CS61A Lecture 3

Amir Kamil
UC Berkeley
January 28, 2013

Announcements

- Reminder: hw0 due tonight, hw1 due Wed.
- In-class quiz on Friday
  - Covers through Wednesday’s lecture
  - Bring a writing implement
- Hog project out
  - Get started early!
  - More on hog next time

The Elements of Programming

The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language

The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language
- Means of Combination
  - Compound elements built from simpler ones

The Elements of Programming

- Primitive Expressions and Statements
  - The simplest building blocks of a language
- Means of Combination
  - Compound elements built from simpler ones
- Means of Abstraction
  - Elements can be named and manipulated as units

Environment Diagrams

Environment diagrams visualize the interpreter's process.

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

Global frame	
pi	3.1416

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.

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

Global frame	
pi	3.1416

Code (left):

Frames (right):

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.

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

Global frame	
pi	3.1416

Code (left):

Statements and expressions

Frames (right):

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.

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

Global frame	
pi	3.1416

Code (left):

Statements and expressions

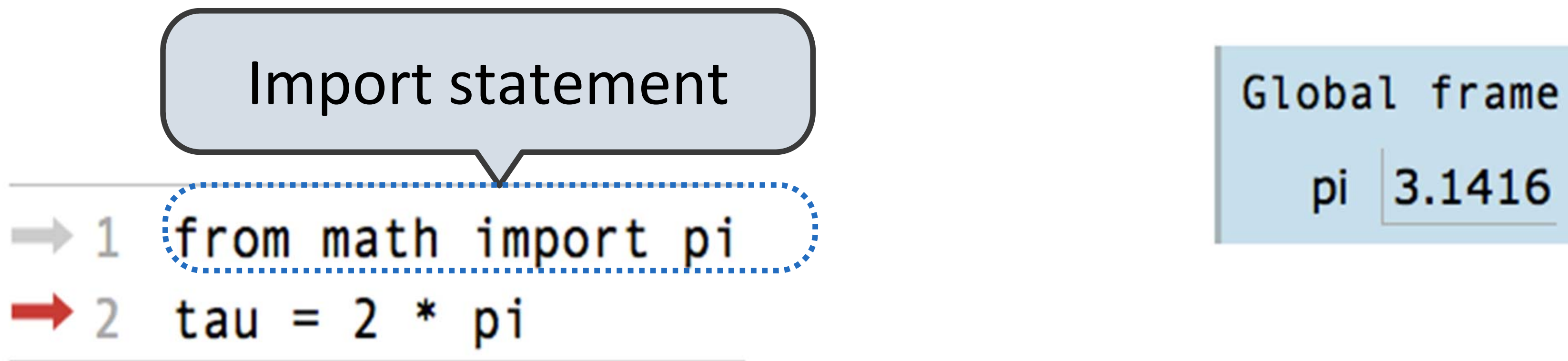
Next line is highlighted

Frames (right):

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.



Code (left):

Statements and expressions

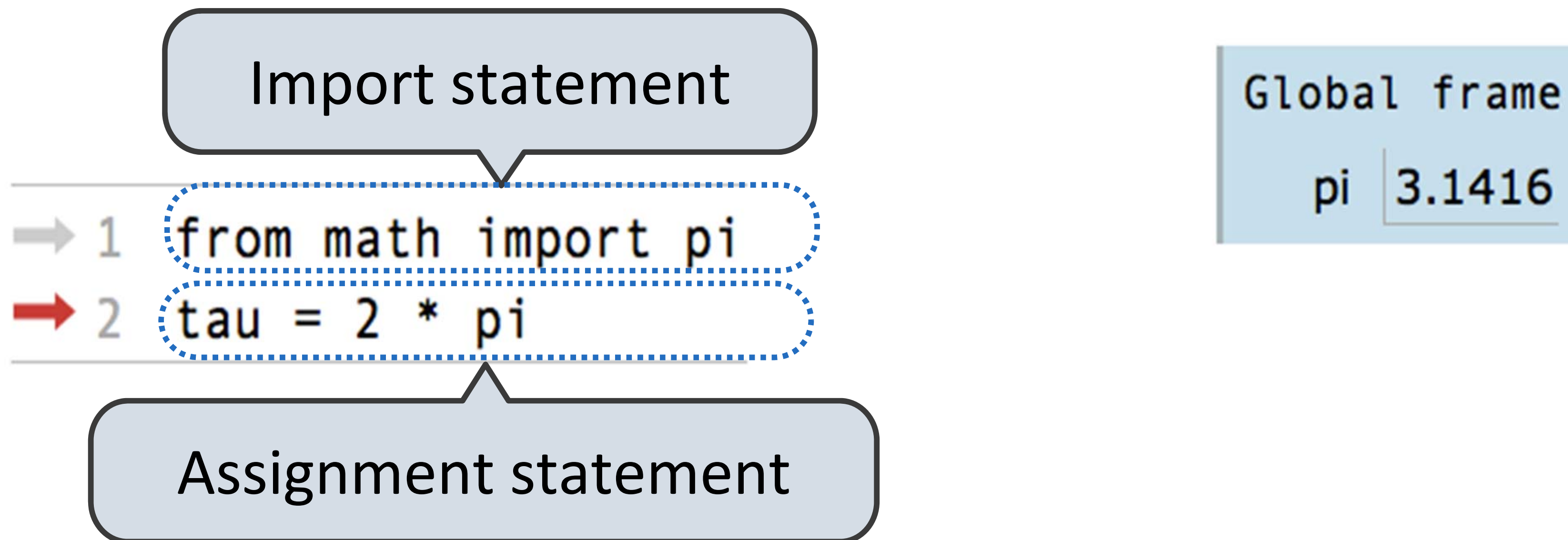
Next line is highlighted

Frames (right):

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.



Code (left):

Frames (right):

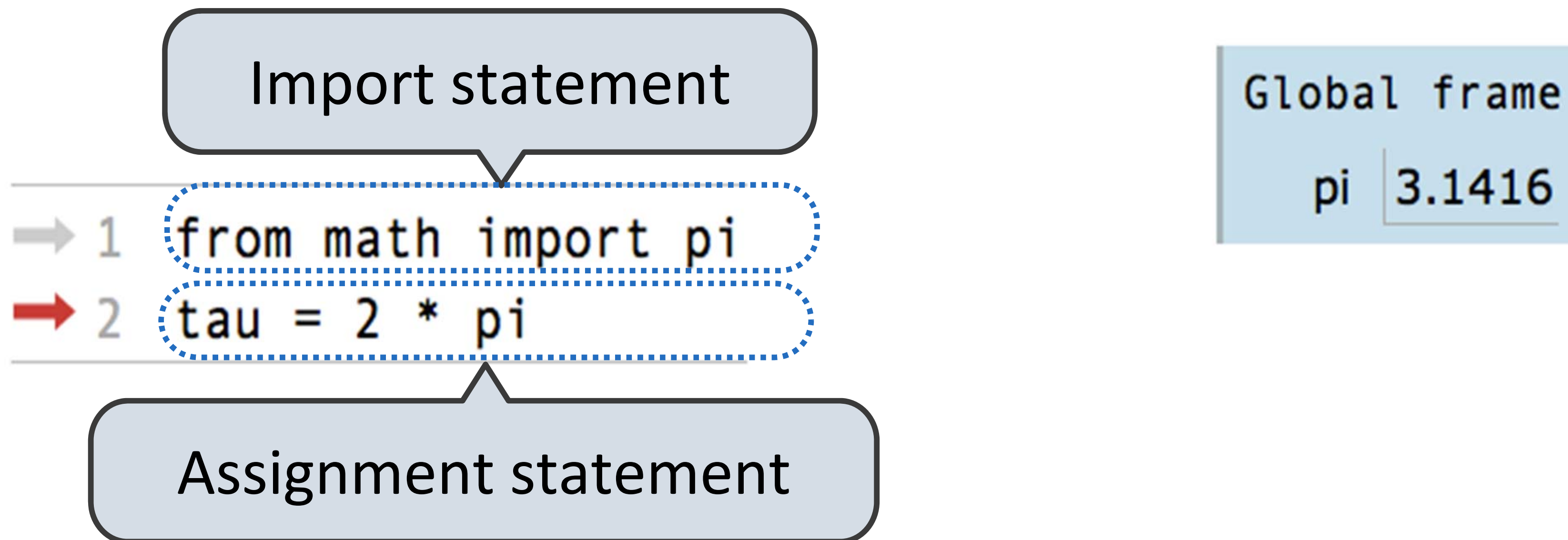
Statements and expressions

Next line is highlighted

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.



Code (left):

Statements and expressions

Next line is highlighted

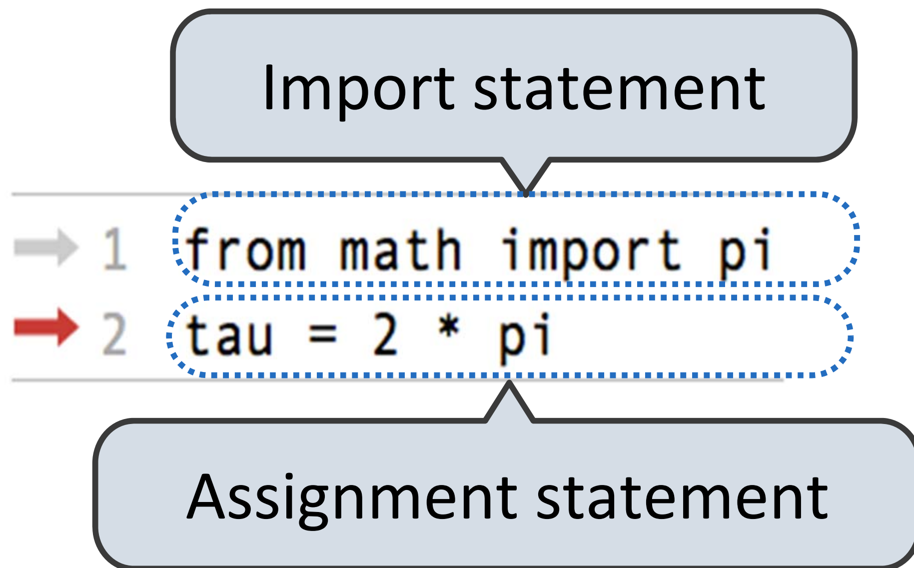
Frames (right):

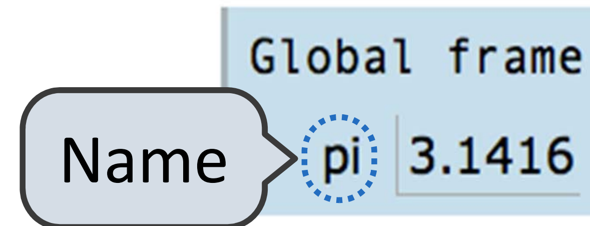
A name is bound to a value

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Next line is highlighted

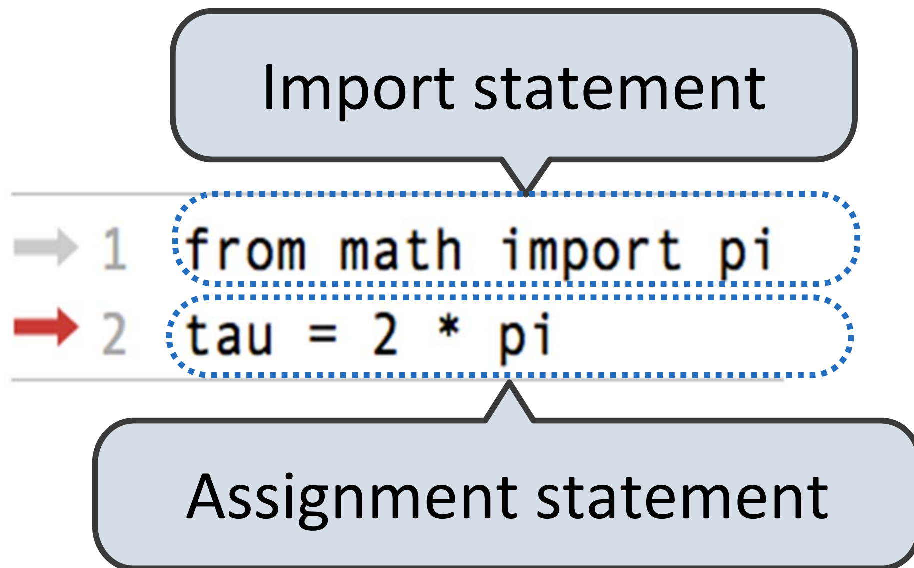
Frames (right):

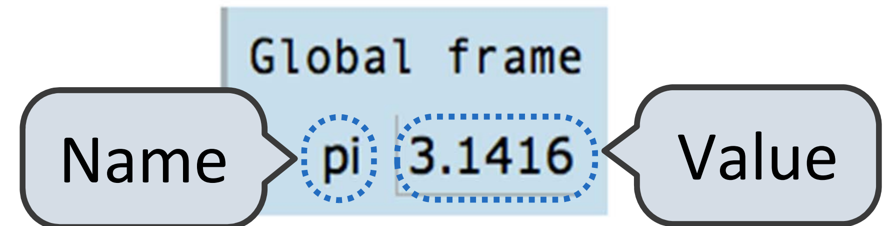
A name is bound to a value

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Next line is highlighted

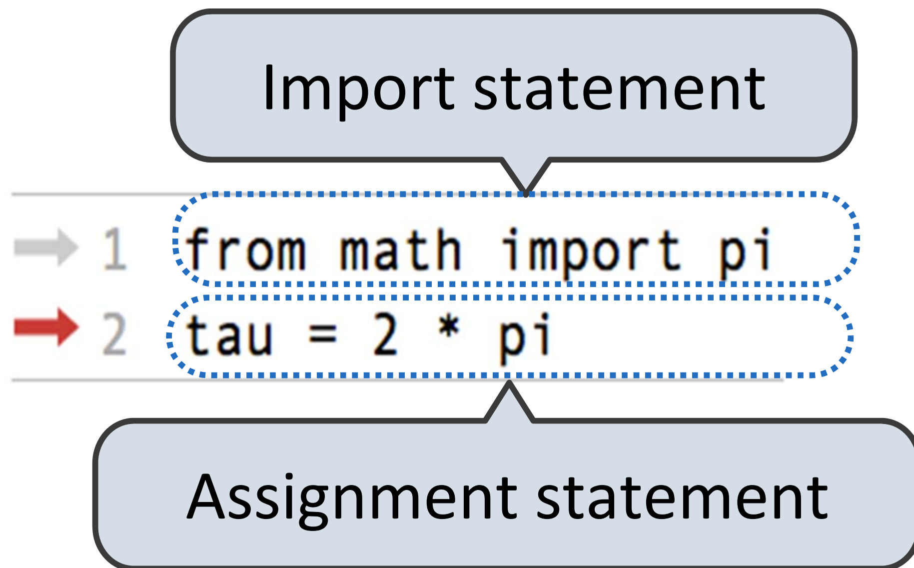
Frames (right):

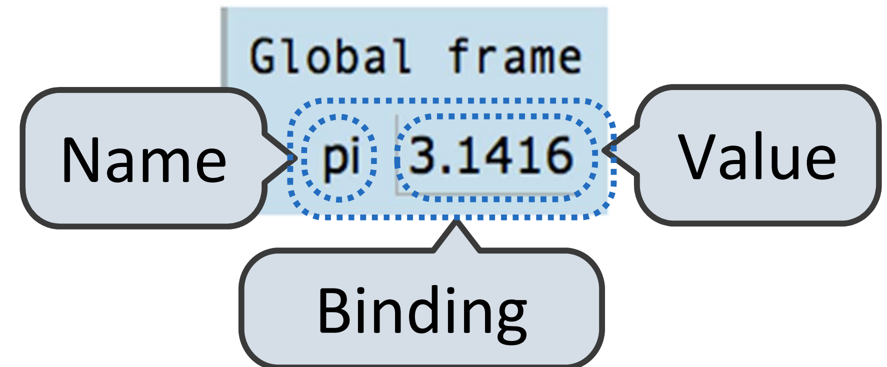
A name is bound to a value

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Next line is highlighted

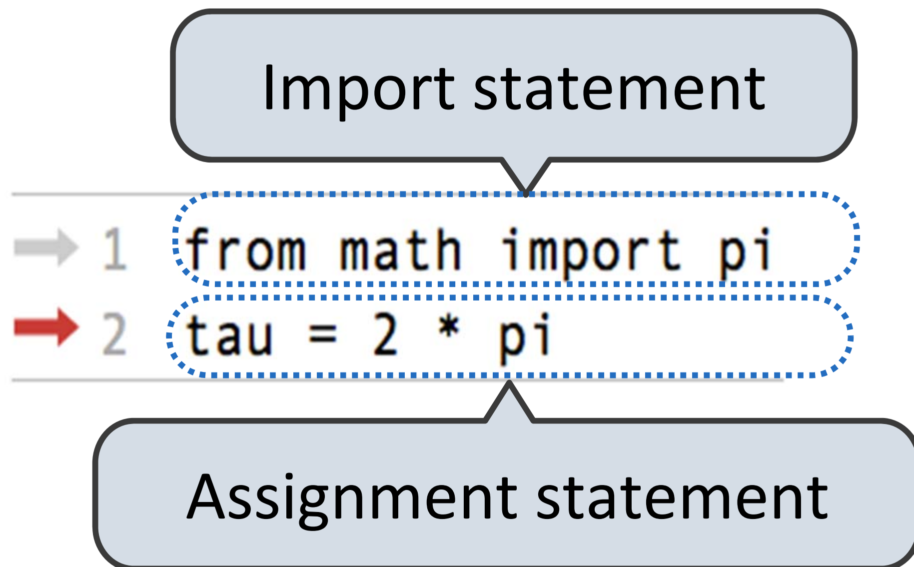
Frames (right):

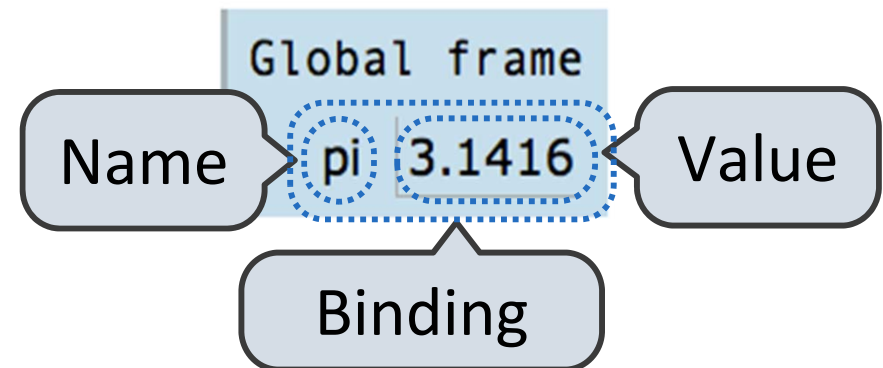
A name is bound to a value

Example: http://goo.gl/SK13i

Environment Diagrams

Environment diagrams visualize the interpreter's process.





Code (left):

Statements and expressions

Next line is highlighted

Frames (right):

A name is bound to a value

In a frame, there is at most one binding per name

Example: http://goo.gl/SK13i

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

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

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function “signature” indicates how many parameters

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

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function “signature” indicates how many parameters

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

Function “body” defines a computational process

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function "signature" indicates how many parameters

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

Function "body" defines a computational process

Execution procedure for def statements:

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function "signature" indicates how many parameters

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

Function "body" defines a computational process

Execution procedure for def statements:

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

User-Defined Functions

Named values are a simple means of abstraction

Named computational processes are a more powerful means of abstraction

Function "signature" indicates how many parameters

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

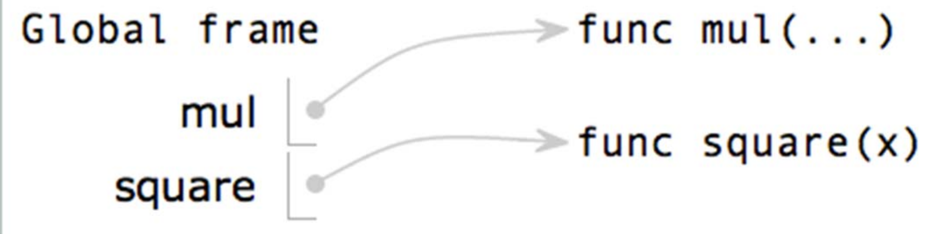
Function "body" defines a computational process

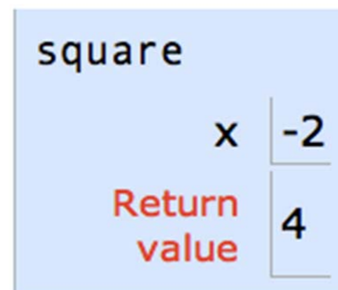
Execution procedure for def statements:

1. Create a function value with signature <name>(<formal parameters>)
2. Bind <name> to that value in the current frame

Calling User-Defined Functions

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



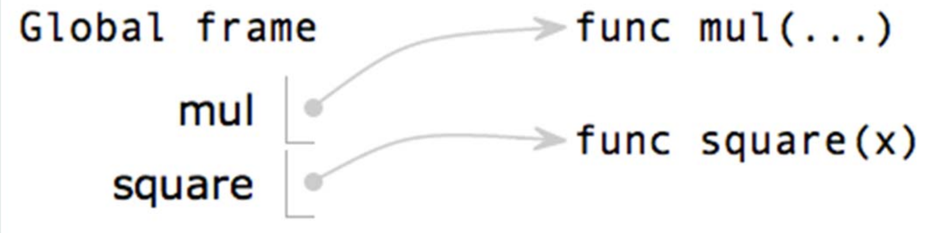


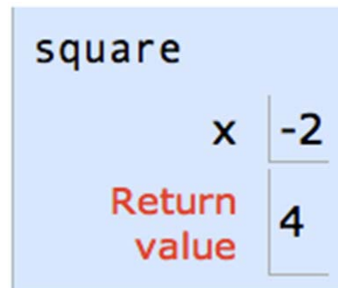
Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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





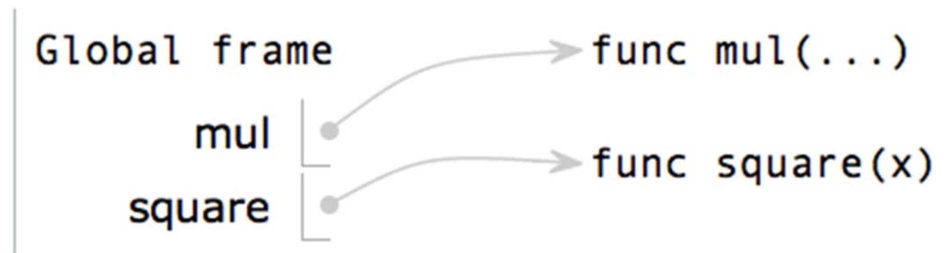
Example: http://goo.gl/boCk0

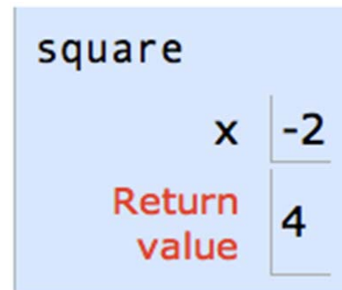
Calling User-Defined Functions

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

1. Add a local frame

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





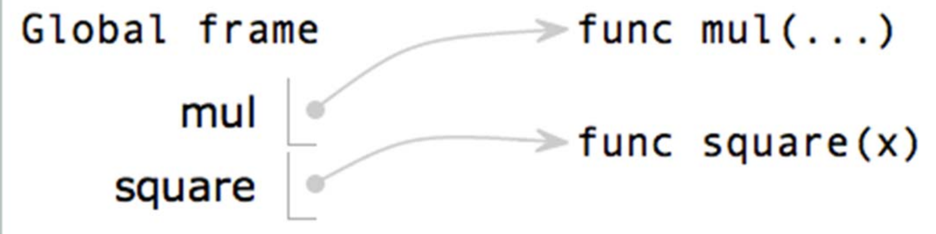
Example: http://goo.gl/boCk0

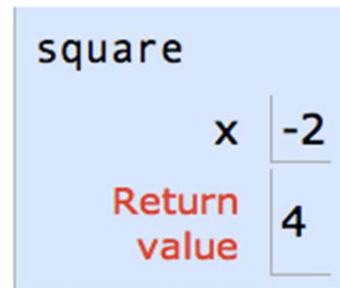
Calling User-Defined Functions

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

1. Add a local frame
2. Bind formal parameters to arguments in that frame

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





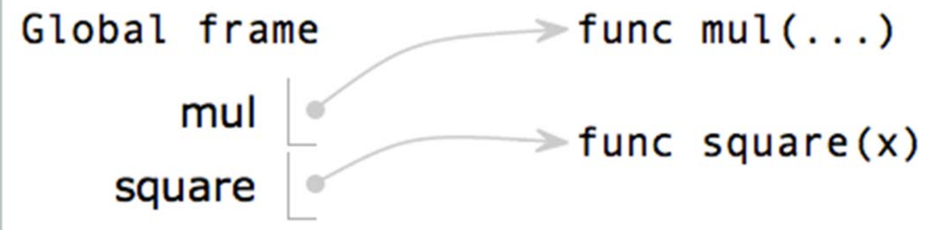
Example: http://goo.gl/boCk0

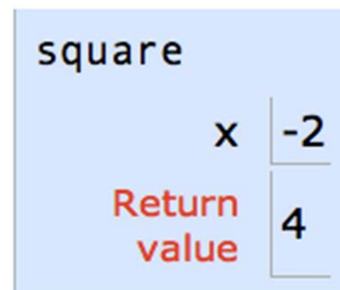
Calling User-Defined Functions

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

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

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





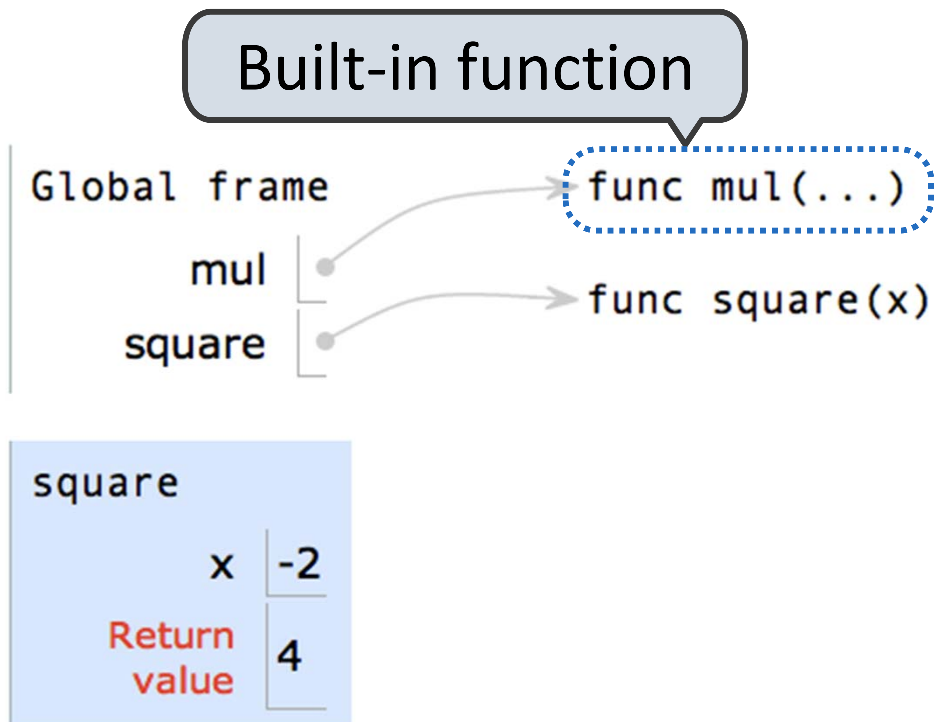
Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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



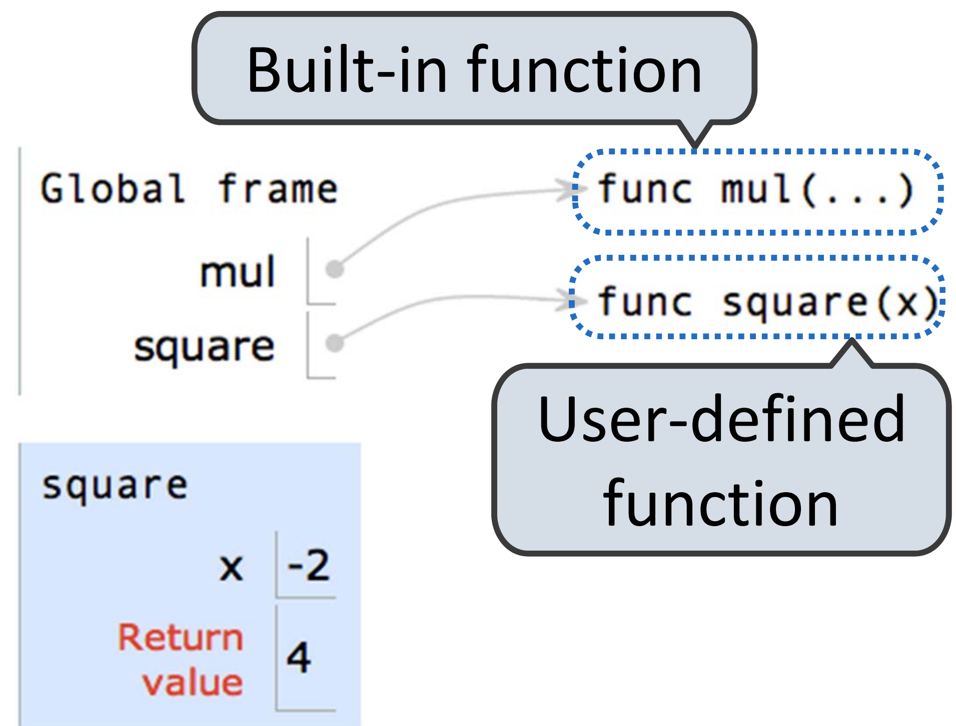
Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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



Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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


Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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


Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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


Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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

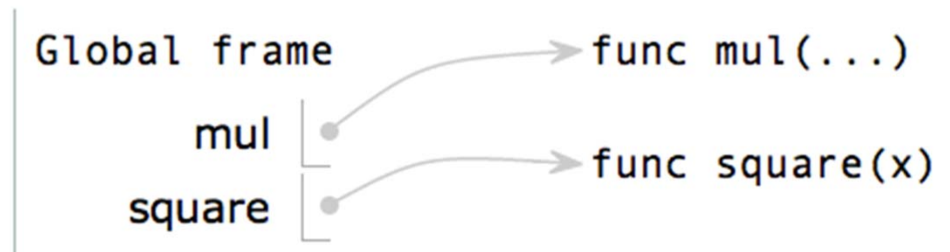

Example: http://goo.gl/boCk0

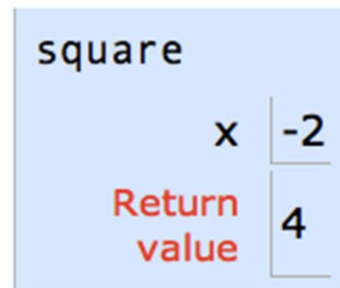
Calling User-Defined Functions

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

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

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





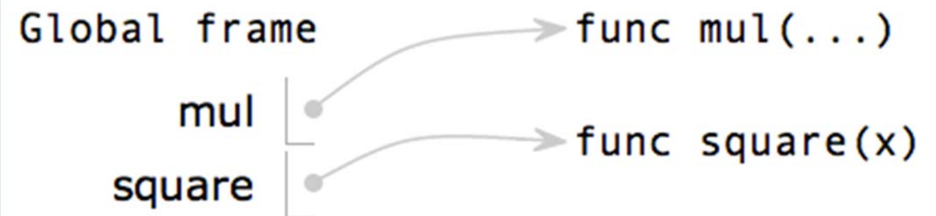
Example: http://goo.gl/boCk0

Calling User-Defined Functions

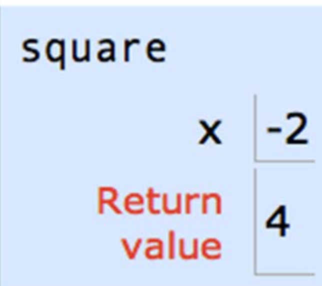
Procedure for applying user-defined functions (version 1):

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the 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 to create a local frame



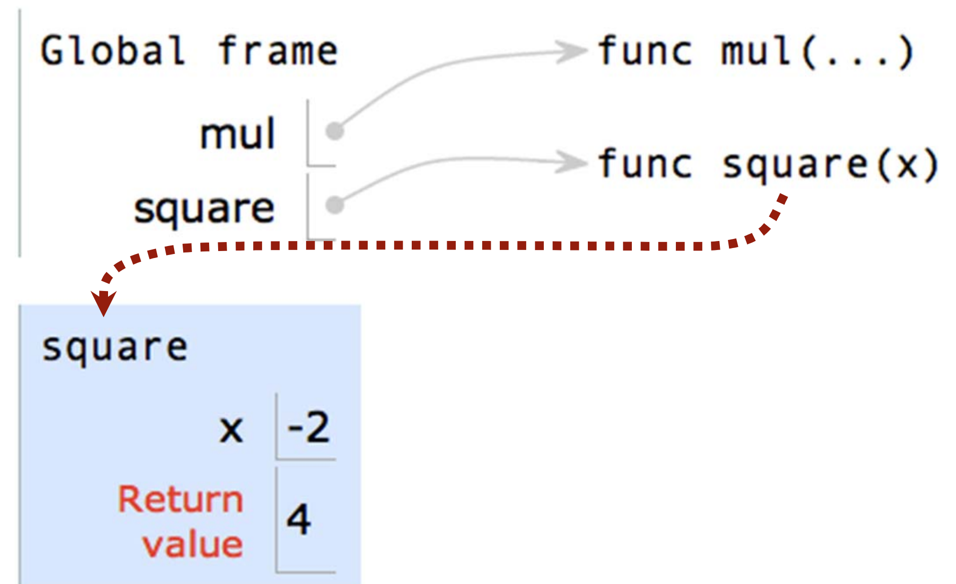
Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

1. Add a local frame
2. Bind formal parameters to arguments in that frame
3. Execute the body of the function in the 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 to create a local frame

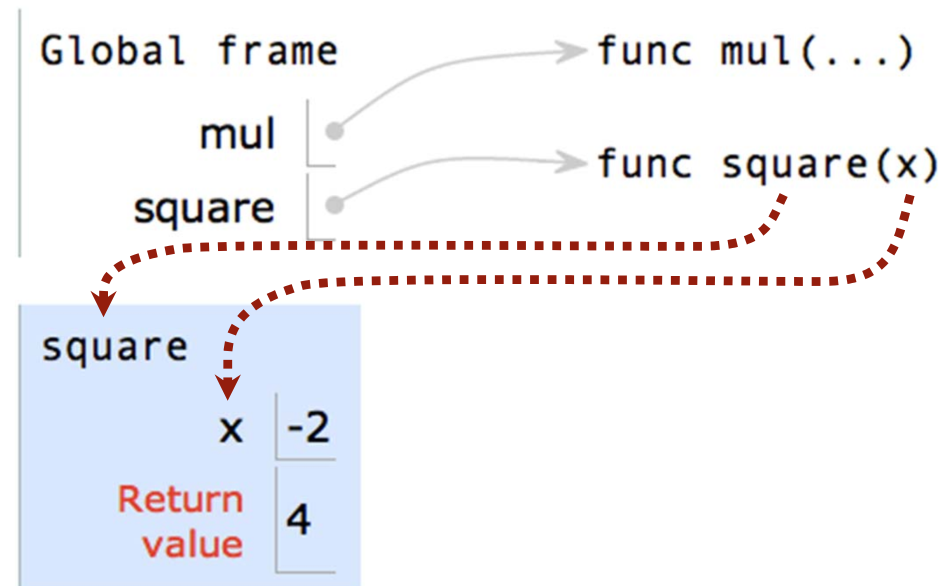
Example: http://goo.gl/boCk0

Calling User-Defined Functions

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

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

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



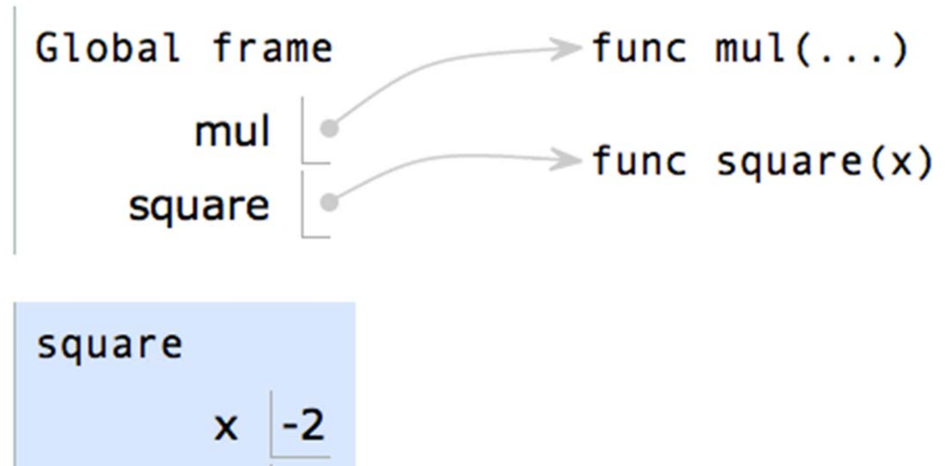
A function's signature has all the information to create a local frame

Example: http://goo.gl/boCk0

Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

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



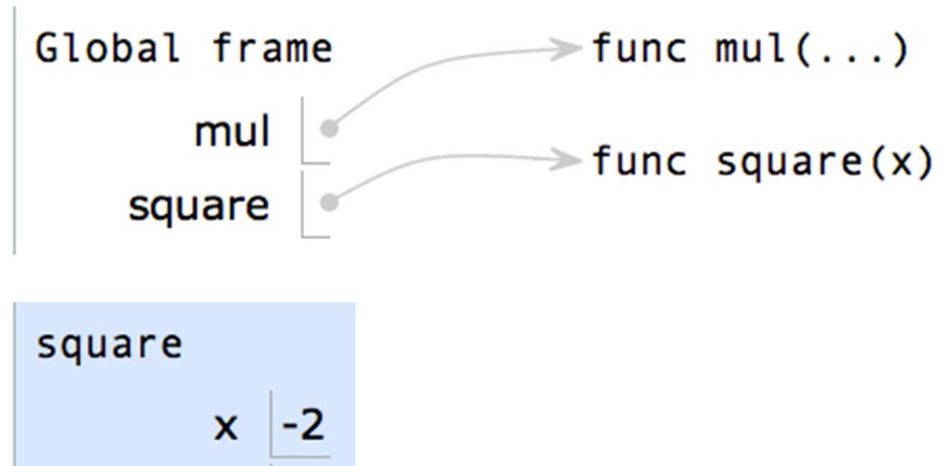
Example: http://goo.gl/boCk0

Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame

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



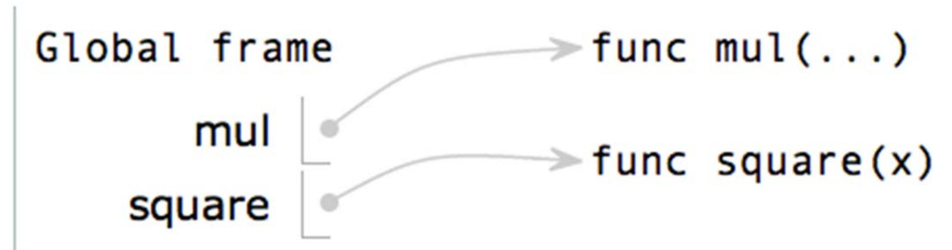
Example: http://goo.gl/boCk0

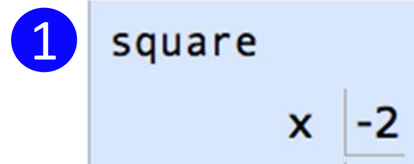
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame

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





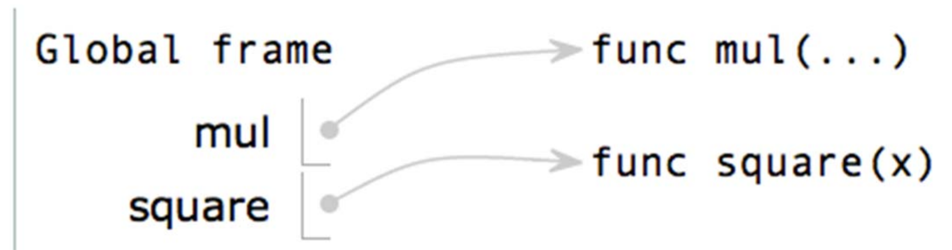
Example: http://goo.gl/boCk0

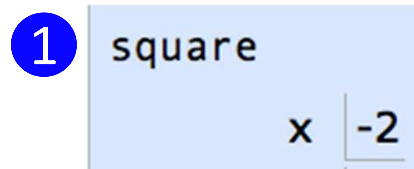
Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame

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





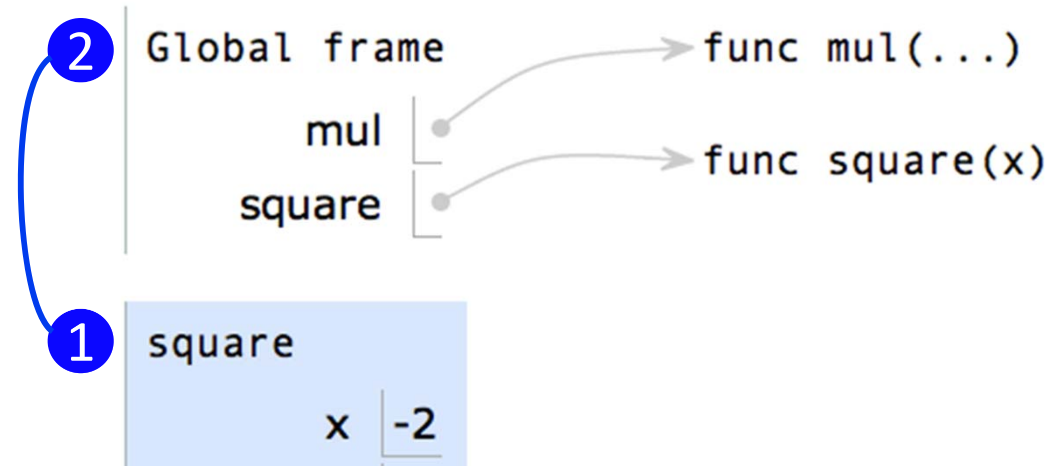
Example: http://goo.gl/boCk0

Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame

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



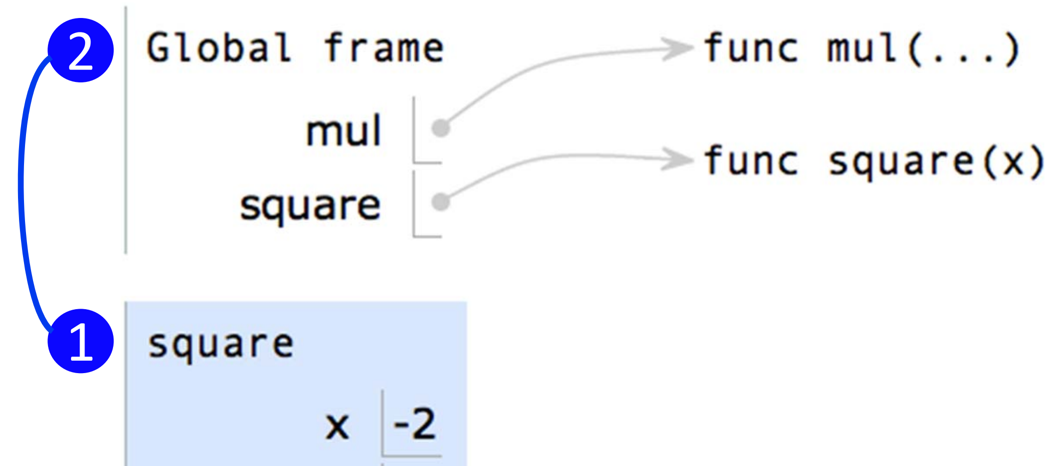
Example: http://goo.gl/boCk0

Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error

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

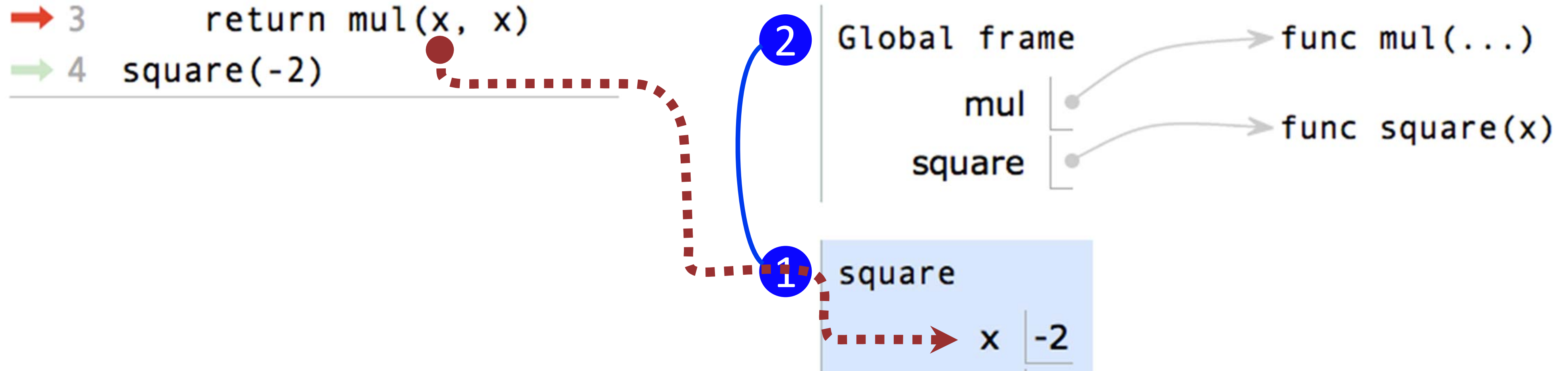


Example: http://goo.gl/boCk0

Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error



Example: http://goo.gl/boCk0

Looking Up Names

Procedure for looking up a name from inside a function (v. 1):

1. Look it up in the local frame
2. If not in local frame, look it up in the global frame
3. If in neither frame, generate error

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


Example: http://goo.gl/boCk0

General Lookup Procedure

General Lookup Procedure

- Every expression is evaluated in the context of an environment

General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:

General Lookup Procedure

- Every expression is evaluated in the context of an environment
- So far, the current environment is either:
  - The global frame alone, or

General Lookup Procedure

- 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

General Lookup Procedure

- 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
- **Important properties of environments:**

General Lookup Procedure

- 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
- **Important properties of environments:**
  - An environment is a sequence of frames

General Lookup Procedure

- 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

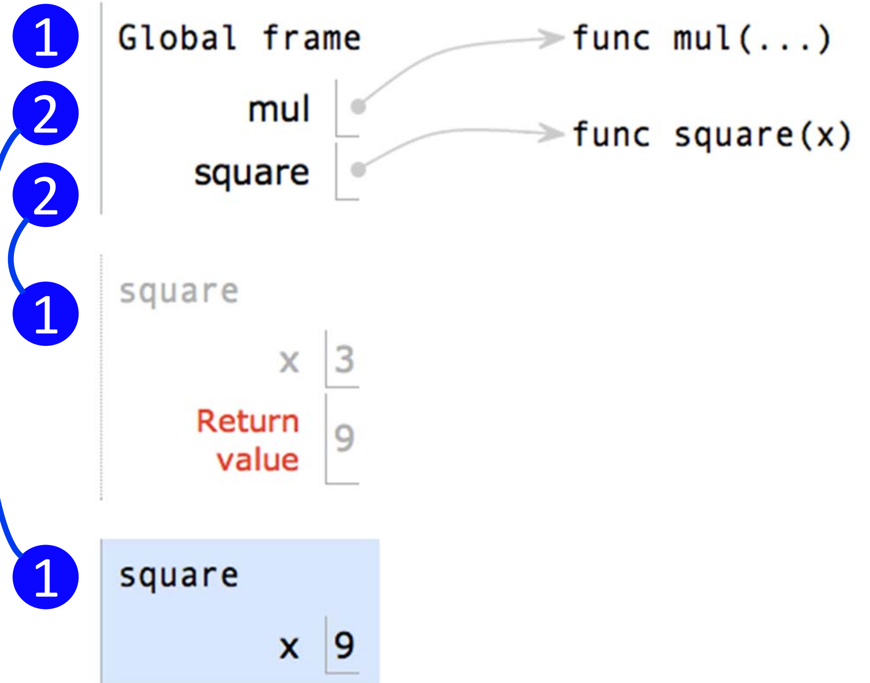
- **Important properties of environments:**
  - An environment is a sequence of frames
  - The earliest frame that contains a binding for a name determines the value that the name evaluates to

General Lookup Procedure

- 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
- **Important properties of environments:**
  - An environment is a sequence of frames
  - The earliest frame that contains a binding for a name determines the value that the name evaluates to
- The *scope* of a name is the region of code that has access to it

Multiple Environments in a Diagram

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

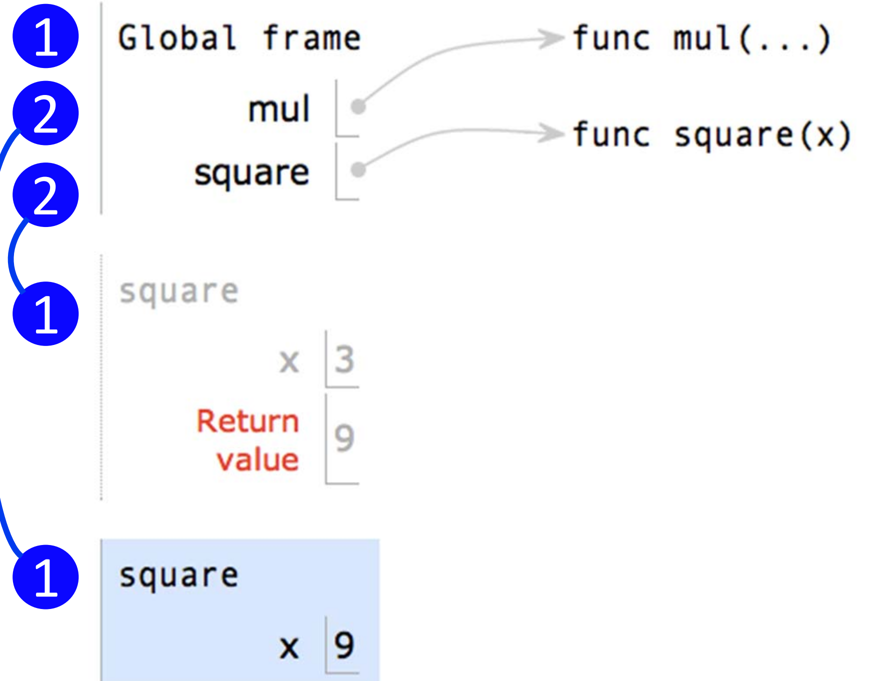


Example: http://goo.gl/hrfnV

Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

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



Example: http://goo.gl/hrfnV

Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

```
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)

square
x 3
Return value 9

square
x 9

Example: http://goo.gl/hrfnV

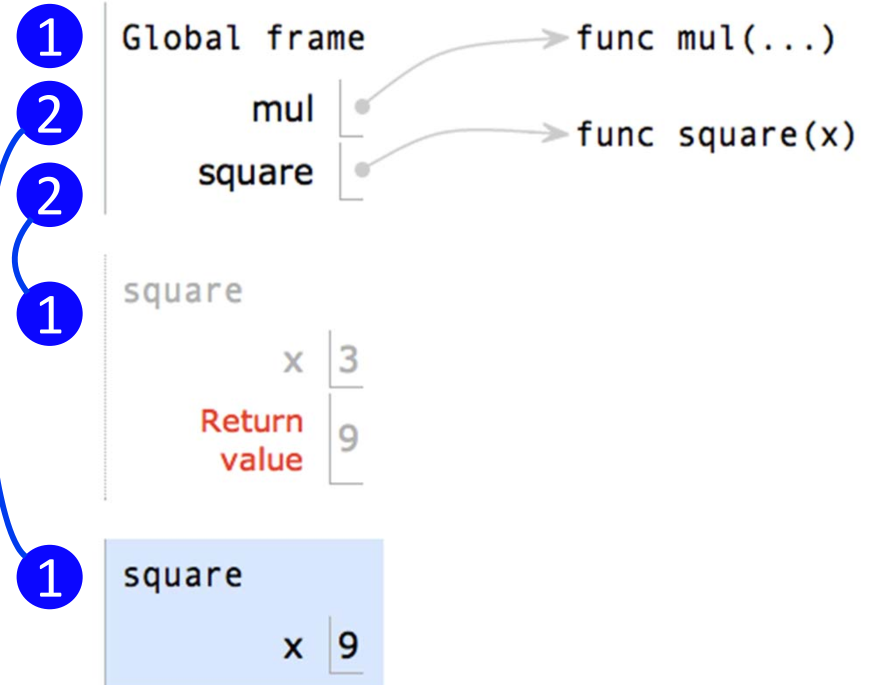
Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

mul(x, x)

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



Example: http://goo.gl/hrfnV

Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

mul(x, x)

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


Example: http://goo.gl/hrfnV

Multiple Environments in a Diagram

Every expression is evaluated in the context of an environment.

The earliest frame that contains a binding for a name determines the value that the name evaluates to.

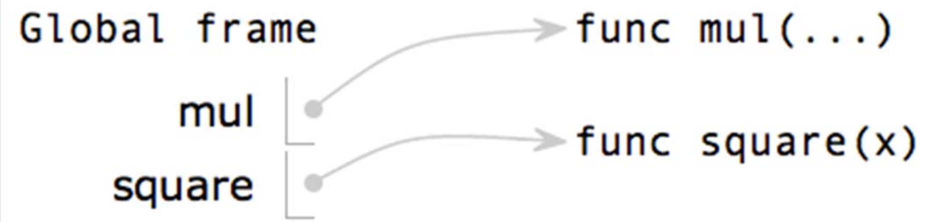
mul(x, x)

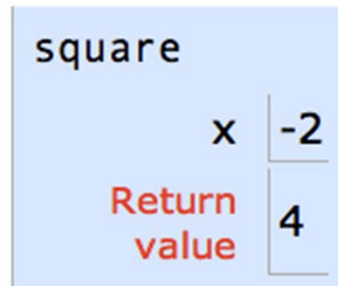
```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(square(3))
```


Example: http://goo.gl/hrfnV

Formal Parameters

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



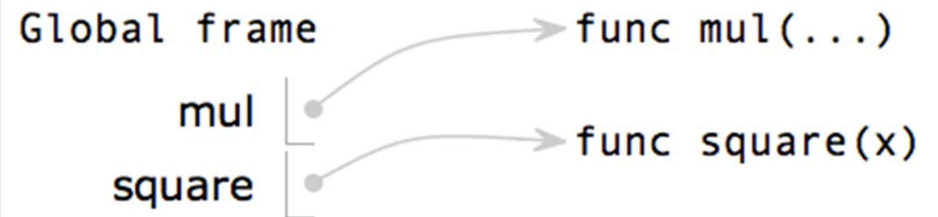


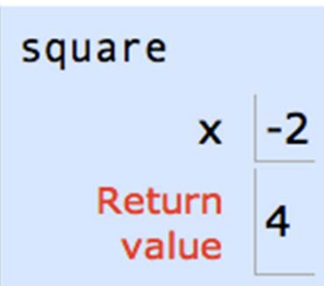
Example: http://goo.gl/boCk0

Formal Parameters

```
def square(x):
    return mul(x, x)
```

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





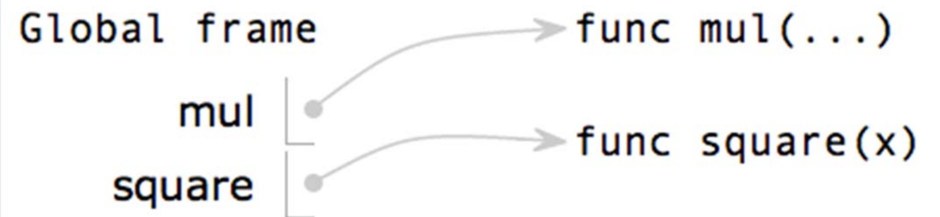
Example: http://goo.gl/boCk0

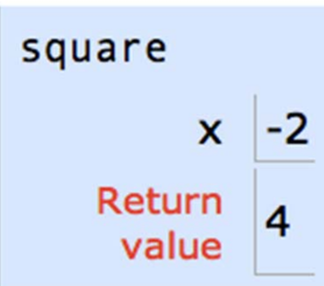
Formal Parameters

def square(x):
    return mul(x, x)

vs

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





Example: http://goo.gl/boCk0

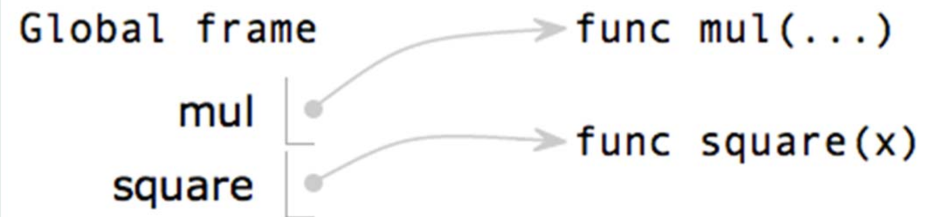
Formal Parameters

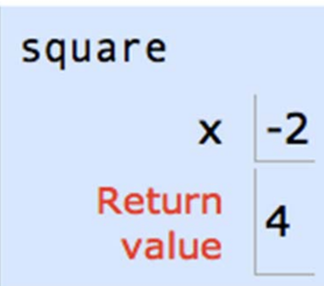
```
def square(x):
    return mul(x, x)
```

vs

```
def square(y):
    return mul(y, y)
```

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





Example: http://goo.gl/boCk0

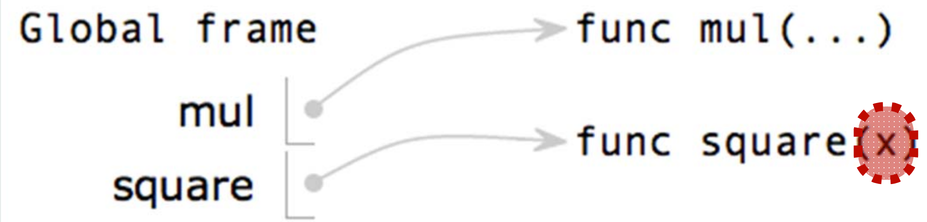
Formal Parameters

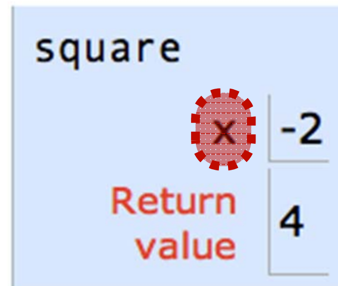
def square(x):
    return mul(x, x)

vs

def square(y):
    return mul(y, y)

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





Example: http://goo.gl/boCk0

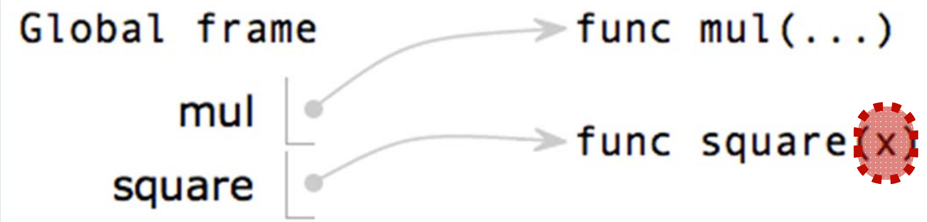
Formal Parameters

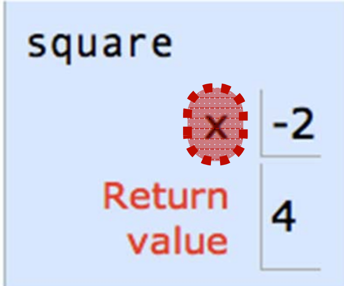
def square(x):
    return mul(x, x)

vs

def square(y):
    return mul(y, y)

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





Formal parameters have local scope

Example: http://goo.gl/boCk0

Life Cycle of a User-Defined Function

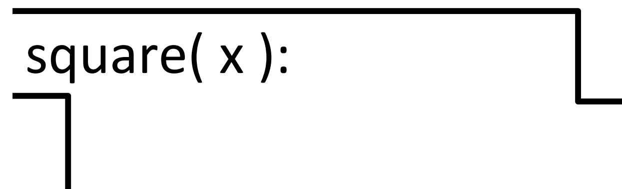
What happens?

Def statement:

```
>>> def    square( x ):
               return mul(x, x)
```

Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

What happens?

Def statement:

```
>>> def     square( x ):
                return mul(x, x)
```

Def statement

Call expression: square(2+2)

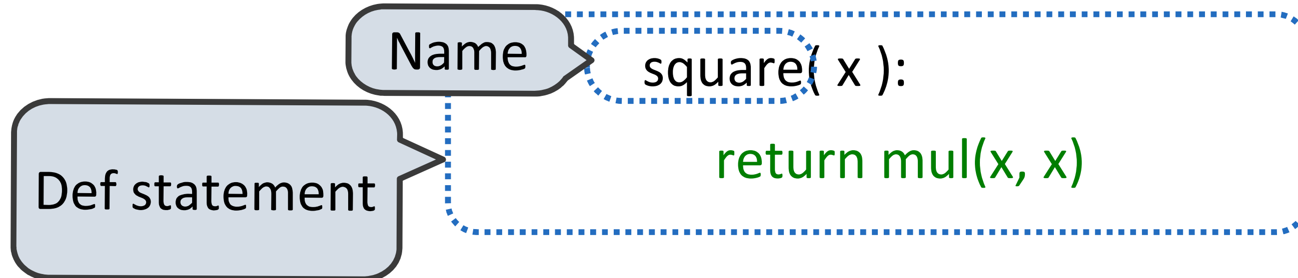
Calling/Applying:

square(x):

Life Cycle of a User-Defined Function

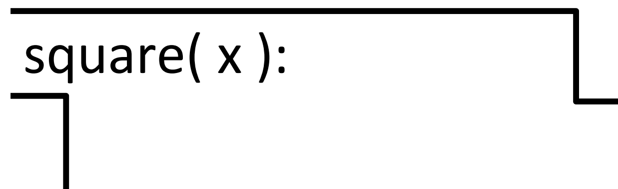
Def statement:

What happens?



Call expression: square(2+2)

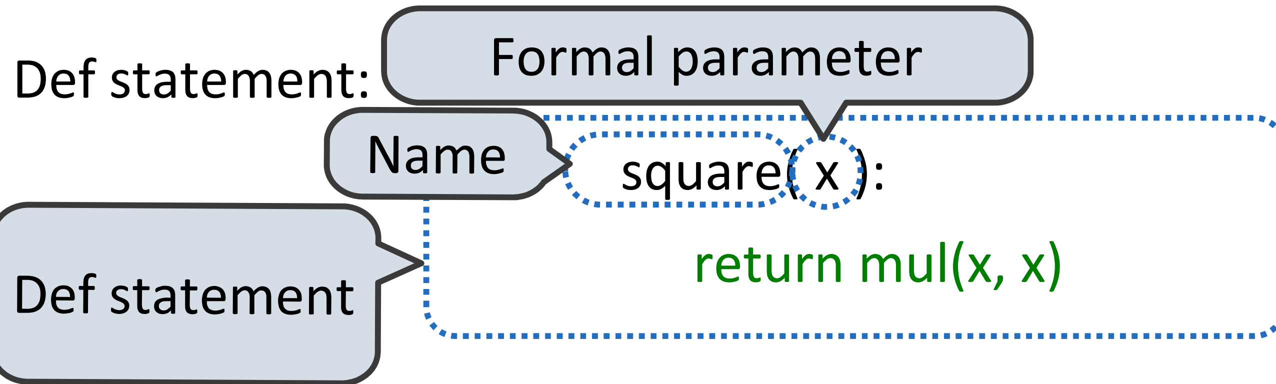
Calling/Applying:



Life Cycle of a User-Defined Function

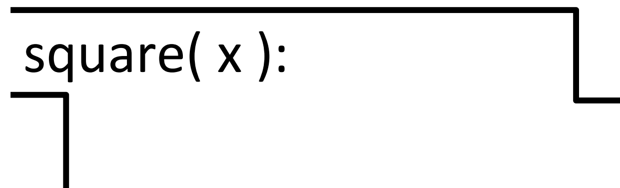
Def statement:

What happens?



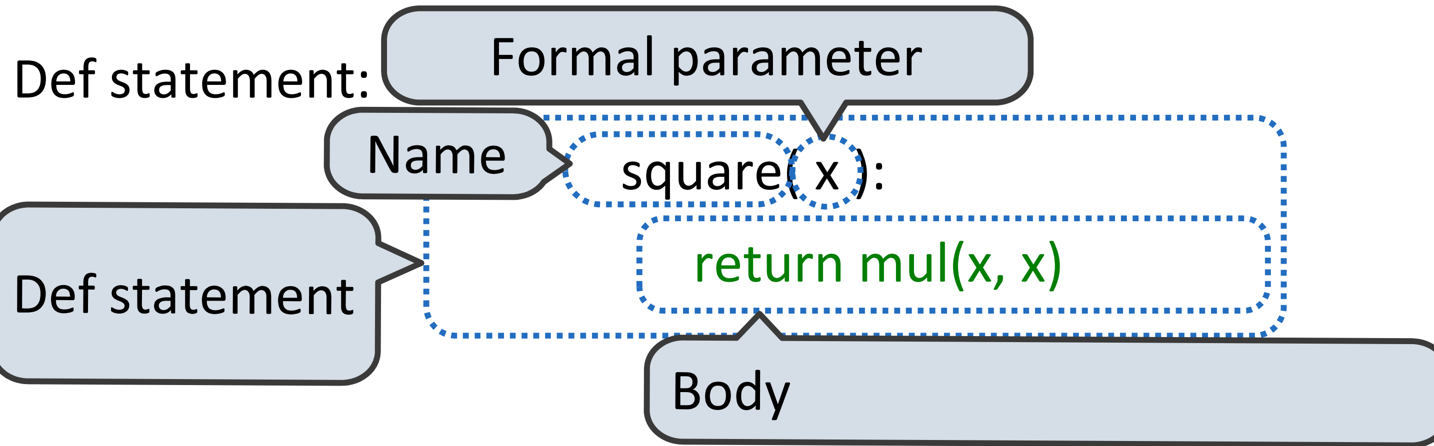
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

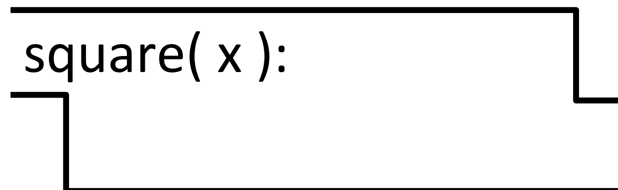
Def statement:



What happens?

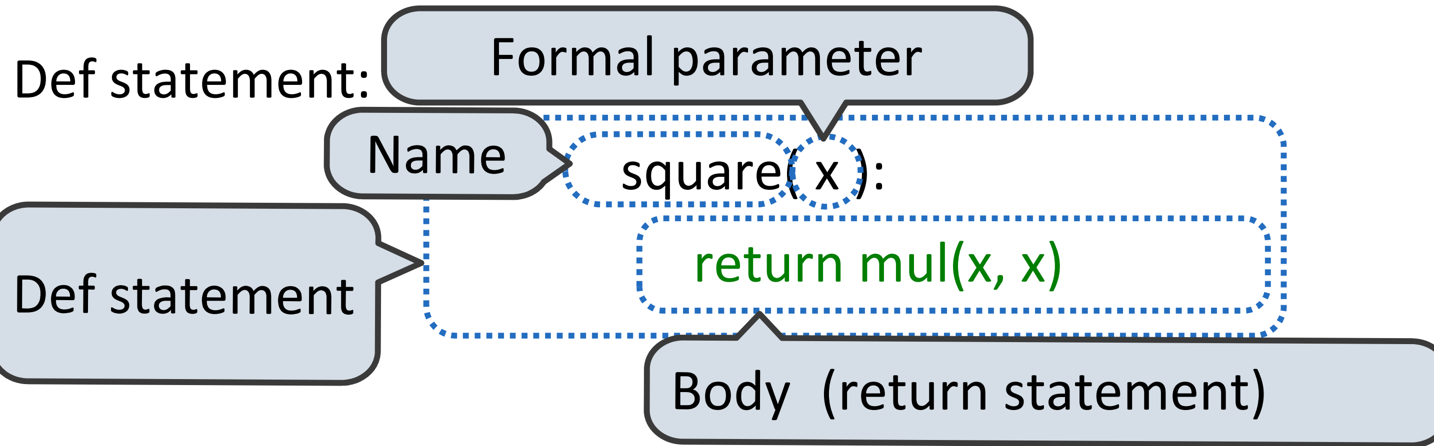
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

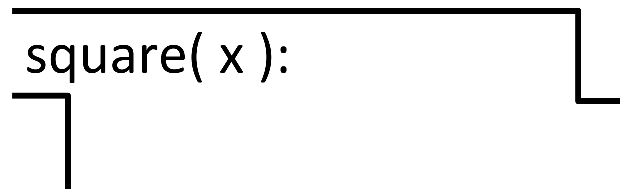
Def statement:



What happens?

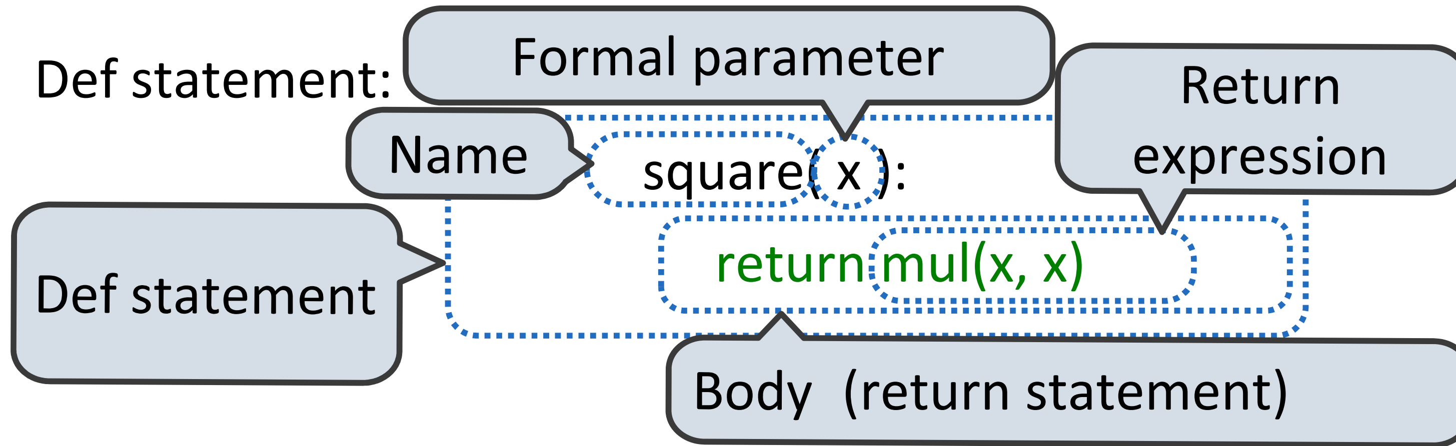
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

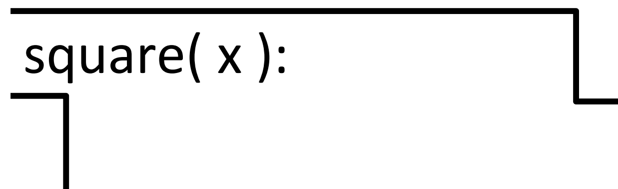
Def statement:



What happens?

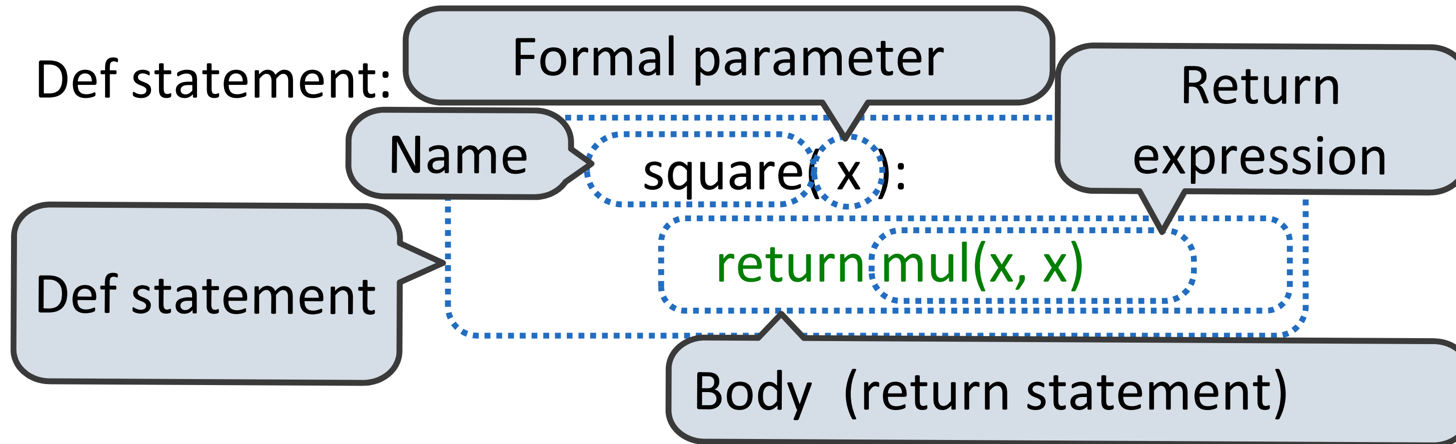
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

Def statement:

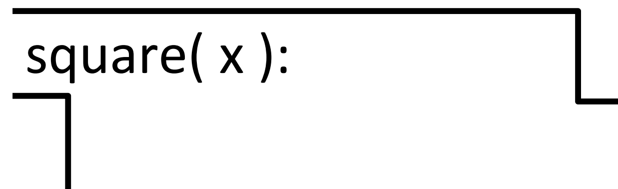


What happens?

Function created

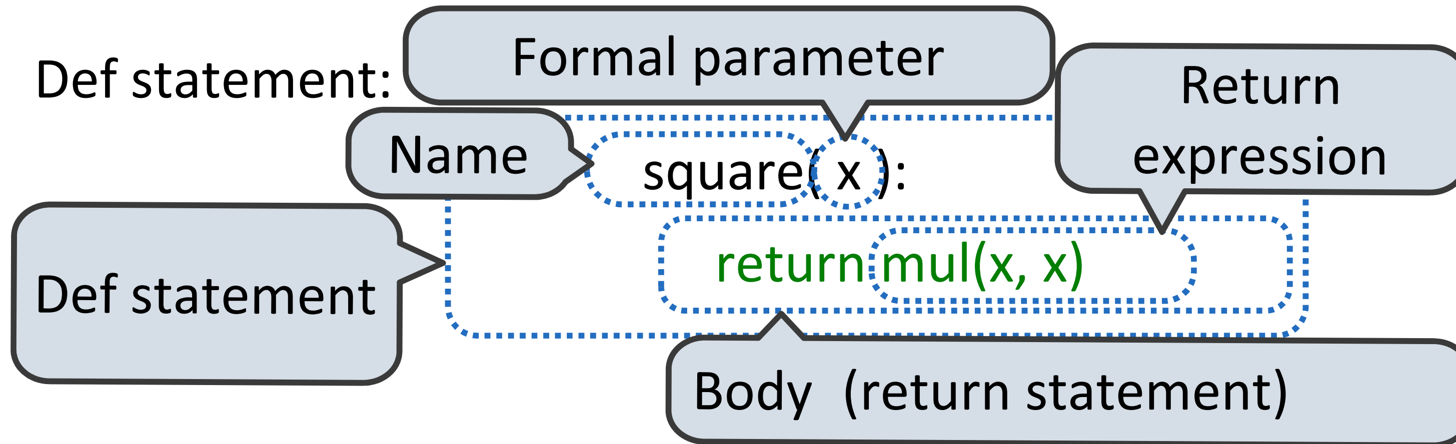
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

Def statement:



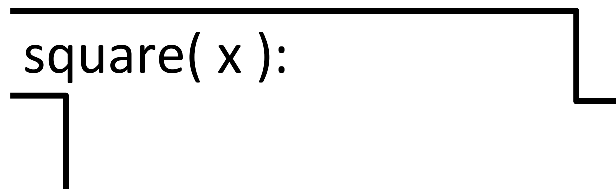
What happens?

Function created

Name bound

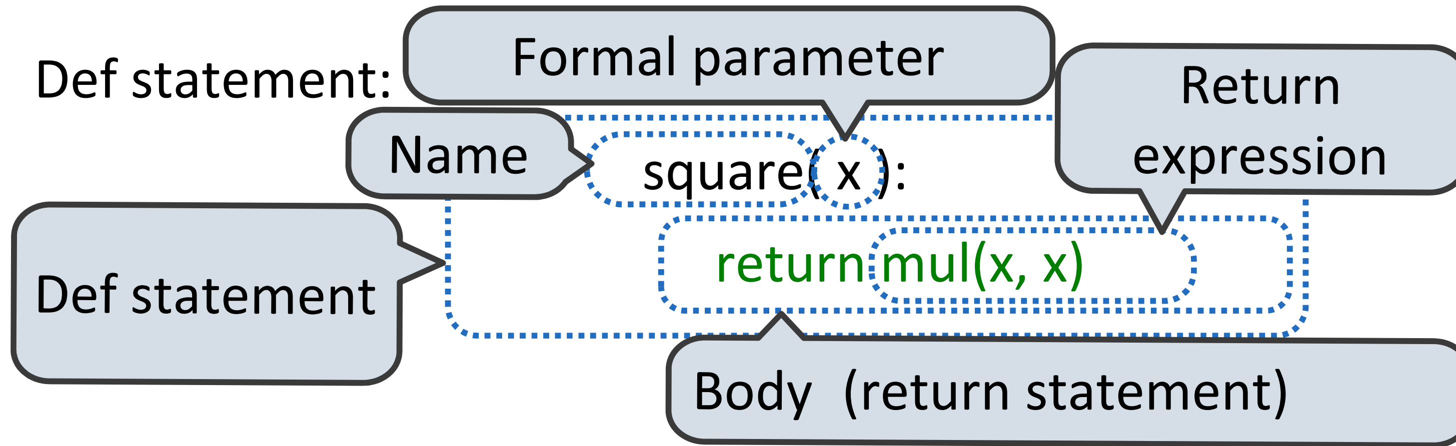
Call expression: square(2+2)

Calling/Applying:



Life Cycle of a User-Defined Function

Def statement:

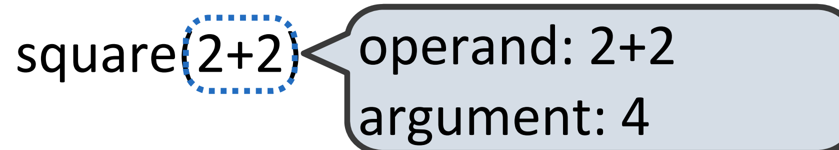


What happens?

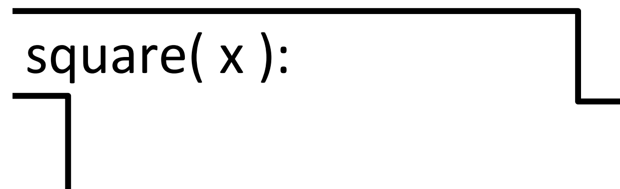
Function created

Name bound

Call expression:

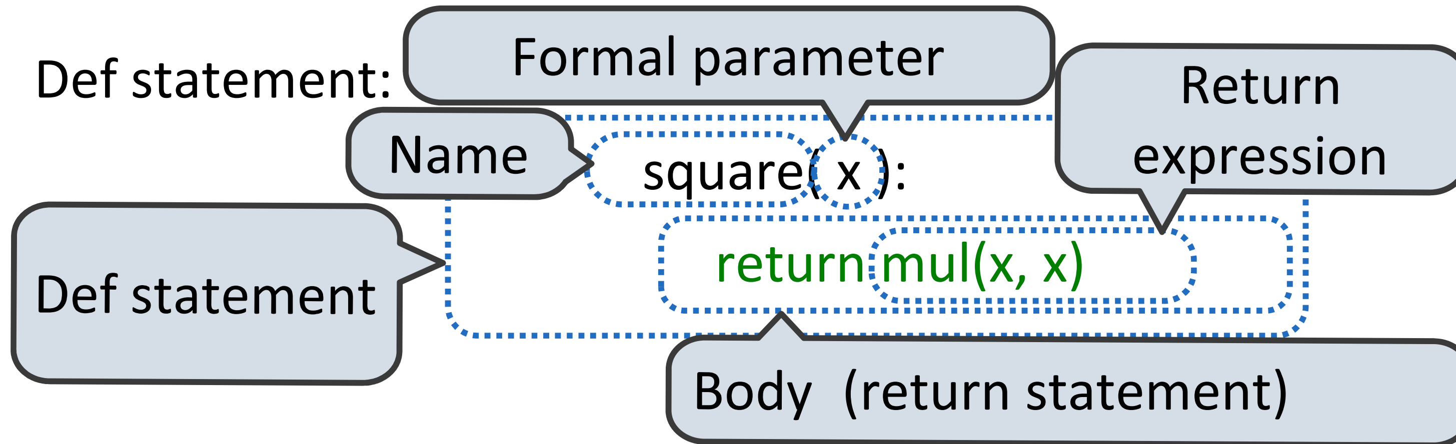


Calling/Applying:



Life Cycle of a User-Defined Function

Def statement:

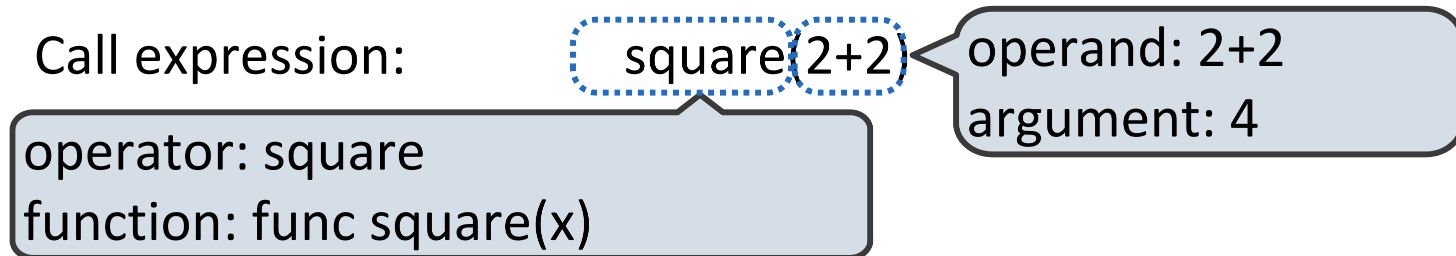


What happens?

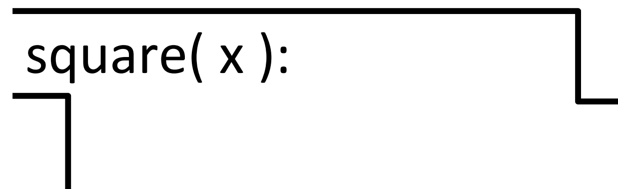
Function created

Name bound

Call expression:

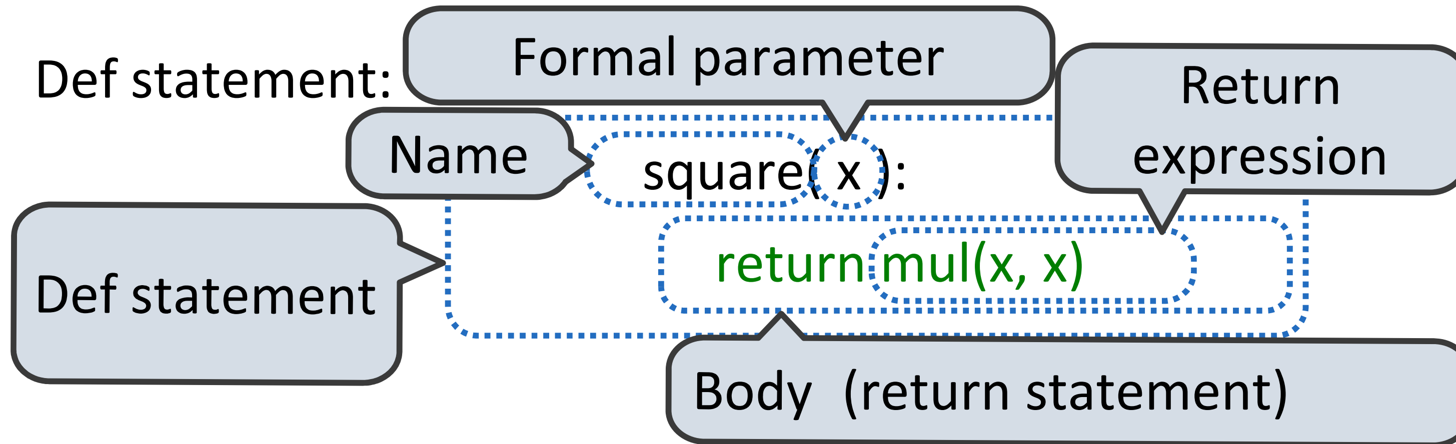


Calling/Applying:



Life Cycle of a User-Defined Function

Def statement:

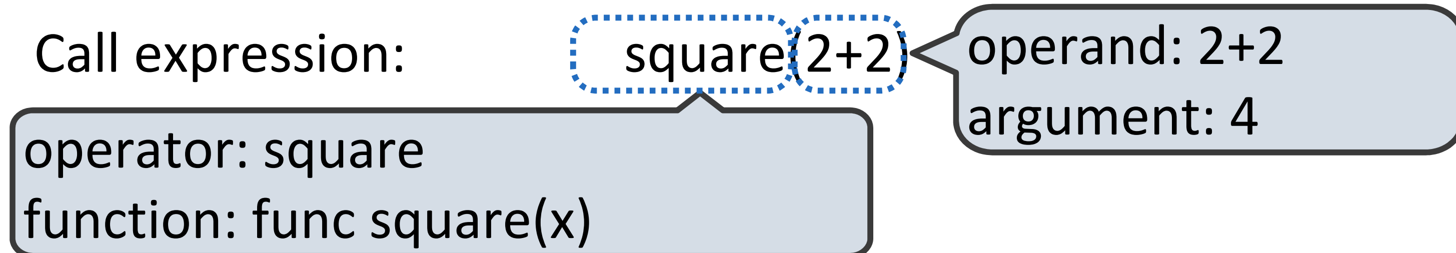


What happens?

Function created

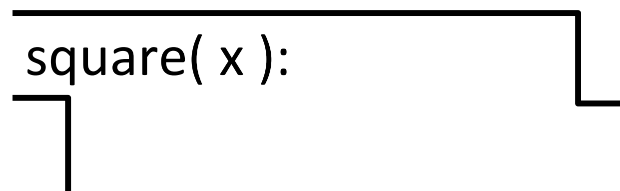
Name bound

Call expression:



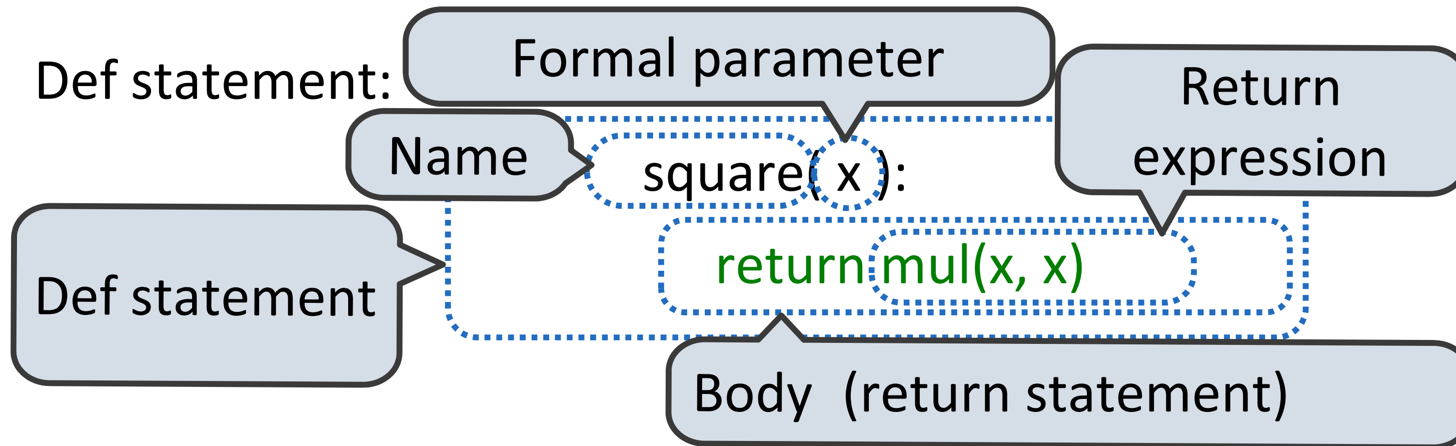
Op's evaluated

Calling/Applying:



Life Cycle of a User-Defined Function

Def statement:

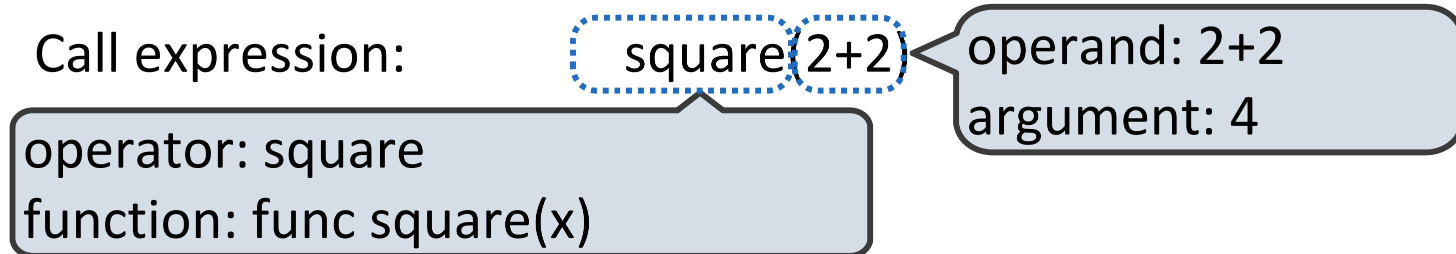


What happens?

Function created

Name bound

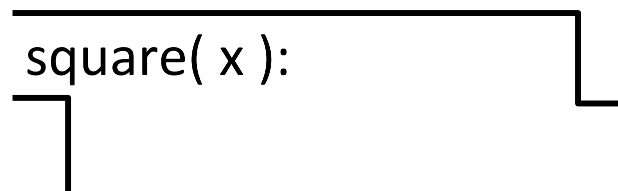
Call expression:



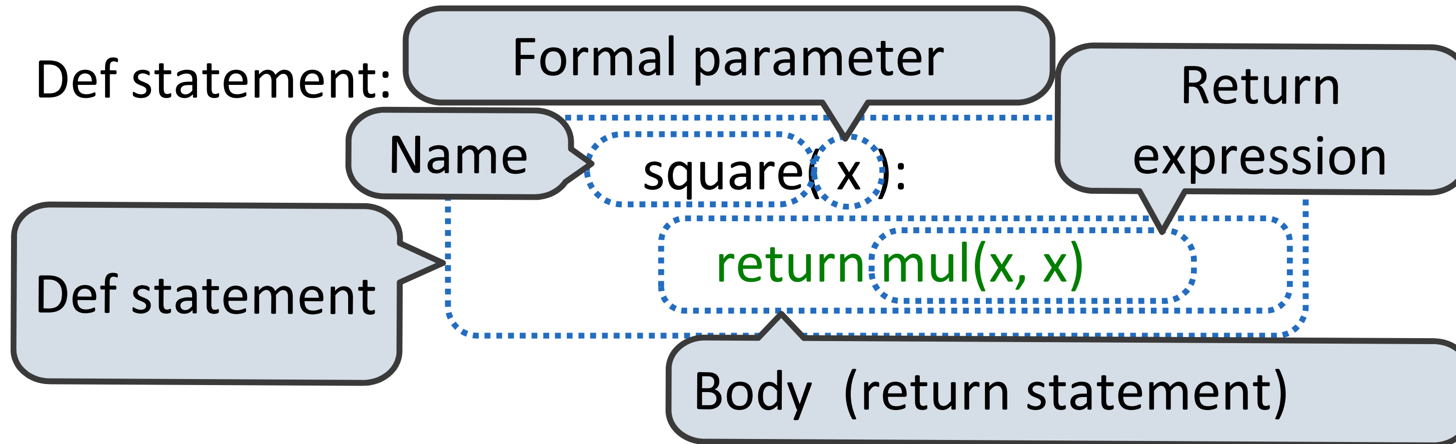
Op's evaluated

Function called with argument(s)

Calling/Applying:



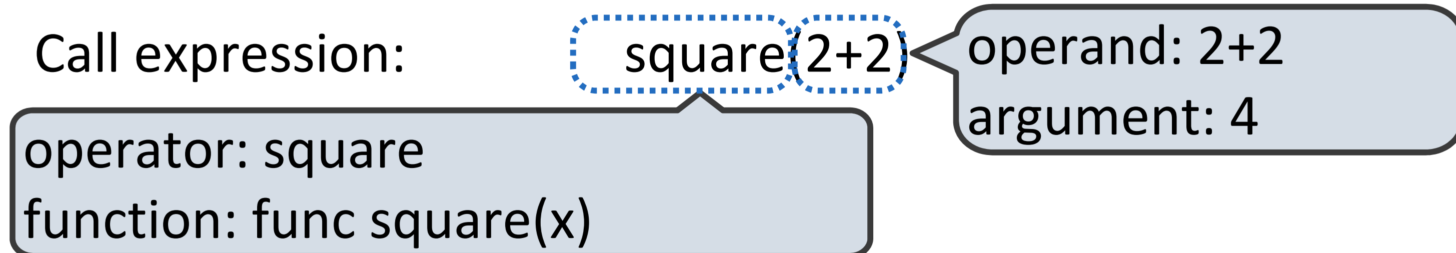
Life Cycle of a User-Defined Function



What happens?

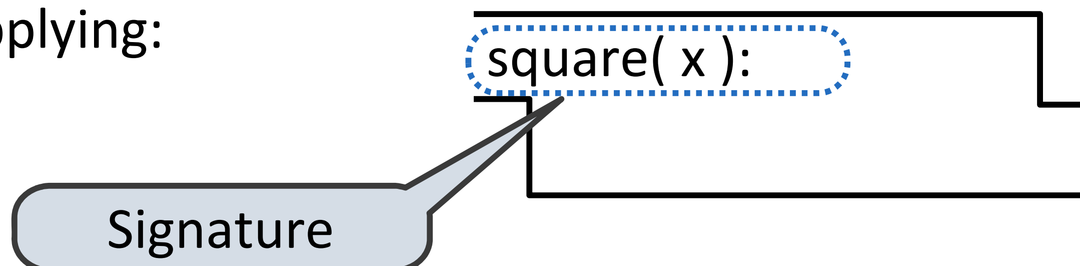
Function created

Name bound

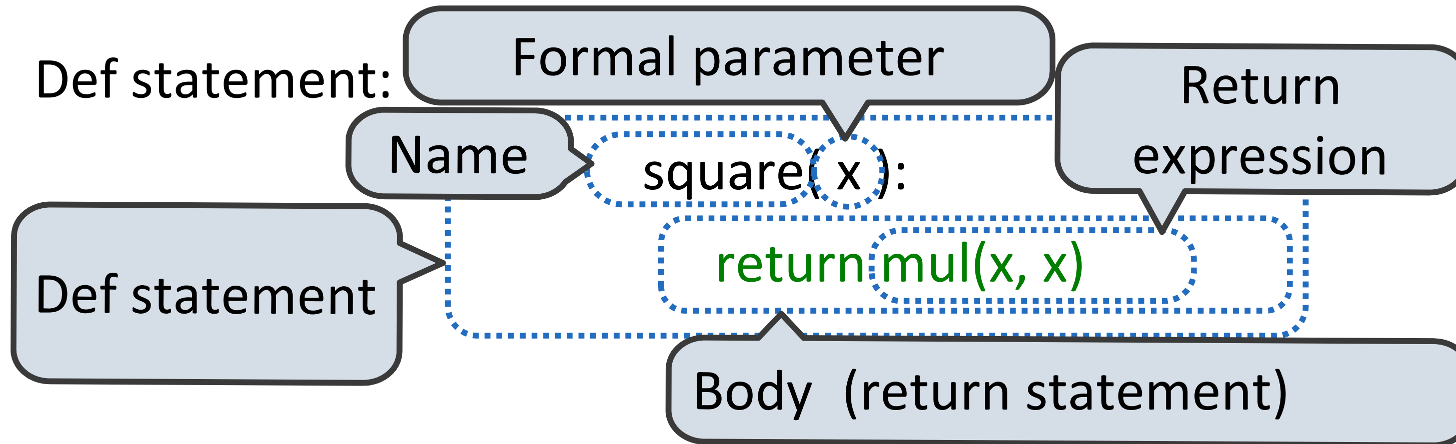


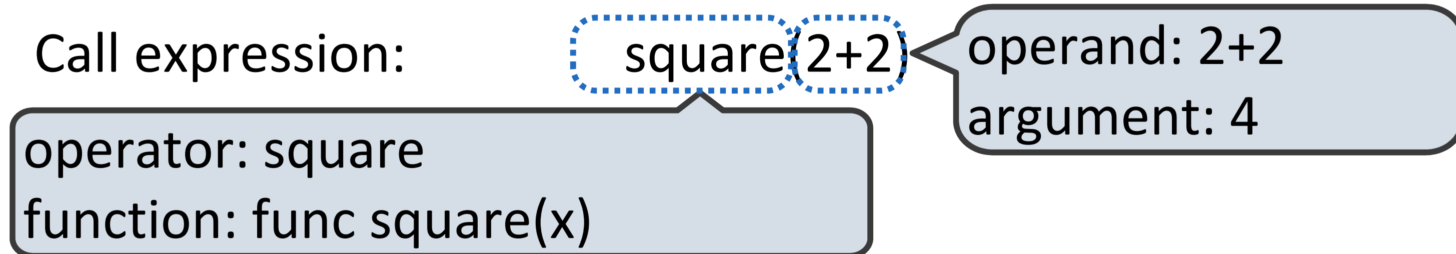
Op's evaluated

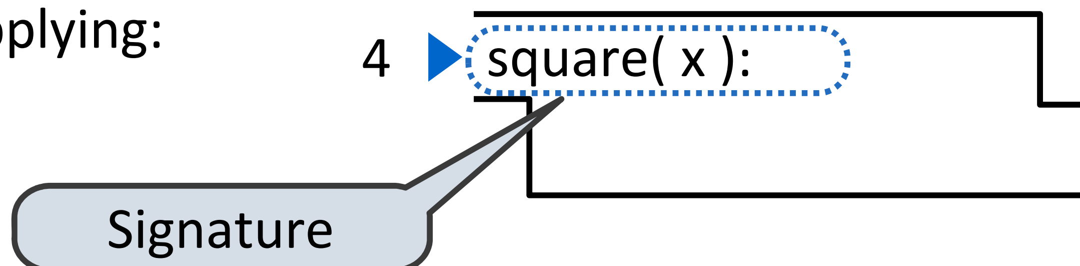
Function called with argument(s)



Life Cycle of a User-Defined Function







What happens?

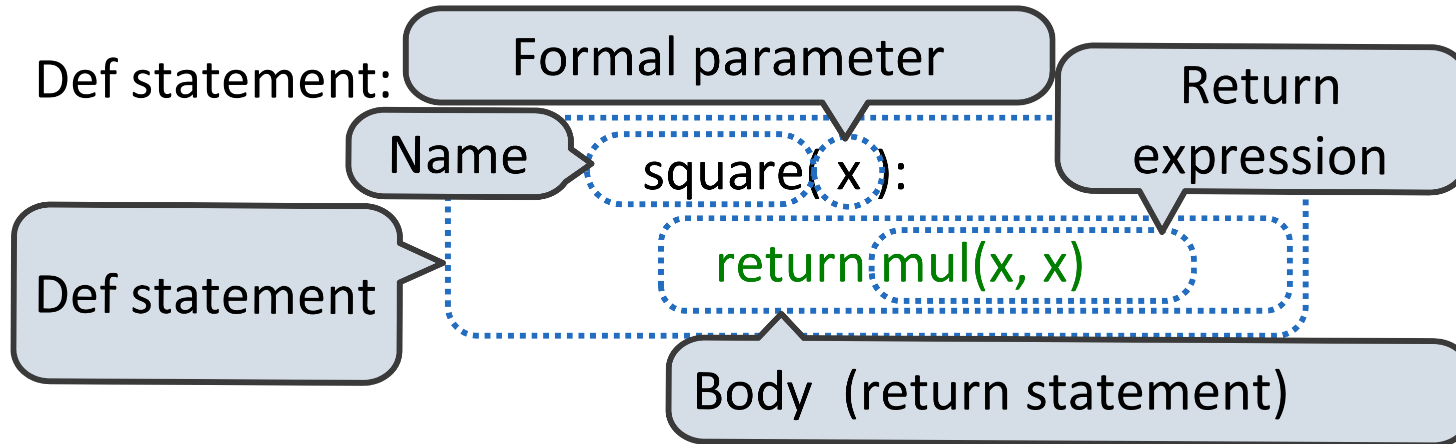
Function created

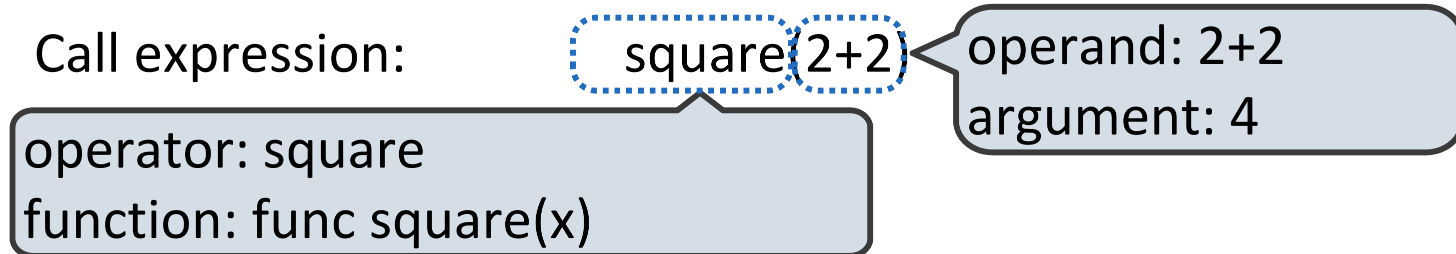
Name bound

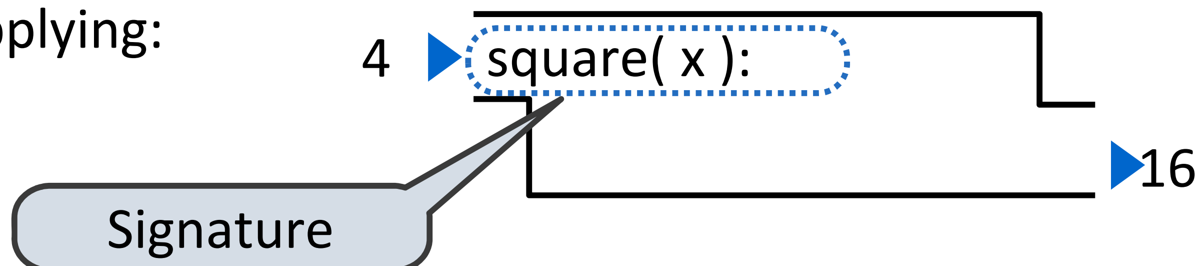
Op's evaluated

Function called with argument(s)

Life Cycle of a User-Defined Function







What happens?

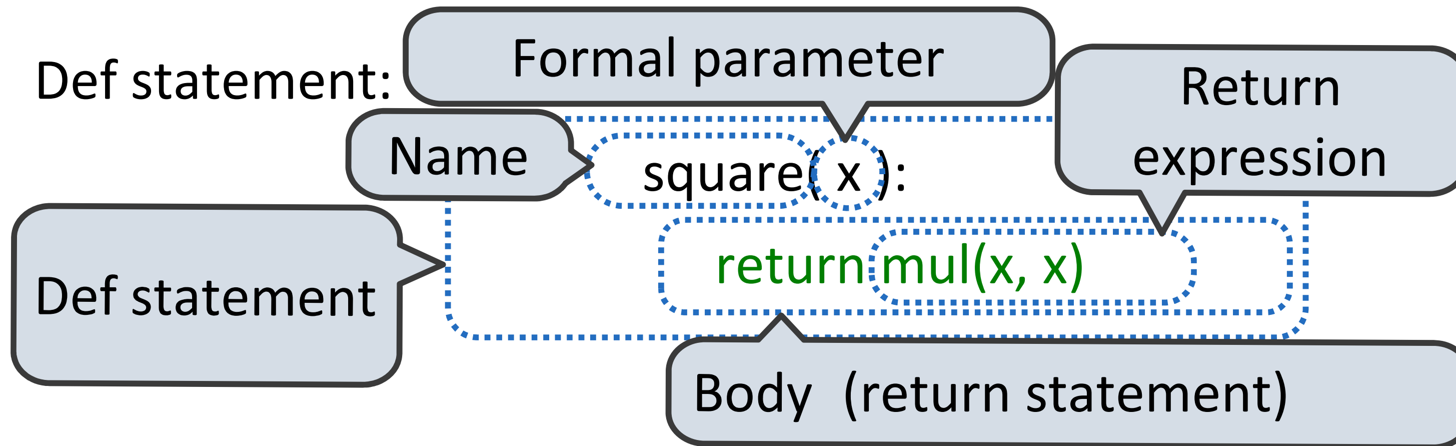
Function created

Name bound

Op's evaluated

Function called with argument(s)

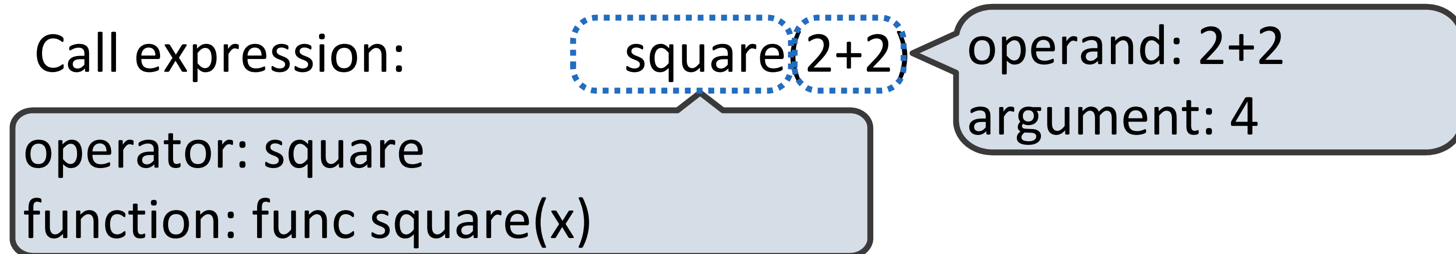
Life Cycle of a User-Defined Function



What happens?

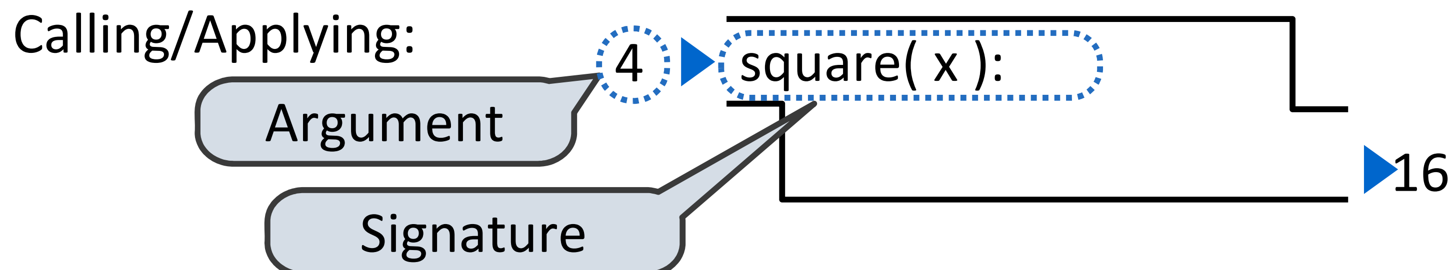
Function created

Name bound

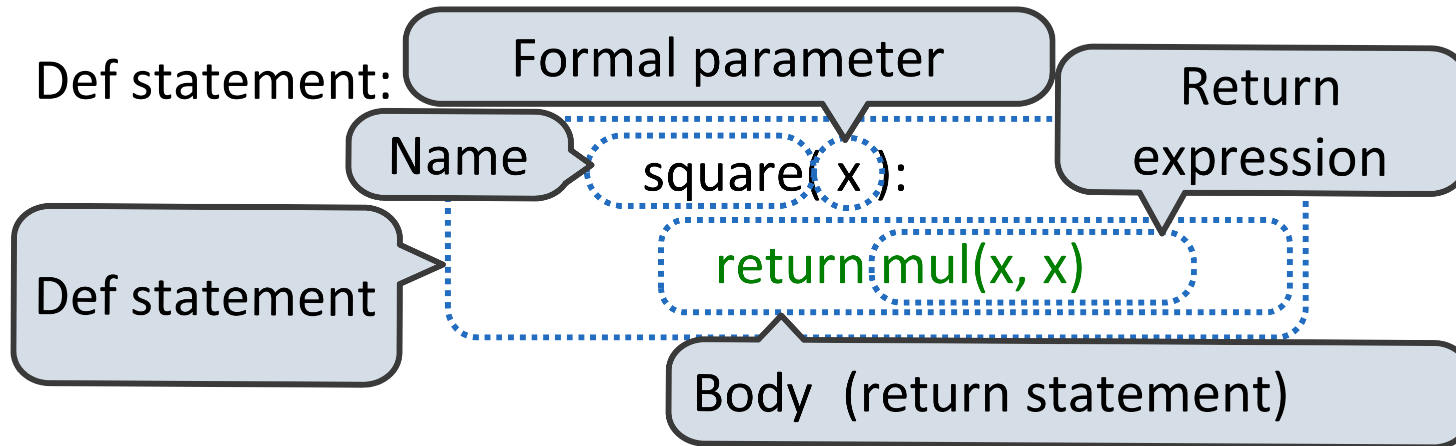


Op's evaluated

Function called with argument(s)



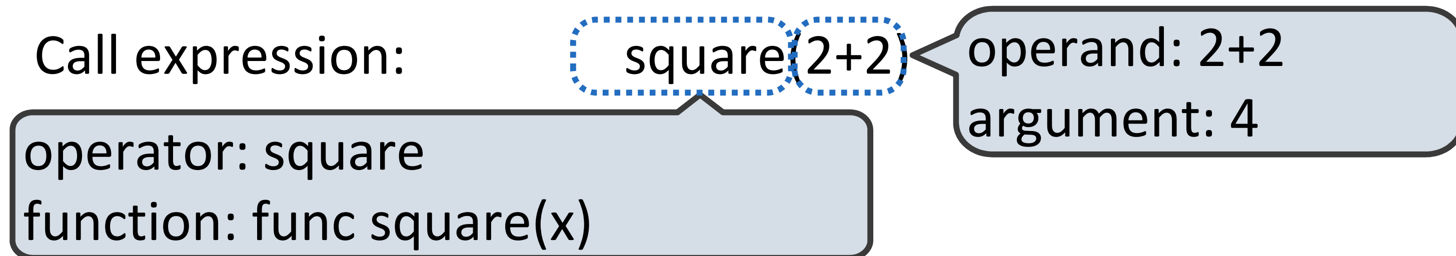
Life Cycle of a User-Defined Function



What happens?

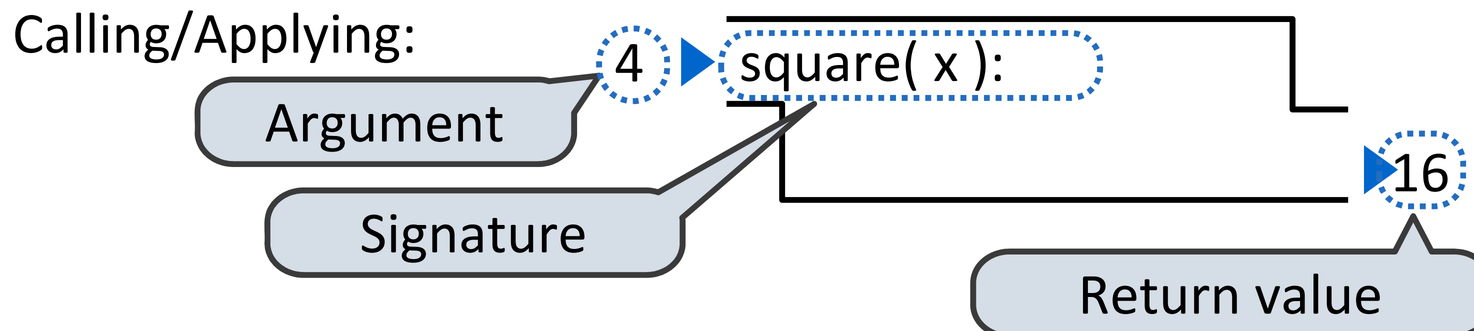
Function created

Name bound



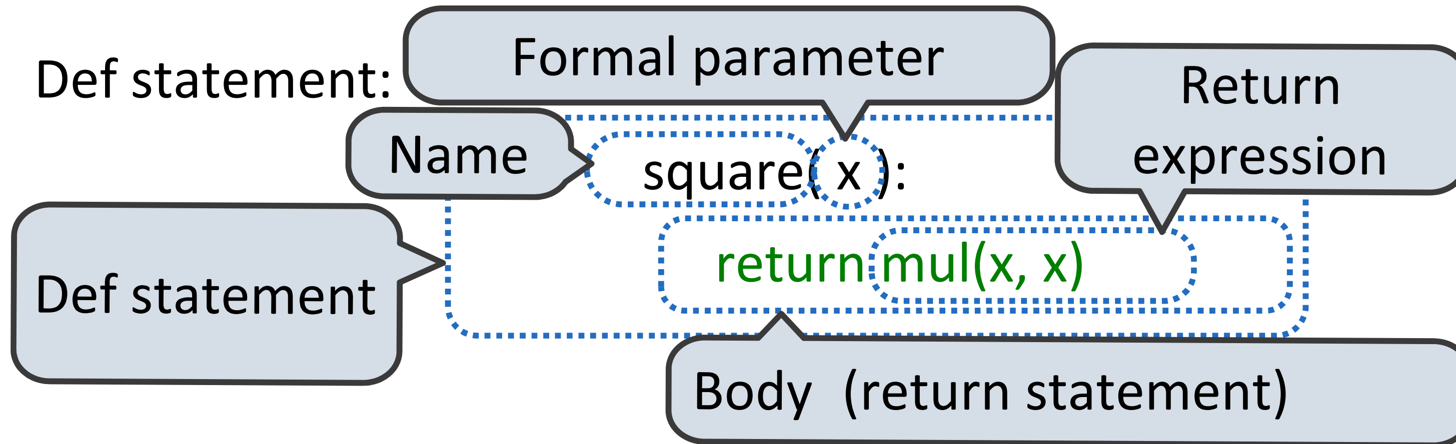
Op's evaluated

Function called with argument(s)

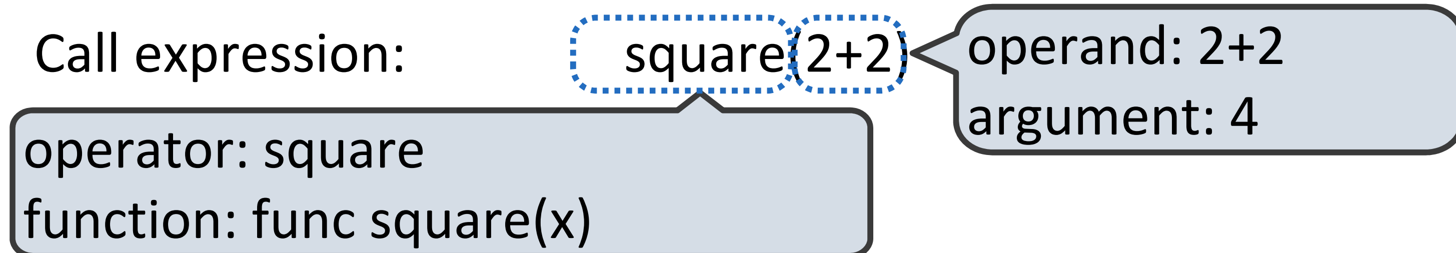


Life Cycle of a User-Defined Function

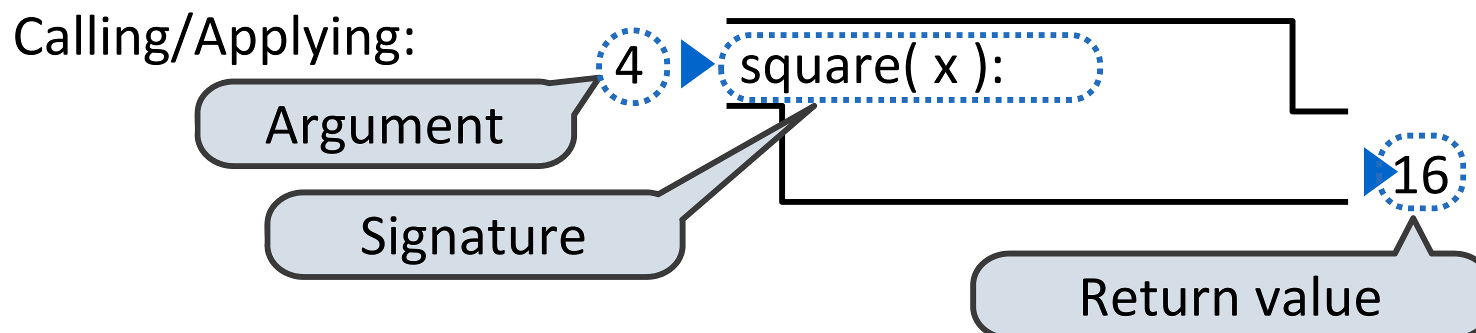
Def statement:



Call expression:



Calling/Applying:



What happens?

Function created

Name bound

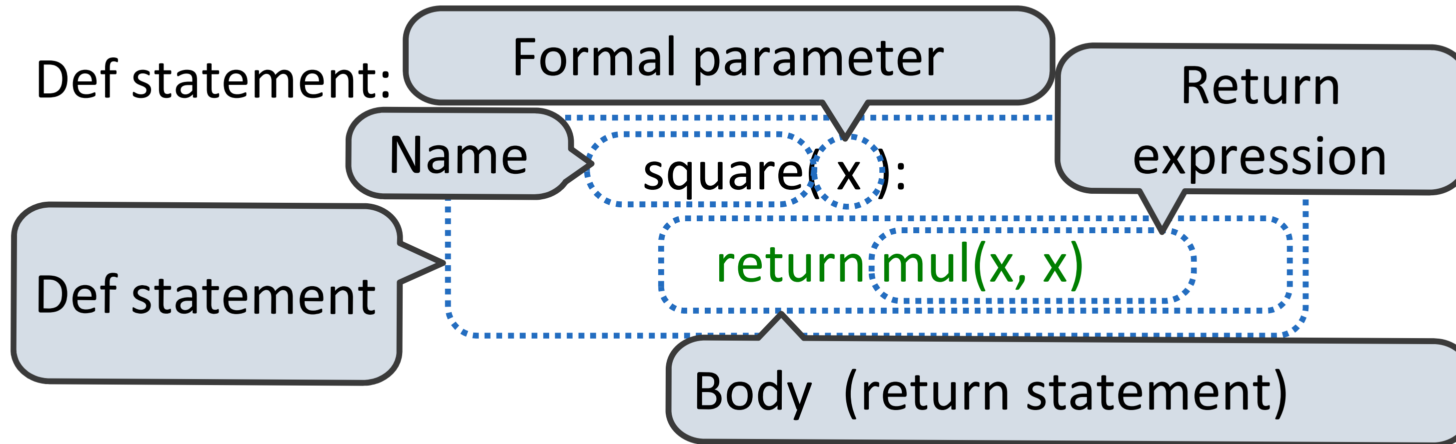
Op's evaluated

Function called with argument(s)

New frame!

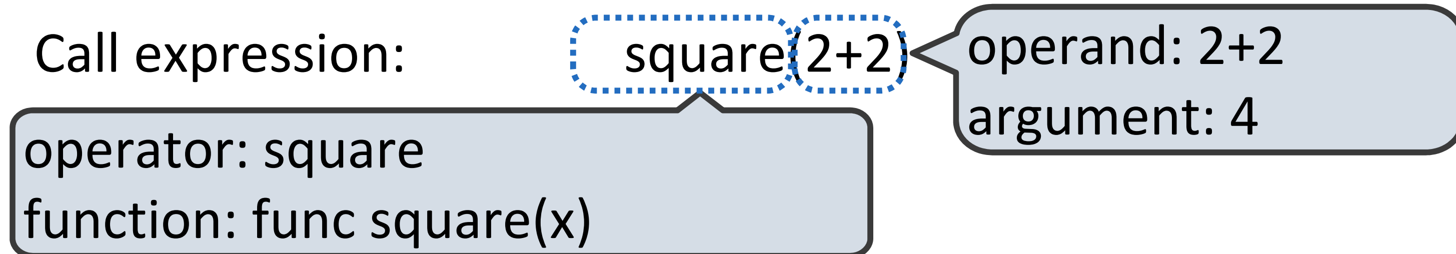
Life Cycle of a User-Defined Function

What happens?



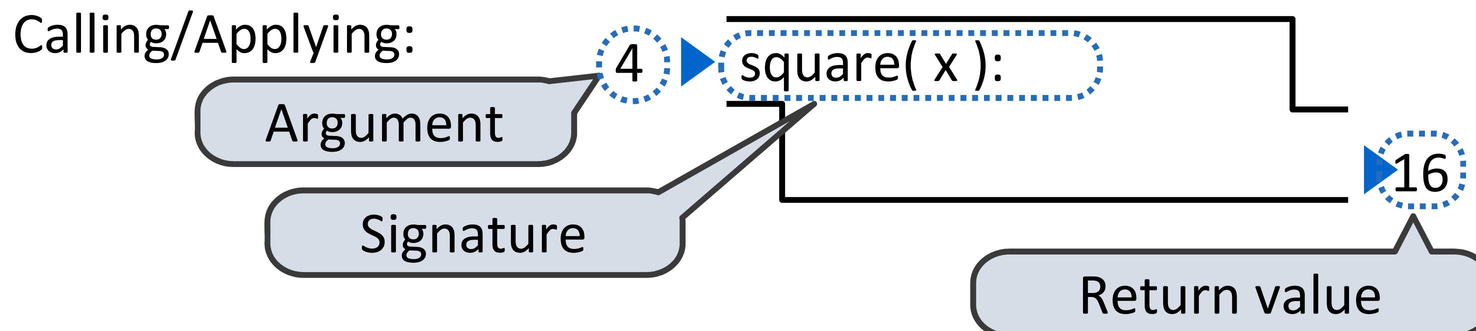
Function created

Name bound



Op's evaluated

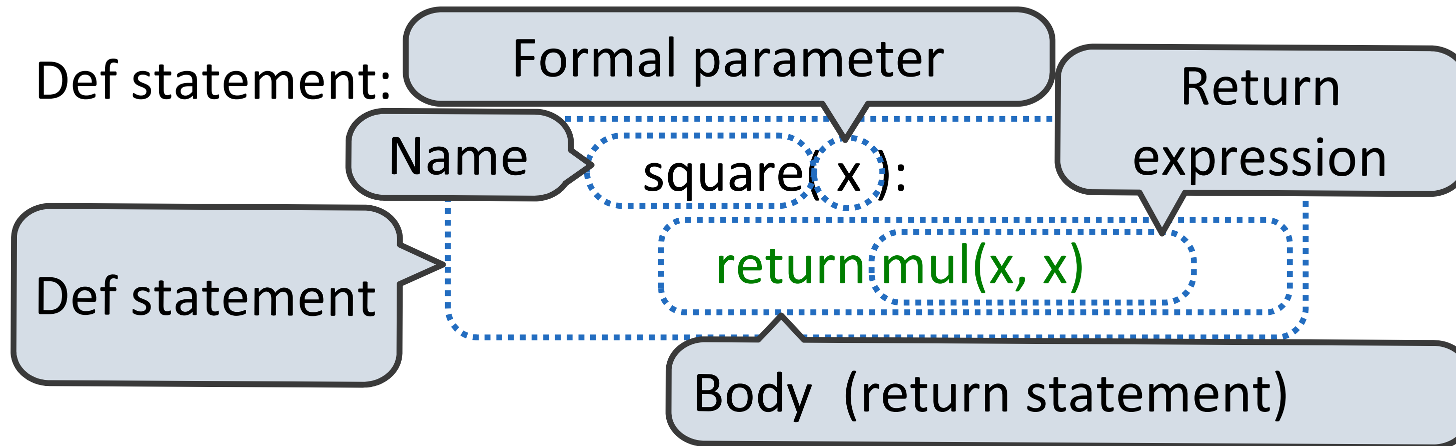
Function called with argument(s)

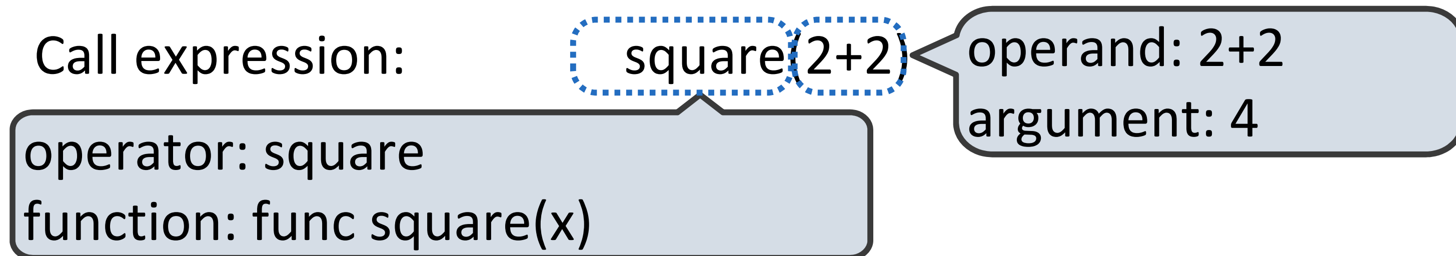


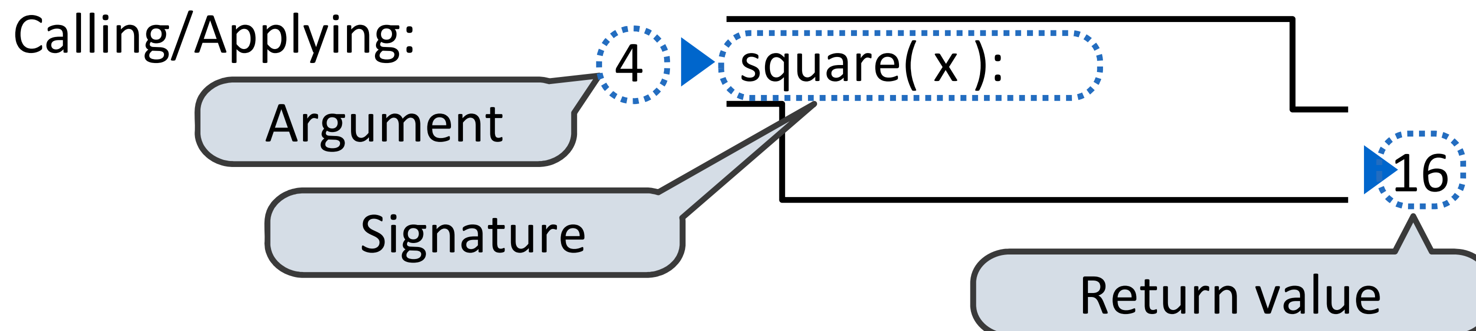
New frame!

Params bound

Life Cycle of a User-Defined Function







What happens?

Function created

Name bound

Op's evaluated

Function called with argument(s)

New frame!

Params bound

Body executed