CS 61A Fall 2018 — Structure and Interpretation of Computer Programs

MIDTERM 1

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one hand-written 8.5" × 11" crib sheet of your own creation and the official CS 61A midterm 1 study guide.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.

Last name	
First name	
Student ID number	
CalCentral email (`_@berkeley.edu`)	
TA	
Name of the person to your left	
Name of the person to your right	
All the work on this exam is my own. **(please sign)**	

POLICIES & CLARIFICATIONS

- If you need to use the restroom, bring your phone and exam to the front of the room.
- You may use built-in Python functions that do not require import, such as `min`, `max`, `pow`, `len`, and `abs`.
- You **may not** use example functions defined on your study guides unless a problem clearly states you can.
- For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.

1. (10 points) What Would Python Display

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. If an error occurs, write "Error", but include all output displayed before the error. If evaluation would run forever, write "Forever". To display a function value, write "Function". The first two rows have been provided as examples.

The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

Assume that you have first started `python3` and executed the statements on the left.

```
identity = lambda x: x
increment = lambda x: x + 1


def fif(c, t, f, x):
    if c(x):
        return t(x)
    else:
        return f(x)


def bounce(x, y):
    while x < y:
        if x <= (y and x):
            print('a')
        if x > 0:
            print('b')
        elif x > -5:
            print('c')
        x, y = -y, increment(x - y)
        print(y)


crazy = lambda rich: 100 * rich
crazy = lambda rich: crazy(crazy(rich))


def ok(py):
    def h(w):
        print(py // 10)
        return ok(py)
    return lambda h: h(py)
```

Expression	Interactive Output
`pow(10, 2)`	100
`print(4, 5) + 1`	4 5 Error
`print(None, print(1, 2))`	
`fif(abs, print, print, -2)`	
`bounce(1, 2)`	
`crazy(88)`	
`ok(314)(identity)`	

2. (4 points) Implement `factorial`, which computes the factorial of a positive integer n. You may use any of the functions defined above. **You may <u>not</u> write `if`, `else`, `and`, `or`, or `lambda` in your solution.**

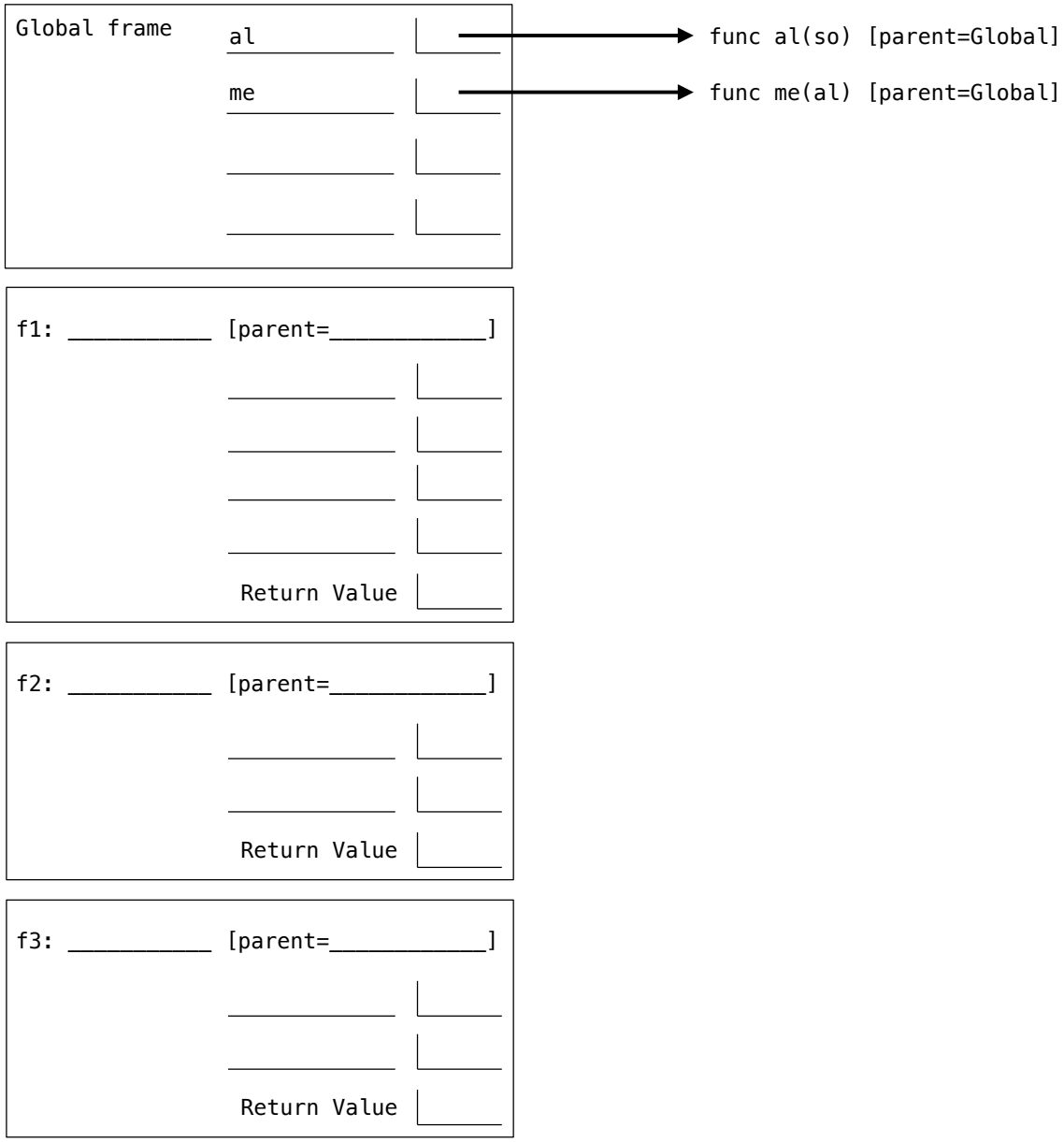
```
def factorial(n):
    """The factorial of positive n: n * (n-1) * (n-2) * ... * 1

    >>> factorial(4)  # 4 * 3 * 2 * 1
    24
    >>> factorial(1)
    1
    """
    return n * fif(_________________, _________________, _________________, _________________)
```

3. (6 points) Also Some Meal

Fill in the environment diagram that results from executing the code on the right until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.* A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
1  def al(so):
2      me = 1
3      def al(to):
4          return so + me
5      so = 2
6      return al
7
8  def me(al):
9      me = 3
10     return al(lambda: 4) + so
11
12 so = 5
13 so = me(al(6)) + so
```



4. (5 points) Getting Rect

Implement `rect`, which takes two positive integer arguments, `perimeter` and `area`. It returns the **integer** length of the longest side of a rectangle with integer side lengths ℓ and h which has the given perimeter and area. If no such rectangle exists, it returns `False`.

The perimeter of a rectangle with sides ℓ and h is $2\ell + 2h$. The area is $\ell \cdot h$.

Hint: The built-in function `round` takes a number as its argument and returns the nearest integer. For example, `round(2.0)` evaluates to `2`, and `round(2.5)` evaluates to `3`.

```
def rect(area, perimeter):
    """Return the longest side of a rectangle with area and perimeter that has integer sides.

    >>> rect(10, 14)  # A 2 x 5 rectangle
    5
    >>> rect(5, 12)   # A 1 x 5 rectangle
    5
    >>> rect(25, 20)  # A 5 x 5 rectangle
    5
    >>> rect(25, 25)  # A 2.5 x 10 rectangle doesn't count because sides are not integers
    False
    >>> rect(25, 29)  # A 2 x 12.5 rectangle doesn't count because sides are not integers
    False
    >>> rect(100, 50) # A 5 x 20 rectangle
    20
    """


    side = 1


    while side * side ____________________________________________________________________ area:


        other = round(_____________________________________________________________________________)


        if ___________________________________________________________________________________:


            ___________________________________________________________________________


        side = side + 1


    return False
```

5. (6 points) Dig It

Implement `sequence`, which takes a positive integer `n` and a function `term`. It returns an integer whose digits show the n elements of the sequence `term(1)`, `term(2)`, ..., `term(n)` in order. Assume the `term` function takes a positive integer argument and returns a positive integer.

Important: You may not use `pow`, `**`, `log`, `str`, or `len` in your solution.

```
def sequence(n, term):
    """Return the first n terms of a sequence as an integer.

    >>> sequence(6, abs)                  # Terms are 1, 2, 3, 4, 5, 6
    123456
    >>> sequence(5, lambda k: k+8)        # Terms are 9, 10, 11, 12, 13
    910111213
    >>> sequence(4, lambda k: pow(10, k)) # Terms are 10, 100, 1000, 10000
    10100100010000
    """


    t, k = 0, 1


    while ____________________________________________________________________________________:


        m = 1


        x = ____________________________________________________________________________


        ____________________________________________________________________________ m <= x:


            ____________________________________________________________________________


        t = ____________________________________________________________________________


        k = k + 1


    return t
```

6. (9 points) This Again?

Definitions. A *repeatable integer* function takes an integer argument and returns a repeatable integer function.

(a) (6 pt) Implement `repeat`, which is a repeatable integer function that detects repeated arguments. As a side effect of repeated calls, it prints each argument that has been used before in a sequence of repeated calls. Therefore, if an argument appears n times, it is printed $n - 1$ times in total, each time other than the first. The `detector` function is part of the implementation of `repeat`; you must determine how it is used.

Important: You may <u>not</u> use a list, set, or any other data type not covered yet in the course.

```
def repeat(k):
    """When called repeatedly, print each repeated argument.

    >>> f = repeat(1)(7)(7)(3)(4)(2)(5)(1)(6)(5)(1)
    7
    1
    5
    1
    """

    return ____________________________________________________________________(k)


def detector(f):

    def g(i):


        if ____________________________________________________________________:


            ____________________________________________________________________


        return ____________________________________________________________________


    return g
```

(b) (3 pt) Implement `repeat_digits`, which takes a non-negative integer `n`. For the digits of `n` from right to left, it prints each digit that also appears somewhere to its right. Assume `repeat` is implemented correctly.

```
def repeat_digits(n):
    """Print the repeated digits of non-negative integer n.

    >>> repeat_digits(581002821)
    2
    0
    1
    8
    """

    f = ____________________________________________________________________________

    while n:

        f, n = ___________________________________ , ___________________________________
```