CS 61A
Spring 2026

Environment Diagrams, Higher-Order Functions

Discussion 2: February 4, 2026

Attendance

Your TA will come around during discussion to check you in. You can start on the worksheet before being checked in; you don't need to wait for your TA to get started.

Getting Started

If your group has 6 or more students, you're welcome to split into two sub-groups and then sync up at the end.

Q1: Warm Up

What is the value of `result` after executing `result = (lambda x: 2 * (lambda x: 3)(4) * x)(5)`?

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.)

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.

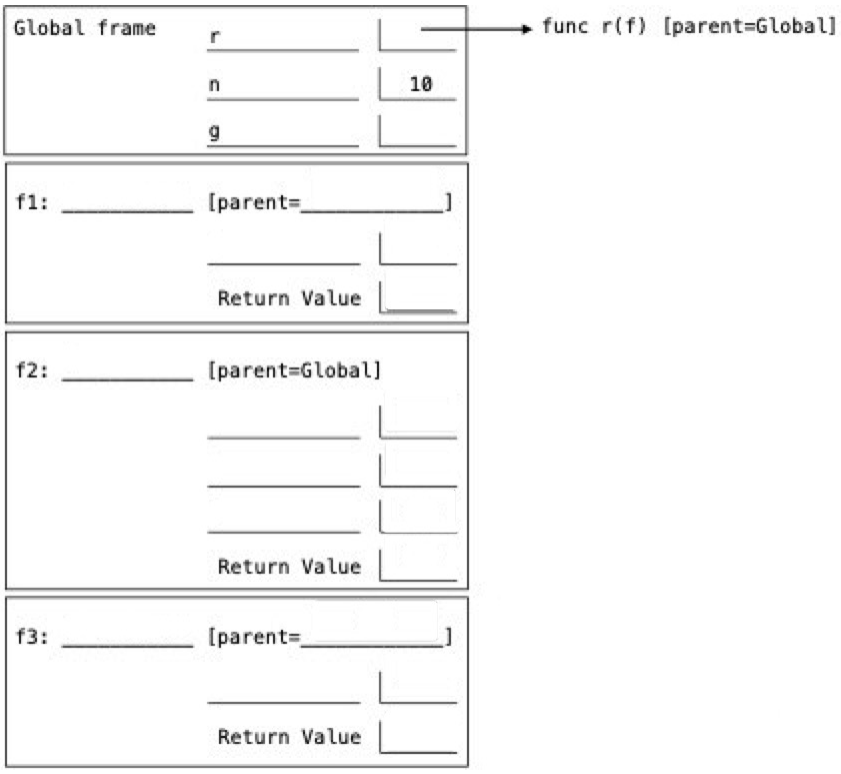
```
def make_keeper(n):
    """Returns a function that takes one parameter cond and prints
    out all integers 1..i..n where calling cond(i) returns True.

    >>> 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
    """
    "*** YOUR CODE HERE ***"
```

Call Expressions

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. This problem's video contains the solution.




```
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: Match Maker

Assume that `find_digit(k)` is correctly implemented. `find_digit` 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`.

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 may call `find_digit`.

Tip: 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

        ____________________________
    ____________________________
```

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

Q5: Ups and Downs A

Definition. Two adjacent digits in a non-negative integer are an *increase* if the left digit is smaller than the right digit, and a *decrease* if the left digit is larger than the right digit.

For example, 61127 has 2 increases ($1 \to 2$ and $2 \to 7$) and 1 decrease ($6 \to 1$).

You may use the sign function defined below in all parts of this question.

```
def sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return- 1
    else:
        return 0
```

Implement `ramp`, which takes a non-negative integer `n` and returns whether it has more *increases* than *decreases* when reading its digits from left to right (see the definition above).

```
def ramp(n):
    """Return whether non-negative integer N has more increases than decreases.

    >>> ramp(123)   # 2 increases (1-> 2, 2-> 3) and 0 decreases
    True
    >>> ramp(1315)  # 2 increases (1-> 3, 1-> 5) and 1 decrease (3-> 1)
    True
    >>> ramp(176)   # 1 increase (1-> 7) and 1 decrease (7-> 6)
    False
    >>> ramp(5)     # 0 increases and 0 decreases
    False
    """
    n, last, tally = ____________________________, ____________________________, 0

    while n:
        n, last, tally = n // 10, n % 10, ____________________________
    return ____________________________
```

Q6: Ups and Downs C

The process function below uses `tally` and `result` functions to analyze all adjacent pairs of digits in a non-negative integer `n`. A `tally` function is called on each pair of adjacent digits.

```
def process(n, tally, result):
    """Process all pairs of adjacent digits in N using functions TALLY and RESULT.
    """

    while n >= 10:
        tally, result = tally(n % 100 // 10, n % 10)
        n = n // 10
    return result()
```

Implement `ups`, which returns two functions that can be passed as `tally` and `result` arguments to `process`, so that `process` computes whether a non-negative integer `n` has exactly `k` *increases*.

Hint: You can use `sign` from the previous page and the built-in `max` and `min` functions.

```
def ups(k):
    """Return tally and result functions that compute whether N has exactly K increases.

    >>> f, g = ups(3)
    >>> process(1200849, f, g)    # Exactly 3 increases: 1 -> 2, 0 -> 8, 4 -> 9
    True
    >>> process(94004, f, g)      # 1 increase: 0 -> 4
    False
    >>> process(122333445, f, g)  # 4 increases: 1 -> 2, 2 -> 3, 3 -> 4, 4 -> 5
    False
    >>> process(0, f, g)          # 0 increases
    False
    """
    def f(left, right):
        return __________(___________________________________)
    return _____________________________, _____________________________
```

Q7: Choose Wisely A

Definition: A *digit test* is a function that takes a non-negative integer less than 10 and returns `True` or `False`.

Implement `only` which takes a non-negative integer `n` and a *digit test* `t`. It returns a non-negative integer containing only the digits of `n` for which `t` returns `True` when called on each of those digits. The digits should appear in the same order as they did in `n`. The number 0 has no digits. **You may not use `str` or `repr` or `[` or `]` or `for`.**

```
def only(n, t):
    """Return only the digits of n for which t returns True when called on each digit

    >>> only(23344567, lambda d: d % 2 == 0)
    2446
    >>> only(987654349675, lambda d: d < 7)
    6543465
    >>> only(2023, lambda d: False)
    0
    """
    keep = 0
    while n:
        n, d = n // 10, n % 10
        if ____________________________:
            keep = 10 * keep + ___________
    ____________________________:
        n, keep = ____________________________
    return n
```

Q8: Choose Wisely B

Implement `every` which takes a *digit test* `t` and returns a function `digit` that takes a positive integer `n`. The `digit` function returns whether `t` returns True for every digit of `n`.

```
def every(t):
    """Return a function that returns whether t is True
    for every digit of non-negative n.

    >>> f = every(lambda d: d % 2 == 1)
    >>> f(37511)  # every digit is odd
    True
    >>> f(2023)   # Not every digit is odd
    False
    """
    def digit(n):
        while n:
            if ____________________________:
                ____________________________
            n = n // 10
        return ____________________________
    return ____________________________
```

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