61A Lecture 2

Announcements

Names, Assignment, and User-Defined Functions

(Demo)

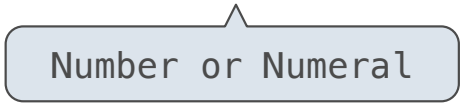
Types of Expressions

Types of Expressions

Primitive expressions:

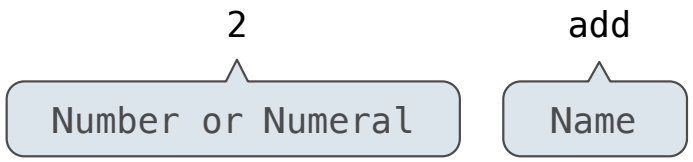
Types of Expressions

Primitive expressions: 2



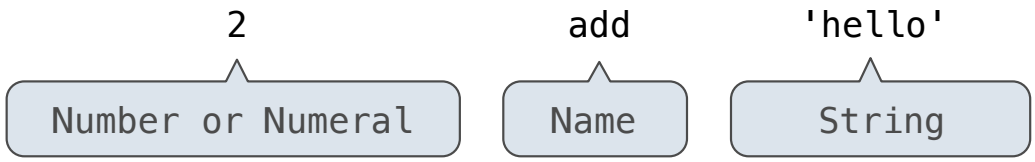
Types of Expressions

Primitive expressions:



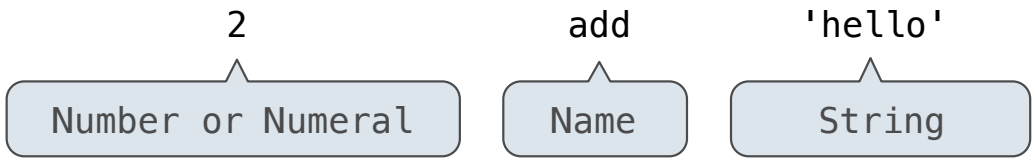
Types of Expressions

Primitive expressions:



Types of Expressions

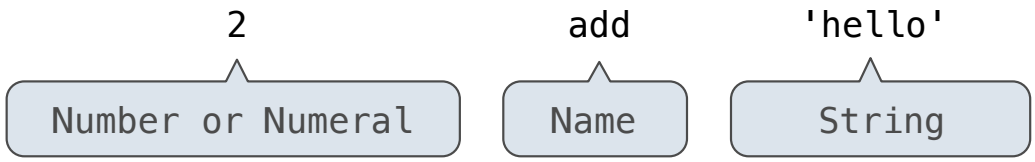
Primitive expressions:



Call expressions:

Types of Expressions

Primitive expressions:

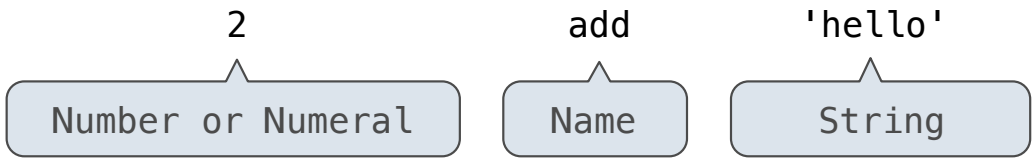


Call expressions:

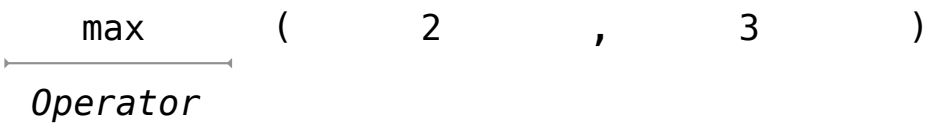
```
max ( 2 , 3 )
```

Types of Expressions

Primitive expressions:

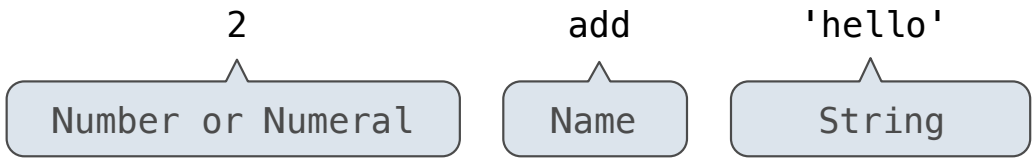


Call expressions:

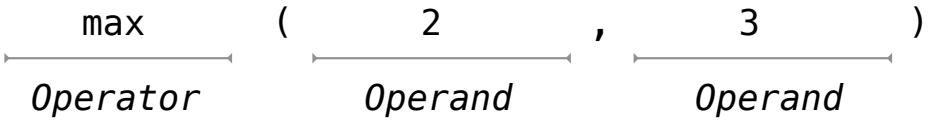


Types of Expressions

Primitive expressions:

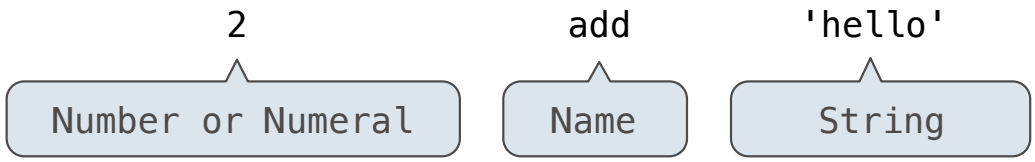


Call expressions:

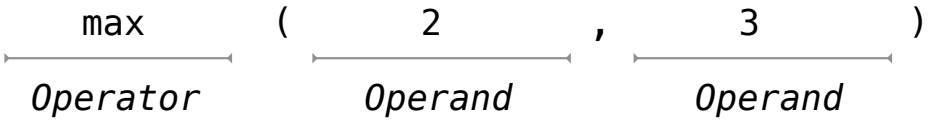


Types of Expressions

Primitive expressions:



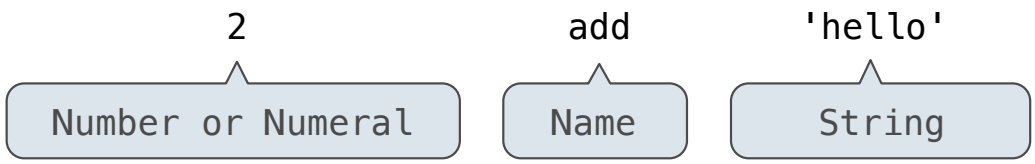
Call expressions:



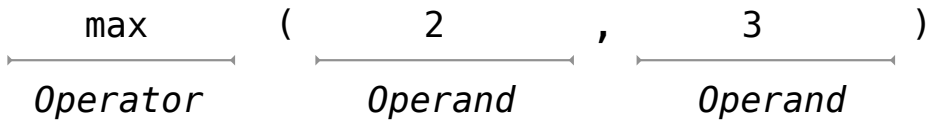

```
max(min(pow(3, 5), -4), min(1, -2))
```

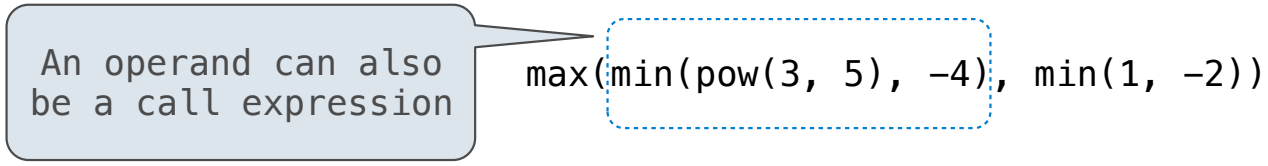
Types of Expressions

Primitive expressions:



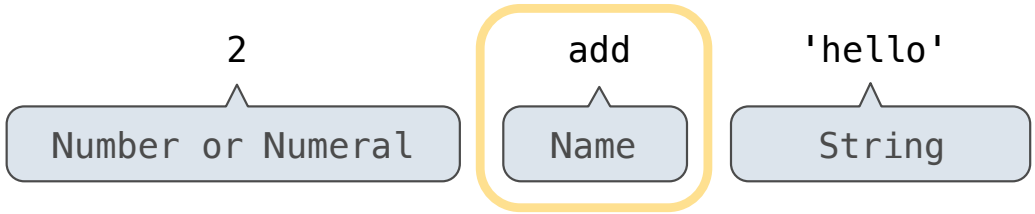
Call expressions:



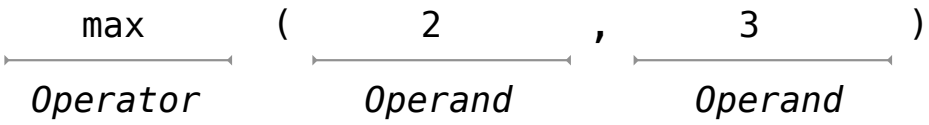


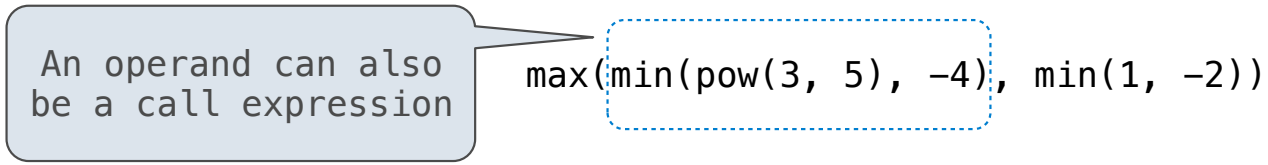
Types of Expressions

Primitive expressions:



Call expressions:





Discussion Question 1

Discussion Question 1

What is the value of the final expression in this sequence?

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
```

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
>>> f = max
```

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
>>> f = max
>>> g, h = min, max
```

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
```

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

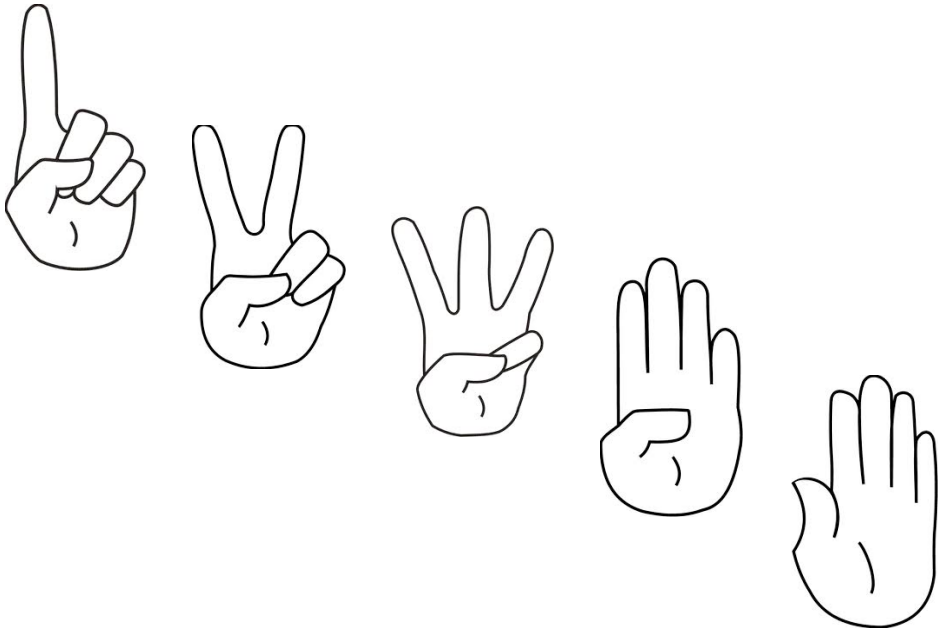
???

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
>>> f = max
>>> g, h = min, max
>>> max = g
>>> max(f(2, g(h(1, 5), 3)), 4)
```

???



Environment Diagrams

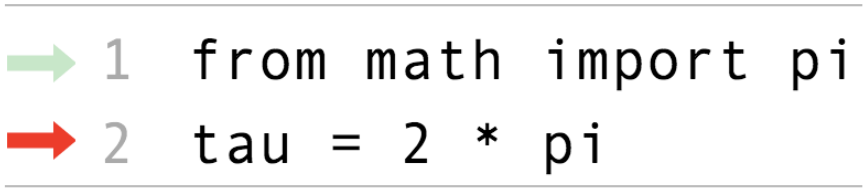
Environment Diagrams

Environment diagrams visualize the interpreter's process.

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.

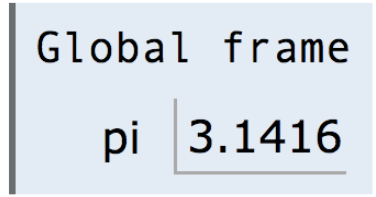


Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.

```
1  from math import pi
2  tau = 2 * pi
```

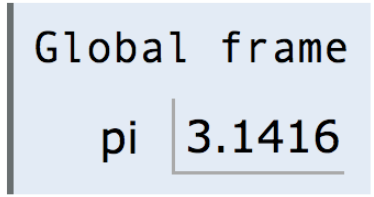


Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.

```
1  from math import pi
2  tau = 2 * pi
```

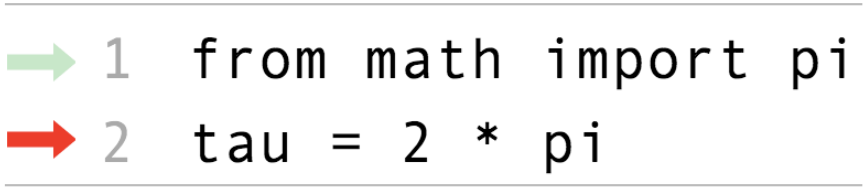


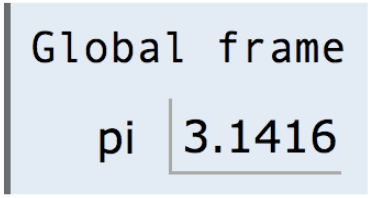
Code (left): **Frames (right):**

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

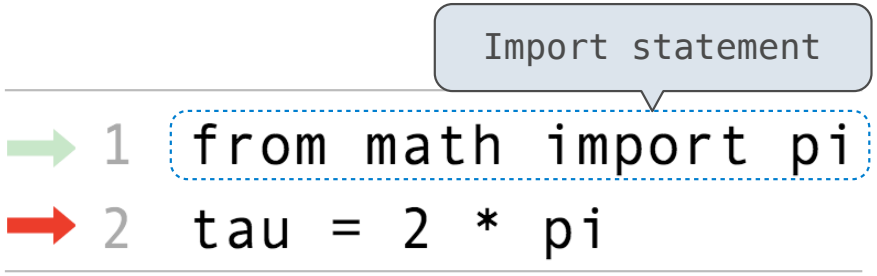
Statements and expressions

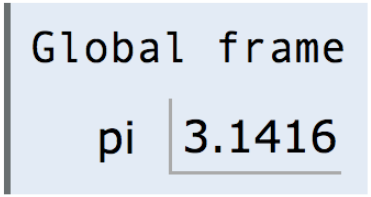
Frames (right):

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

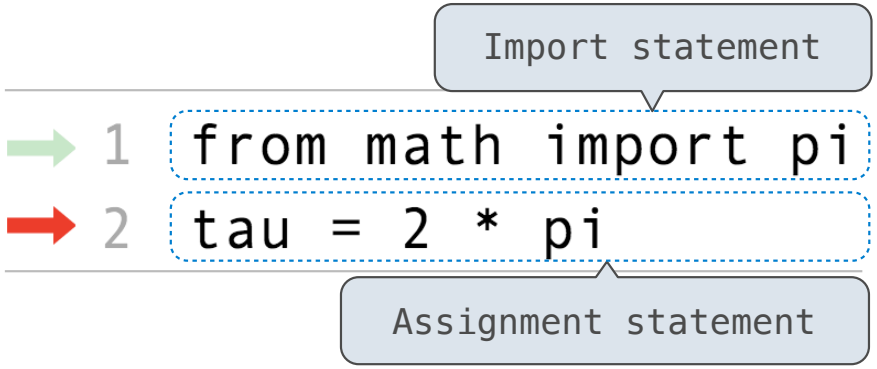
Statements and expressions

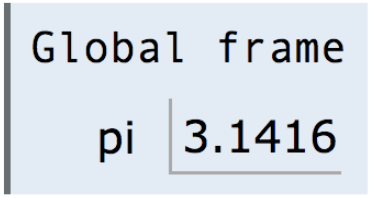
Frames (right):

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





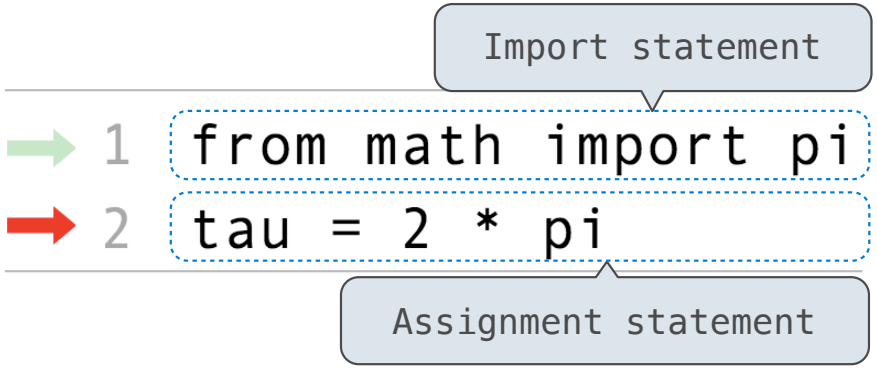
Code (left): **Frames (right):**

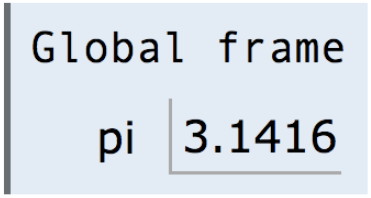
Statements and expressions

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left): **Frames (right):**

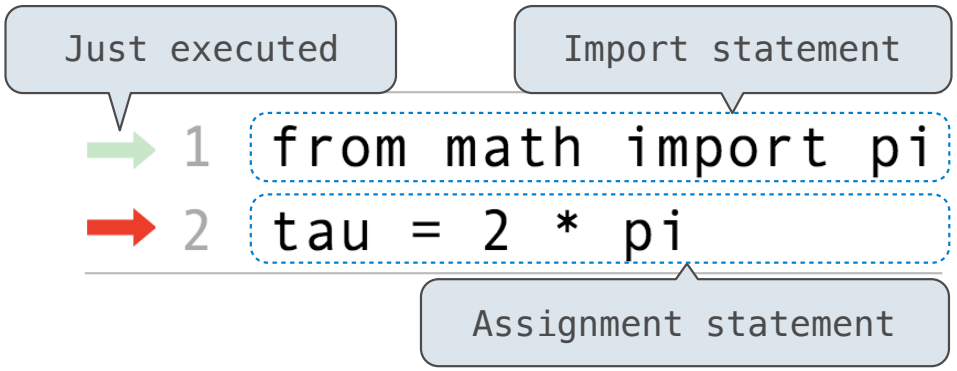
Statements and expressions

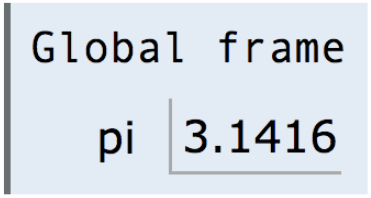
Arrows indicate evaluation order

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left): **Frames (right):**

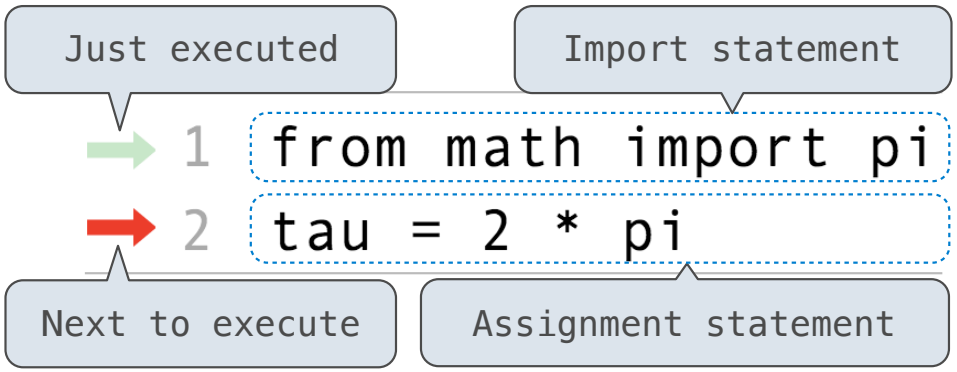
Statements and expressions

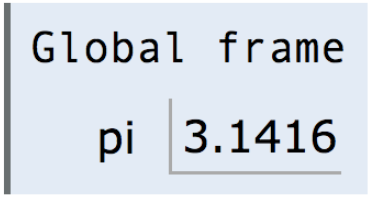
Arrows indicate evaluation order

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

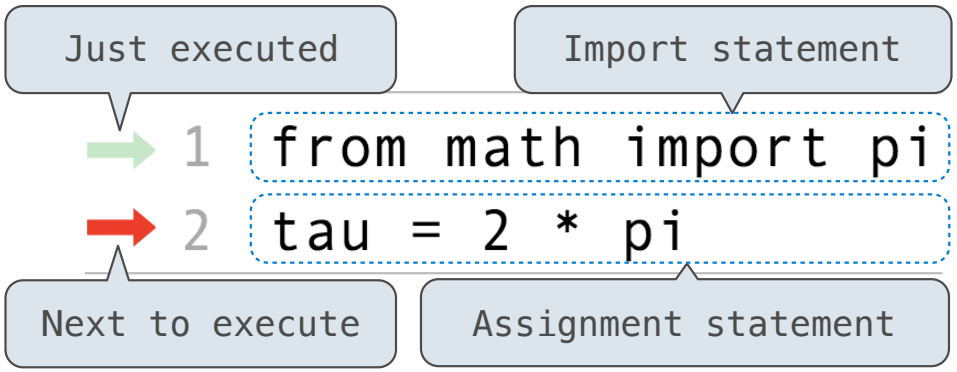
Arrows indicate evaluation order

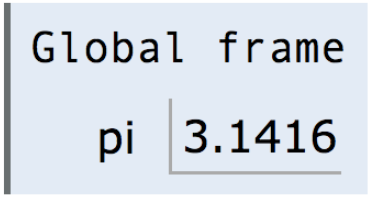
Frames (right):

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Arrows indicate evaluation order

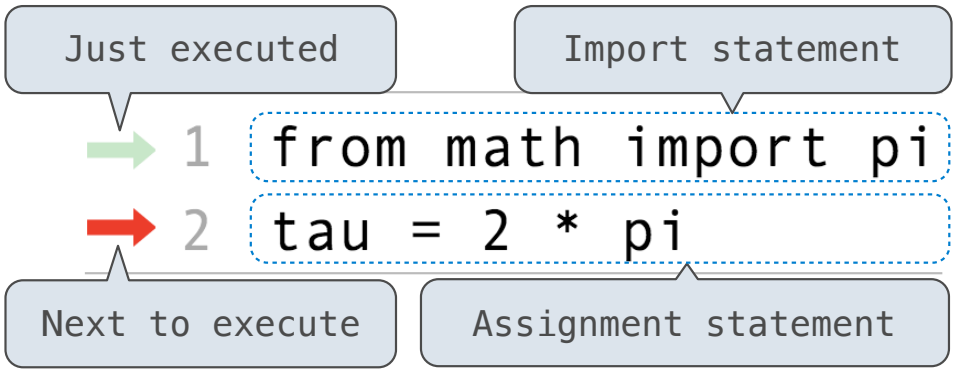
Frames (right):

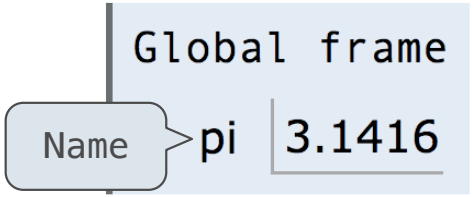
Each name is bound to a value

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Arrows indicate evaluation order

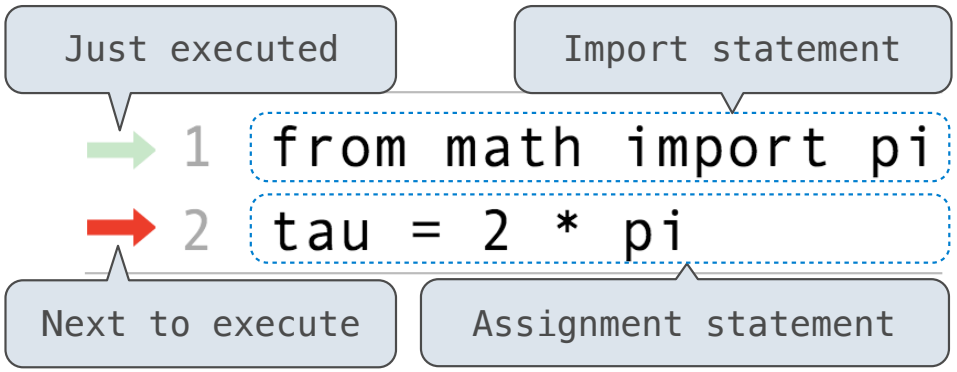
Frames (right):

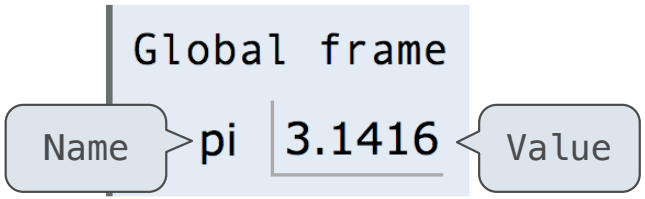
Each name is bound to a value

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Arrows indicate evaluation order

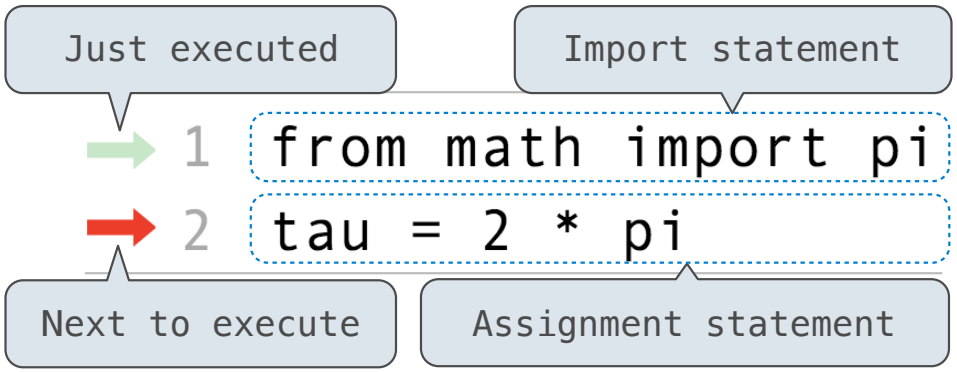
Frames (right):

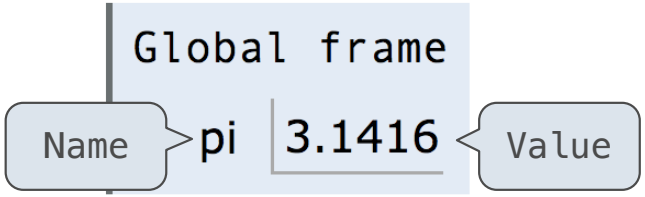
Each name is bound to a value

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Arrows indicate evaluation order

Frames (right):

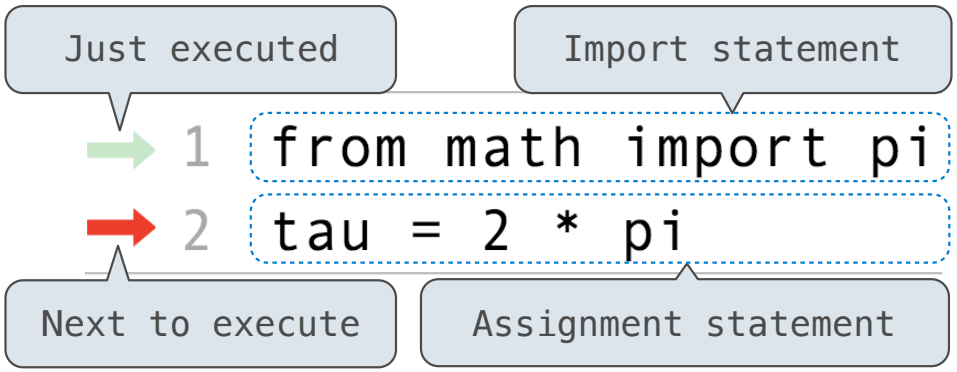
Each name is bound to a value

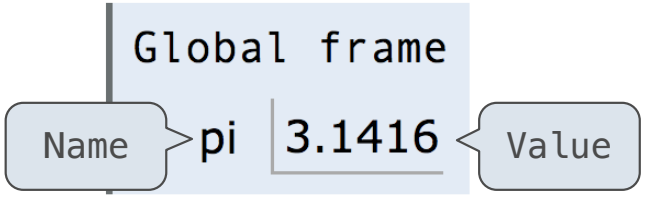
Within a frame, a name cannot be repeated

Interactive Diagram

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Arrows indicate evaluation order

Frames (right):

Each name is bound to a value

Within a frame, a name cannot be repeated

(Demo)

Interactive Diagram

Assignment Statements

Interactive Diagram

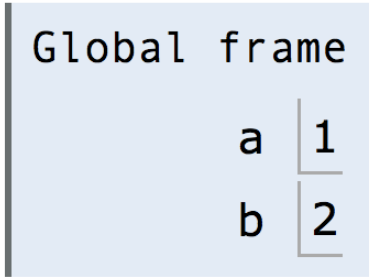
Assignment Statements

```
1  a = 1
2  b = 2
3  b, a = a + b, b
```

Interactive Diagram

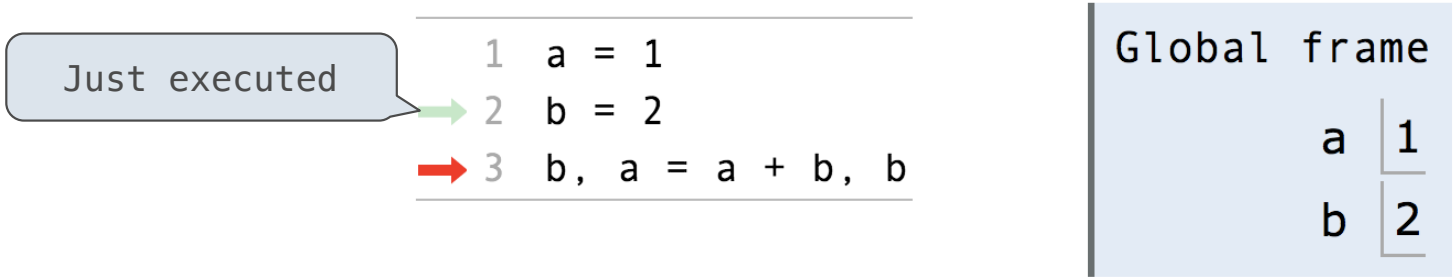
Assignment Statements

```
1  a = 1
2  b = 2
3  b, a = a + b, b
```



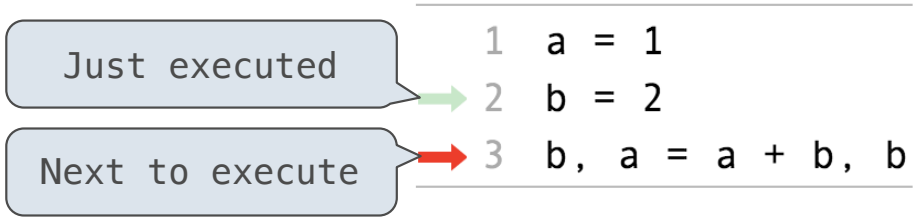
Interactive Diagram

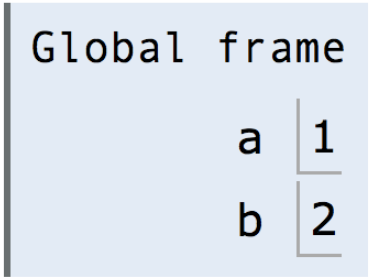
Assignment Statements



Interactive Diagram

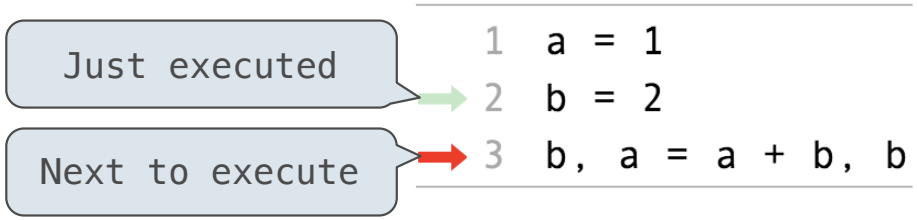
Assignment Statements

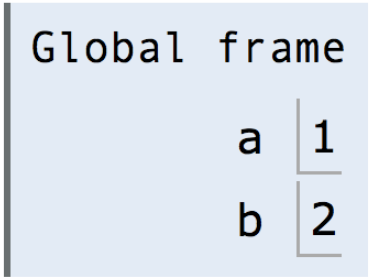




Interactive Diagram

Assignment Statements

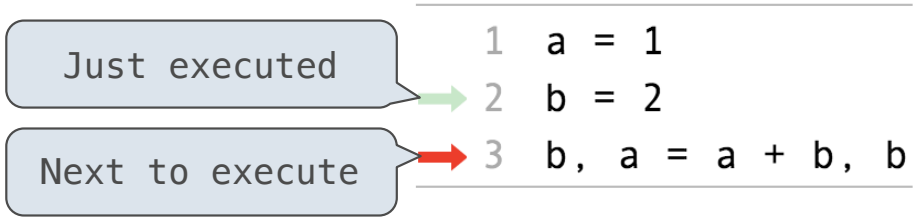


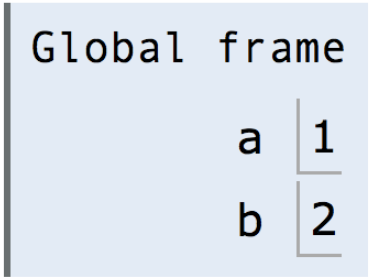


Execution rule for assignment statements:

Interactive Diagram

Assignment Statements



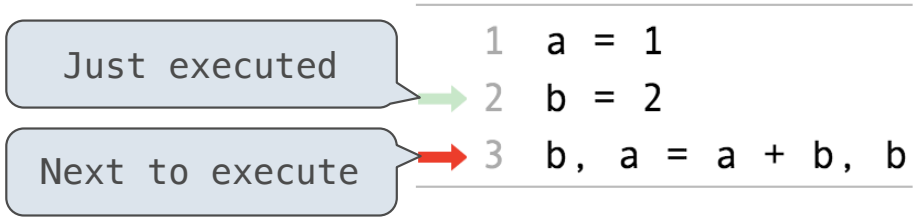


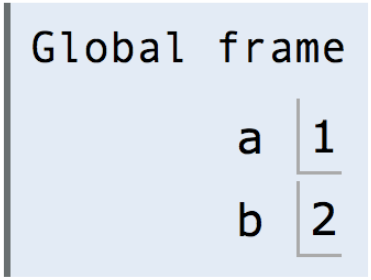
Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.

Interactive Diagram

Assignment Statements





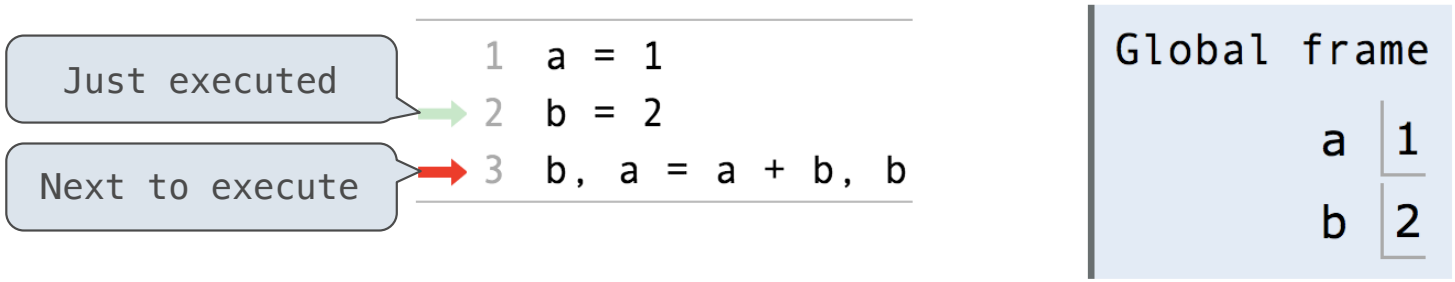
Execution rule for assignment statements:

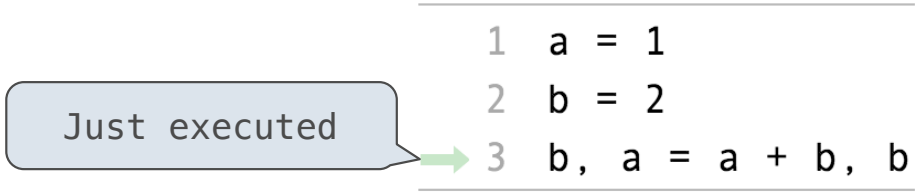
1. Evaluate all expressions to the right of = from left to right.

2. Bind all names to the left of = to those resulting values in the current frame.

Interactive Diagram

Assignment Statements





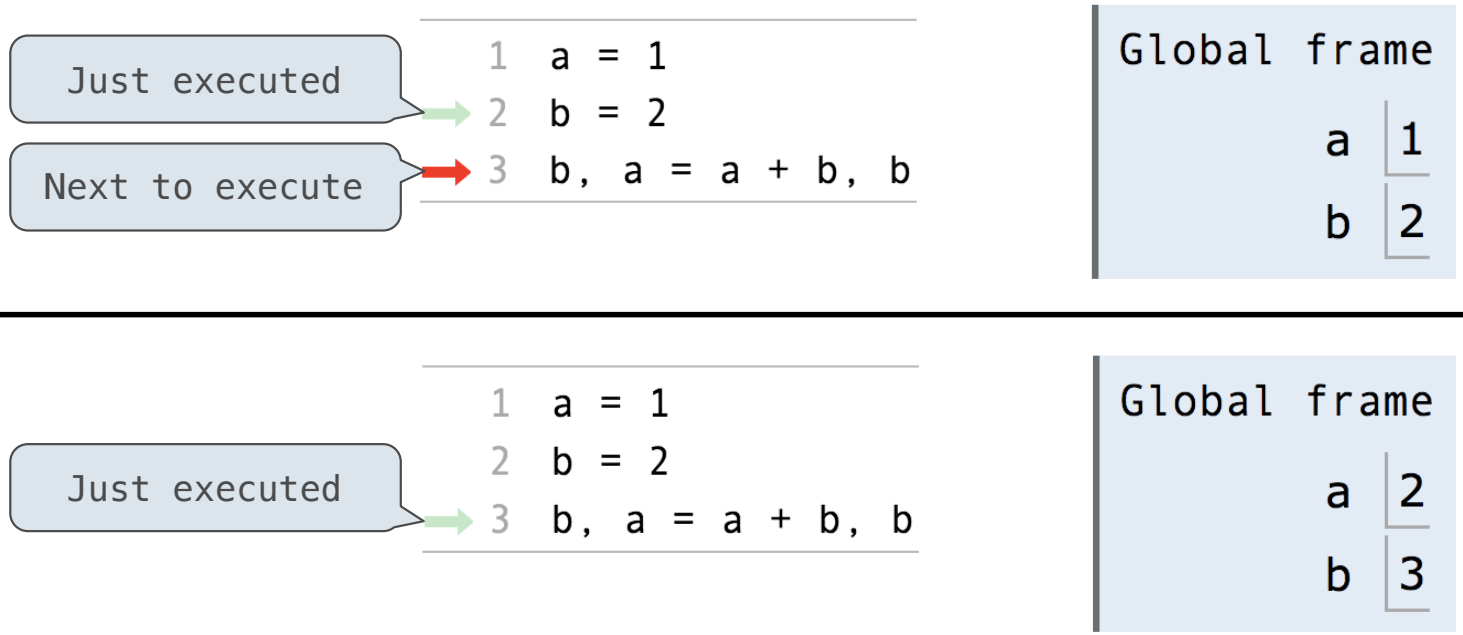
Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.

2. Bind all names to the left of = to those resulting values in the current frame.

Interactive Diagram

Assignment Statements



Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.

2. Bind all names to the left of = to those resulting values in the current frame.

Interactive Diagram

Discussion Question 1 Solution

(Demo)

Interactive Diagram

Discussion Question 1 Solution

(Demo)

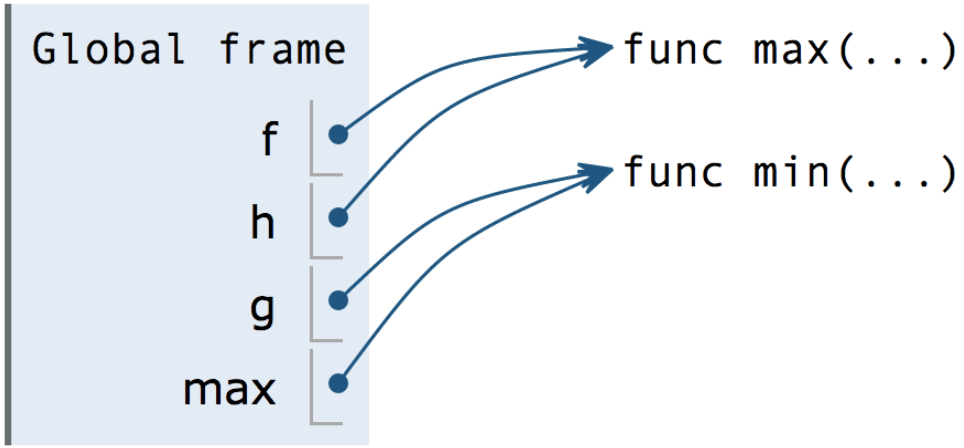
```
1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)
```

Interactive Diagram

Discussion Question 1 Solution

```
1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)
```

(Demo)

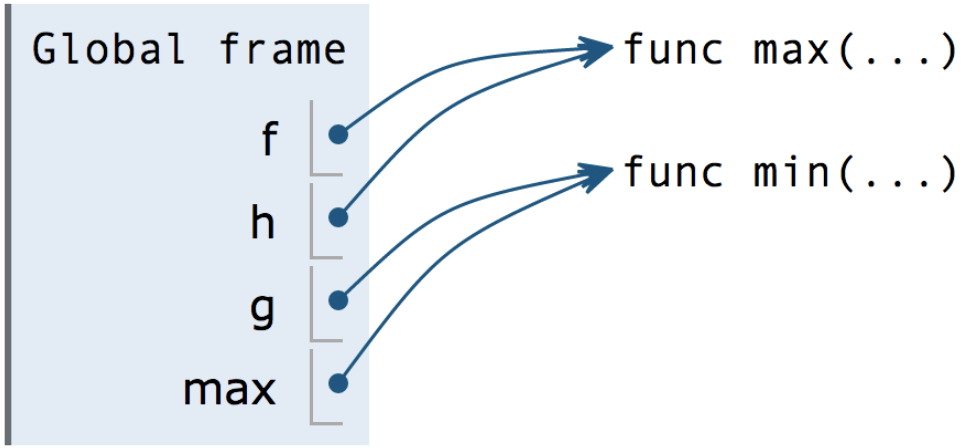


Interactive Diagram

Discussion Question 1 Solution

```
1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)
```

(Demo)



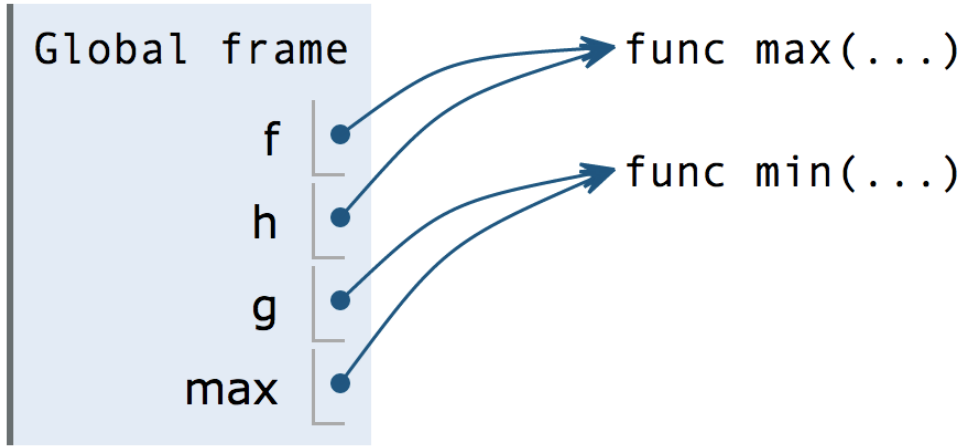
Interactive Diagram

Discussion Question 1 Solution

```
1  f = min
2  f = max
3  g, h = min, max
4  max = g
5  max(f(2, g(h(1, 5), 3)), 4)
```

func min(...)

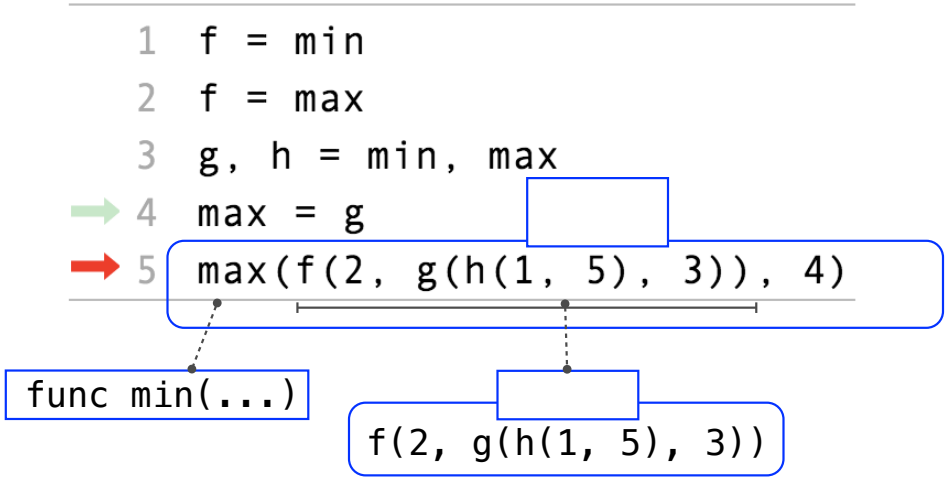
(Demo)

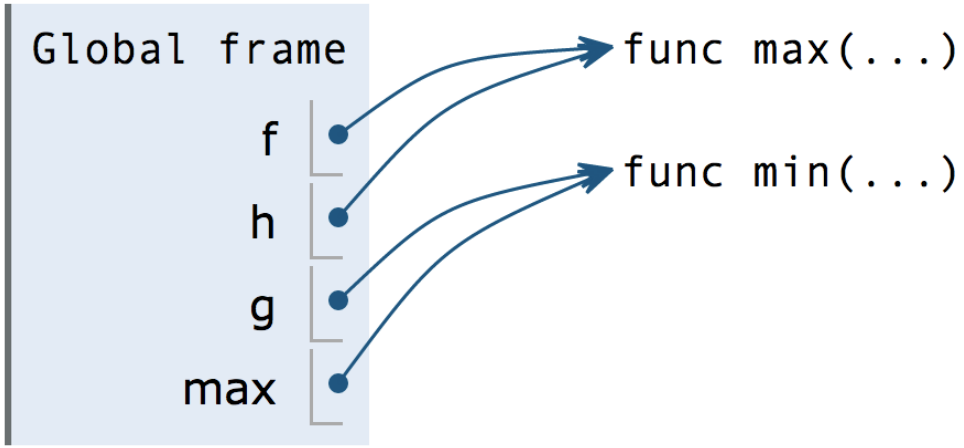


Interactive Diagram

Discussion Question 1 Solution

(Demo)

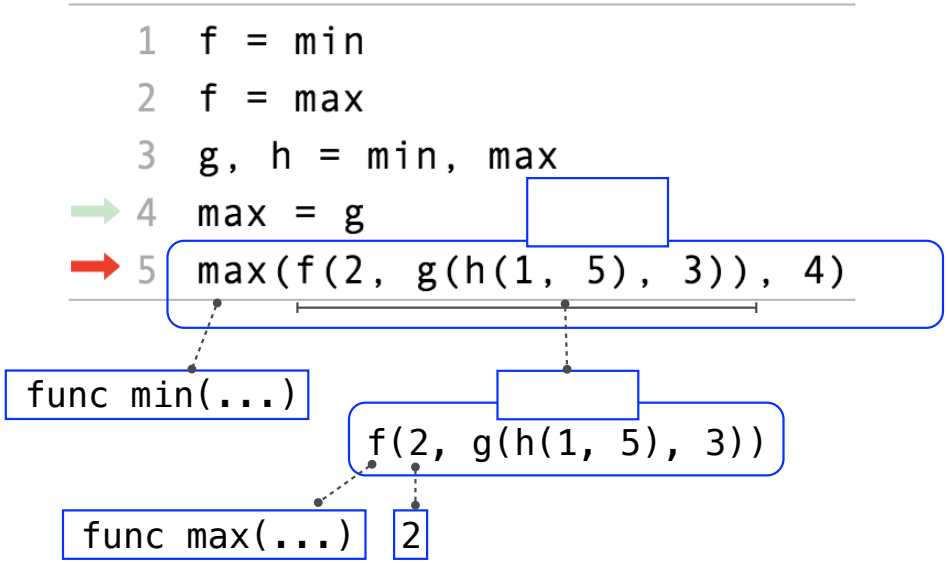


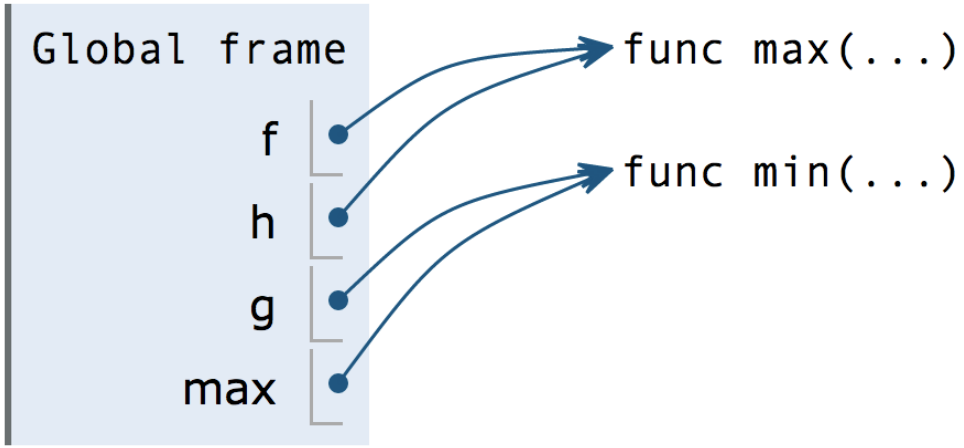


Interactive Diagram

Discussion Question 1 Solution

(Demo)

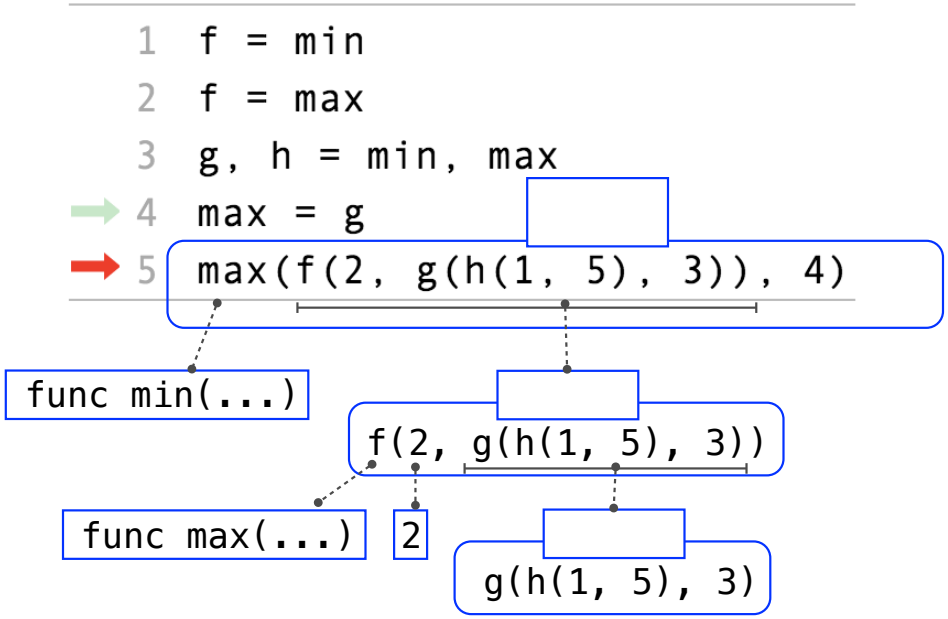


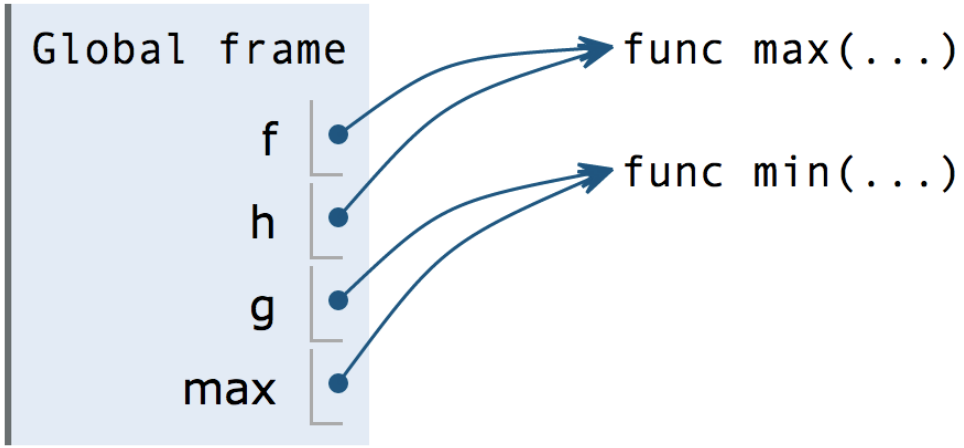


Interactive Diagram

Discussion Question 1 Solution

(Demo)

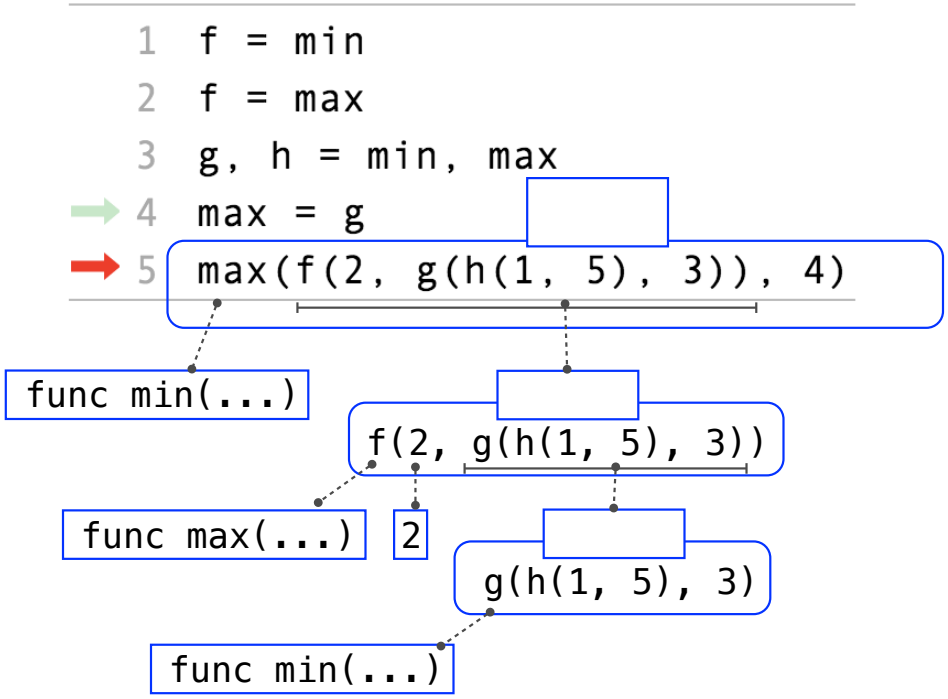


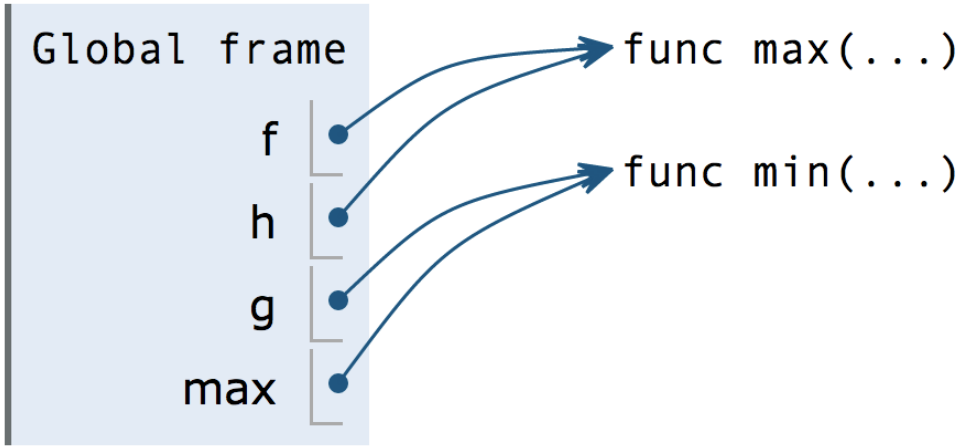


Interactive Diagram

Discussion Question 1 Solution

(Demo)

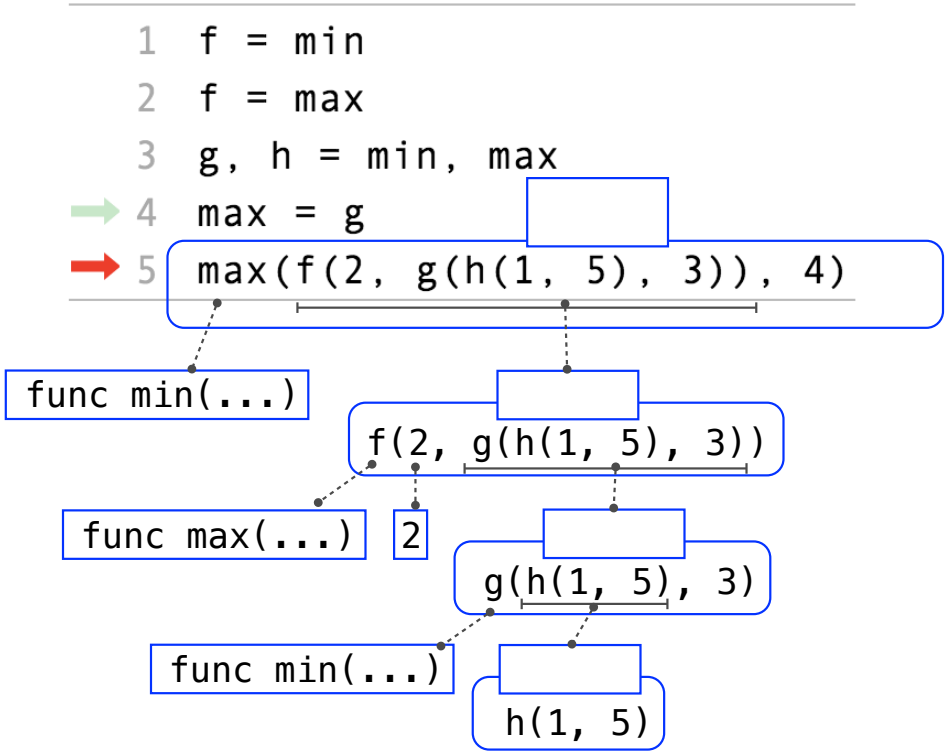


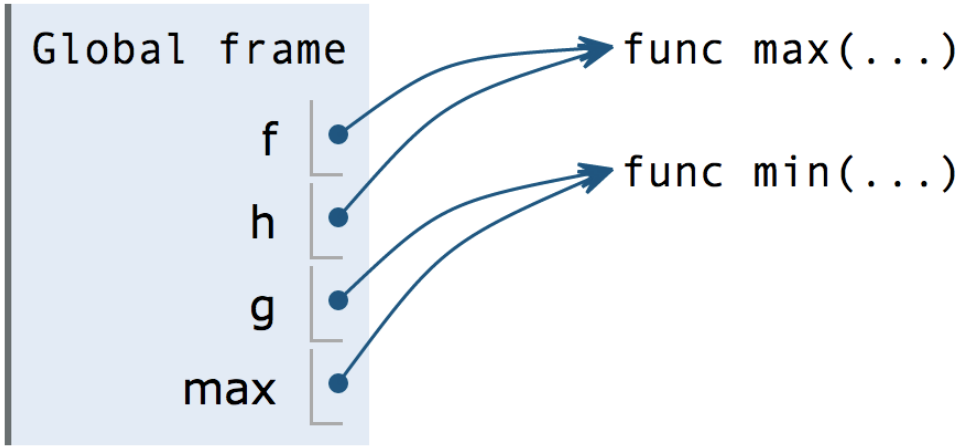


Interactive Diagram

Discussion Question 1 Solution

(Demo)

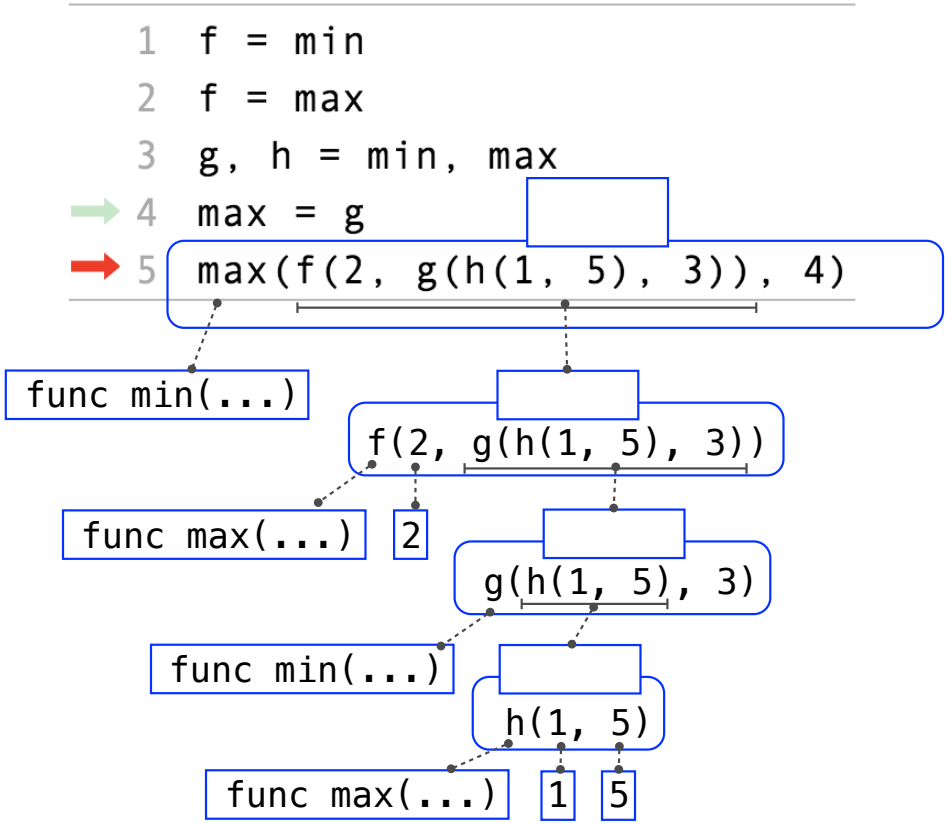


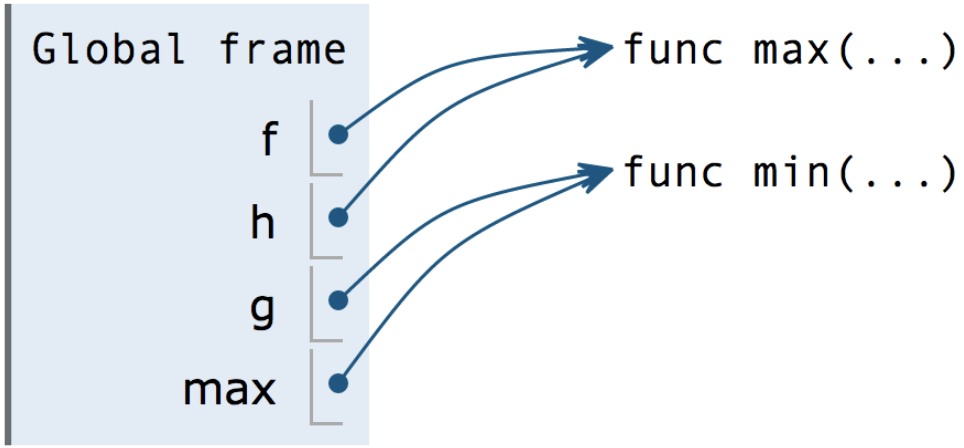


Interactive Diagram

Discussion Question 1 Solution

(Demo)

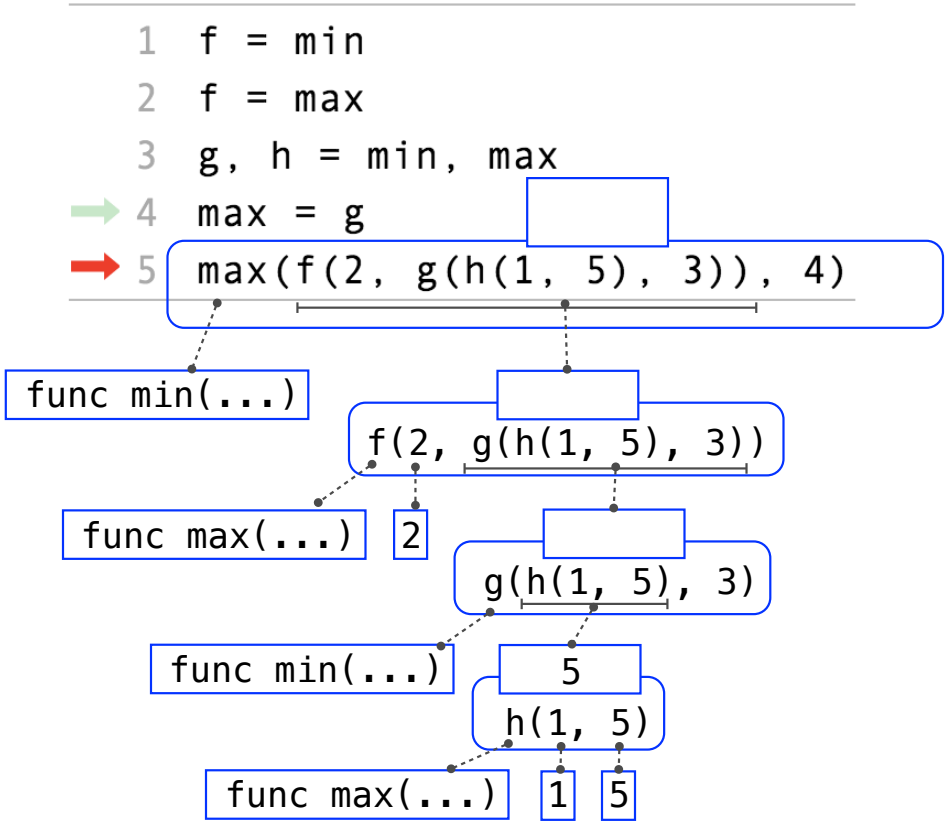


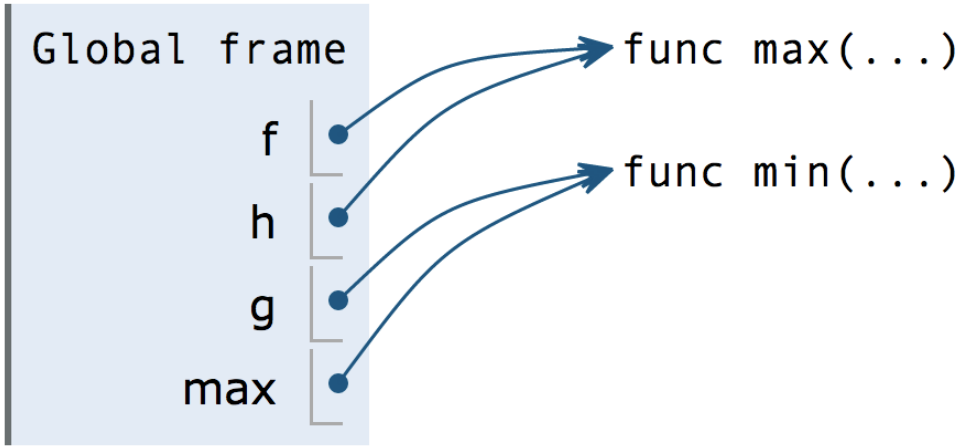


Interactive Diagram

Discussion Question 1 Solution

(Demo)

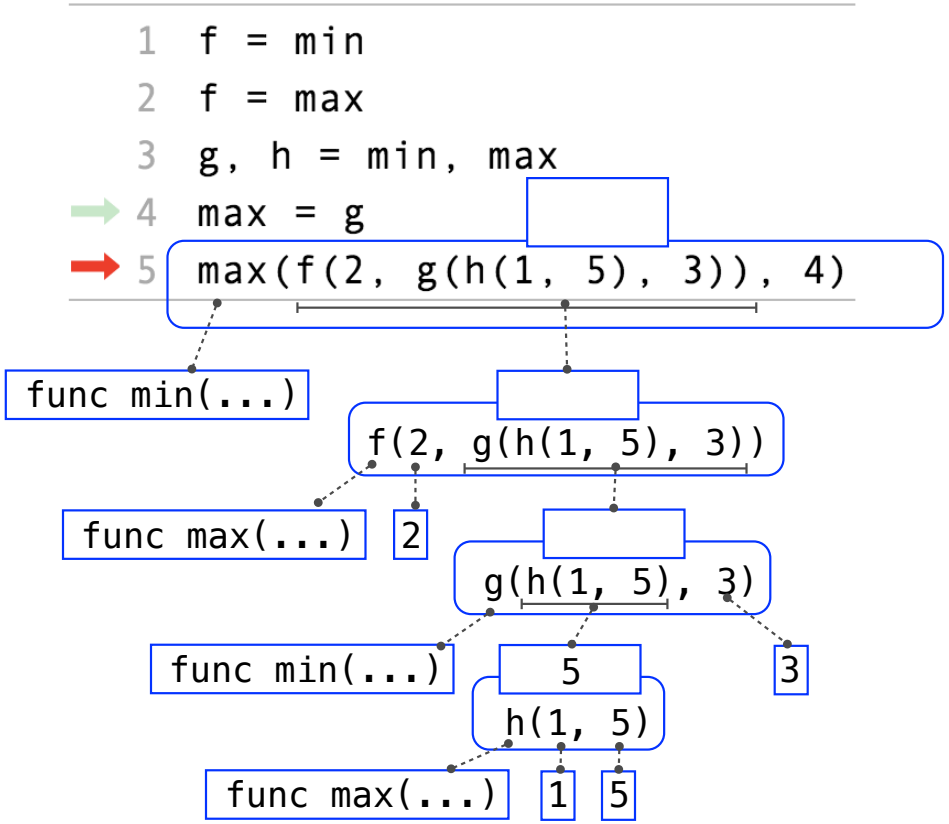


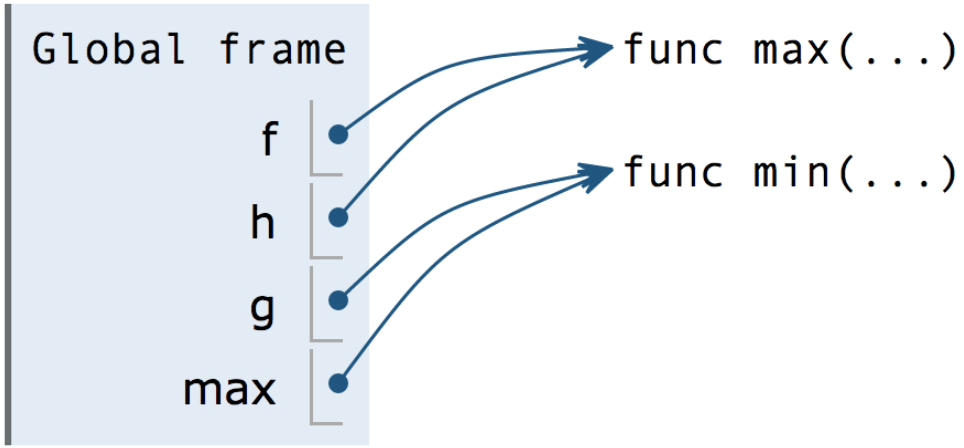


Interactive Diagram

Discussion Question 1 Solution

(Demo)

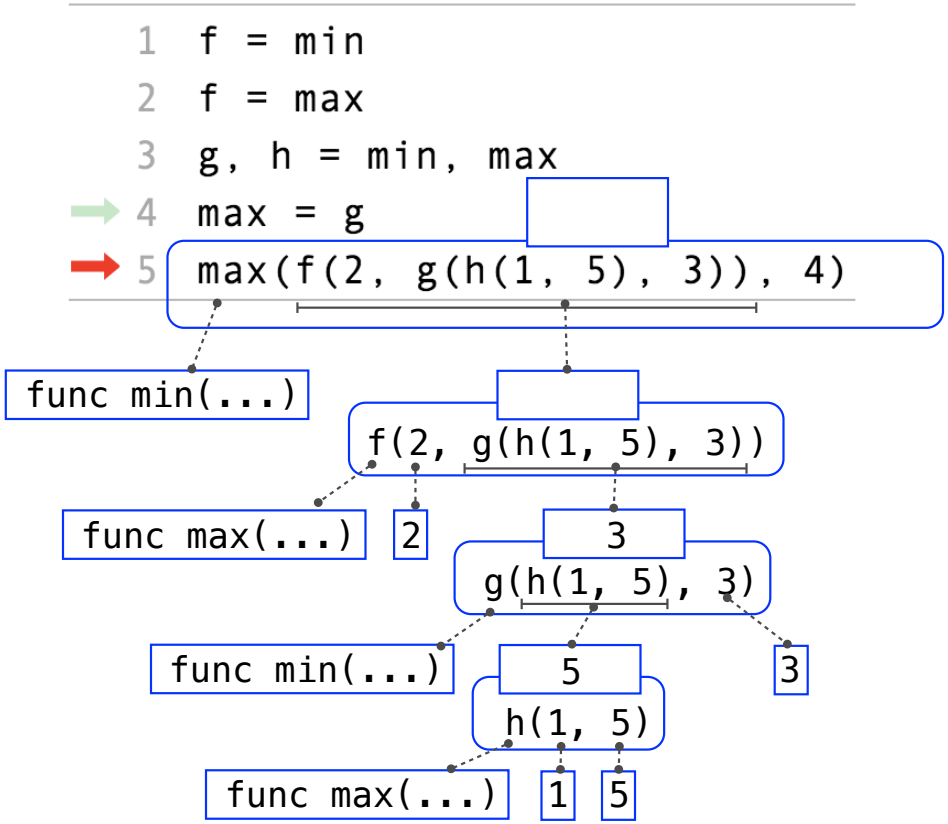


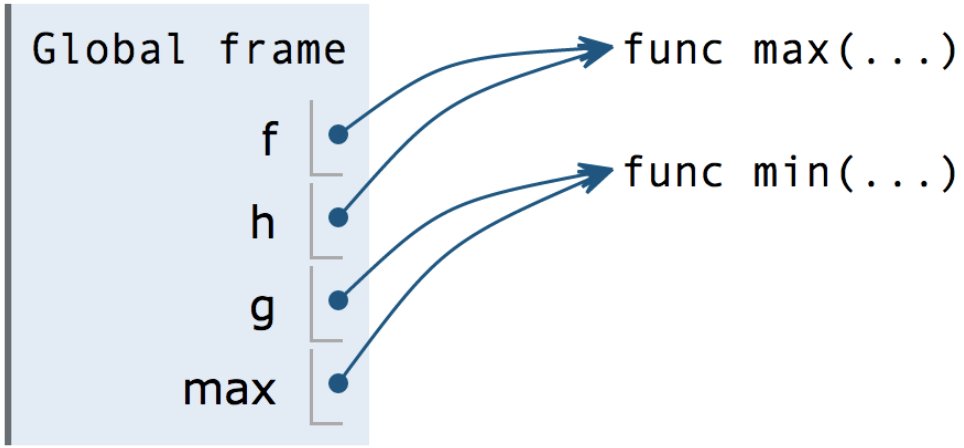


Interactive Diagram

Discussion Question 1 Solution

(Demo)

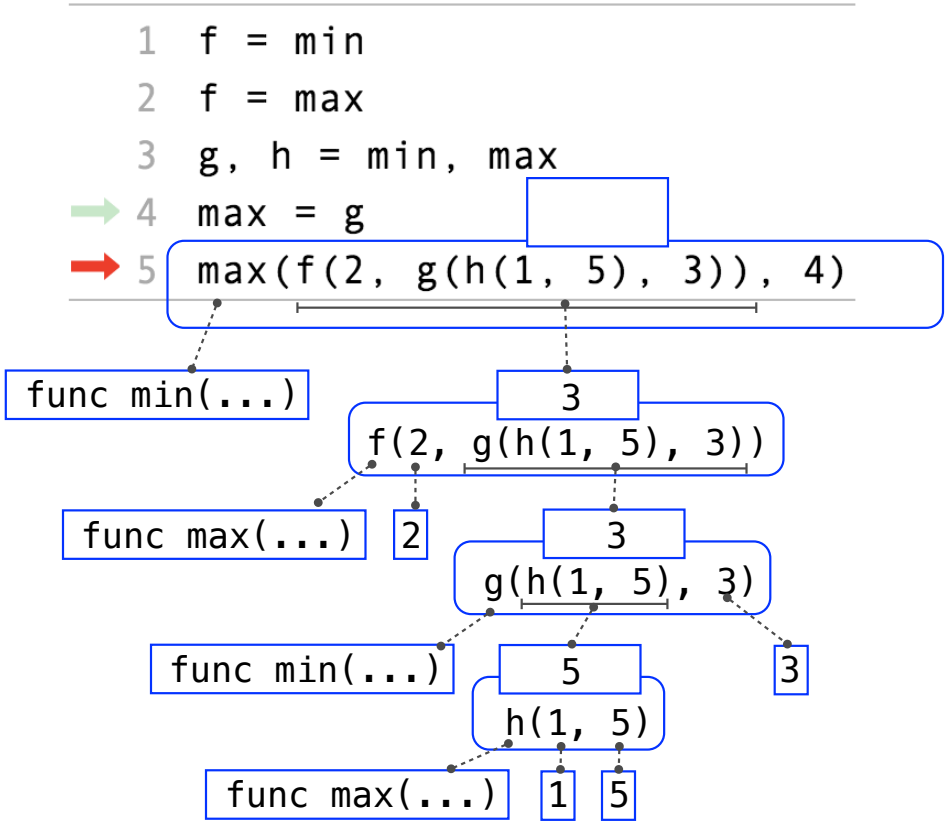


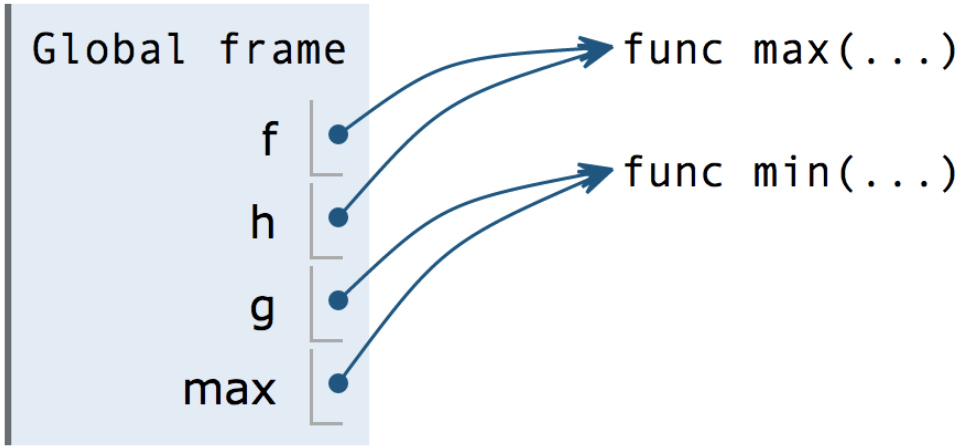


Interactive Diagram

Discussion Question 1 Solution

(Demo)

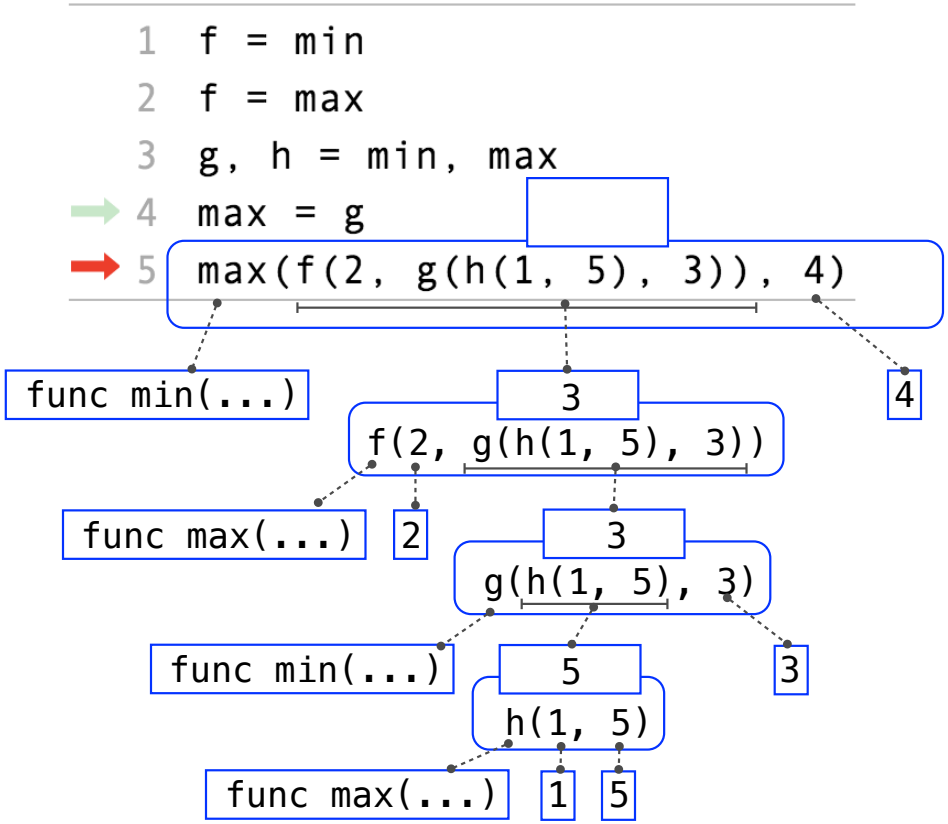


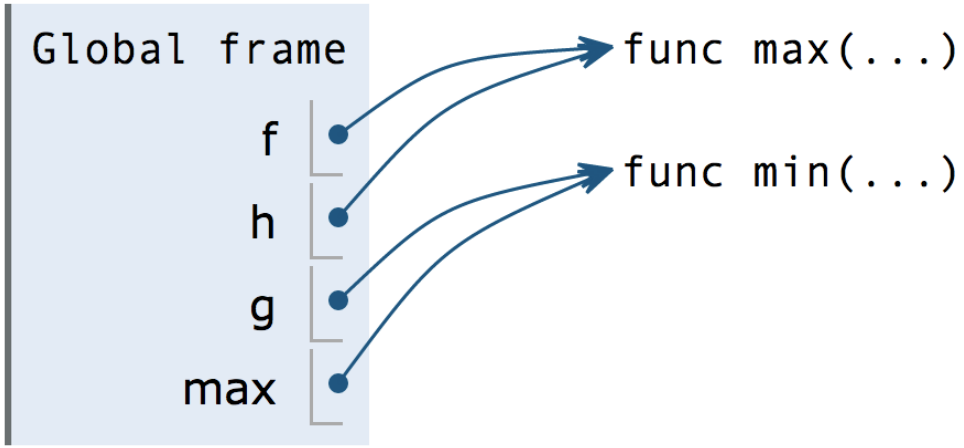


Interactive Diagram

Discussion Question 1 Solution

(Demo)

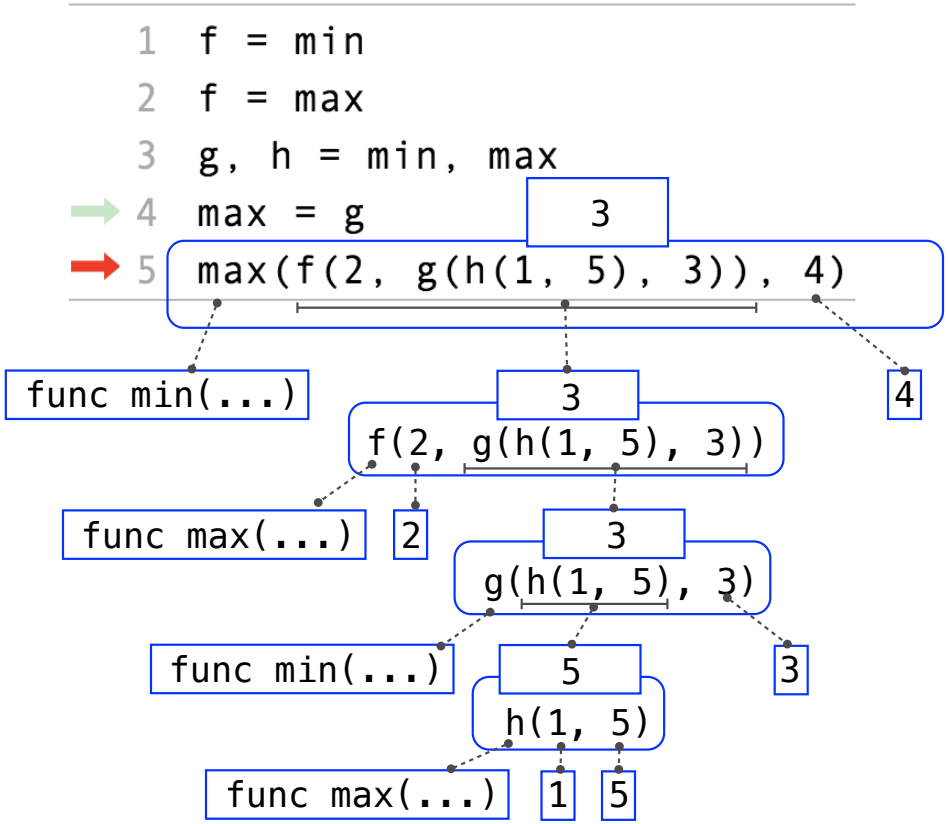


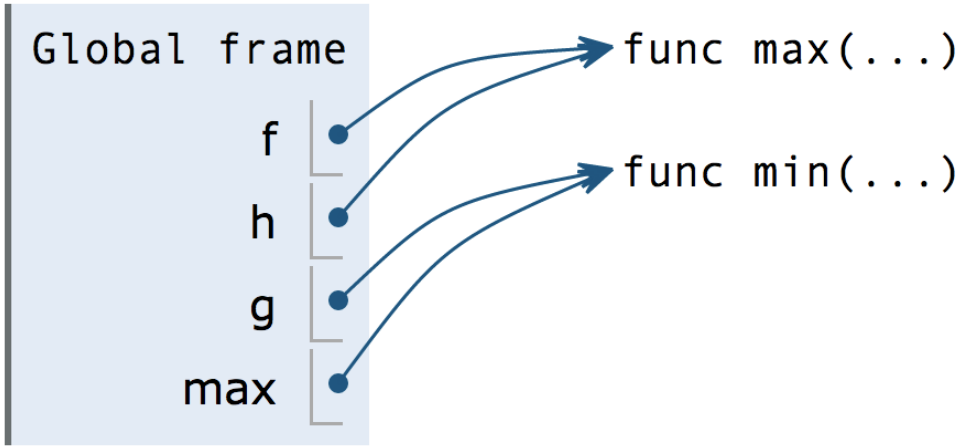


Interactive Diagram

Discussion Question 1 Solution

(Demo)

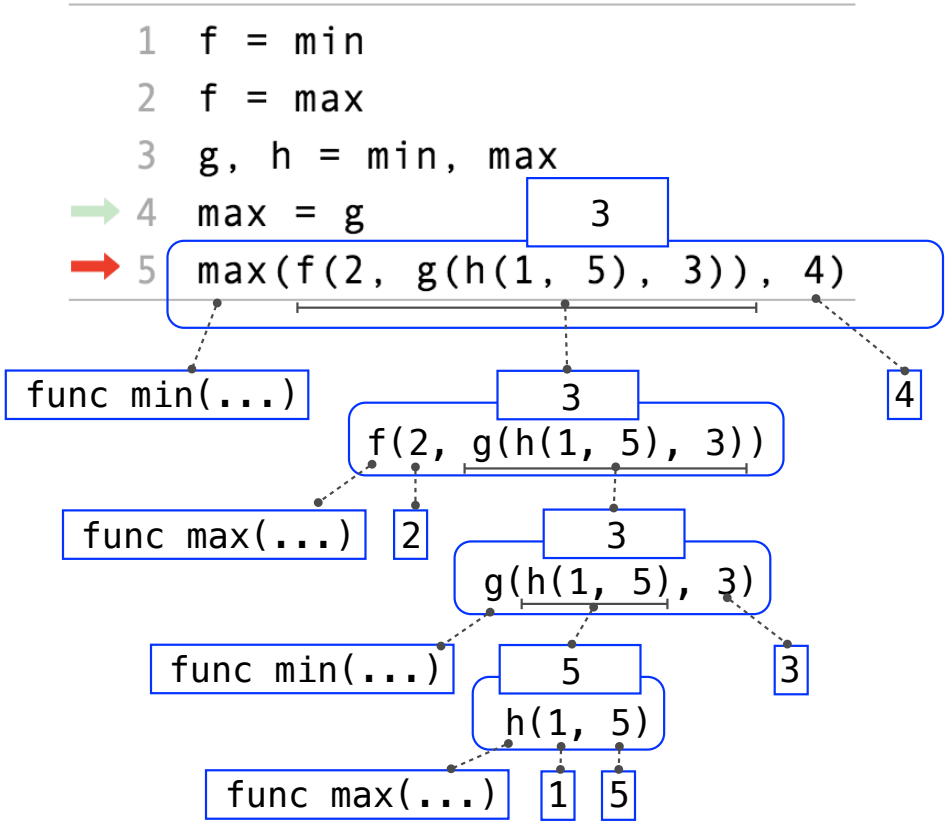


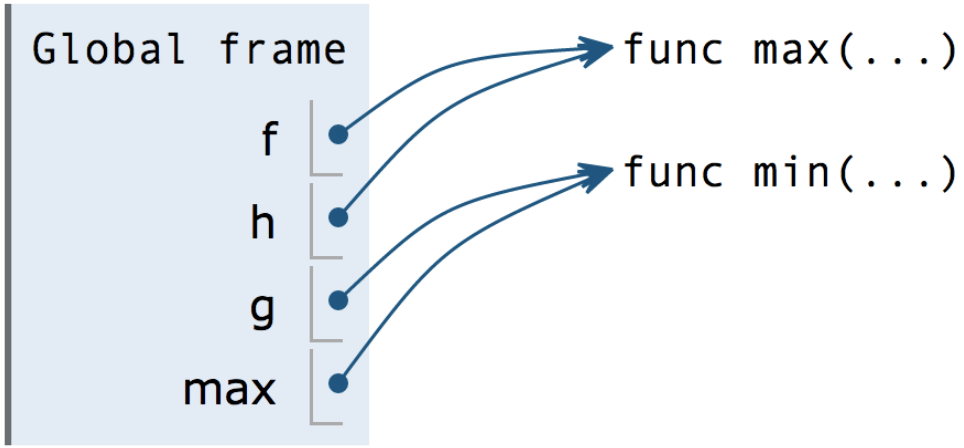


Interactive Diagram

Discussion Question 1 Solution

(Demo)





3

Interactive Diagram

Defining Functions

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

```
>>> def <name>(<formal parameters>):
        return <return expression>
```

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

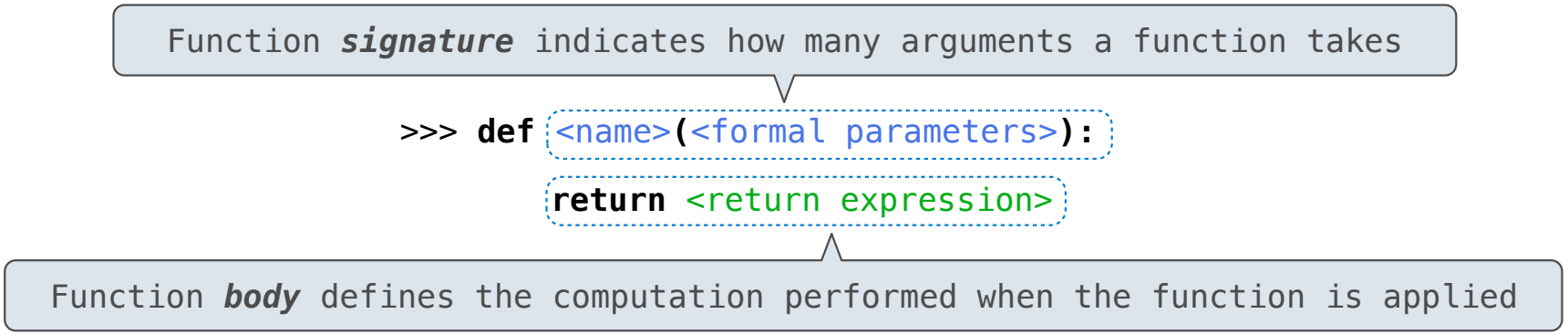
Function ***signature*** indicates how many arguments a function takes

```
>>> def <name>(<formal parameters>):
        return <return expression>
```

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions



Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function ***signature*** indicates how many arguments a function takes

```
>>> def <name>(<formal parameters>):
        return <return expression>
```

Function ***body*** defines the computation performed when the function is applied

Execution procedure for def statements:

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function ***signature*** indicates how many arguments a function takes

```
>>> def <name>(<formal parameters>):
        return <return expression>
```

Function ***body*** defines the computation performed when the function is applied

Execution procedure for def statements:

1. Create a function with signature <name>(<formal parameters>)

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function ***signature*** indicates how many arguments a function takes

```
>>> def <name>(<formal parameters>):
        return <return expression>
```

Function ***body*** defines the computation performed when the function is applied

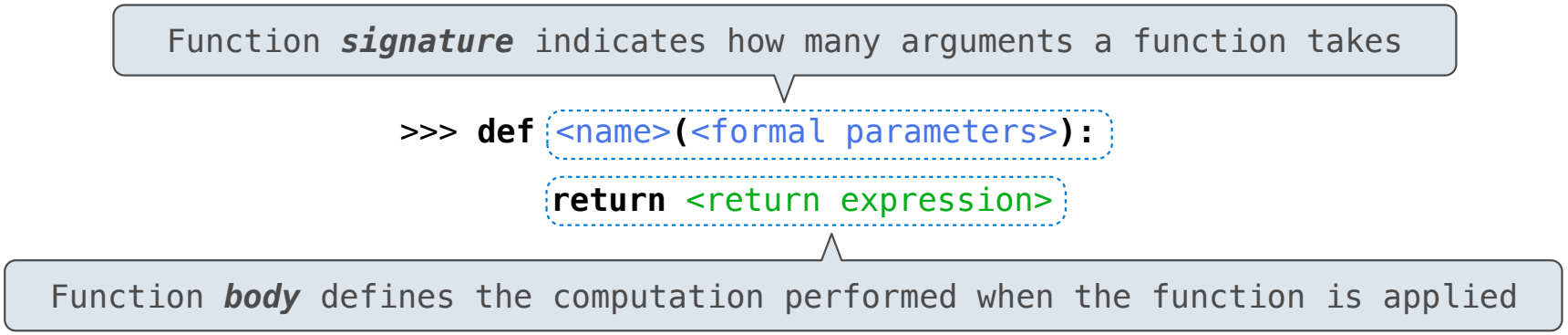
Execution procedure for def statements:

1. Create a function with signature `<name>(<formal parameters>)`
2. Set the body of that function to be everything indented after the first line

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions



Execution procedure for def statements:

1. Create a function with signature <name>(<formal parameters>)
2. Set the body of that function to be everything indented after the first line
3. Bind <name> to that function in the current frame

Calling User-Defined Functions

Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment

Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame

Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

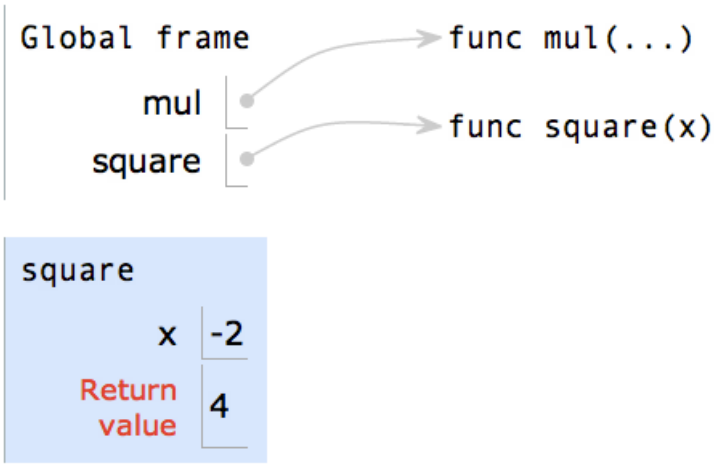
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



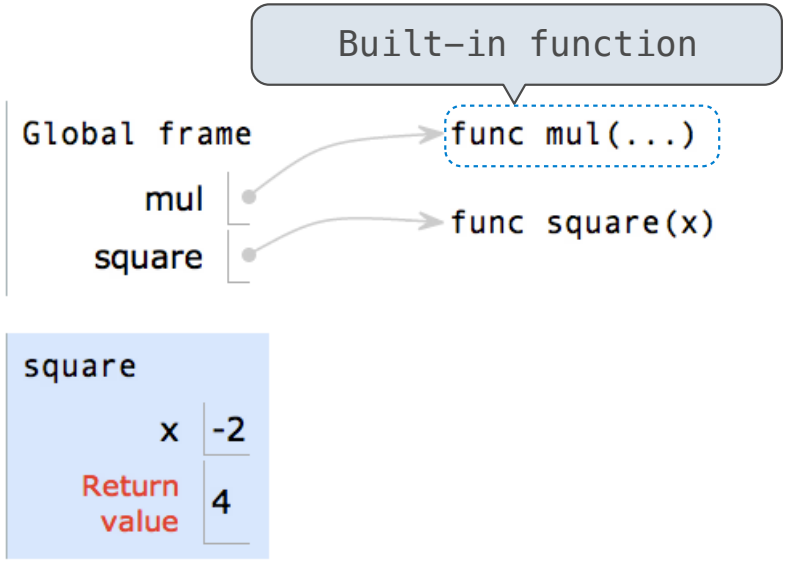
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



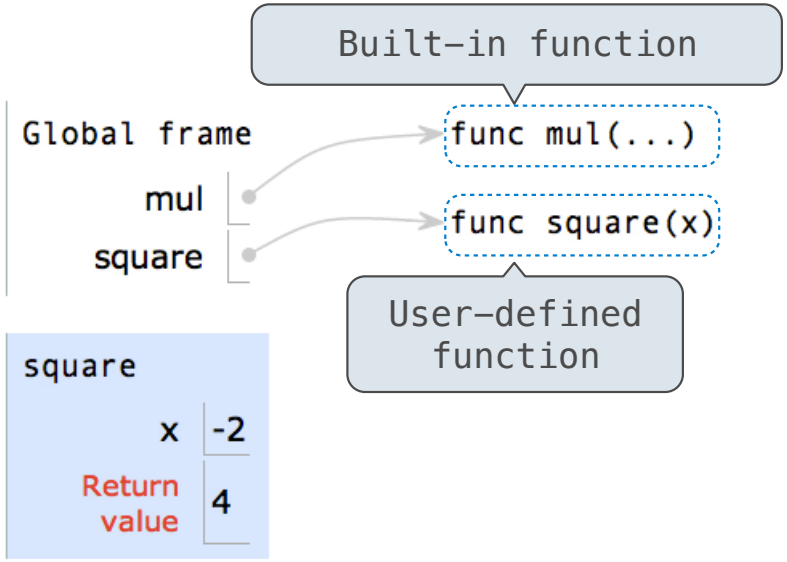
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



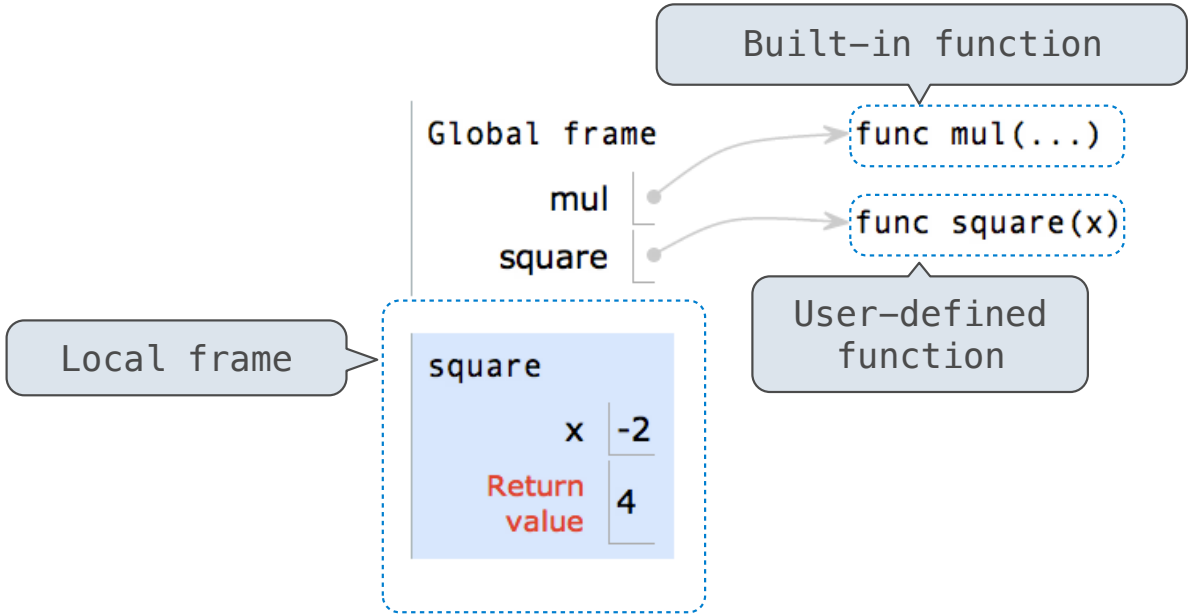
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



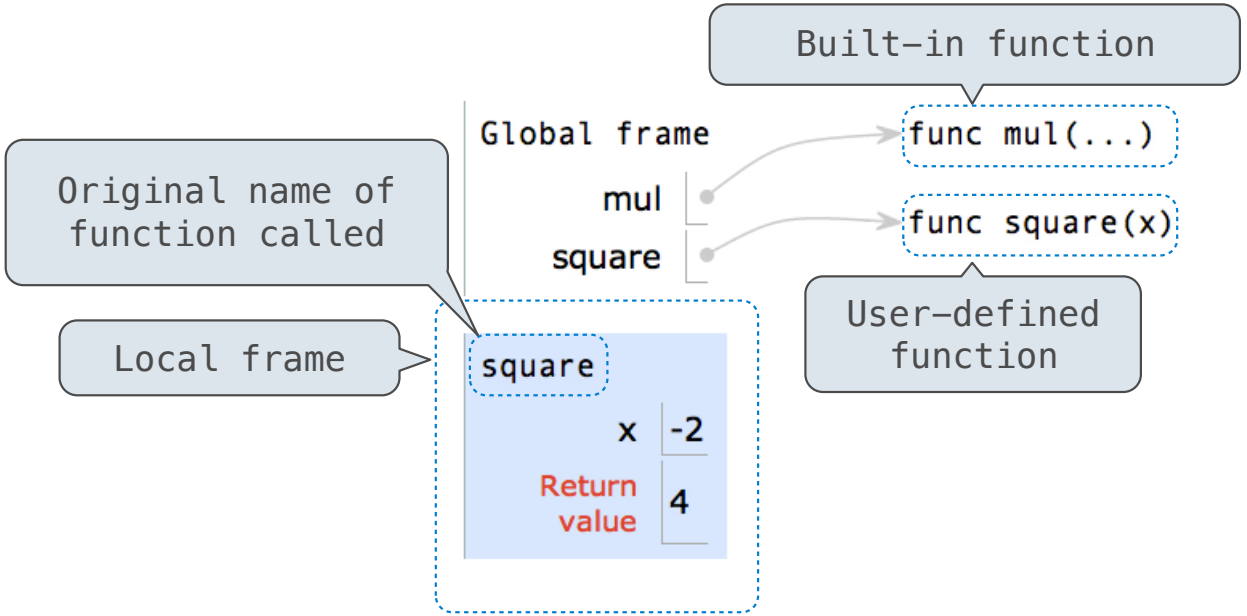
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



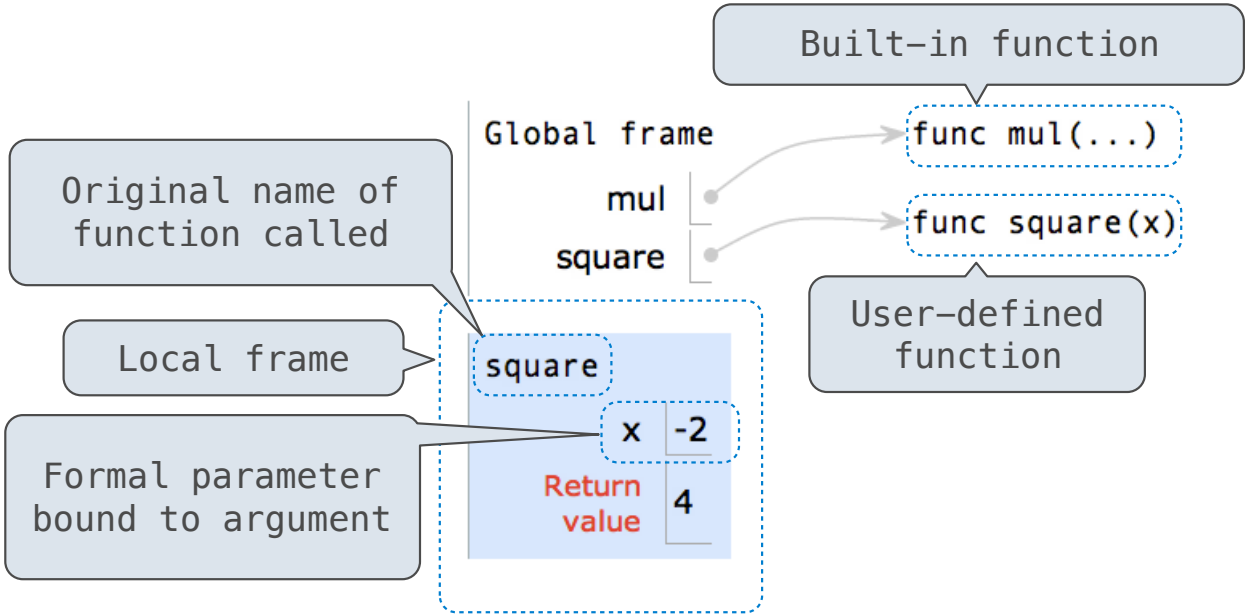
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



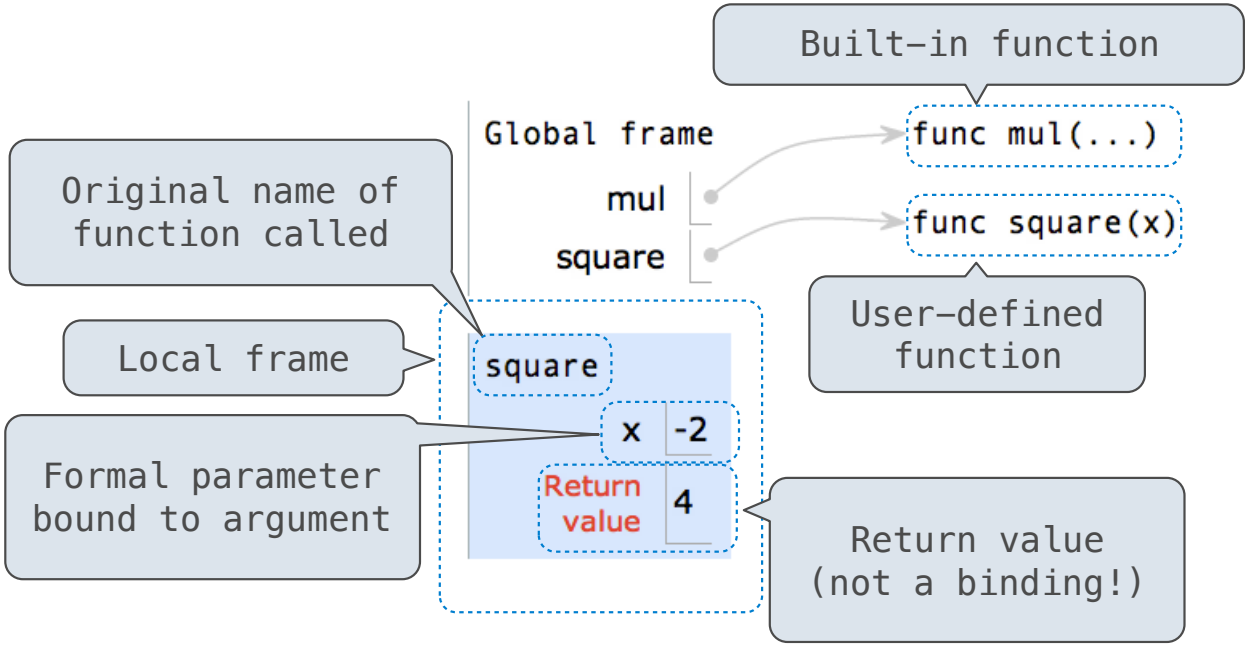
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



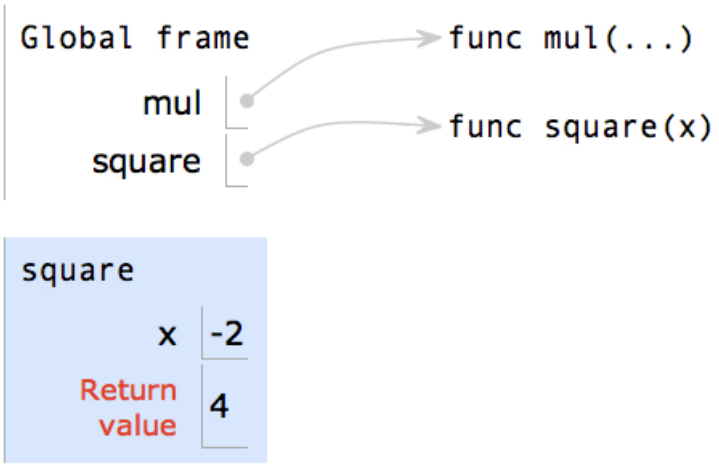
Interactive Diagram

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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



Interactive Diagram

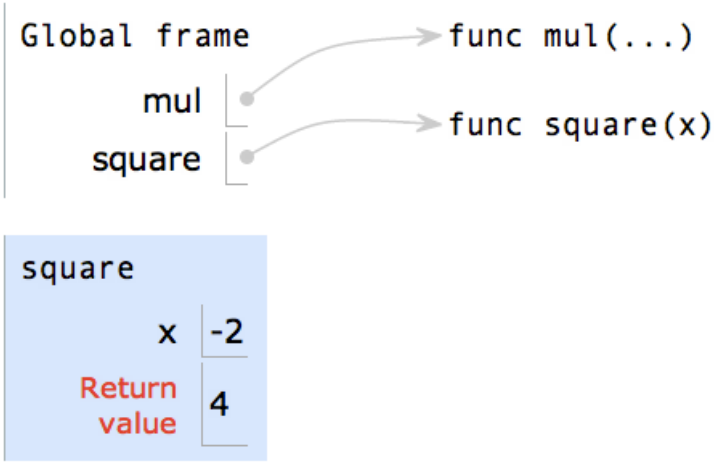
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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

A function's signature has all the information needed to create a local frame



Interactive Diagram

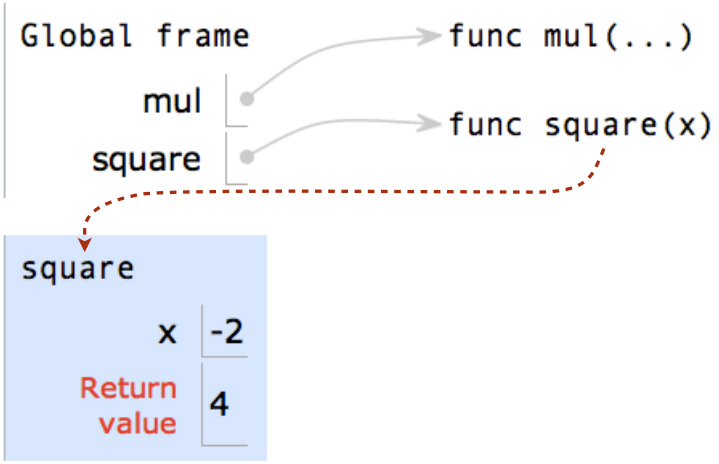
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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

A function's signature has all the information needed to create a local frame



Interactive Diagram

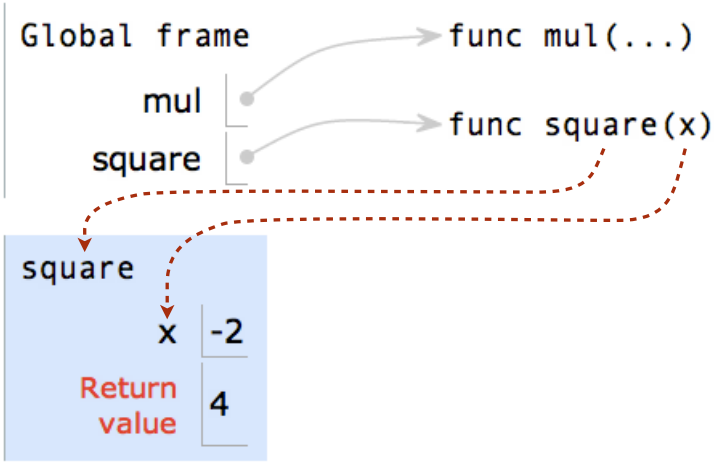
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

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

A function's signature has all the information needed to create a local frame



Interactive Diagram

Looking Up Names In Environments

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:

- Look for that name in the local frame.

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:

- Look for that name in the local frame.
- If not found, look for it in the global frame.
  (Built-in names like "max" are in the global frame too,
  but we don't draw them in environment diagrams.)

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:

- Look for that name in the local frame.
- If not found, look for it in the global frame.
  (Built-in names like "max" are in the global frame too,
  but we don't draw them in environment diagrams.)

(Demo)