Mini-Lecture 7

Function Definitions

Recall: Modules

- Modules provide extra functions, variables
  - **Example**: math provides math.cos(), math.pi
  - Access them with the `import` command
- Python provides a lot of them for us
- **This Lecture**: How to make modules
  - Atom Editor to *make* a module
  - Python to *use* the module

$\left.\begin{array}{l}\\ \\ \end{array}\right\}$ Two different programs

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Definition

- Defines what function **does**

```
def plus(n):
    return n+1
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Definition

- Defines what function **does**

Function **Header**

```
def plus(n):
    return n+1
```

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
>>>
```

Function Definition

- Defines what function **does**

Function **Header**

```
def plus(n):
    return n+1
```

Function **Body** (indented)

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

We Write Programs to Do Things

- Functions are the **key doers**

Function Call

- Command to **do** the function

```
>>> plus(23)
24
```

argument to assign to n

Function Definition

- Defines what function **does**

```
def plus(n):
    return n+1
```

Function **Header**

Function **Body** (indented)

declaration of **parameter** n

- **Parameter**: variable that is listed within the parentheses of a method header.
- **Argument**: a value to assign to the method parameter when it is called

Anatomy of a Function Definition

name

parameters

```
def plus(n):
    """Returns the number n+1

       Parameter n: number to add to
       Precondition: n is a number"""
    x = n+1
    return x
```

Function **Header**

Docstring **Specification**

Statements to execute when called

Anatomy of a Function Definition

name

parameters

```
def plus(n):
    """Returns the number n+1

       Parameter n: number to add to
       Precondition: n is a number"""
    x = n+1
    return x
```

Function **Header**

Docstring **Specification**

Statements to execute when called

The vertical line indicates indentation

Use vertical lines when you write Python on **exams** so we can see indentation

The **return** Statement

- **Format**: return <*expression*>
  - Used to evaluate ***function call*** (as an expression)
  - Also stops executing the function!
  - Any statements after a **return** are ignored
- **Example**: temperature converter function

```
def to_centigrade(x):
    """Returns: x converted to centigrade"""
    return 5*(x-32)/9.0
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text

    # Start of middle third

    # End of middle third

    # Get the text

    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text

    # Start of middle third

    # End of middle third

    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text


    # Start of middle third


    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text


    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

- Start w/ string variable
  - Holds string to work on
  - Make it the parameter
- Body is all assignments
  - Make variables as needed
  - But last line is a return
- Try to work in **reverse**
  - Start with the return
  - Figure ops you need
  - Make a variable if unsure
  - Assign on previous line

```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Defining a String Function

```
>>> middle('abc')
'b'
>>> middle('aabbcc')
'bb'
>>> middle('aaabbbccc')
'bbb'
```

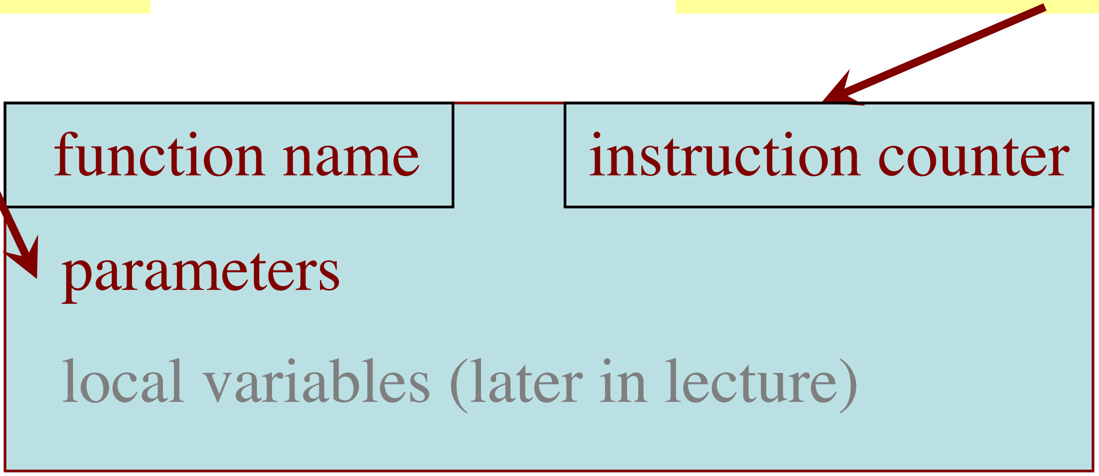
```
def middle(text):
    """Returns: middle 3rd of text
    Param text: a string"""

    # Get length of text
    size = len(text)
    # Start of middle third
    start = size//3
    # End of middle third
    end = 2*size//3
    # Get the text
    result = text[start:end]
    # Return the result
    return result
```

Understanding How Functions Work

- **Function Frame**: Representation of function call
- A **conceptual model** of Python

Draw parameters as variables (named boxes)

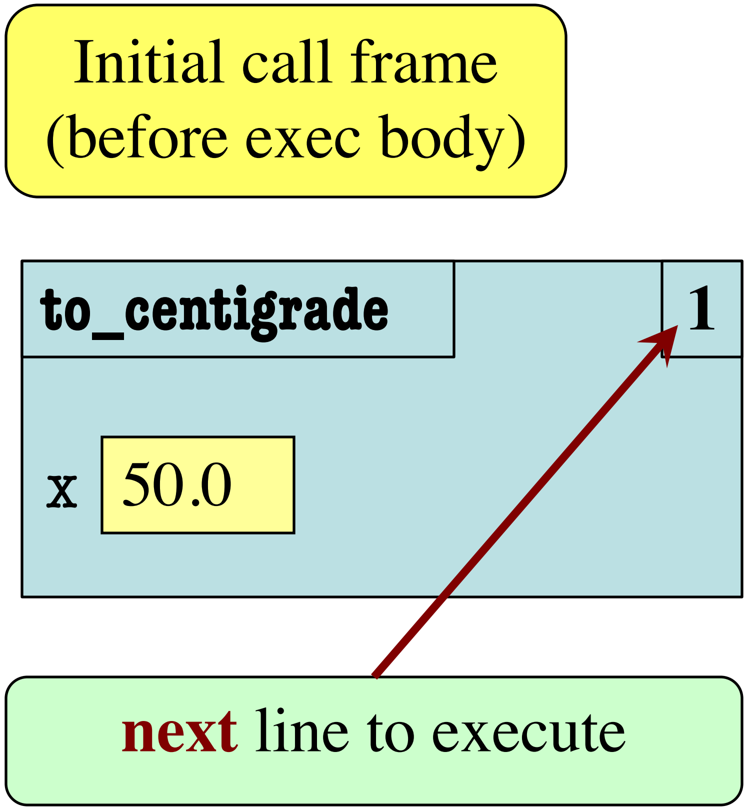
- Number of statement in the function body to execute next
- **Starts with 1**



Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
    def to_centigrade(x):
1       return 5*(x-32)/9.0
```

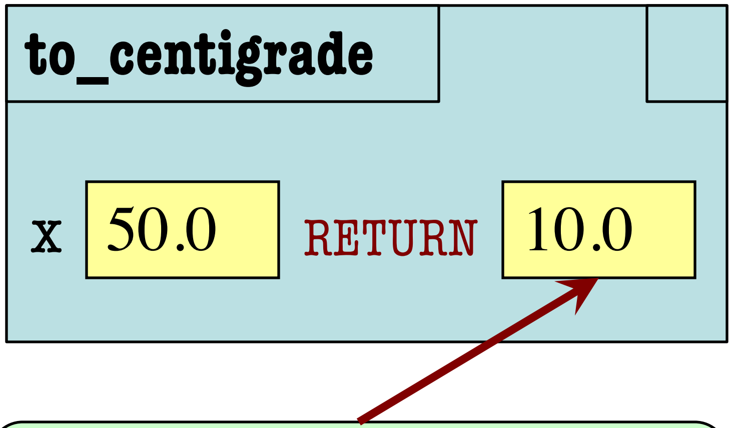


Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

Executing the return statement



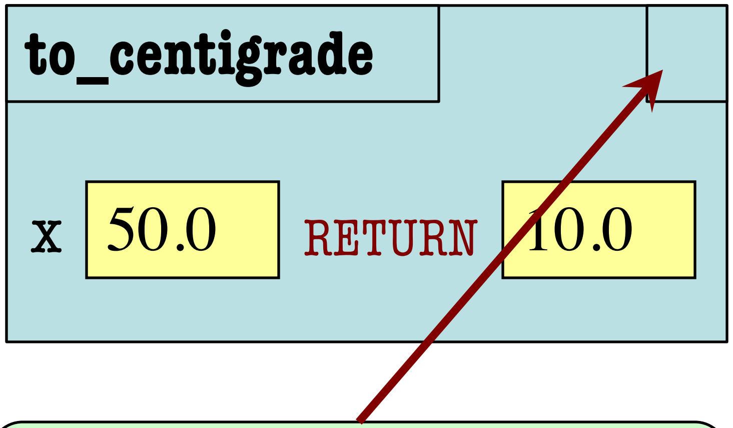
Return statement creates a special variable for result

Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):
1    return 5*(x-32)/9.0
```

Executing the return statement



The return terminates; no next line to execute

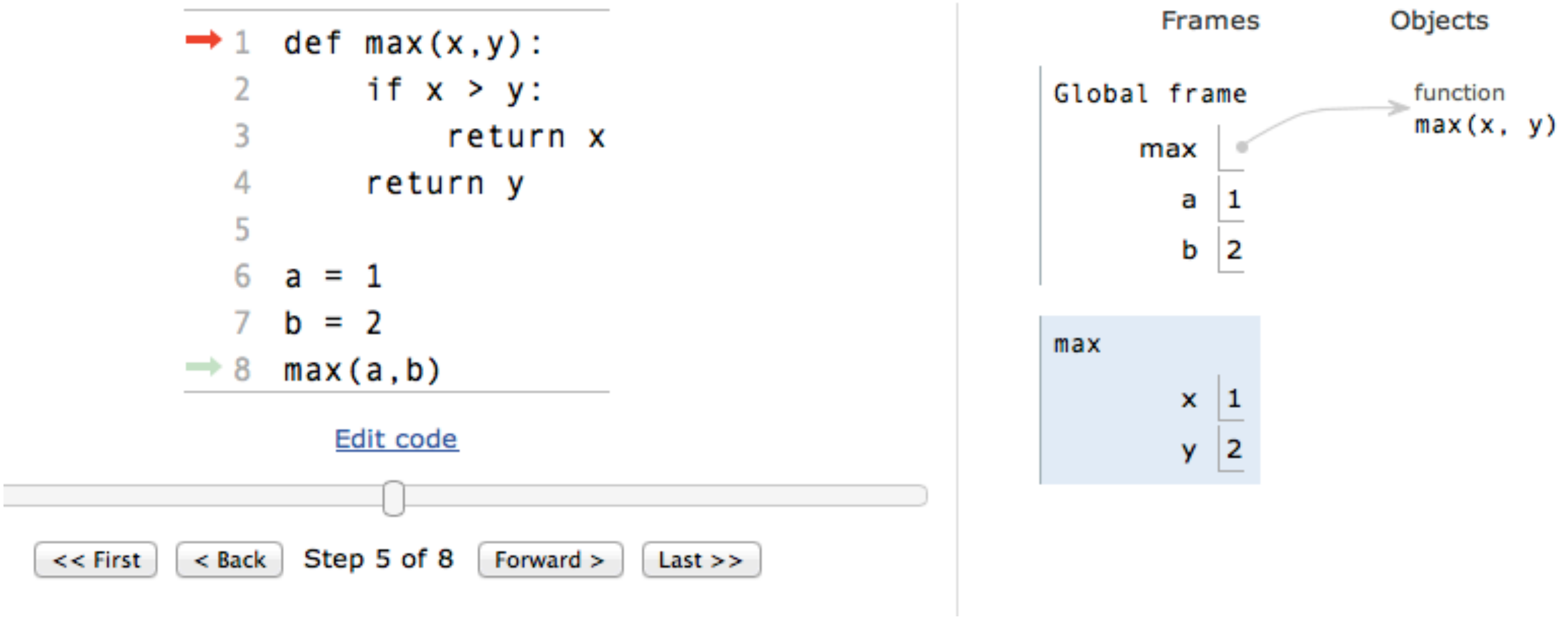
Example: to_centigrade(50.0)

1. Draw a frame for the call
2. Assign the argument value to the parameter (in frame)
3. Execute the function body
   - Look for variables in the frame
   - If not there, look for global variables with that name
4. Erase the frame for the call

```
def to_centigrade(x):
    return 5*(x-32)/9.0
```

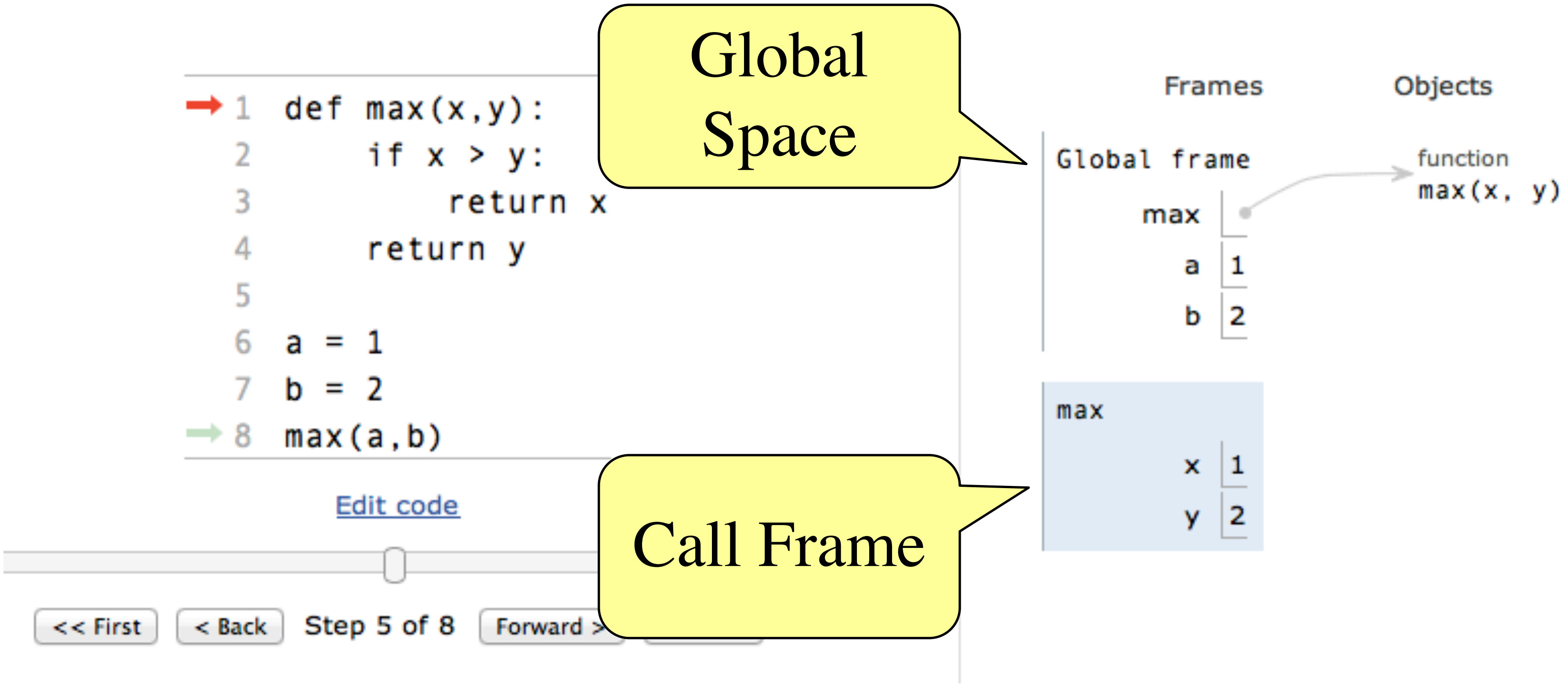
ERASE WHOLE FRAME

Visualizing Frames: The Python Tutor



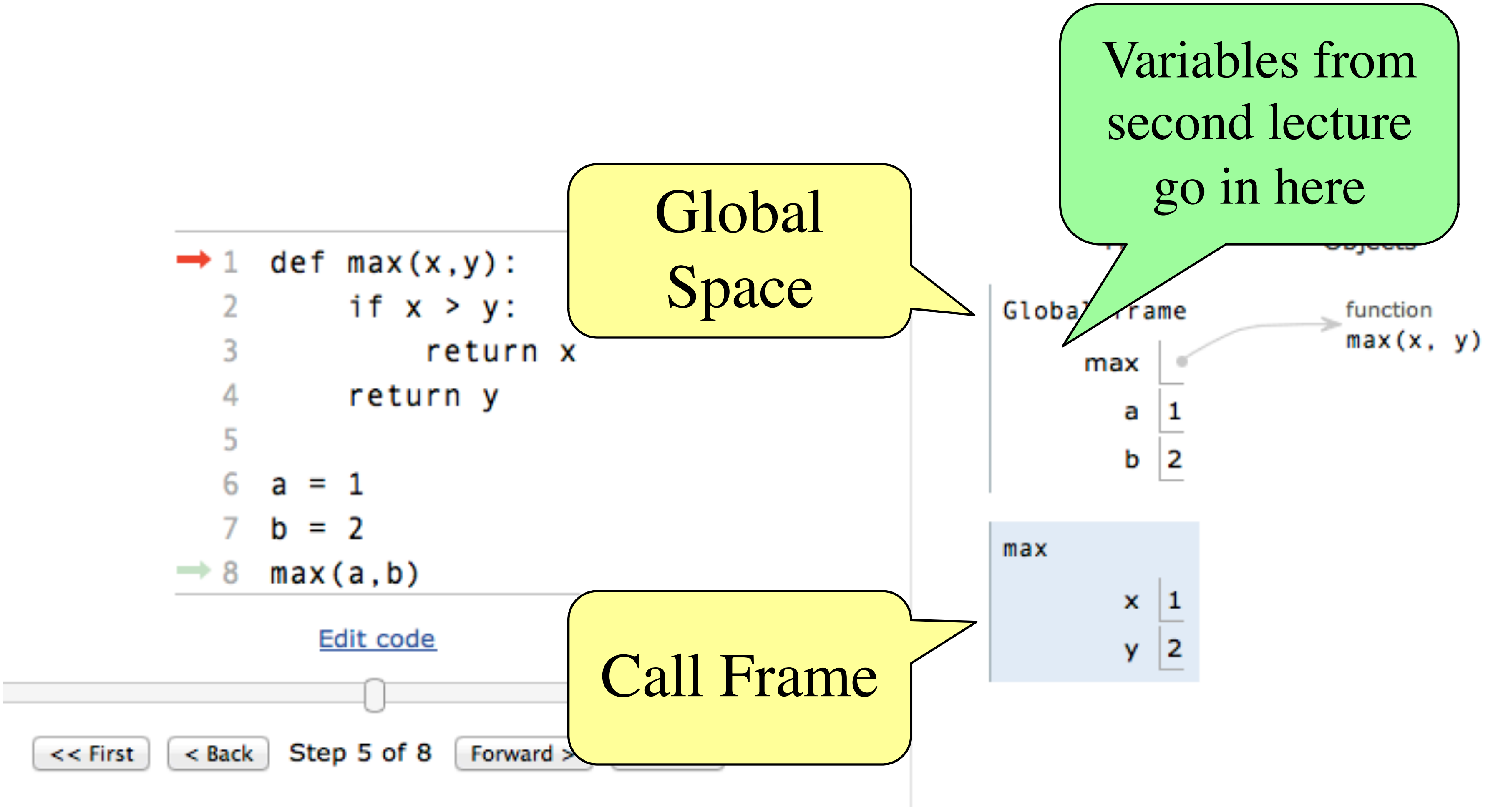
```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Visualizing Frames: The Python Tutor



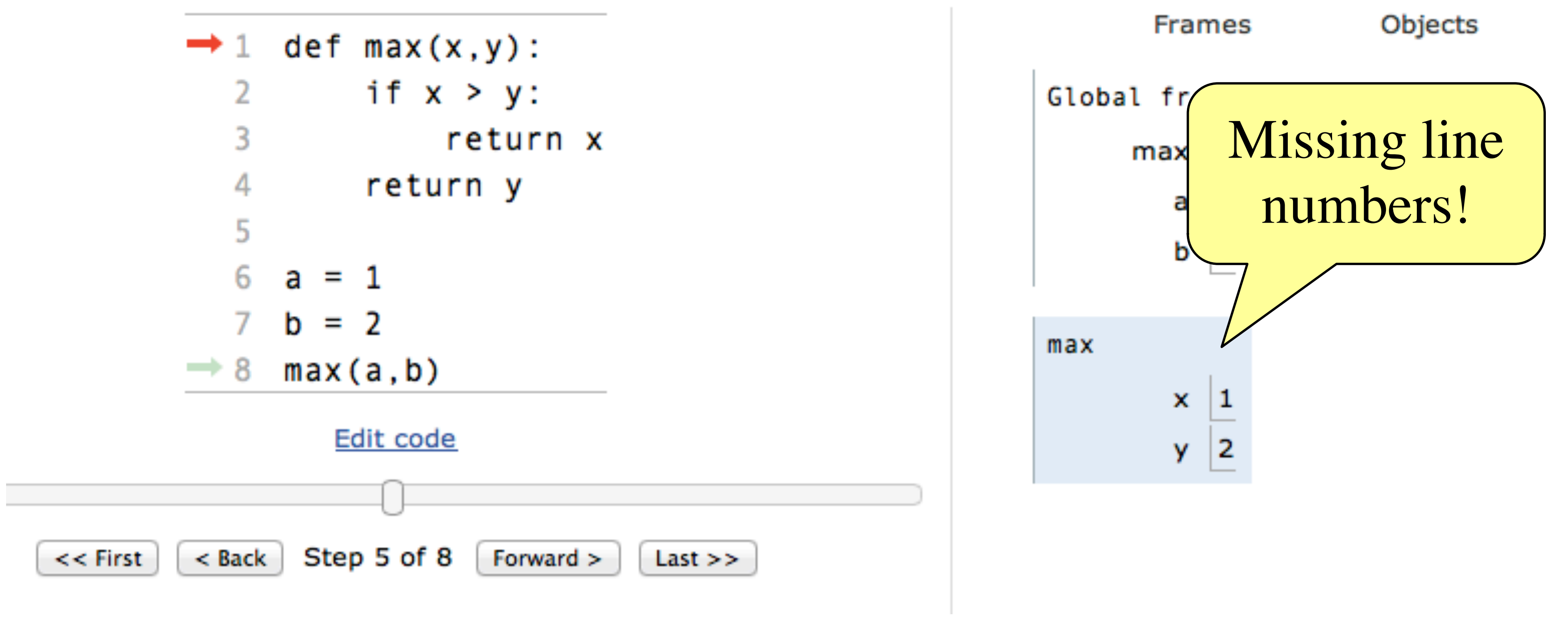
```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Visualizing Frames: The Python Tutor



```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Visualizing Frames: The Python Tutor

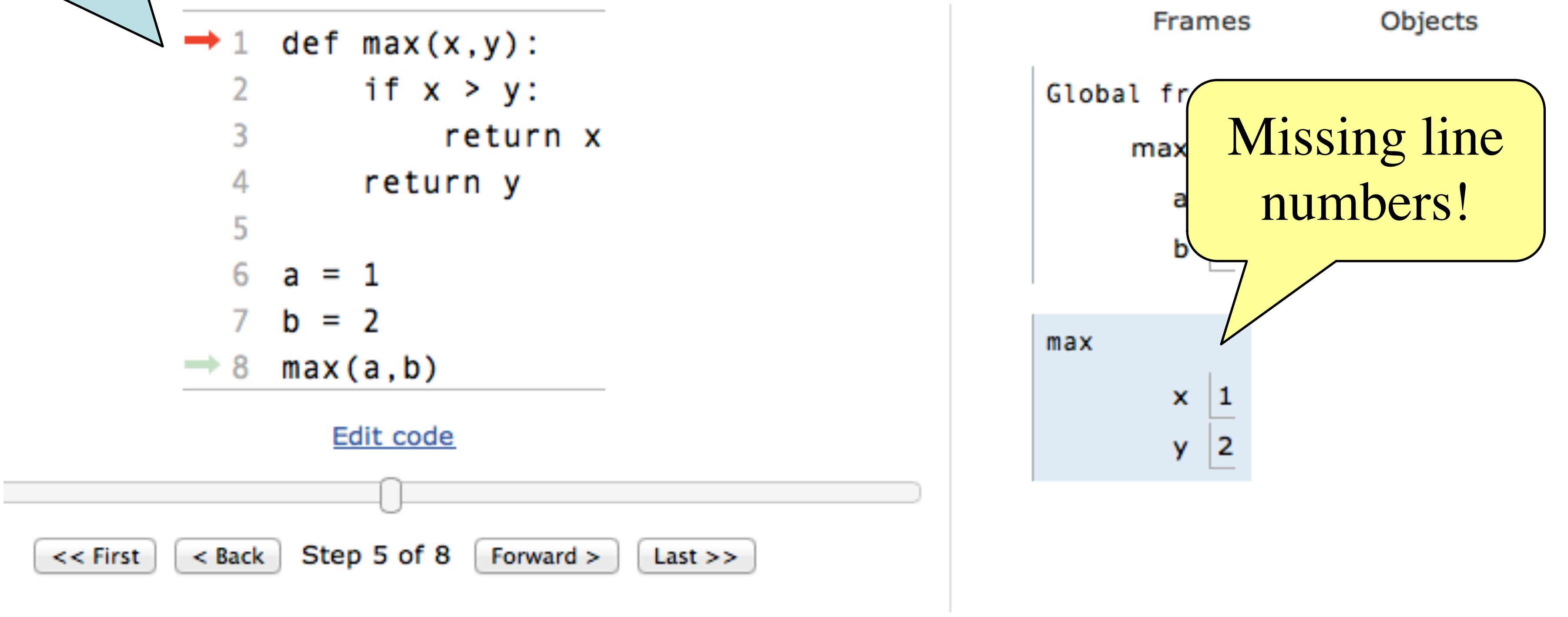


```
1  def max(x,y):
2      if x > y:
3          return x
4      return y
5
6  a = 1
7  b = 2
8  max(a,b)
```

Missing line numbers!

Visualizing Frames: The Python Tutor

Line number marked here (sort-of)



```
def max(x,y):
    if x > y:
        return x
    return y

a = 1
b = 2
max(a,b)
```

Missing line numbers!