CS 61A
DISCUSSION 1

CONTROL, ENVIRONMENTS, AND HIGHER ORDER FUNCTIONS

Raymond Chan
Discussion 134
UC Berkeley Fall 16

AGENDA

- Announcements
- If statements
- Boolean operators
- While loops
- Environment Diagrams

ANNOUNCEMENTS

- Lab 01 due Friday
- Homework 01 due Tonight
- Project 1 Hog is released. Due Thu 9/8

IF STATEMENTS

- Execute different code based on different conditions
- Evaluates conditional expressions with the **bool** function to True or False values
  - Boolean operators, comparing numbers (3 == 5; x > 3)

```
if <conditional expr>:
   <suite of statements>
elif <conditional expr>:
   <suite of statements>
                else:
   <suite of statements>
      <rest of code>
```

IF STATEMENTS

- The suite that is indented under the first if/elif with a True conditional is evaluated.
- If all the conditionals fail, the suite under else is evaluated.
- There is only one else clause.

```
if <conditional expr>:
    <suite of statements>
elif <conditional expr>:
    <suite of statements>
                else:
    <suite of statements>
        <rest of code>
```

IF STATEMENTS

- Execute different code based on different conditions

```
if <conditional>:
<suite of statements>
 elif <conditional>:
<suite of statements>
        else:
<suite of statements>
   <rest of code>
```

```
if <conditional>:
<suite of statements>
   if <conditional>:
<suite of statements>
   if <conditional>:
<suite of statements>
   <rest of code>
```

BOOLEAN OPERATORS

- not: returns the opposite
  - always returns True or False
- and: evaluates and returns the first False expression
  - if all True -> evaluates and returns the last expression
- or: evaluates and returns the first True expression
  - if all False -> evaluates and returns the last expression

BOOLEAN OPERATORS

- and/or uses the bool(x) function to determine True/False values
- do not have to return True/False
- False values: False, 0, None, “”, []...
- True values: True, non-zero integers, almost everything else

WHILE LOOPS

- As long as the conditional evaluates to True, the body is executed
- Watch out for infinite loops!
- Within the body, the conditional needs to change after each iteration

```
while <conditional>:
        <body>
```

```
i = 0
while i < n:
        <body>
        ...
        i = i + 1
```

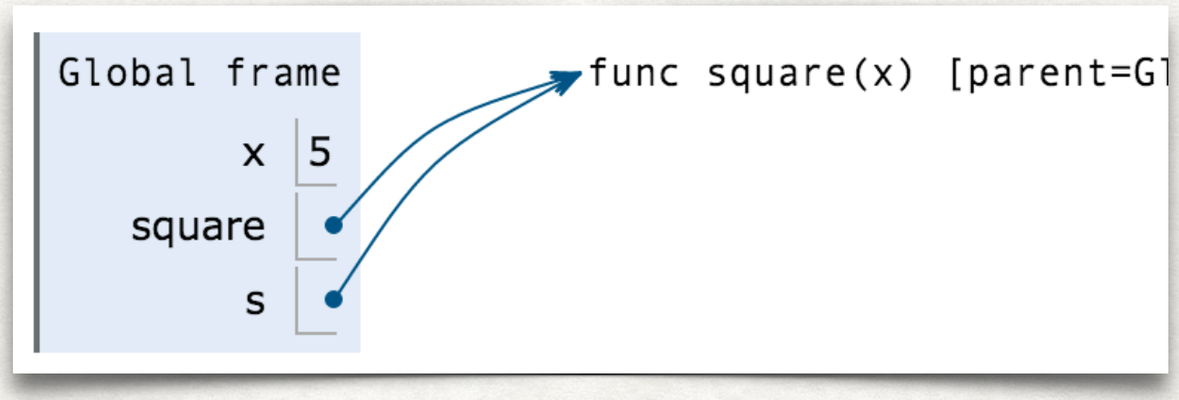
ENVIRONMENT DIAGRAMS

- Environment diagrams allow us to keep track of variables that have been defined and the values they are bound to.
- Visualization of the execution of Python code.
- Assignment Statements
- Def Statements
- Function Calls

ASSIGNMENT STATEMENTS

- Evaluate the expression on the right hand side of the = sign.
  - Look up names in the current frame. If it does not exist, look up in the parent frame.
- Evaluate primitive expressions and operations

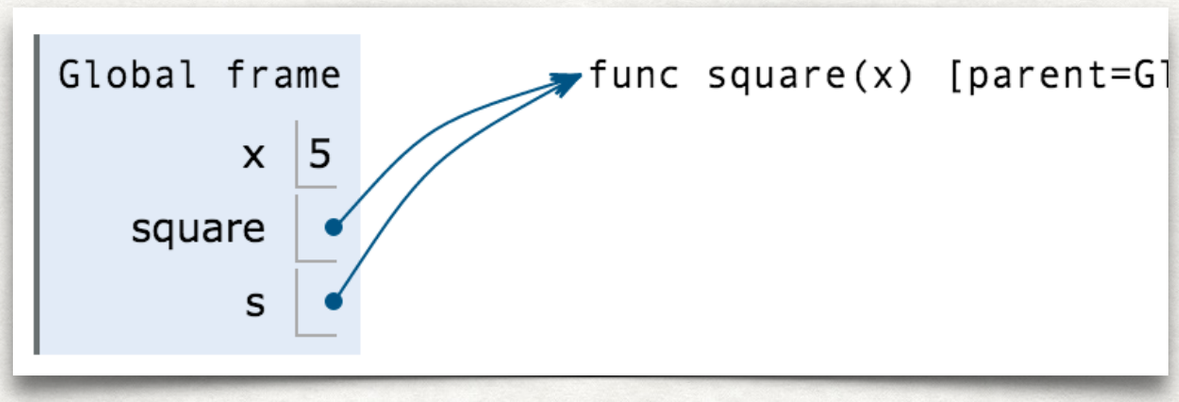
```
1  x = 5
2  def square(x):
3      return x**2
4
5  s = square
```



ASSIGNMENT STATEMENTS

- If the variable name on the left hand side of '=' does not exist, create it in the current local frame.
- Write the expression value next to the variable name.
- If the variable already exists, cross out and replace the current value with the evaluated value.

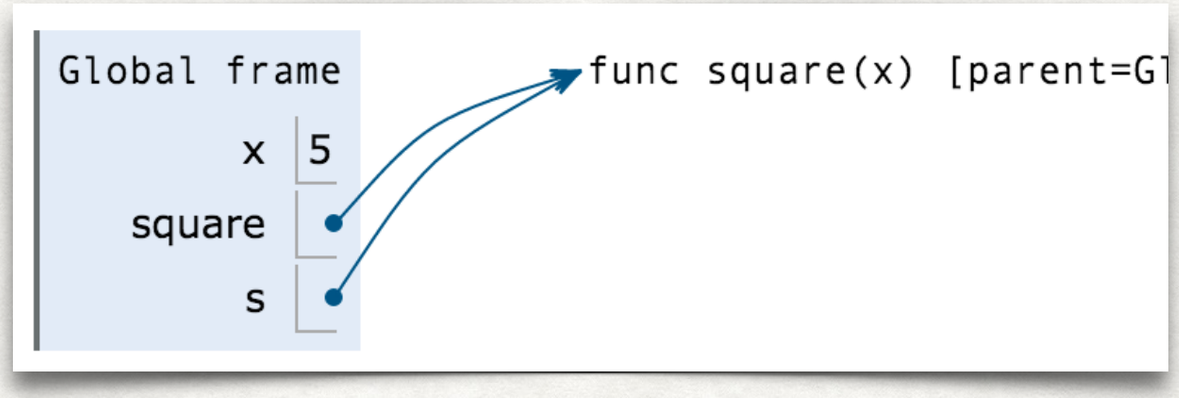
```
1  x = 5
2  def square(x):
3      return x**2
4
5  s = square
```



ASSIGNMENT STATEMENTS

- Variable assignments on the right hand side **only** checks for variables in the **local** frame.
- If the expression is a function, draw a reference arrow from the variable name to the function object.

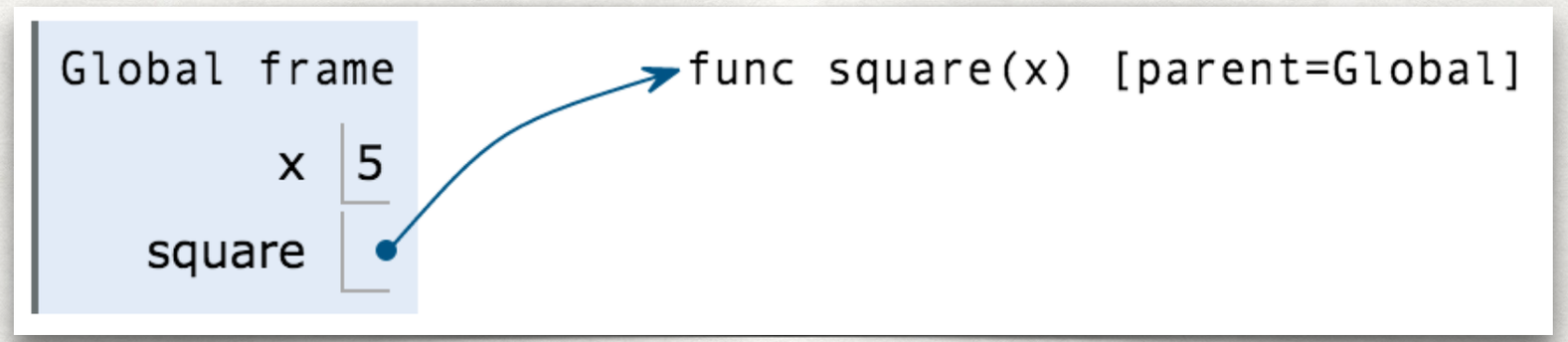
```
1  x = 5
2  def square(x):
3      return x**2
4
5  s = square
```



DEF STATEMENTS

- Create the function object with the function signature and parent frame.
- The parent frame is the frame in which the frame is defined.

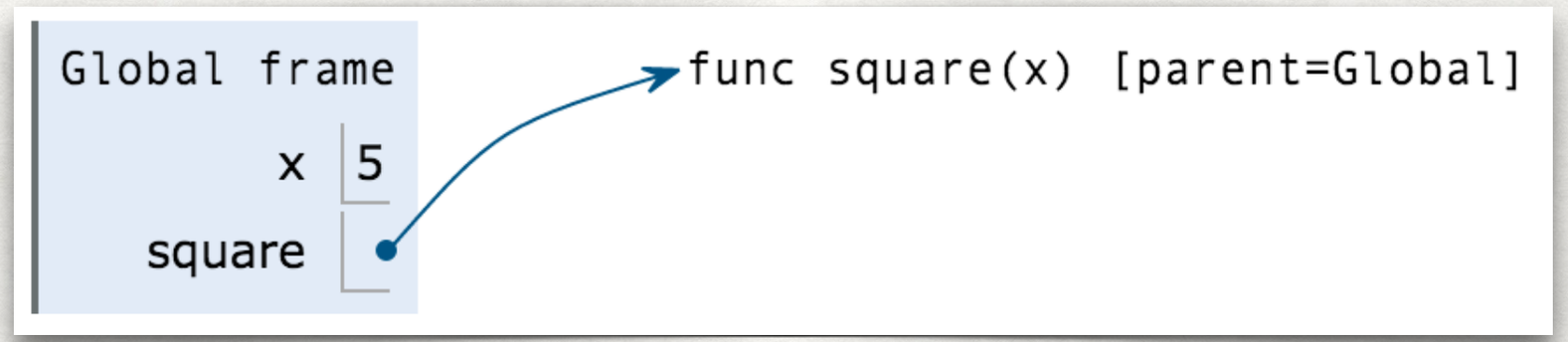
```
1  x = 5
2  def square(x):
3      return x**2
4
```



DEF STATEMENTS

- Use a reference arrow to bind the function name to the function object.
- **Do not** evaluate the body of the function at this time.

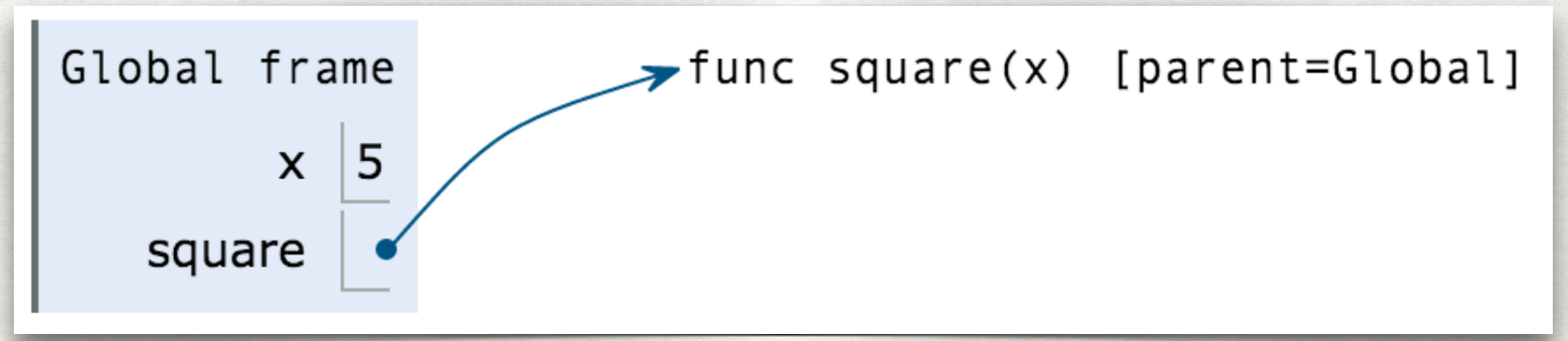
```
1  x = 5
2  def square(x):
3      return x**2
4
```



DEF STATEMENTS

- Function signature contains the function's intrinsic name and the formal parameters.

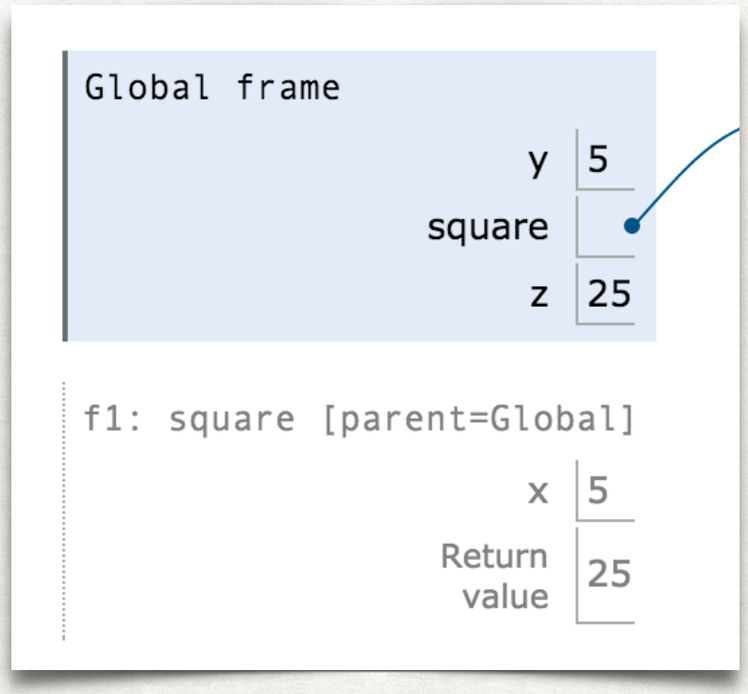
```
1  x = 5
2  def square(x):
3      return x**2
4
```



CALL EXPRESSIONS

- Evaluate the operator.
- Evaluate the operands from left to right.

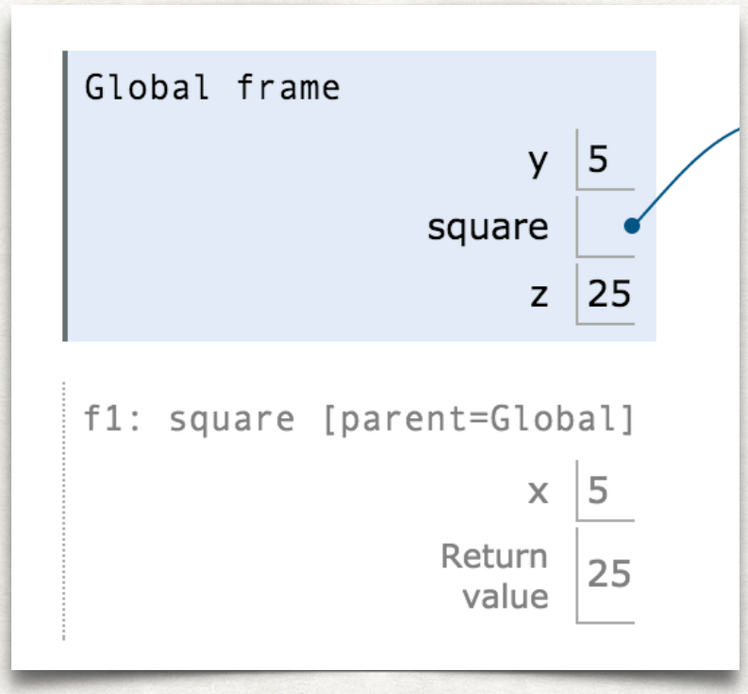
```
1  y = 5
2  def square(x):
3      return x**2
4
5  z = square(y)
```



CALL EXPRESSIONS

- Apply the evaluated operands on the operator.
- Create a new frame.

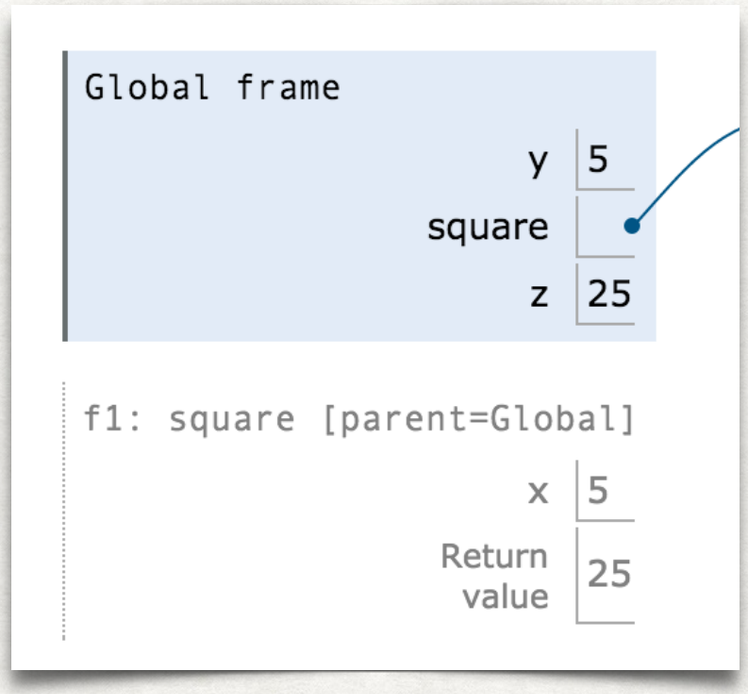
```
1  y = 5
2  def square(x):
3      return x**2
4
5  z = square(y)
```



CALL EXPRESSIONS

- Apply the evaluated operands on the operator.
- Draw a new frame with a unique frame index, the function's intrinsic name, and the parent frame.

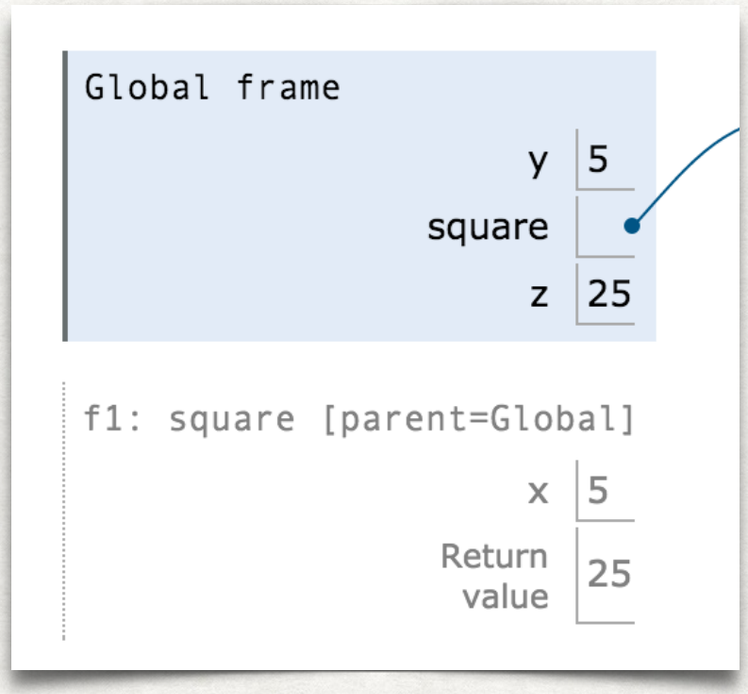
```
1  y = 5
2  def square(x):
3      return x**2
4
5  z = square(y)
```



CALL EXPRESSIONS

- Bind the formal parameters to the argument(s) passed in.
- Evaluate the body of the function.

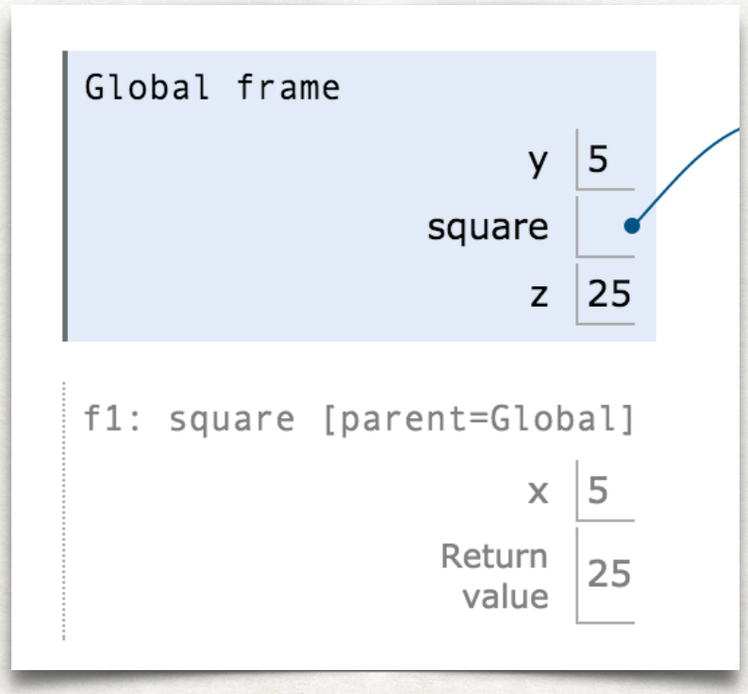
```
1  y = 5
2  def square(x):
3      return x**2
4
5  z = square(y)
```



CALL EXPRESSIONS

- Remember to denote the return value. If a function does not return anything, the return value is by default **None**.
- If we are assigning a variable to a call expression, assign the return value to the variable in the frame of the call expression.

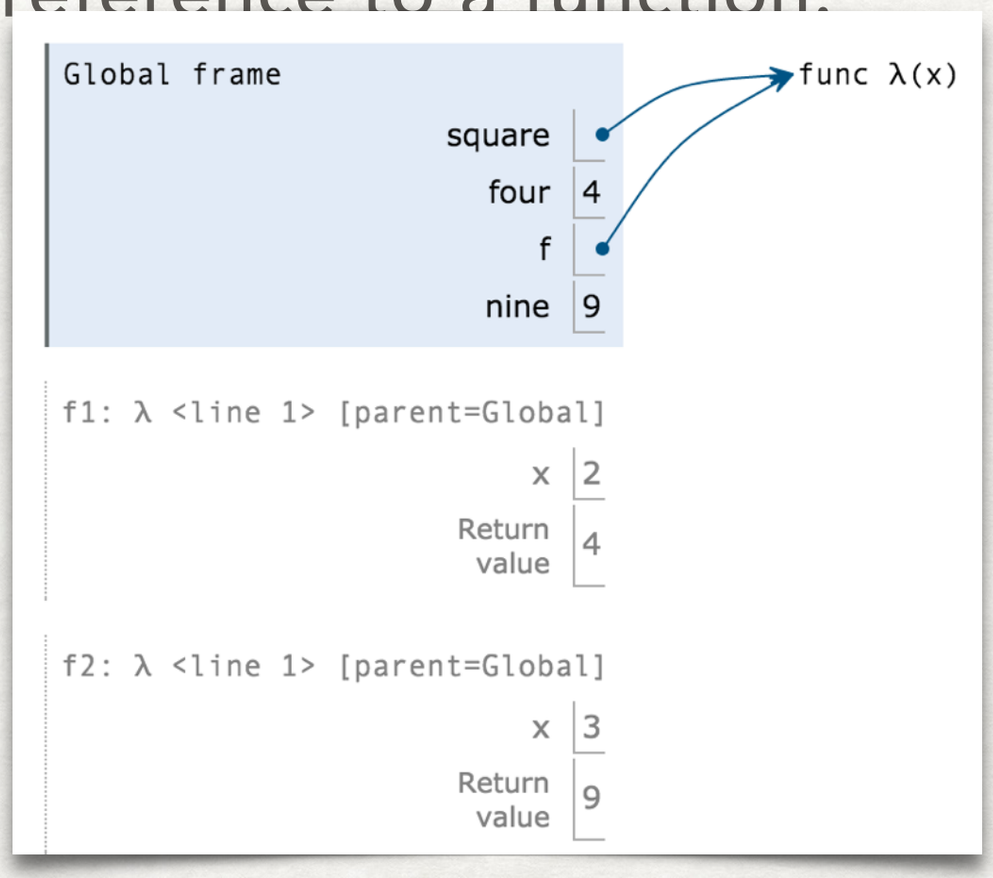
```
1  y = 5
2  def square(x):
3      return x**2
4
5  z = square(y)
```



FUNCTION CALL VS. FUNCTION

- Variables can be assigned to the return value of a function call or to a function object itself.
- Variables are assigned to the result of evaluating the right hand side, which could be a reference to a function.

```
1  square = lambda x: x * x
2  four = square(2)
3  f = square
4  nine = f(3)
```



HIGHER ORDER FUNCTIONS

- Function that manipulates other functions by:
  - Taking functions as arguments,
  - Returning a function, or
  - Both.

HIGHER ORDER FUNCTIONS

- Function arguments can be other functions.
  - Pass in the name of the function.
  - Don't make a function call.

```
def square(x):
    return x * x

def negate(f, x):
    return -f(x)
```

```
>>> negate(square, 5)
25
```

HIGHER ORDER FUNCTIONS

- Functions can also return other functions
  - Return function name.
  - Or can be a function call that returns a function.

```
def f(x):
    z = 6
    def g(y):
        return x * y + 6
    return g

>>> f(5)(5)
25
```

```
def h(z):
    return f(z)

>>> h(10)(5)
50
```