CS 61A Spring 2025

Environment Diagrams, Higher-Order Functions

Discussion 2: February 5, 2025

Switch to Pensieve:

- **Everyone**: Go to pensieve.co, log in with your @berkeley.edu email, and **enter your group number** as the room number (which was in the email that assigned you to this discussion). As long as you all enter the same number (any number), you'll all be using a shared document.

Once you're on Pensieve, you don't need to return to this page; Pensieve has all the same content (but more features). If for some reason Penseive doesn't work, return to this page and continue with the discussion.

Attendance

Fill out this discussion attendance form with the unique number you receive from your TA. As soon as you get your number, fill out the form, selecting *arrival* (not *departure* – that's later).

Getting Started [5 minutes]

Say your name and a city (or place) that you like, which is not Berkeley and is not where you have lived. Feel free to share why you like it.

VERY IMPORTANT: In this discussion, don't press *Check Answer* or run any Python code until your whole group is sure that the answer is right. Your goal should be to have **all checks pass the first time**! Figure things out and check your work by *thinking* about what your code will do. Not sure? Talk to your group! (You won't get to run Python during the midterm, so get used to solving problems without it now.)

Q1: Warm Up

What is the value of `result` after executing `result = (lambda x: 2 * (lambda x: 3)(4) * x)(5)`? Talk about it with your whole group and make sure you all agree before anybody checks the answer.

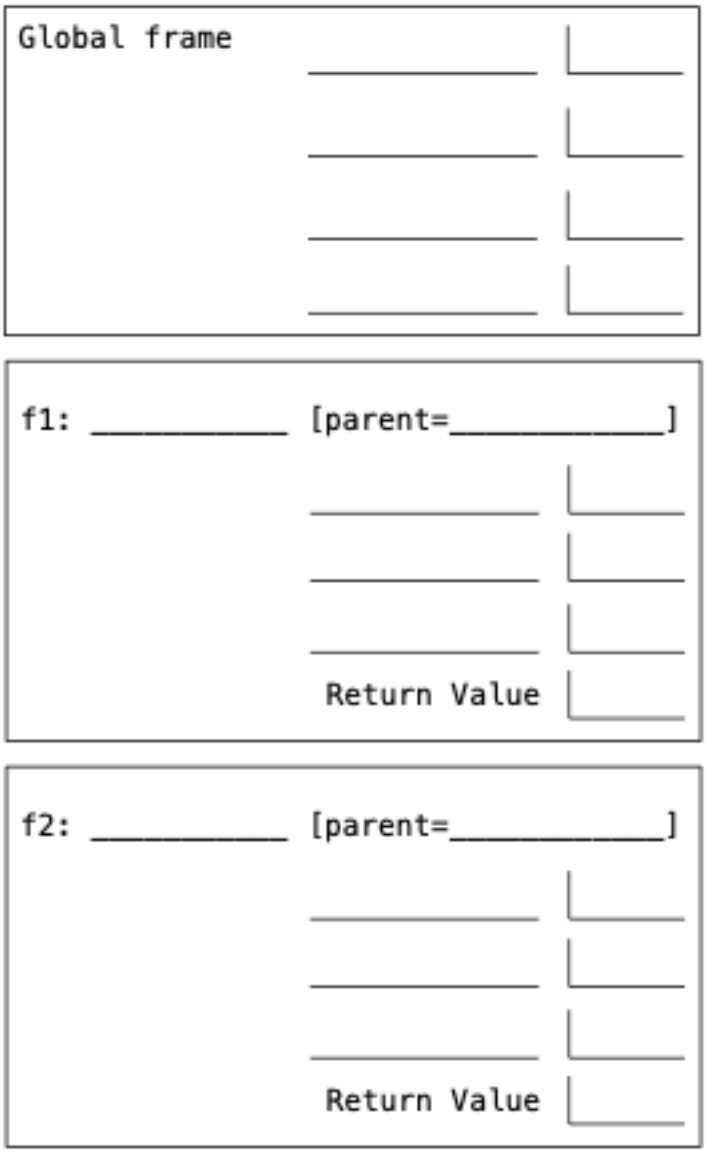
Call Expressions [15 minutes]

Draw an environment diagram for the code below. You can use paper or a tablet or the whiteboard. Talk to your group about how you are going to draw it, then go through each step *together*. Then, step through the diagram generated by Python Tutor to check your work.

See the web version of this resource for the environment diagram.

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

If you have questions, ask them instead of just looking up the answer! First ask your group, and then your TA.



template

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

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
```

No peeking! First try to implement it without the hint.

To return a function `f`, include `def f(cond):` as the first line of the implementation and `return f` as the last. The `f` function should introduce `i = 1` in order to loop through all integers, calling `cond(i)` to determine whether `cond`

returns true for each integer.

Don't run Python to check your work. You can check it just by thinking!.

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. If you want feedback, you can also present to your TA.

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. Try to solve this problem using only one line.

Hint: Lambda expressions.

Hint: Use 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
```

First remove all of the digits after digit `k`, at which point digit `k` will be the last digit.

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.

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):
        while x // (10 ** k) > 0:
            if (x % 10) != (x // (10 ** k)) % 10:
                return False
            x //= 10
        return True
    return check
```

In each iteration, compare the last digit with the one that is `k` positions before it.

Document the occasion

Let your TA know you're done so that you can each get a **departure** number, and fill out the attendance form again (this time selecting *departure* instead of *arrival*). If your TA isn't in the room, go find them next door.