61A Lecture 3

Announcements

Multiple Environments

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)
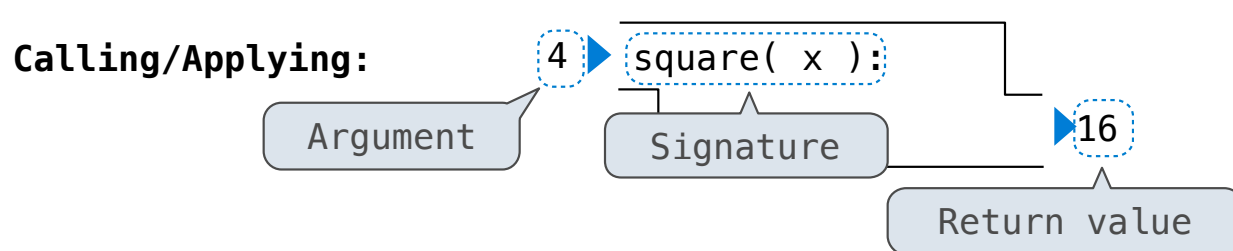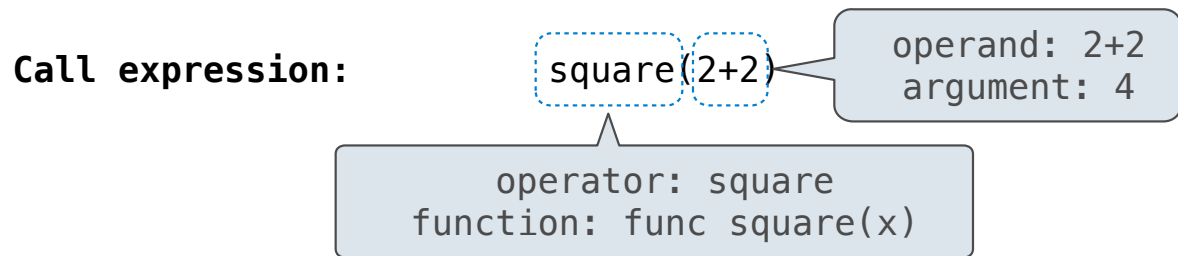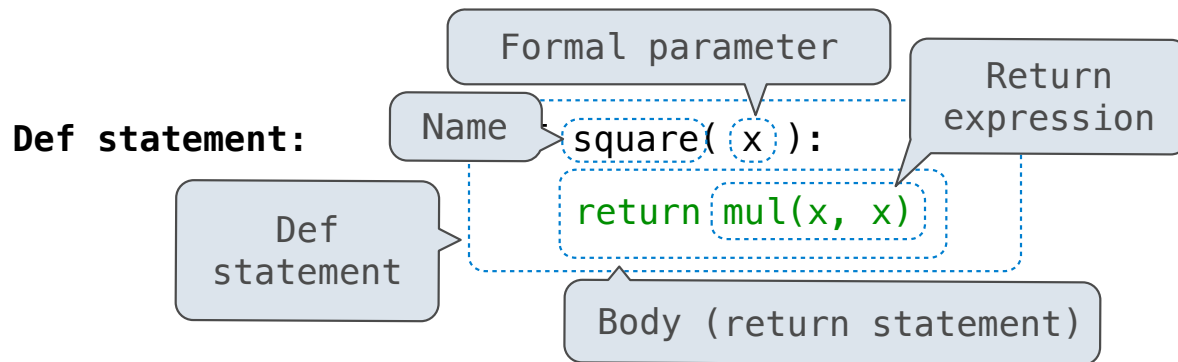
What happens?

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 ):
```

Argument

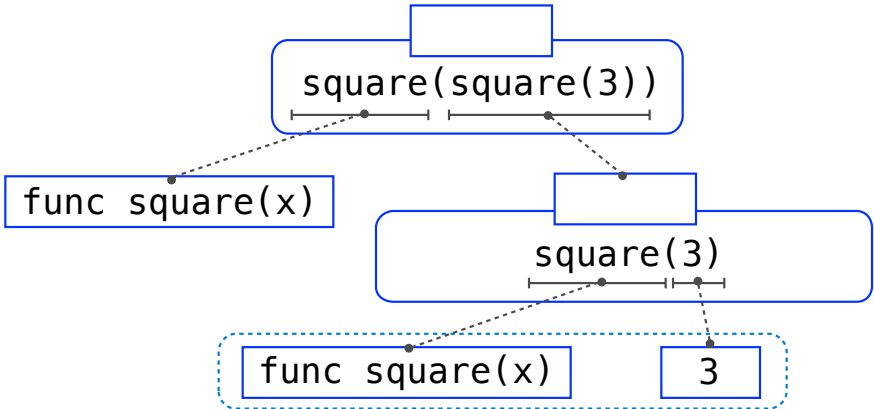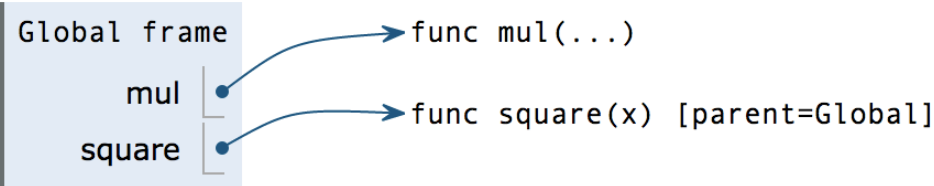Signature

▶16

Return value

A new frame is created!

Parameters bound to arguments

Body is executed in that new environment

4

Multiple Environments in One Diagram!

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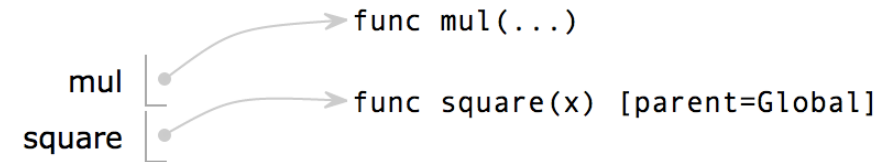

Interactive Diagram

Multiple Environments in One Diagram!

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

Global frame   func mul(...)

                                    mul
                                                func square(x) [parent=Global]
                                square

f1: square [parent=Global]

                                    x   3

                        Return   9
                         value

square(square(3))

func square(x)   9

                        square(3)

            func square(x)   3

Multiple Environments in One Diagram!
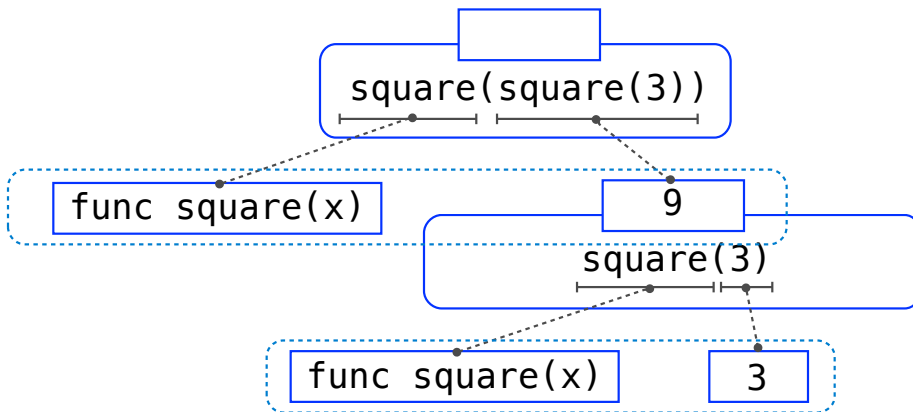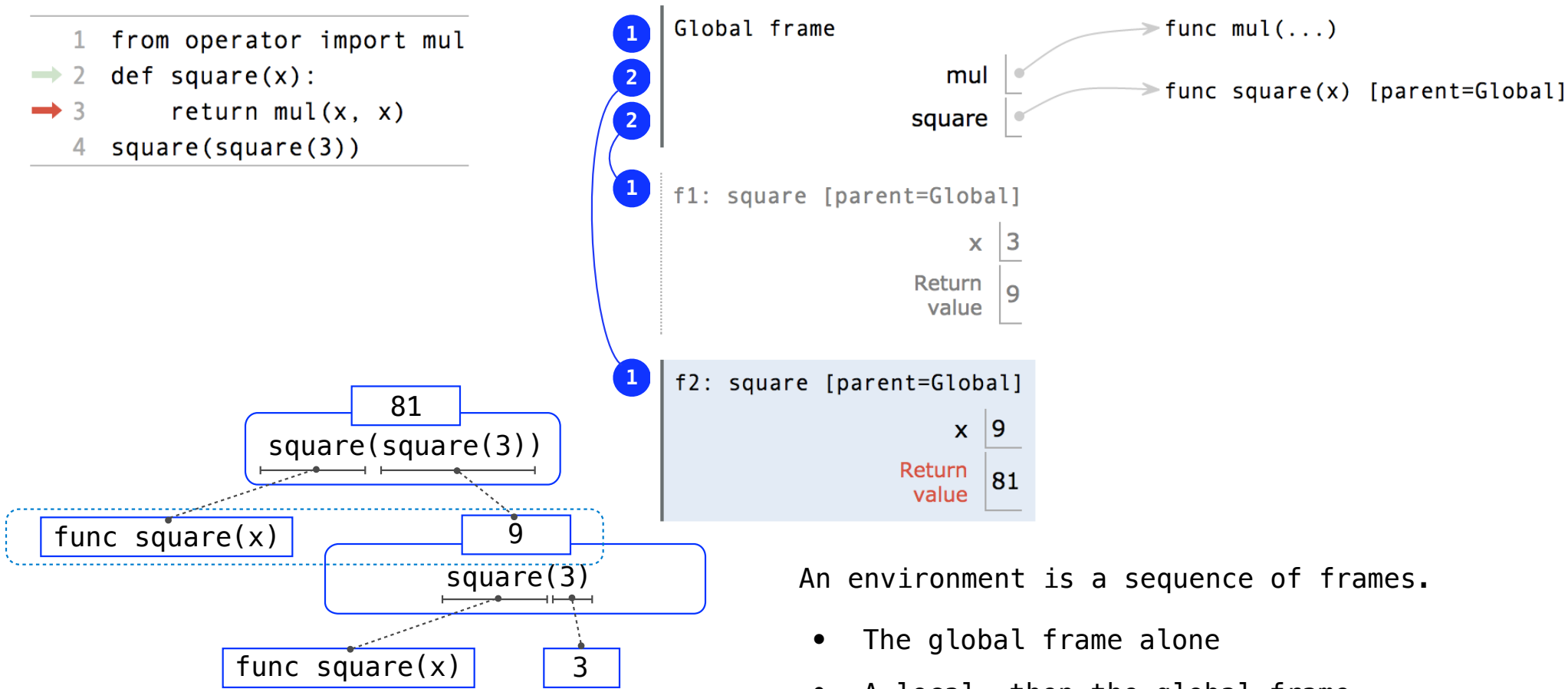
```
1   from operator import mul
2   def square(x):
3       return mul(x, x)
4   square(square(3))
```

1 Global frame   func mul(...)

2   mul

2   square   func square(x) [parent=Global]

1 f1: square [parent=Global]

                                x   3

                          Return   9
                          value

1 f2: square [parent=Global]

```
81
square(square(3))
```

```
func square(x)              9
                       square(3)
```

```
func square(x)           3
```

                                x   9

                          Return
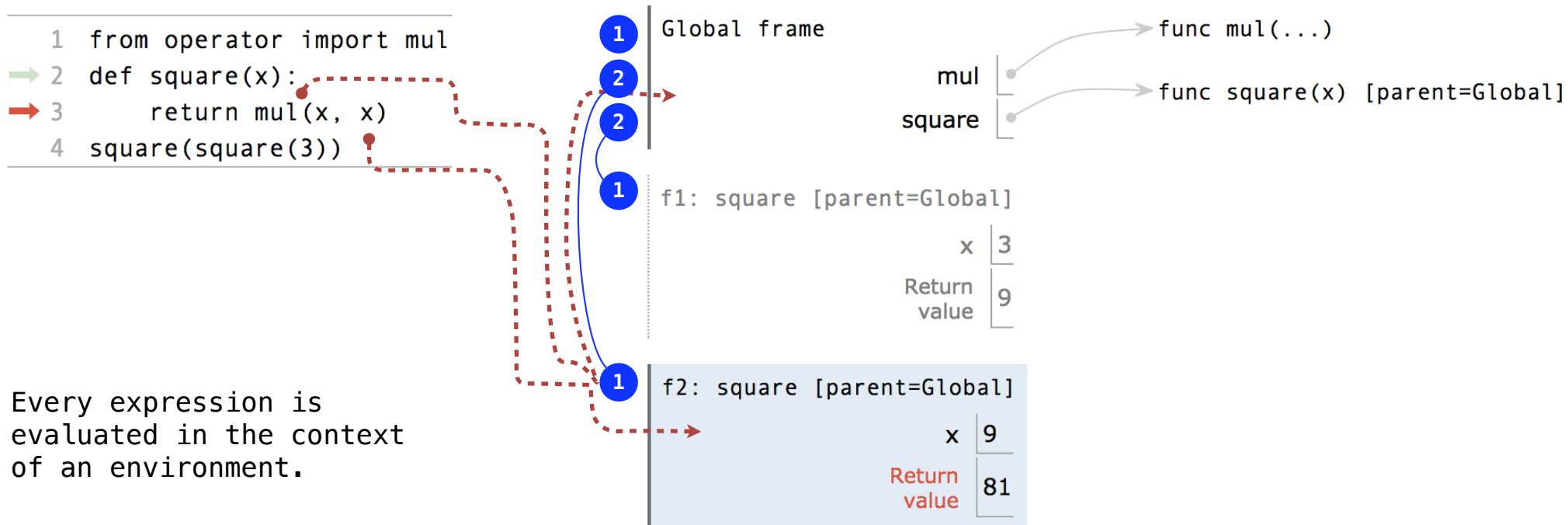   value   81

An environment is a sequence of frames.

- The global frame alone

- A local, then the global frame

<u>Interactive Diagram</u>

7

Names Have No Meaning Without Environments

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

Global frame

```
mul  •────────────→ func mul(...)
square  •──────────→ 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.
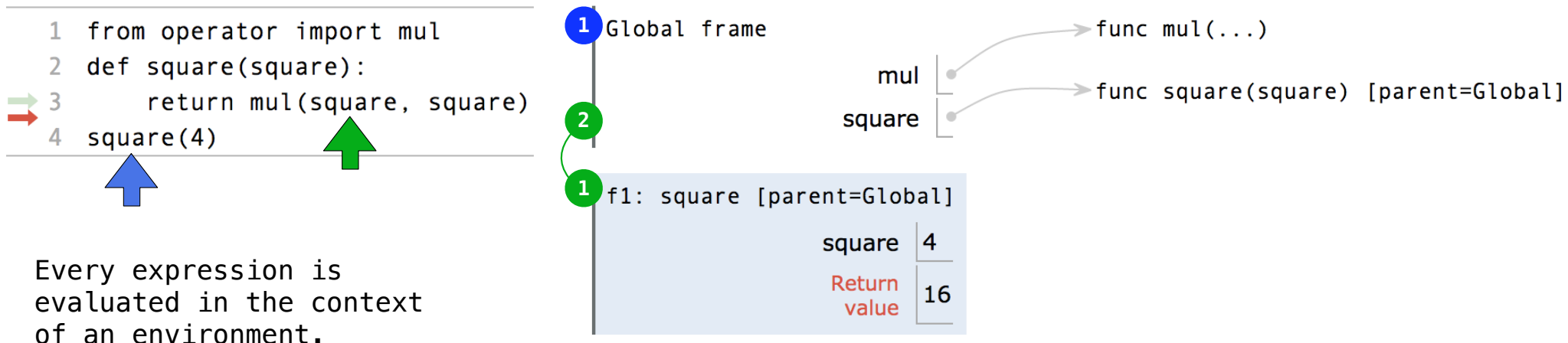
An environment is a sequence of frames.

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

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

```
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.

1 Global frame   func mul(...)

mul •   func square(square) [parent=Global]

square •

1 f1: square [parent=Global]

square 4

Return
value 16

Interactive Diagram

9

Miscellaneous Python Features

```
                Division
         Multiple Return Values
             Source Files
               Doctests
          Default Arguments


               (Demo)
```
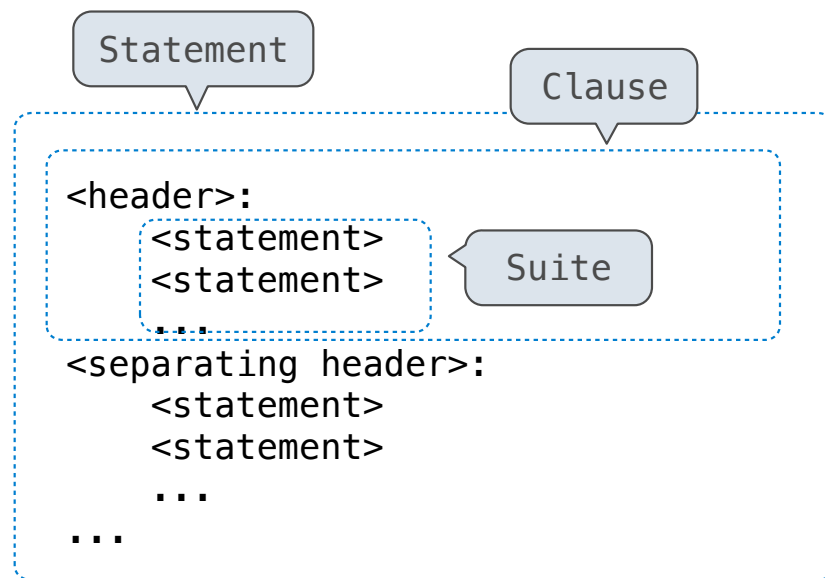
Conditional Statements

Statements

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

Compound statements:

Statement

Clause

```
<header>:
    <statement>        Suite
    <statement>
    ...
<separating header>:
    <statement>
    <statement>
    ...
...
```
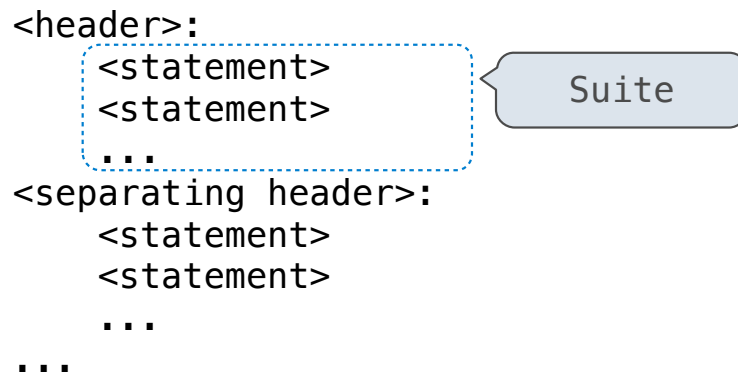
The first header determines a statement's type

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
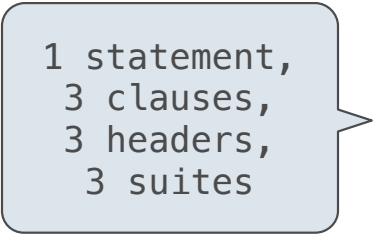
Execution Rule for a sequence of statements:

• Execute the first statement

• Unless directed otherwise, execute the rest

Conditional Statements

(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
```

> 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.

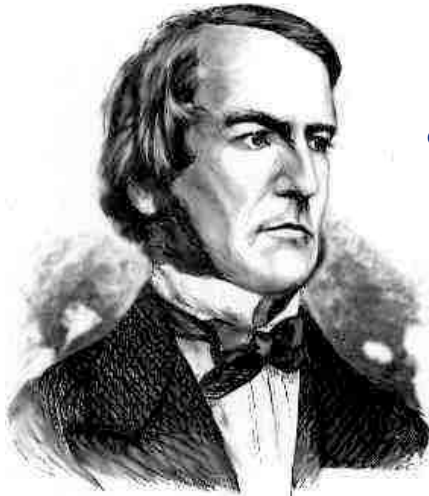3. Zero or one "else" clause,
   always at the end.

Boolean Contexts

George Boole

```python
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!

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

Iteration

While Statements

(Demo)

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

Global frame

i   0̶ 1̶ 2̶ 3

total   0̶ 1̶ 3̶ 6

George Boole

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.