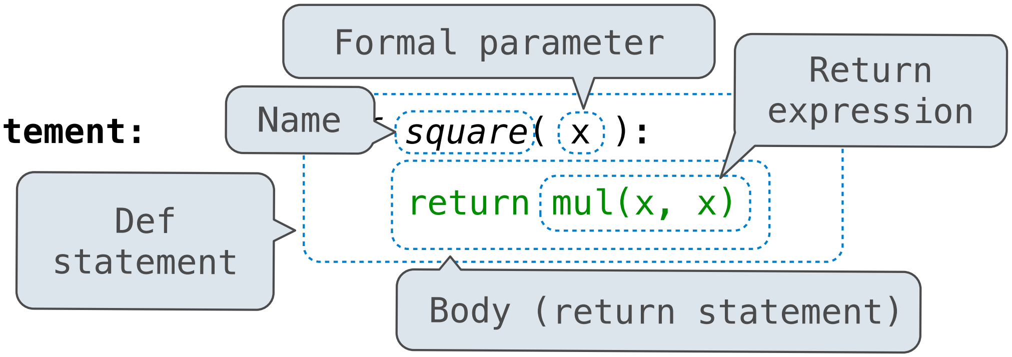
A new function is created!

Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

Def statement:



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

What happens?

A new function is created!

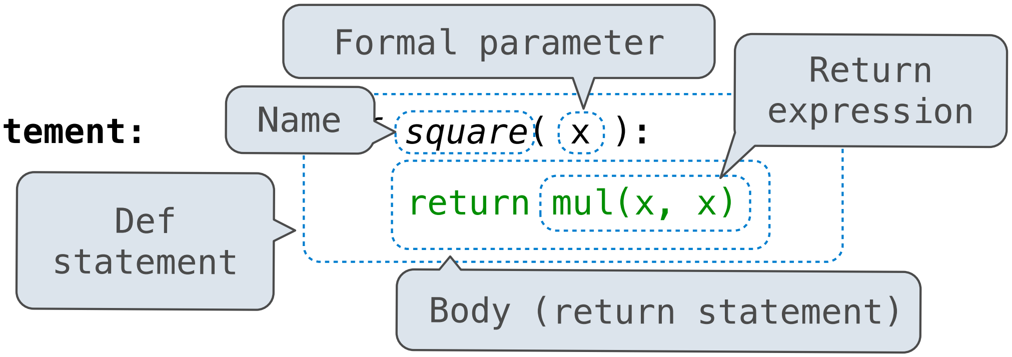
Name bound to that function in the current frame

Call expression:

Calling/Applying:

Life Cycle of a User-Defined Function

Def statement:



Formal parameter

Name

Return expression

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

Def statement

Body (return statement)

What happens?

A new function is created!

Name bound to that function in the current frame

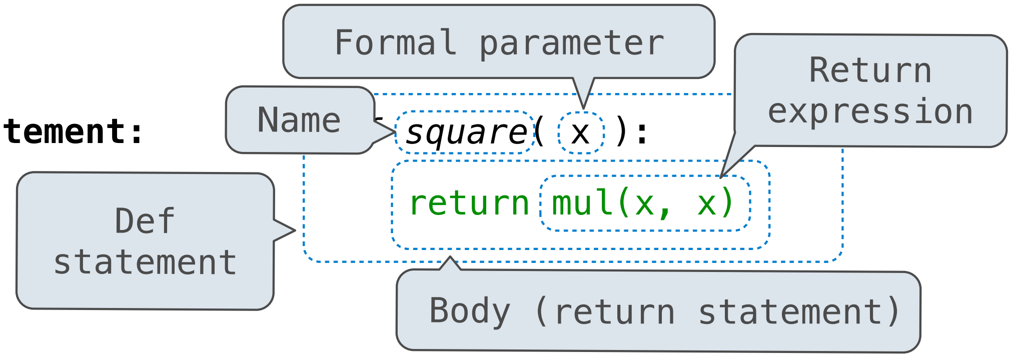
Call expression:

```
square(2+2)
```

Calling/Applying:

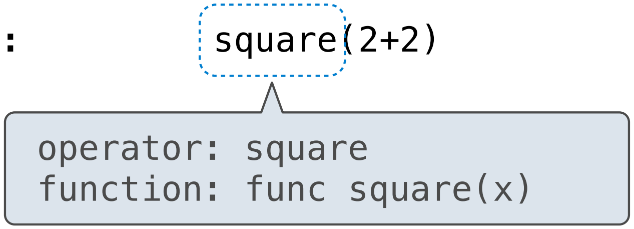
Life Cycle of a User-Defined Function

Def statement:



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

Call expression:



```
square(2+2)
```

```
operator: square
function: func square(x)
```

Calling/Applying:

What happens?

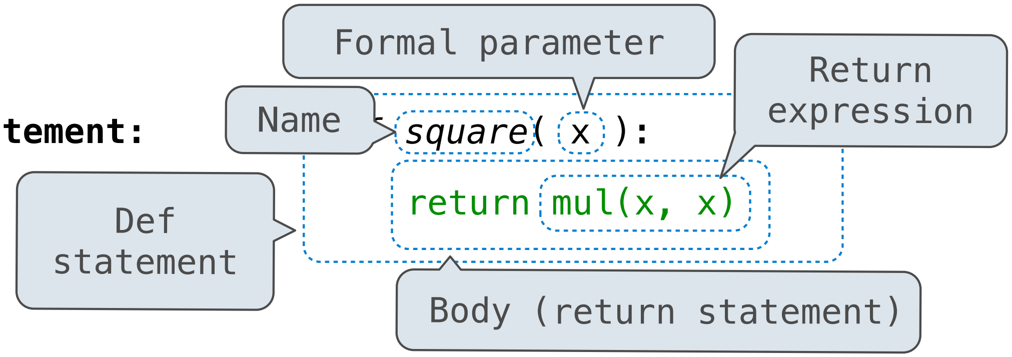
A new function is created!

Name bound to that function in the current frame

Life Cycle of a User-Defined Function

What happens?

Def statement:

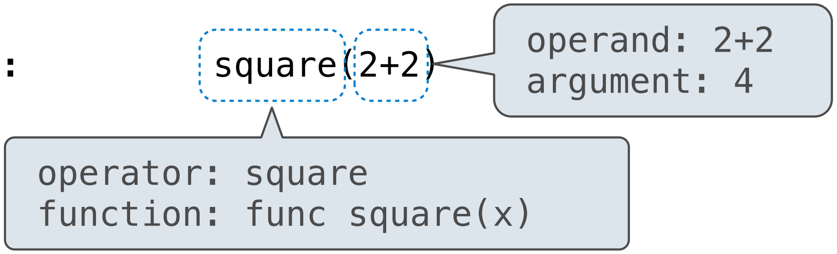


```
square( x ):
    return mul(x, x)
```

A new function is created!

Name bound to that function in the current frame

Call expression:



```
square(2+2)
```

operand: 2+2
argument: 4

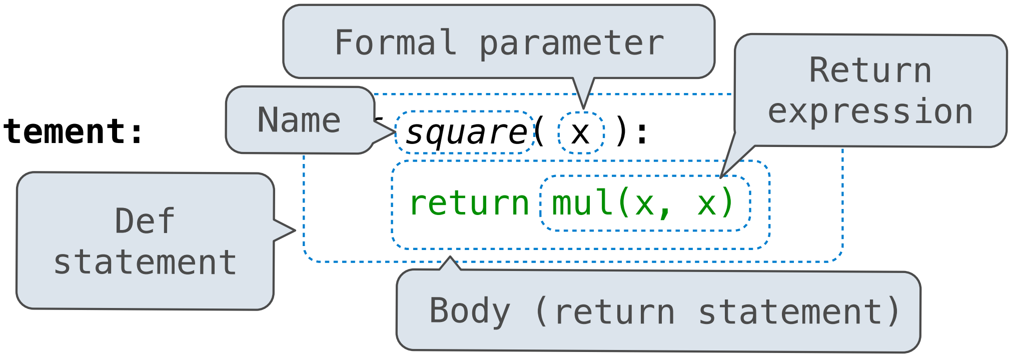
operator: square
function: func square(x)

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



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

Formal parameter

Name

Return expression

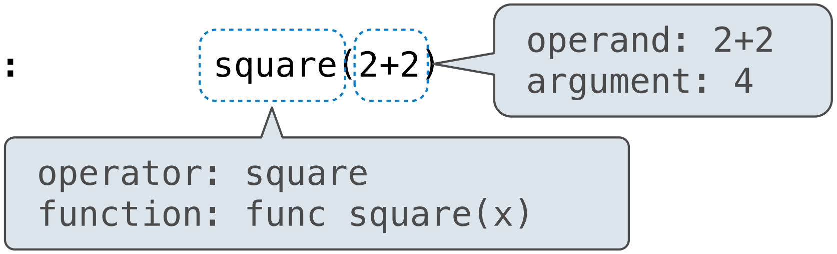
Def statement

Body (return statement)

A new function is created!

Name bound to that function in the current frame

Call expression:



```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

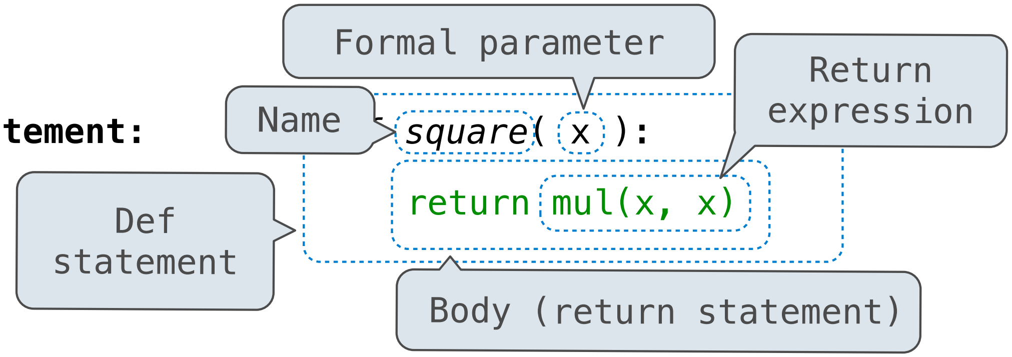
Operator & operands evaluated

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

Def statement:



Formal parameter

Name

Return expression

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

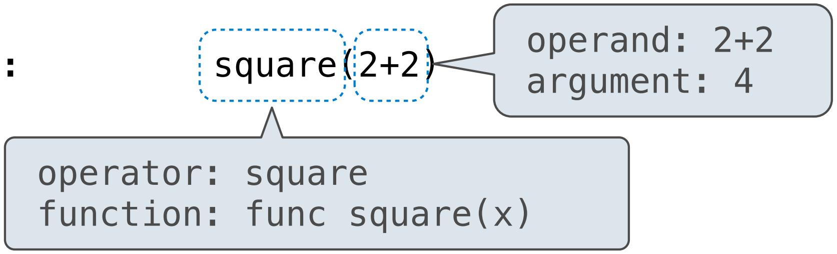
Def statement

Body (return statement)

A new function is created!

Name bound to that function in the current frame

Call expression:



```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

Operator & operands evaluated

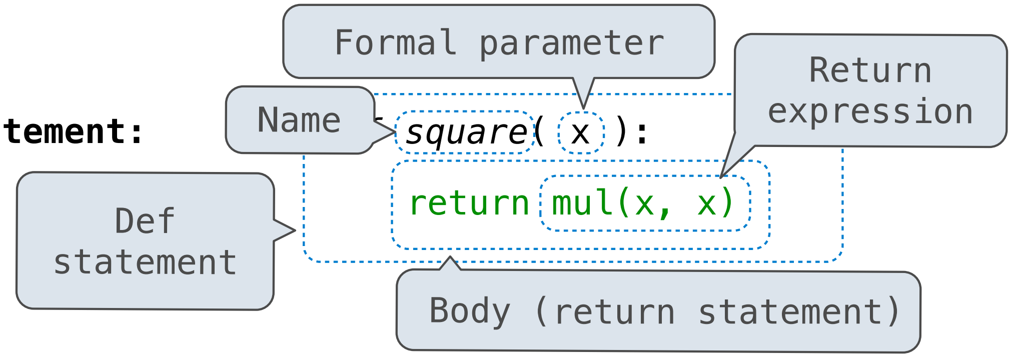
Function (value of operator) called on arguments (values of operands)

Calling/Applying:

Life Cycle of a User-Defined Function

What happens?

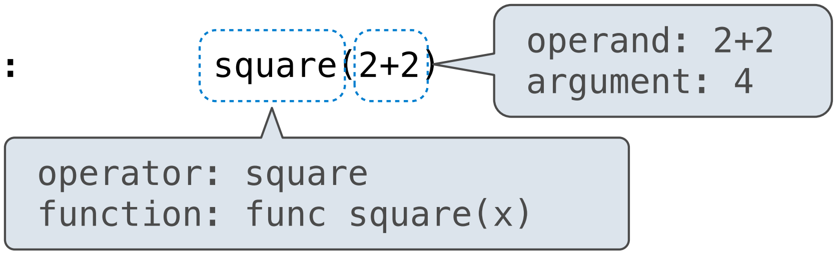
Def statement:



A new function is created!

Name bound to that function
in the current frame

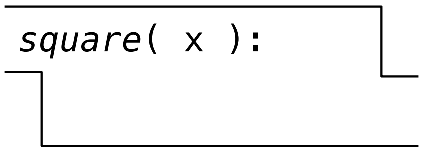
Call expression:



Operator & operands evaluated

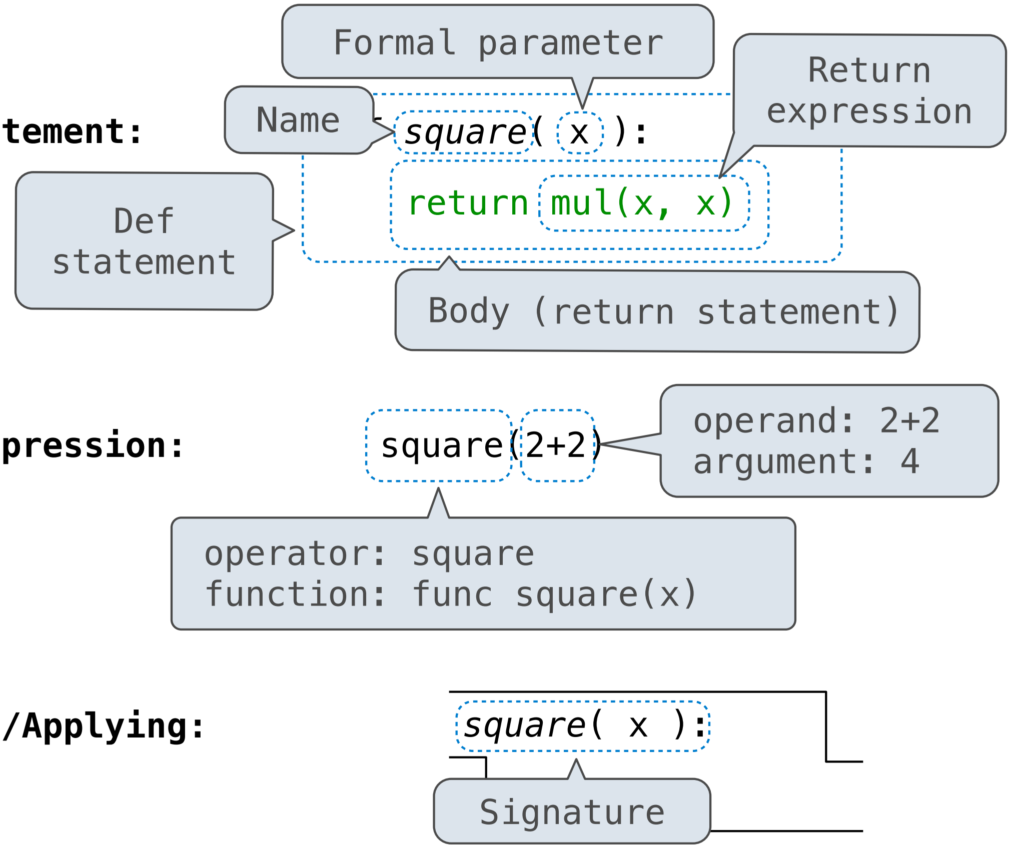
Function (value of operator)
called on arguments
(values of operands)

Calling/Applying:



Life Cycle of a User-Defined Function

What happens?



Def statement:

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

Formal parameter

Name

Return expression

Def statement

Body (return statement)

A new function is created!

Name bound to that function in the current frame

Call expression:

```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

Calling/Applying:

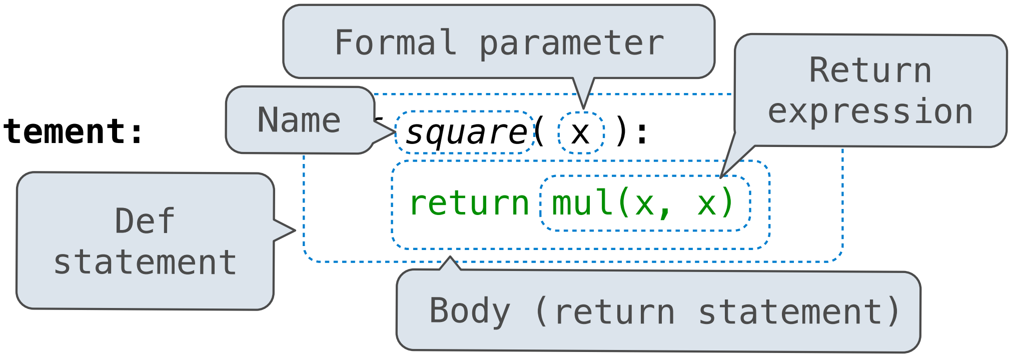
```
square( x ):
```

Signature

Life Cycle of a User-Defined Function

What happens?

Def statement:



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

Formal parameter

Name

Return expression

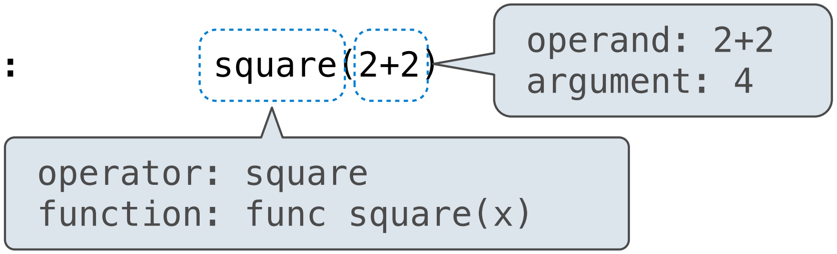
Def statement

Body (return statement)

A new function is created!

Name bound to that function in the current frame

Call expression:



```
square(2+2)
```

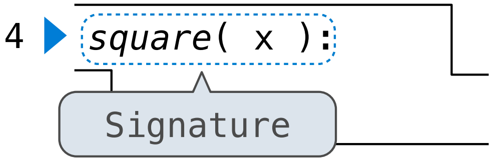
operand: 2+2
argument: 4

operator: square
function: func square(x)

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

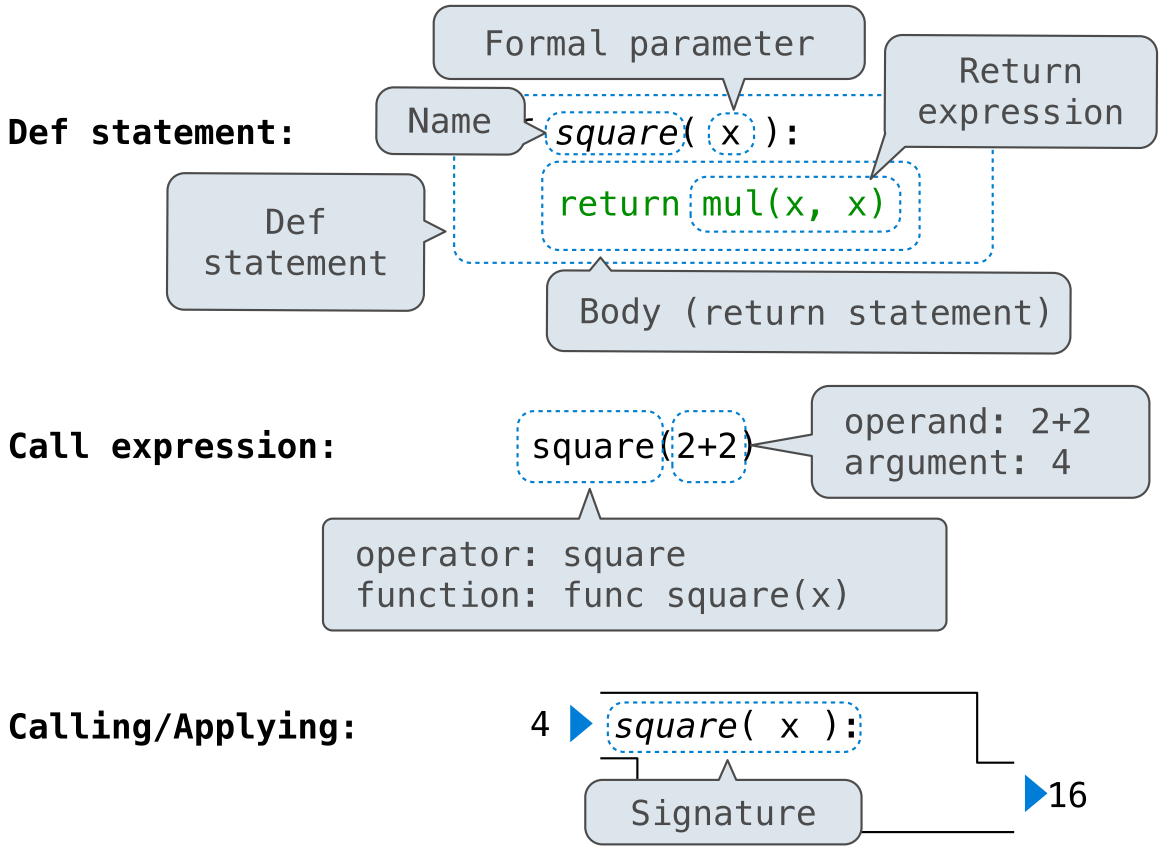
Calling/Applying:



4 ▶ *square*(x):

Signature

Life Cycle of a User-Defined Function



Def statement:

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

Call expression:

```
square(2+2)
```

Calling/Applying:

What happens?

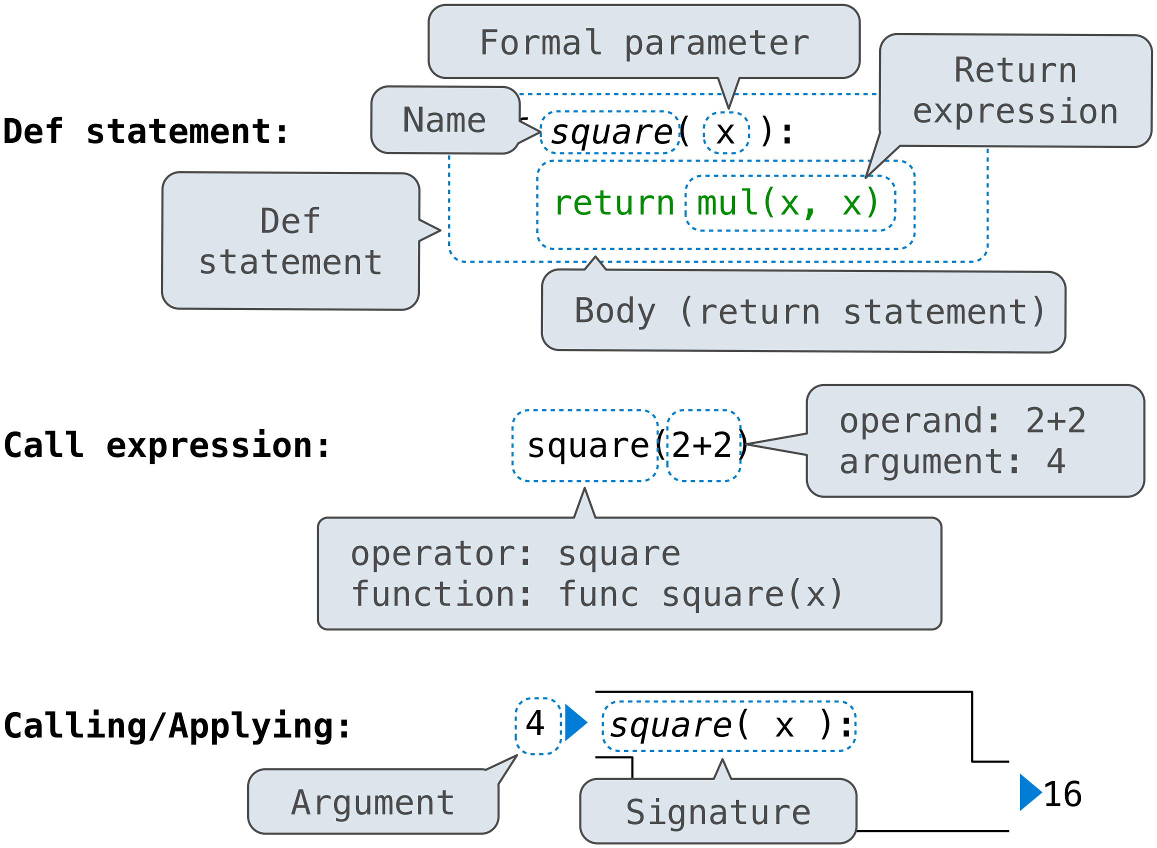
A new function is created!

Name bound to that function
in the current frame

Operator & operands evaluated

Function (value of operator)
called on arguments
(values of operands)

Life Cycle of a User-Defined Function



Def statement:

Formal parameter

Name

Return expression

Def statement

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

Body (return statement)

Call expression:

```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

Calling/Applying:

4

```
square( x ):
```

16

Argument

Signature

What happens?

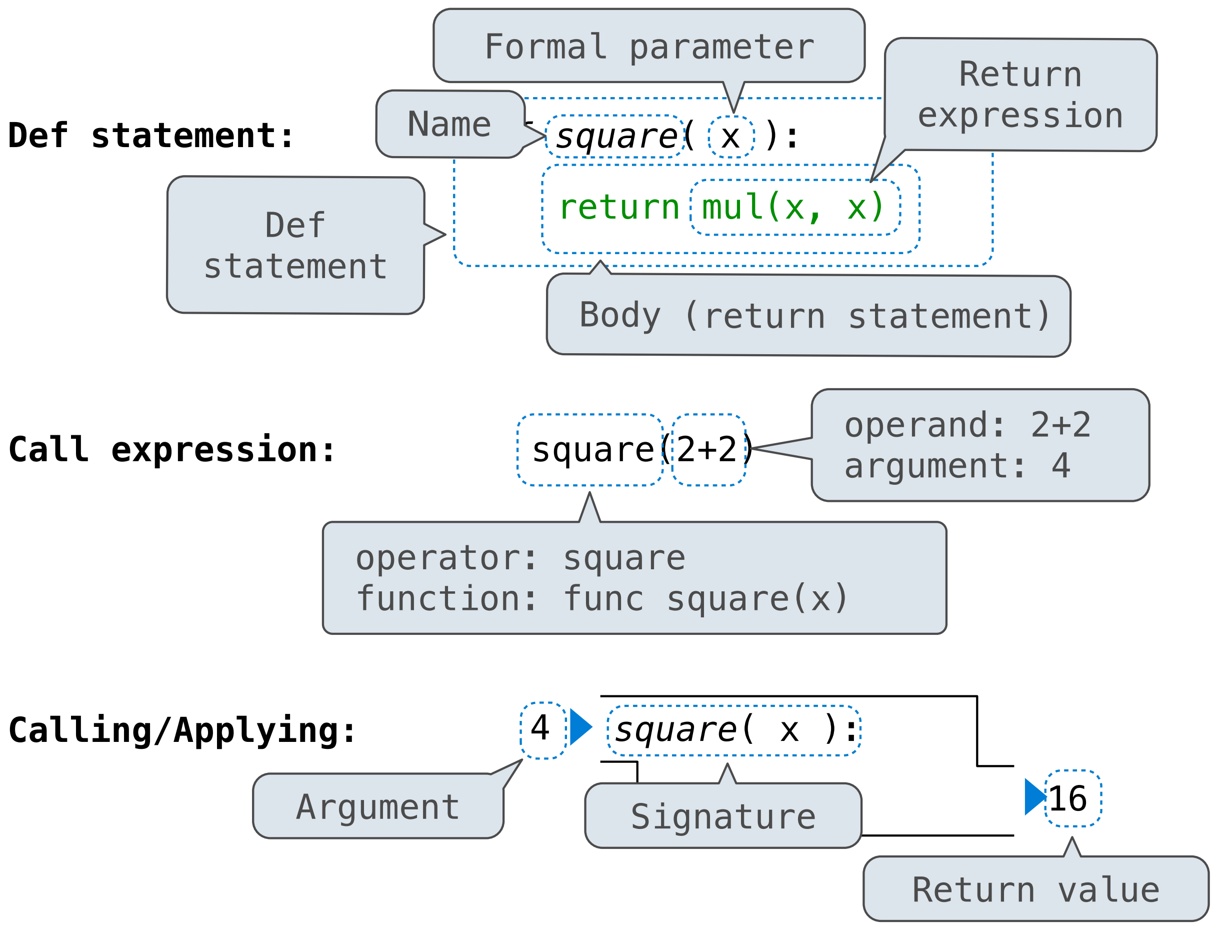
A new function is created!

Name bound to that function in the current frame

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

Life Cycle of a User-Defined Function



Def statement:

Name

Formal parameter

Return expression

Def statement

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

Body (return statement)

Call expression:

```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

Calling/Applying:

4

```
square( x ):
```

16

Argument

Signature

Return value

What happens?

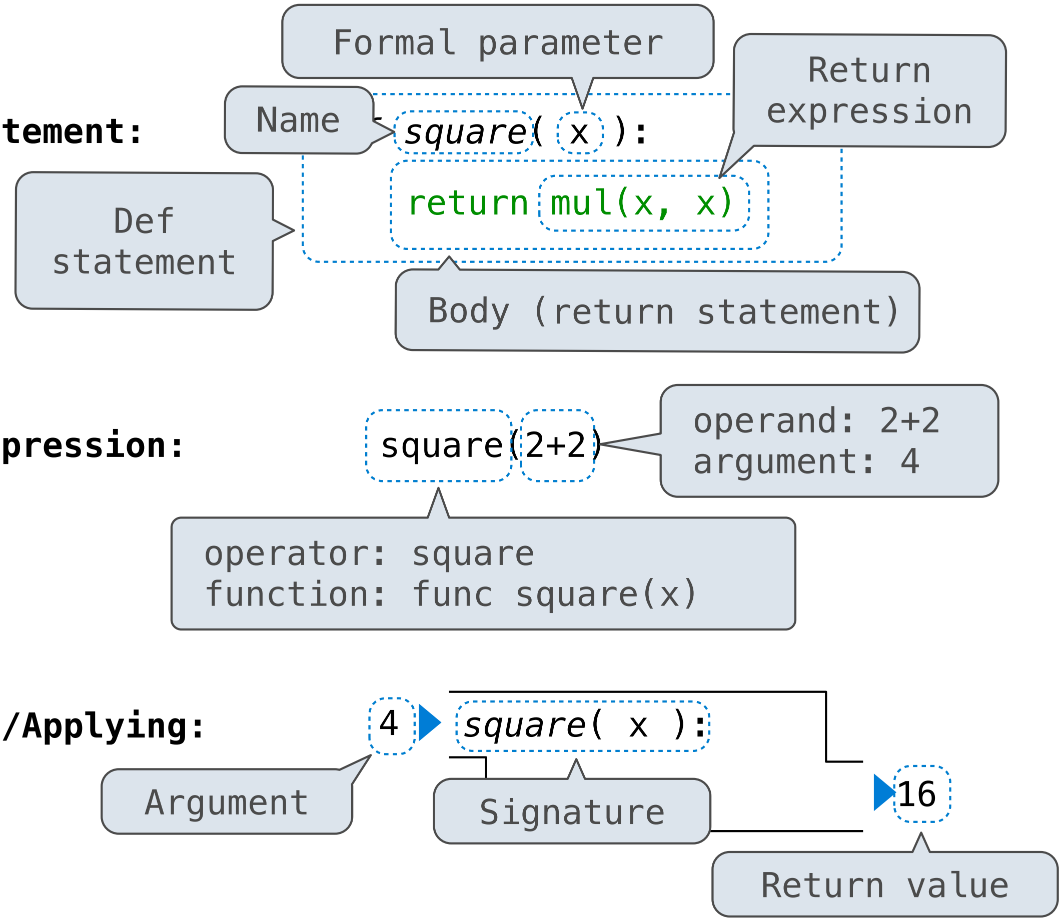
A new function is created!

Name bound to that function
in the current frame

Operator & operands evaluated

Function (value of operator)
called on arguments
(values of operands)

Life Cycle of a User-Defined Function



Def statement:

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

Formal parameter

Name

Return expression

Def statement

Body (return statement)

Call expression:

```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

Calling/Applying:

```
4 square( x ):
16
```

Argument

Signature

Return value

What happens?

A new function is created!

Name bound to that function
in the current frame

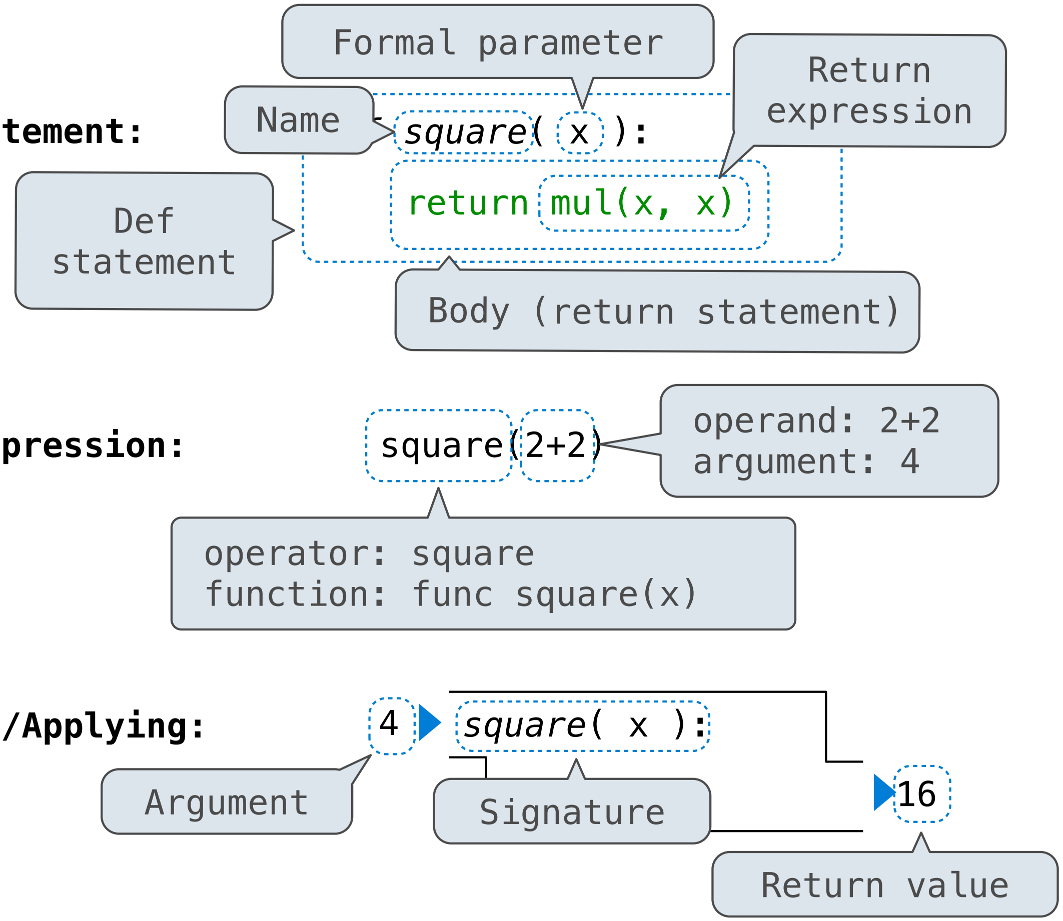
Operator & operands evaluated

Function (value of operator)
called on arguments
(values of operands)

A new frame is created!

Life Cycle of a User-Defined Function

What happens?



Def statement:

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

A new function is created!

Name bound to that function
in the current frame

Call expression:

```
square(2+2)
```

Operator & operands evaluated

Function (value of operator)
called on arguments
(values of operands)

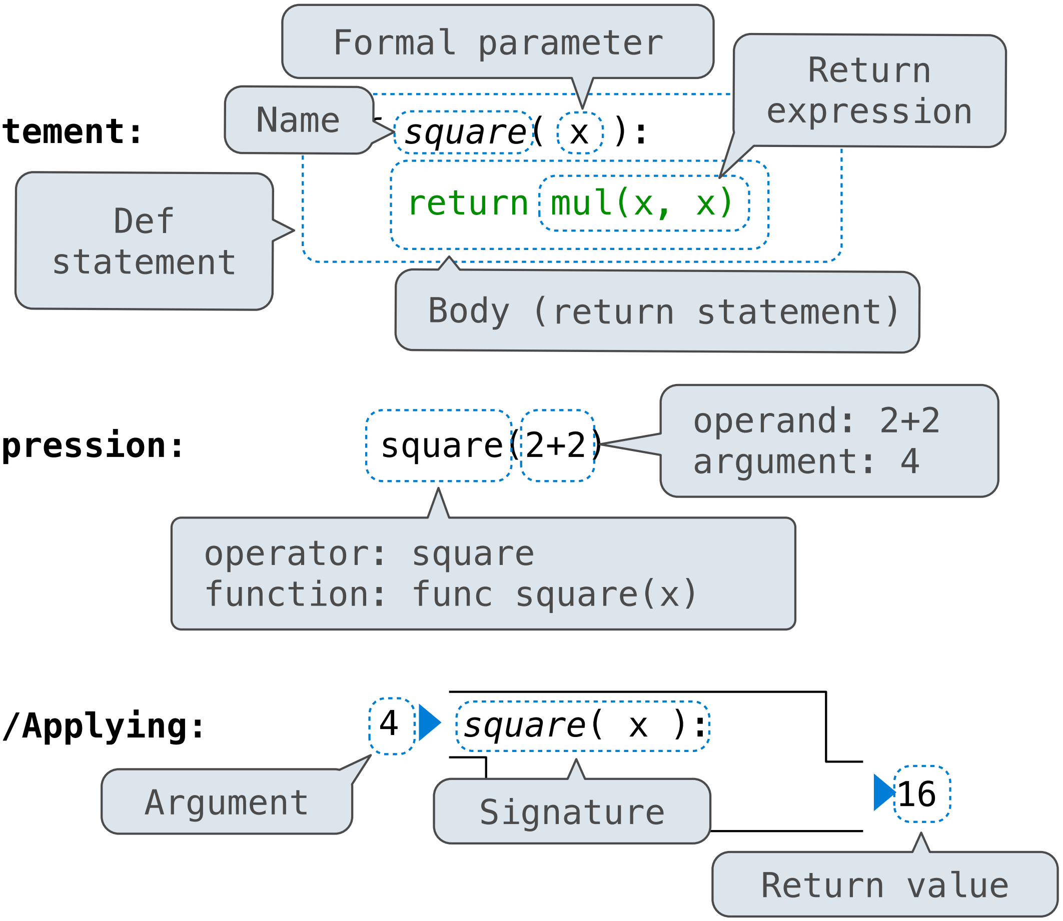
Calling/Applying:

```
4 square( x ): 16
```

A new frame is created!

Parameters bound to arguments

Life Cycle of a User-Defined Function



What happens?

Def statement:

Formal parameter

Name

Return expression

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

Def statement

Body (return statement)

A new function is created!

Name bound to that function in the current frame

Call expression:

```
square(2+2)
```

operand: 2+2
argument: 4

operator: square
function: func square(x)

Operator & operands evaluated

Function (value of operator) called on arguments (values of operands)

Calling/Applying:

```
4 ▶ square( x ): ▶ 16
```

Argument

Signature

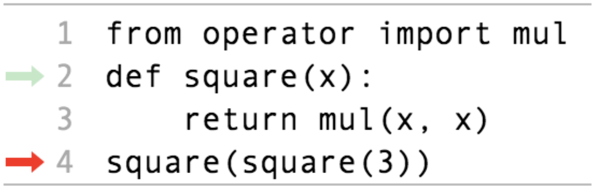
Return value

A new frame is created!

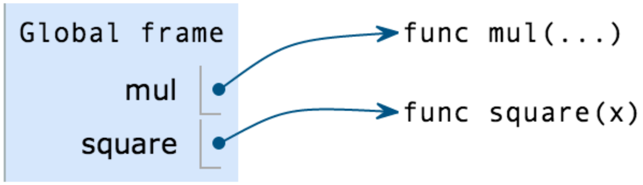
Parameters bound to arguments

Body is executed in that new environment

Multiple Environments in One Diagram!

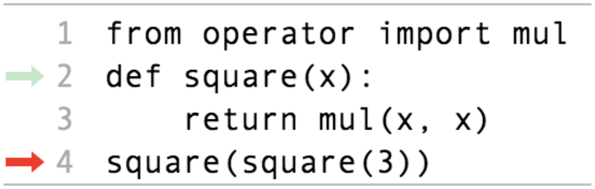


```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

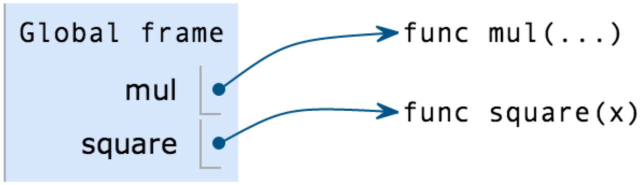


Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



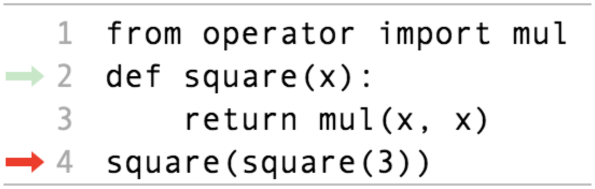
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



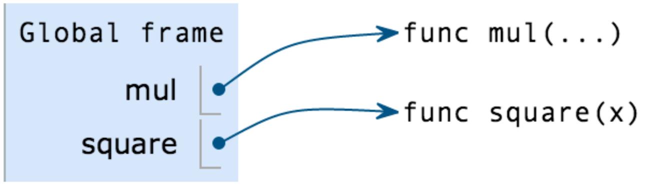
```
square(square(3))
```

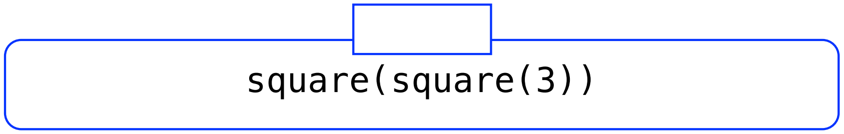
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



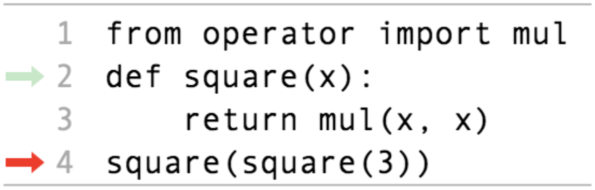
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



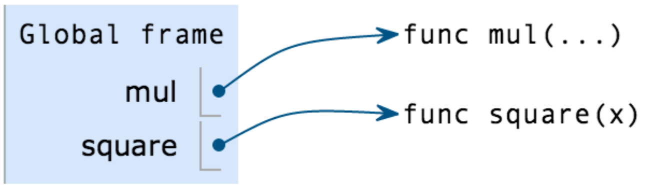


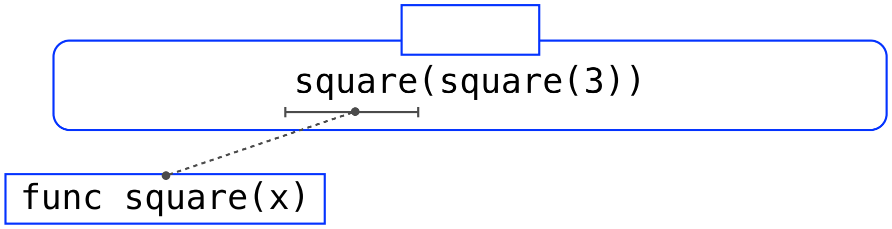
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



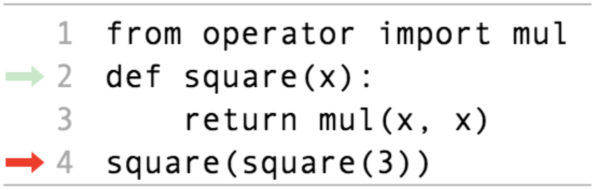
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



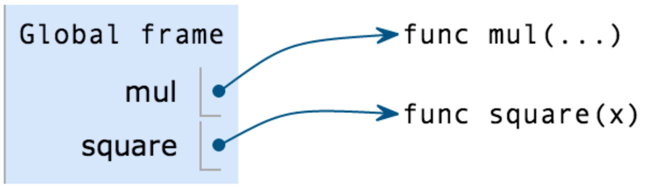


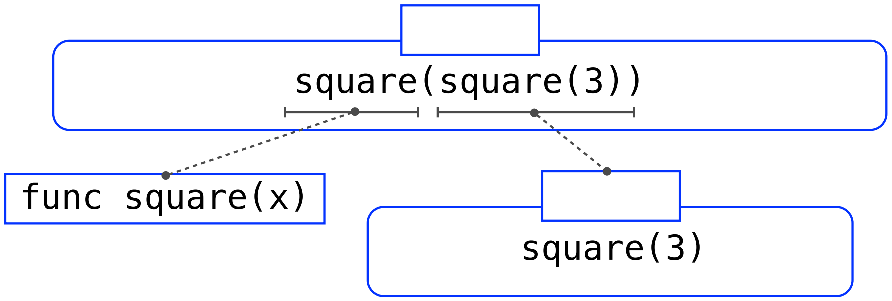
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



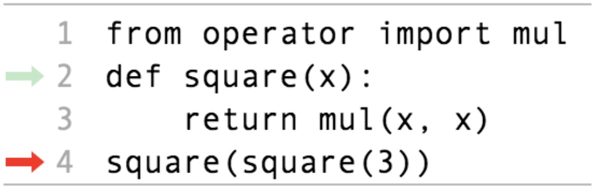
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



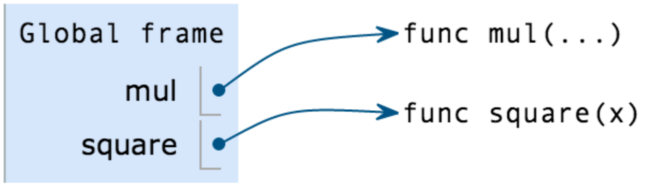


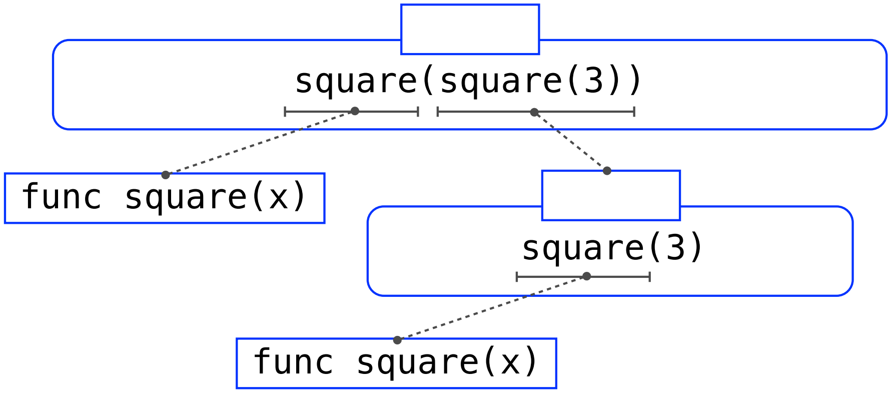
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



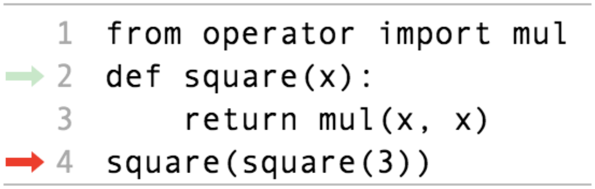
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



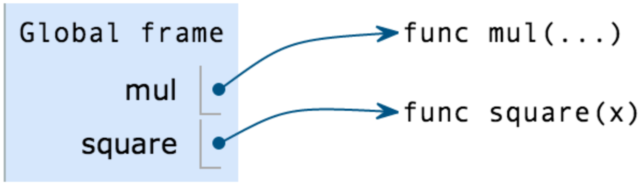


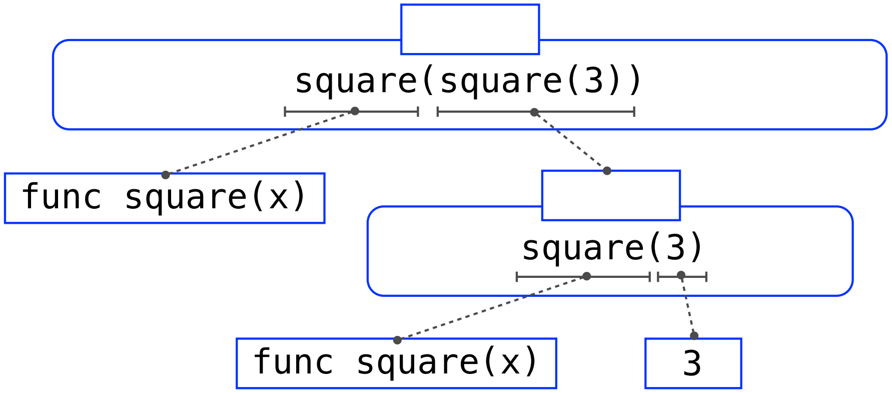
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



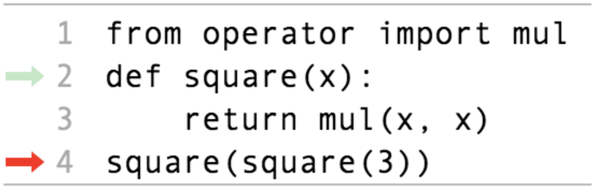
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



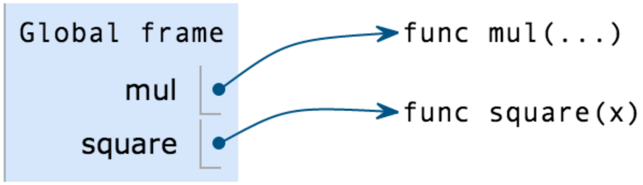


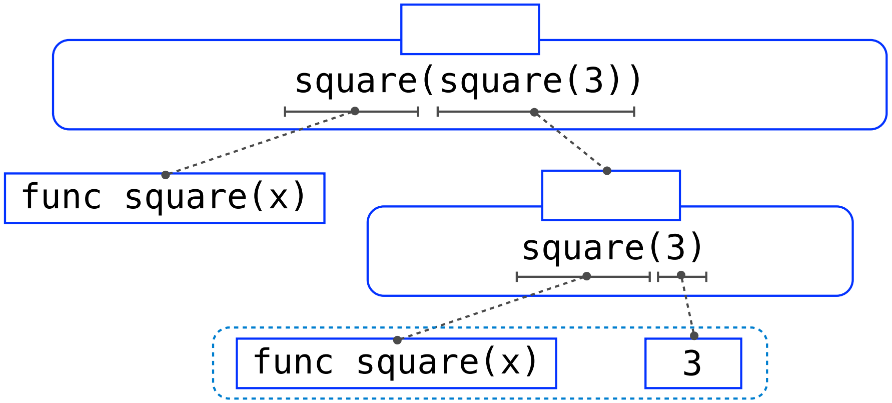
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



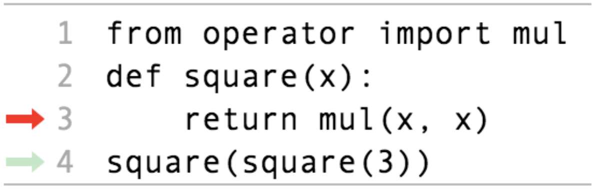
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



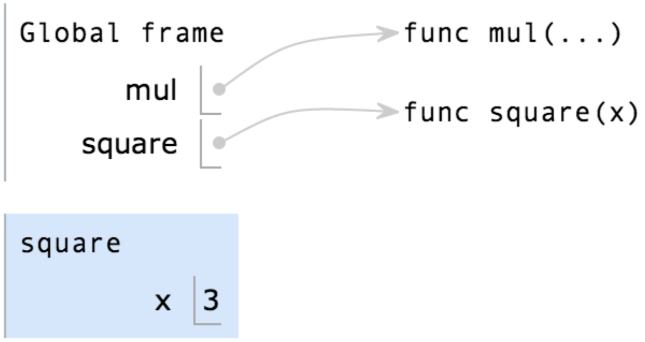


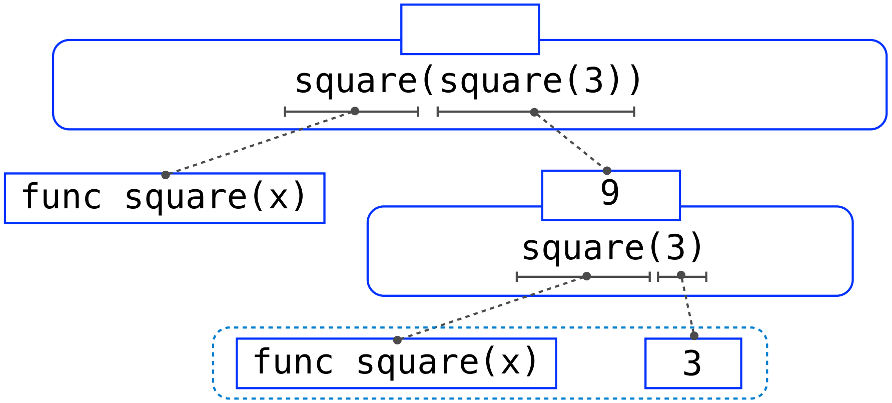
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



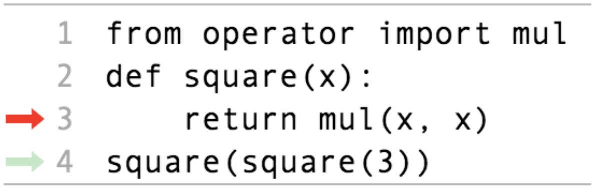
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



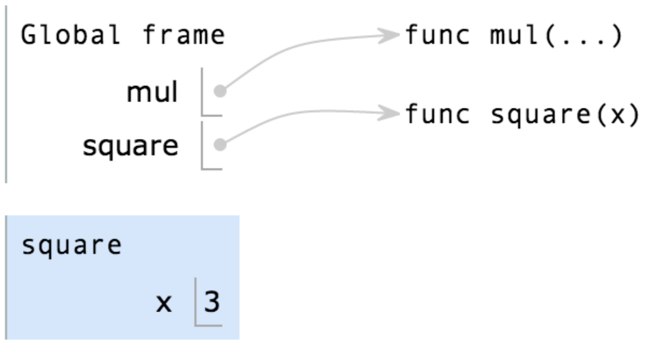


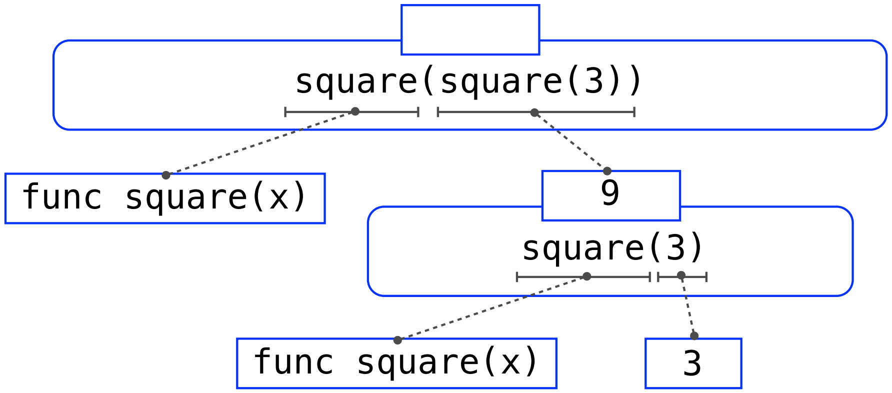
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



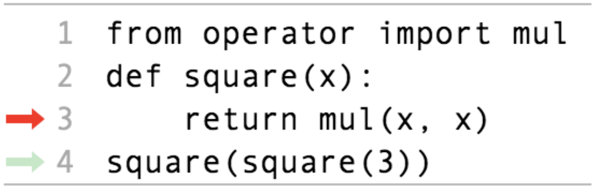
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



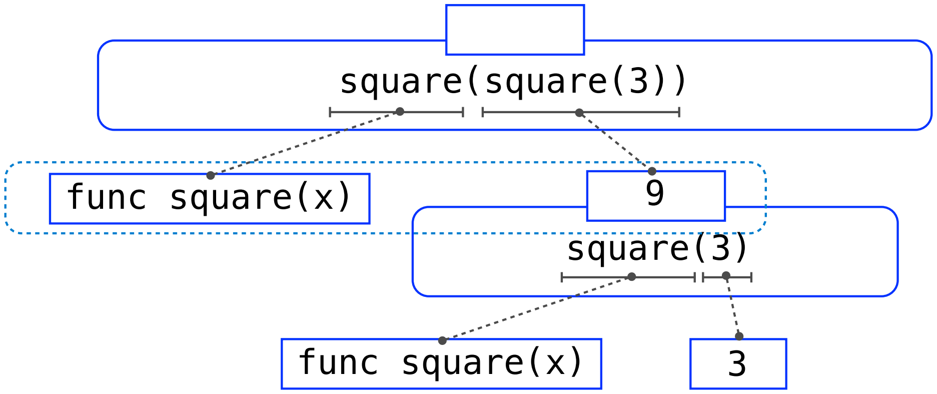


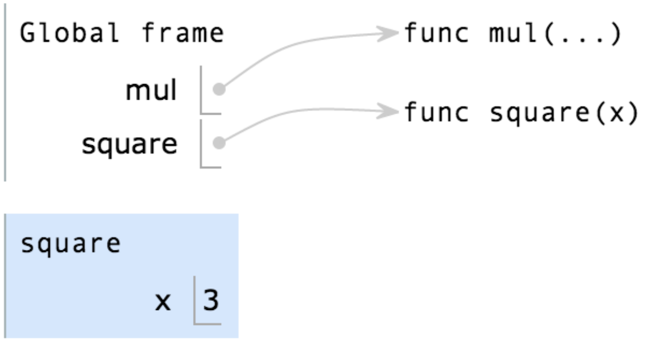
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



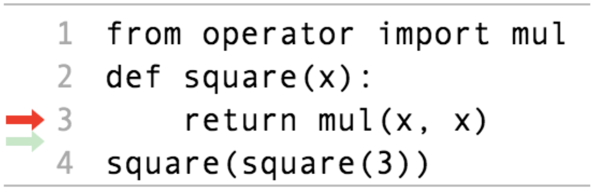
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



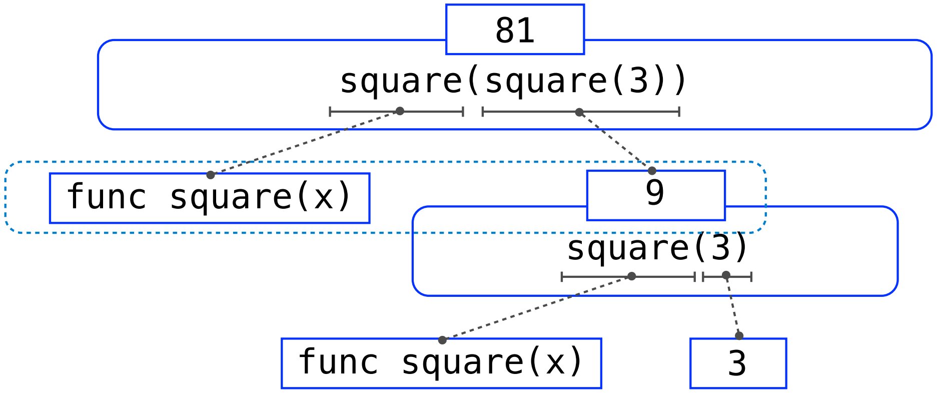


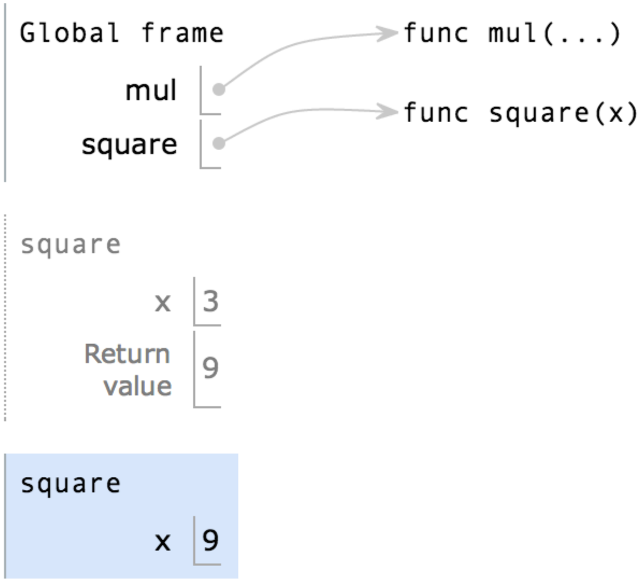
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



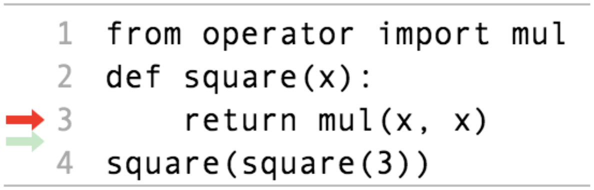
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



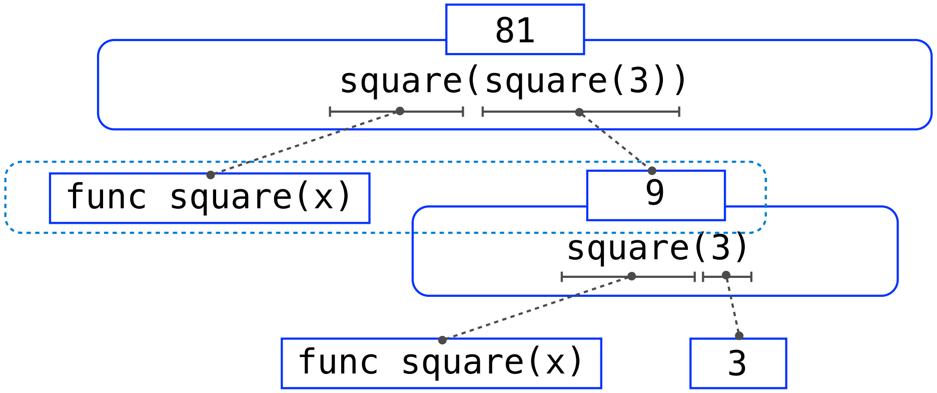


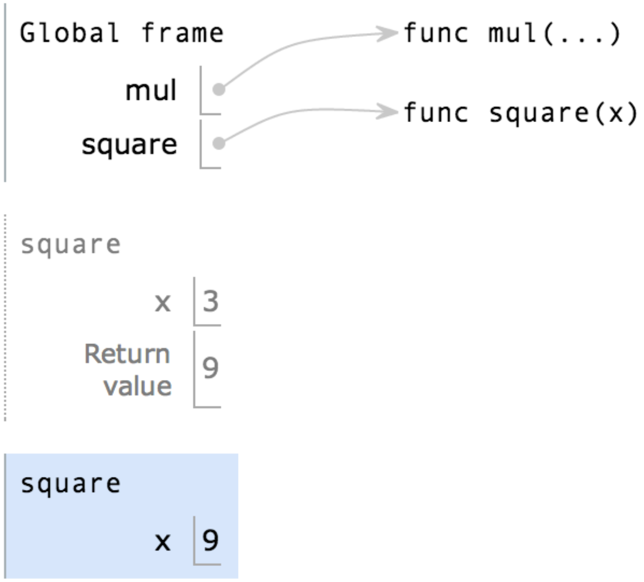
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

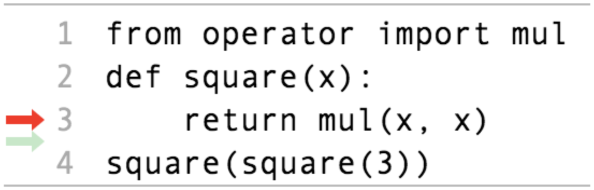




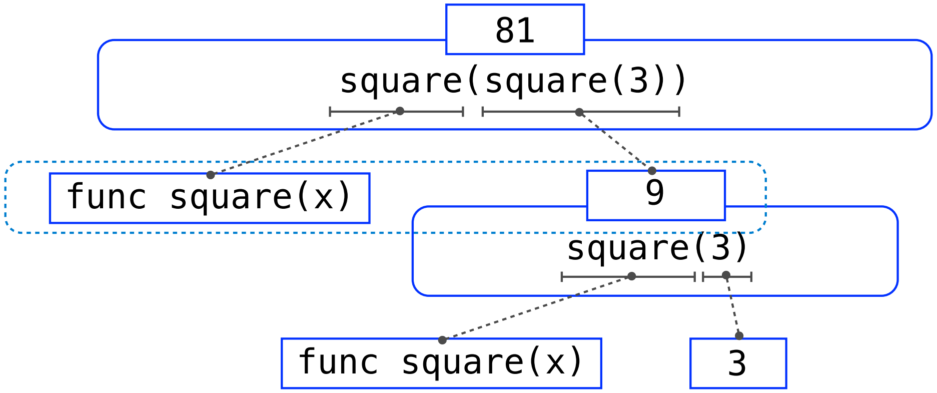
An **environment** is a *sequence of frames*.

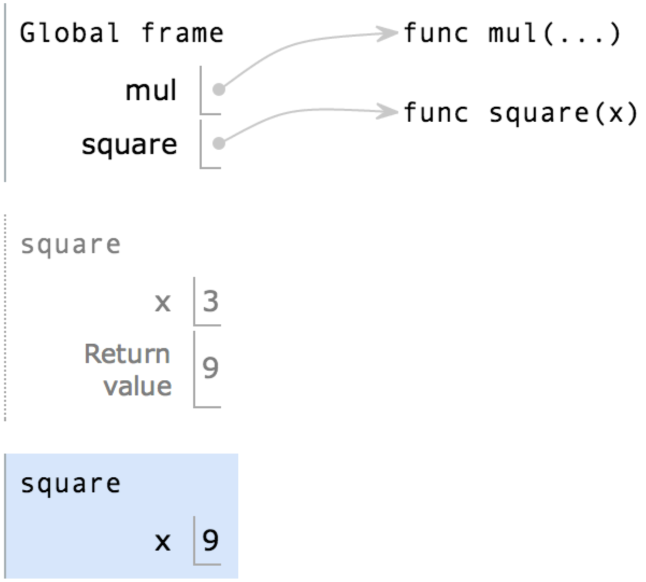
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



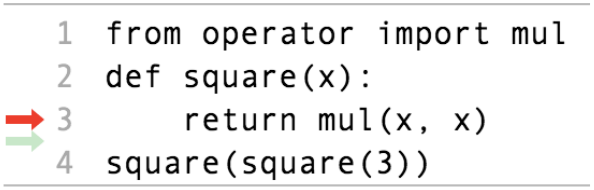


An **environment** is a *sequence of frames*.

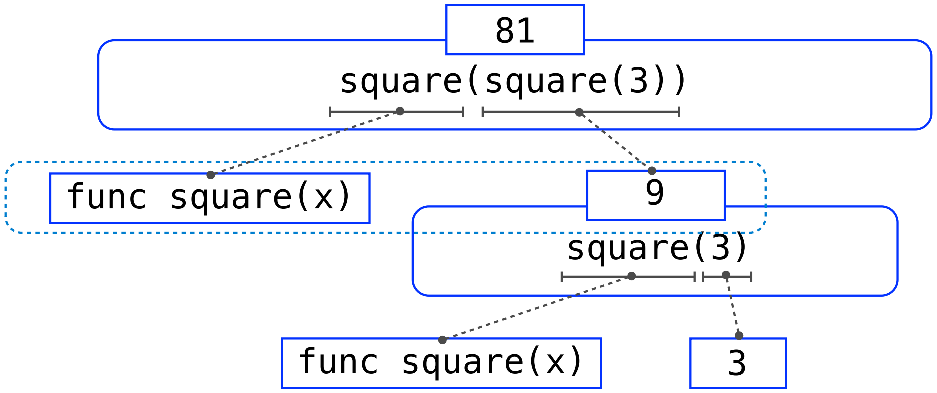
- The global frame alone
- A local, then the global frame

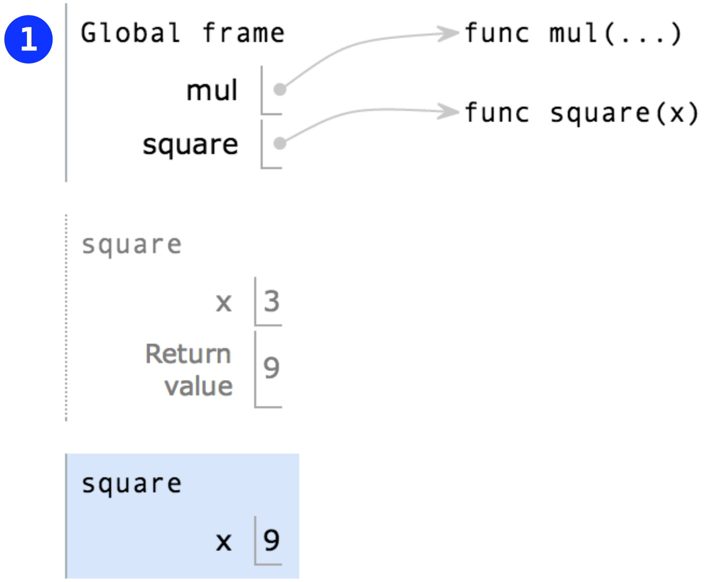
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```





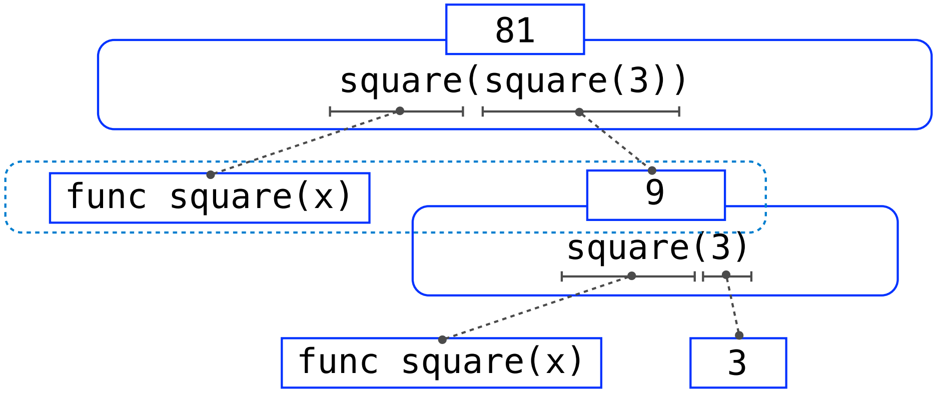
An **environment** is a *sequence of frames*.

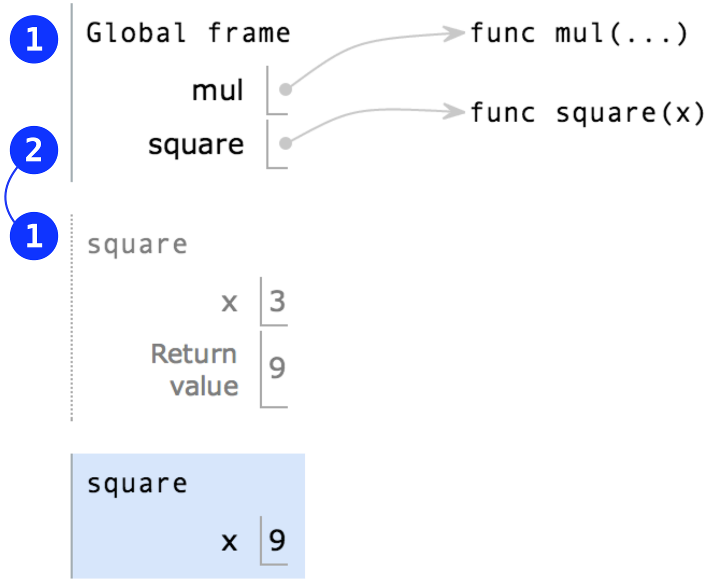
- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!

```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



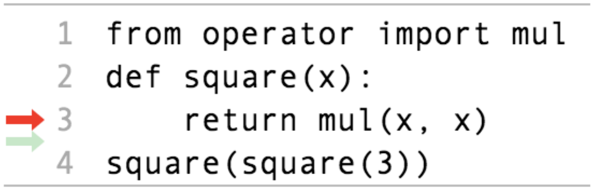


An **environment** is a *sequence of frames*.

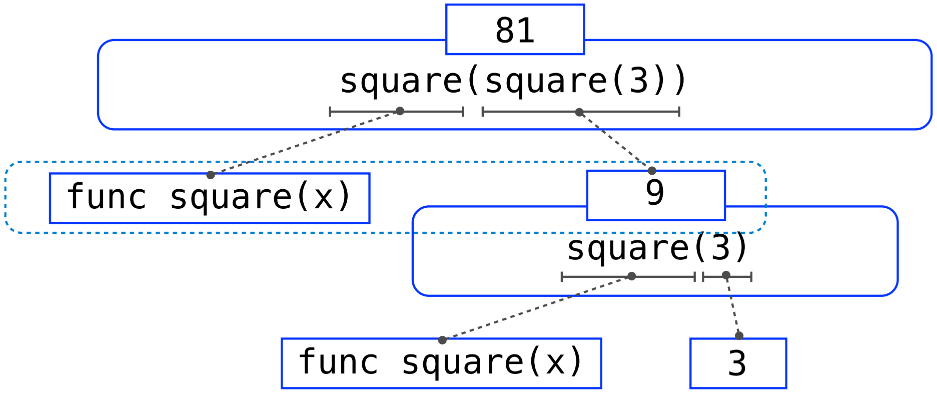
- The global frame alone
- A local, then the global frame

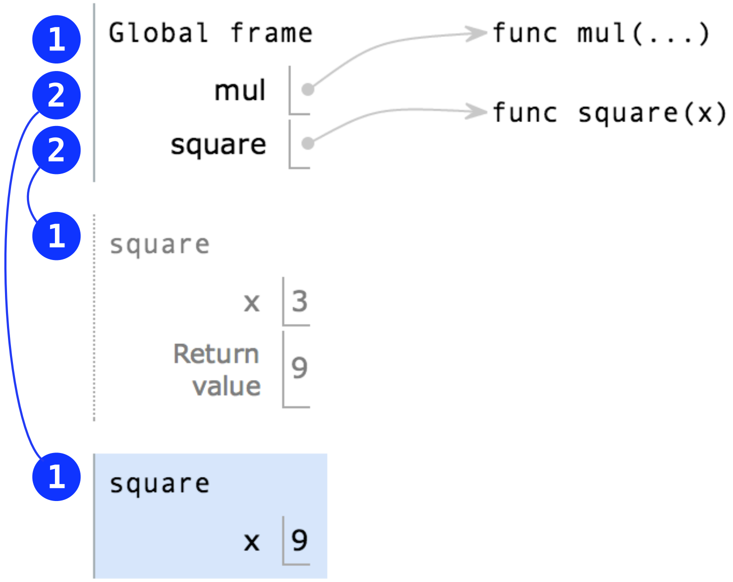
Example: http://goo.gl/XVtEms

Multiple Environments in One Diagram!



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



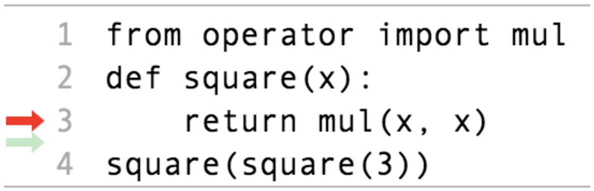


An **environment** is a *sequence of frames*.

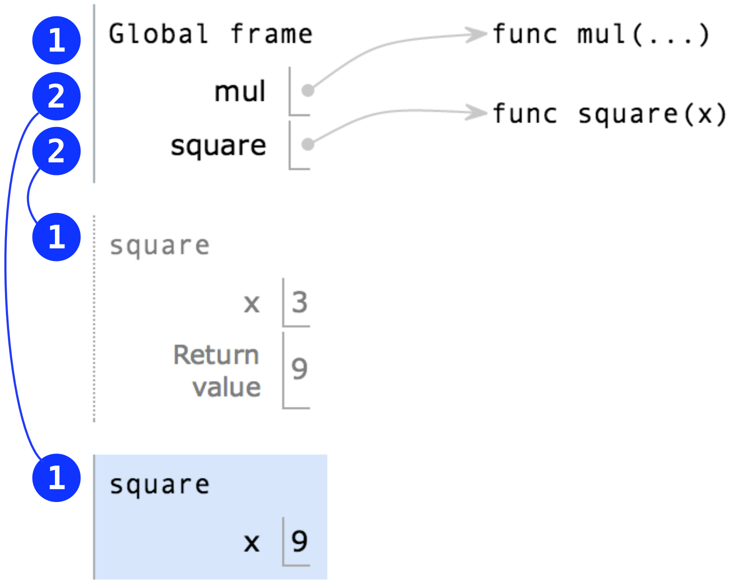
- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Names Have No Meaning Without Environments



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

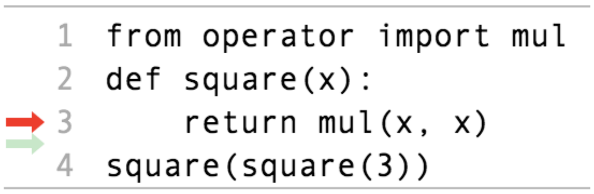


An **environment** is a *sequence of frames*.

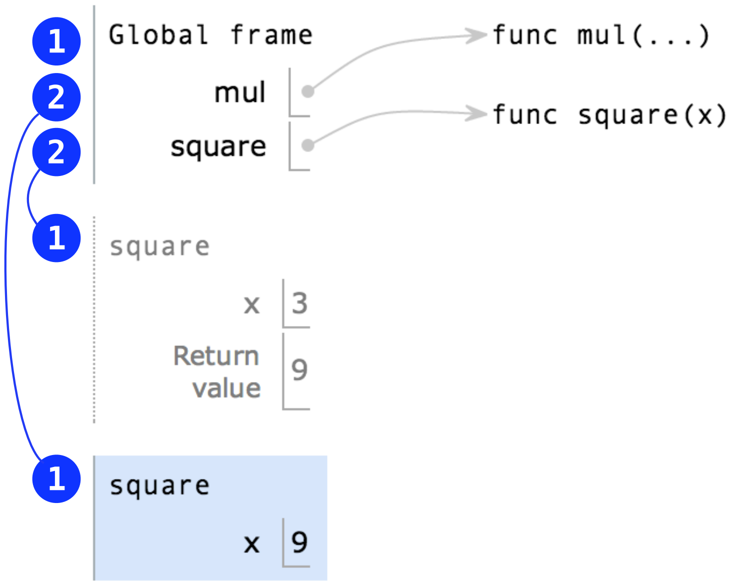
- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Names Have No Meaning Without Environments



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



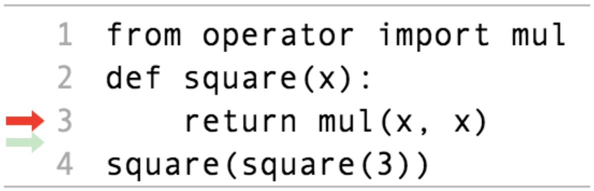
Every expression is evaluated in the context of an environment.

An **environment** is a *sequence of frames*.

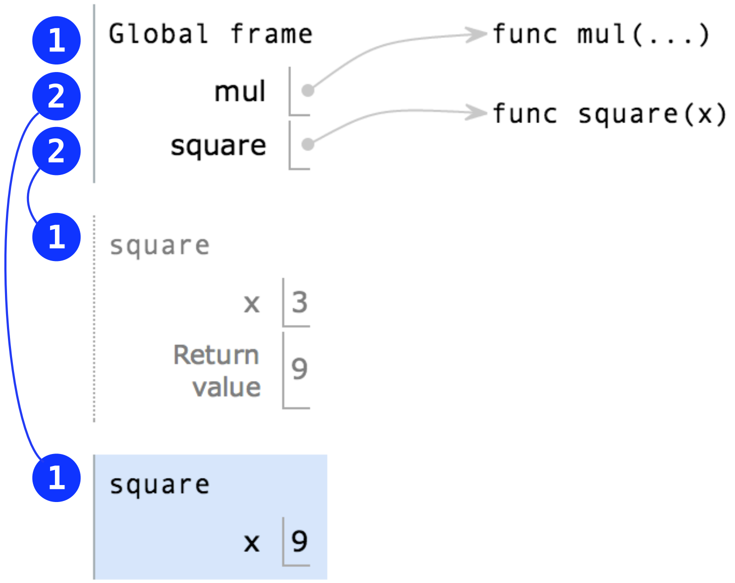
- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Names Have No Meaning Without Environments



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



Every expression is evaluated in the context of an environment.

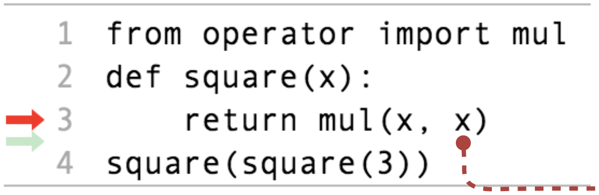
A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

An **environment** is a *sequence of frames*.

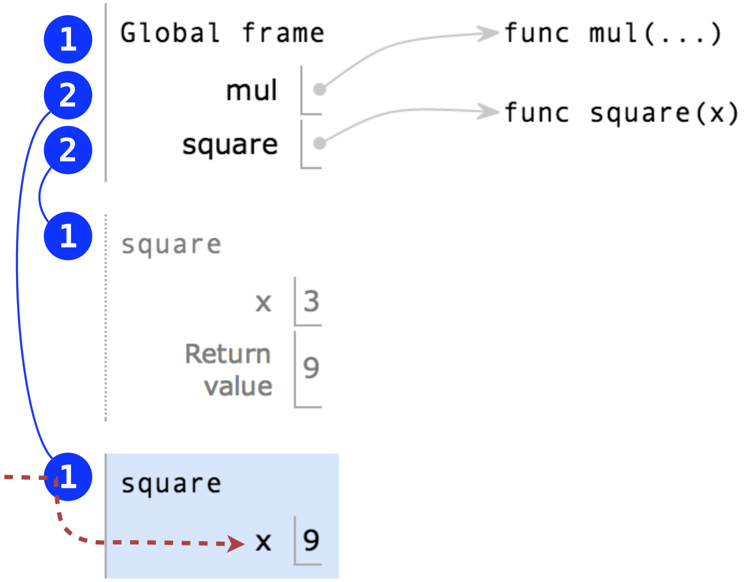
- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Names Have No Meaning Without Environments



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



Every expression is evaluated in the context of an environment.

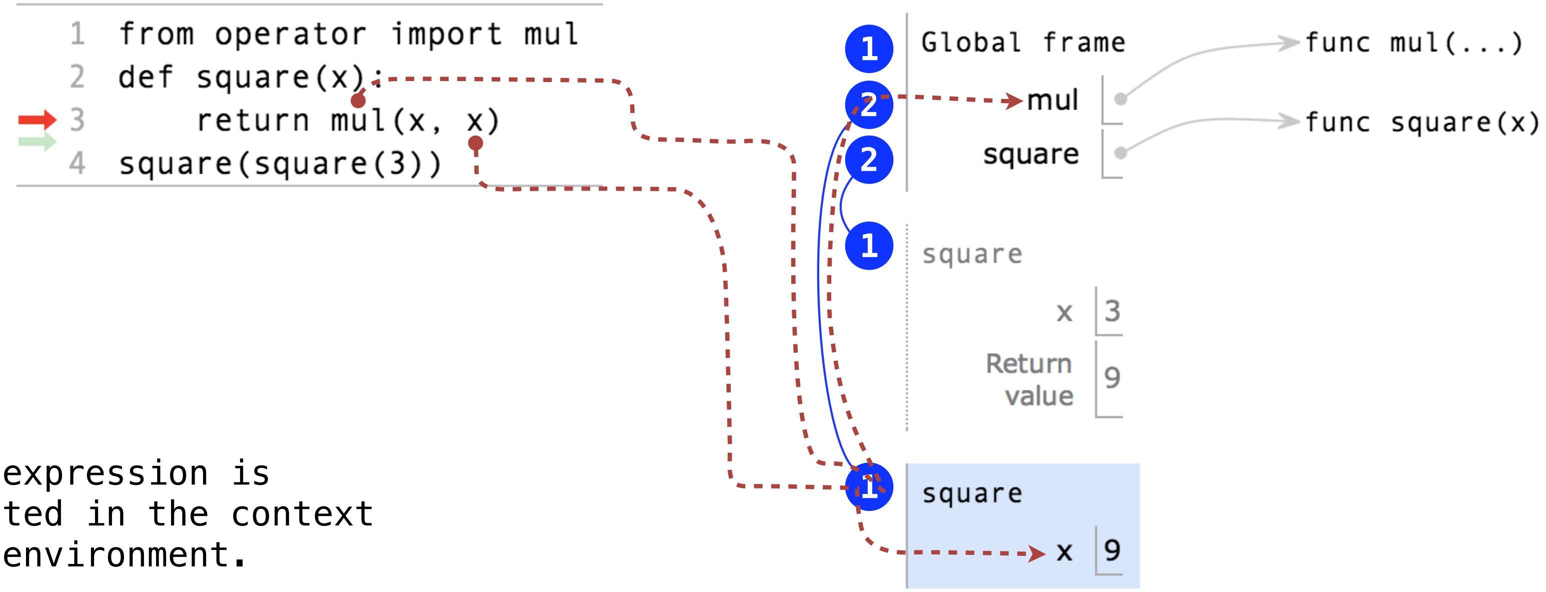
A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

An **environment** is a *sequence of frames*.

- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Names Have No Meaning Without Environments



```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

Every expression is evaluated in the context of an environment.

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

An **environment** is a *sequence of frames*.

- The global frame alone
- A local, then the global frame

Example: http://goo.gl/XVtEms

Miscellaneous Python Features

Operators

Multiple Return Values

Docstrings

Doctests

Default Arguments

(Demo)

Conditional Statements

Statements

Statements

A ***statement*** is executed by the interpreter to perform an action

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:

```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:

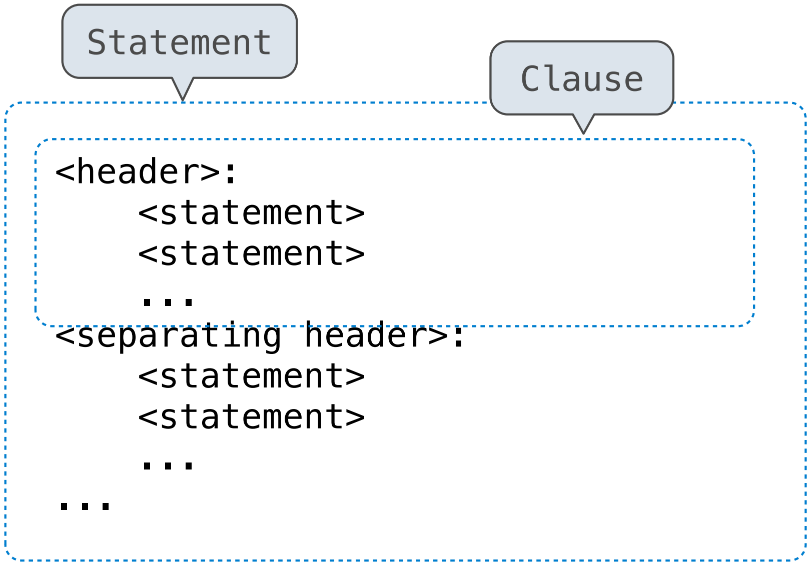
Statement

```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:



Statement

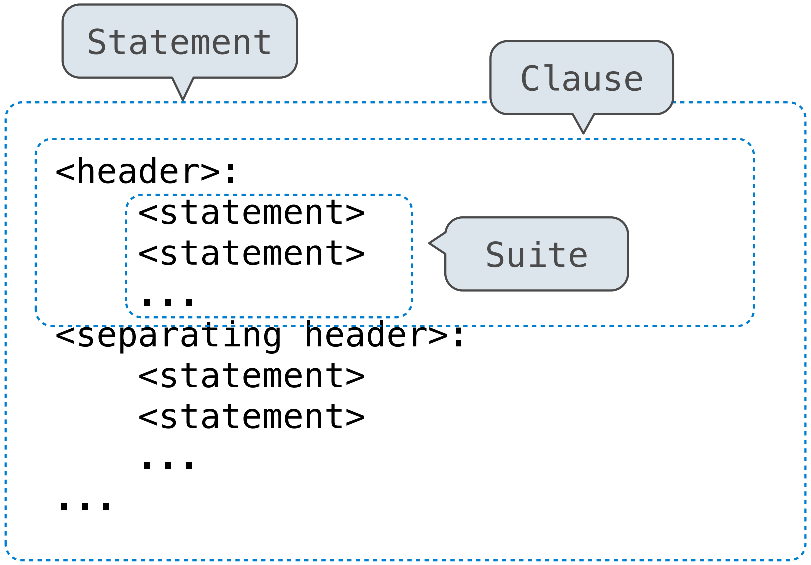
Clause

```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:

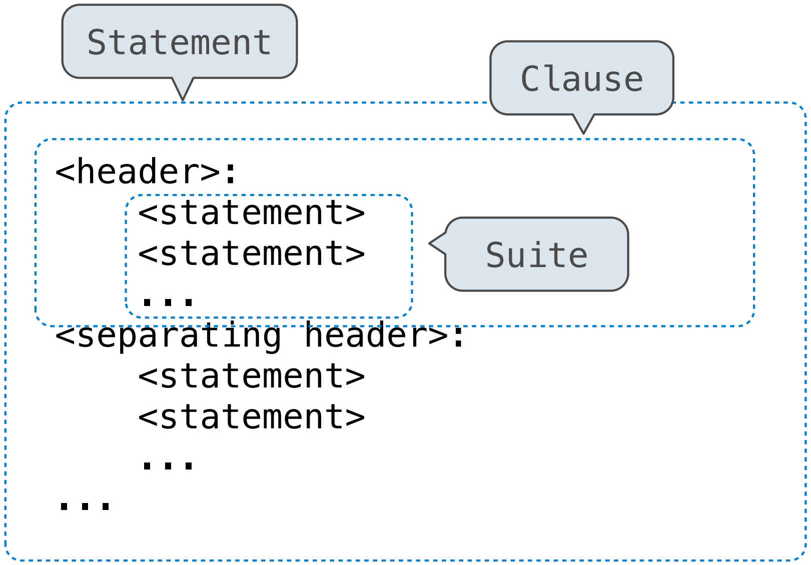


```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:



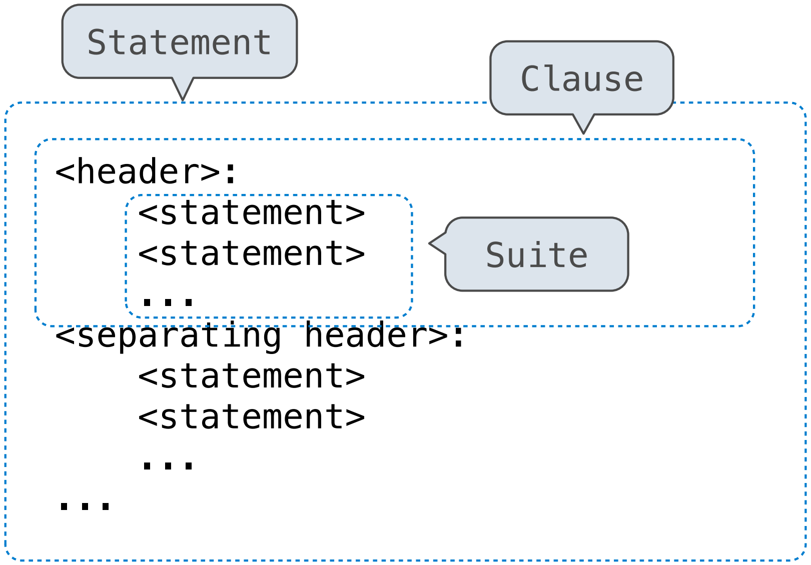
```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

The first header determines a statement’s type

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:



```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

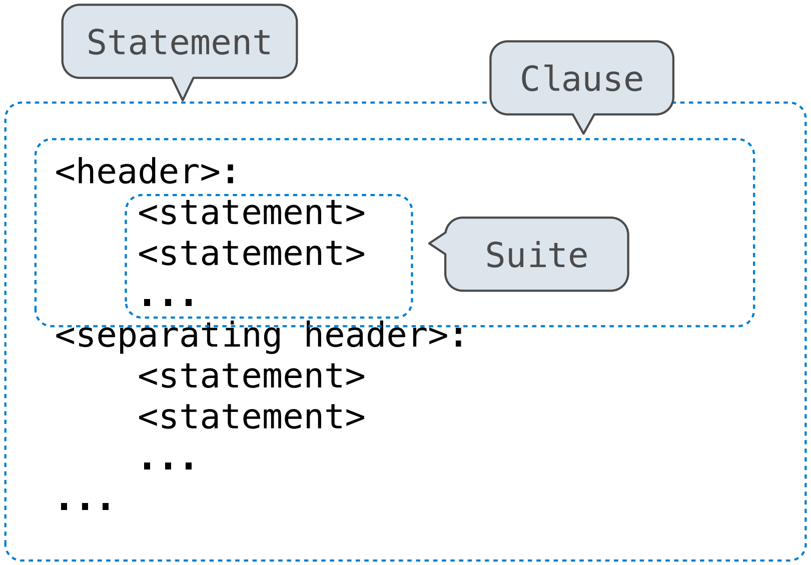
The first header determines a statement’s type

The header of a clause “controls” the suite that follows

Statements

A ***statement*** is executed by the interpreter to perform an action

Compound statements:



```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

The first header determines a statement's type

The header of a clause "controls" the suite that follows

def statements are compound statements

Compound Statements

Compound statements:

```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Suite

Compound Statements

Compound statements:

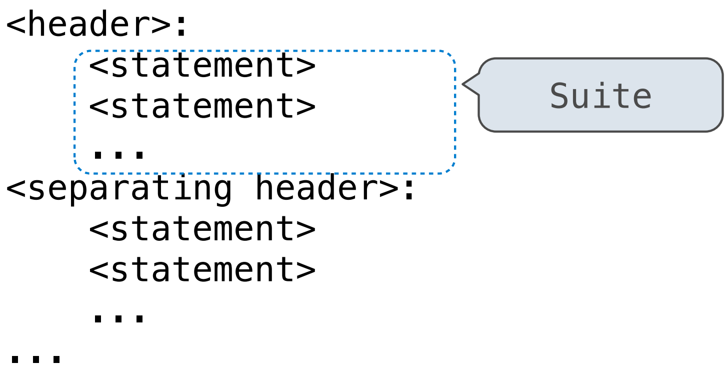
```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Suite

A suite is a sequence of statements

Compound Statements

Compound statements:



```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

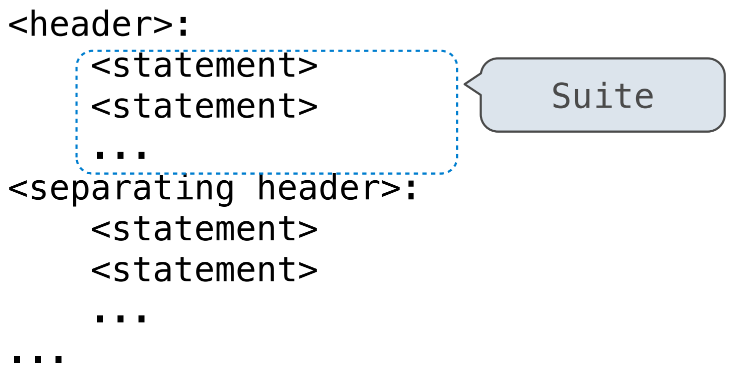
Suite

A suite is a sequence of statements

To "execute" a suite means to execute its sequence of statements, in order

Compound Statements

Compound statements:



```
<header>:
    <statement>
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```

Suite

A suite is a sequence of statements

To “execute” a suite means to execute its sequence of statements, in order

Execution Rule for a sequence of statements:

- Execute the first statement
- Unless directed otherwise, execute the rest

Conditional Statements

(Demo)

Conditional Statements

(Demo)

```python
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Conditional Statements

(Demo)

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

1 statement,
3 clauses,
3 headers,
3 suites

Conditional Statements

(Demo)

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

1 statement,
3 clauses,
3 headers,
3 suites

Execution rule for conditional statements:

Conditional Statements

(Demo)

1 statement,
3 clauses,
3 headers,
3 suites

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Execution rule for conditional statements:

Each clause is considered in order.

1. Evaluate the header's expression.

2. If it is a true value,
   execute the suite & skip the remaining clauses.

Conditional Statements

(Demo)

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

1 statement,
3 clauses,
3 headers,
3 suites

Execution rule for conditional statements:

Each clause is considered in order.

1. Evaluate the header's expression.

2. If it is a true value,
   execute the suite & skip the remaining clauses.

Syntax Tips

Conditional Statements

(Demo)

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

1 statement,
3 clauses,
3 headers,
3 suites

Execution rule for conditional statements:

Each clause is considered in order.

1. Evaluate the header's expression.
2. If it is a true value,
   execute the suite & skip the remaining clauses.

Syntax Tips

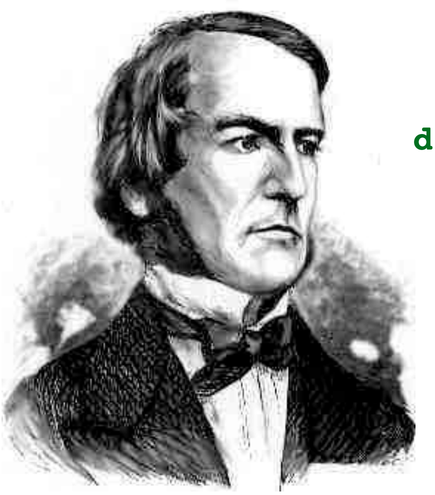
1. Always starts with "if" clause.
2. Zero or more "elif" clauses.
3. Zero or one "else" clause,
   always at the end.

Boolean Contexts

George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

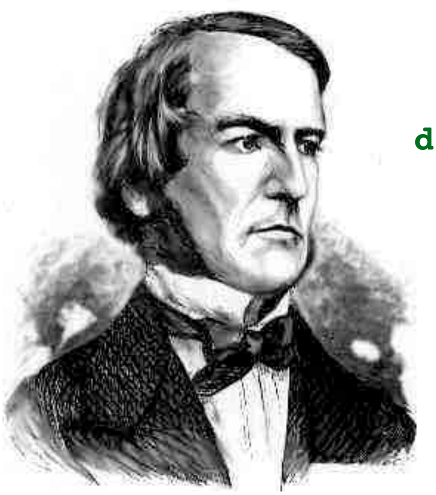
Boolean Contexts



George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Boolean Contexts

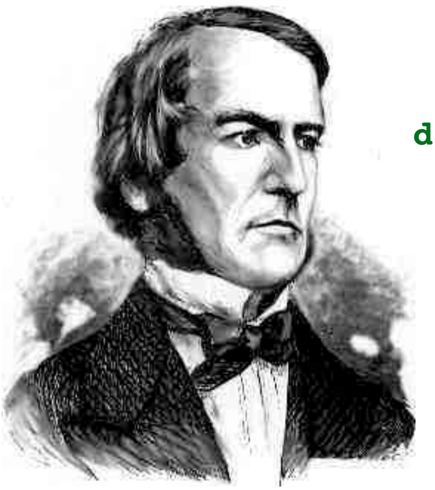


George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

Boolean Contexts



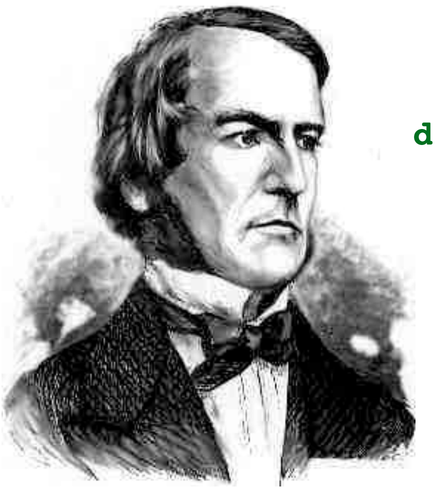
George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

False values in Python: False, 0, '', None

Boolean Contexts



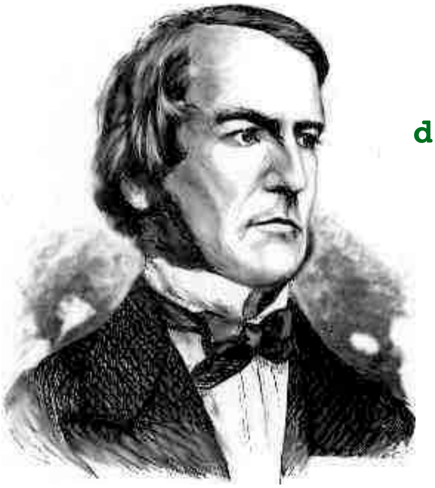
George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

False values in Python: False, 0, '', None   *(more to come)*

Boolean Contexts



George Boole

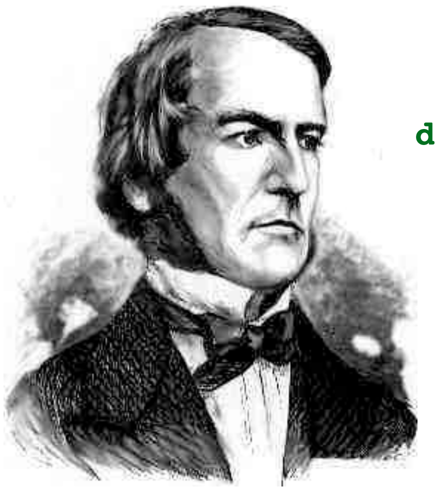
```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

False values in Python: False, 0, '', None *(more to come)*

True values in Python: Anything else (True)

Boolean Contexts



George Boole

```
def absolute_value(x):
    """Return the absolute value of x."""
    if x < 0:
        return -x
    elif x == 0:
        return 0
    else:
        return x
```

Two boolean contexts

False values in Python: False, 0, '', None *(more to come)*

True values in Python: Anything else (True)

Read Section 1.5.4!

Reading: http://composingprograms.com/pages/15-control.html#conditional-statements

Iteration

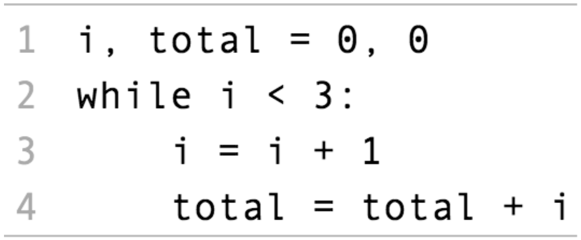
While Statements

(Demo)

Example: http://goo.gl/0d2cjF

While Statements

(Demo)

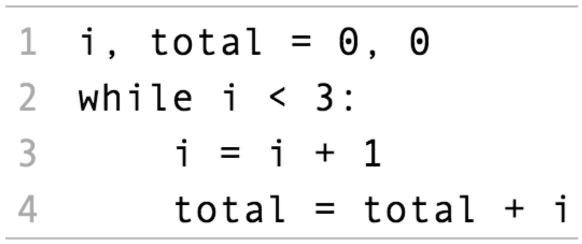


```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Example: http://goo.gl/0d2cjF

While Statements

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Execution rule for while statements:

1. Evaluate the header's expression.
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Execution rule for while statements:

1. Evaluate the header's expression.

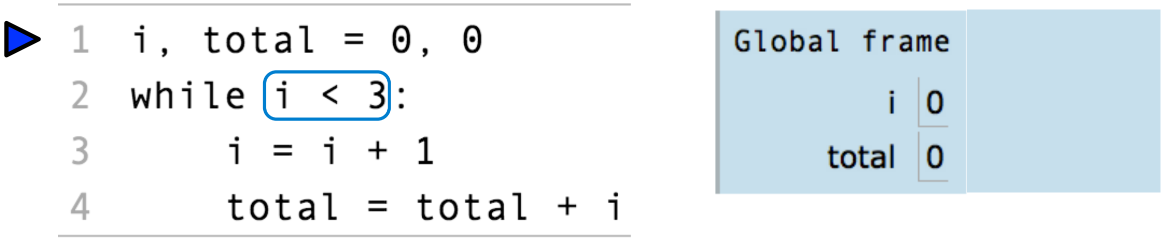
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	0
total	0

Execution rule for while statements:

1. Evaluate the header’s expression.

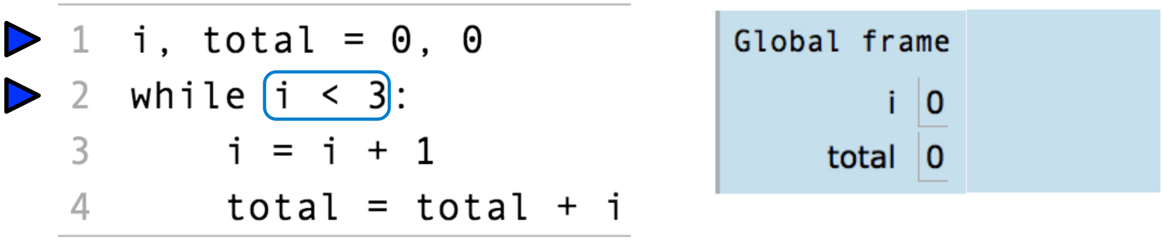
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	0
total	0

Execution rule for while statements:

1. Evaluate the header's expression.

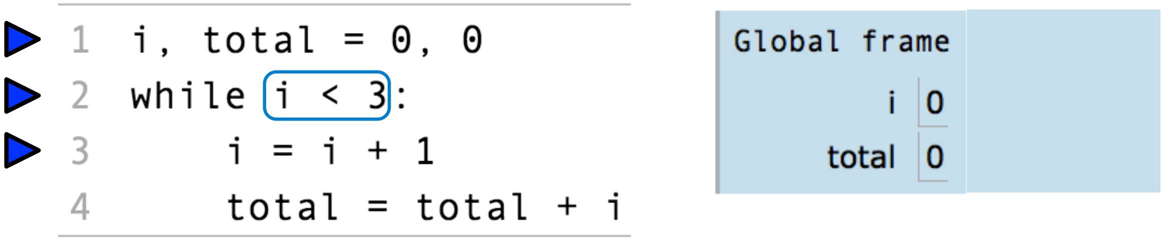
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	0
total	0

Execution rule for while statements:

1. Evaluate the header’s expression.

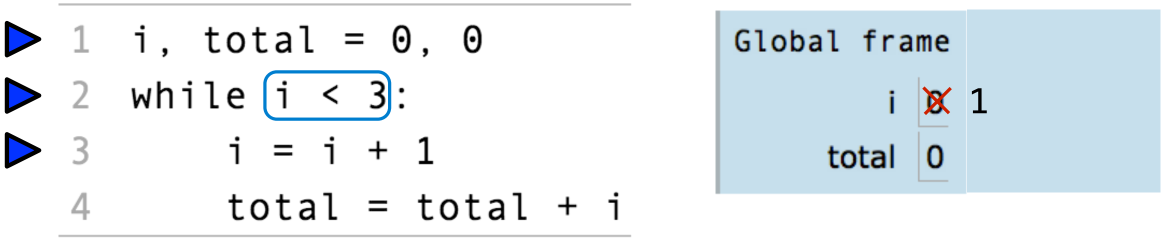
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ 1

total 0

Execution rule for while statements:

1. Evaluate the header's expression.

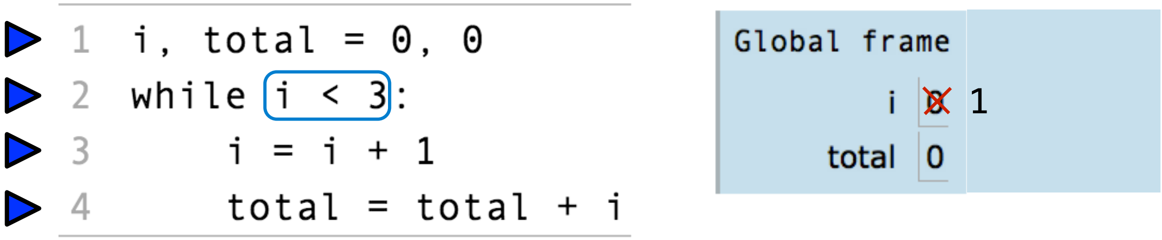
2. If it is a true value, execute the (*whole*) suite, then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ 1

total 0

Execution rule for while statements:

1. Evaluate the header’s expression.

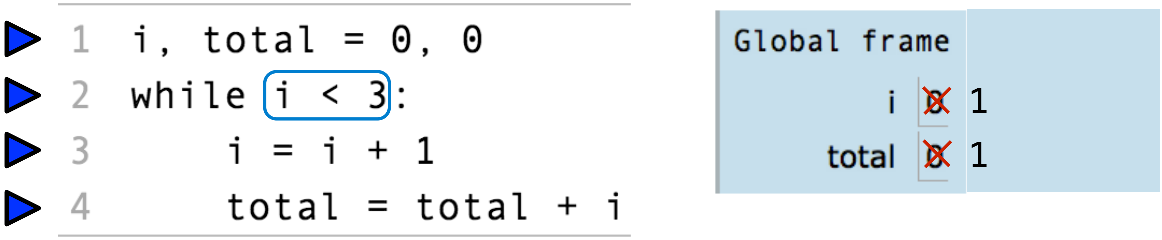
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ 1

total ~~0~~ 1

Execution rule for while statements:

1. Evaluate the header's expression.

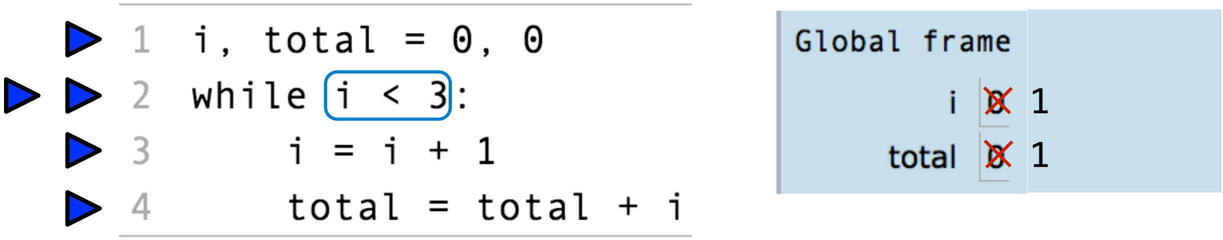
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ 1

total ~~0~~ 1

Execution rule for while statements:

1. Evaluate the header's expression.

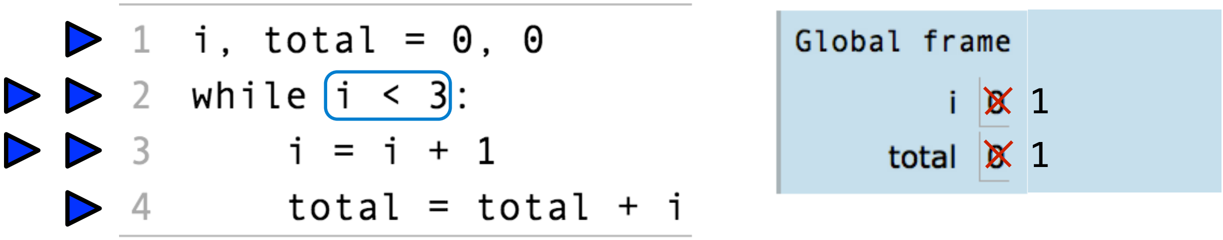
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	~~0~~ 1
total	~~0~~ 1

Execution rule for while statements:

1. Evaluate the header’s expression.

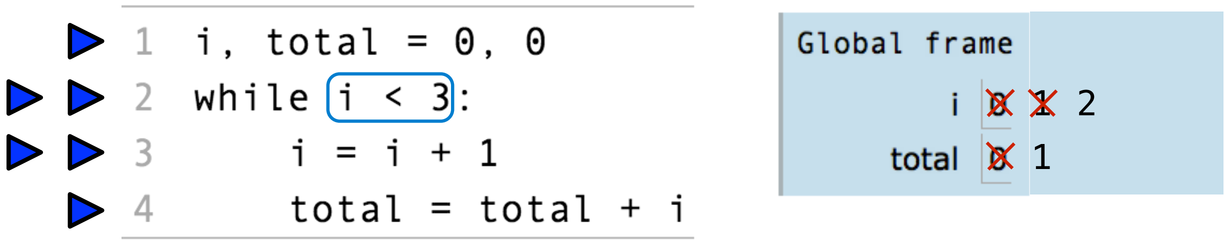
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Execution rule for while statements:

1. Evaluate the header’s expression.

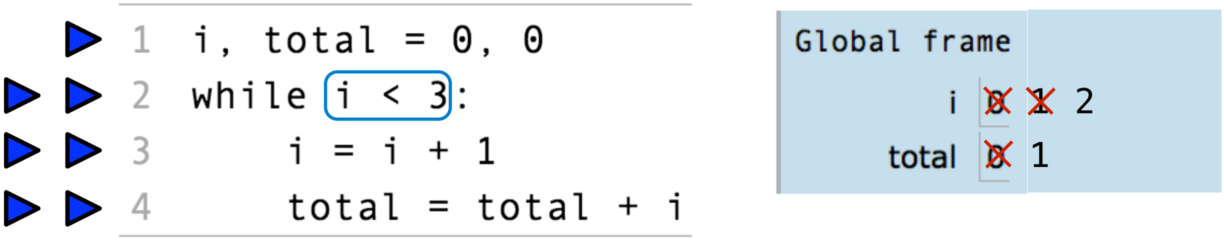
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	~~0~~ ~~1~~ 2
total	~~0~~ 1

Execution rule for while statements:

1. Evaluate the header's expression.

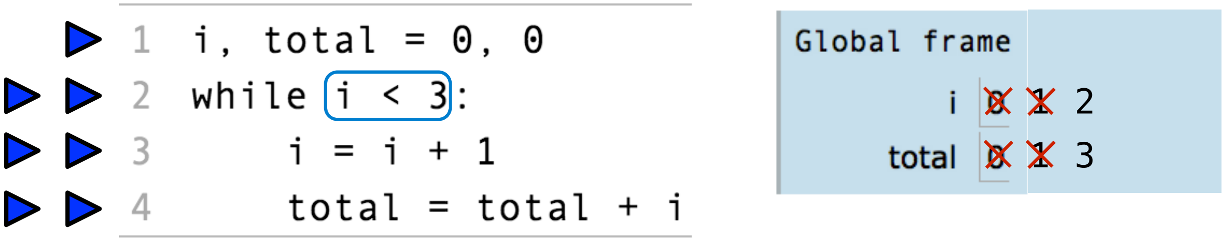
2. If it is a true value, execute the (*whole*) suite, then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	~~0~~ ~~1~~ 2
total	~~0~~ ~~1~~ 3

Execution rule for while statements:

1. Evaluate the header’s expression.

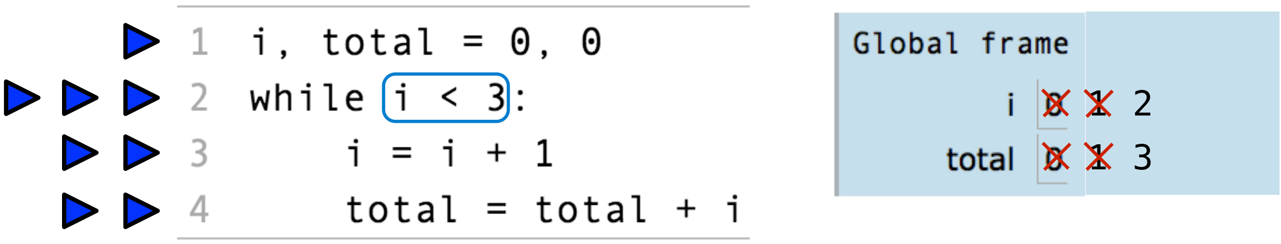
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i	~~0~~ ~~1~~ 2
total	~~0~~ ~~1~~ 3

Execution rule for while statements:

1. Evaluate the header’s expression.

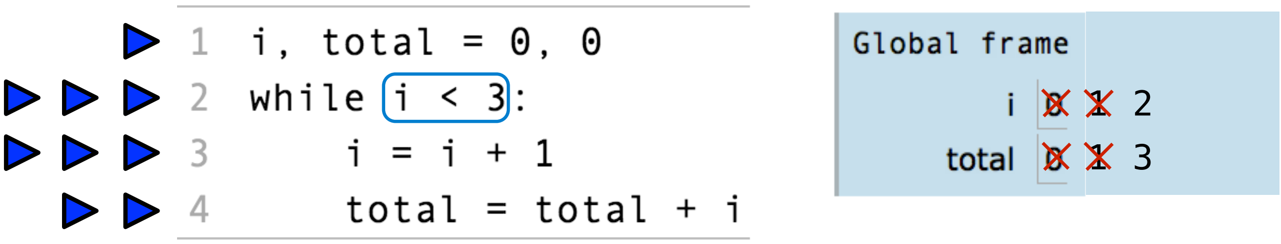
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ ~~1~~ 2

total ~~0~~ ~~1~~ 3

Execution rule for while statements:

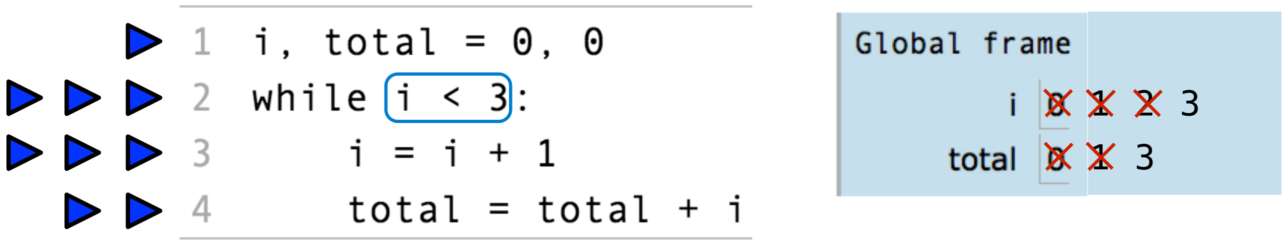
1. Evaluate the header's expression.
2. If it is a true value, execute the (*whole*) suite, then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ ~~1~~ ~~2~~ 3

total ~~0~~ ~~1~~ 3

Execution rule for while statements:

1. Evaluate the header's expression.

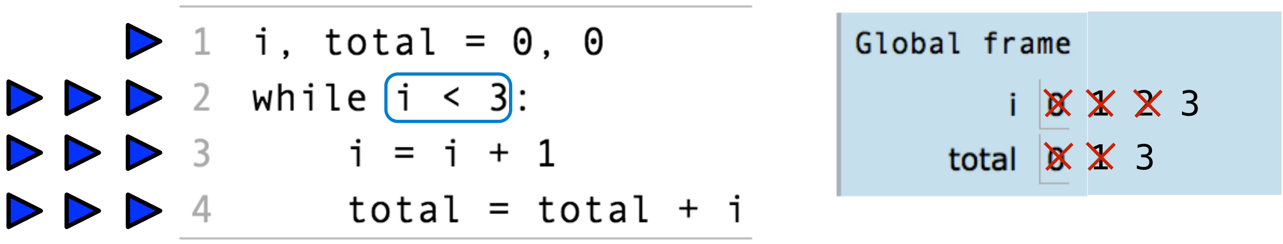
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ ~~1~~ ~~2~~ 3

total ~~0~~ ~~1~~ 3

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

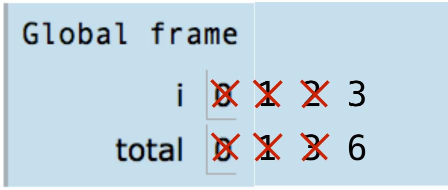
Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)

```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```



Global frame

i ~~0~~ ~~1~~ ~~2~~ 3

total ~~0~~ ~~1~~ ~~3~~ 6

Execution rule for while statements:

1. Evaluate the header’s expression.

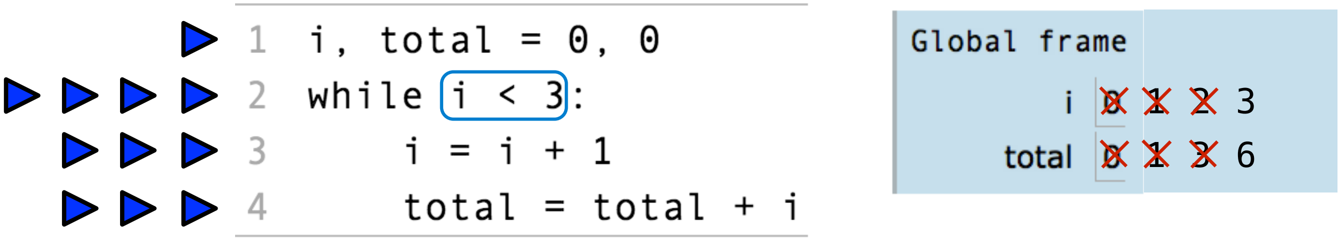
2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

While Statements

George Boole

(Demo)



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

Global frame

i ~~0~~ ~~1~~ ~~2~~ 3

total ~~0~~ ~~1~~ ~~3~~ 6

Execution rule for while statements:

1. Evaluate the header’s expression.

2. If it is a true value,
   execute the (*whole*) suite,
   then return to step 1.

Example: http://goo.gl/0d2cjF

Discussion Question

```
Complete the following definition by placing an expression in ________________________.
```

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

Discussion Question

Complete the following definition by placing an expression in ______________________________.

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

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

Discussion Question

Complete the following definition by placing an expression in ________________________.

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

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

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

Discussion Question

Complete the following definition by placing an expression in ________________________.

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

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

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ________________________.

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

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

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

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

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

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

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

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * ______________________, total - 1
    return ways
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \;\ldots\; \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \;\ldots\; \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * ___total // selected___, total - 1
    return ways
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \;\ldots\; \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \;\ldots\; \cdot 2 \cdot 1}$$

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

Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways *  total // selected  , total - 1
    return ways
```

$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

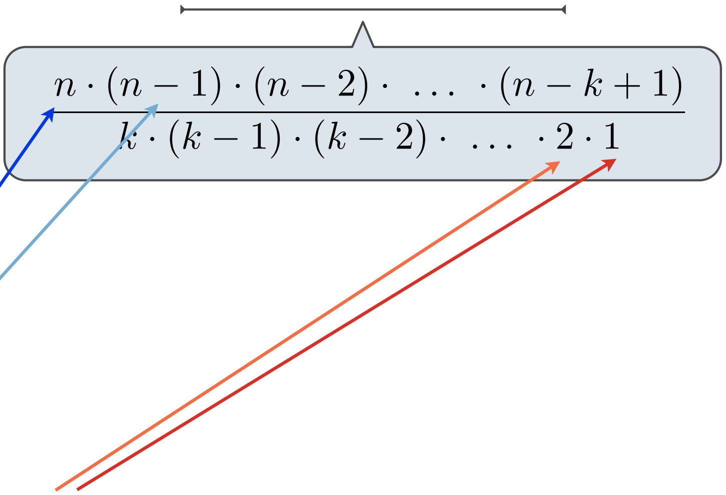
Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways * total // selected, total - 1
    return ways
```



$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \;\ldots\; \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \;\ldots\; \cdot 2 \cdot 1}$$

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

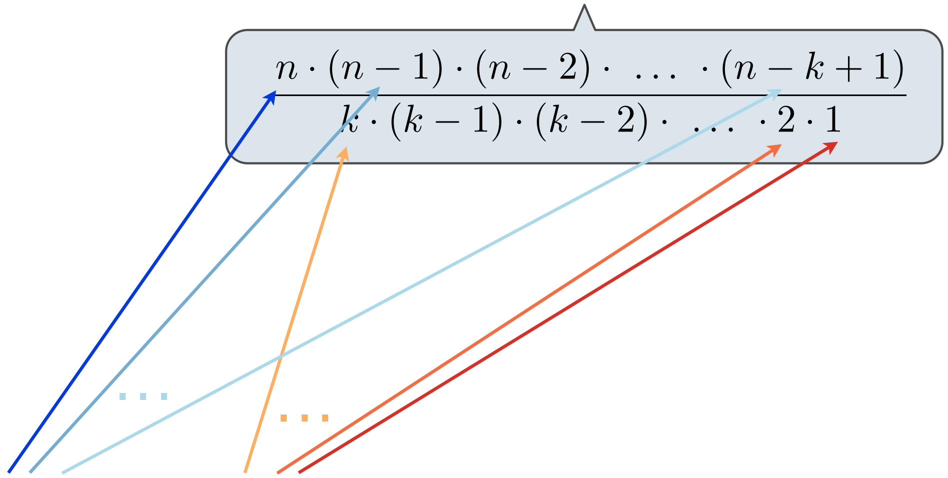
Discussion Question

Complete the following definition by placing an expression in ______________________.

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

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

    >>> choose(5, 2)
    10
    >>> choose(20, 6)
    38760
    """
    ways = 1
    selected = 0
    while selected < selection:
        selected = selected + 1
        ways, total = ways *  total // selected , total - 1
    return ways
```



$$\frac{n \cdot (n-1) \cdot (n-2) \cdot \ \ldots \ \cdot (n-k+1)}{k \cdot (k-1) \cdot (k-2) \cdot \ \ldots \ \cdot 2 \cdot 1}$$

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