Control

Announcements

Print and None

(Demo)

None Indicates that Nothing is Returned

The special value None represents nothing in Python

A function that does not explicitly return a value will return None

Careful: None is *not displayed* by the interpreter as the value of an expression

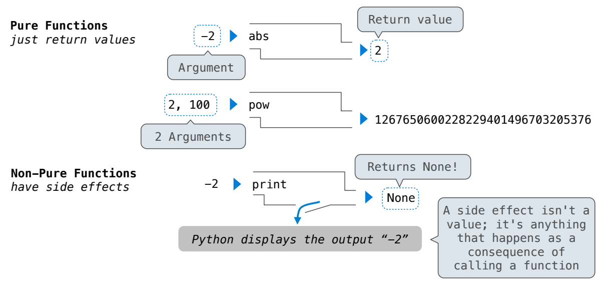
```
>>> def does_not_return_square(x):
...     x * x
...
>>> does_not_return_square(4)
>>> sixteen = does_not_return_square(4)
>>> sixteen + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

No **return**

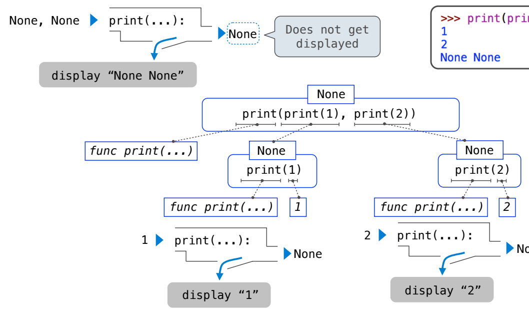
None value is not displayed

The name **sixteen** is now bound to the value **None**

Pure Functions & Non-Pure Functions



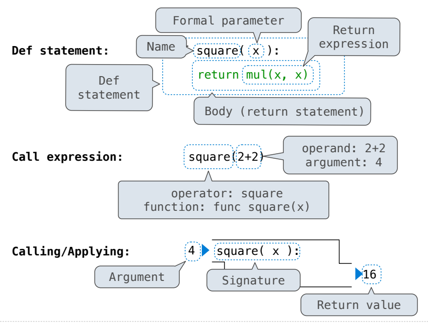
Nested Expressions with Print



```
>>> print(print(1), print(2))
1
2
None None
```

Multiple Environments

Life Cycle of a User-Defined Function



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)

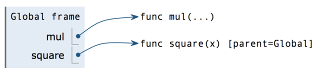
A new frame is created!

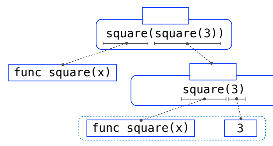
Parameters bound to arguments

Body is executed in that new environment

Multiple Environments in One Diagram!

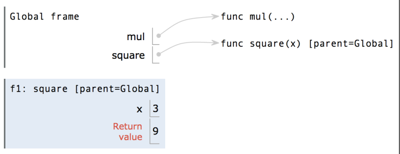
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```

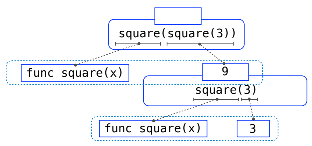




Multiple Environments in One Diagram!

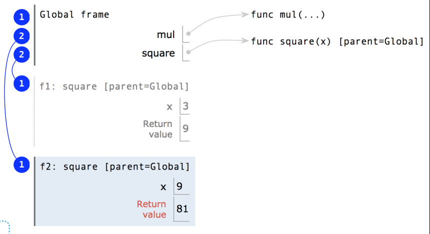
```
from operator import mul
def square(x):
    return mul(x, x)
square(square(3))
```



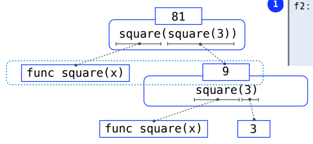


Multiple Environments in One Diagram!

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



1
2
2
1
1
Global frame
mul
square
func mul(...)
func square(x) [parent=Global]
f1: square [parent=Global]
x 3
Return value 9
f2: square [parent=Global]
x 9
Return value 81

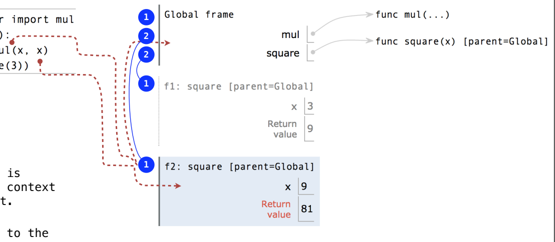


An environment is a sequence of frames.

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

Names Have No Meaning Without Environments

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



1
2
2
1
1
Global frame
mul
square
func mul(...)
func square(x) [parent=Global]
f1: square [parent=Global]
x 3
Return value 9
f2: square [parent=Global]
x 9
Return value 81

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.

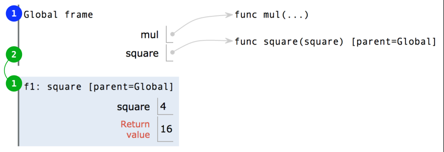
An environment is a sequence of frames.

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

Names Have Different Meanings in Different Environments

A call expression and the body of the function being called are evaluated in different environments

```
from operator import mul
def square(square):
    return mul(square, square)
square(4)
```



1
Global frame
2
1
mul
square
func mul(...)
func square(square) [parent=Global]
f1: square [parent=Global]
square 4
Return value 16

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.

Miscellaneous Python Features

Division
Multiple Return Values
Source Files
Doctests
Default Arguments

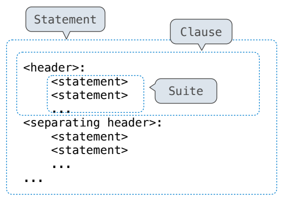
(Demo)

Conditional Statements

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

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

```
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:

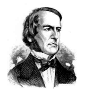
Each clause is considered in order.

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

Syntax Tips:

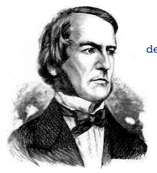
1. Always starts with "if" clause.
2. Zero or more "elif" clauses.
3. Zero or one "else" clause, always at the end.

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
```

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!

(Demo)

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

Iteration

While Statements

(Demo)

George Boole

```
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.

(Demo)

Example: Prime Factorization

Prime Factorization

Each positive integer n has a set of prime factors: primes whose product is n

```
...
8  = 2 * 2 * 2
9  = 3 * 3
10 = 2 * 5
11 = 11
12 = 2 * 2 * 3
...
```

One approach: Find the smallest prime factor of n, then divide by it

858 = 2 * 429 = 2 * 3 * 143 = 2 * 3 * 11 * 13

(Demo)