CS 61A Higher-Order Functions, Environment Diagrams
Summer 2025 Discussion 2: July 1, 2025

# Getting Started

**VERY IMPORTANT:** In this discussion, don't run any Python code or look at the correct environment diagram until you are sure that the answer is right. Your goal should be to get the question right without having to have a computer check your work! Figure things out and check your work by *thinking* about what your code will do. (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)`?

# Environment Diagrams

An **environment diagram** keeps track of names and their values in frames, which are drawn as boxes.

### Q2: Bottles

Answer the following questions and afterwards, step through the diagram to check your answers.

1) What determines how many different frames appear in an environment diagram?

a) The number of functions defined in the code
b) The number of call expressions in the code
c) The number of return statements in the code
d) The number of times user-defined functions are called when running the code

2) What happens to the return value of `pass_it(bottles)`?

a) It is used as the new value of `remaining` in the global frame
b) It is used as the new value of `bottles` in the global frame
c) It is used as the new value of `pass_it` in the global frame
d) None of the above

3) What effect does the line `bottles = 98` have on the global frame?

a) It temporarily changes the value bound to `bottles` in the global frame.

b) It permanently changes the value bound to `bottles` in the global frame.

c) It has no effect on the global frame.

bottles = 99 take = 1

def pass_it(around): bottles = 98 return take

remaining = bottles - pass_it(bottles) bottles = remaining

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

### Q3: Silence of the Lambda

Draw the environment diagram on paper or a tablet (without having the computer draw it for you)! Then, check your work by stepping through the diagram with PythonTutor

```
def r(f):
    k = 2
    k, m = k + 1, f(k)
    return n

n = 10
g = (lambda n: lambda k: print(k * n))(-1)
r(g)
```

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

### Q4: Dream Work

Draw an environment diagram for the code below. Then, step through the diagram with PythonTutor to check your work.

```
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
```

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

# Higher-Order Functions

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.)

### Q5: 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 to get 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
    "*** YOUR CODE HERE ***"
```

### Q6: 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
    """
    "*** YOUR CODE HERE ***"
```

### Q7: 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 ____________________________:
                return ____________________________
            x //= 10
        ____________________________
    ____________________________
```

# Submit Attendance

You're done! Excellent work this week. Please be sure to fill out your TA's attendance form to get credit for this discussion!