CS 61A Structure and Interpretation of Computer Programs
Summer 2021

Diagnostic Solutions

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

You can complete and submit these questions before the exam starts.

(a) What is your full name?

(b) What is your student ID number? A regex restricts inputs to numerical responses only.

1. (1.0 points) Applications are Closed

Write a higher order function `apply_until_n` that takes in `f`, a one-argument function, and a positive integer `n` > 0. `apply_until_n` should return a function that takes in a number `x` and applies `f` onto `x` as many times as possible before the result exceeds `n`.

This function should return the latest result that doesn't exceed `n`. You may assume that the return value of `f` is always strictly greater than the input.

```
def apply_until_n(f, n):
    """
    >>> square = lambda x: x * x
    >>> add_one = lambda x: x + 1
    >>> apply_until_n(add_one, 5)(3)
    5
    >>> apply_until_n(square, 10)(2)
    4
    >>> apply_until_n(square, 20)(2)
    16
    """
    def applier(x):
        while ________:
        #        (a)
            ____________
            #    (b)
        return x
    return ________
    #         (c)
```

(a) Fill in blank (a)?

f(x) <= n

(b) Fill in blank (b)?

x = f(x)

(c) Fill in blank (c)?

applier

2. (1.0 points) Catch that Bug!

Your classmates are trying to complete a coding assignment but are struggling. The assignment is to write a function `digit_counter`, which takes in `f`, a one-argument function, and `n`, a non-negative integer, and returns the number of digits in `n` for which `f(digit)` returns `True`.

Here's a doctest showing the intended behavior:

```
>>> is_even = lambda x: (x % 2) == 0
>>> digit_counter(is_even, 1112)
1
>>> digit_counter(is_even, 9843)
2
>>> greater_than_three = lambda x: x > 3
>>> digit_counter(greater_than_three, 123456789)
6
```

In all below parts of this problem, you don't need to indent your answer when writing code. Each solution can be made correct by changing exactly one line.

(a) Albert has decided to use a while loop to complete the assignment, but it's not working!

```
1 def digit_counter(f, n):
2     counter = 0
3     while n >= 0:
4         if f(n % 10):
5             counter += 1
6         n = n // 10
7     return counter
```

Which line number is Albert's error on?

3

(b) What should that line be replaced with?

n > 0

(c) Alex has decided to use recursion instead, but it's also not working!

```
1 def digit_counter(f, n):
2     if n < 10 and f(n):
3         return n
4     if f(n % 10):
5         return 1 + digit_counter(f, n // 10)
6     return digit_counter(f, n // 10)
```

Which line number is Alex's error on?

2

(d) What should that line be replaced with?

if n == 0:

(e) Catherine has taken an unconventional approach and has decided to use a helper function – and it's still wrong!

```
1 def digit_counter(f, n):
2     def helper(x, sofar):
3         if x > n:
4             return sofar
5         last = (n // x) % 10
6         return helper(x * 10, sofar + f(last))
7     return helper(0, 0)
```

Which line number is Catherine's error on?

7

(f) What should that line be replaced with?

return helper(1, 0)

3. (1.0 points) Oh, Camel!

Definition: A *camel sequence* is an integer in which each digit is either strictly less than or strictly greater than both of its adjacent digits. Write a function `is_camel_sequence` that takes in a nonnegative integer `n` and returns whether `n` is a *camel sequence*.

Note: Any single digit integer is a valid camel sequence.

Restrictions: You may not use `int`, `str`, `[` or `]` in your solution.

```
def is_camel_sequence(n):
    """
    >>> is_camel_sequence(15263) # 1 < 5, 5 > 2, 2 < 6, 6 > 3
    True
    >>> is_camel_sequence(98989)
    True
    >>> is_camel_sequence(123) # 1 < 2, but 2 is not greater than 3.
    False
    >>> is_camel_sequence(4114) # 1 is not strictly less than 1
    False
    >>> is_camel_sequence(1)
    True
    >>> is_camel_sequence(12)
    True
    >>> is_camel_sequence(11)
    False
    """
    def helper(n, incr):
        if _____________:
        #      (a)
            return True
        elif incr:
            return _______________ and helper(_____________)
            #            (b)                       (c)
        else:
            return _______________ and helper(_____________)
            #            (d)                       (e)
    return ____________ or _____________
    #          (f)             (g)
```

(a) Fill in blank (a)?

n < 10

(b) Which of the following could fill in blank (b)?

- [x] `n % 10 < n // 10 % 10`
- [ ] `n % 10 <= n // 10 % 10`
- [ ] `n % 10 < n // 10`
- [ ] `n % 10 <= n // 10`
- [ ] `n % 10 < n % 100`
- [ ] `n % 10 <= n % 100`
- [ ] `n % 10 < n % 10 // 10`
- [ ] `n % 10 <= n % 10 // 10`

(c) Fill in blank (c)?

n // 10, not incr

(d) Which of the following could fill in blank (d)?

- [x] `n % 10 > n // 10 % 10`
- [ ] `n % 10 >= n // 10 % 10`
- [ ] `n % 10 > n // 10`
- [ ] `n % 10 >= n // 10`
- [ ] `n % 10 > n % 100`
- [ ] `n % 10 >= n % 100`
- [ ] `n % 10 > n % 10 // 10`
- [ ] `n % 10 >= n % 10 // 10`

(e) Fill in blank (e)?

n // 10, not incr

(f) Fill in blank (f)?

helper(n, True)

(g) Fill in blank (g)?

helper(n, False)

No more questions.