Environments

Announcements

Twenty-One Reviewed — Why use HOFs?

We modeled the game by writing:

play(player_0_strategy_fn, player_1_strategy_fn)

- We can have many different strategies!
- **Returning a new function** allows us to make dynamic strategies.

```
def intelligent_ish_strategy(current_score):
   if current_score < 18:
        return three_strat
    if current_score == 18:
        return two_strat
    if current_score == 19:
        return one_strat
    else:
        return two_strat
```

Functions as Return Values

(Demo)

Locally Defined Functions

Functions defined within other function bodies are bound to names in a local frame

A function that returns a function

```
def make_adder(n):
    """Return a function that takes one argument k and returns k + n.

    >>> add_three = make_adder(3)
    >>> add_three(4)
    7
    """
    def adder(k):
        return k + n
    return adder
```

The name add_three is bound to a function

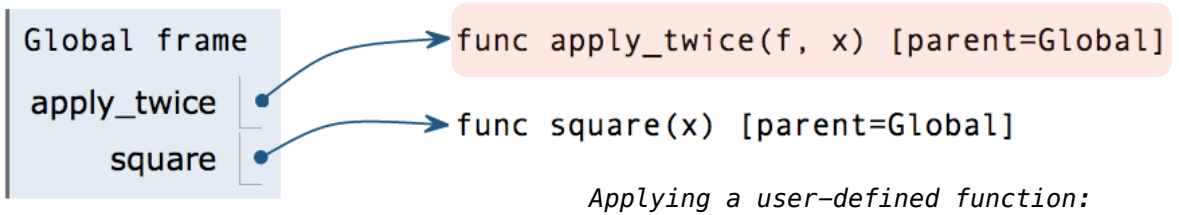
A def statement within another def statement

Can refer to names in the enclosing function

Environments for Higher-Order Functions

Names can be Bound to Functional Arguments

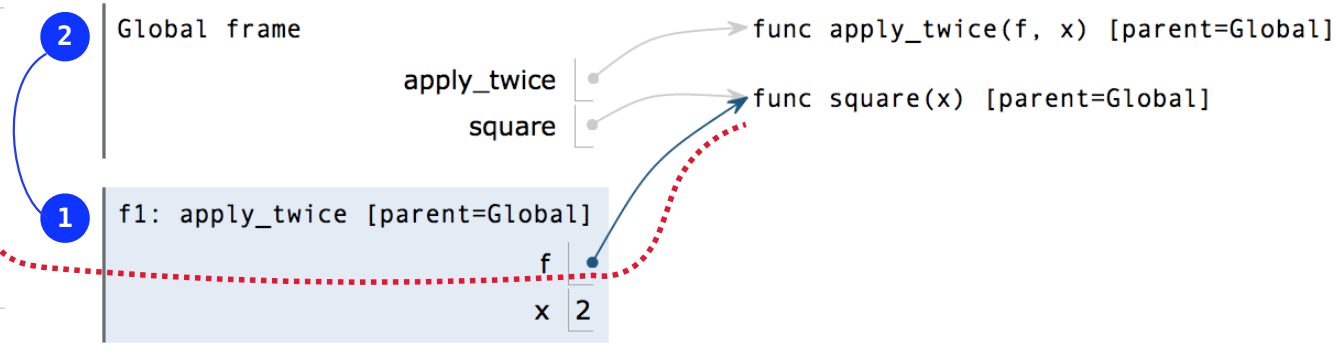
```
1  def apply_twice(f, x):
2      return f(f(x))
3
4  def square(x):
5      return x * x
6
7  result = apply_twice(square, 2)
```



Applying a user-defined function:

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

```
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)
```

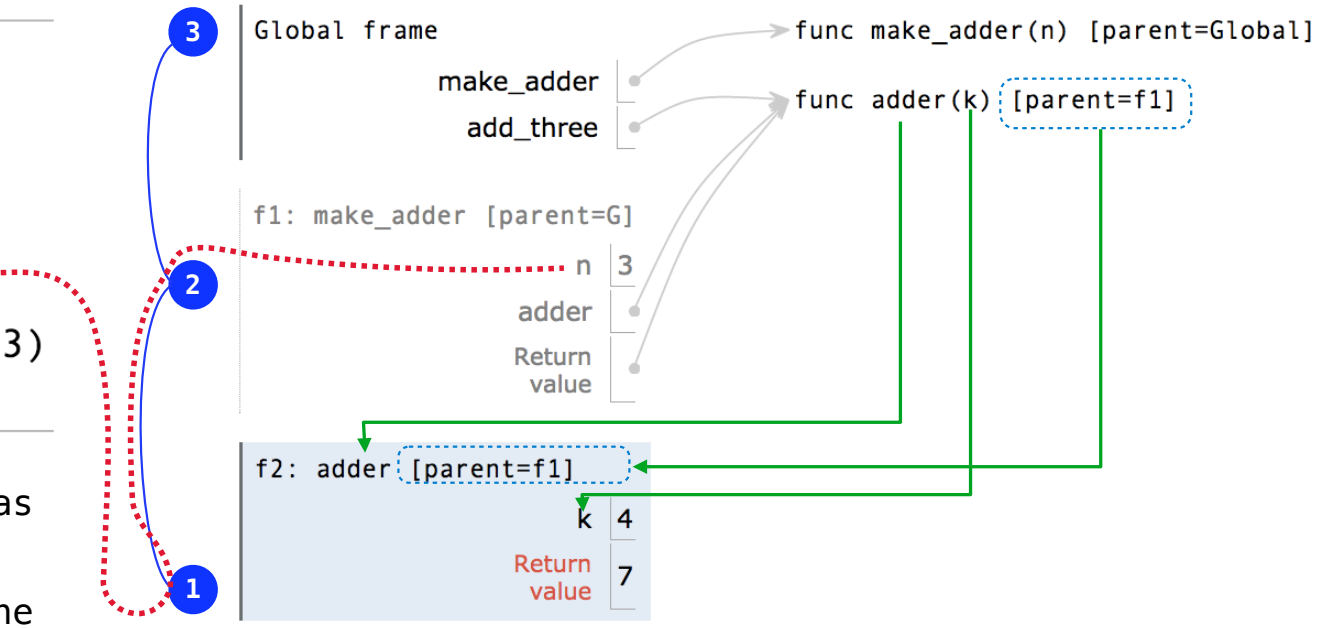


pythontutor.com/composingprograms.html#code=def%20apply_twice%28f,%20x%29%3A%0A%20%20%20%20return%20f%28f%28x%29%29%0A%20%20%20%20%0Adef%20square%28x%29%3A%0A%20%20%20%20return%20x%20*%20x%0A%20%20%20%20%0Aresult%20%3D%20apply_twice%28square,%202%29&mode=display&origin=composingprograms.js&cumulative=true&py=3&rawInputLstJSON=[]&curInstr=0

Environment Diagrams for Nested Def Statements

Nested def

```
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)
```



- 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 global)
- The parent of a frame is the parent of the function called

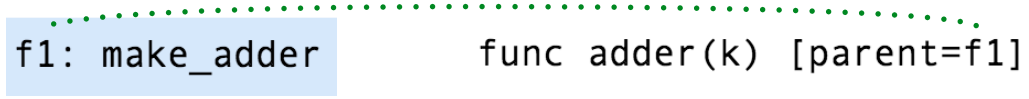
http://pythontutor.com/composingprograms.html#code=def%20make_adder%28n%29%3A%0A%20%20%20%20def%20adder%28k%29%3A%0A%20%20%20%20%20%20%20%20return%20k%20%2B%20n%0A%20%20%20%20return%20adder%0A%20%20%20%20%0Athree_more_than%20%3D%20make_adder%283%29%0Aresult%20%3D%20three_more_than%284%29&cumulative=false&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

How to Draw an Environment Diagram

When a function is defined:

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

Its parent is the current frame.



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.

Lambda Expressions

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=def%20apply_twice%28f,%20x%29%3A%0A%20%20%20%20return%20f%28f%28x%29%29%0A%20%20%20%20%0Ax%20%3D%203%0Aresult%20%3D%20apply_twice%28lambda%20y%3A%20x%20*%20y,%202%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

https://pythontutor.com/cp/composingprograms.html#code=bear%20%3D%20-1%0Aoski%20%3D%20lambda%20print%3A%20print%28bear%29%0Abear%20%3D%20-2%0Aprint%28oski%28abs%29%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Currying

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)

Curry: Transform a multi-argument function into a single-argument, higher-order function

Optional:
Environment Diagram Practice

An Old Midterm Question

- **The Diagram**
- **Annotations**

```
 1: def f(x):
 2:     """f(x)(t) returns max(x*x, 3*x)
 3:     if t(x) > 0, and 0 otherwise.
 4:     """
 5:     y = max(x * x, 3 * x)
 6:     def zero(t):
 7:         if t(x) > 0:
 8:             return y
 9:         return 0
10:     return zero
11:
12: # Find the largest positive y below 10
13: # for which f(y)(lambda z: z - y + 10)
14: # is not 0.
15: y = 1
16: while y < 10:
17:     if f(y)(lambda z: z - y + 10):
18:         max = y
19:     y = y + 1
```