CS 61A Summer 2026

Higher-Order Functions, Environment Diagrams

Discussion 2: June 30, 2026

Getting Started

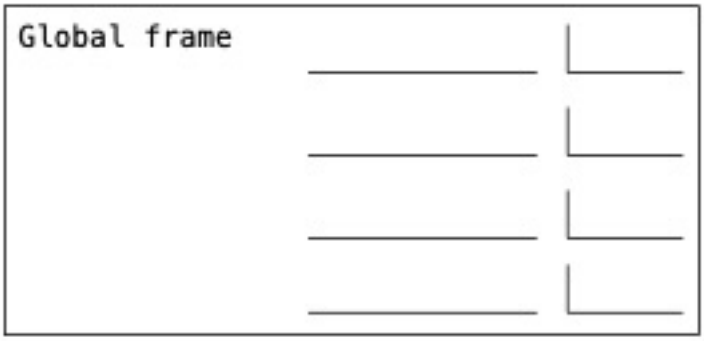
Q1: Warm Up

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

Environment Diagrams

Q2: Double Trouble

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



template

```
def double(x):
    return x * 2

def triple(x):
    return x * 3

hat = double
double = triple
```

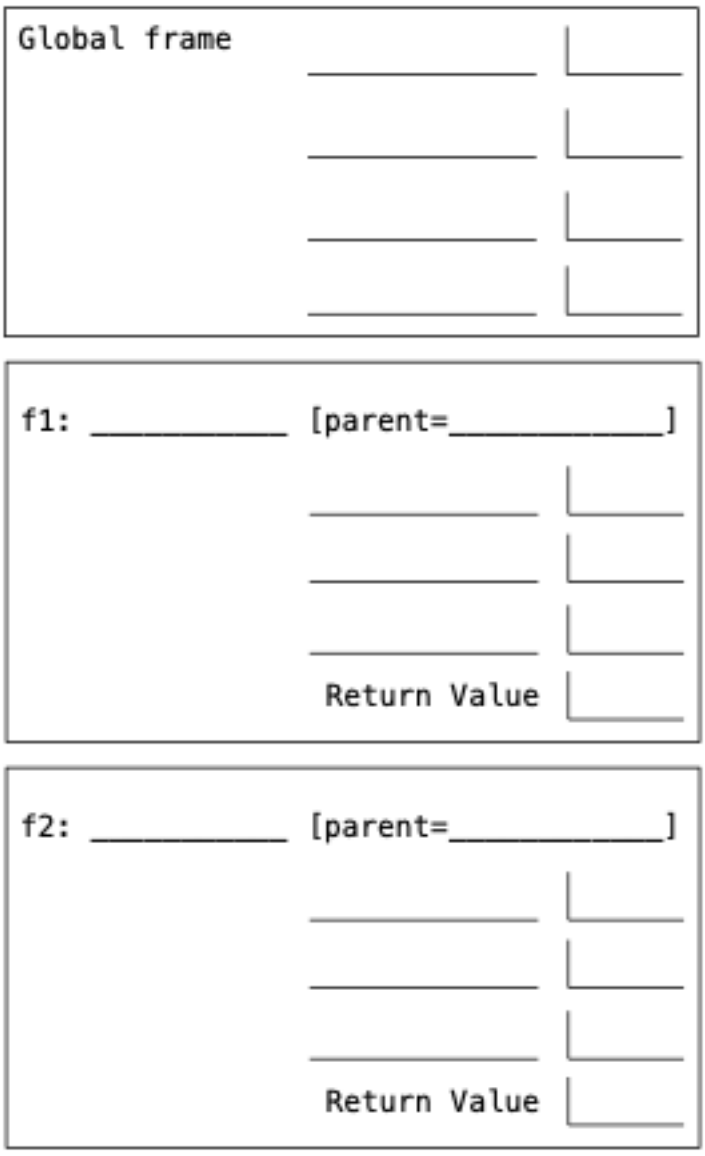
See the web version of this resource for the environment diagram.

We first define the two functions `double` and `triple`, each bound to their corresponding name. In the next line, we assign the name `hat` to the function object that `double` refers to. Finally, we assign the name `double` to the function object that `triple` refers to.

Video walkthrough

Q3: 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.

Q4: 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.

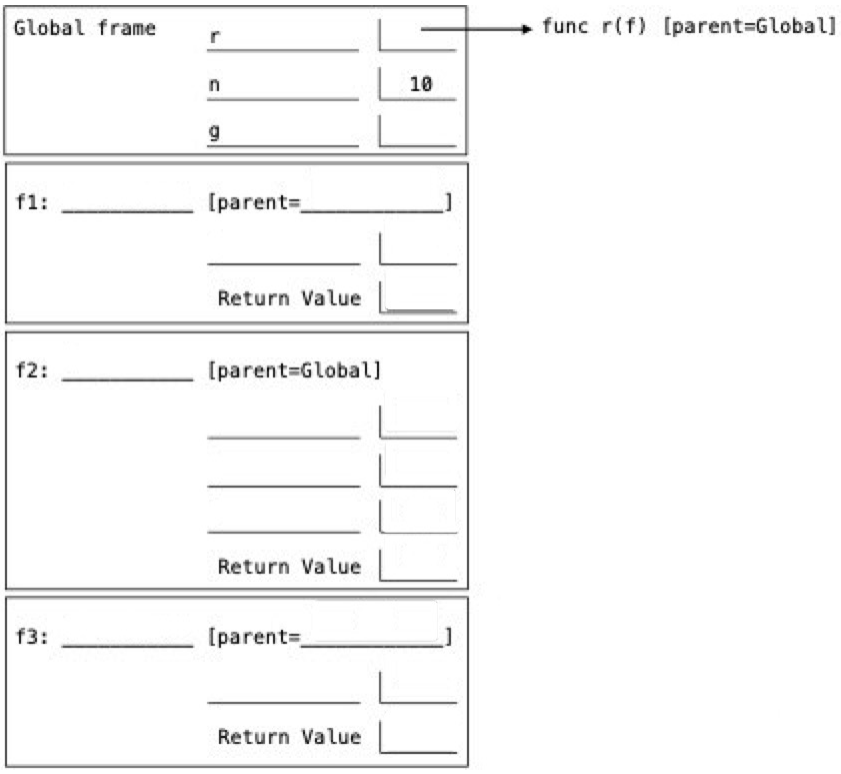
```
def make_keeper(n):
    """Returns a function that takes one parameter cond and prints
    out all integers 1..i..n where calling cond(i) returns True.

    >>> 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
```

Call Expressions

Q5: 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.

Q6: 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.

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 (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.

Q7: 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 sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return- 1
    else:
        return 0

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 = n // 10, n % 10, 0

    while n:
        n, last, tally = n // 10, n % 10, tally + sign(last - n % 10)
    return tally > 0
```

Q8: 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 sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return- 1
    else:
        return 0

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()

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 ups(min(k, k + sign(left - right)))
    return f, lambda: k == 0

def ups_alt(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 ups_alt(k - max(0, sign(right - left)))
    return f, lambda: k == 0
```

Q9: 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 t(d):
            keep = keep * 10 + d
    while keep:
        n, keep = 10 * n + keep % 10, keep // 10
    return n
```

Q10: 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 not t(n % 10):
                return False
            n = n // 10
        return True
    return digit
```