Lecture 5: Environments

CS 61A - Summer 2024
Charlotte Le

Announcements

Contents

Review

Higher-Order Functions

A function that:

- **takes a function as an argument value or**
- **returns a function as a return value**

```
def composer(func1, func2):
    """Return a function f, such that f(x) = func1(func2(x))."""
    def f(x):
        return func1(func2(x))
    return f
```

Lambda Expressions

- Does not bind to a name
- Body is not evaluated until lambda is called
- Can be used as an operator or an operand

```
def multiply(x, y):
    return x * y
>>> multiply(2, 3)
6
```

```
multiply = lambda x, y : x * y

>>> multiply(2, 3)
6
>>> (lambda x, y : x * y)(2, 3)
6
```

Lambda Expressions

- Does not bind to a name
- Body is not evaluated until lambda is called
- Can be used as an operator or an operand

```
def multiply(x, y):
    return x * y
>>> multiply(2, 3)
6
```

```
multiply = lambda x, y : x * y

>>> multiply(2, 3)
6
>>> (lambda x, y : x * y)(2, 3)
6
```

```
negate = lambda f, x : -f(x)
negate(lambda x : x * x, 3)
```

Lambda Expressions

- Does not bind to a name
- Body is not evaluated until lambda is called
- Can be used as an operator or an operand

```
def multiply(x, y):
    return x * y
>>> multiply(2, 3)
6
```

```
multiply = lambda x, y : x * y

>>> multiply(2, 3)
6
>>> (lambda x, y : x * y)(2, 3)
6
```

```
negate = lambda f, x : -f(x)
negate(lambda x : x * x, 3)
```

Lambda Expressions

- **Does not bind to a name**
- **Body is not evaluated until lambda is called**
- **Can be used as an operator or an operand**

```
def multiply(x, y):
    return x * y
>>> multiply(2, 3)
6
```

```
multiply = lambda x, y : x * y

>>> multiply(2, 3)
6
>>> (lambda x, y : x * y)(2, 3)
6
```

- **negate** has two parameters **f** and **x** and returns **-f(x)**

```
negate = lambda f, x : -f(x)
negate(lambda x : x * x, 3)
```

Lambda Expressions

- Does not bind to a name
- Body is not evaluated until lambda is called
- Can be used as an operator or an operand

```
def multiply(x, y):
    return x * y
>>> multiply(2, 3)
6
```

```
multiply = lambda x, y : x * y

>>> multiply(2, 3)
6
>>> (lambda x, y : x * y)(2, 3)
6
```

```
negate = lambda f, x : -f(x)
negate(lambda x : x * x, 3)
```

- **negate** has two parameters **f** and **x** and returns **-f(x)**
- **f** → **lambda x : x * x**
- **x** → **3**

Lambda Expressions

- Does not bind to a name
- Body is not evaluated until lambda is called
- Can be used as an operator or an operand

```
def multiply(x, y):
    return x * y
>>> multiply(2, 3)
6
```

```
multiply = lambda x, y : x * y

>>> multiply(2, 3)
6
>>> (lambda x, y : x * y)(2, 3)
6
```

```
negate = lambda f, x : -f(x)
negate(lambda x : x * x, 3)
```

- **negate** has two parameters **f** and **x** and returns **-f(x)**
- **f** → **lambda x : x * x**
- **x** → **3**
- **negate** returns:
  - **- f(x)** → **- (lambda x : x * x)(3)** → **- 9**

Currying

Converts a function that takes multiple arguments into a chain of functions that each take a single argument e.g., f(x, y) → f(x)(y)

```
def my_add(x, y):
    return x + y



>>> my_add(10, 5)
15
>>> my_add(10, 72)
82
>>> my_add(10, 16)
26
```

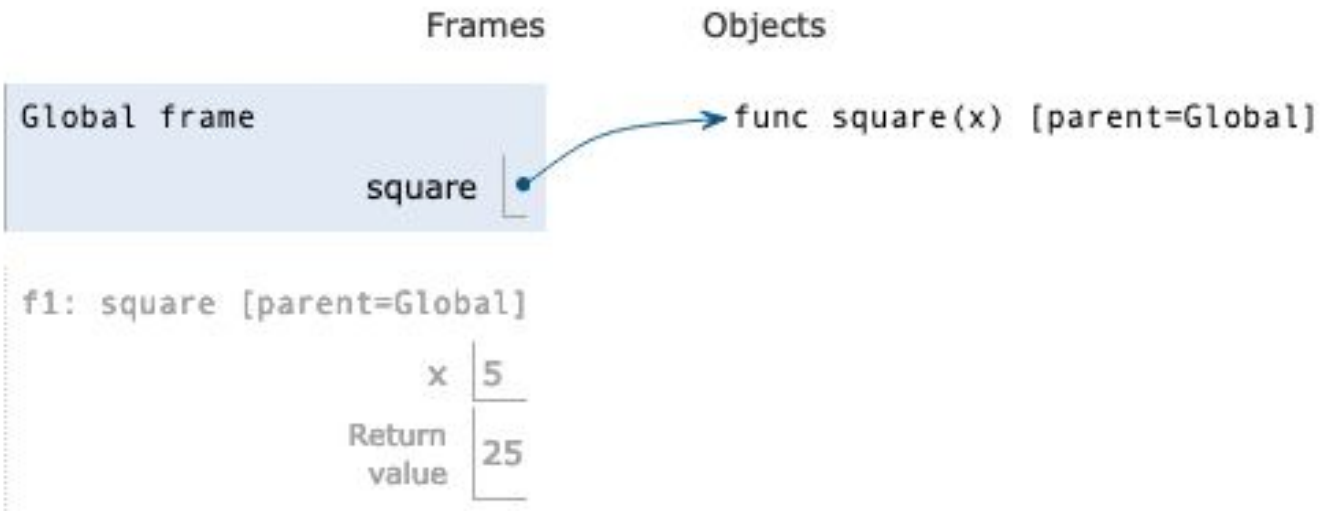
```
def outer(x):
    def inner(y):
        return x + y
    return inner


>>> add_ten = outer(10)
>>> add_ten(5)
15
>>> add_ten(72)
82
>>> add_ten(16)
26
```

Environment Diagrams: Overview

What Are Environment Diagrams?

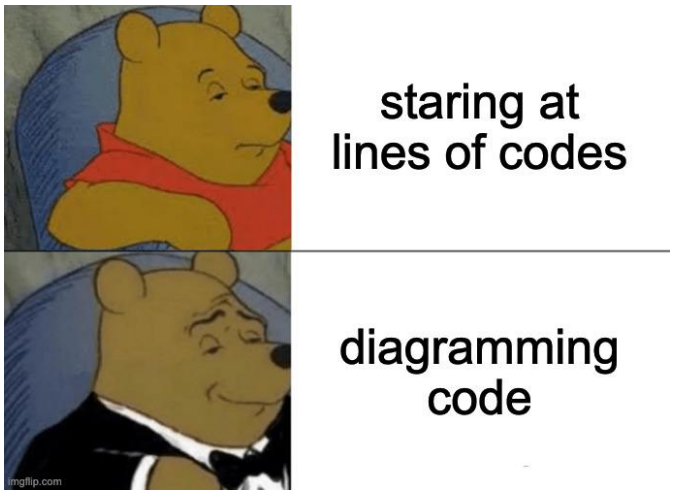
- **A visual tool to keep track of bindings & state of a computer program**
- **pythontutor.com**



Why Do We Use Environment Diagrams?

- **Conceptual**
  - **Understand *why* programs work the way they do**
  - **Confidently predict how a program will behave**

- **Helpful for debugging**
  - **When you're really stuck:**



Definitions

- A frame keeps track of variable-to-value bindings
  - Every call expression has a corresponding frame
- The global frame is the starting frame
  - Does not correspond to a specific call expression
- Parent frames
  - If you can't find a variable in the current frame, you check its parent, and so on
  - If you can't find the variable: NameError

Assignment Statements

GUIDE

Step-by-Step: Assignment Statements in Environment Diagrams

(1) evaluate the expression on the right of the = sign to get a value/object

- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes

- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no

- bind the new name to the value/object

notes

- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings

Step-by-Step: Assignment Statements in Environment Diagrams Example 1

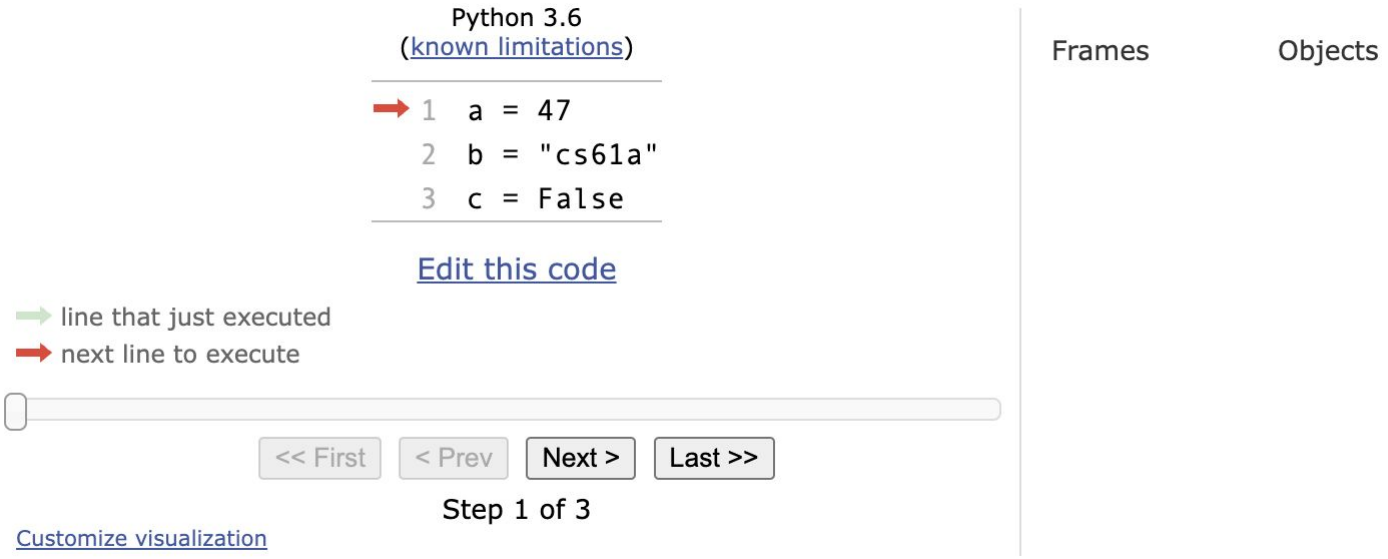
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 1

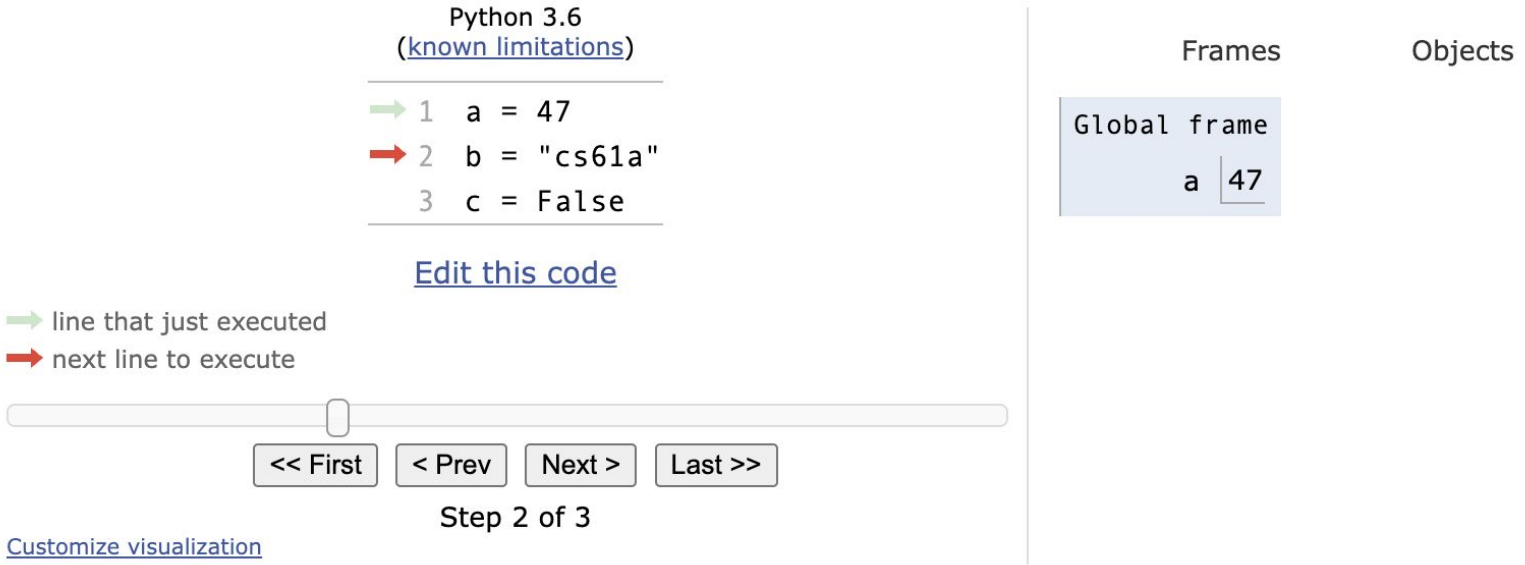
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 1

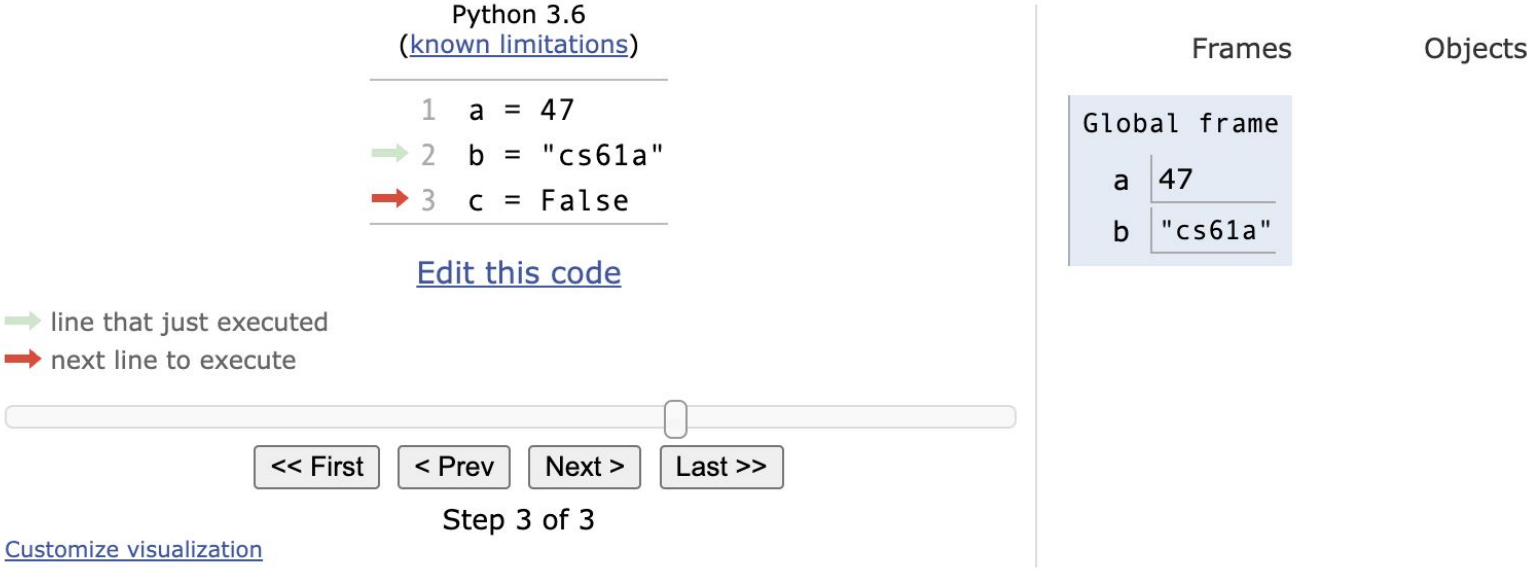
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 1

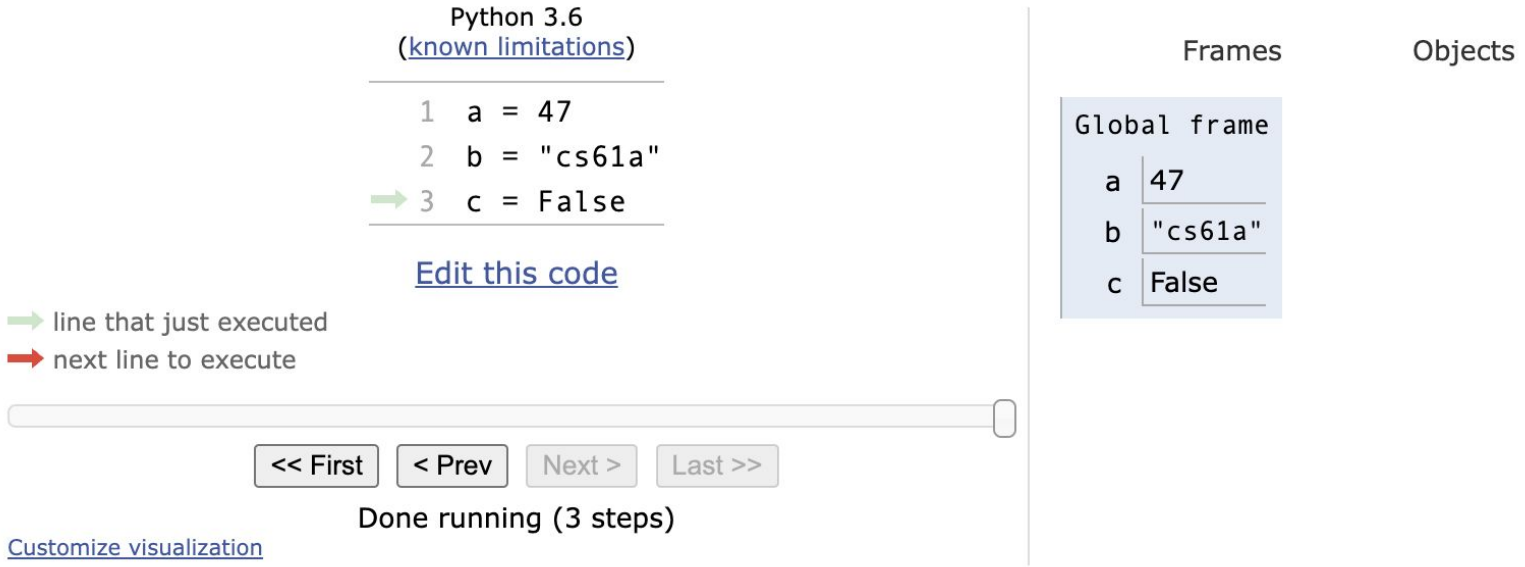
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 2

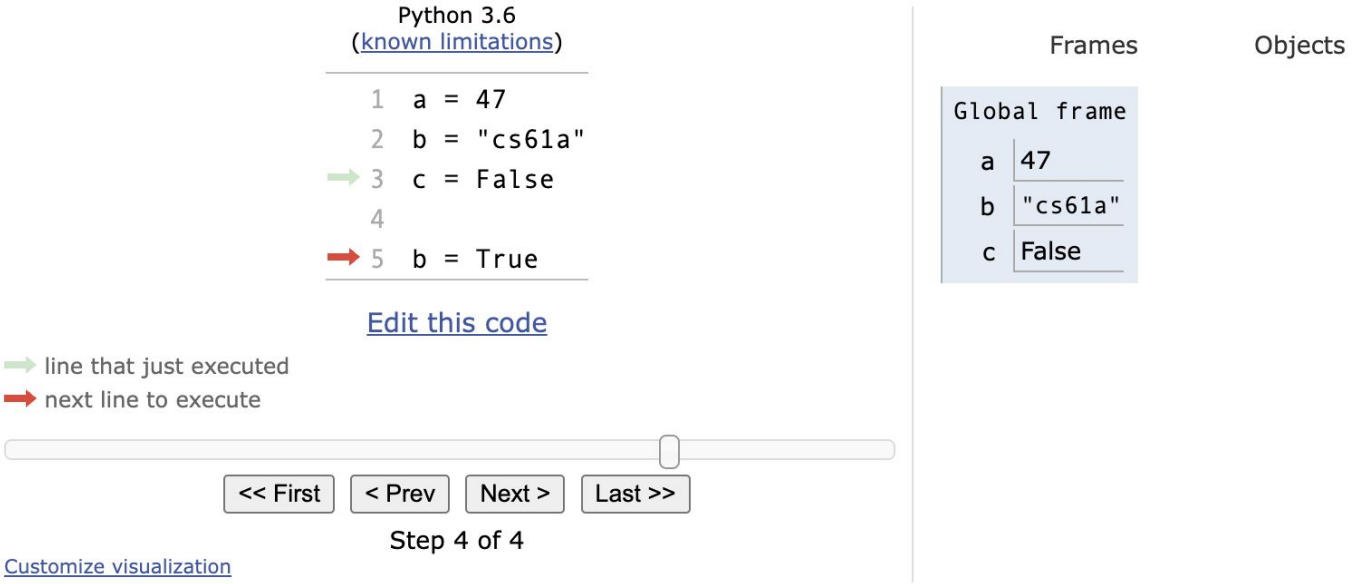
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 2

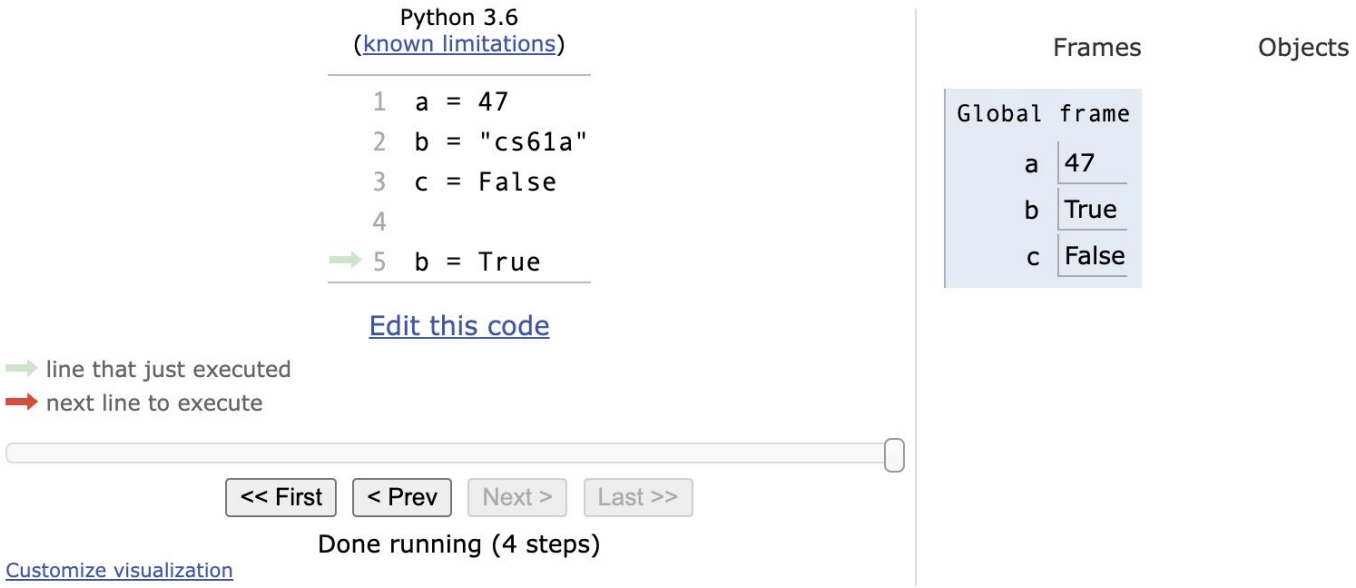
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 3

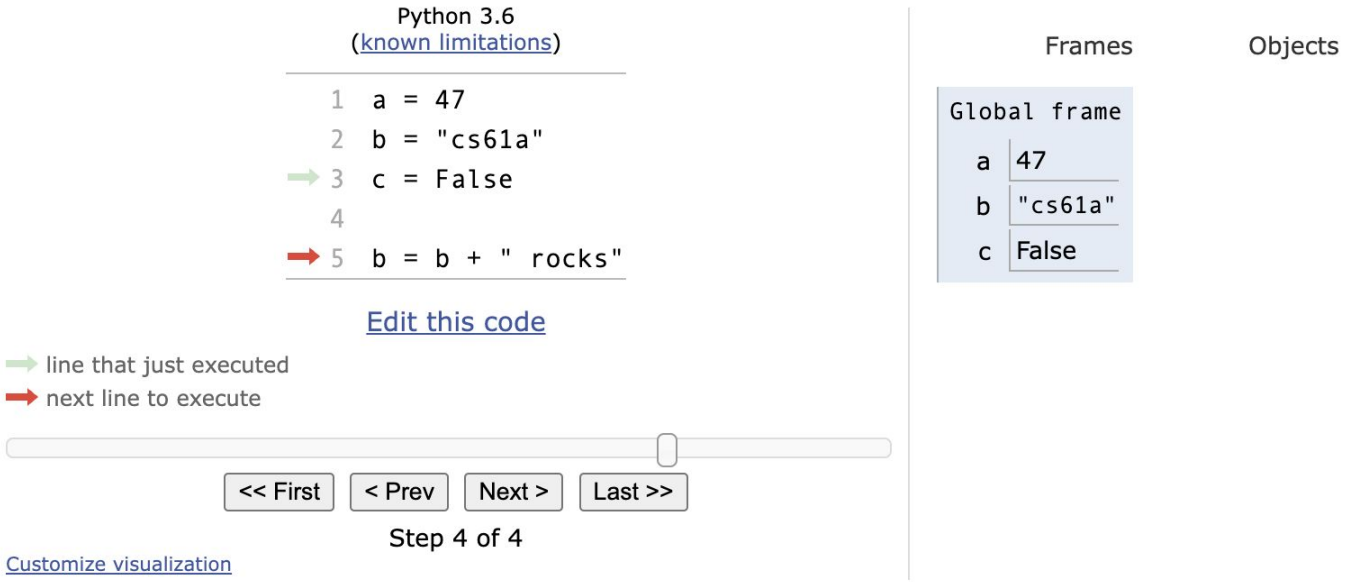
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 3

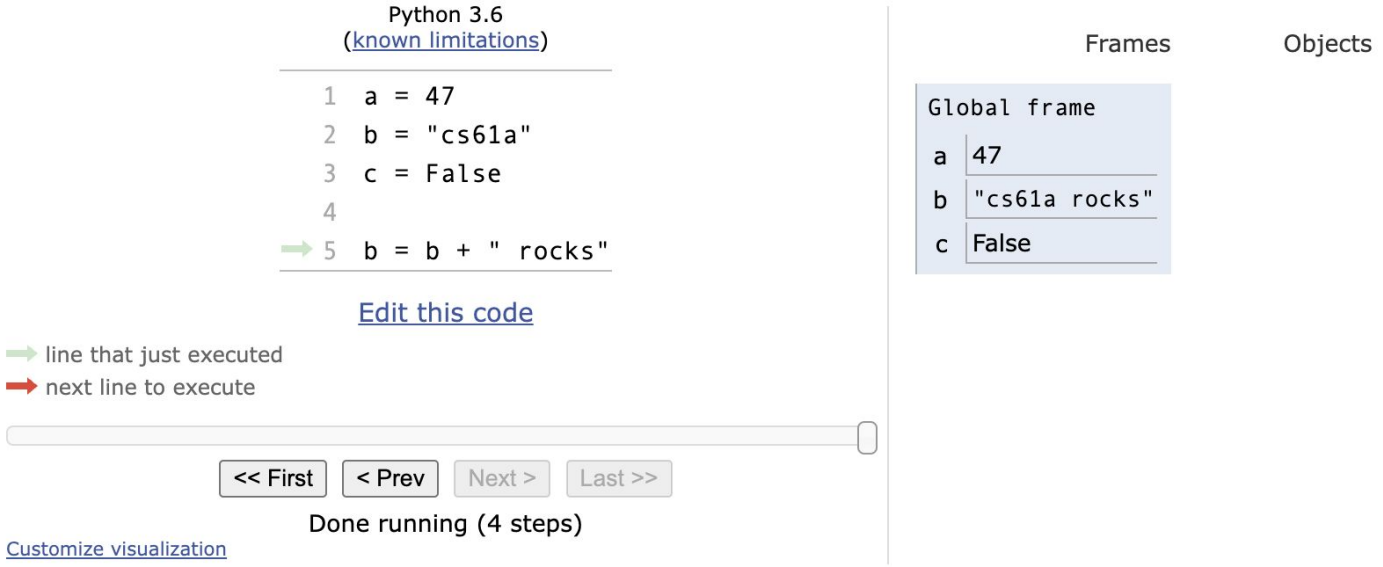
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams
Example 4

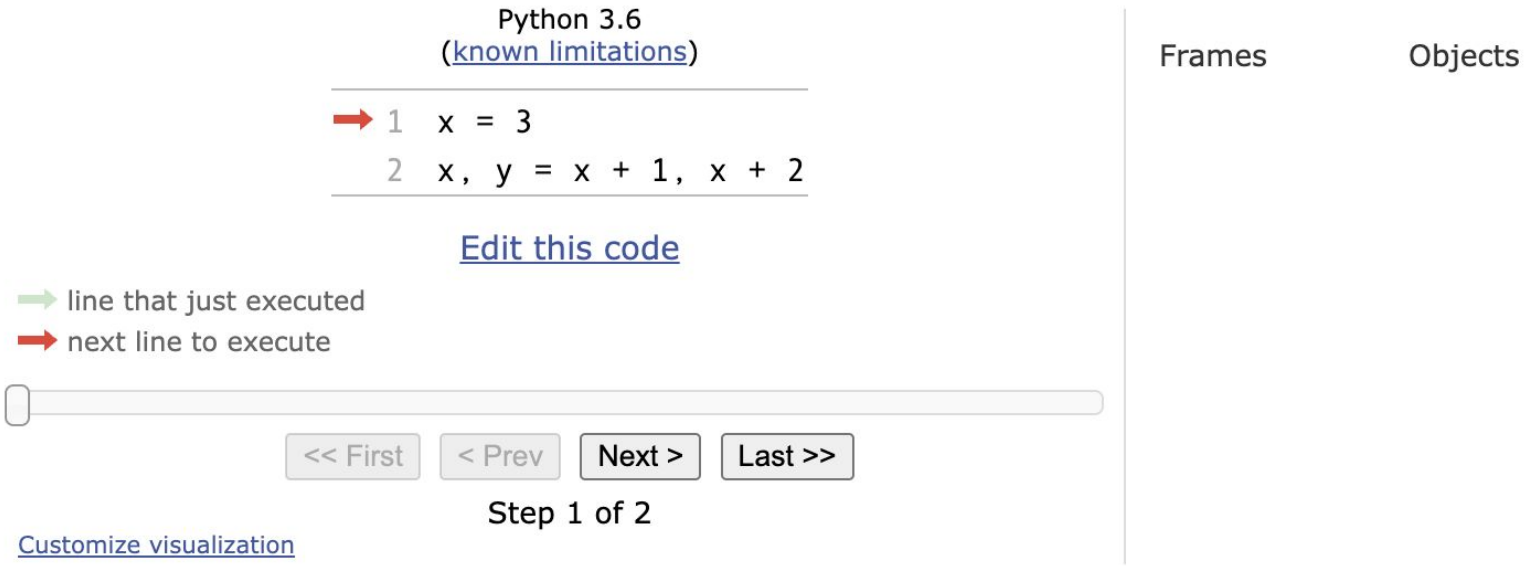
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 4

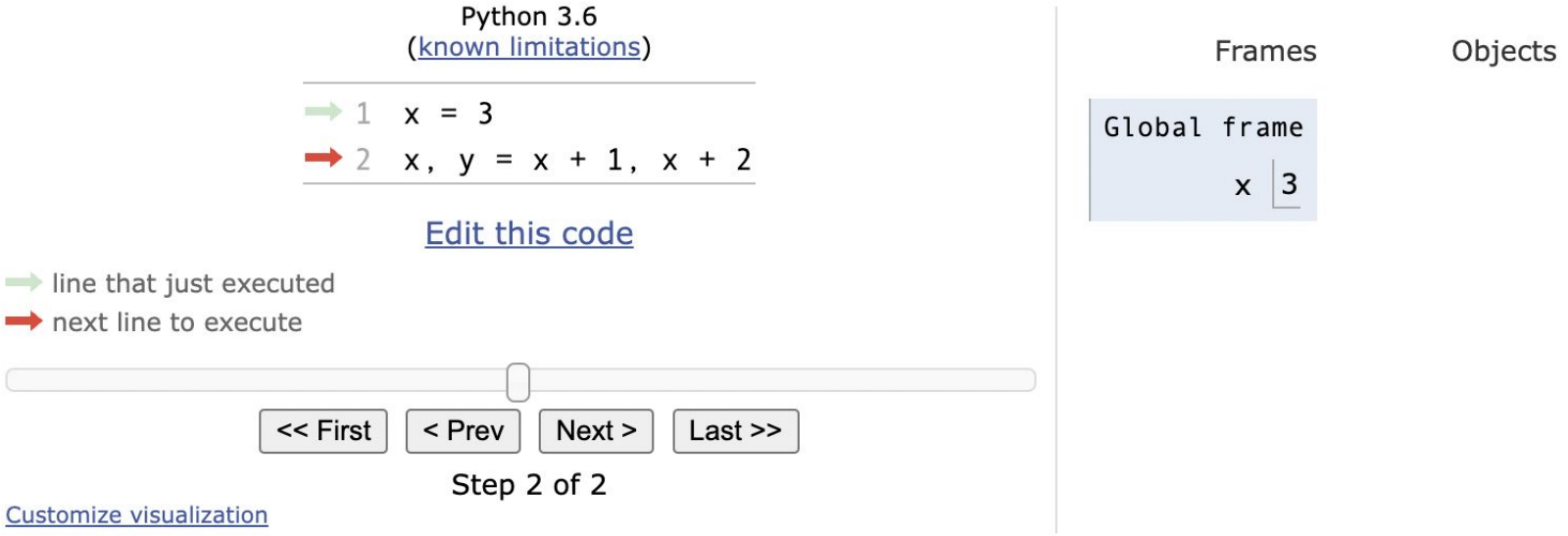
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



Step-by-Step: Assignment Statements in Environment Diagrams Example 4

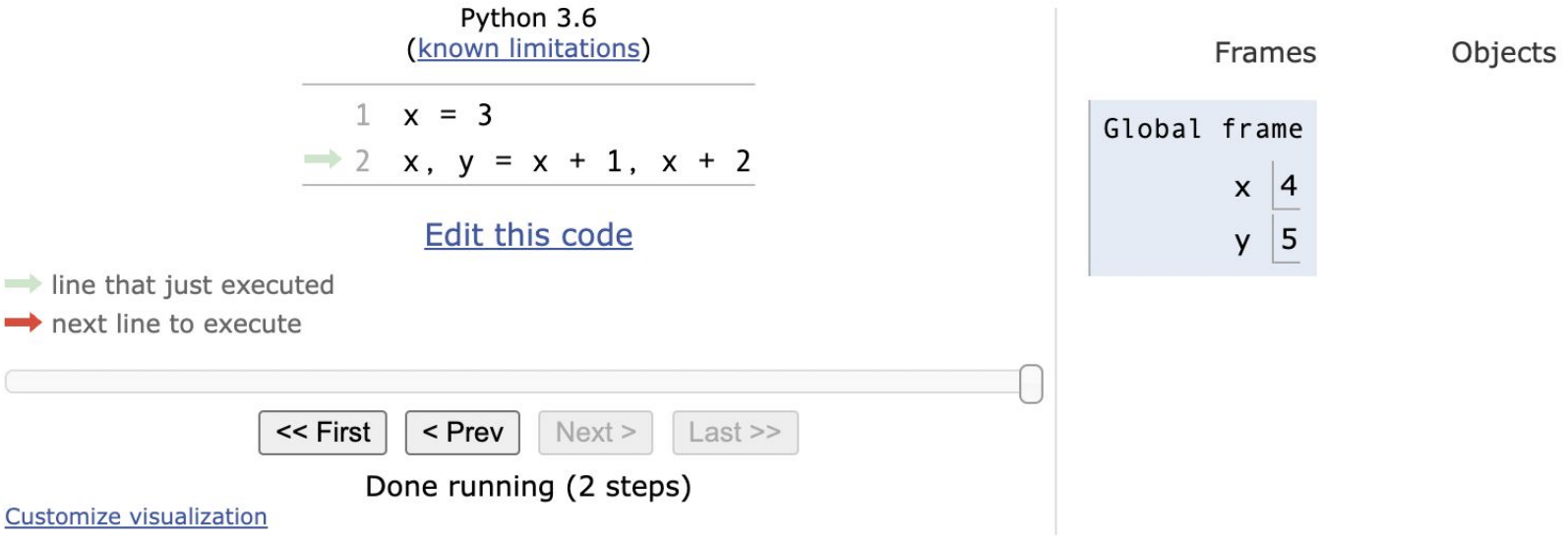
(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings



def Statements

GUIDE
Step-by-Step: def Statements in Environment Diagrams

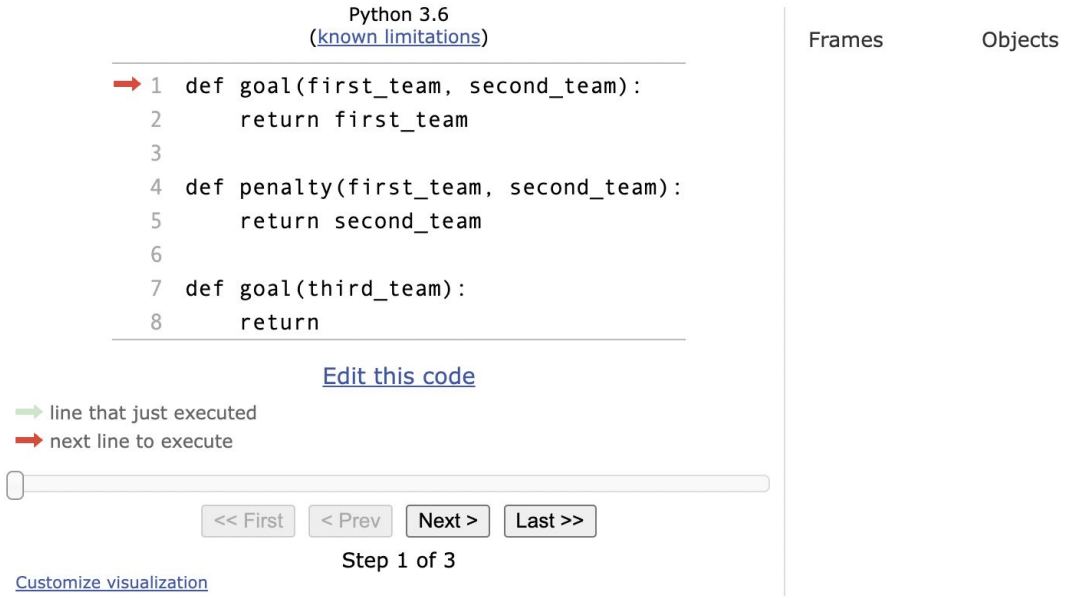
(1) draw the function object with: func & intrinsic name & formal parameters & parent frame
(2) does the intrinsic name of the function already exist in the current frame?
    ↳ yes
        - erase the current bindings
    ↳ no
        - write it in
(3) bind the newly created function object to this name

notes
- a function's parent frame is the frame in which the function was *defined*

Step-by-Step: def Statements in Environment Diagrams
Example 1

(1) draw the function object with: func & intrinsic name & formal parameters & parent frame
(2) does the intrinsic name of the function already exist in the current frame?
    ↳ yes
        - erase the current bindings
    ↳ no
        - write it in
(3) bind the newly created function object to this name

notes
- a function's parent frame is the frame in which the function was *defined*



Draw

Step-by-Step: def Statements in Environment Diagrams
Example 1

(1) draw the function object with: func & intrinsic name & formal parameters & parent frame
(2) does the intrinsic name of the function already exist in the current frame?
    ↳ yes
        - erase the current bindings
    ↳ no
        - write it in
(3) bind the newly created function object to this name

notes
- a function's parent frame is the frame in which the function was *defined*

Python 3.6
(known limitations)

```
1 def goal(first_team, second_team):
2     return first_team
3
4 def penalty(first_team, second_team):
5     return second_team
6
7 def goal(third_team):
8     return
```

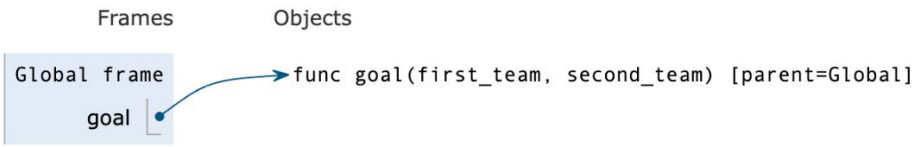
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 2 of 3

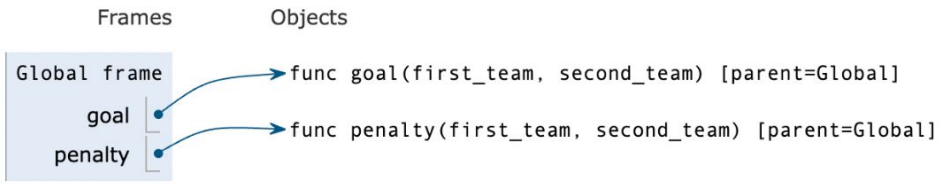
Customize visualization



Step-by-Step: def Statements in Environment Diagrams
Example 1

(1) draw the function object with: func & intrinsic name & formal parameters & parent frame
(2) does the intrinsic name of the function already exist in the current frame?
    ↳ yes
        - erase the current bindings
    ↳ no
        - write it in
(3) bind the newly created function object to this name

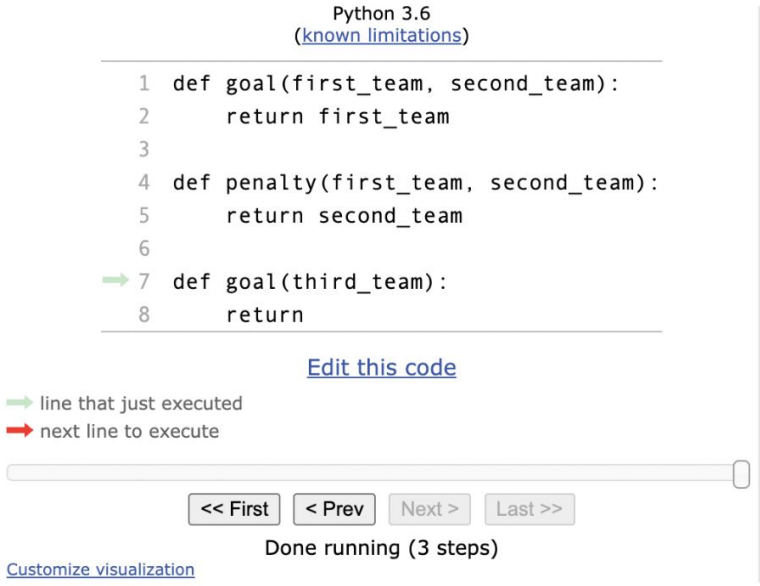
notes
- a function's parent frame is the frame in which the function was *defined*

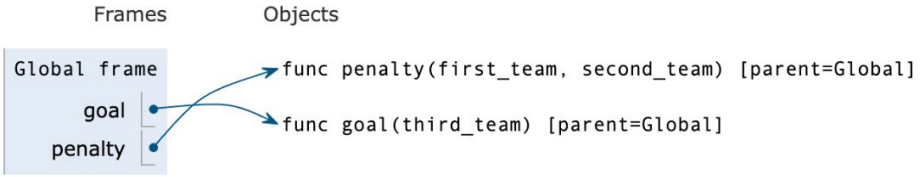


Step-by-Step: def Statements in Environment Diagrams
Example 1

(1) draw the function object with: func & intrinsic name & formal parameters & parent frame
(2) does the intrinsic name of the function already exist in the current frame?
    ↳ yes
        - erase the current bindings
    ↳ no
        - write it in
(3) bind the newly created function object to this name

notes
- a function's parent frame is the frame in which the function was *defined*





Call Expressions

GUIDE
Step-by-Step: Call Expressions in Environment Diagrams

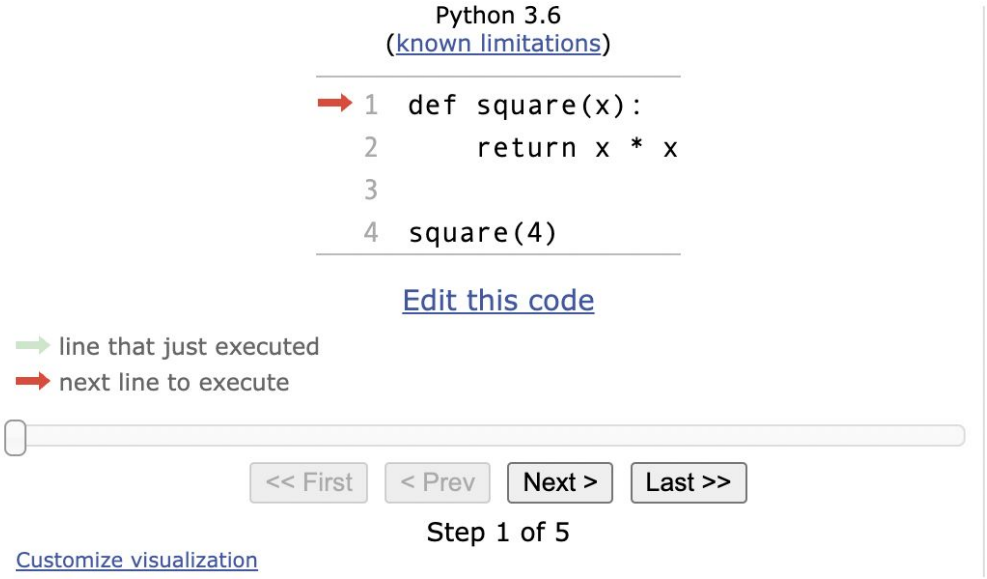
(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

Step-by-Step: Call Expressions in Environment Diagrams
Example 1

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

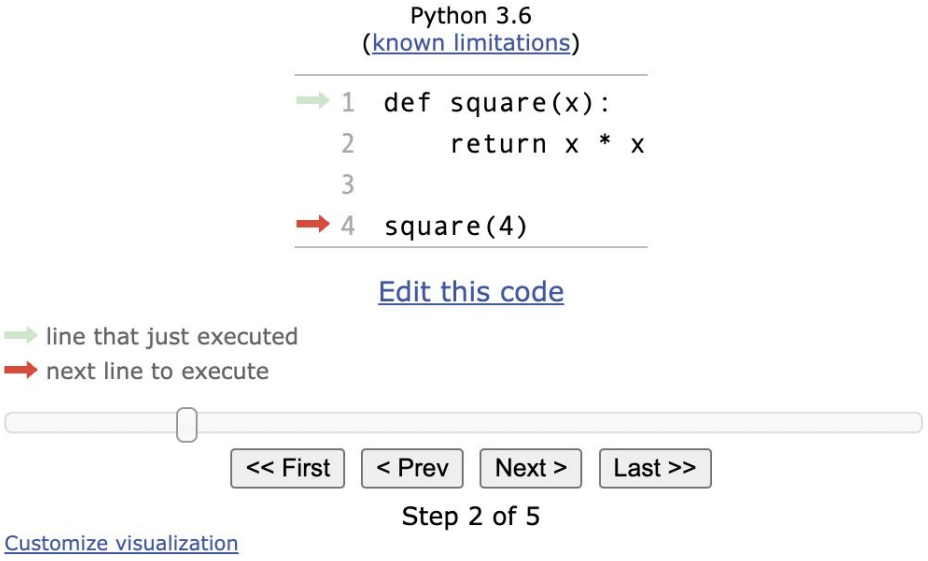


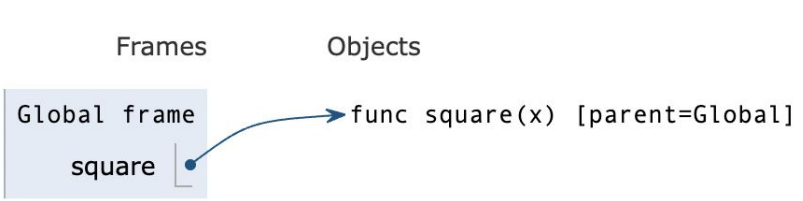
Draw

Step-by-Step: Call Expressions in Environment Diagrams Example 1

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

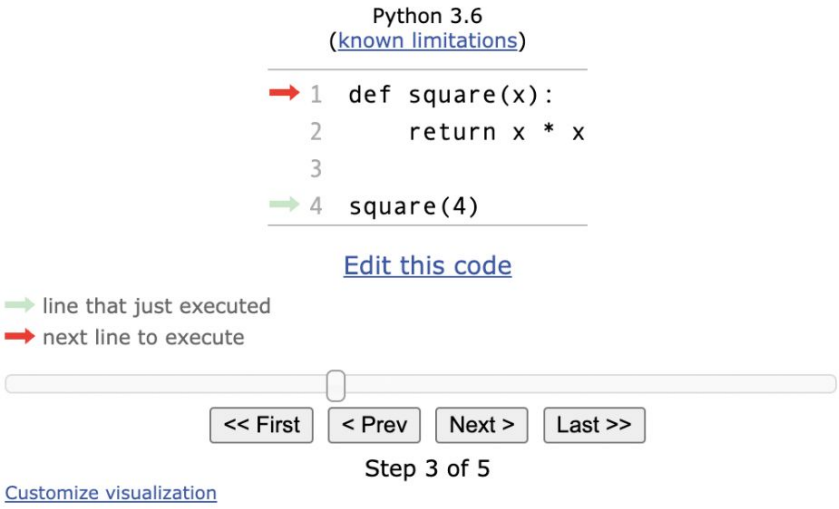


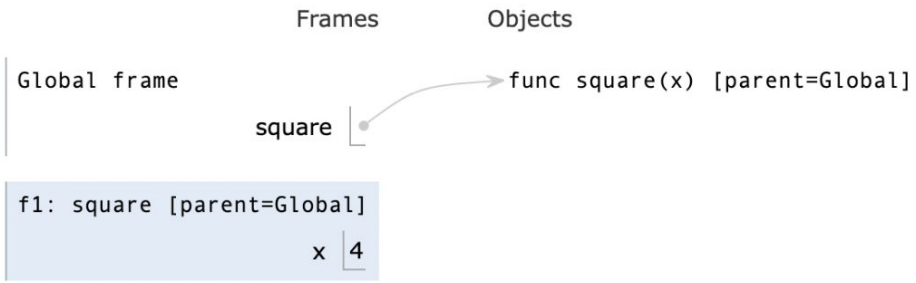


Step-by-Step: Call Expressions in Environment Diagrams
Example 1

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

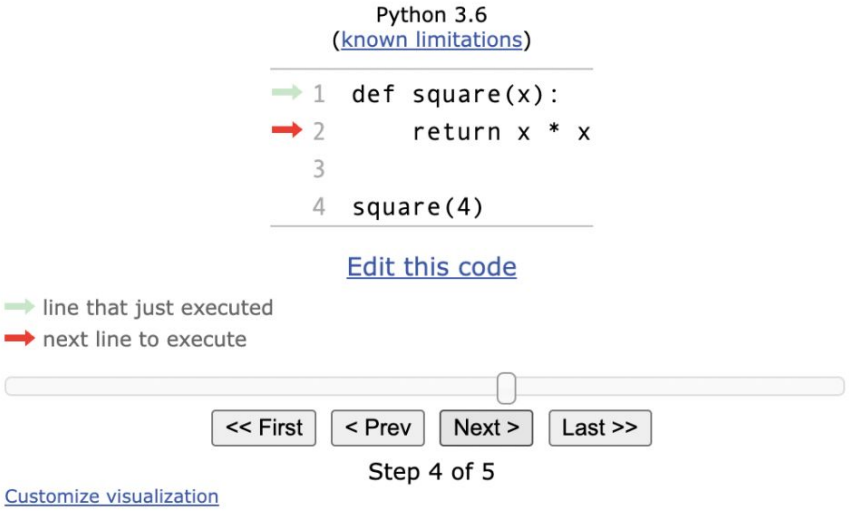


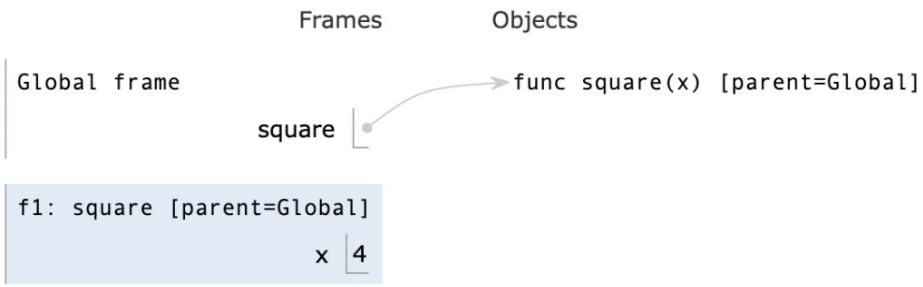


Step-by-Step: Call Expressions in Environment Diagrams Example 1

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

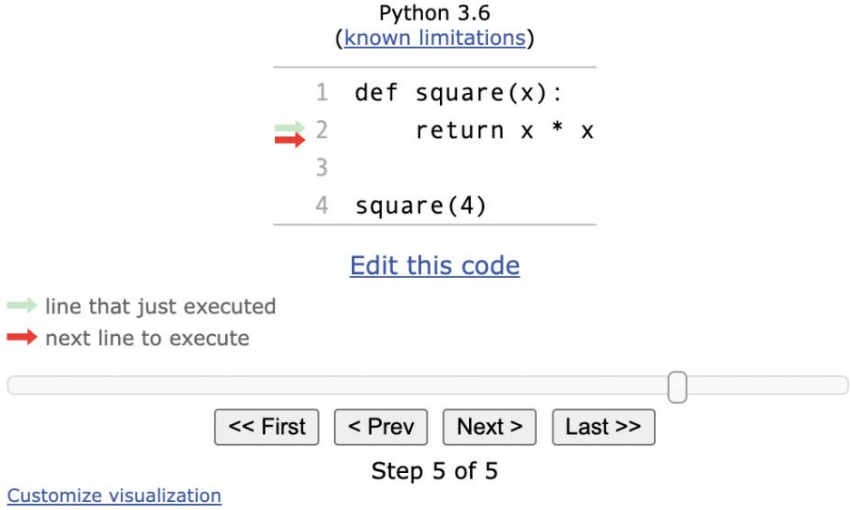


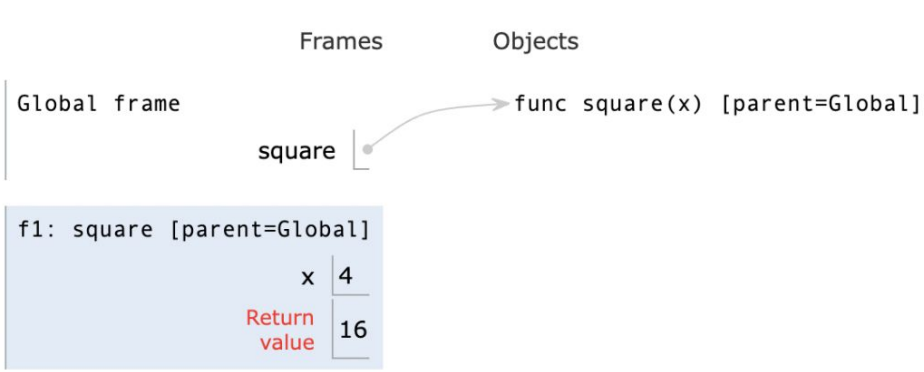


Step-by-Step: Call Expressions in Environment Diagrams Example 1

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called





Step-by-Step: Call Expressions in Environment Diagrams Example 2

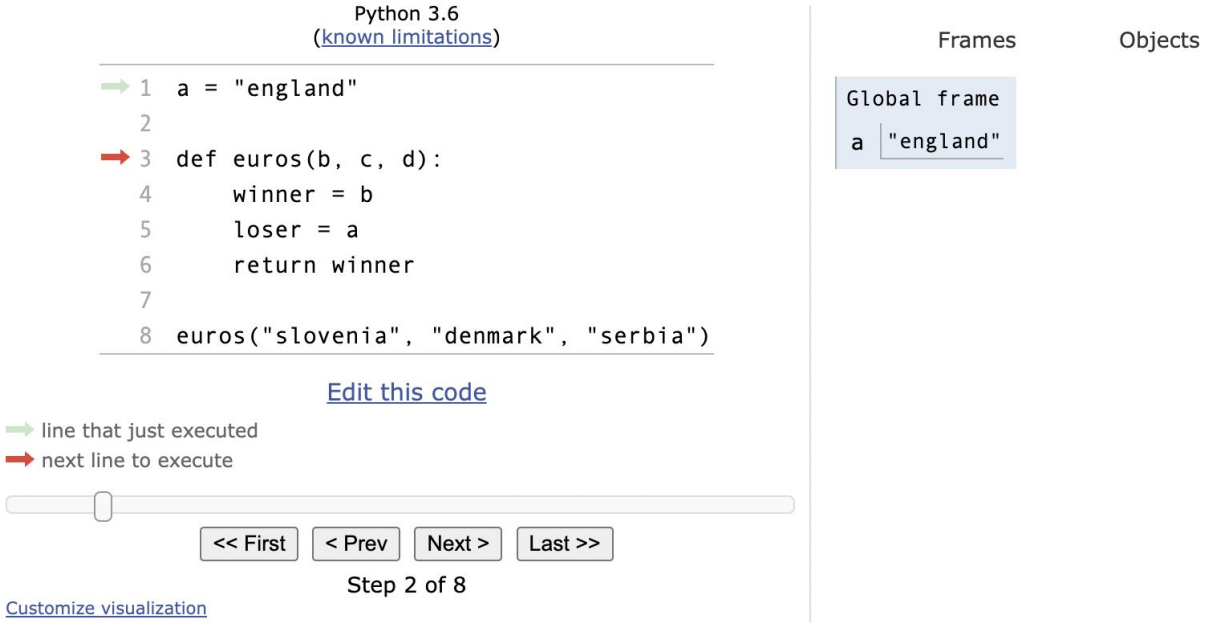
(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

Step-by-Step: Call Expressions in Environment Diagrams Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

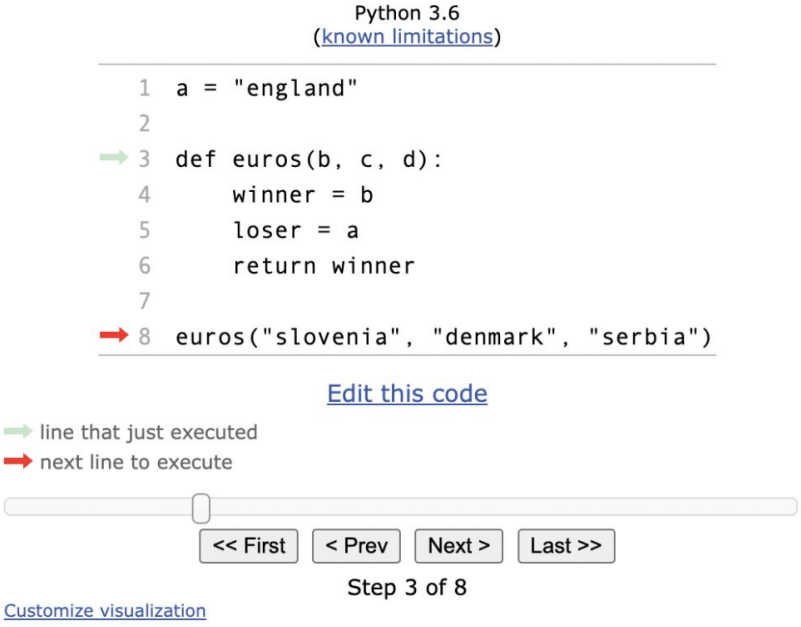
notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

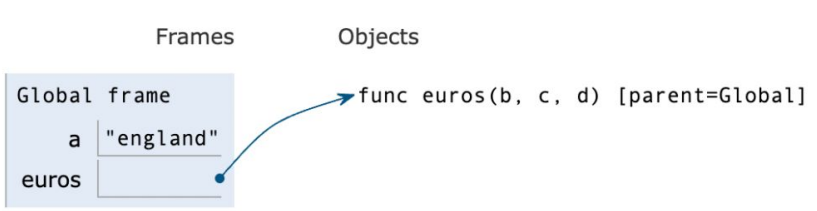


Step-by-Step: Call Expressions in Environment Diagrams Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

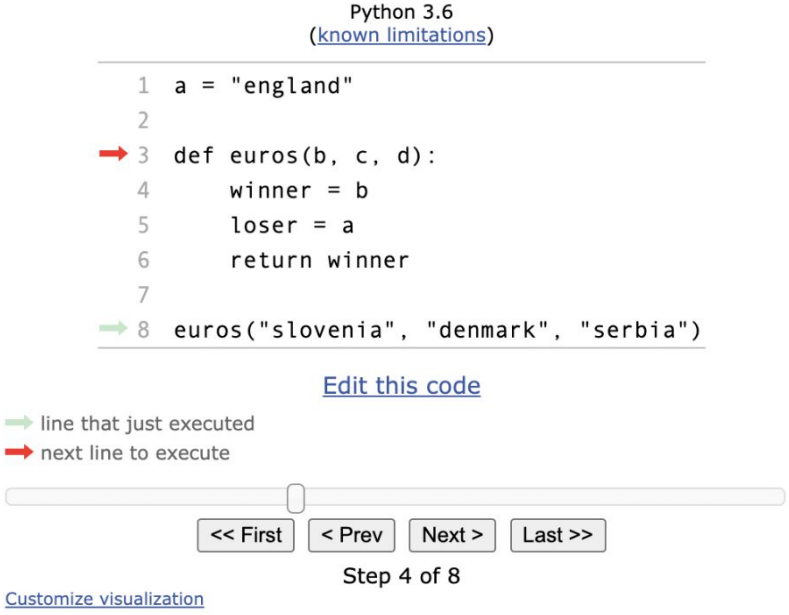


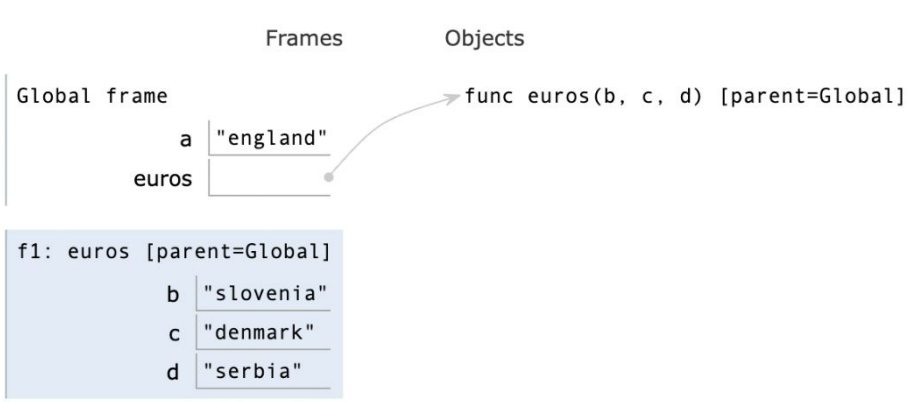


Step-by-Step: Call Expressions in Environment Diagrams Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called





Step-by-Step: Call Expressions in Environment Diagrams
Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

Python 3.6
(known limitations)

```
1  a = "england"
2
3  def euros(b, c, d):
4      winner = b
5      loser = a
6      return winner
7
8  euros("slovenia", "denmark", "serbia")
```

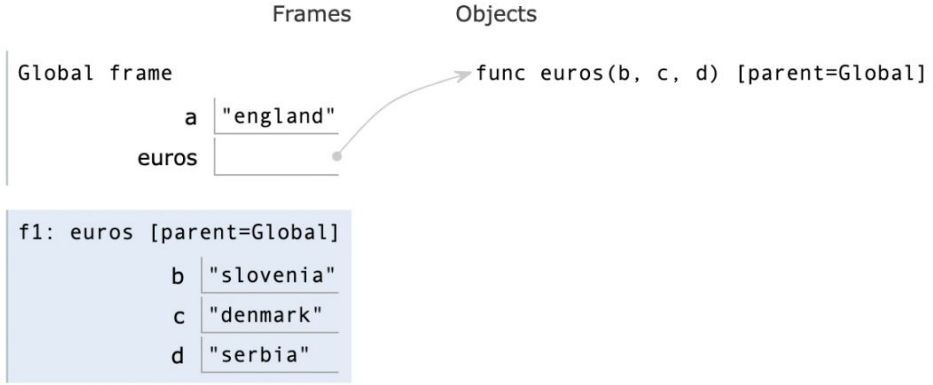
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 5 of 8

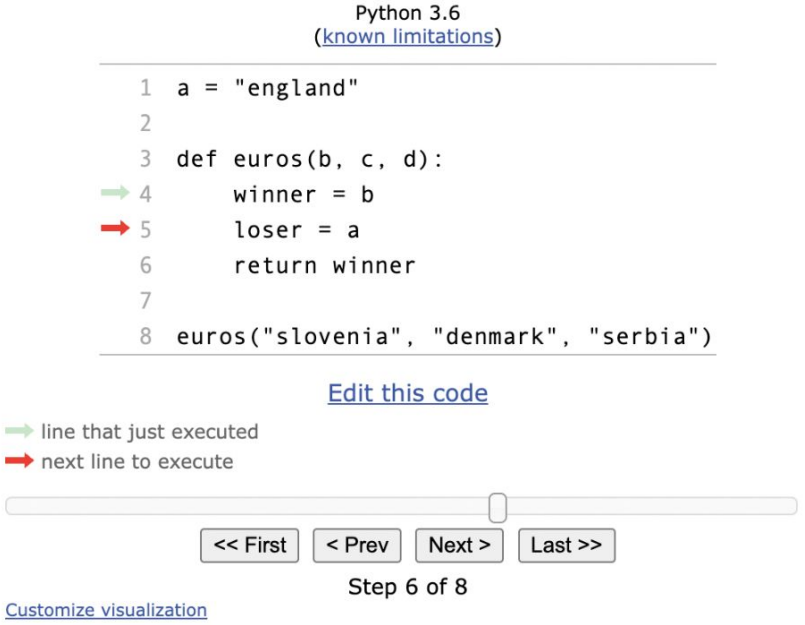
Customize visualization

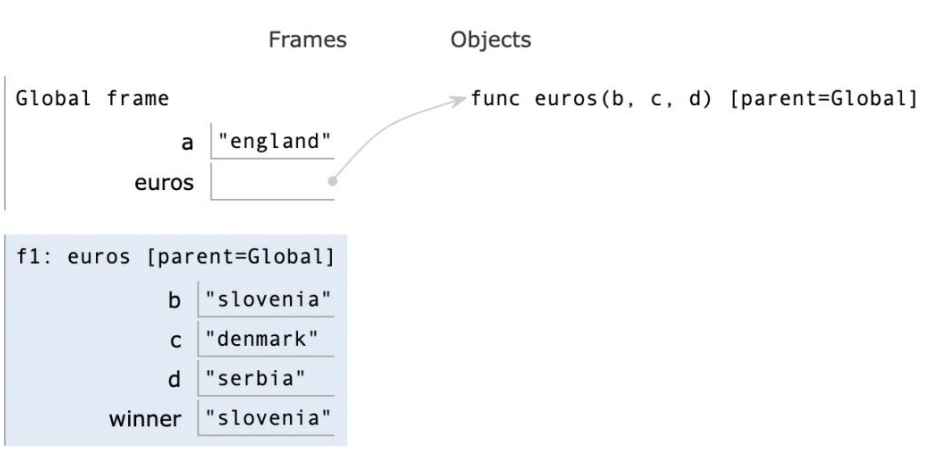


Step-by-Step: Call Expressions in Environment Diagrams Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

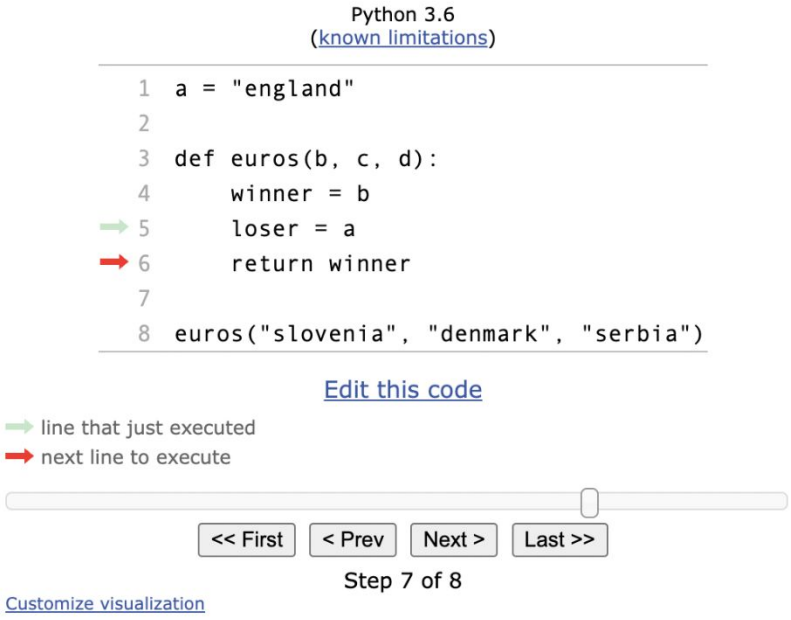


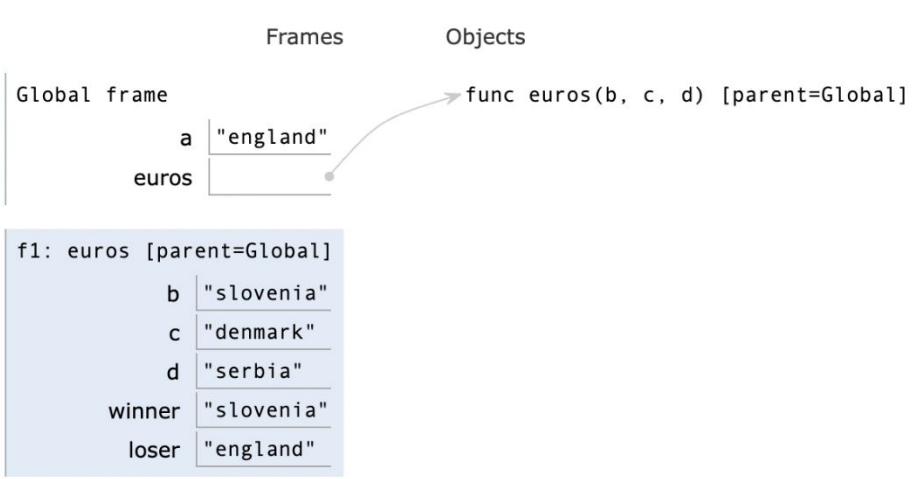


Step-by-Step: Call Expressions in Environment Diagrams Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called





Step-by-Step: Call Expressions in Environment Diagrams Example 2

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

Python 3.6
(known limitations)

```
1  a = "england"
2
3  def euros(b, c, d):
4      winner = b
5      loser = a
6      return winner
7
8  euros("slovenia", "denmark", "serbia")
```

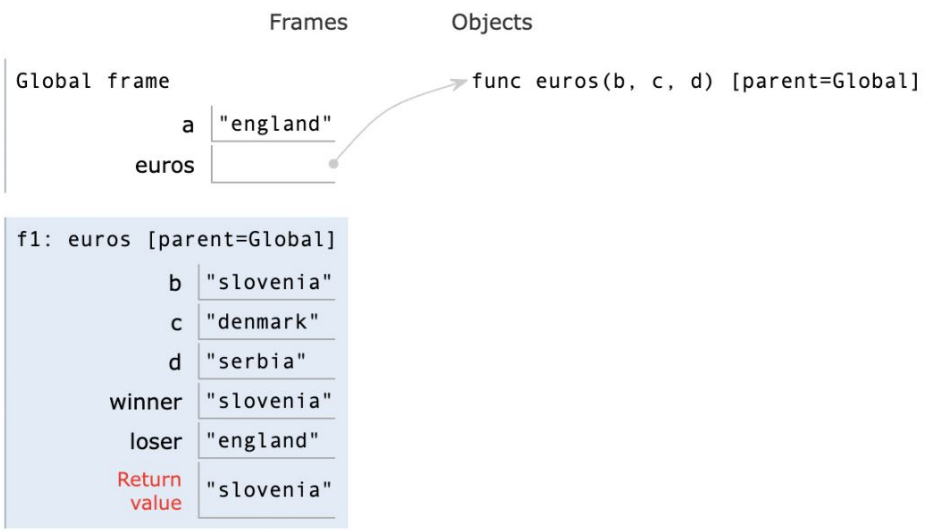
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 8 of 8

Customize visualization



Lambda

GUIDE
Step-by-Step: Lambdas in Environment Diagrams

(1) draw the lambda function object with: func & λ & formal parameters & parent frame

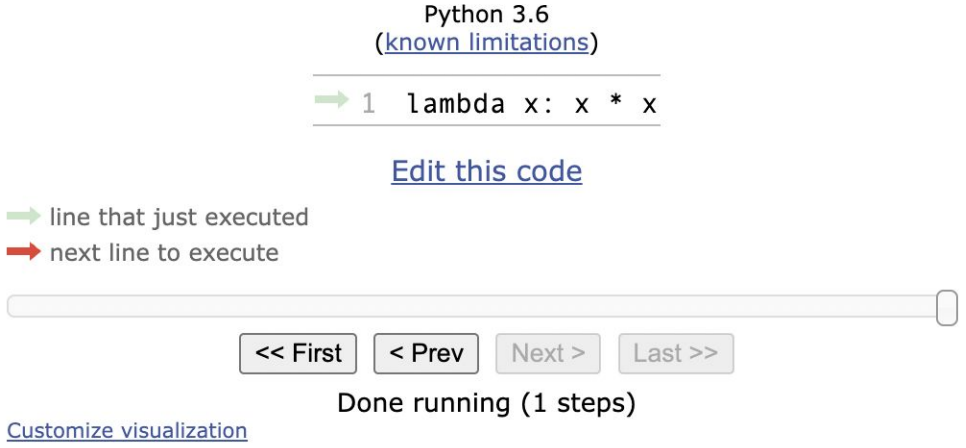
notes
- a function's parent frame is the frame in which the function was defined
- lambda expressions (unlike def statements) do not create any new bindings in the environment

Step-by-Step: Lambdas in Environment Diagrams
Example 1

(1) draw the lambda function object with: func & λ & formal parameters & parent frame

notes

- **a function's parent frame is the frame in which the function was defined**
- **lambda expressions (unlike def statements) do not create any new bindings in the environment**



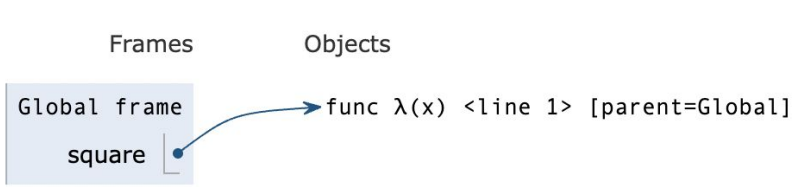
Draw

Step-by-Step: Lambdas in Environment Diagrams
Example 1

(1) draw the lambda function object with: func & λ & formal parameters & parent frame

notes

- **a function's parent frame is the frame in which the function was defined**
- **lambda expressions (unlike def statements) do not create any new bindings in the environment**



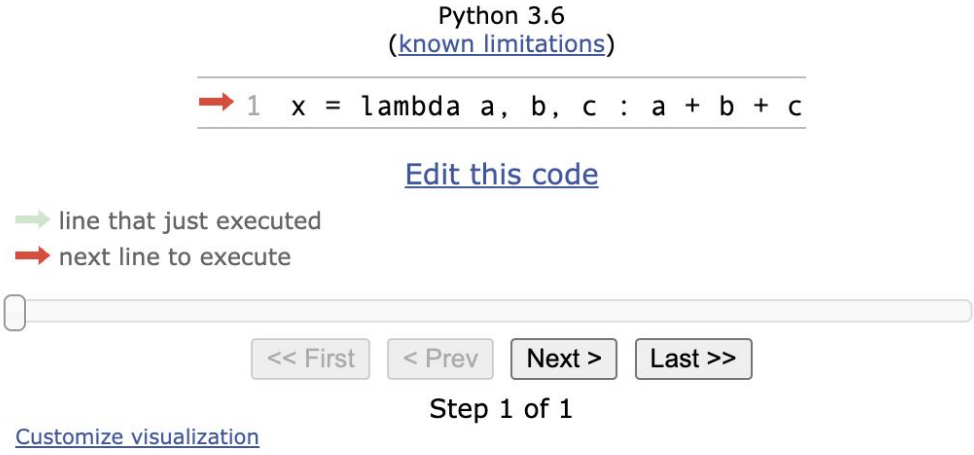
Draw

Step-by-Step: Lambdas in Environment Diagrams
Example 2

(1) draw the lambda function object with: func & λ & formal parameters & parent frame

notes

- **a function's parent frame is the frame in which the function was defined**
- **lambda expressions (unlike def statements) do not create any new bindings in the environment**



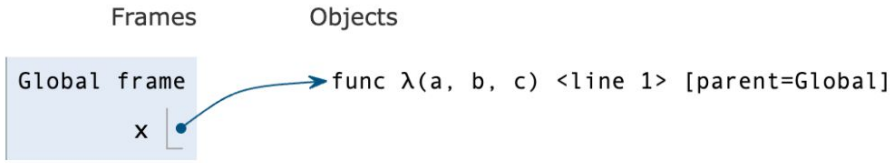
Draw

Step-by-Step: Lambdas in Environment Diagrams
Example 2

(1) draw the lambda function object with: func & λ & formal parameters & parent frame

notes

- **a function's parent frame is the frame in which the function was defined**
- **lambda expressions (unlike def statements) do not create any new bindings in the environment**



Draw

Environment Diagrams: Important Examples

Step-by-Step: Assignment Statements

(1) evaluate the expression on the right of the = sign to get a value/object
- when encountering a name while evaluating, always search the current frame first
- then, search the parent frame, and then that frame's parent frame, etc. (until global frame)

(2) does the name on the left of the = already exist in the *current* frame?

↳ yes
- erase the current binding (either a value or object)
- bind the name to the value/object from (1)

↳ no
- bind the new name to the value/object

notes
- if there are multiple expressions in a statement, evaluate all expressions first from left to right before making any bindings

Step-by-Step: Lambdas

(1) draw the lambda function object with: func & λ & formal parameters & parent frame

notes
- a function's parent frame is the frame in which the function was defined
- lambda expressions (unlike def statements) do not create any new bindings in the environment

Step-by-Step: def Statements

(1) draw the function object with: func & intrinsic name & formal parameters & parent frame
(2) does the intrinsic name of the function already exist in the current frame?
    ↳ yes
        - erase the current bindings
    ↳ no
        - write it in
(3) bind the newly created function object to this name

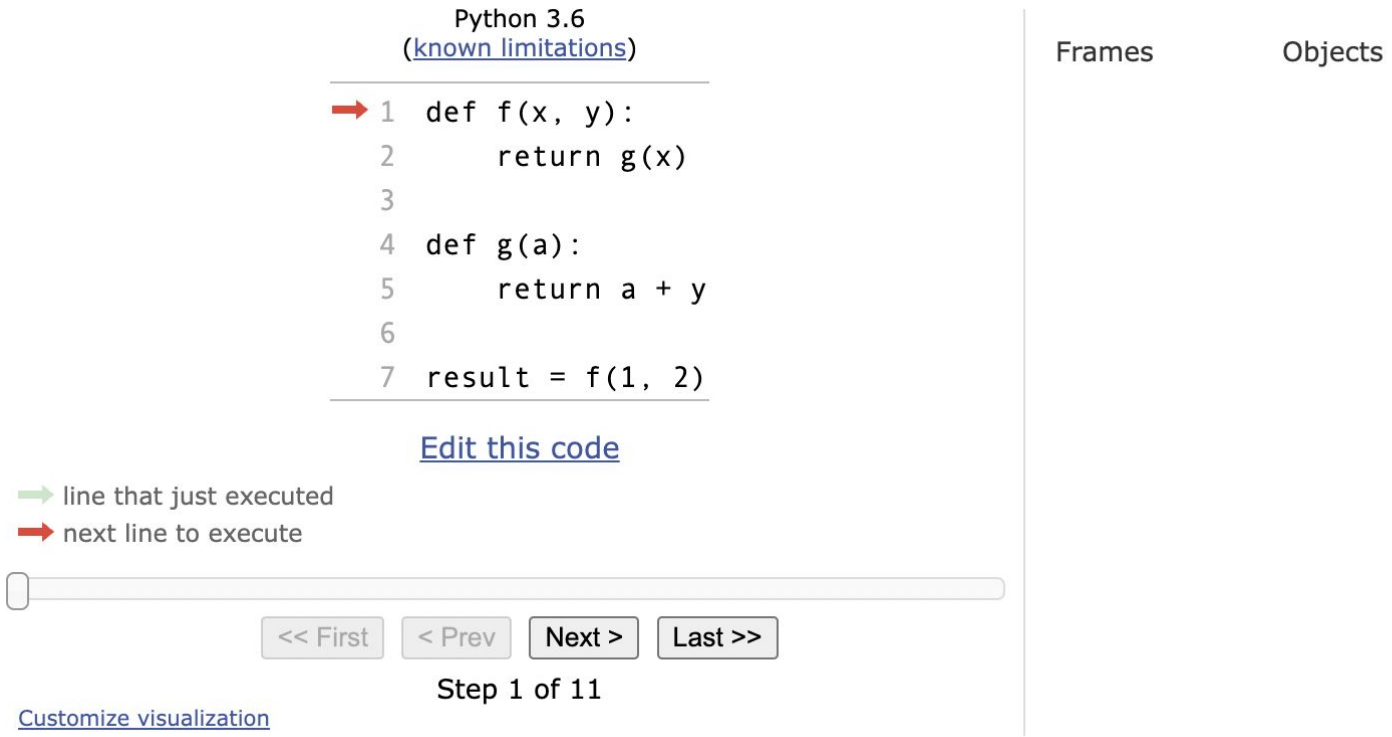
notes
- a function's parent frame is the frame in which the function was *defined*

Step-by-Step: Call Expressions

(1) evaluate the operator (should be a function)
(2) evaluate the operands left to right to obtain a value/object for each
(3) open a new frame (necessary for every call expression)
(4) label the new frame with: sequential frame number & intrinsic name & parent frame of function
(5) bind the formal parameters of the function to the arguments whose values/objects you found in (2)
(6) execute the body of the function until a return value is obtained
(7) write down the return value in the frame

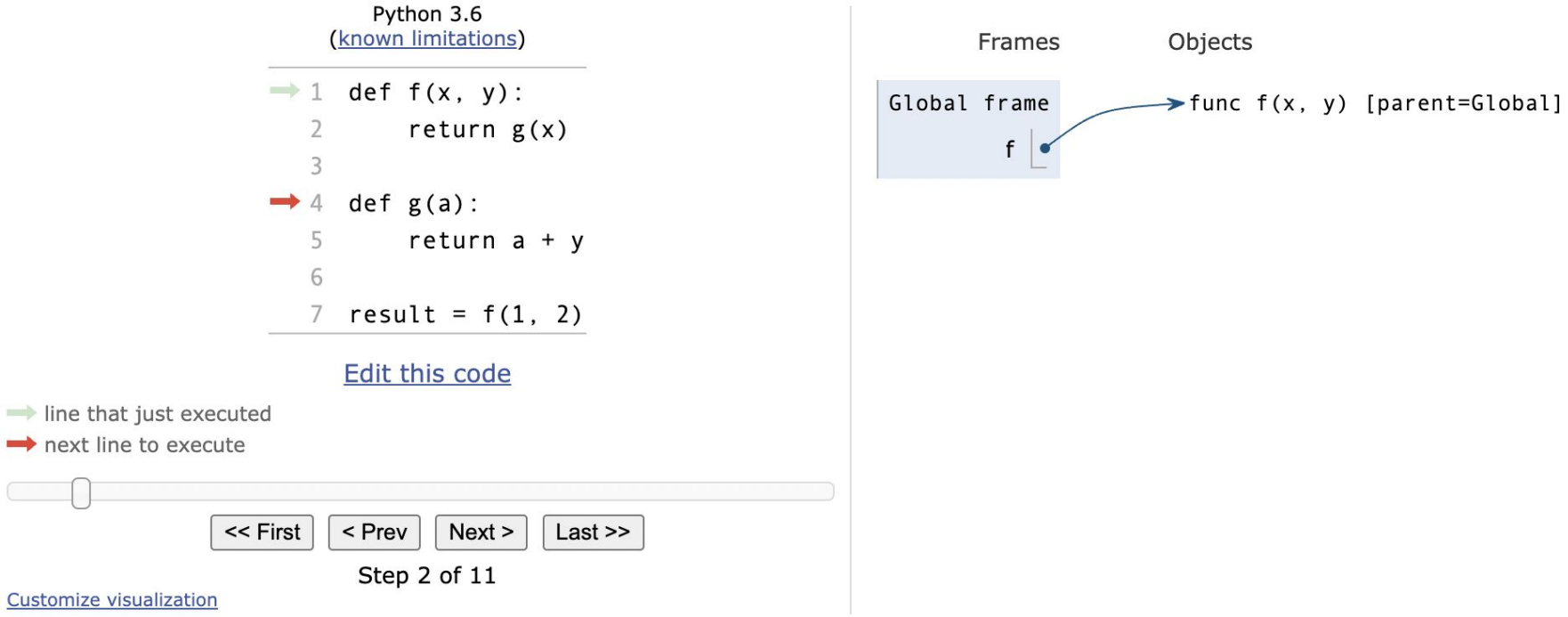
notes
- if a function does not have a return value, it implicitly returns None
- do not draw frames for built-in or imported functions e.g., min(...) and add(...)
- with nested call expressions, remember to open frames in the other that they are called

Example 1: Local Names



Draw

Example 1: Local Names



Example 1: Local Names

Python 3.6
(known limitations)

```
1  def f(x, y):
2      return g(x)
3
4  def g(a):
5      return a + y
6
7  result = f(1, 2)
```

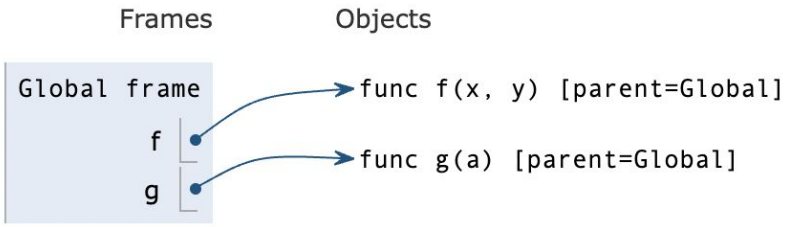
Edit this code

line that just executed
next line to execute

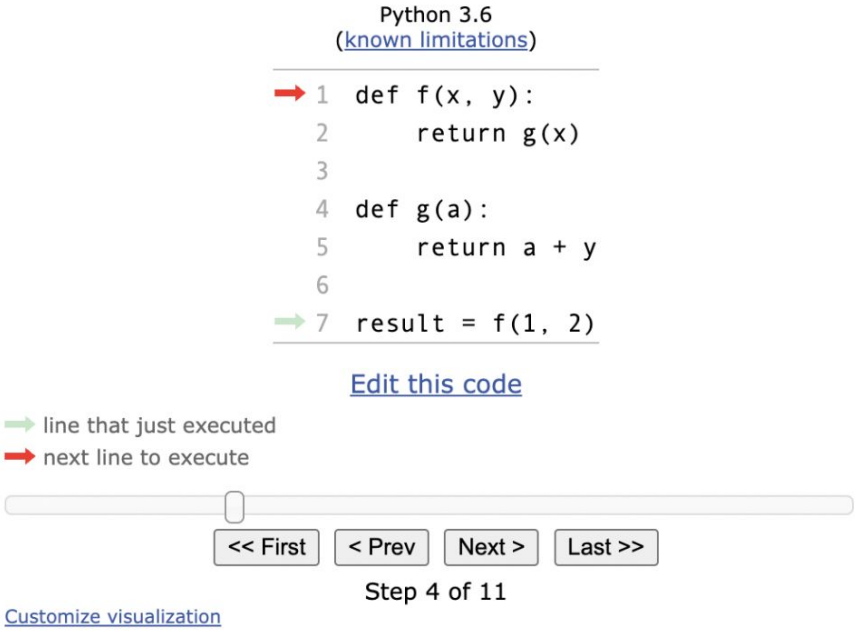
<< First | < Prev | Next > | Last >>

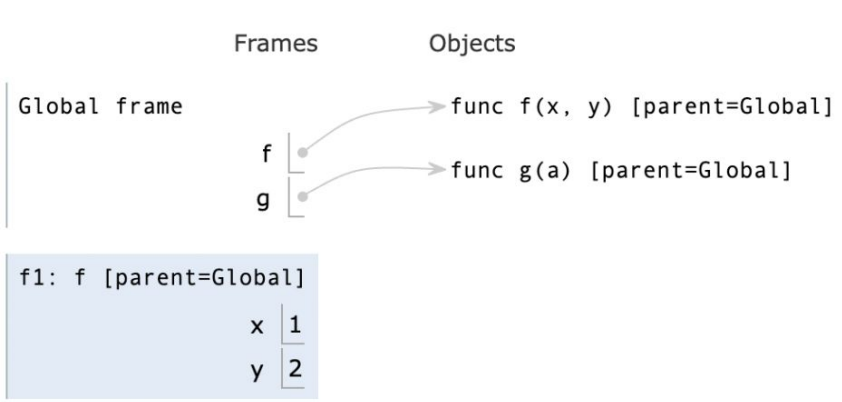
Step 3 of 11

Customize visualization

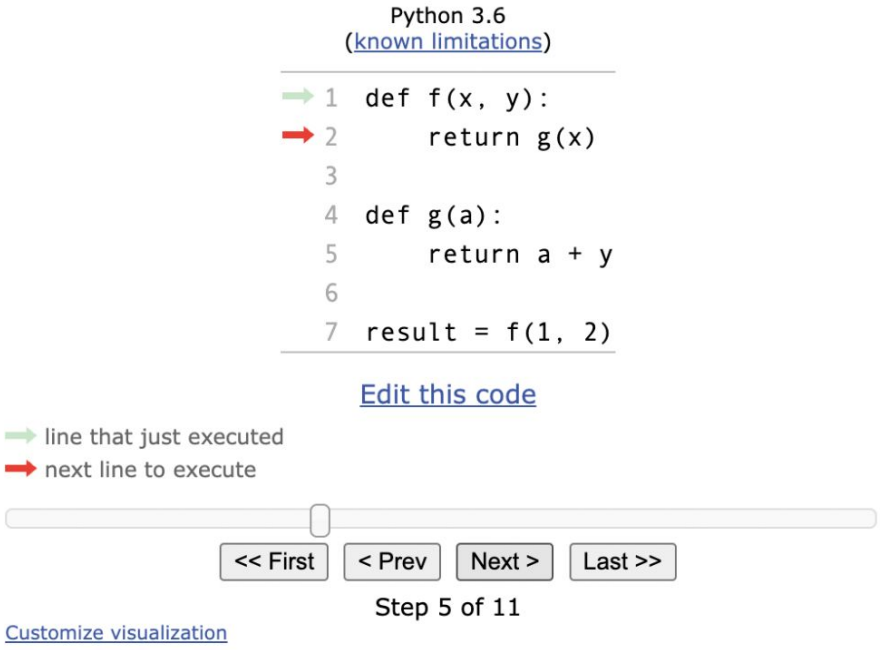


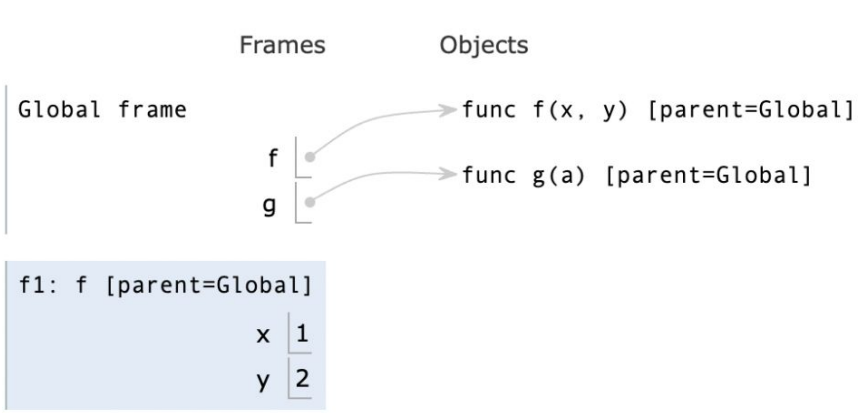
Example 1: Local Names



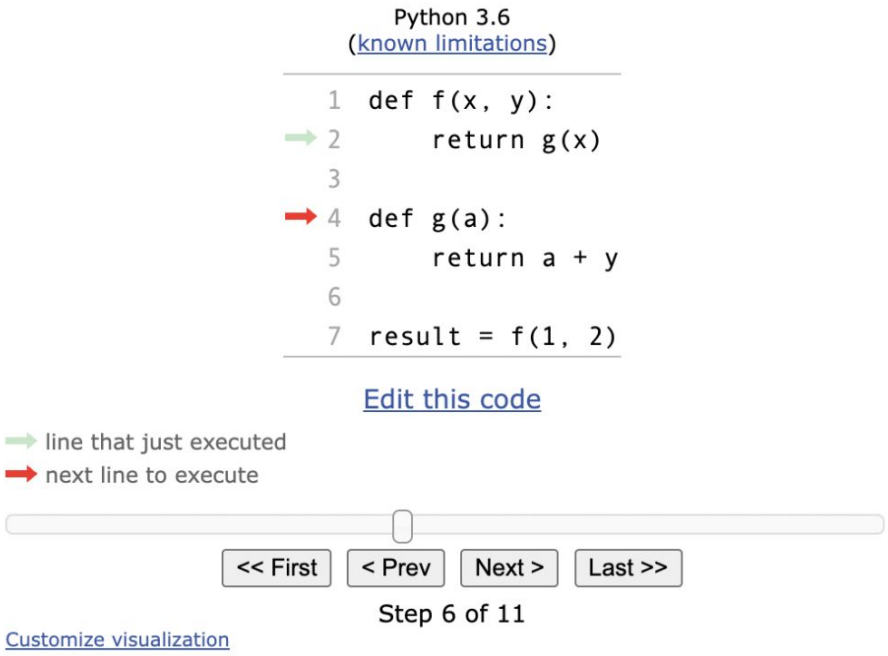


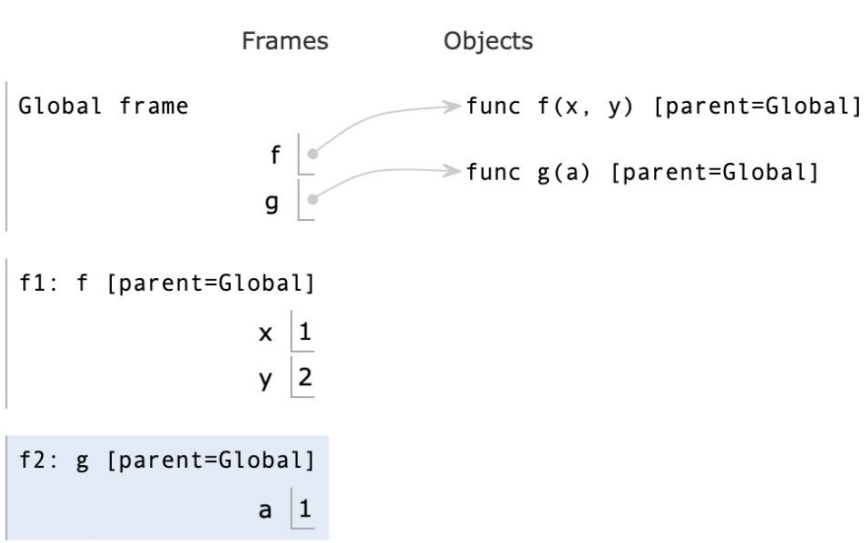
Example 1: Local Names



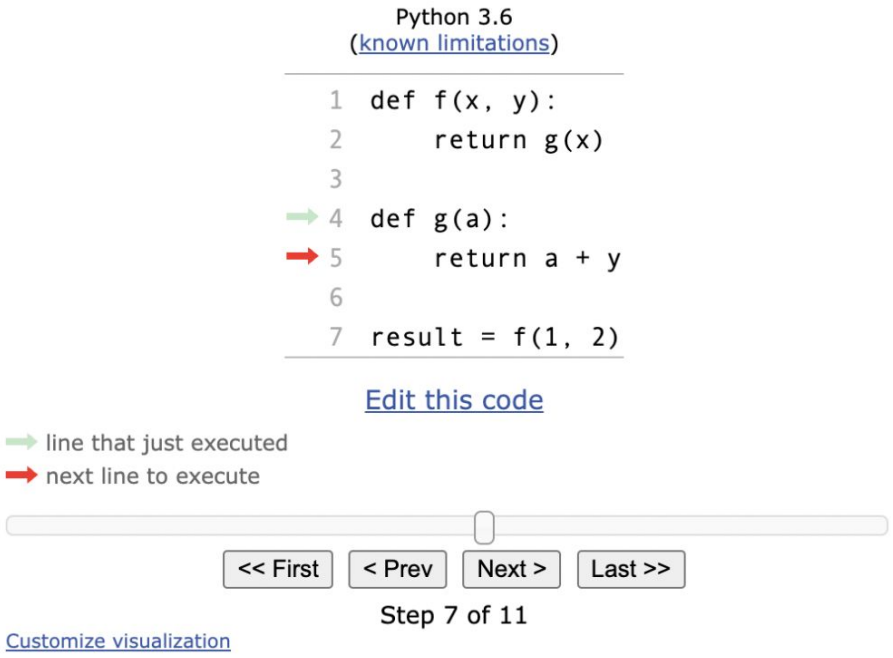


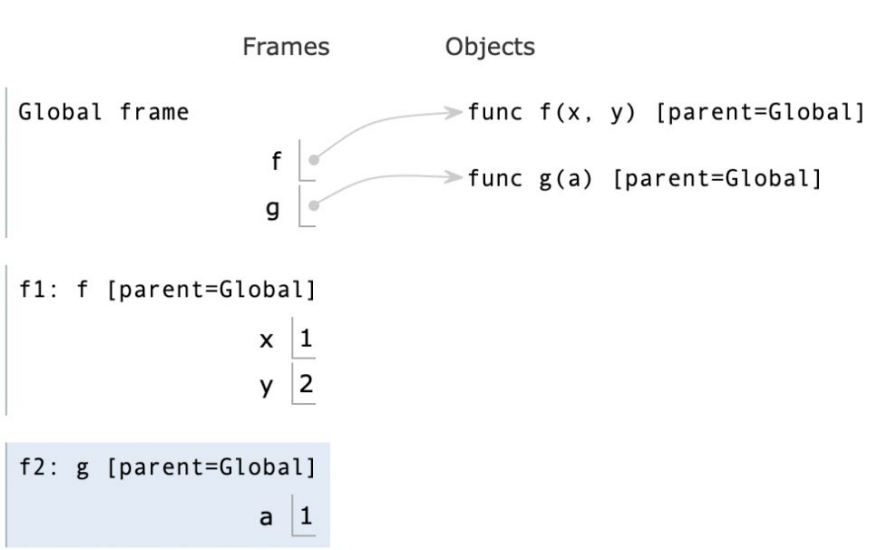
Example 1: Local Names



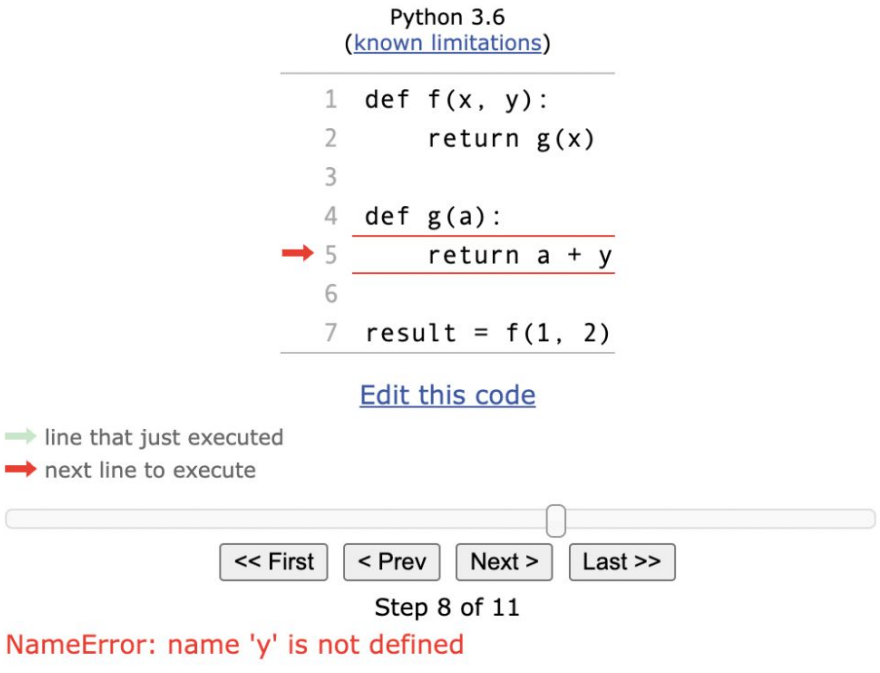


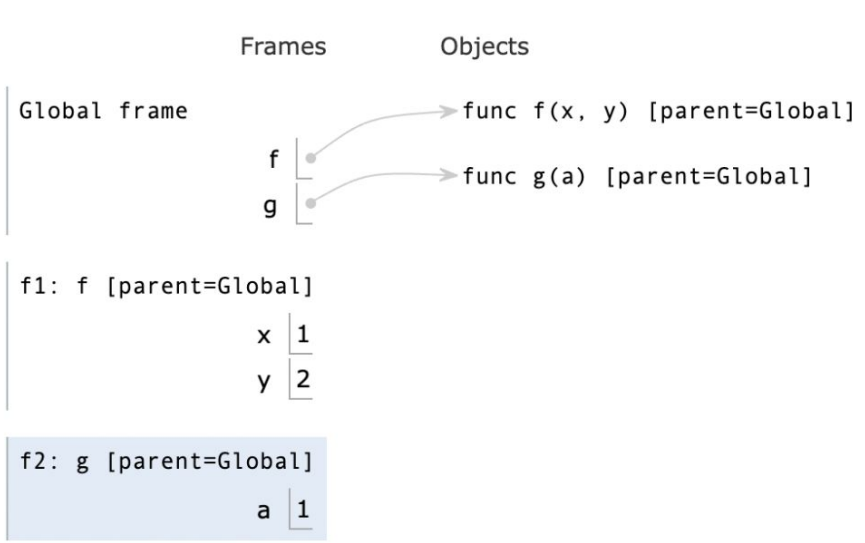
Example 1: Local Names



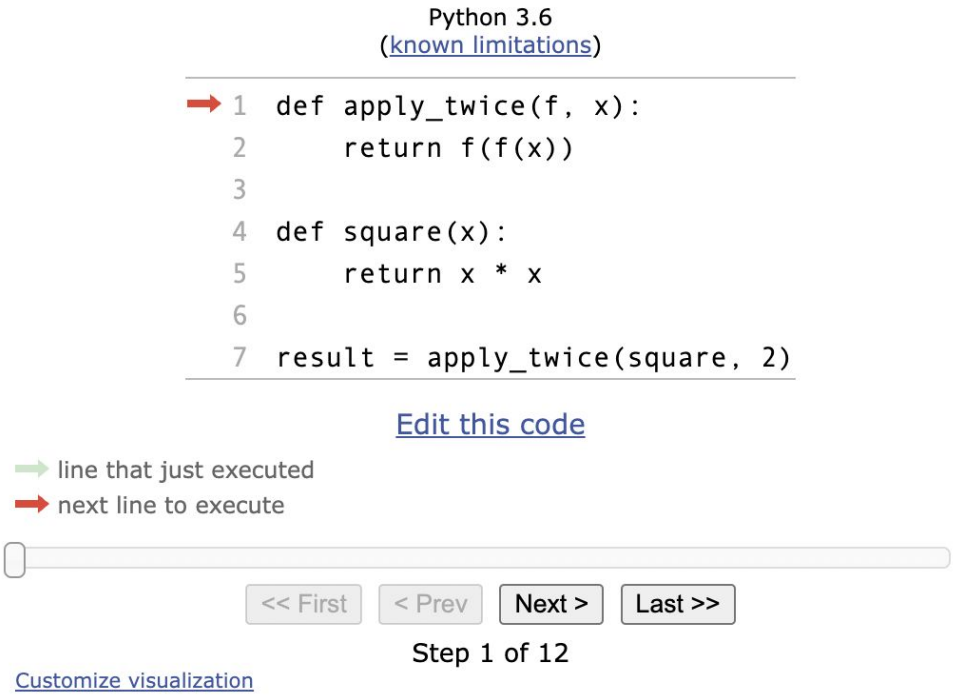


Example 1: Local Names





Example 2: Higher-Order Function



Draw

Example 2: Higher-Order Function

Python 3.6
(known limitations)

```
1 def apply_twice(f, x):
2     return f(f(x))
3
4 def square(x):
5     return x * x
6
7 result = apply_twice(square, 2)
```

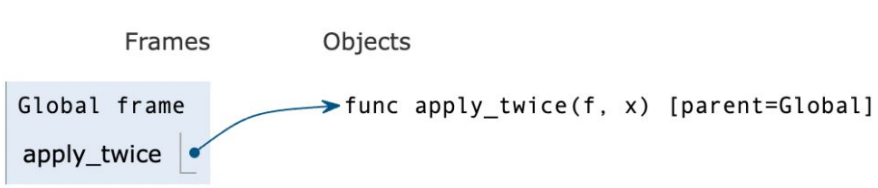
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 2 of 12

Customize visualization



Example 2: Higher-Order Function

Python 3.6
(known limitations)

```
1  def apply_twice(f, x):
2      return f(f(x))
3
4  def square(x):
5      return x * x
6
7  result = apply_twice(square, 2)
```

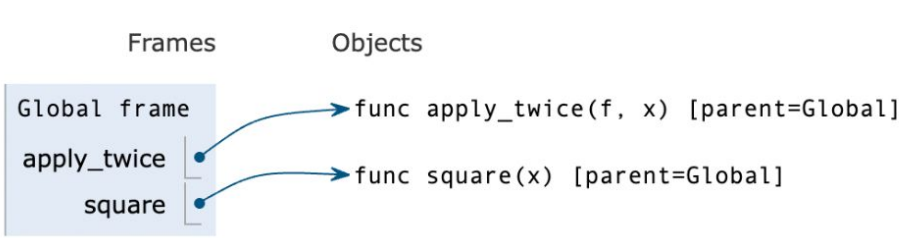
Edit this code

line that just executed
next line to execute

<< First | < Prev | Next > | Last >>

Step 3 of 12

Customize visualization



Example 2: Higher-Order Function

Python 3.6
(known limitations)

```
1  def apply_twice(f, x):
2      return f(f(x))
3
4  def square(x):
5      return x * x
6
7  result = apply_twice(square, 2)
```

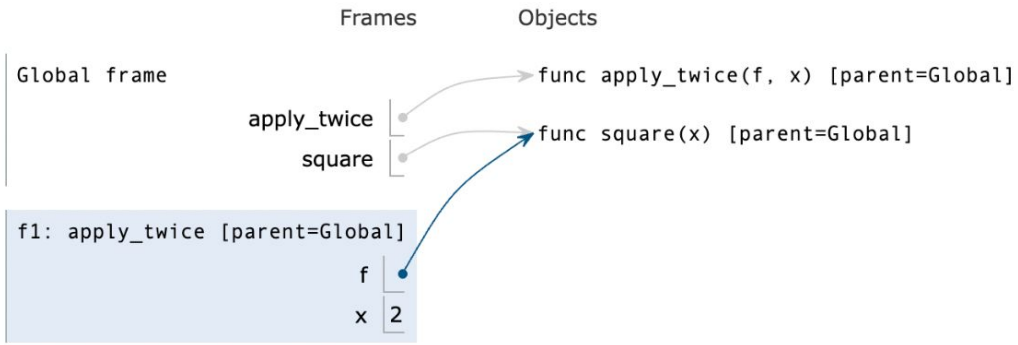
Edit this code

line that just executed
next line to execute

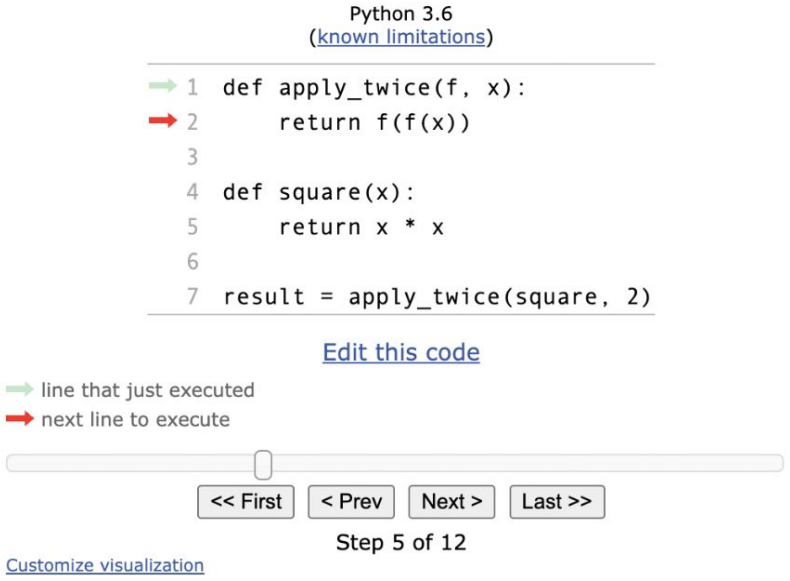
<< First | < Prev | Next > | Last >>

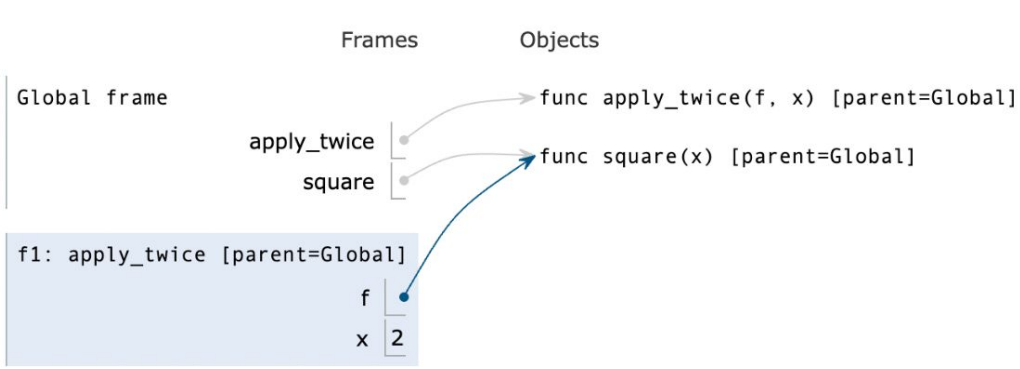
Step 4 of 12

Customize visualization

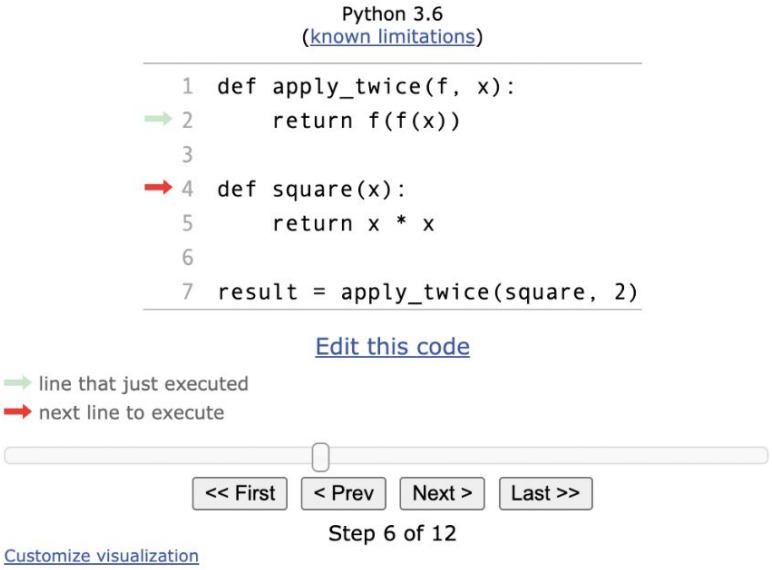


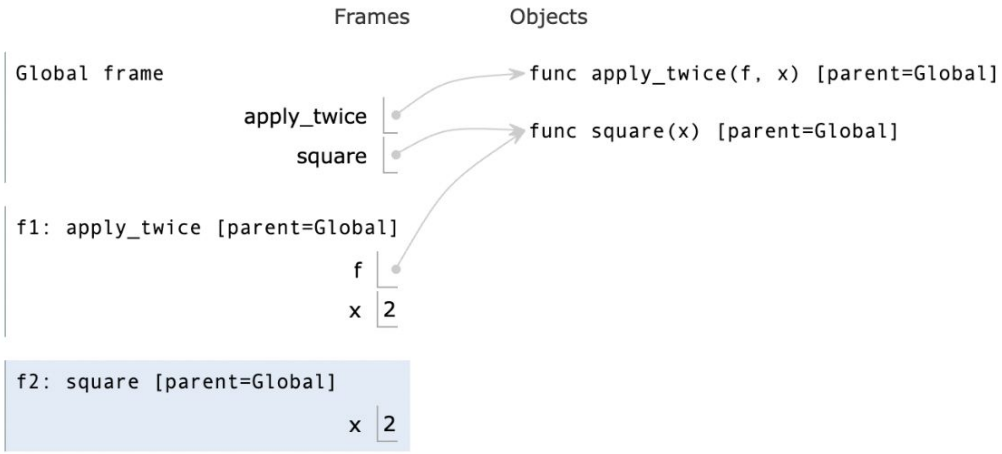
Example 2: Higher-Order Function



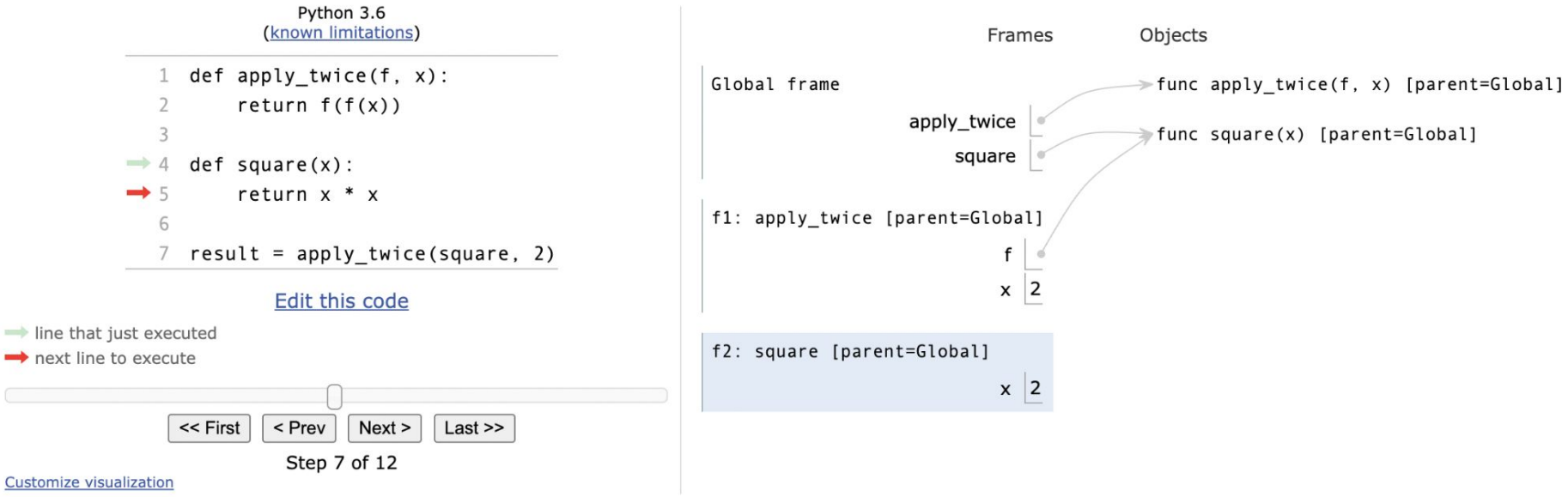


Example 2: Higher-Order Function

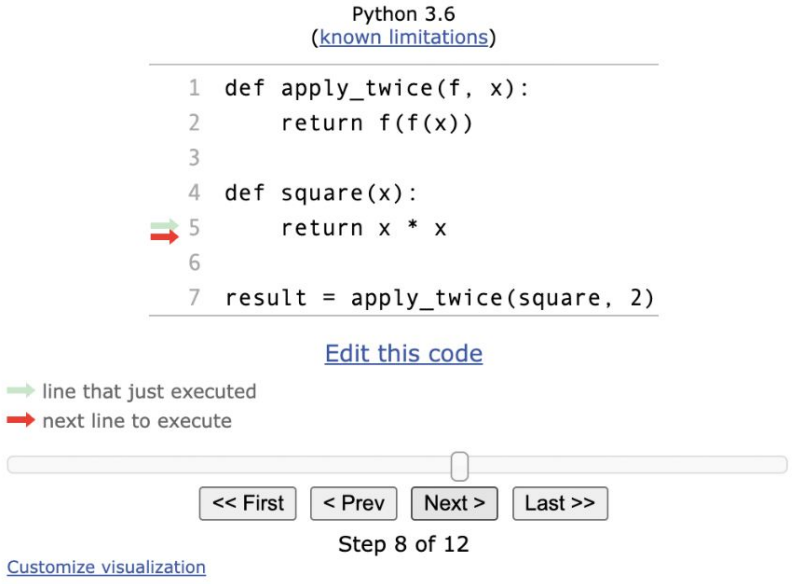


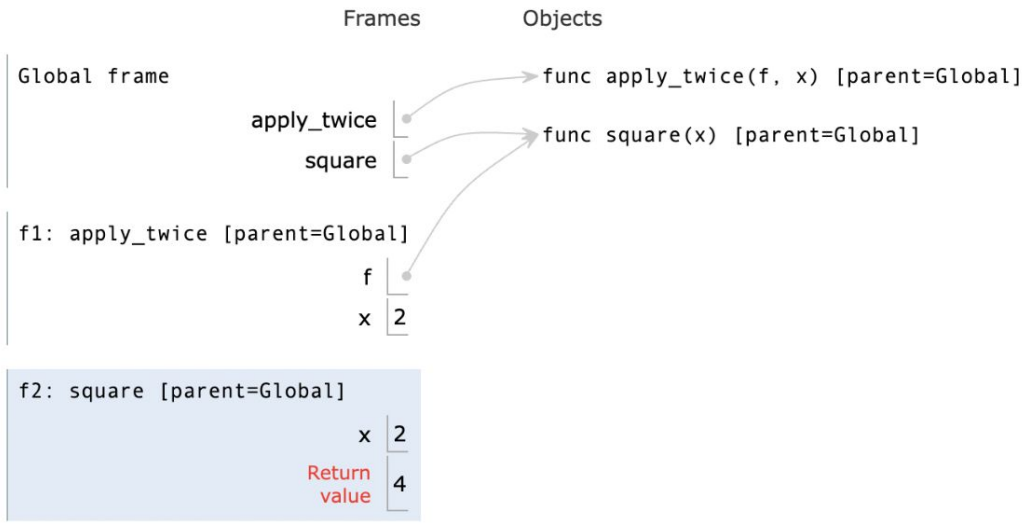


Example 2: Higher-Order Function

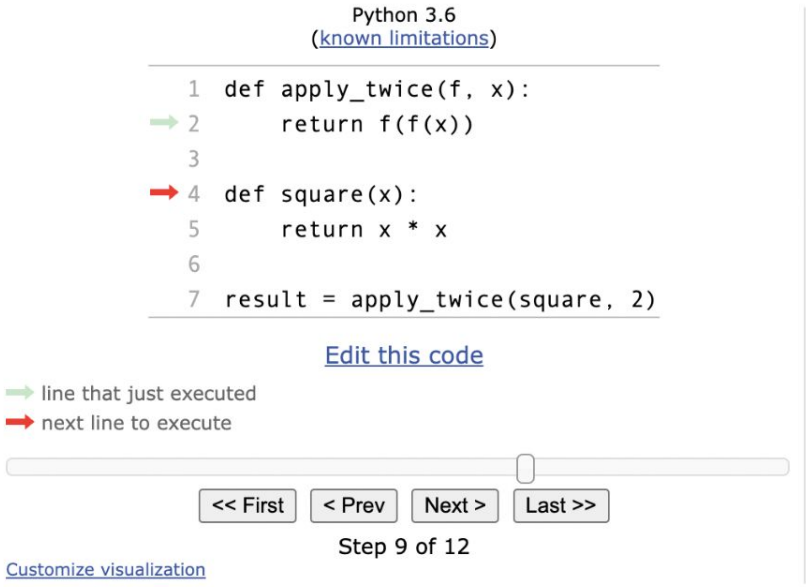


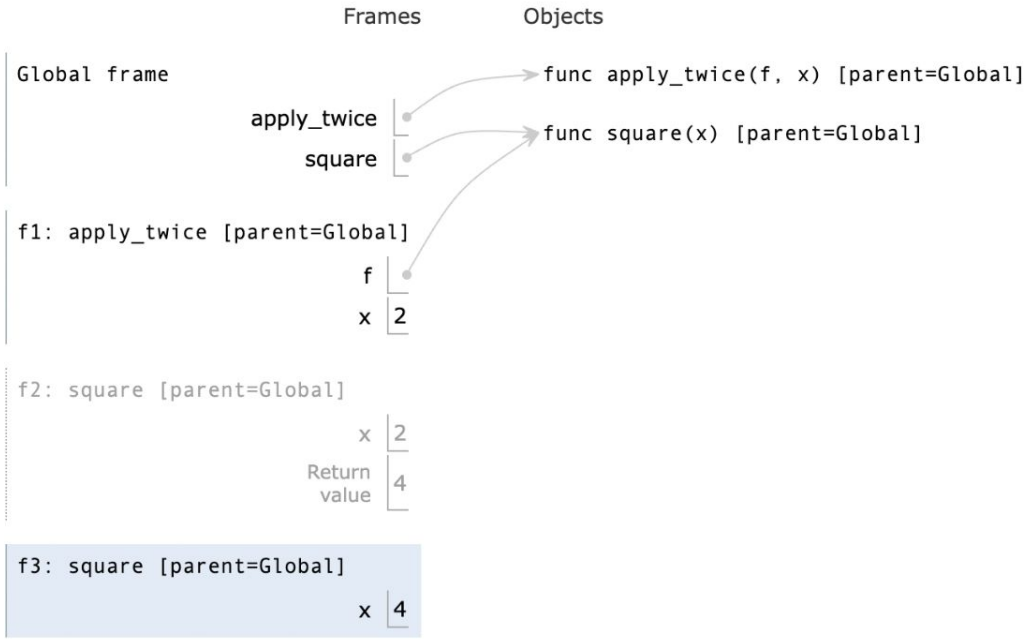
Example 2: Higher-Order Function



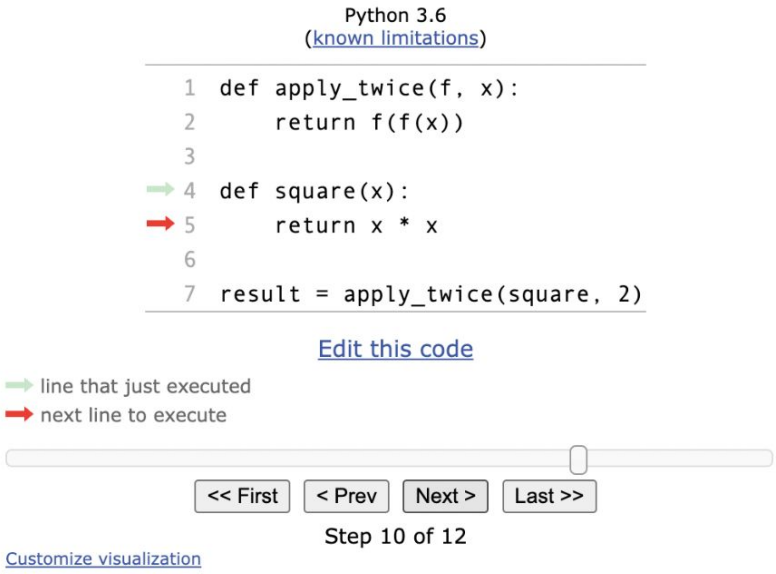


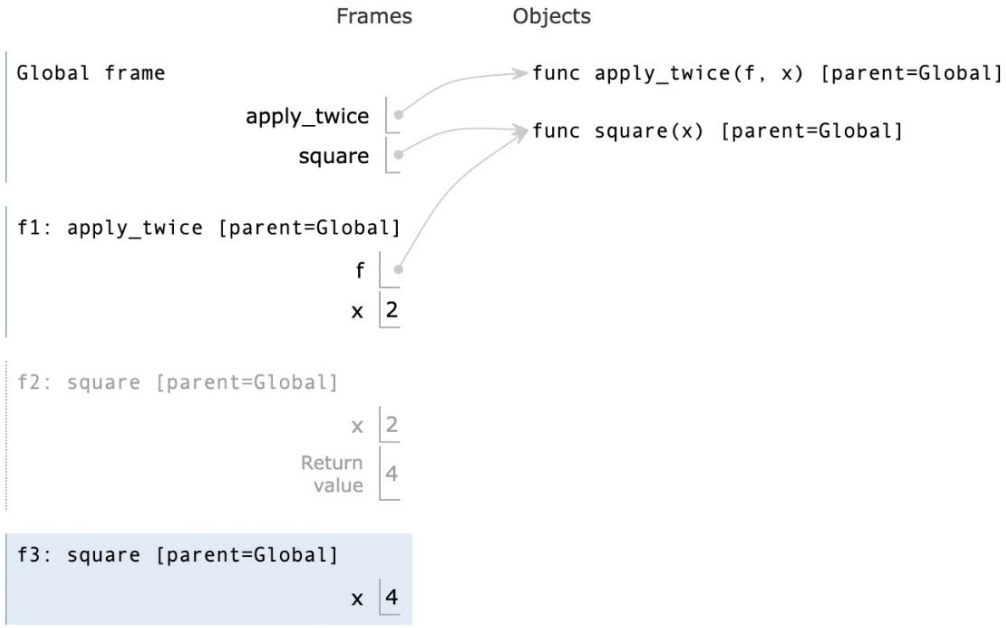
Example 2: Higher-Order Function





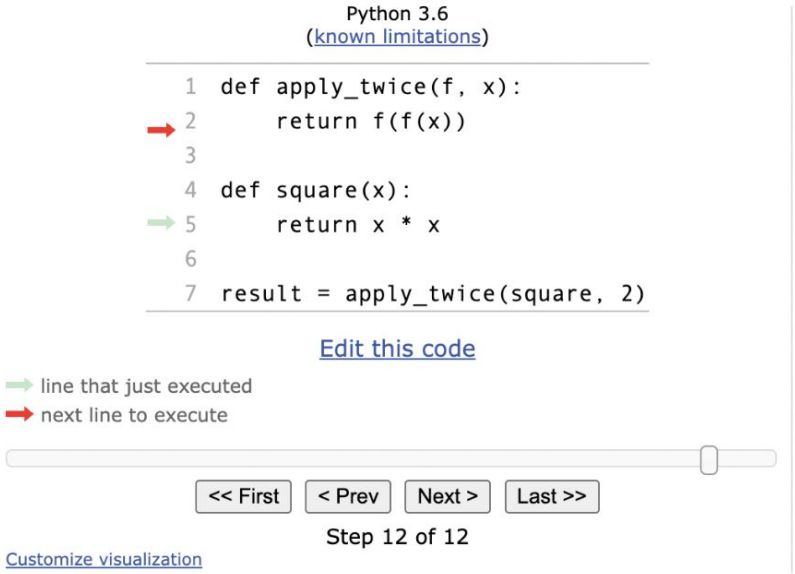
Example 2: Higher-Order Function

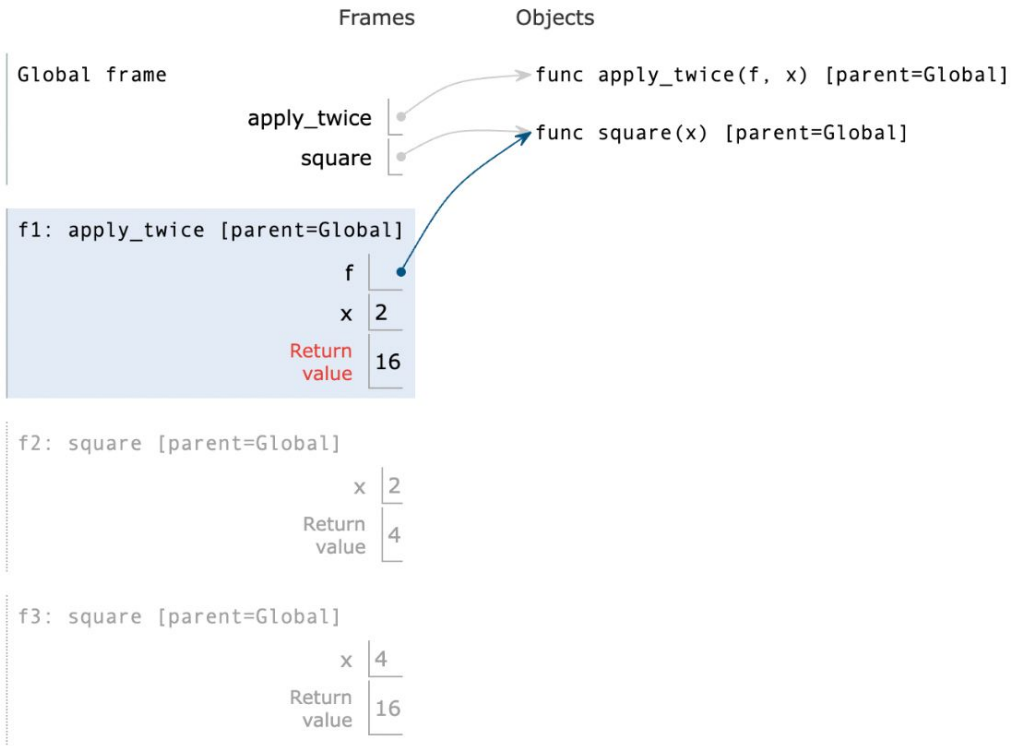




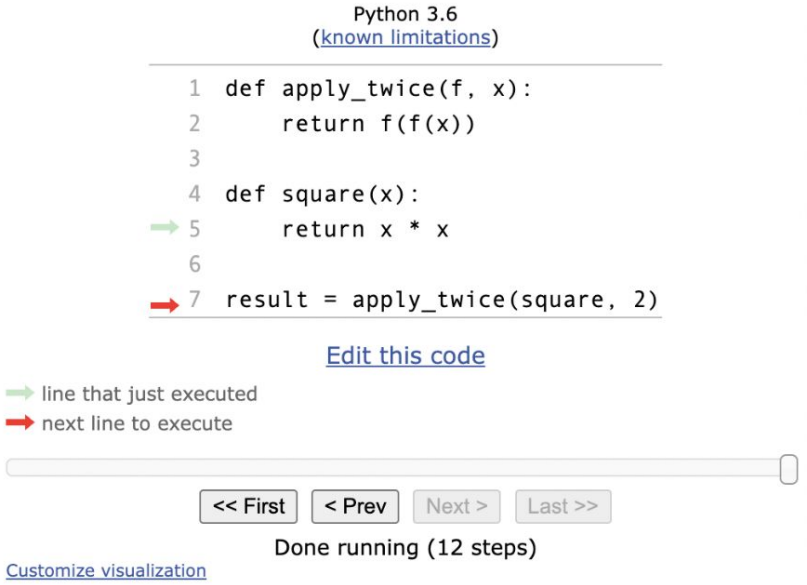
Example 2: Higher-Order Function

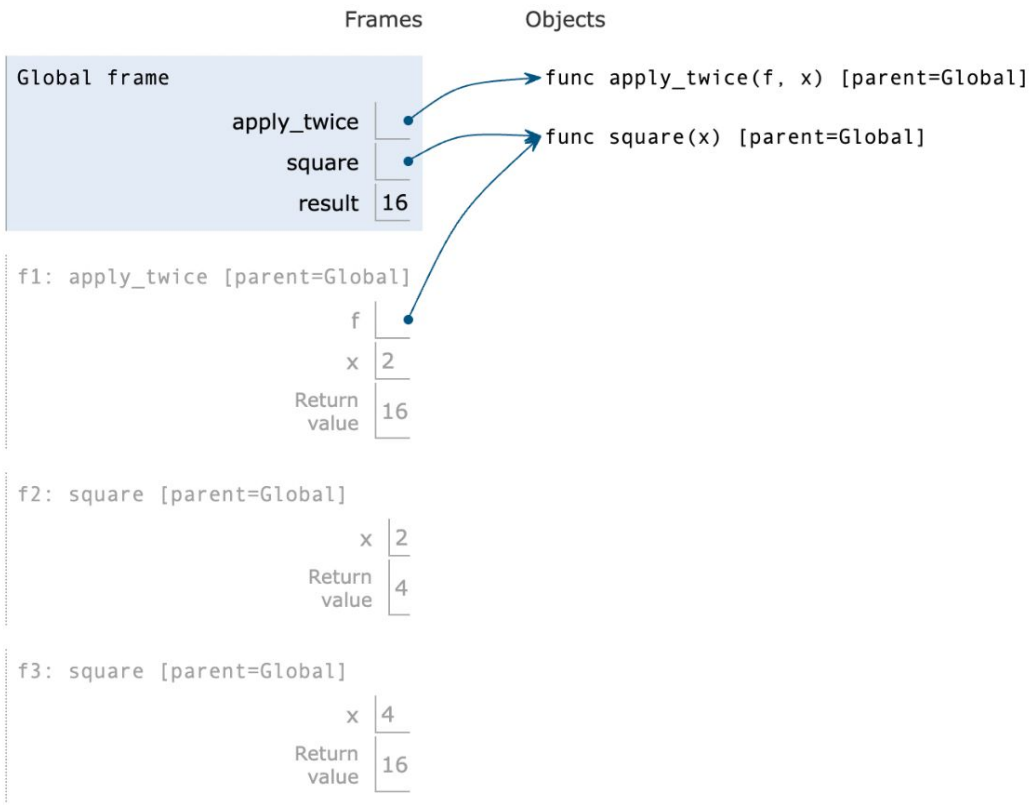
Example 2: Higher-Order Function





Example 2: Higher-Order Function



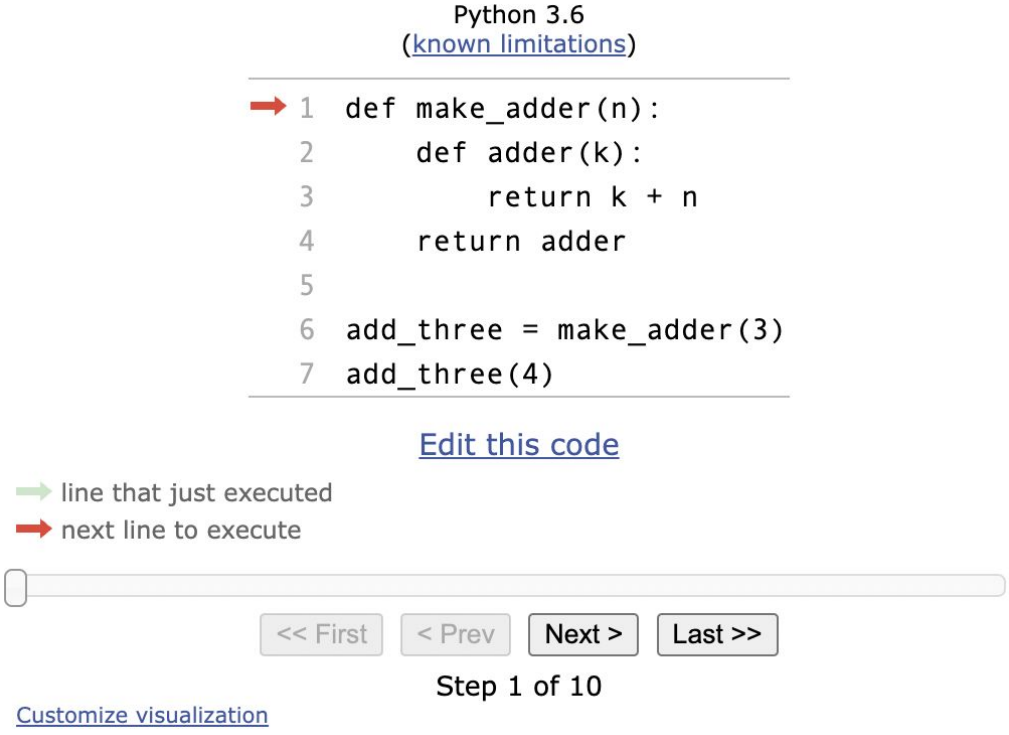


Example 3: Nested Definition

```
1 def make_adder(n):
2     def adder(k):
3         return k + n
4     return adder
5
```

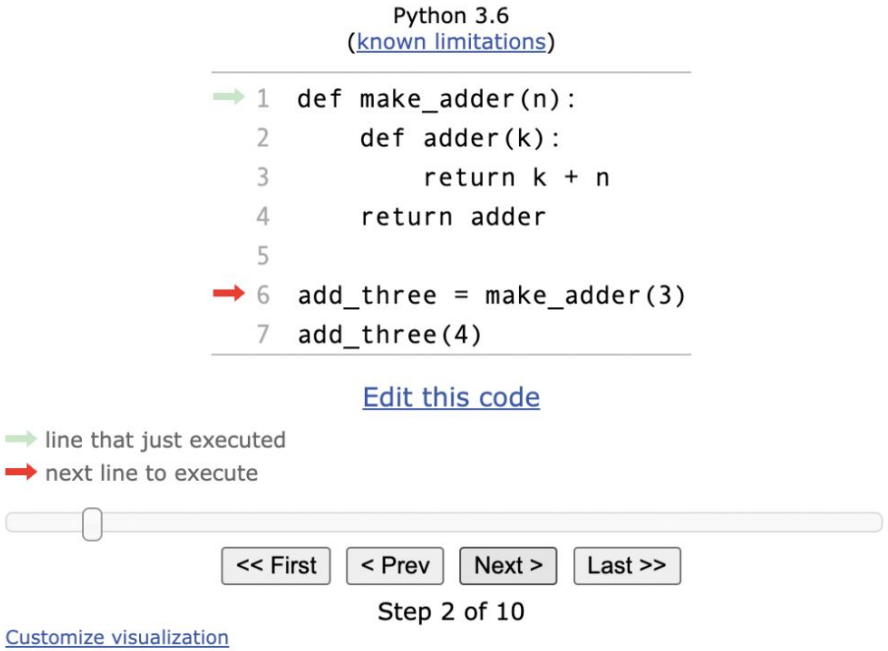
```
>>> add_three = make_adder(3)
>>> add_three(4)
7
>>> add_three(5)
8
>>> add_three(6)
9
```

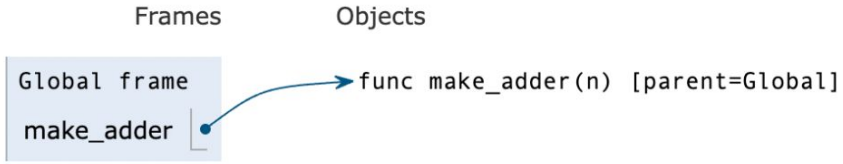
Example 3: Nested Definition



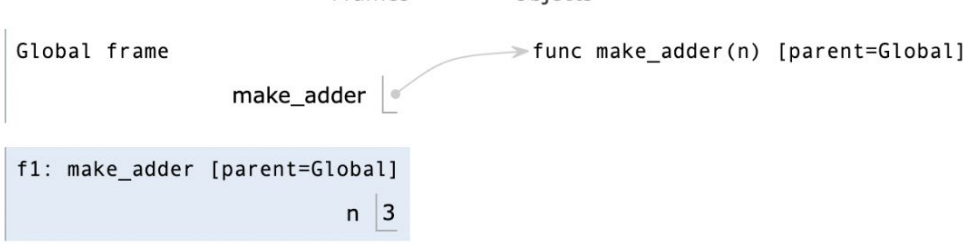
Draw

Example 3: Nested Definition

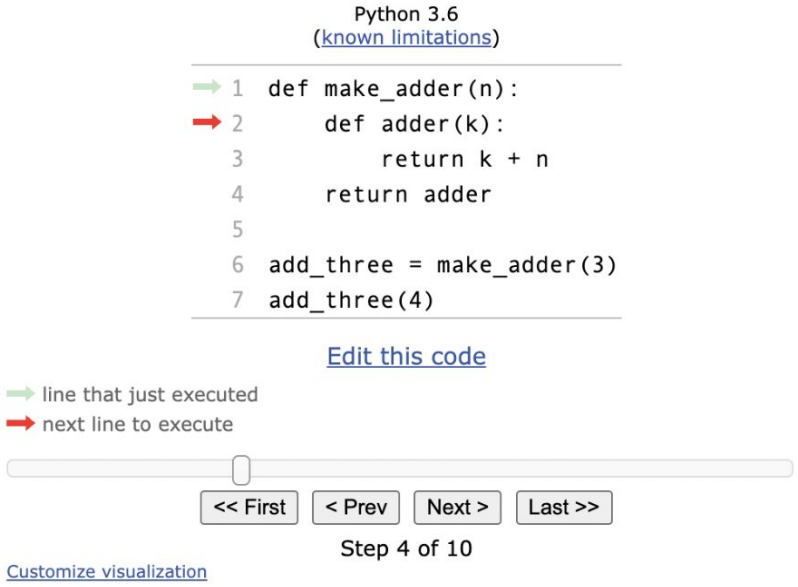


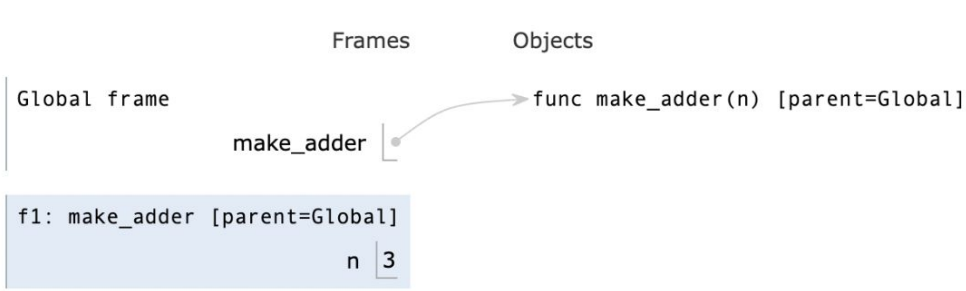


Example 3: Nested Definition

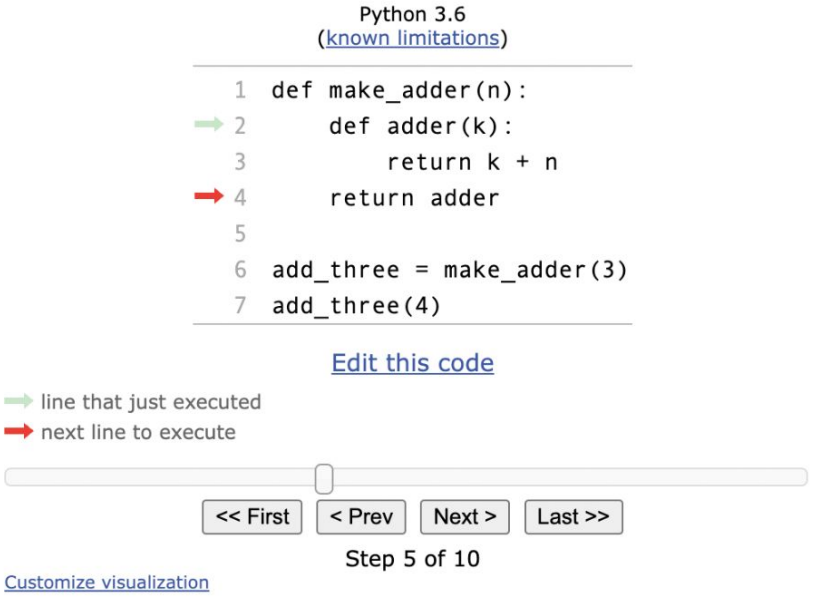


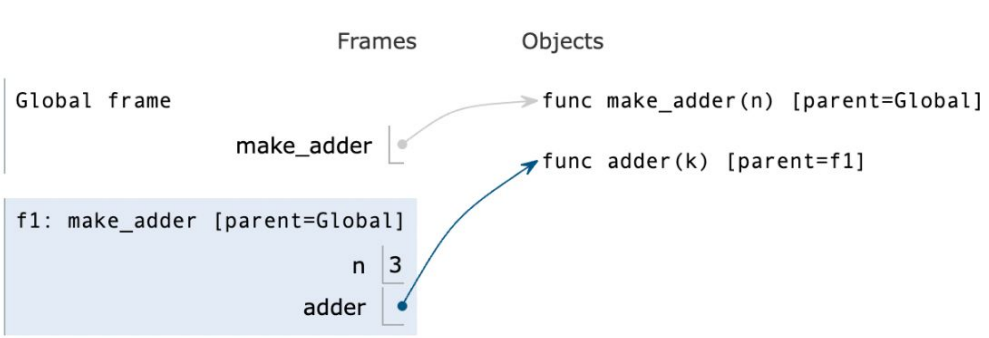
Example 3: Nested Definition



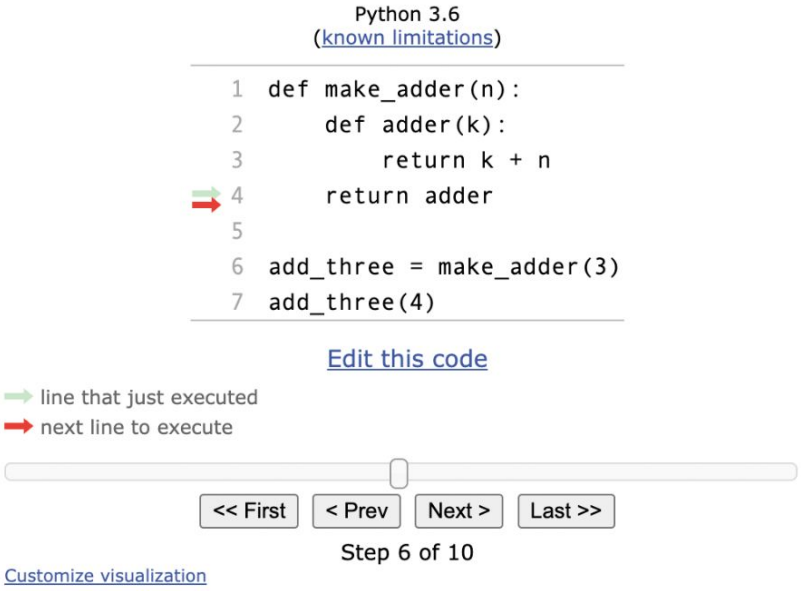


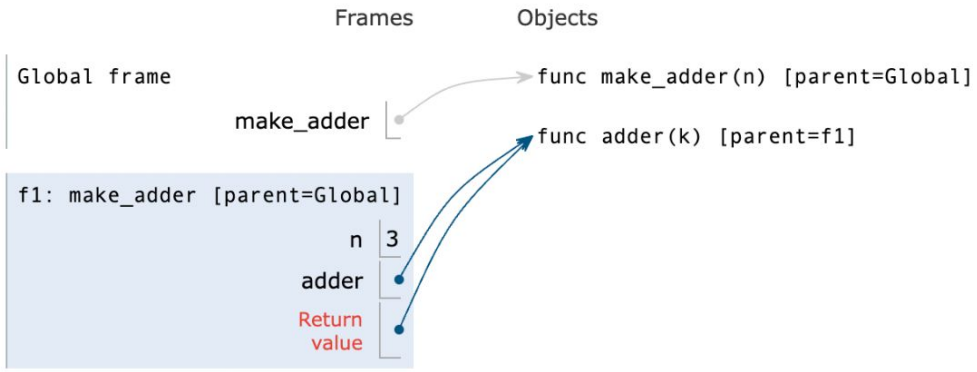
Example 3: Nested Definition





Example 3: Nested Definition





Example 3: Nested Definition

Python 3.6
(known limitations)

```
1 def make_adder(n):
2     def adder(k):
3         return k + n
4     return adder
5
6 add_three = make_adder(3)
7 add_three(4)
```

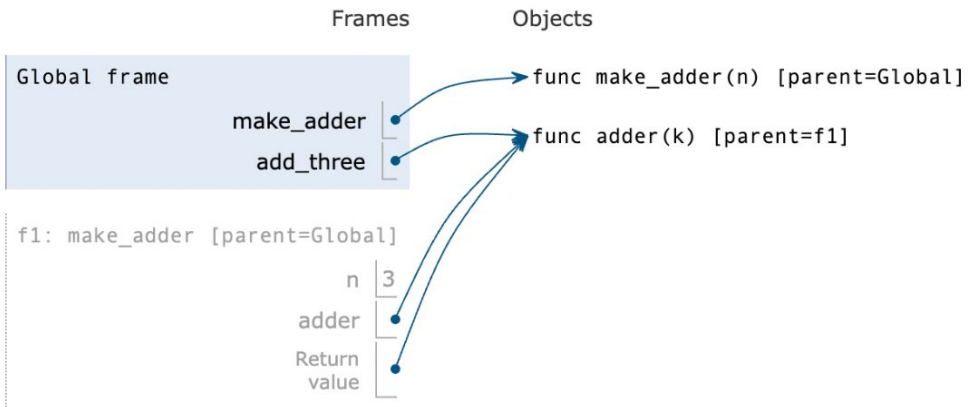
Edit this code

line that just executed
next line to execute

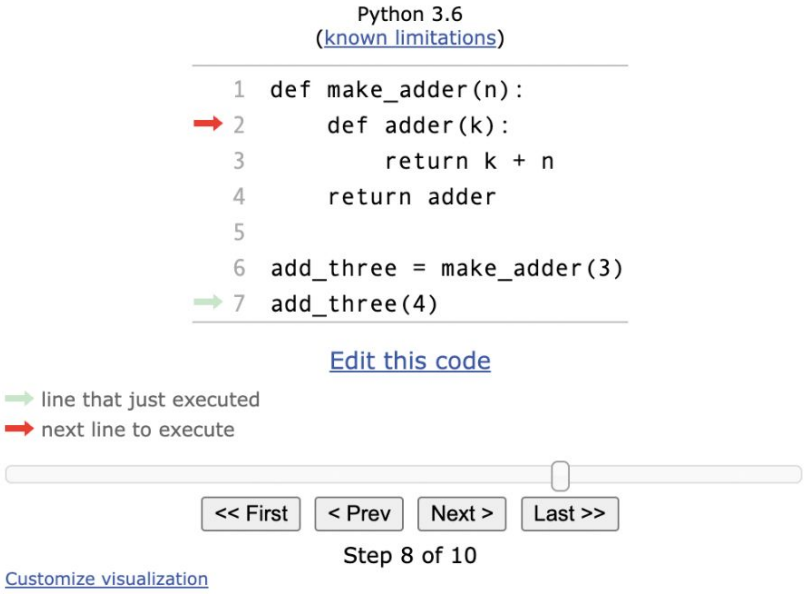
<< First | < Prev | Next > | Last >>

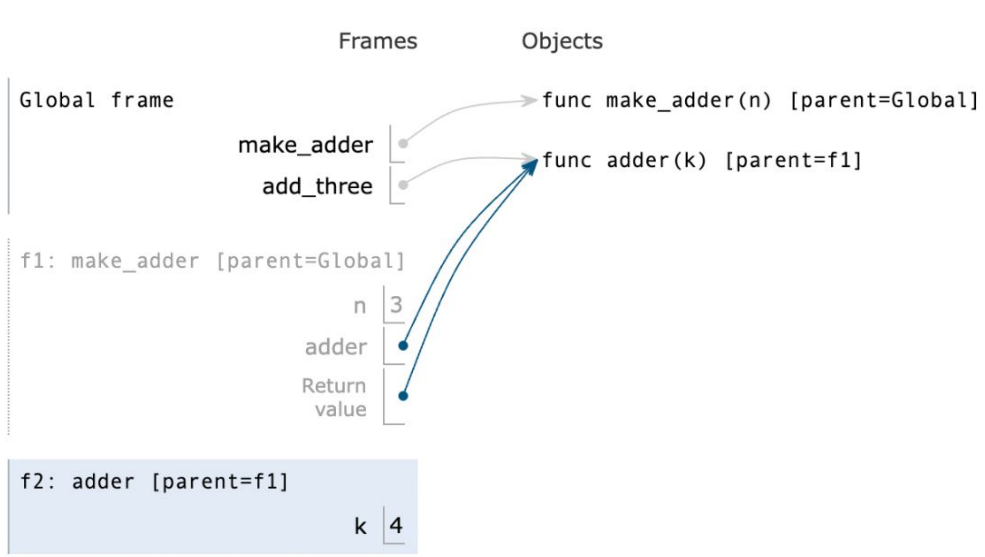
Step 7 of 10

Customize visualization

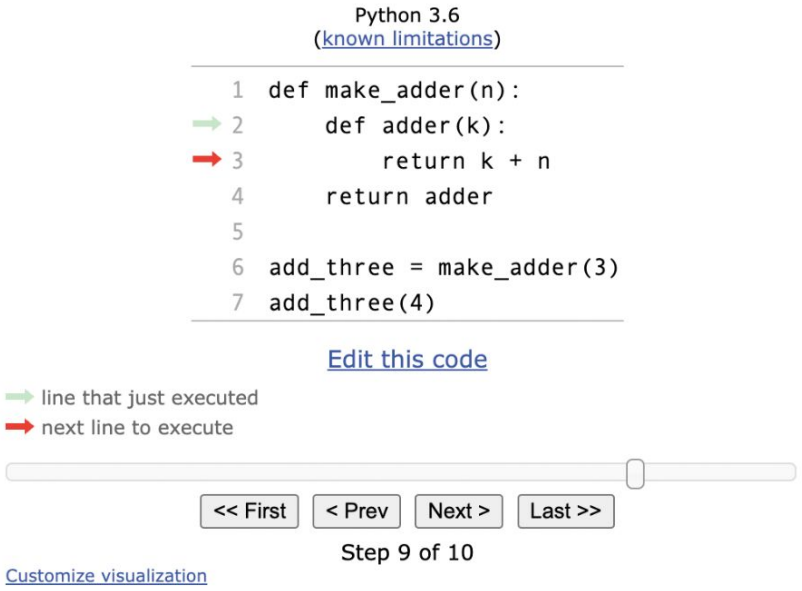


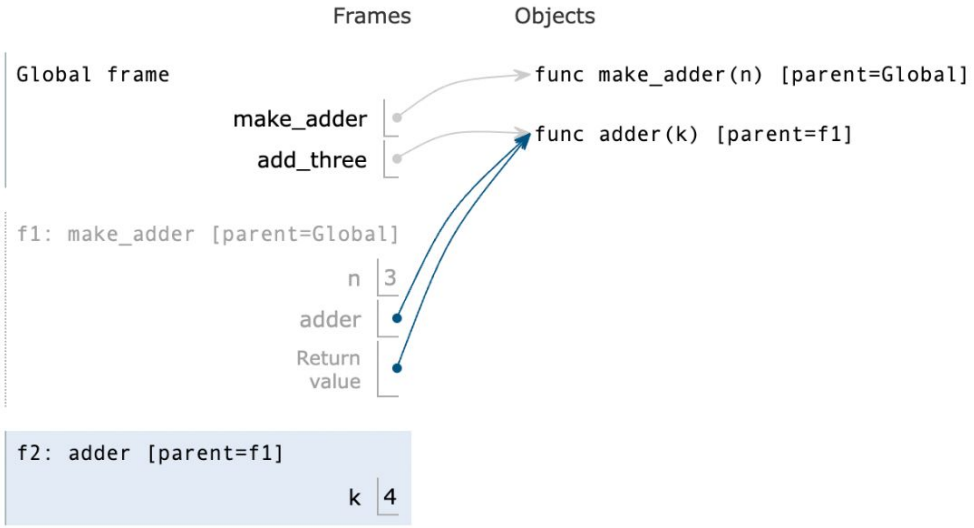
Example 3: Nested Definition



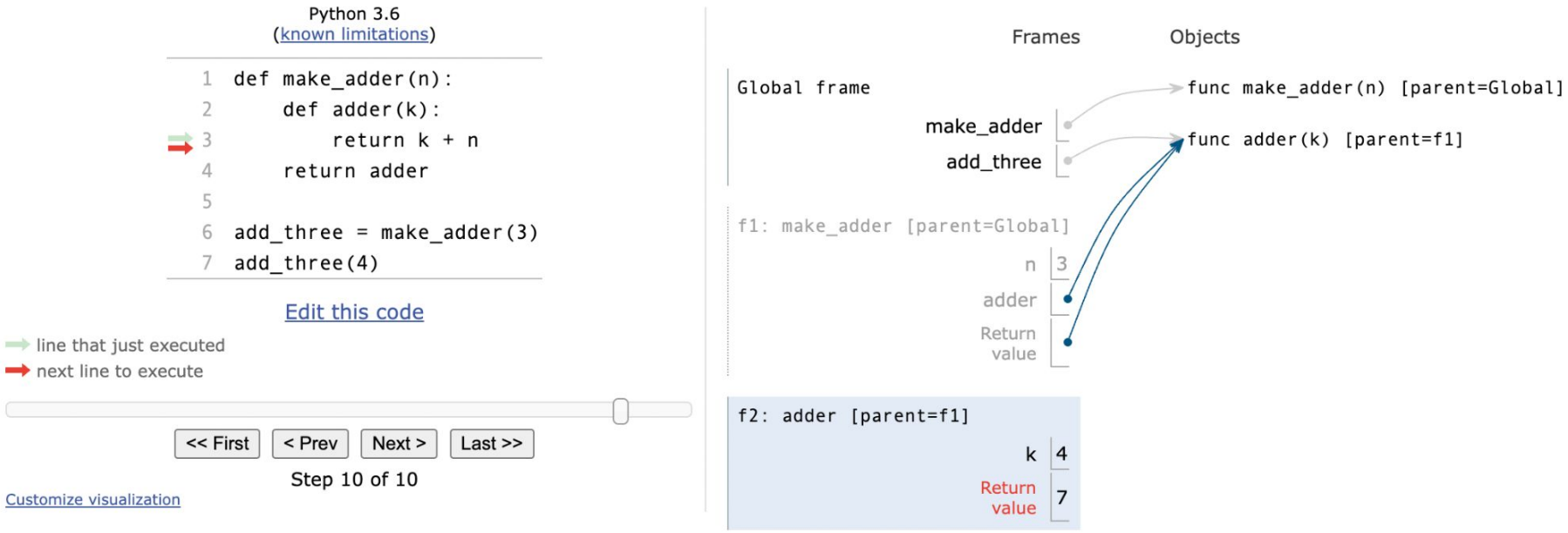


Example 3: Nested Definition





Example 3: Nested Definition



Example 3: Nested Definition

Example 4: Function Composition

[Python Tutor Link]

```
1  def make_adder(n):
2      def adder(k):
3          return k + n
4      return adder
5
6  def square(x):
7      return x * x
8
9  def triple(x):
10     return 3 * x
11
12 def compose1(f, g):
13     def h(x):
14         return f(g(x))
15     return h
```

```
>>> square(5)
25
>>> triple(5)
15
>>> squiple = compose1(square, triple)
>>> squiple(5)
225
>>> tripare = compose1(triple, square)
>>> tripare(5)
75
>>> squadder = compose1(square, make_adder(2))
>>> squadder(3)
25
>>> compose1(square, make_adder(2))(3)
25
```