CS 61A
Fall 2017

Structure and Interpretation of Computer Programs

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)**	

1. (10 points) I Wonder What Python Would 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. 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.

```
aaron, burr = 2, 5
aaron, burr = 4, aaron + 1

hamil = 10

def alex(hamil):
    def g(w):
        hamil = 2 * w
        print(hamil, w)
        w = hamil
        return hamil
    w = 5
    alex = g(w + 1)
    print(w, alex, hamil)

def el(i, za):
    def angelica():
        return i + 1
    if i > 10:
        return za()
    elif i > 4:
        print(angelica())
        return el(i * i, za)
    else:
        return el(i * i,
                  angelica)

K = lambda x: lambda y: x

def pr(x):
    print(x)
    return x
```

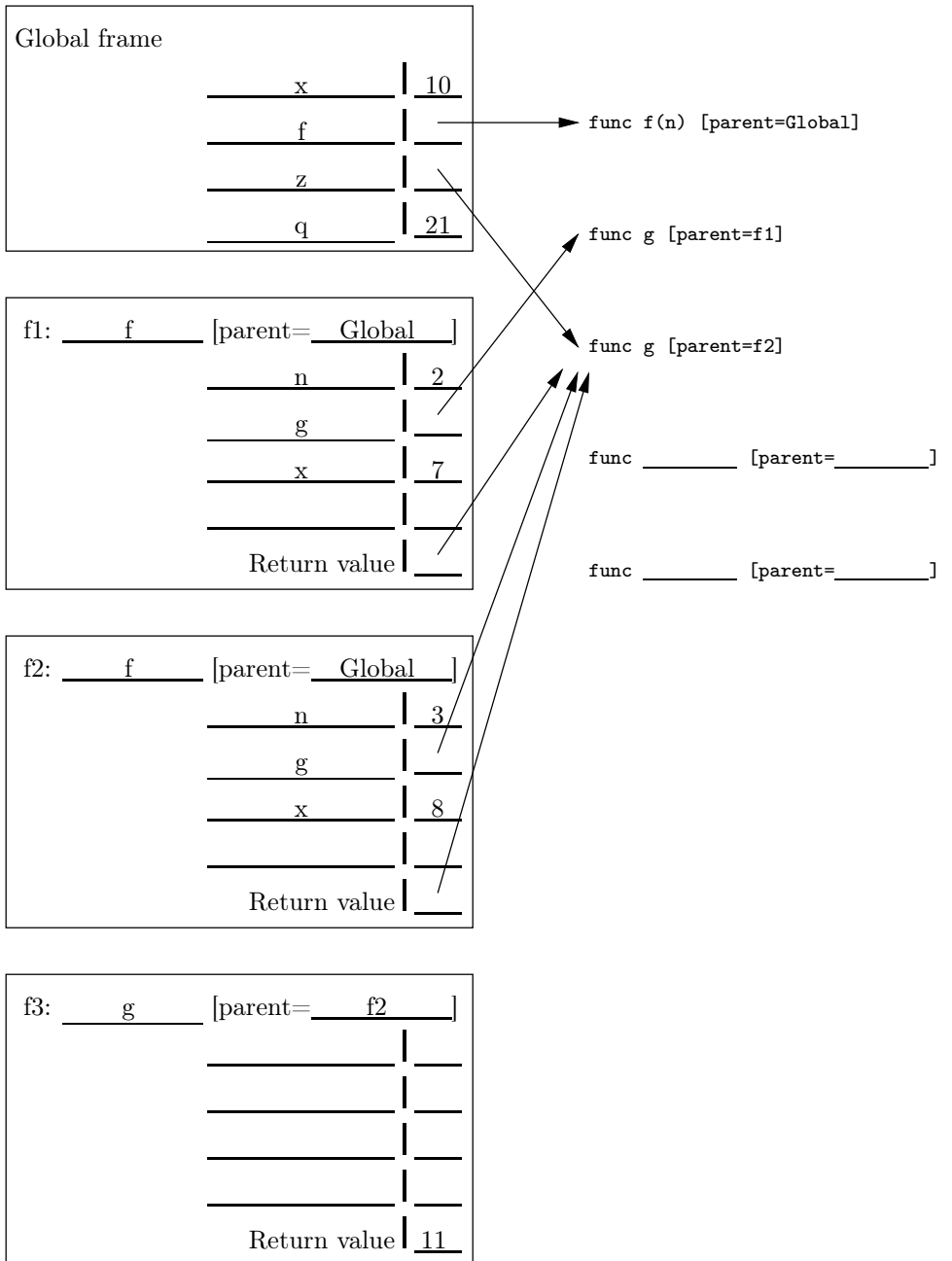| Expression	Interactive Output
pow(2, 3)	8
print(4, 5) + 1	4 5 Error
print(aaron, burr)	4 3
alex(3)	12 6 5 12 3
el(3, el)	10 4
K	Function
K(3)	Function
K(3)(2)	3
pr(True) and pr(0) and pr(1)	True 0 0

2. (8 points)   Environmental Influences

Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces, frames, or function values.*

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   x = 1
2   def f(n):
3       def g():
4           return n + x
5       x = n + 5
6       if n % 3 == 0:
7           return g
8       else:
9           return f(n + 1)
10  x = 10
11  z = f(2)
12  q = x + z()
```

Global frame

x	10
f	
z	
q	21

func f(n) [parent=Global]

func g [parent=f1]

func g [parent=f2]

f1: _____f_____ [parent=____Global____]

n	2
g	
x	7
Return value	

func _____ [parent=_____]

f2: _____f_____ [parent=____Global____]

n	3
g	
x	8
Return value	

func _____ [parent=_____]

f3: _____g_____ [parent=____f2____]

Return value	11

3. (3 points) Triangulate

It's easy to see that in any triangle, each side must be shorter than the sum of the other two. Implement `triangle`, which takes three positive numbers, `a`, `b`, and `c`, and returns whether these three numbers could possibly be the lengths of the three sides of a triangle.

```python
def triangle(a, b, c):
    """Return whether a, b, and c could be the legs of a triangle.

    >>> triangle(3, 4, 5)
    True
    >>> triangle(3, 4, 6)
    True
    >>> triangle(6, 3, 4)
    True
    >>> triangle(3, 6, 4)
    True
    >>> triangle(9, 2, 2)
    False
    >>> triangle(2, 4, 2)
    False
    """




    longest = max(a, b, c)



    sum_of_others = a + b + c - longest  # or min(a+b, a+c, b+c)



    return longest < sum_of_others
```

4. (9 points)   Digital

(a) **(3 pt)** Implement `collapse`, which takes a non-negative integer, and returns the result of removing all
digits from it that duplicate the digit immediately to their right.

```python
def collapse(n):
    """For non-negative N, the result of removing all digits that are equal
    to the digit on their right, so that no adjacent digits are the same.

    >>> collapse(1234)
    1234
    >>> collapse(12234441)
    12341
    >>> collapse(0)
    0
    >>> collapse(3)
    3
    >>> collapse(11200000013333)
    12013
    """

    left, last = n // 10, n % 10

    if left == 0:

        return last

    elif last == left % 10:

        return collapse(left)

    else:

        return collapse(left) * 10 + last
```

(b) (6 pt) Implement `find_pair`, which takes a two-argument function, `p`, as input and returns another function. The returned function takes a non-negative integer `n`; it returns `True` if and only if `p` returns a true value when called on at least one pair of adjacent digits in `n`, and `False` otherwise.

```python
def find_pair(p):
    """Given a two-argument function P, return a function that takes a
    non-negative integer and returns True if and only if two adjacent digits
    in that integer satisfy P (that is, cause P to return a true value).

    >>> z = find_pair(lambda a, b: a == b)   # Adjacent equal digits
    >>> z(1313)
    False
    >>> z(12334)
    True
    >>> z = find_pair(lambda a, b: a > b)
    >>> z(1234)
    False
    >>> z(123412)
    True
    >>> find_pair(lambda a, b: a <= b)(9753)
    False
    >>> find_pair(lambda a, b: a == 1)(1)    # Only one digit; no pairs.
    False
    """

    def find(n):


        while n >= 10:


            if p((n // 10) % 10, n % 10):


                return True


            else:


                n = n // 10


        return False


    return find
```

5. (10 points)   Please Confirm

Definition. A *confirming function* for a sequence of digits, called a *code*, takes a single digit as its only argument. If the digit does not match the first (left-most) digit of the code to be confirmed, it returns `False`. If the digit does match, then the confirming function returns `True` if the code has only one digit, or another confirming function for the rest of the code if there are more digits to confirm.

(a) (5 pt) Implement `confirmer` so that when `confirmer` takes a positive integer `code`, it returns a confirming function for the digits of that `code`.

```
def confirmer(code):
    """Return a confirming function for CODE.

    >>> confirmer(204)(2)(0)(4) # The digits of 204 are 2, then 0, then 4.
    True
    >>> confirmer(204)(2)(0)(0) # The third digit of 204 is not 0.
    False
    >>> confirmer(204)(2)(1)    # The second digit of 204 is not 1.
    False
    >>> confirmer(204)(20)      # The first digit of 204 is not 20.
    False
    """
    def confirm1(d, t):
        def result(digit):
            if d == digit:
                return t
            else:
                return False
        return result

    def extend(prefix, rest):
        """Return a confirming function that returns REST when given the digits of PREFIX.
        For example, if c = extend(12, confirmer(34)), then c(1)(2) returns confirmer(34),
        so that c is a confirming function for 1234."""

        left, last = prefix // 10, prefix % 10



        if prefix < 10:



            return confirm1(prefix, rest)



        else:



            return extend(left, confirm1(last, rest))  # or extend(left, extend(last, rest))



    return extend(code, True)
```

(b) (5 pt) Given a confirming function, one can find the code it confirms, one digit at a time. Implement `decode`, which takes a confirming function `f` and returns the code that it confirms.

```python
from confirm_soln import confirmer

def decode(f, y=0):
    """Return the code for a confirming function f.

    >>> decode(confirmer(12001))
    12001
    >>> decode(confirmer(56789))
    56789
    """

    d = 0

    while d < 10:

        x, code = f(d), 10 * y + d

        if x == True:

            return code

        elif x == False:

            d = d + 1

        else:

            return decode(x, code)
```