CS 61A
Fall 2023

Environment Diagrams, Higher-Order Functions

Discussion 2: September 6, 2023

Getting Started [5 minutes]

The most common student suggestion from the previous discussion was to encourage groups to talk more. Please make sure that you work through the worksheet together. If you get ahead, stop and help your other group members for a while until they are caught up with you. Stay in sync like a dance group.

VERY IMPORTANT: In this discussion, you're not allowed to use a Python interpreter or run any code. Figure things out and check your work by *thinking* about what your code will do. Not sure? Talk to your group! Someone will know. If not… (You won't get to run Python during the midterm, so get used to solving problems without it now.)

To get help from a TA, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number.

What everyone will need: * A way to view this worksheet (on your phone is fine) * A way to take notes (paper or a device)

What the group will need: * Someone taking notes on a device (perhaps right here on this webpage) who is willing to share their screen of notes with a TA. They should open Discord. (It's fine if multiple people do this, but not everyone has to.)

Warm up question: what do you think `(lambda x: 2 * (lambda x: 3)(4))(5)` evaluates to? Talk about it with your whole group and then post a message to your group's Discord text channel that says: "We all think it's ###" (but replace ### with the answer).

Call Expressions [15 minutes]

It's time to draw an environment diagram. You can use paper or a tablet. Talk to your group about how you are going to draw it, then go through each step *together*.

Q1: Nested Calls Diagram

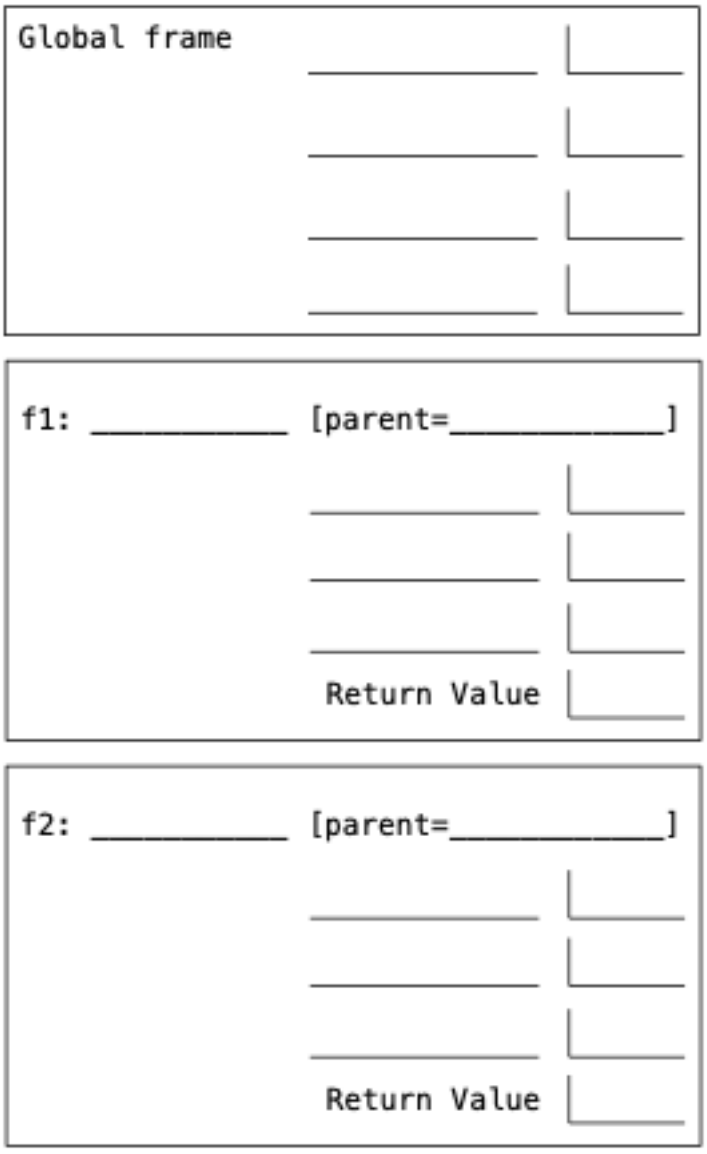
Draw the environment diagram that results from executing the code below.

```
def team(work):
    return t(work)-1

def dream(work, s):
    if work(s-2):
        t = not s
    return not t

work, t = 3, abs
team = dream(team, work + 1) and t
```

Here's a blank diagram in case you're using a tablet:



template

When you've all agreed on the diagram, check it against the solution on tutor.cs61a.org. (No peeking until your group has come up with an answer!)

If you didn't get it right and don't understand why, send a message to the `discuss-queue` channel with the @discuss tag and your discussion group number and talk it through with someone from the course staff.

Higher-Order Functions [60 minutes]

Remember the problem-solving approach from last discussion; it works just as well for implementing higher-order functions.

1. Pick an example input and corresponding output. *(This time it might be a function.)*
2. Describe a process (in English) that computes the output from the input using simple steps.
3. Figure out what additional names you'll need to carry out this process.
4. Implement the process in code using those additional names.
5. Determine whether the implementation really works on your original example.
6. Determine whether the implementation really works on other examples. (If not, you might need to revise step 2.)

Q2: Make Keeper

Implement `make_keeper`, which takes a positive integer `n` and returns a function `f` that takes as its argument another one-argument function `cond`. When `f` is called on `cond`, it prints out the integers from 1 to `n` (including `n`) for which `cond` returns a true value when called on each of those integers. Each integer is printed on a separate line.

```
def make_keeper(n):
    """Returns a function that takes one parameter cond and prints
    out all integers 1..i..n where calling cond(i) returns True.

    >>> def is_even(x): # Even numbers have remainder 0 when divided by 2.
    ...     return x % 2 == 0
    >>> make_keeper(5)(is_even)
    2
    4
    >>> make_keeper(5)(lambda x: True)
    1
    2
    3
    4
    5
    >>> make_keeper(5)(lambda x: False)  # Nothing is printed
    """
    def f(cond):
        i = 1
        while i <= n:
            if cond(i):
                print(i)
            i += 1
    return f
```

Don't run Python to check your work. You can check it just by thinking!. If you get stuck, ask the staff for help.

Once your group has converged on a solution, now it's time to practice your ability to describe your own code. A

good description is like a good program: concise and accurate. Nominate someone to describe how your solution works and have them present to the group for practice. Then, send a message to the `discuss-queue` channel with the @discuss tag, your discussion group number, and the message "Listen up!" and a member of the course staff will join your voice channel to hear your description. If no one comes right away, feel free to work on the next problem until someone from the staff shows up.

Q3: Digit Finder

Implement `find_digit`, which takes in a positive integer `k` and returns a function that takes in a positive integer `x` and returns the `k`th digit from the right of `x`. If `x` has fewer than `k` digits, it returns 0.

For example, in the number 4567, 7 is the 1st digit from the right, 6 is the 2nd digit from the right, and the 5th digit from the right is 0 (since there are only 4 digits).

Important: You may not use strings or indexing for this problem.

Hint: Floor dividing by a power of 10 gets rid of the rightmost digits.

```
def find_digit(k):
    """Returns a function that returns the kth digit of x.

    >>> find_digit(2)(3456)
    5
    >>> find_digit(2)(5678)
    7
    >>> find_digit(1)(10)
    0
    >>> find_digit(4)(789)
    0
    """
    assert k > 0
    return lambda x: (x // pow(10, k-1)) % 10
```

Stop here until everyone understands the solution.

Q4: Match Maker

Implement `match_k`, which takes in an integer `k` and returns a function that takes in a variable `x` and returns `True` if all the digits in `x` that are `k` apart are the same.

For example, `match_k(2)` returns a one argument function that takes in `x` and checks if digits that are 2 away in `x` are the same.

`match_k(2)(1010)` has the value of `x = 1010` and digits 1, 0, 1, 0 going from left to right. `1 == 1` and `0 == 0`, so the `match_k(2)(1010)` results in `True`.

`match_k(2)(2010)` has the value of `x = 2010` and digits 2, 0, 1, 0 going from left to right. `2 != 1` and `0 == 0`, so the `match_k(2)(2010)` results in `False`.

Important: You may not use strings or indexing for this problem. You do not have to use all the lines; one staff solution does not use the line directly above the while loop.

Hint: Floor dividing by powers of 10 gets rid of the rightmost digits.

```
def match_k(k):
    """Returns a function that checks if digits k apart match.

    >>> match_k(2)(1010)
    True
    >>> match_k(2)(2010)
    False
    >>> match_k(1)(1010)
    False
    >>> match_k(1)(1)
    True
    >>> match_k(1)(2111111111111111)
    False
    >>> match_k(3)(123123)
    True
    >>> match_k(2)(123123)
    False
    """
    def check(x):
        i = 0
        while 10 ** (i + k) < x:
            if (x // 10**i) % 10 != (x // 10 ** (i + k)) % 10:
                return False
            i = i + 1
        return True
    return check
```

Here's an alternate solution:

```
def match_k_alt(k):
    """ Return a function that checks if digits k apart match

    >>> match_k_alt(2)(1010)
    True
    >>> match_k_alt(2)(2010)
    False
    >>> match_k_alt(1)(1010)
    False
    >>> match_k_alt(1)(1)
    True
    >>> match_k_alt(1)(2111111111111111)
    False
    >>> match_k_alt(3)(123123)
    True
    >>> match_k_alt(2)(123123)
    False
    """
    def check(x):
        while x // (10 ** k):
            if (x % 10) != (x // (10 ** k)) % 10:
                return False
            x //= 10
        return True
    return check
```

If you have time, practice your ability to describe your own code again. Nominate *someone else* to describe how your solution works and have them present to the group for practice. Then, send a message to the `discuss-queue` channel with the @discuss tag, your discussion group number, and the message "Match with us!" and a member of the course staff will join your voice channel to hear your description.

Lambda Lightning round [5 minutes]

If you have time, try these. If you don't have time, that's ok. You've already worked on the most important part of the discussion.

Q5: Make Your Own Lambdas

For the following problem, first read the doctests for functions `f1`, `f2`, `f3`, and `f4`. Then, implement the functions to conform to the doctests without causing any errors. **Be sure to use lambdas in your function definition instead of nested `def` statements.** Each function should have a one line solution.

```
def f1():
    """
    >>> f1()
    3
    """
    return 3

def f2():
    """
    >>> f2()()
    3
    """
    return lambda: 3

def f3():
    """
    >>> f3()(3)
    3
    """
    return lambda x: x

def f4():
    """
    >>> f4()()(3)()
    3
    """
    return lambda: lambda x: lambda: x
```

Document the occasion

Please all fill out the attendance form (one submission per person per week).