Lecture 3: Control

June 23, 2022
Laryn Qi

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'
```

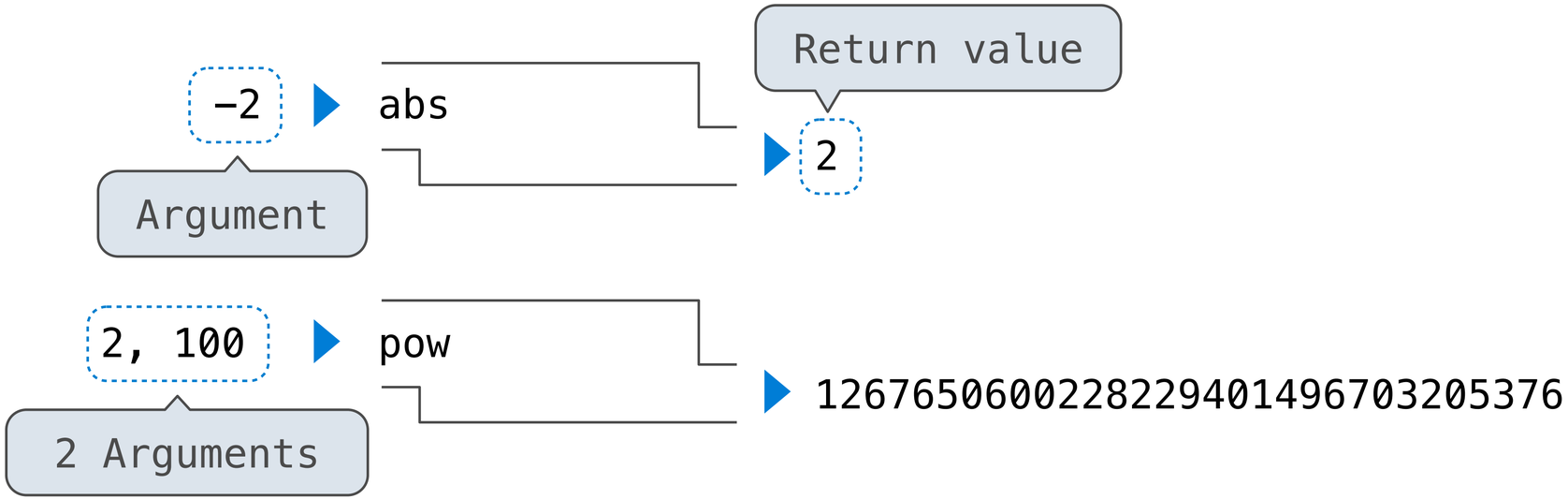
No **return**

None value is not displayed

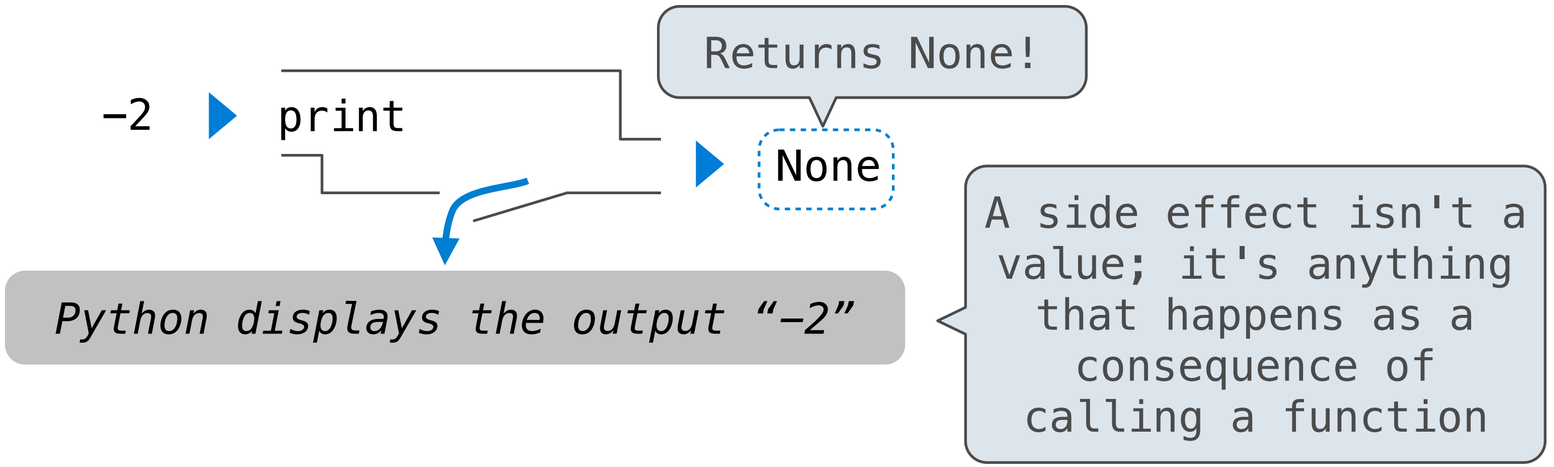
The name **sixteen** is now bound to the value **None**

Pure Functions & Non-Pure Functions

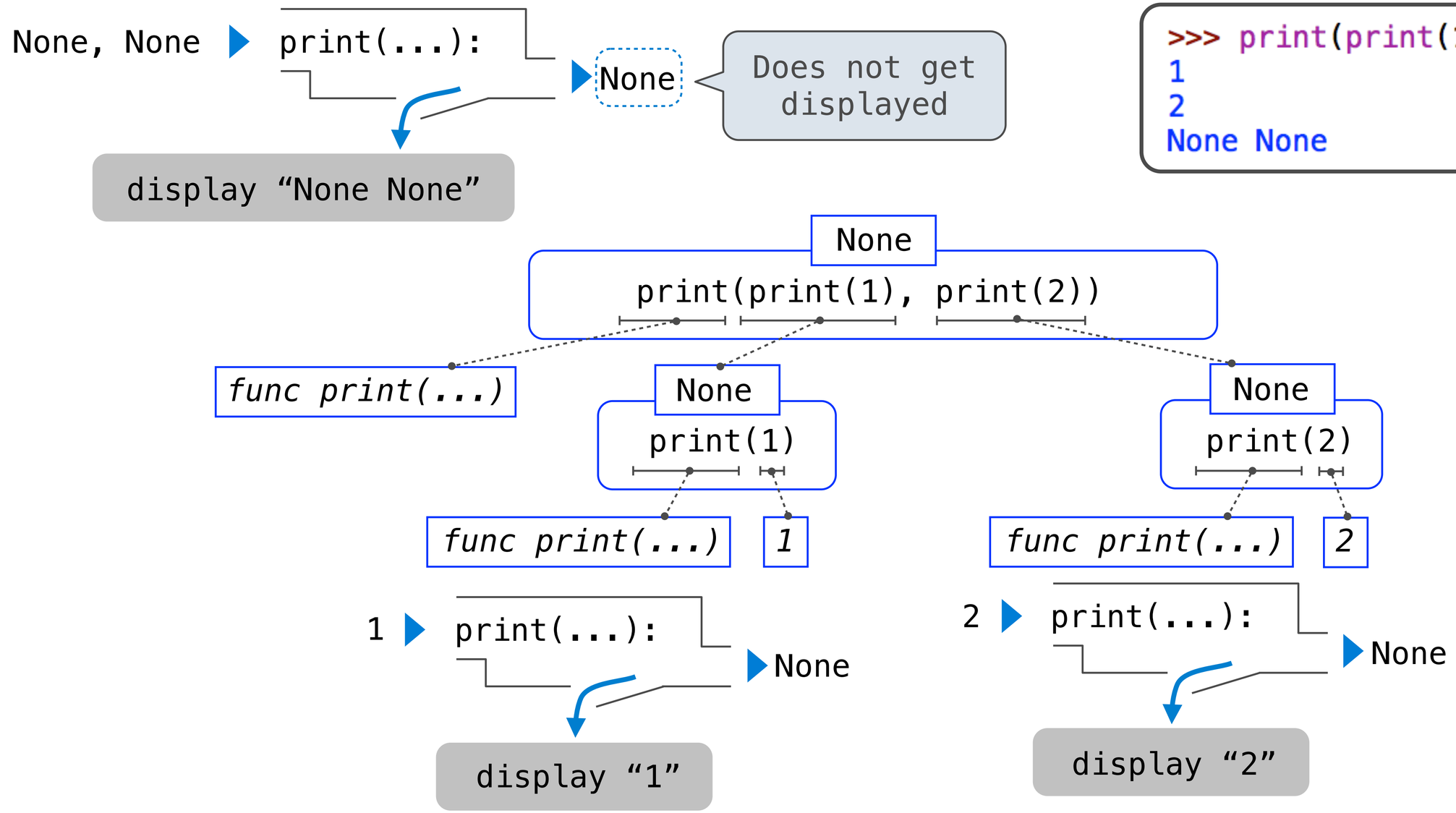
Pure Functions
just return values



Non-Pure Functions
have side effects



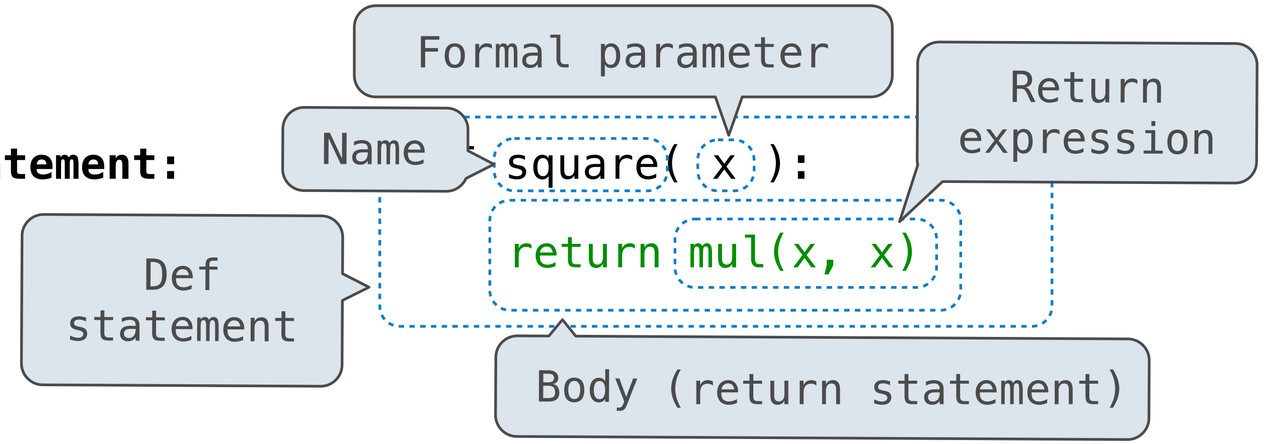
Nested Expressions with Print



Multiple Environments

Life Cycle of a User-Defined Function

Def statement:

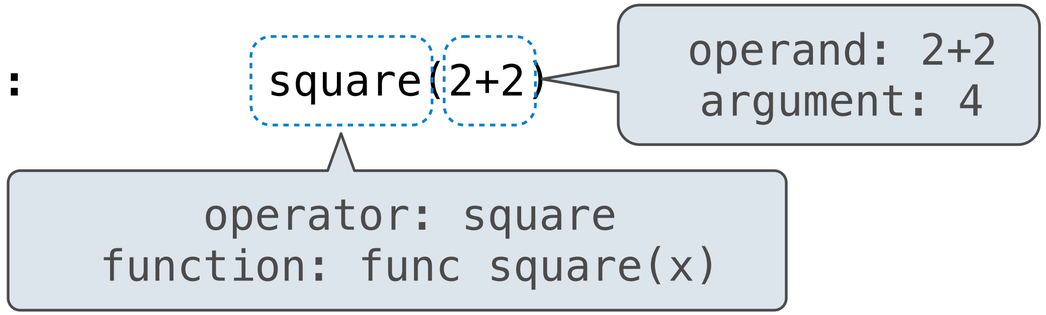


What happens?

A new function is created!

Name bound to that function in the current frame

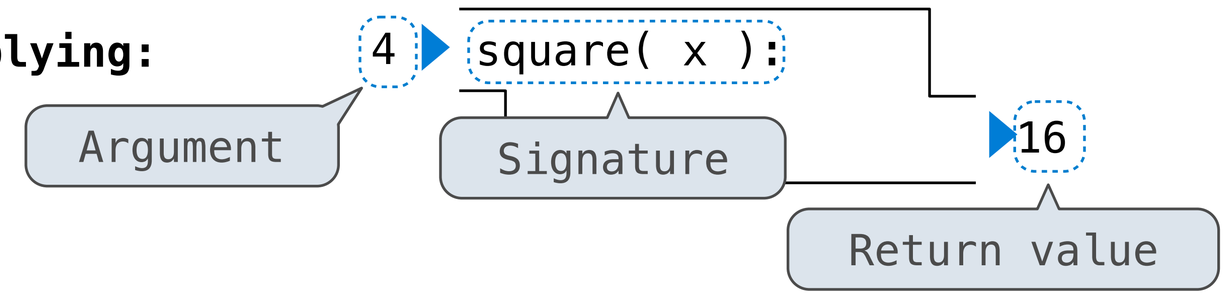
Call expression:



Operator & operands evaluated

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

Calling/Applying:



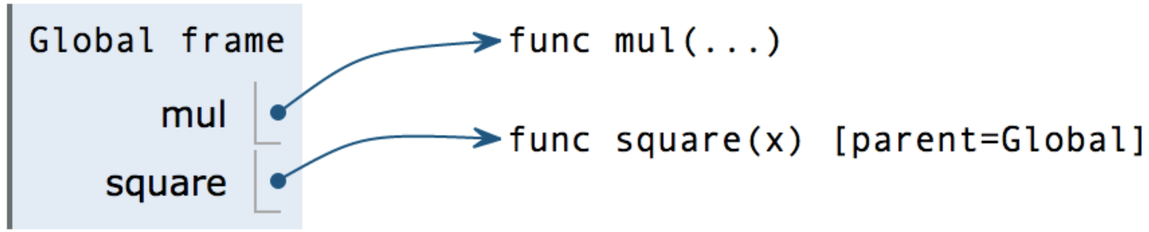
A new frame is created!

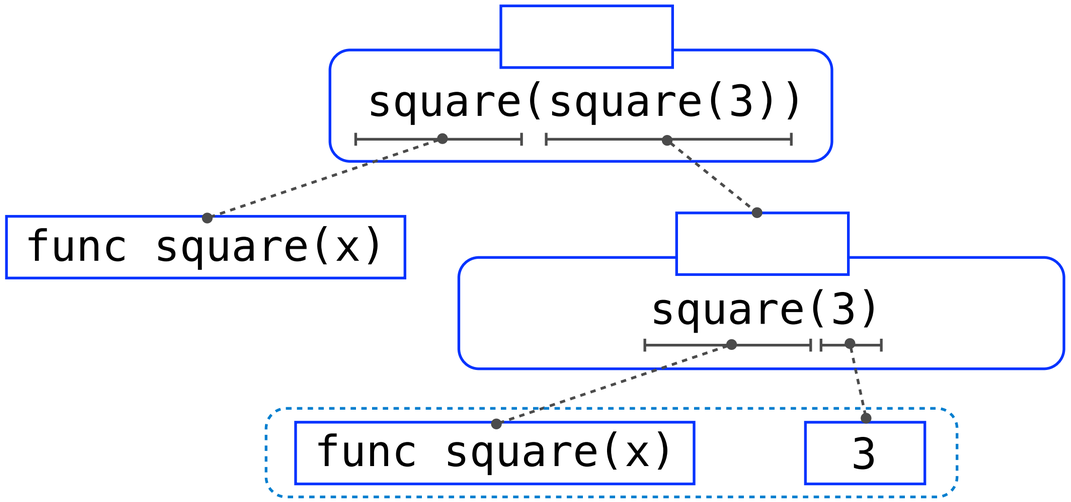
Parameters bound to arguments

Body is executed in that new environment

Multiple Environments in One Diagram!

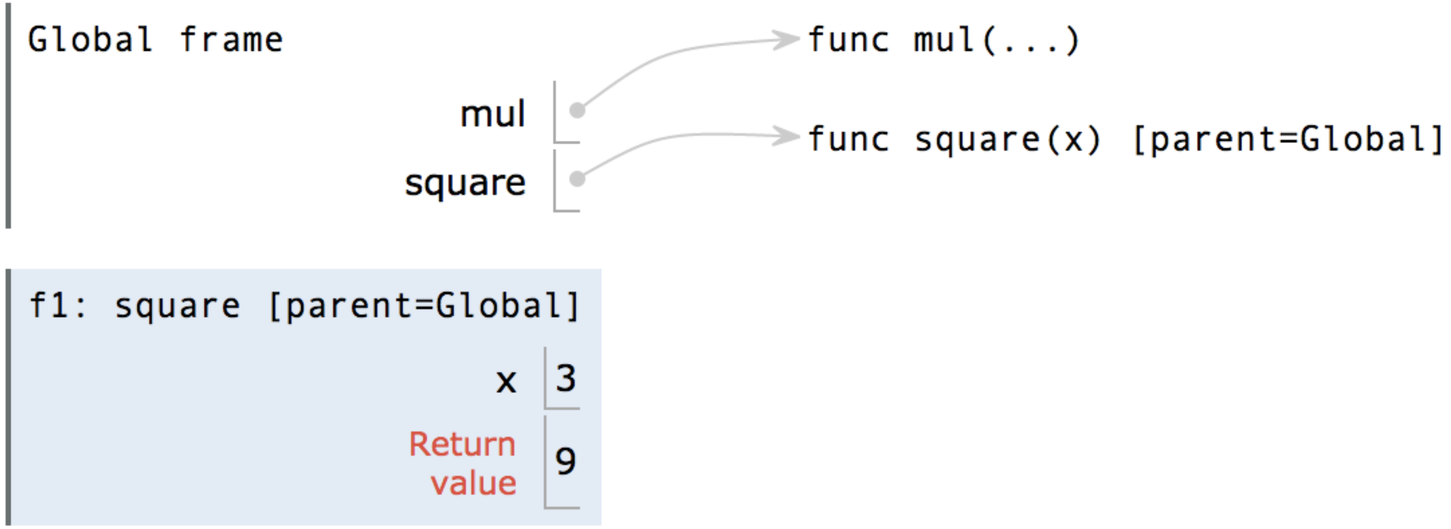
```
1  from operator import mul
2  def square(x):
3      return mul(x, x)
4  square(square(3))
```

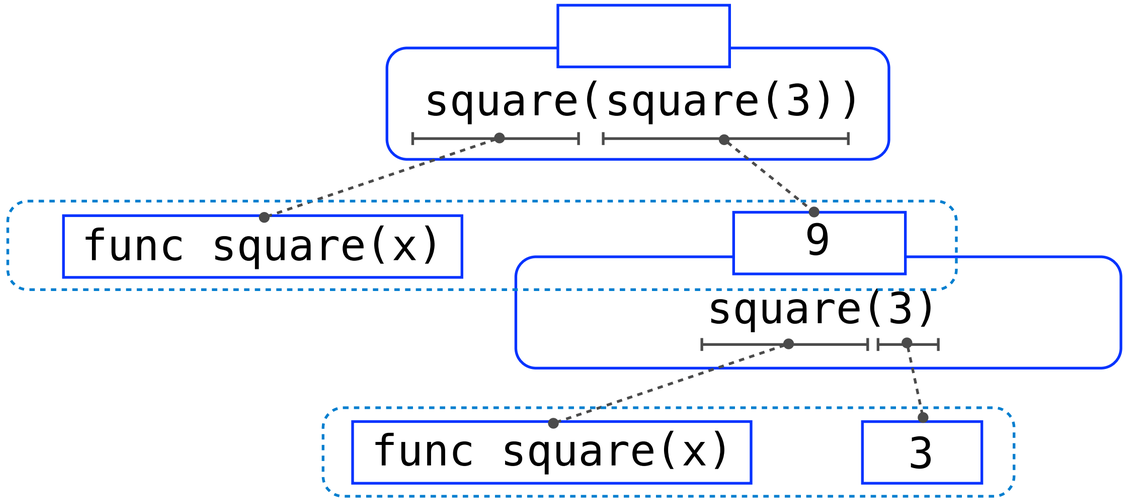




Multiple Environments in One Diagram!

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

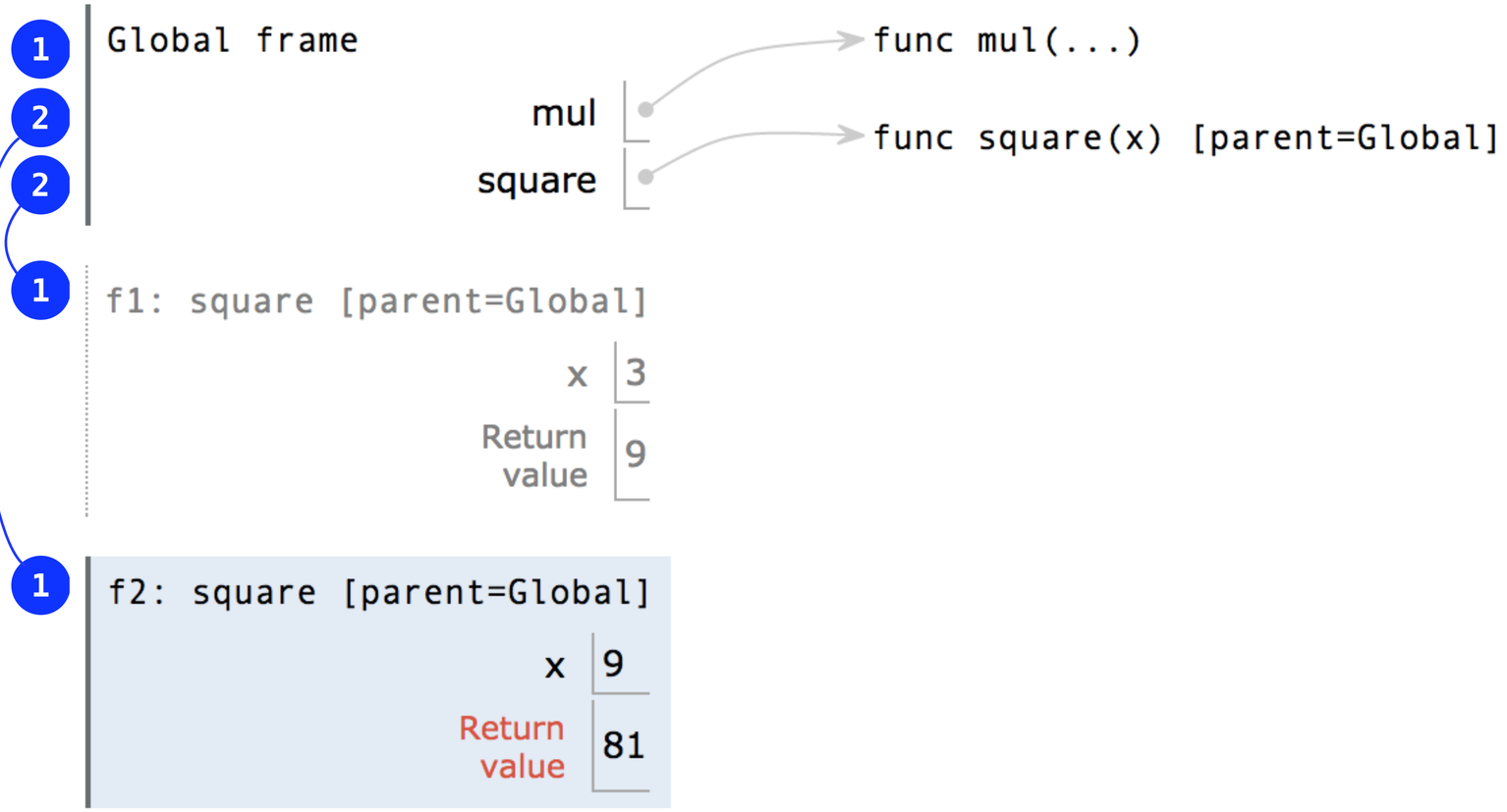


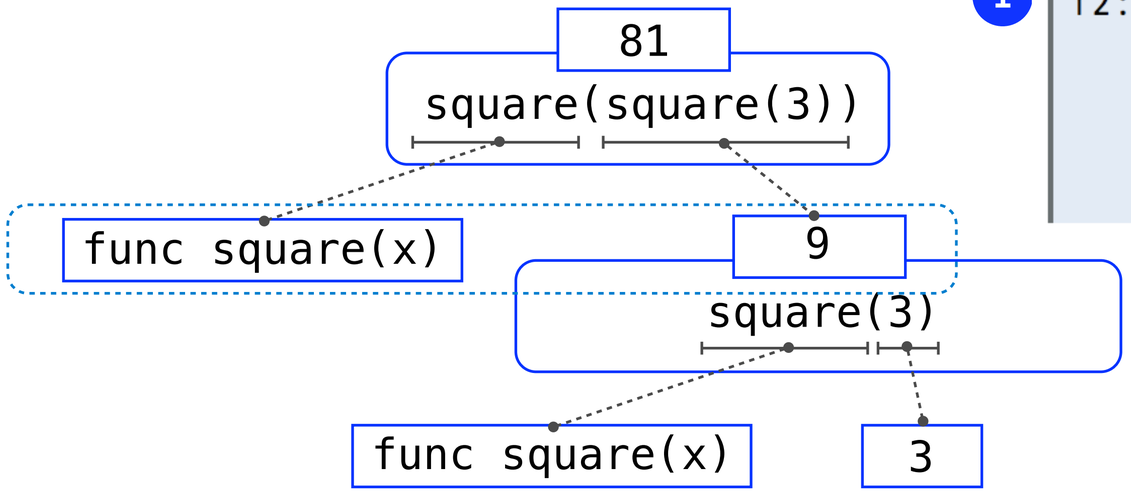


http://pythontutor.com/composingprograms.html#code=from%20operator%20import%20mul%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20mul%28x,%20x%29%0Asquare%28square%283%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Multiple Environments in One Diagram!

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

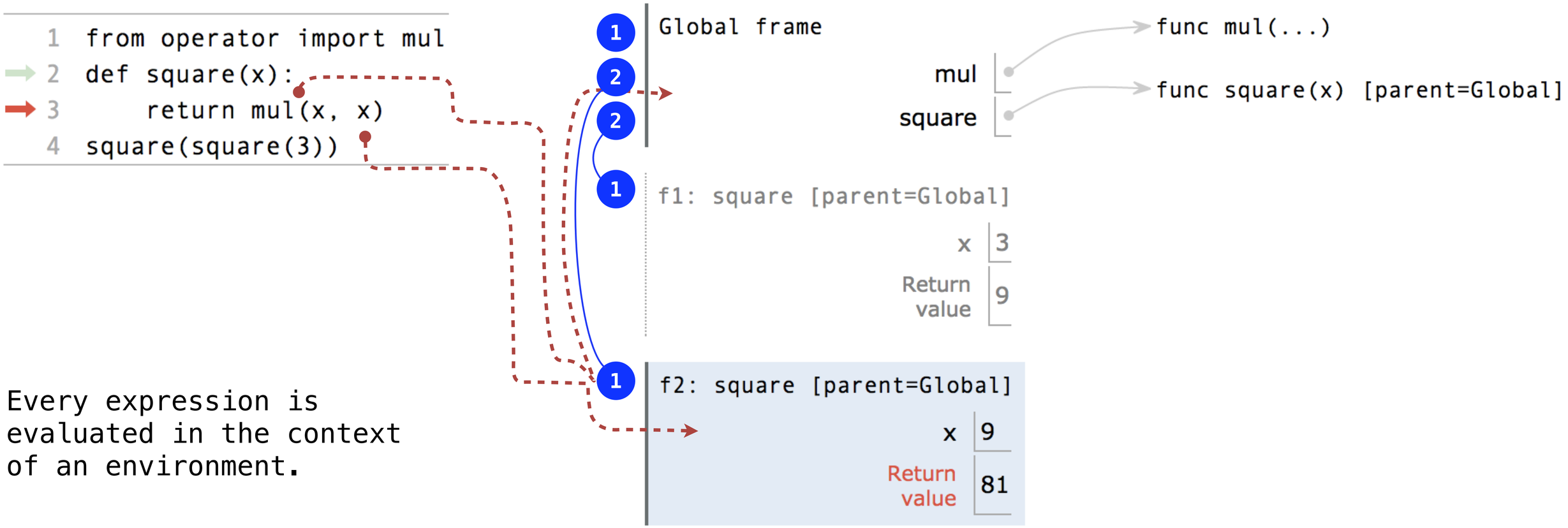




An environment is a sequence of frames.

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

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.

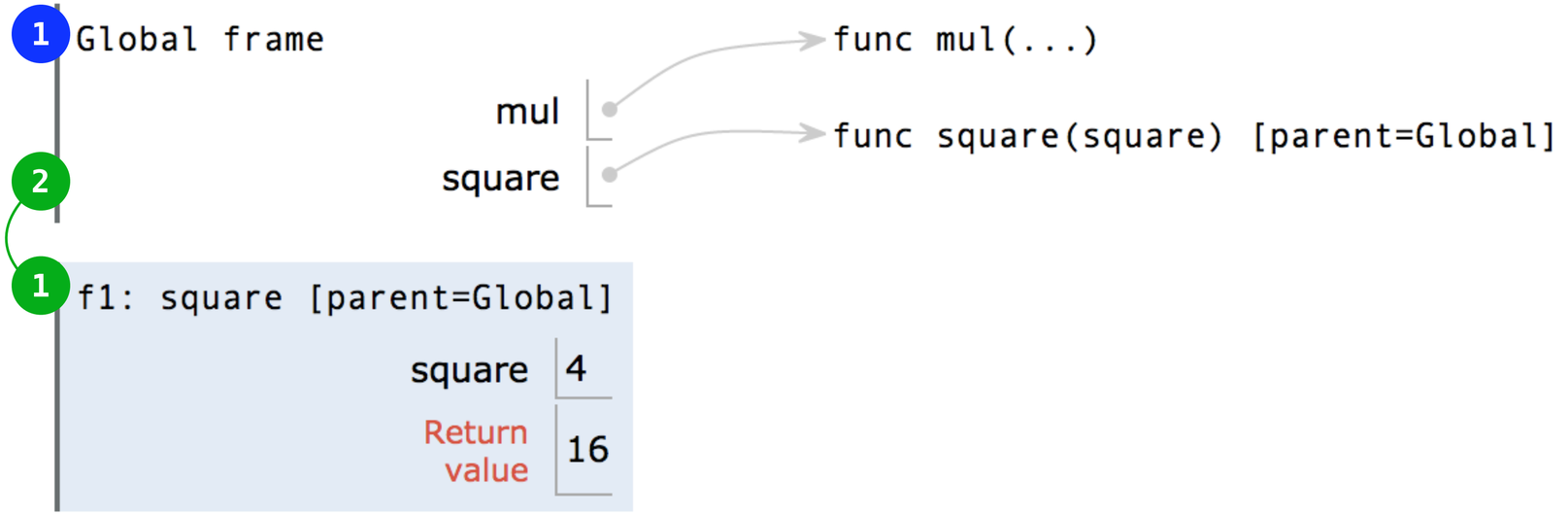
- The global frame alone
- A local, then the global frame

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

Miscellaneous Python Features

Division

Multiple Return Values

Source Files

Doctests

Default Arguments

(Demo)

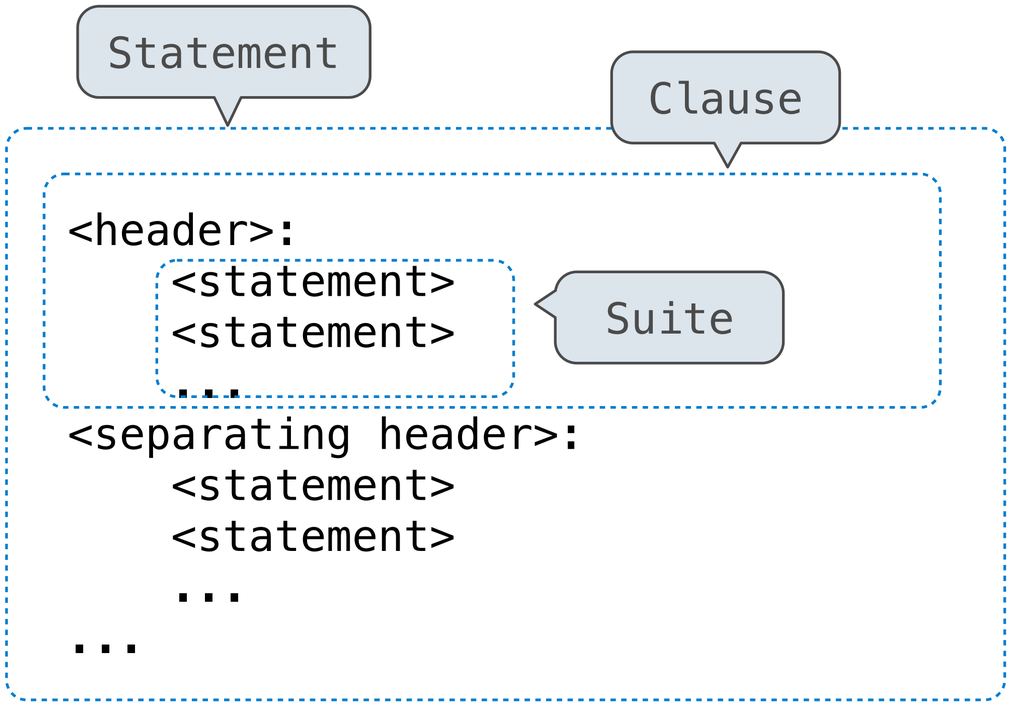
Break

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

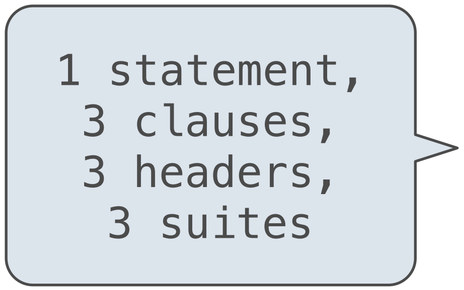
A suite is a sequence of statements

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




```
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.

Syntax Tips:

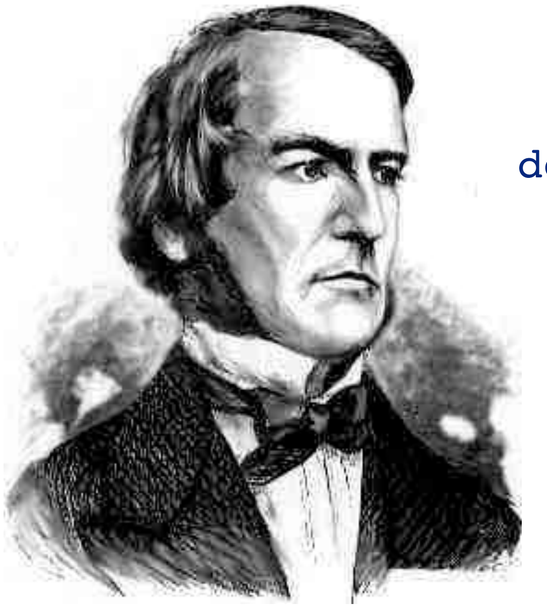
1. Always starts with "if" clause.
2. Zero or more "elif" clauses.
3. Zero or one "else" clause,
   always at the end.

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

Truth-y values in Python: Anything else (True)

Read Section 1.5.4!

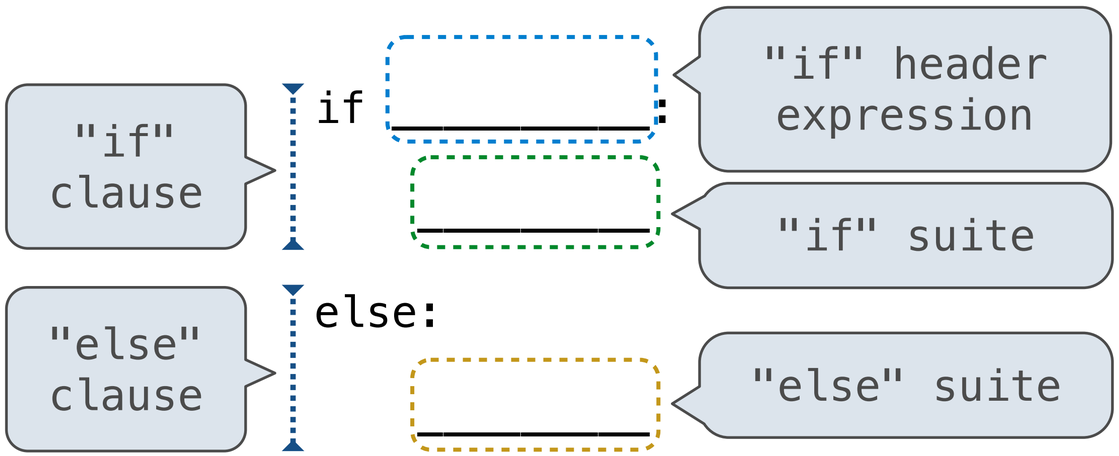
Conditional Statements Practice

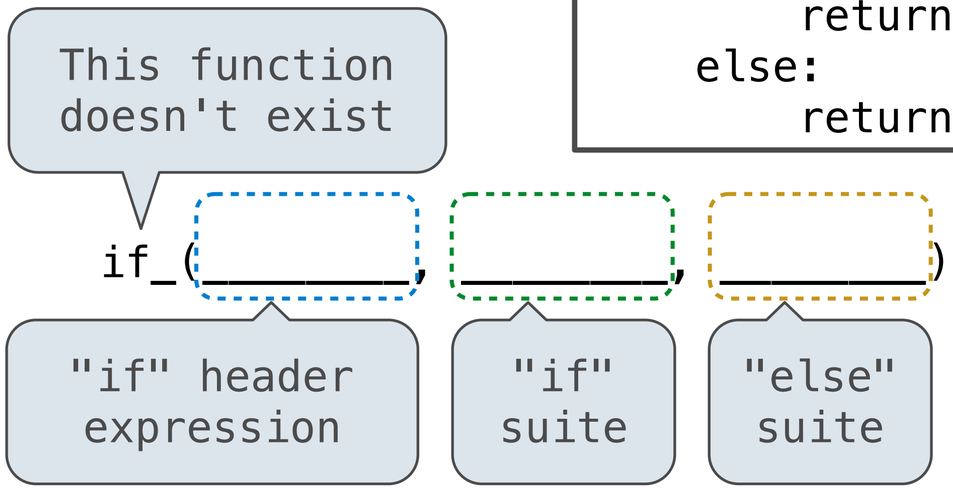

Always has been
Wait, it's all call expressions?

If Statements and Call Expressions

Let's try to write a function that does the same thing as an if statement.

```
def if_(c, t, f):
    if c:
        return t
    else:
        return f
```





Execution Rule for Conditional Statements:

Each clause is considered in order.

1. Evaluate the header's expression (if present).
2. If it is a true value (or an else header), execute the suite & skip the remaining clauses.

(Demo)

Evaluation Rule for Call Expressions:

1. Evaluate the operator and then the operand subexpressions
2. Apply the function that is the value of the operator to the arguments that are the values of the operands

Control Expressions

Logical Operators

To evaluate the expression **<left>** **and** **<right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a false-y value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

```
True and 4
```

```
0 and True
```

```
12 and None and 1 / 0
```

Logical Operators

To evaluate the expression **<left>** **or** **<right>**:

1. Evaluate the subexpression **<left>**.
2. If the result is a truth-y value **v**, then the expression evaluates to **v**.
3. Otherwise, the expression evaluates to the value of the subexpression **<right>**.

```
4 or True
```

```
0 or 12
```

```
False or 2 or 1 / 0
```

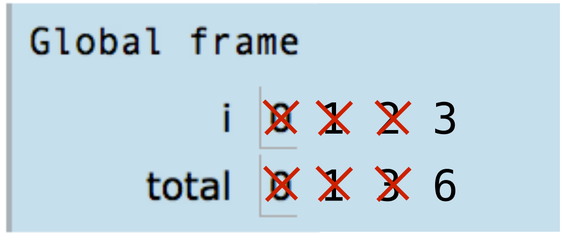
Iteration

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.

Example: Max Digit

(Demo)