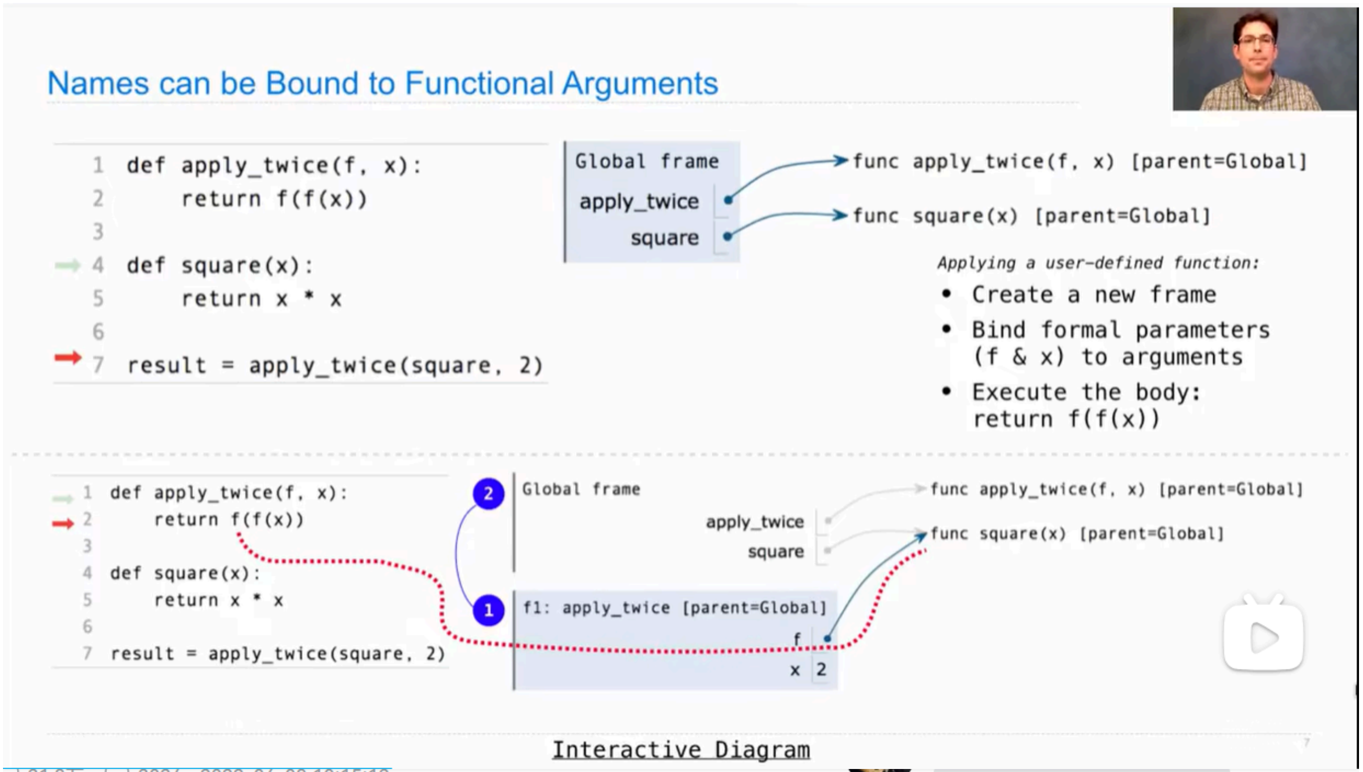
```
today we investigate how the mechanics of higher order functions can be
expressed and tracked using environment diagrams
the aim? is to understand how higher order functions really work.
```



Names can be Bound to Functional Arguments

```
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

```
Global frame
    apply_twice  -> func apply_twice(f, x) [parent=Global]
    square       -> func square(x) [parent=Global]
```

Applying a user-defined function:

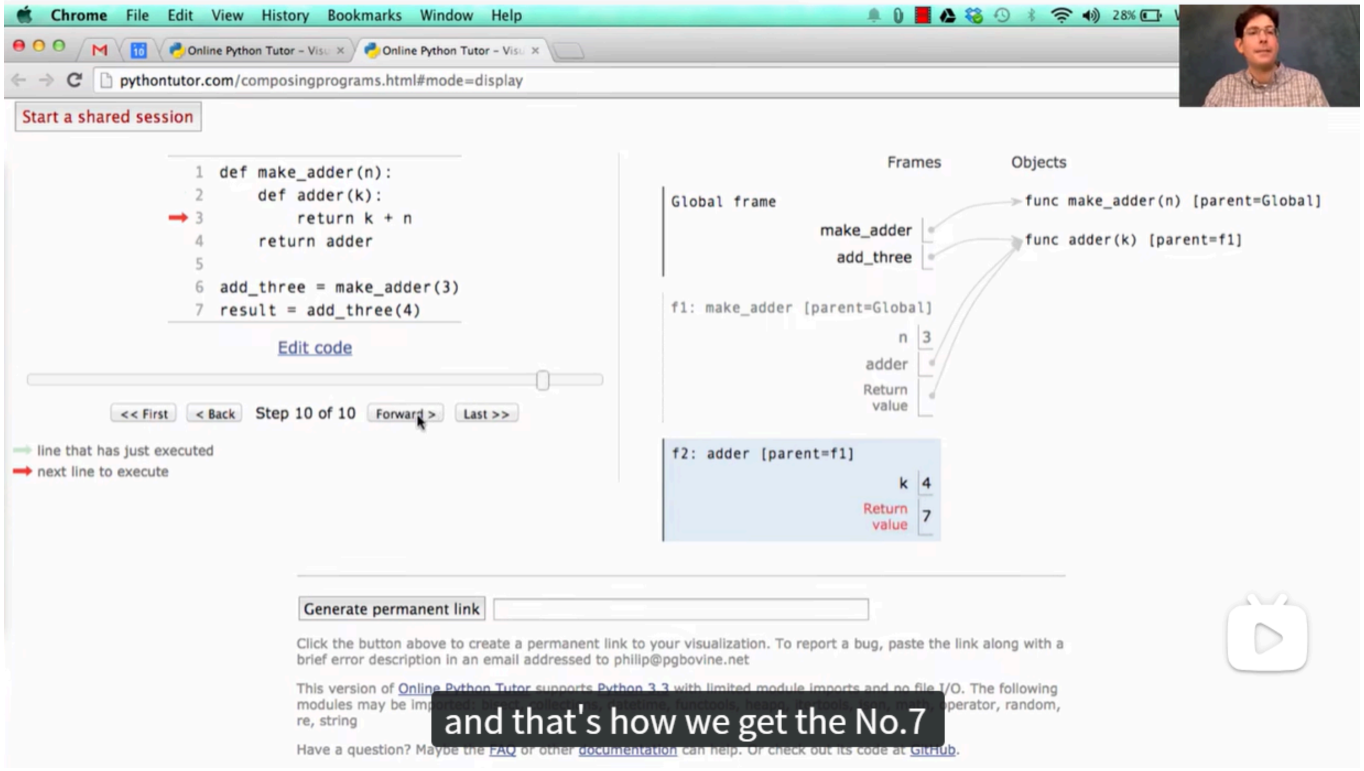
- Create a new frame
- Bind formal parameters (f & x) to arguments
- Execute the body: return f(f(x))

```
def apply_twice(f, x):
    return f(f(x))

def square(x):
    return x * x

result = apply_twice(square, 2)
```

```
Global frame
    apply_twice  -> func apply_twice(f, x) [parent=Global]
    square       -> func square(x) [parent=Global]

f1: apply_twice [parent=Global]
    f  -> func square(x) [parent=Global]
    x  2
```

Interactive Diagram

31.8万 3024 2022-06-08 10:15:13 已关注 4792



Chrome File Edit View History Bookmarks Window Help
Online Python Tutor - Visu
Online Python Tutor - Visu
pythontutor.com/composingprograms.html#mode=display
Start a shared session

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
result = add_three(4)
```

Edit code
<< First
< Back
Step 10 of 10
Forward >
Last >>

line that has just executed
next line to execute

```
Frames                          Objects
Global frame
    make_adder   -> func make_adder(n) [parent=Global]
    add_three    -> func adder(k) [parent=f1]

f1: make_adder [parent=Global]
    n  3
    adder  -> func adder(k) [parent=f1]
    Return value  -> func adder(k) [parent=f1]

f2: adder [parent=f1]
    k  4
    Return value  7
```

Click the button above to create a permanent link to your visualization. To report a bug, paste the link along with a brief error description in an email addressed to philip@pgbovine.net

This version of Online Python Tutor supports Python 3.3 with limited module imports and no file I/O. The following modules may be imported: bisect, collections, datetime, functools, heapq, itertools, json, math, operator, random, re, string

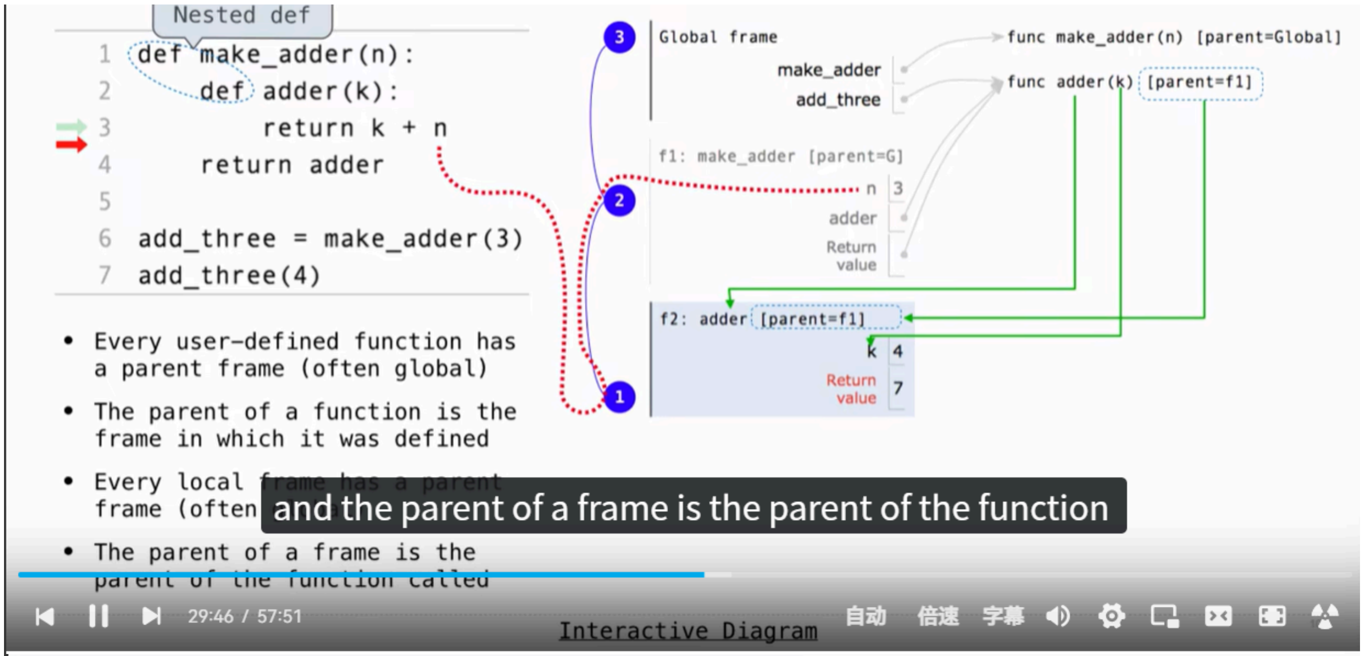
Have a question? Maybe the FAQ or other documentation can help. Or check out its code at GitHub.

and that's how we get the No.7

in python, the current frame you created a function would be the parent frame of the function.

like here, the body of the make adder is the place where adder was created, so f1 becomes its parent frame.

besides the parameters and names, we also bind down the parent environment into the new frame(created by calling the function). knowing this helps the computer where to find the names and variables if it couldn't find the things it need in the current frame.



Nested def

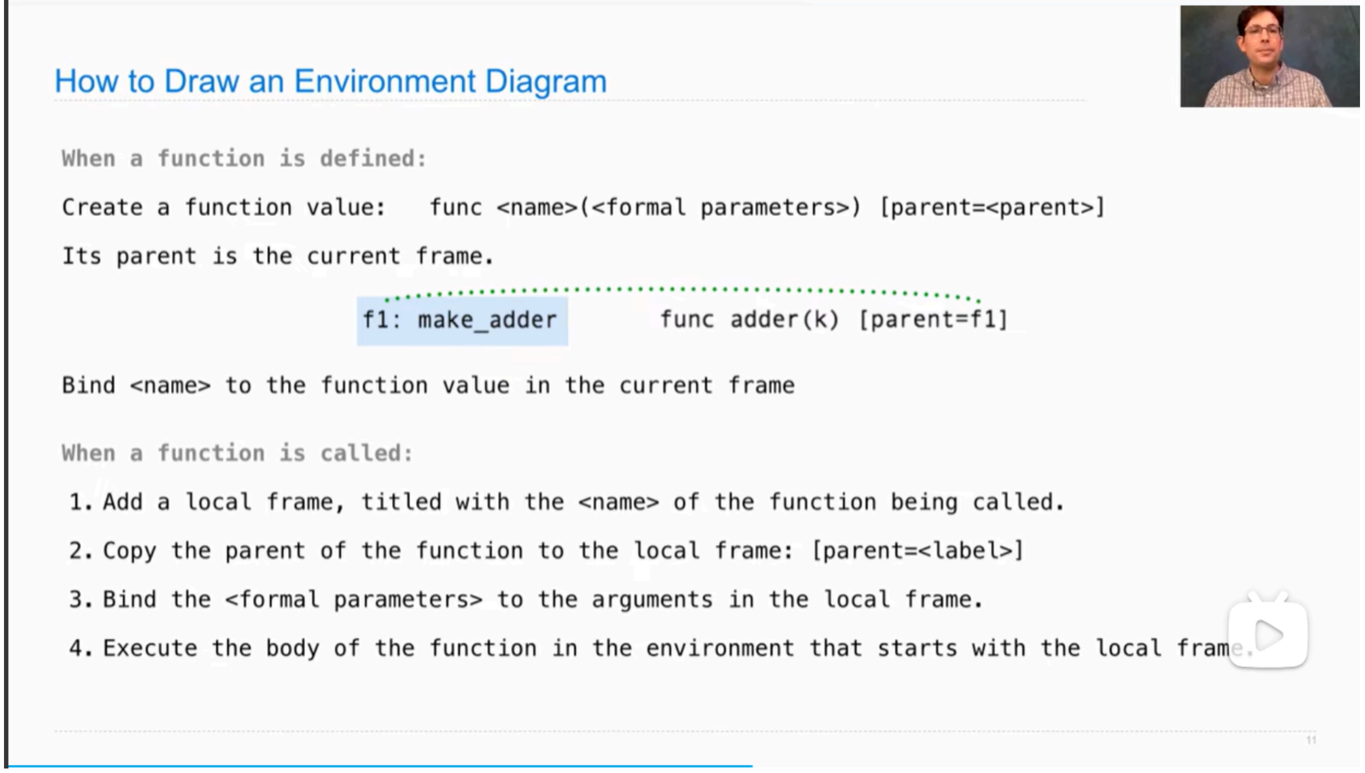
```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_three = make_adder(3)
add_three(4)
```

- Every user-defined function has a parent frame (often global)
- The parent of a function is the frame in which it was defined
- Every local frame has a parent frame (often
- The parent of a frame is the parent of the function called

and the parent of a frame is the parent of the function

```
Global frame
        make_adder        func make_adder(n) [parent=Global]
        add_three         func adder(k) [parent=f1]

f1: make_adder [parent=G]
        n       3
        adder
        Return value

f2: adder [parent=f1]
        k       4
        Return value  7
```

Interactive Diagram



How to Draw an Environment Diagram

When a function is defined:

Create a function value: func <name>(<formal parameters>) [parent=<parent>]

Its parent is the current frame.

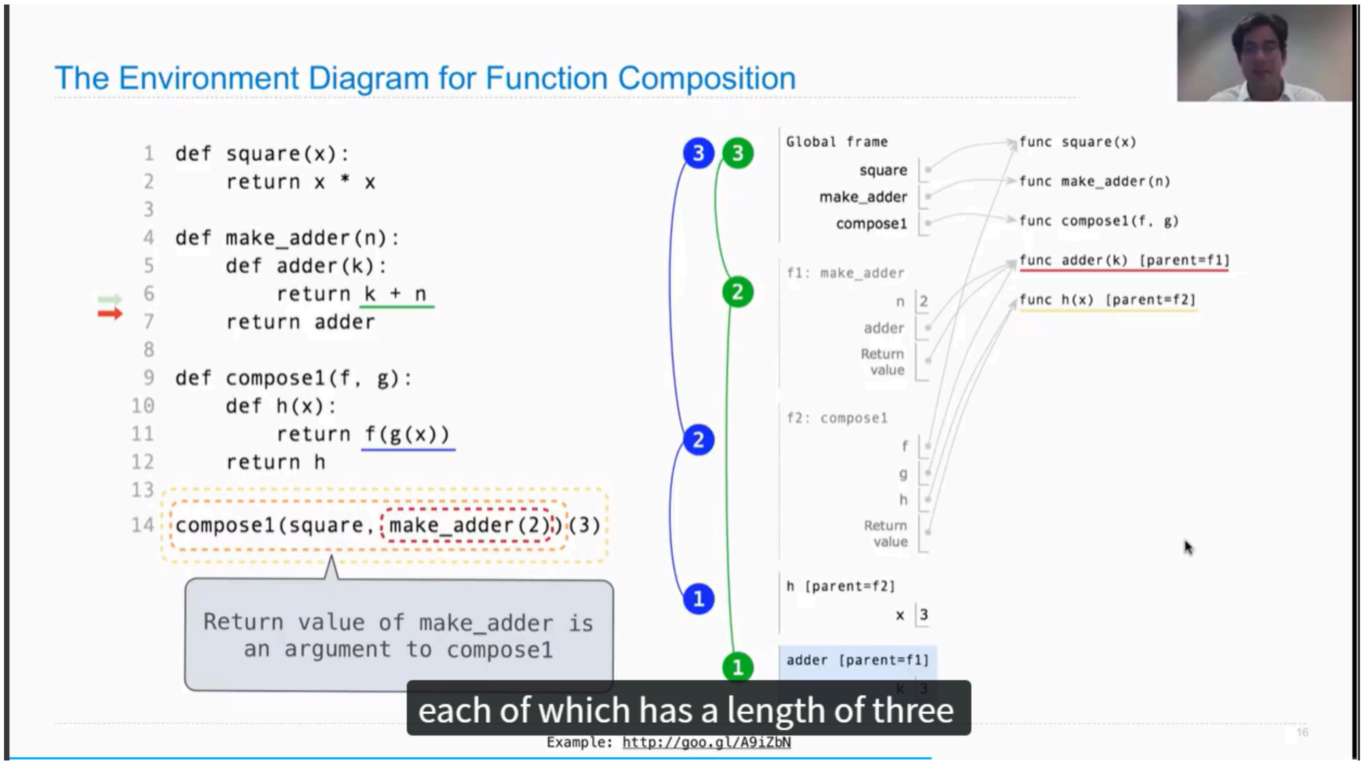
f1: make_adder   func adder(k) [parent=f1]

Bind <name> to the function value in the current frame

When a function is called:

1. Add a local frame, titled with the <name> of the function being called.
2. Copy the parent of the function to the local frame: [parent=<label>]
3. Bind the <formal parameters> to the arguments in the local frame.
4. Execute the body of the function in the environment that starts with the local frame.

drawing a image-- for sure-- will help you to get a good insight of how the program is working.



The Environment Diagram for Function Composition

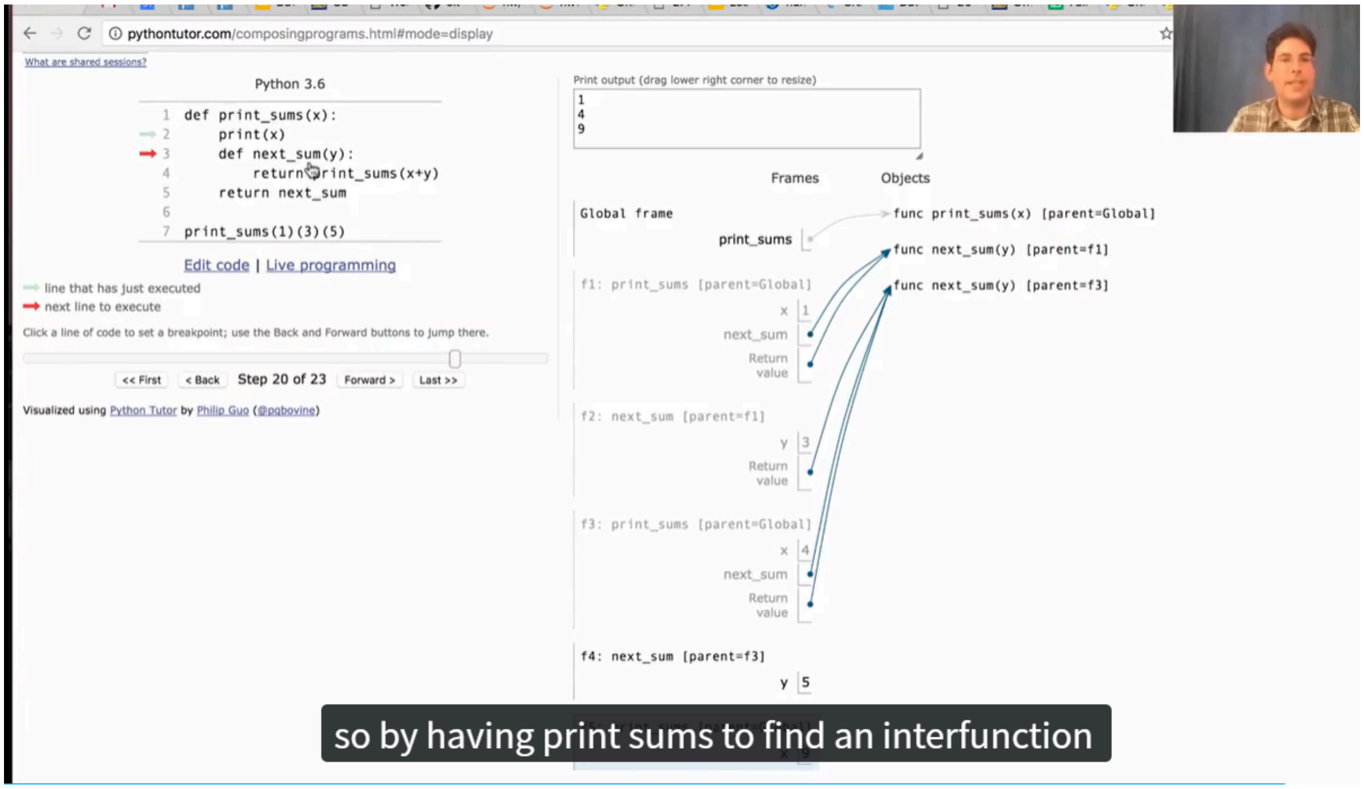
```python
def square(x):
    return x * x

def make_adder(n):
    def adder(k):
        return k + n
    return adder

def compose1(f, g):
    def h(x):
        return f(g(x))
    return h

compose1(square, make_adder(2))(3)
```

Return value of make_adder is an argument to compose1

each of which has a length of three

Example: http://goo.gl/A9iZbN

self reference



pythontutor.com/composingprograms.html#mode=display

What are shared sessions?

Python 3.6

```python
def print_sums(x):
    print(x)
    def next_sum(y):
        return print_sums(x+y)
    return next_sum

print_sums(1)(3)(5)
```

Edit code | Live programming

line that has just executed
next line to execute

Click a line of code to set a breakpoint; use the Back and Forward buttons to jump there.

<< First | < Back | Step 20 of 23 | Forward > | Last >>

Visualized using Python Tutor by Philip Guo (@pgbovine)

Print output (drag lower right corner to resize)

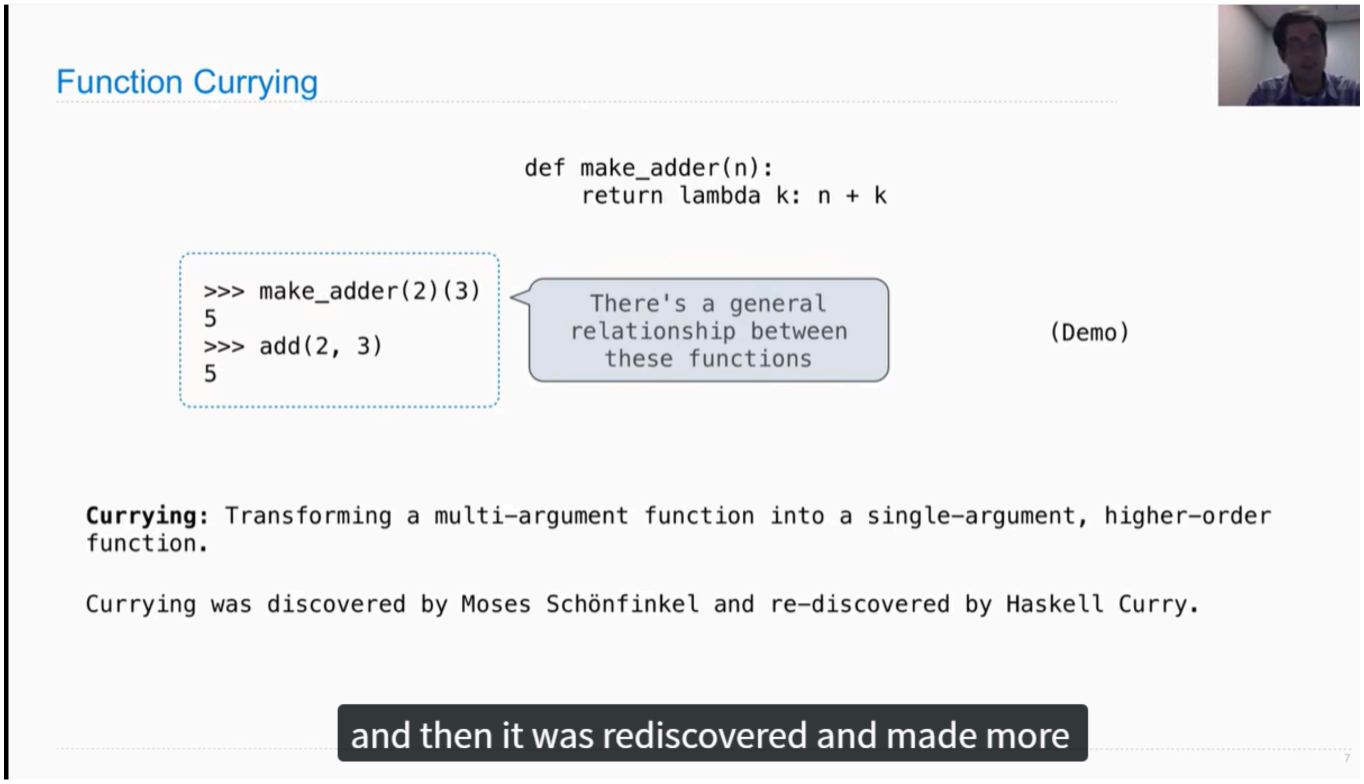
```
1
4
9
```

Frames
Objects
Global frame
print_sums
func print_sums(x) [parent=Global]
func next_sum(y) [parent=f1]
func next_sum(y) [parent=f3]
f1: print_sums [parent=Global]
x 1
next_sum
Return value
f2: next_sum [parent=f1]
y 3
Return value
f3: print_sums [parent=Global]
x 4
next_sum
Return value
f4: next_sum [parent=f3]
y 5

so by having print sums to find an interfunction

in this way, we are allowed to passed down data " stored in the last frames"

> 这也是我们在project1 中用到的所谓“函数状态机思想”


Function Currying

```
def make_adder(n):
    return lambda k: n + k
```

```
>>> make_adder(2)(3)
5
>>> add(2, 3)
5
```

There's a general relationship between these functions

(Demo)

Currying: Transforming a multi-argument function into a single-argument, higher-order function.

Currying was discovered by Moses Schönfinkel and re-discovered by Haskell Curry.

and then it was rediscovered and made more

in a word, function currying allows people to input the data step by step instead of requiring all the data at once.