Environment Diagrams and Higher Order Functions Solutions

Computer Science Mentors 61A

September 7–September 9, 2022

1 Environment Diagrams

1. When do we make a new frame in an environment diagram?

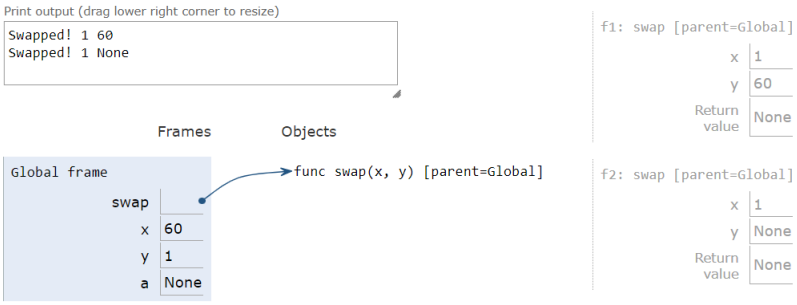
   We make a new frame in an environment diagram when calling a user-defined function, or when we are applying the operator to the operand(s). This occurs after both the operator and operand(s) are evaluated.

   New frames are *not* created when a function is defined, only when it is called.

2. Give the environment diagram and console output that result from running the following code.

```
def swap(x, y):
    x, y = y, x
    return print("Swapped!", x, y)

x, y = 60, 1
a = swap(x, y)
swap(a, y)
```



https://tinyurl.com/y68m6qdj

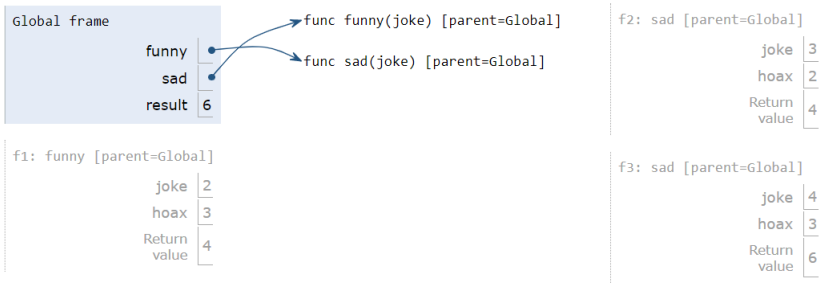
Vibha Tantry and Jordan Schwartz, with
Gabe Classon, Aditya Balasubramanian, Oliver Yu, Hanze Tan, Gene Pan, Carolyn Wang, Raman Varma, Noor Haq, Angela Zhang, Abigail Brooks, Ted Kim, Aiden Joshua Vehemente, Marie Chorpita, Ashley Chiu, Bill Hu, Omar Yu, Angel Alberto Aldaco, Andrew Park, Chase Clements, Grace Yi, Preshtha Garg, Laksith Venkatesh Prabu, Aditya Murali, Irene Yang, Andy Chen, and Hailey Park

3. Draw the environment diagram that results from running the following code.

```
def funny(joke):
    hoax = joke + 1
    return funny(hoax)

def sad(joke):
    hoax = joke - 1
    return hoax + hoax

funny, sad = sad, funny
result = funny(sad(2))
```



https://tinyurl.com/y5lc4fez

2 Higher-Order Functions

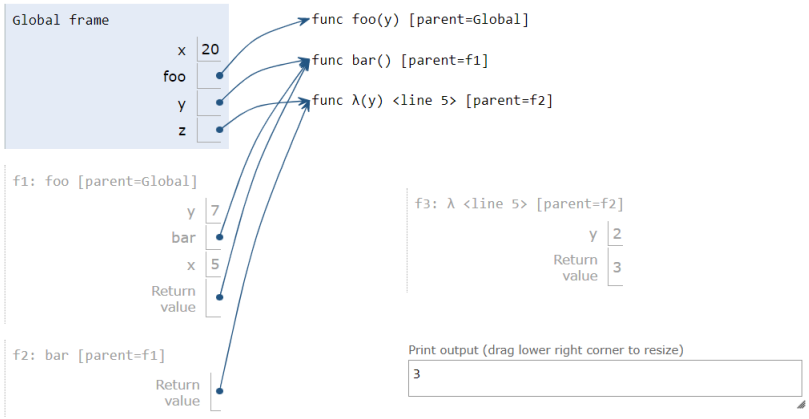
1. Why and where do we use lambda and higher-order functions?

   In practice, we use lambda functions to pass code as data in a concise manner. One specific example to illustrate the use of lambdas is the optional `key` parameter for **`min`** and **`max`** functions. Higher order functions serve as a tool of abstraction, allowing us to simplify repeated actions into one function that we can use over and over again, also referred to as currying.

2. Give the environment diagram and console output that result from running the following code.

```
x = 20
def foo(y):
    x = 5
    def bar():
        return lambda y: x - y
    return bar

y = foo(7)
z = y()
print(z(2))
```

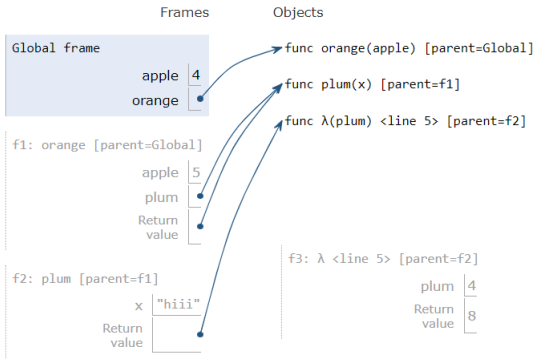


https://tinyurl.com/yxfcvxxa

3. Draw the environment diagram that results from running the code.

```
apple = 4
def orange(apple):
    apple = 5
    def plum(x):
        return lambda plum: plum * 2
    return plum

orange(apple)("hiii")(4)
```



https://tinyurl.com/y5lo34xb

4. Fill in the blanks (*without using any numbers in the first blank*) such that the entire expression evaluates to 9.

```
(lambda x: lambda y: lambda: y(x))(3)(lambda z: z*z)()
```

5. Write a function, `whole_sum`, which takes in an integer, `n`. It returns another function which takes in an integer, and returns `True` if the digits of that integer sum to `n` and `False` otherwise.

```
def whole_sum(n):
    """
    >>> whole_sum(21)(777)
    True
    >>> whole_sum(142)(10010101010)
    False
    """
    def check(x):

        ____________________

        while _______________:

            last = ____________________

            ____________________________

            ____________________________

        return ____________________

    return ____________________
```

```
def whole_sum(n):
    def check(x):
        total = 0
        while x > 0:
            last = x % 10
            x = x // 10
            total += last
        return total == n
    return check
```

6. Write a higher-order function that passes the following doctests.

   Challenge: Write the function body in one line.

```
def mystery(f, x):
    """
    >>> from operator import add, mul
    >>> a = mystery(add, 3)
    >>> a(4) # add(3, 4)
    7
    >>> a(12)
    15
    >>> b = mystery(mul, 5)
    >>> b(7) # mul(5, 7)
    35
    >>> b(1)
    5
    >>> c = mystery(lambda x, y: x * x + y, 4)
    >>> c(5)
    21
    >>> c(7)
    23
    """


    def helper(y):
        return f(x, y)
    return helper
```

   Challenge solution:

```
    return lambda y : f(x, y)
```

7. What would Python display?

```
>>> foo = mystery(lambda a, b: a(b), lambda c: 5 + square(c))
>>> foo(-2)
```

   9