241219

CS 61A Fall 2024 — Structure and Interpretation of Computer Programs

MIDTERM 1

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?

(c) What is your @berkeley.edu email address?

(d) Sign (or type) your name to confirm that all work on this exam will be your own. The penalty for academic misconduct on an exam is an F in the course.

1. (8.0 points) What Would Python Display?

Answer the following questions about the output printed by this code.

```
def mad(max):
    g = lambda: (print(1) or 2) or (print(3) or 4)
    print(max(5, 6))
    return g
print(mad(print)(), 7, print(8))
```

输出: 5 6
None
1 3
8
2 7 None

(a) (2.0 pt) What is the last line printed?

- ○ 1 7 8
- ○ 2 7 8
- ● 2 7 None
- ○ 4 7 8
- ○ 4 7 None
- ○ None 7 8
- ○ None 7 None

(b) (4.0 pt) Which of these **whole lines** appear **somewhere** in the printed output? Select all that apply.

- ☑ 1
- ☐ 1 2
- ☐ 1 2 3
- ☐ 1 2 3 4
- ☐ 3
- ☐ 5
- ☑ 5 6
- ☐ 6
- ☑ 7
- ☑ 8
- ☑ None
- ☐ None None
- ☐ True

注意：mad(print)中调用了print(5,6)

(c) (2.0 pt) What **order** do 1, 6, and 8 appear in the printed output? These numbers may appear as part of longer lines. If a number appears twice, consider just the first occurence.

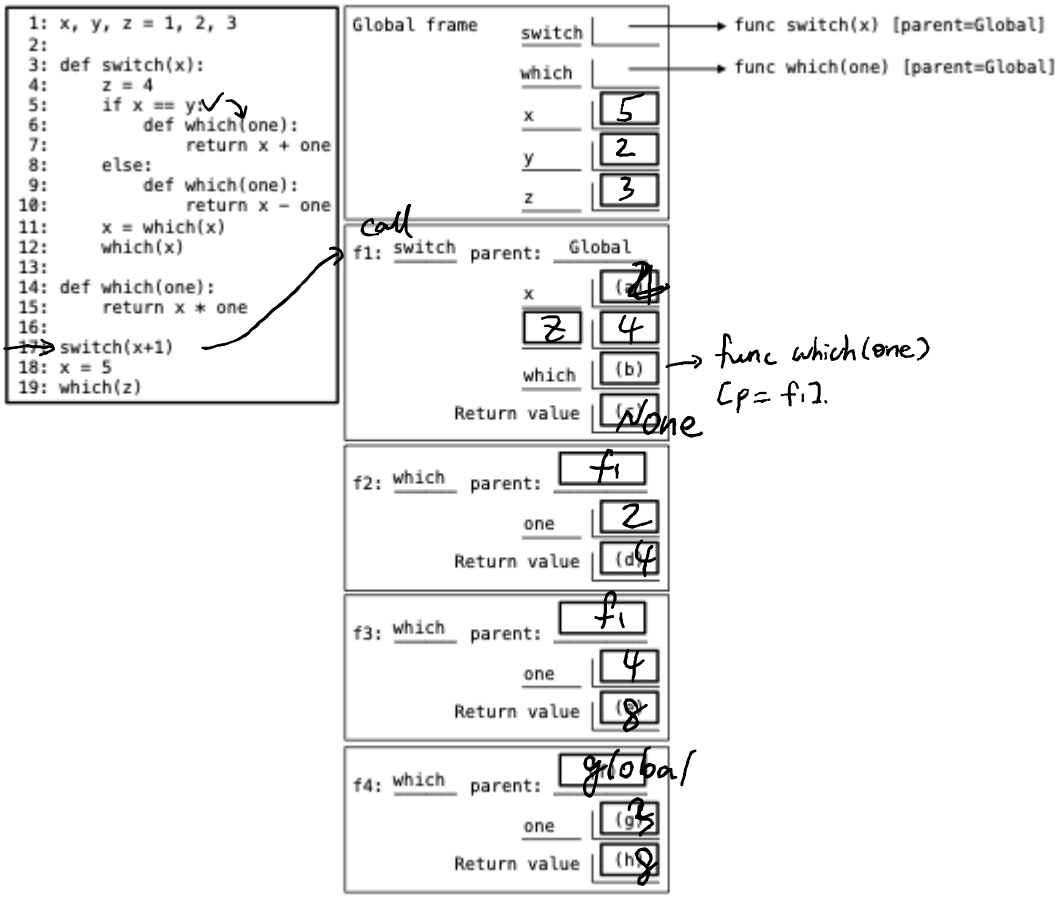
- ○ 1 6 8
- ○ 1 8 6
- ● 6 1 8
- ○ 6 8 1
- ○ 8 1 6
- ○ 8 6 1

168/618

先print才return

2. (8.0 points) Which One

Complete the environment diagram below and then answer the questions that follow. There is one question for each labeled blank in the diagram. The blanks with no labels have no questions associated with them and are not scored.



(a) (1.0 pt) Fill in blank (a).

- ◯ 0
- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4

(b) (1.0 pt) Fill in blank (b).

- ◯ `func which(one) [parent=f1]`
- ◯ `func which(one) [parent=Global]`
- ◯ `func which(one) [parent=switch]`

(c) (1.0 pt) Fill in blank (c).

None

(d) (1.0 pt) Fill in blank (d).

- ◯ `0`
- ◯ `1`
- ◯ `2`
- ◯ `3`
- ◉ `4`

(e) (1.0 pt) Fill in blank (e).

8

(f) (1.0 pt) Fill in blank (f).

- ◉ `Global`
- ◯ `f1`
- ◯ `f2`
- ◯ `f3`
- ◯ `f4`

(g) (1.0 pt) Fill in blank (g).

- ◯ `1`
- ◯ `2`
- ◉ `3`
- ◯ `4`
- ◯ `5`

(h) (1.0 pt) Fill in blank (h).

8

3. (6.0 points) Final Digit

Implement `final_digit`, which takes a non-negative integer `n`. As long as `n` has more than one digit, replace `n` with the sum of the digits of `n`. This process repeats until `n` becomes a single-digit number, which is returned.

```
def final_digit(n):
    """Sum the digits of n repeatedly to reach one digit.

    >>> final_digit(321)       # 3 + 2 + 1 = 6
    6
    >>> final_digit(987)       # 9 + 8 + 7 = 24, and 2 + 4 = 6
    6
    >>> final_digit(989898989) # The digit sum is 77, 7 + 7 = 14, and 1 + 4 = 5
    5
    """
    while n >= 10:
        s = ____0____ ⇐ store sum of each digit of n
                (a)
        while n : ⇐ 遍历n的每一位数字
        -------
          (b)
            n, s = n // 10, _______ 同时进行
                   抛弃个位    (c)
        ___n=s___
          (d)
    return n
```

↖ 当n为0, 把s传入n当作新的n来计算. 直到s只有个位数.

(a) (1.0 pt) Fill in blank (a).

- ☑ `0`
- ○ `n`
- ○ `n % 10`

(b) (1.0 pt) Fill in blank (b).

- ○ `if s`
- ○ `if n`
- ○ `while s`
- ☑ `while n`

(c) (2.0 pt) Fill in blank (c).

- ○ `n`
- ○ `s // 10`
- ○ `n % 10`
- ☑ `s + n % 10`
- ○ `10 * s + n % 10`

(d) (2.0 pt) Fill in blank (d).

n = s

4. (8.0 points) Close Enough

Implement `close`, which takes two non-negative integers `m` and `n`. It returns whether `m` can be changed into `n` by either inserting one digit, removing one digit, or changing one digit. If `m` and `n` are the same number, they are *not* `close`.

```
def close(m, n):
    """Return whether m can result from starting with n and adding, removing,
    or changing one digit.

    >>> close(3756, 3456) and close(3456, 346) and close(346, 3456) and close(456, 56)
    True
    >>> close(5, 5) or close(3456, 3546) or close(3456, 36) or close(34, 3456) or close(345, 456)
    False
    """
    if m < n:
        m, n = n, m
    while m or n:
        if _______:
             (a)
            m, n = _______
                     (b)
        else:
            return _______ or _______  # Hint: check here that just one change is enough
                     (c)        (d)
    return _______
             (e)
```

(a) **(1.0 pt)** Fill in blank (a).

- ○ `m == n`
- ○ `m // 10 == n // 10`
- ○ `m // 10 == n`
- ○ `m == n // 10`
- ✓ `m % 10 == n % 10`

(b) **(2.0 pt)** Fill in blank (b).

- ○ `n, m`
- ✓ `m // 10, n // 10`
- ○ `m // 10, n`
- ○ `m, n // 10`
- ○ `m % 10, n % 10`

(c) (2.0 pt) Fill in blank (c).

- ◯ `m == n`
- ◯ `m - n < 10`
- ◯ `n == 0`
- ✓ `m // 10 == n // 10`
- ◯ `m % 10 == n % 10`
- ◯ `m % 100 == n % 100`

(d) (2.0 pt) Fill in blank (d).

- ✓ `m // 10 == n`
- ◯ `m // 10 - n < 10`
- ◯ `n == 0`
- ◯ `m // 10 == 0`
- ◯ `m // 100 == n // 10`
- ◯ `(m // 10) % 10 == n % 10`

(e) (1.0 pt) Fill in blank (e).

- ◯ `m == n`
- ◯ `m == 0`
- ◯ `n == 0`
- ◯ `True`
- ✓ `False`

5. (10.0 points) Shifty

(a) (4.0 points)

Implement `shift`, which takes a number `k` and a one-argument function `f`. It returns a one-argument function `g` that takes a number `x`. For all numbers `x`, `g(x)` is equal to `f(x + k)`.

```
def shift(k, f):
    """Return a function of x that returns f(x+k).

    >>> square = lambda x: x * x
    >>> g = shift(2, square)
    >>> g(3)     # square(3 + 2)
    25
    """
    return _______
           (a)
```

lambda x: f(x+k)

i. (2.0 pt) Fill in blank (a).

- ◯ `f(x) + k`
- ◯ `f(x + k)`
- ◯ `f(lambda x: x + k)`
- ◯ `f(lambda x: x) + k`
- ◯ `lambda x: f(x) + k`
- ✓ `lambda x: f(x + k)`
- ◯ `(lambda x: f(x))(x + k)`
- ◯ `g(x) + k`
- ◯ `g(x + k)`
- ◯ `g(lambda x: x + k)`
- ◯ `g(lambda x: x) + k`
- ◯ `lambda x: g(x) + k`
- ◯ `lambda x: g(x + k)`
- ◯ `(lambda x: g(x))(x + k)`

ii. (2.0 pt) Provide an alternate solution for blank (a) . This time, your solution **must** call `compose` (defined below), and `f` **may not** be the operator of a call expression. In other words, you can't write `f(` in your answer. You **may not** write `[` or `if`.

```
def compose(f, g):
    """Return a function that takes x and calls f on g of x."""
    return lambda x: f(g(x))
```

compose(f, lambda x: x+k)

(b) (6.0 points)

Implement `sum_range`, which takes positive integers `p` and `q` with `p <= q`, as well as a one-argument function `term`. It returns the sum of the return values of `term` called on each consecutive integer starting with `p` and ending with `q` (including both `p` and `q`). You may call `shift`, `summation`, and `compose`. Assume `shift` is implemented correctly.

```
def summation(n, term):
    """Sum the first n terms of a sequence: term(1) + term(2) + ... + term(n).

    >>> summation(5, lambda x: x*x)  # 1*1 + 2*2 + 3*3 + 4*4 + 5*5
    55
    """
    total, k = 0, 1
    while k <= n:
        total, k = total + term(k), k + 1
    return total

def sum_range(p, q, term):
    """Sum terms p through q of a sequence: term(p) + term(p+1) + ... + term(q).

    >>> sum_range(1, 5, lambda x: x*x)   # 1*1 + 2*2 + 3*3 + 4*4 + 5*5
    55
    >>> sum_range(4, 5, lambda x: x*x)   # 4*4 + 5*5
    41
    >>> sum_range(5, 5, lambda x: x*x)   # 5*5
    25
    """
    assert p <= q
    return _______ ( _______ , _______ )
             (a)       (b)       (c)
```

terms ↓ shift(p-1) term

i. (1.0 pt) Fill in blank (a).

- ◉ `summation`
- ○ `shift`
- ○ `compose`
- ○ `term`

ii. (2.0 pt) Fill in blank (b).

- ○ `p - q - 1`
- ○ `p - q`
- ○ `p - q + 1`
- ○ `q - p - 1`
- ○ `q - p`
- ◉ `q - p + 1`

iii. (3.0 pt) Fill in blank (c).

(c) (0.0 points)

This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

The `shifter` function below is a curried version of `shift`.

Implement `unshift`, which takes the result of `shifter(k)` for some number `k`. It returns a function that takes the result of `shifter(k)(f)` for some function `f` and returns a function equivalent to `f`. That is:

`f(x) == unshift(shifter(k))(shifter(k)(f))(x)`

You can write your answer on multiple lines if it's long. You can abbreviate `lambda` using the greek symbol lambda.

Your answer **must** be a call to `shift`. You may also call any of `compose`, `summation`, or `sum_range`.

Hint: If you can compute `k`, then you can `shift` backward by `k` to undo the original shift.

```
def shifter(k):
    def shifted(f):
        return shift(k, f)
    return shifted

def unshift(shifted):
    """Assume shifted is the return value of shifter(k) for some k.

    >>> cubic = lambda x: x*x*x - 5*x*x + 1   # Some complicated function
    >>> cubic(2.0)
    -11.0
    >>> cubic(9.0)
    325.0
    >>> do = shifter(4)
    >>> do(cubic)(2.0)        # same as cubic(6.0)
    37.0
    >>> do(cubic)(9.0)        # same as cubic(13.0)
    1353.0
    >>> undo = unshift(do)
    >>> undo(do(cubic))(2.0)  # same as cubic(2.0)
    -11.0
    >>> undo(do(cubic))(9.0)  # same as cubic(9.0)
    325.0
    """
    return lambda g: _______
```

i. (0.0 pt) Fill in the blank with **a single call to `shift`**.

No more questions.