CS 61A Spring 2025 — Structure and Interpretation of Computer Programs

Midterm 1 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?

(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. (6.0 points) What Would Python Print?

Answer the questions about this code. No errors occur while it is executed.

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

def square(f):
    return lambda x: f(x) * f(x)

def inc(f):
    return lambda x: f(x + 1)

def triple(f):
    return lambda x: f(f(f(x)))

def put(x):
    return lambda f: f(x)

one = put(1)
triple(print)(5)
```

(a) (2.0 pt) What is displayed by the last line?

```
5
None
None
```

(b) (2.0 pt) What would be displayed by evaluating `print(one(square(double)))`

```
4
```

(c) (2.0 pt) What would be displayed by evaluating `print(inc(put)(1)(triple(double)))`

```
16
```

(d) **(0.0 pt) 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!**

Write one line of code just after `triple(print)(5)` that would double and add one to all numbers in the answers to parts (b) and (c) above. For example, for an answer containing 100, it would now contain 201 instead.

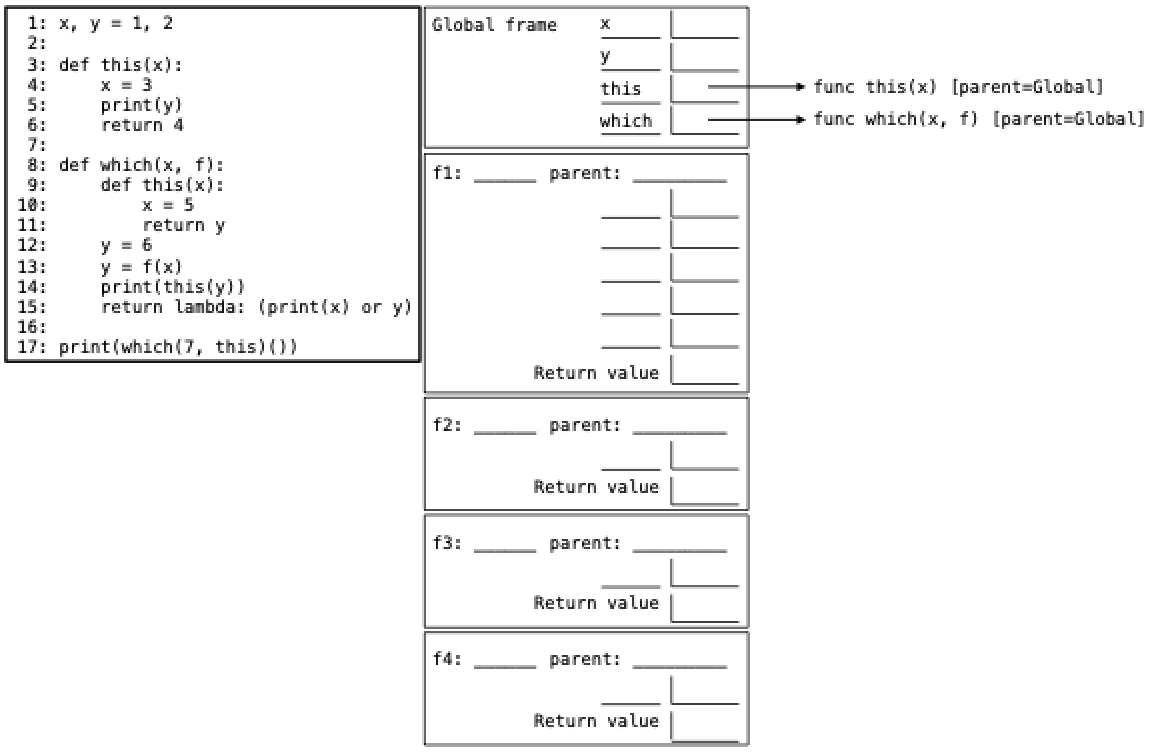
You **may not** use any numbers or arithmetic symbols (such as `+` or `*`) or arithmetic functions (such as `add` or `mul`). You **may not** reassign `triple` or `square`. You may use multiple assignment: `x, y = ___, ___`

Four possible answers:

```
print = (lambda f: lambda x: f(double(x)))(inc(print))
print = (lambda f: lambda x: inc(f)(double(x)))(print)
f, print = inc(print), lambda x: f(double(x))
f, print = print, lambda x: inc(f)(double(x))
```

2. (8.0 points) Which One

Complete the environment diagram below to answer the questons about the calls to `print`. Only the questions will be scored, not the diagram. Each call to `print` is evaluated once, and there are no errors caused by running this code.



```
 1: x, y = 1, 2
 2:
 3: def this(x):
 4:     x = 3
 5:     print(y)
 6:     return 4
 7:
 8: def which(x, f):
 9:     def this(x):
10:         x = 5
11:         return y
12:     y = 6
13:     y = f(x)
14:     print(this(y))
15:     return lambda: (print(x) or y)
16:
17: print(which(7, this)())
```

Global frame
x
y
this
which
func this(x) [parent=Global]
func which(x, f) [parent=Global]
f1: ______ parent: ________
Return value
f2: ______ parent: ________
Return value
f3: ______ parent: ________
Return value
f4: ______ parent: ________
Return value

(a) (2.0 pt) What is displayed by the call to `print` on line 5?

2

(b) (2.0 pt) What is displayed by the call to `print` on line 14?

4

(c) (2.0 pt) What is displayed by the call to `print` on line 15?

7

(d) (2.0 pt) What is displayed by the call to `print` on line 17?

4

3. (13.0 points) Legit Digit

(a) (4.0 points)

Implement `all_digits`, which takes a positive integer `n` and one-argument function `cond` that always returns `True` or `False`. It returns `True` if `cond(d)` returns `True` when called on every digit `d` in `n`, and `False` otherwise.

```
def all_digits(n, cond):
    """Return whether cond returns true for every digit of positive n.

    >> odd = lambda d: d % 2 == 1
    >>> all_digits(123, odd)  # not all digits are odd
    False
    >>> all_digits(357, odd)  # all digits are odd
    True
    """

    while n:

        if _______:
             (a)

            _______
              (b)

        n = n // 10

    return _______
             (c)
```

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

`not cond(n % 10)`

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

- ○ `n = n % 10`
- ○ `n = n * 10`
- ○ `return True`
- [✓ `return False`]
- ○ `return odd`
- ○ `return cond`

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

- ○ `n`
- [✓ `True`]
- ○ `False`
- ○ `odd(n)`
- ○ `cond(n)`

(b) (3.0 points)

Definition. A *prefix* of a positive integer `n` is the value of `n//pow(10, p)` for some non-negative integer `p`. For example, 3456 has prefixes 3456, 345, 34, 3, and 0.

Implement `prefix_digits`, which takes a positive integer `n` and a one-argument function `cond` that always returns `True` or `False`. It returns the largest prefix of `n` for which `cond` returns `True` when called on every digit of the prefix. You may call `all_digits` and `process`.

```
def process(n, check):
    """A function to help implement prefix_digits."""
    while n:
        if check(n):
            return n
        n = n // 10
    return 0

def prefix_digits(n, cond):
    """Return the largest prefix of positive n for which cond returns true for every digit.

    >>> odd = lambda d: d % 2 == 1
    >>> prefix_digits(94720, odd)
    9
    >>> prefix_digits(919321, odd)
    9193
    >>> prefix_digits(2025, odd)
    0
    >>> prefix_digits(20252025, lambda d: d < 4)
    202
    >>> prefix_digits(20252025, lambda d: True)
    20252025
    """
    return process(n, lambda k: _______ )
                                  (d)
```

i. (3.0 pt) Fill in blank (d).

- ○ `all_digits(n, cond)`
- [✓ `all_digits(k, cond)`]
- ○ `all_digits(n, cond(n))`
- ○ `all_digits(k, cond(k))`
- ○ `lambda cond: all_digits(n, cond)`
- ○ `lambda cond: all_digits(k, cond)`
- ○ `lambda cond: all_digits(n, cond(n))`
- ○ `lambda cond: all_digits(k, cond(k))`
- ○ `all_digits(n, lambda n: cond)`
- ○ `all_digits(k, lambda n: cond)`
- ○ `all_digits(n, cond(lambda n: n))`
- ○ `all_digits(k, cond(lambda n: k))`

(c) (6.0 points)

Re-implement `prefix_digits`, which takes a positive integer `n` and a one-argument function `cond` that always returns `True` or `False`. It returns the largest prefix of `n` for which `cond` returns `True` when called on every digit of the prefix. You **may not** call `all_digits` or `process`.

```
def prefix_digits(n, cond):
    """Return the largest prefix of positive n for which cond returns true for every digit.

    >>> odd = lambda d: d % 2 == 1
    >>> prefix_digits(94720, odd)
    9
    >>> prefix_digits(919321, odd)
    9193
    >>> prefix_digits(2025, odd)
    0
    >>> prefix_digits(20252025, lambda d: d < 4)
    202
    """
    k = 0
    while n >= ________:
                  (e)
        if cond(_______):
                  (f)
            k += 1
        else:
            n = n // 10
            _______
              (g)
    return n
```

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

○ `0`

○ `k`

○ `k // 10`

[✓ `pow(10, k)`]

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

○ `n % 10`

○ `n // 10 % 10`

○ `n % pow(10, k)`

○ `n // pow(10, k)`

○ `n // 10 % pow(10, k)`

[✓ `n // pow(10, k) % 10`]

iii. (2.0 pt) Fill in blank (g). You may not call `all_digits` or `process`.

k = 0

4. (4.0 points) Hailstone Returns

Definition. A *hailstone sequence* is formed by picking a positive integer `n` as the start then repeatedly updating `n` until it is 1 using this rule: if `n` is even, divide it by 2; if `n` is odd, multiply it by 3 and add 1.

Implement `hailstone`, which prints each update in a hailstone sequence by displaying: the update number (counting up from 1), the current `n`, an `->` arrow symbol, and the updated `n`.

This implementation may not be possible using the template. If that's the case, respond *Impossible* to the following questions.

```
def hailstone(n):
    """Print numbered updates in the hailstone sequence.

    >>> hailstone(10)
    1 10 -> 5
    2 5 -> 16
    3 16 -> 8
    4 8 -> 4
    5 4 -> 2
    6 2 -> 1
    """
    def f():
        if n % 2 == 1:
            m = 3 * n + 1
        else:
            m = n // 2
        -------
          (a)
        -------
          (b)
    k = 1
    while n > 1:
        k, n = k + 1, f()
```

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

- ○ `print(k, n, ->, m)`
- ● `print(k, n, '->', m)`
- ○ `print(k - 1, n, ->, m)`
- ○ `print(k - 1, n, '->', m)`
- ○ *Impossible*

(b) (2.0 pt) Fill in blank (b). If the problem is not possible using the template, write *Impossible*.

return m

5. (4.0 points) It Takes Two

Implement `at_least_two`, which takes three arguments. It returns `True` if at least two of them are true values and `False` otherwise. The built-in `bool` function takes one argument and returns `True` if it is a true value and `False` if it is a false value.

```
def at_least_two(x, y, z):
    """Returns whether at least two of the arguments are true values.
    >>> at_least_two(1 + 1, 3 + 3, 5 + 5)
    True
    >>> at_least_two(1 + 1, 3 - 3, 5 + 5)
    True
    >>> at_least_two(1 - 1, 3 + 3, 5 + 5)
    True
    >>> at_least_two(1 - 1, 3 + 3, 0)
    False
    >>> at_least_two(1 - 1, 0, 0)
    False
    """
    if _______:
         (a)
        return _______
    else:        (b)
        return _______
                 (c)
```

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

- [✓ `x`]
- ○ `x == True`
- ○ `x and y`
- ○ `x == True and y == True`

(b) (1.0 pt) Fill in blank (b). *Select all correct answers.*

- ☐ `y or z`
- ☐ `y and z`
- [✓ `bool(y or z)`]
- ☐ `bool(y and z)`
- ☐ `not(y or z)`
- ☐ `not(y and z)`

(c) (1.0 pt) Fill in blank (c). *Select all correct answers.*

- ☐ `y or z`
- ☐ `y and z`
- ☐ `bool(y or z)`
- [✓ `bool(y and z)`]
- ☐ `not(y or z)`
- ☐ `not(y and z)`

No more questions.