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**

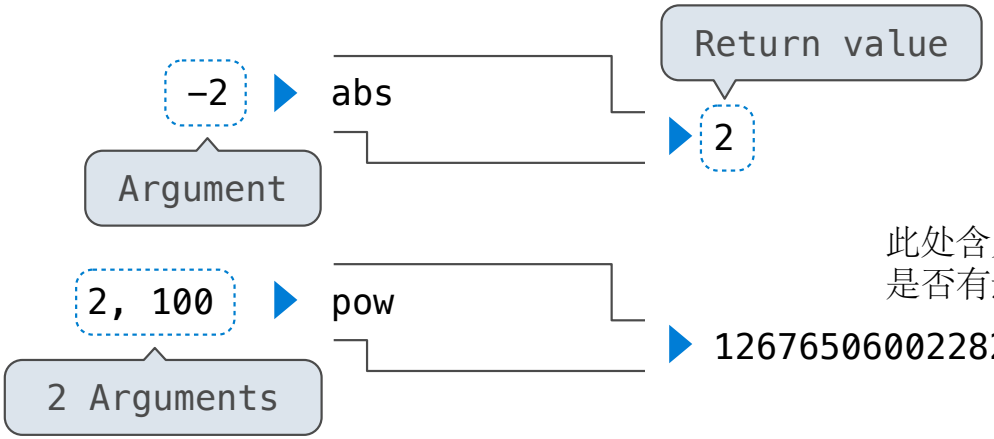
None value is not displayed

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

显然，None与int作为不同的数字类型，不能相加

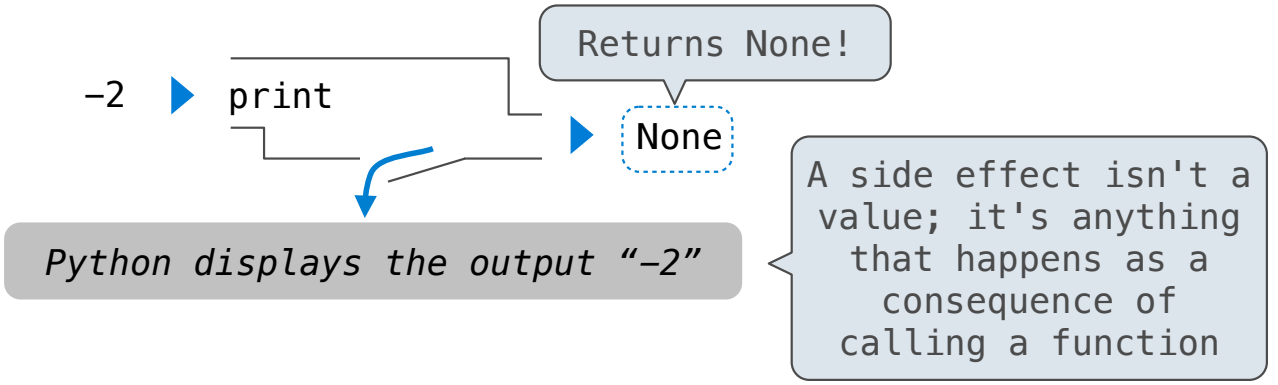
Pure Functions & Non-Pure Functions

Pure Functions
just return values

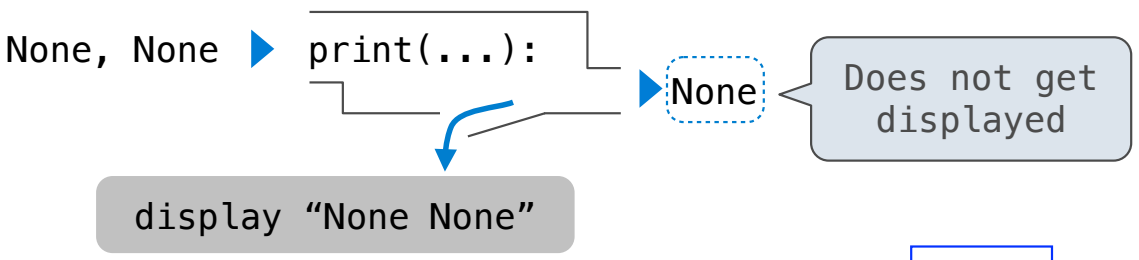


此处含义在于print与abs和pow函数之间是否有返回值return的不同

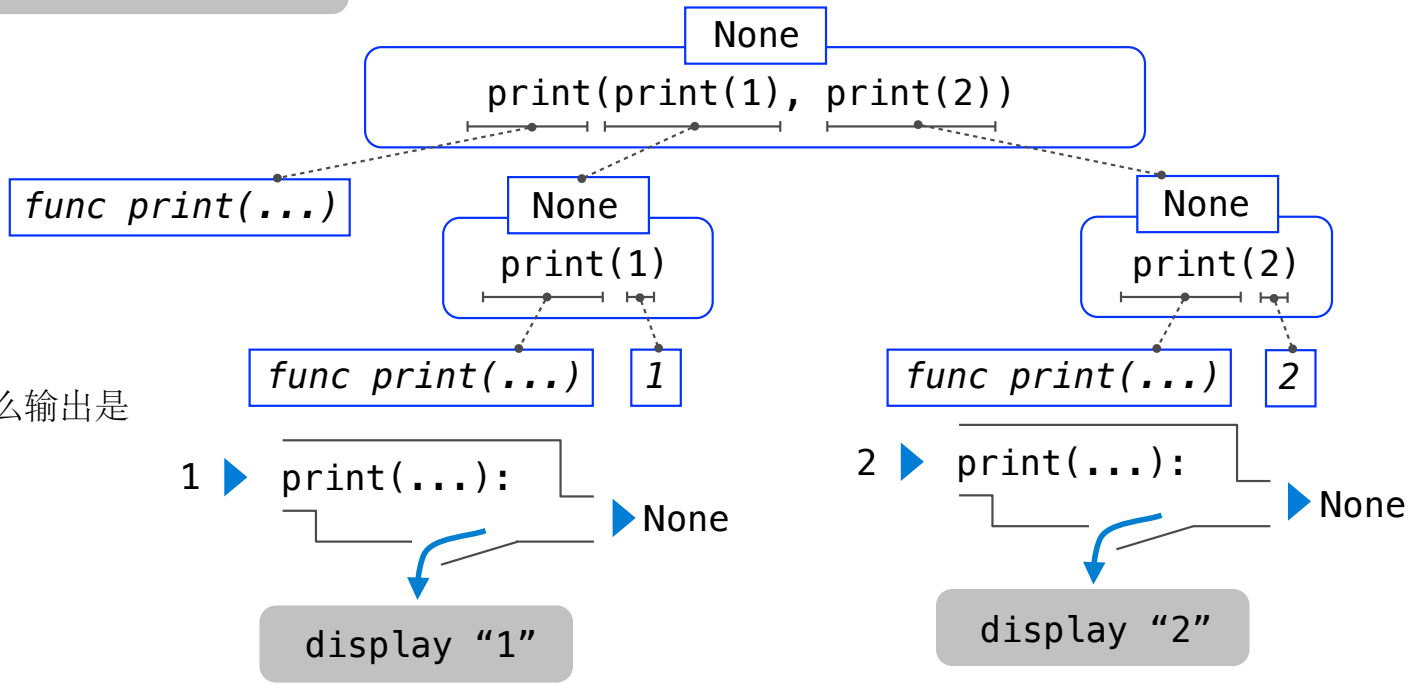
Non-Pure Functions
have side effects



Nested Expressions with Print




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

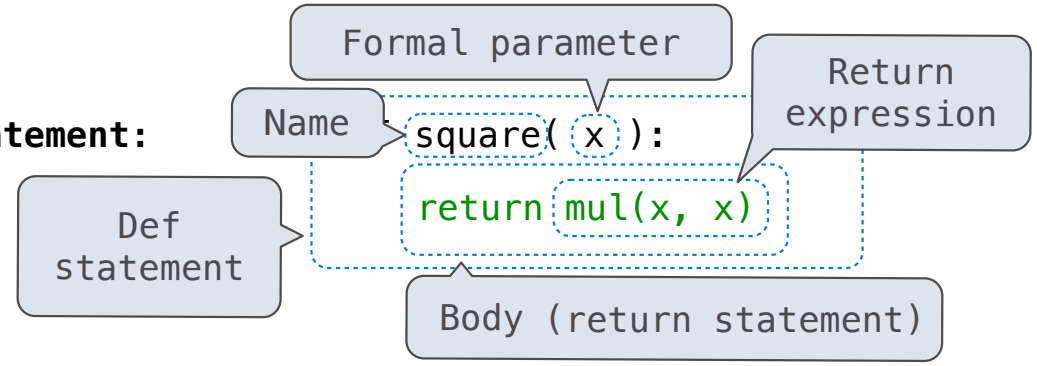


注意理解此处为什么输出是如此

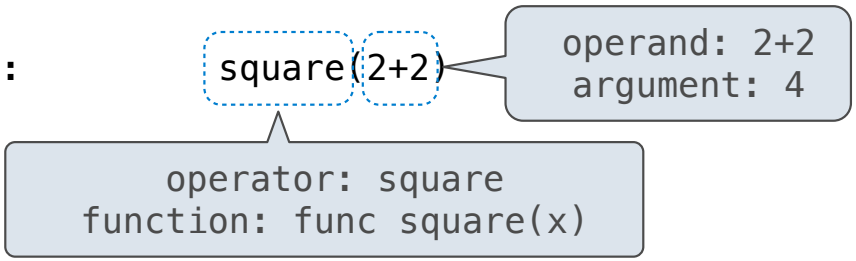
Multiple Environments

Life Cycle of a User-Defined Function

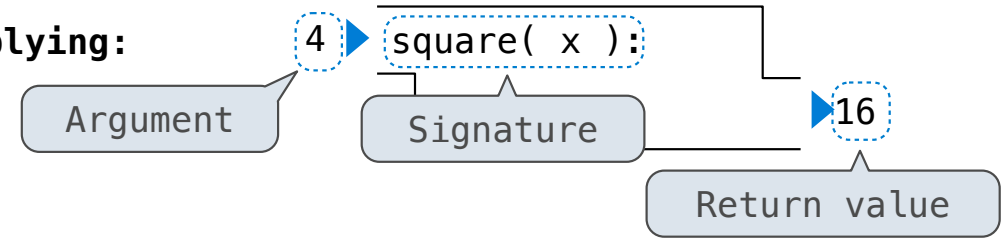
Def statement:



Call expression:



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)

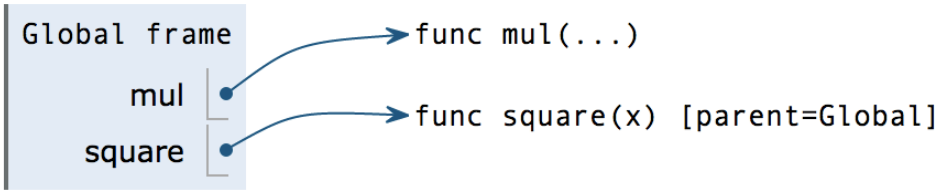
A new frame is created!

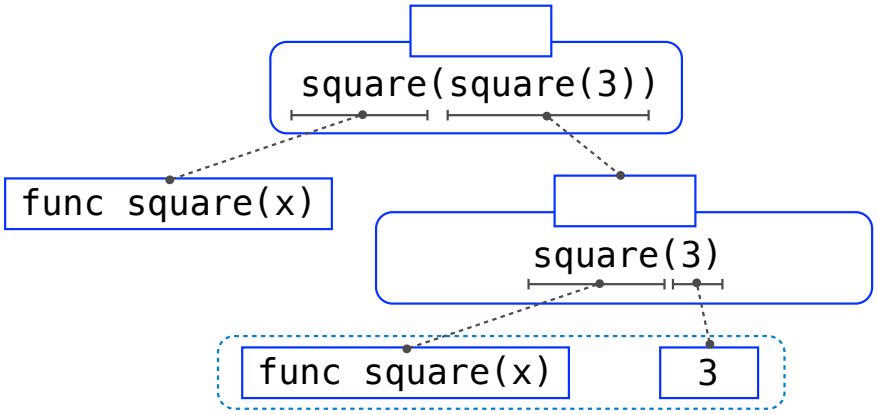
Parameters bound to arguments

Body is executed in that new environment

Multiple Environments in One Diagram!

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

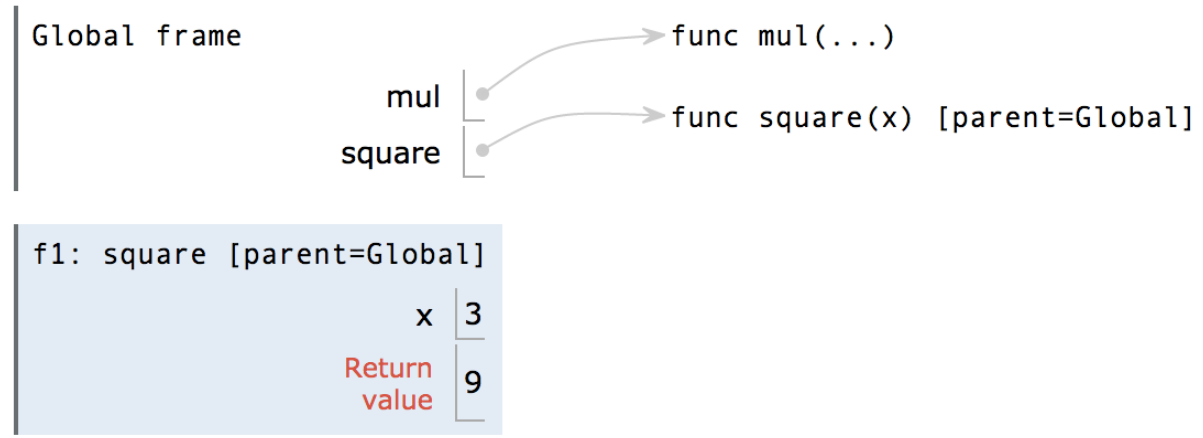


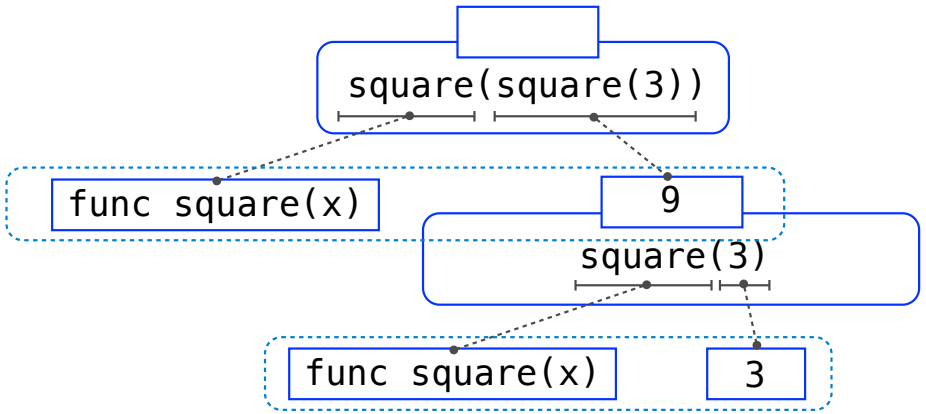


http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28square%283%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Multiple Environments in One Diagram!

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

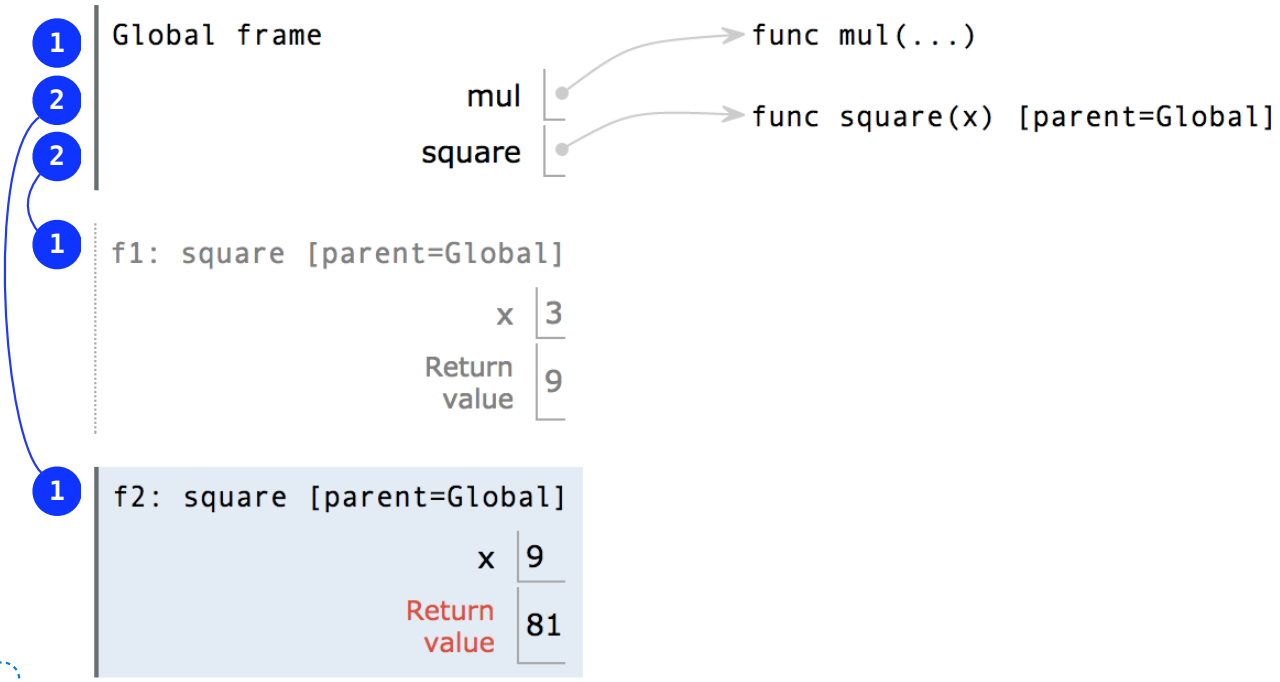


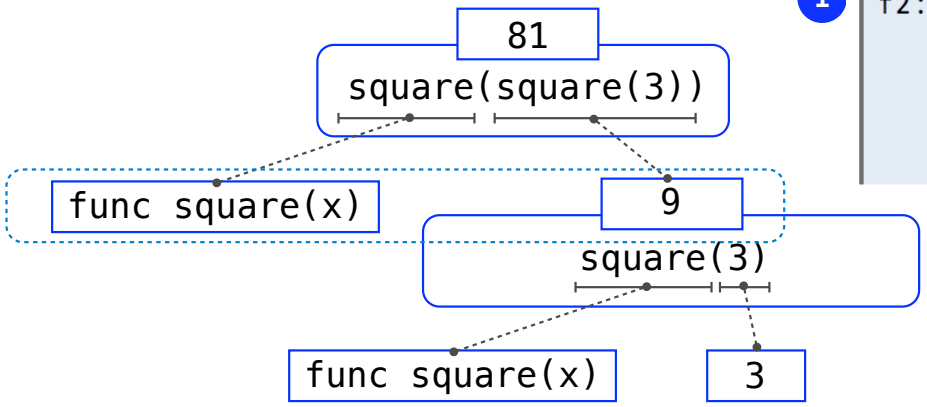


http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28square%283%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Multiple Environments in One Diagram!

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



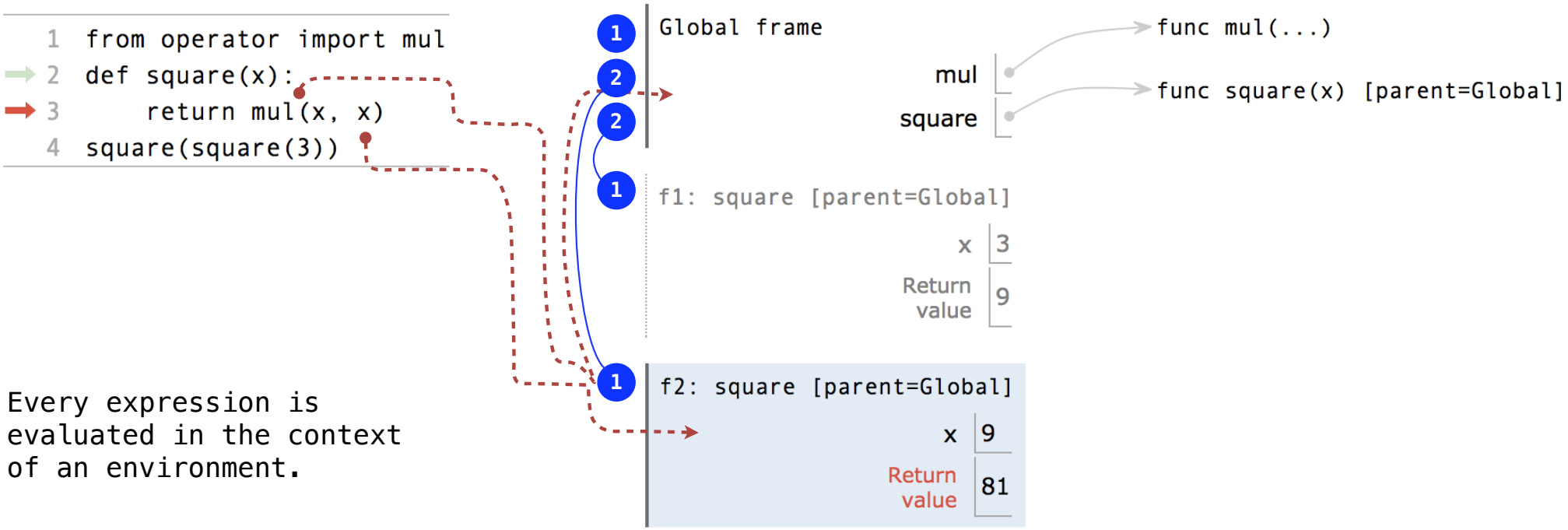


An environment is a sequence of frames.

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

http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28square%283%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Names Have No Meaning Without Environments



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.

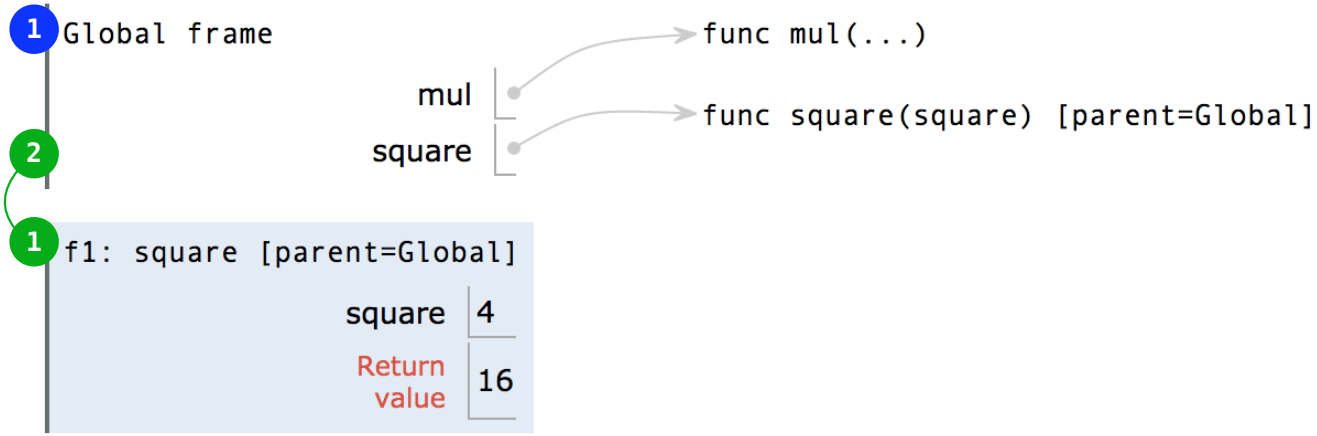
- The global frame alone
- A local, then the global frame

http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28square%283%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Names Have Different Meanings in Different Environments

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

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



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.

http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28square%29%3A%0A%20%20%20%20return%20mul%28square,%20square%29%0Asquare%284%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

2013/10　浮点除法
2013//10　舍弃小数除法
2013%10　取余数

/ ：　truediv（）
// :　floordiv()
% : mod()

```
def divide_exact(n, d):
    return n // d, n % d
quotient, remainder = divide_exact(618, 10)

>>>print(quotient,remainder)
201
3
```

一些python相关的命令

```
yang@DESKTOP-PT0FVKI:~$ python3 Myself.py
201 3
yang@DESKTOP-PT0FVKI:~$ python3 -i Myself.py
201 3
>>> q
201
>>> r
3
>>> divide_exact
<function divide_exact at 0x7f9ff104fd90>
>>> divide_exact(2013,10)
(201, 3)
>>>
```

source file：

```
from operator import floordiv,mod
def divide_exact(n,d):
    return floordiv(n,d),mod(n,d)
```

how to use doctest??

```
def divide_exact(n, d=10):
    """Return the quotient and remainder of dividing N by D.

    >>> quotient, remainder = divide_exact(618, 10)
    >>> quotient
    61
    >>> remainder
    8
    """
    return floordiv(n, d), mod(n, d)
```

And then use
the command:python3 -m doctest -v 03.py　to test the code.

Miscellaneous Python Features

Division
Multiple Return Values
Source Files
Doctests
Default Arguments

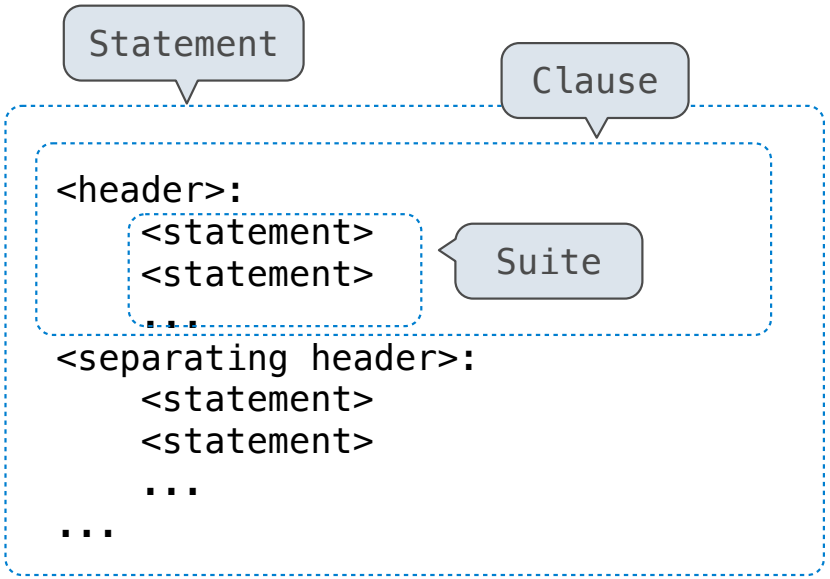
(Demo)

Conditional Statements

Statements

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

Compound statements:



The first header determines a statement's type

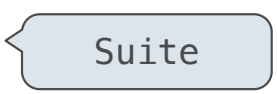
The header of a clause "controls" the suite that follows

def statements are compound statements

Compound Statements

Compound statements:

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



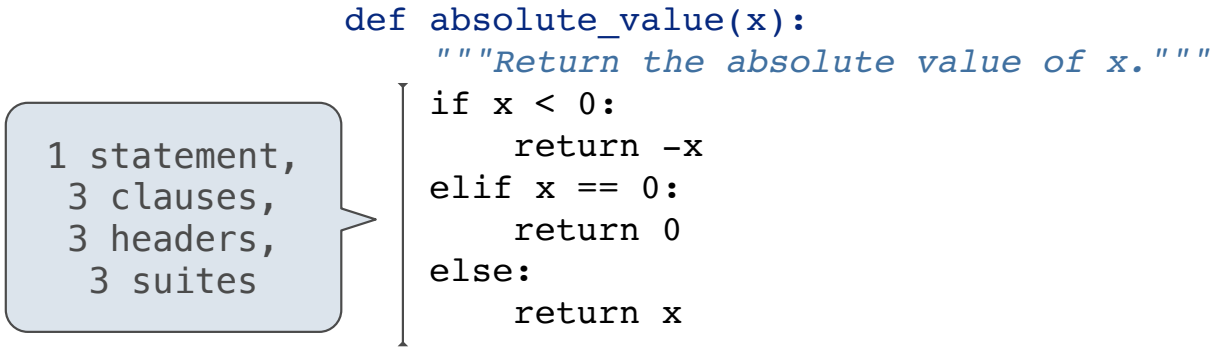
A suite is a sequence of statements

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



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:

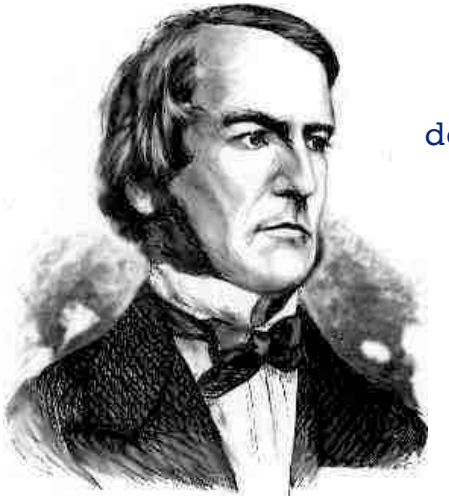
1. Always starts with "if" clause.
2. Zero or more "elif" clauses.
3. Zero or one "else" clause,
   always at the end.

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

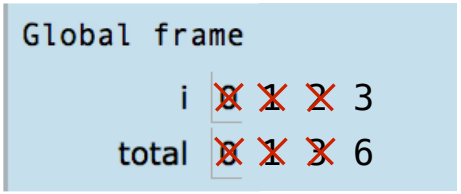
Iteration

While Statements

George Boole

(Demo)

```
1  i, total = 0, 0
2  while i < 3:
3      i = i + 1
4      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.

(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)