61A Lecture 3

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**

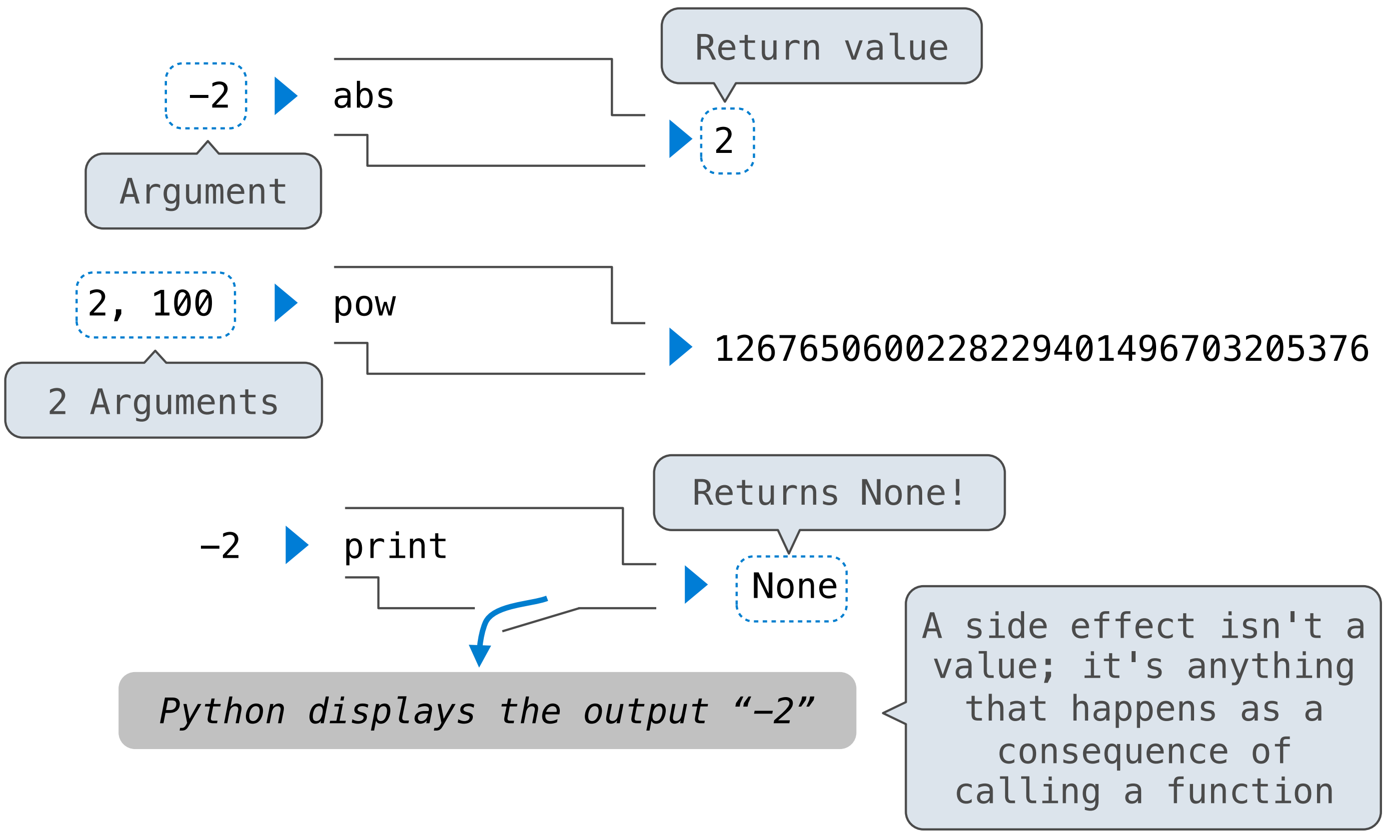
None value is not displayed

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

Pure Functions & Non-Pure Functions

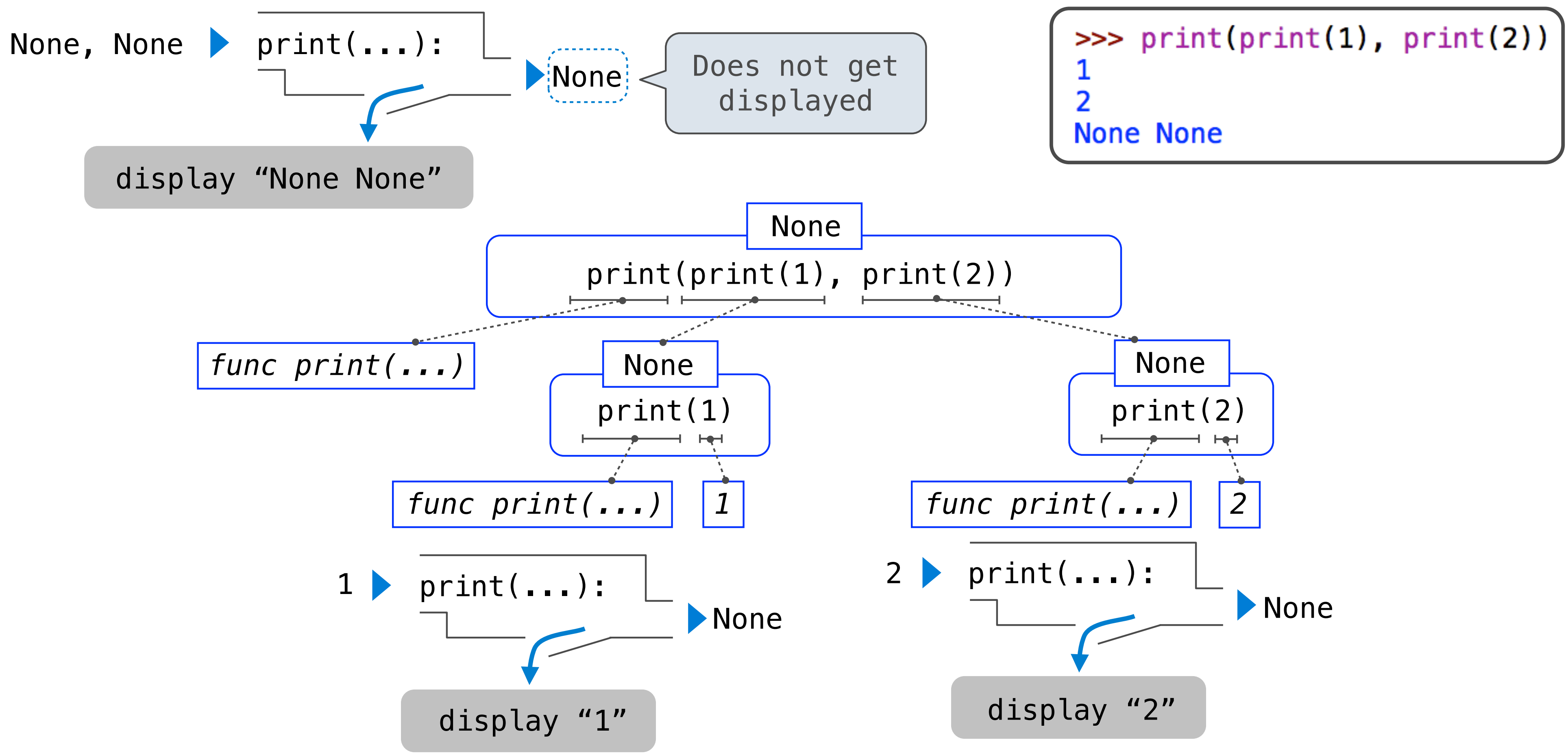
Pure Functions
just return values

Non-Pure Functions
have side effects



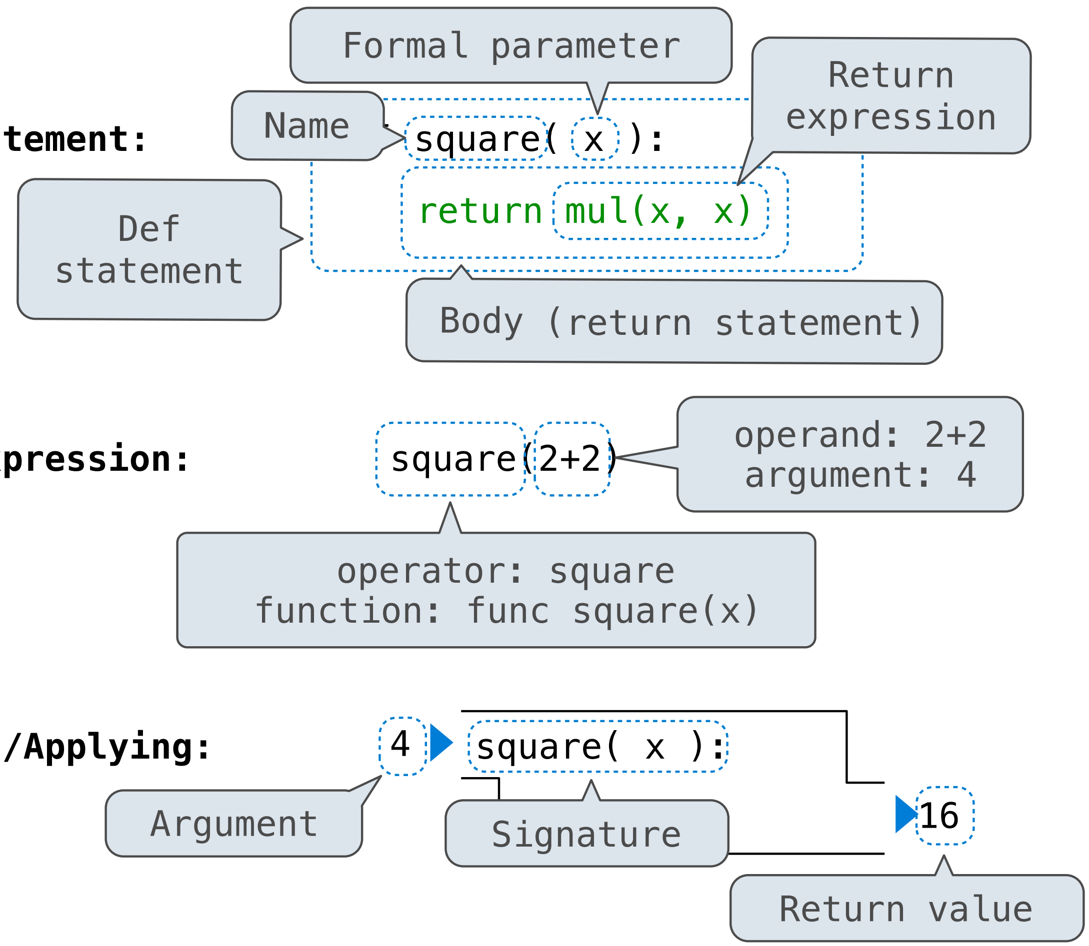
(Demo)

Nested Expressions with Print



Multiple Environments

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:

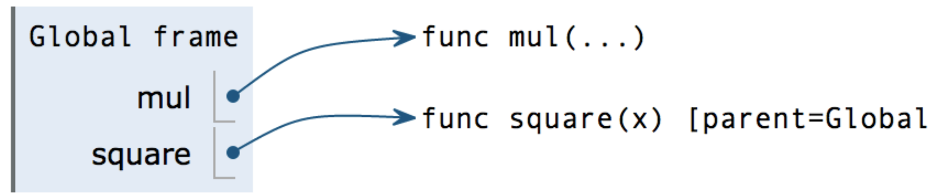
A new frame is created!

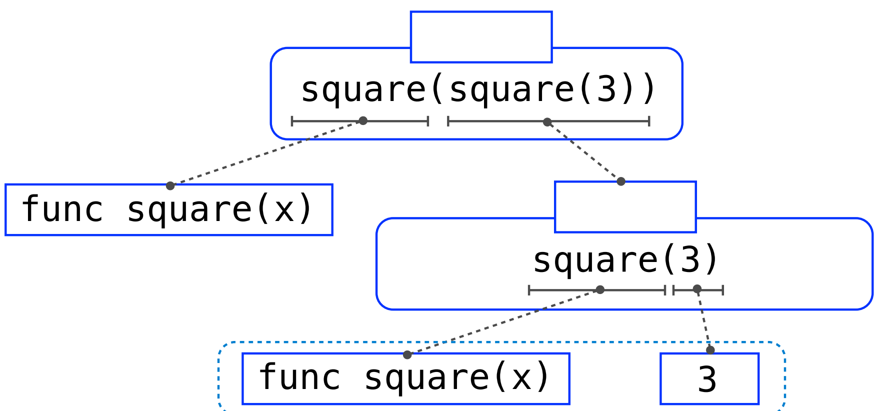
Parameters bound to arguments

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))
```

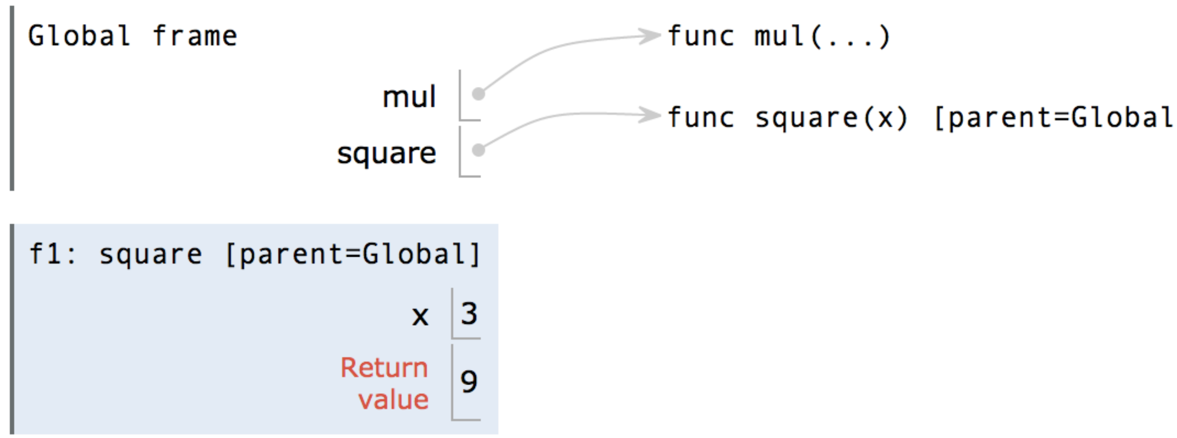


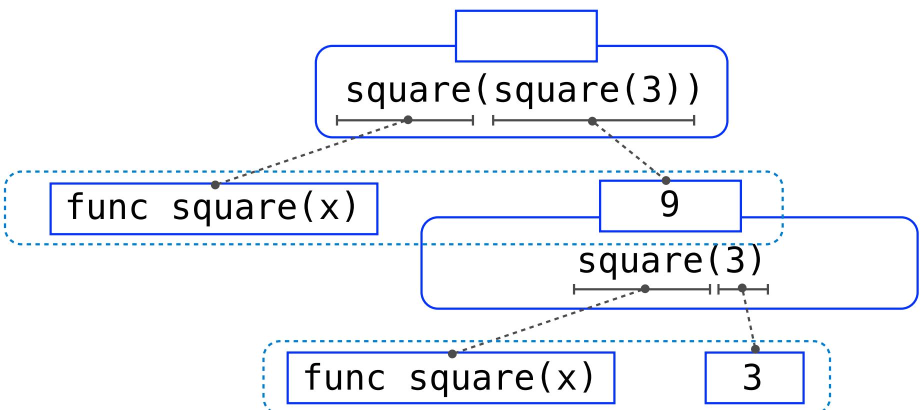


Interactive Diagram

Multiple Environments in One Diagram!

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

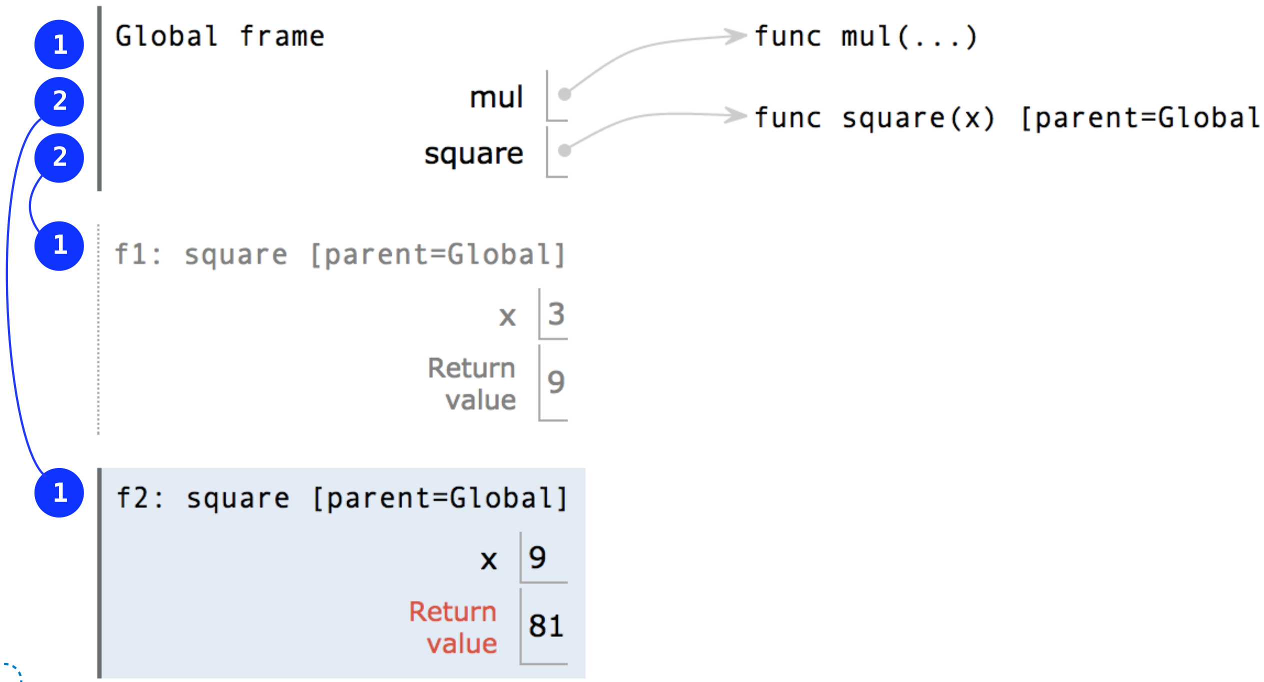


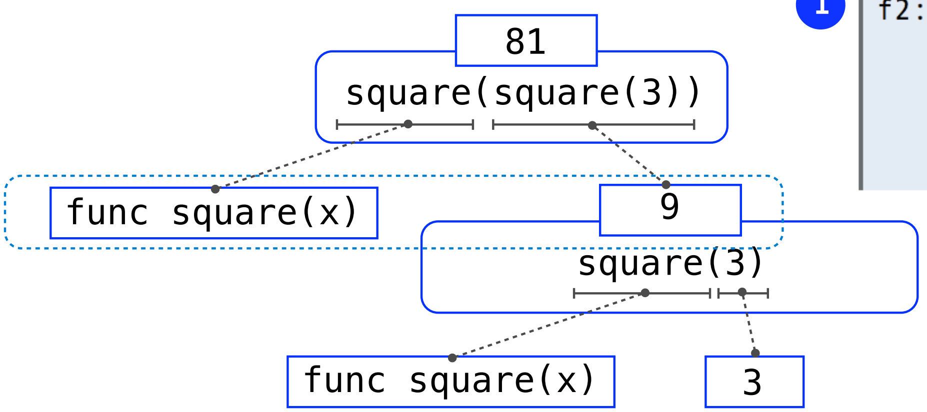


Interactive Diagram

Multiple Environments in One Diagram!

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



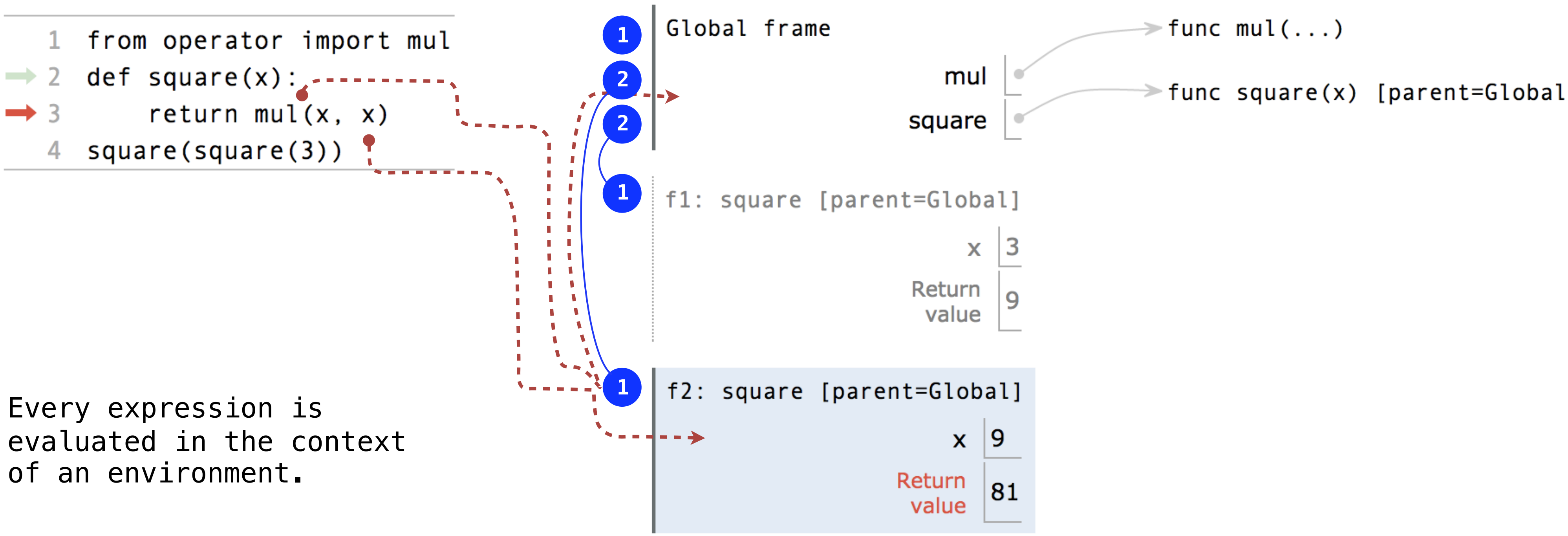


An environment is a sequence of frames.

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

Interactive Diagram

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.

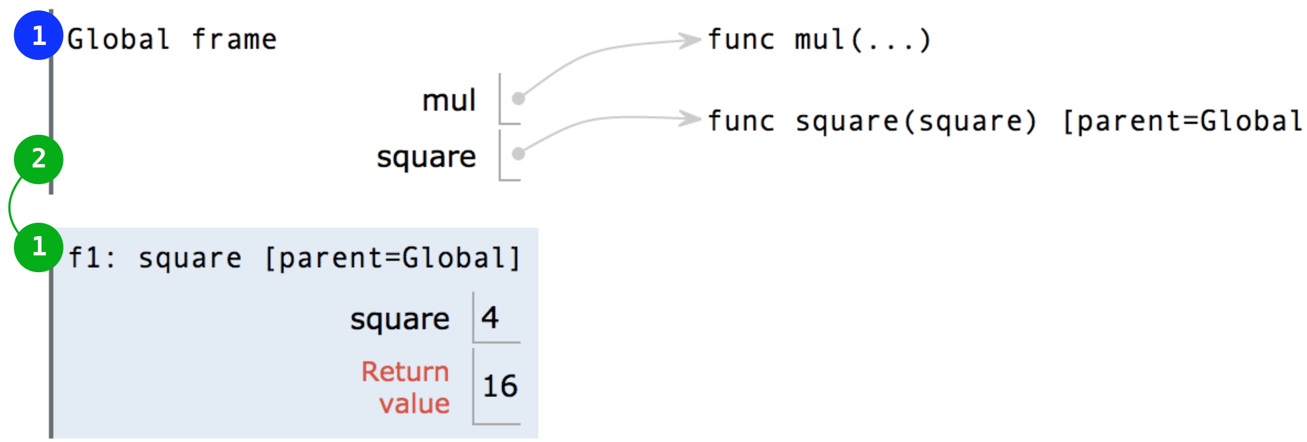
- The global frame alone
- A local, then the global frame

Interactive Diagram

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)
```



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.

Interactive Diagram

Miscellaneous Python Features

Division

Multiple Return Values

Source Files

Doctests

Default Arguments

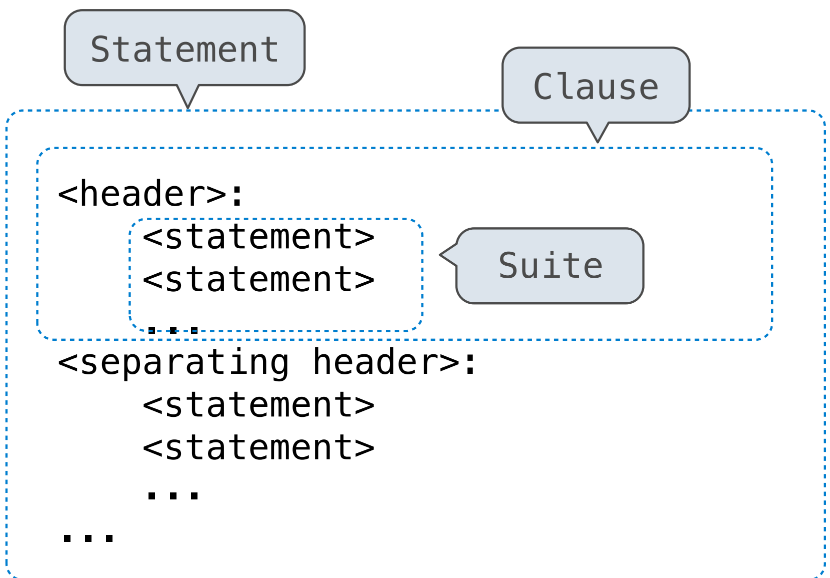
(Demo)

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>
    ...
...
```

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)

```
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:

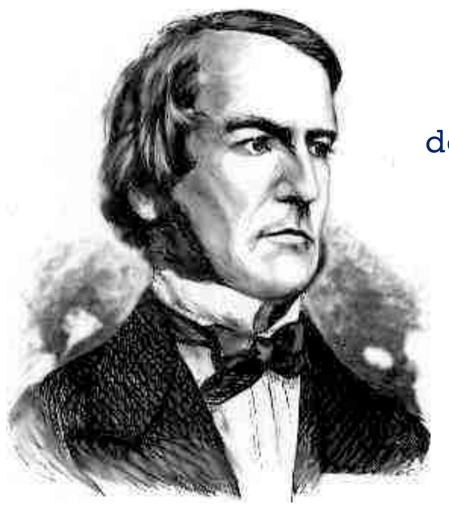
1. Always starts with "if" clause.
2. Zero or more "elif" clauses.
3. Zero or one "else" clause,
   always at the end.

Boolean Contexts

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

George Boole

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

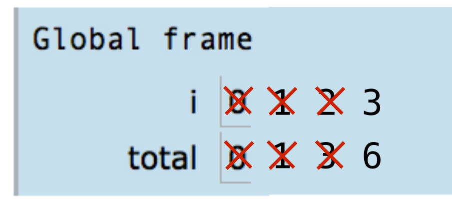
Iteration

While Statements

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.