CS 7 Introduction to Programming and Computer Science

SUMMER 2023	MIDTERM
EXAM VERSION	FIRE NATION
DATE	JUNE 18
TIME	4:10 TO 6:10 PM CAT

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is open book, open notes, closed computer. You may consult any books, notes, or other non-responsive objects available to you.
- There are 4 questions in this exam all worth 40 points. The midterm is worth 10 percent of the total grade.
- Answer on the separate answer sheet. You may use a scratch for your work but make sure to transfer the solutions to the answer sheet. Work not in the answer sheet will not be graded.
- After completing this exam, you will have 10 minutes to scan and upload your answer sheet to the midterm assignment on Gradescope.

SOLUTIONS

Be warned: Computer Science exams are known to cause panic. Fortunately, this reputation is entirely unjustified. Just read all the questions carefully to begin with and first try to answer those parts about which you feel most confident. Do not be alarmed if some of the answers are obvious. Should you feel an attack of anxiety coming on, feel free to jump up and run around the outside of your building once or twice.

1. (11 points) Mudiwa Janet 💔🎸

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".
Hint: No answer requires more than 5 lines. (It's possible that all of them require even fewer.)
The first two rows have been provided as examples.
Recall: The interactive interpreter displays the value of a successfully evaluated expression, unless it is None
Assume that you have started python3 and executed the following statements:

```
from operator import sub

def ndiwe(here):
    return print(here , here)

def janet(mudiwa):
    usambochema, joe = ndiwe , print
    joe(usambochema(mudiwa))
    return vana(mudiwa)

def vana(vana):
    if vana:
        return vana + vana
    elif usambochema(vana)(print)(print):
        return 1000
    else :
        return joe(3)

joe = vana
usambochema = lambda v : lambda a : lambda n : n (5 , a (v))
```

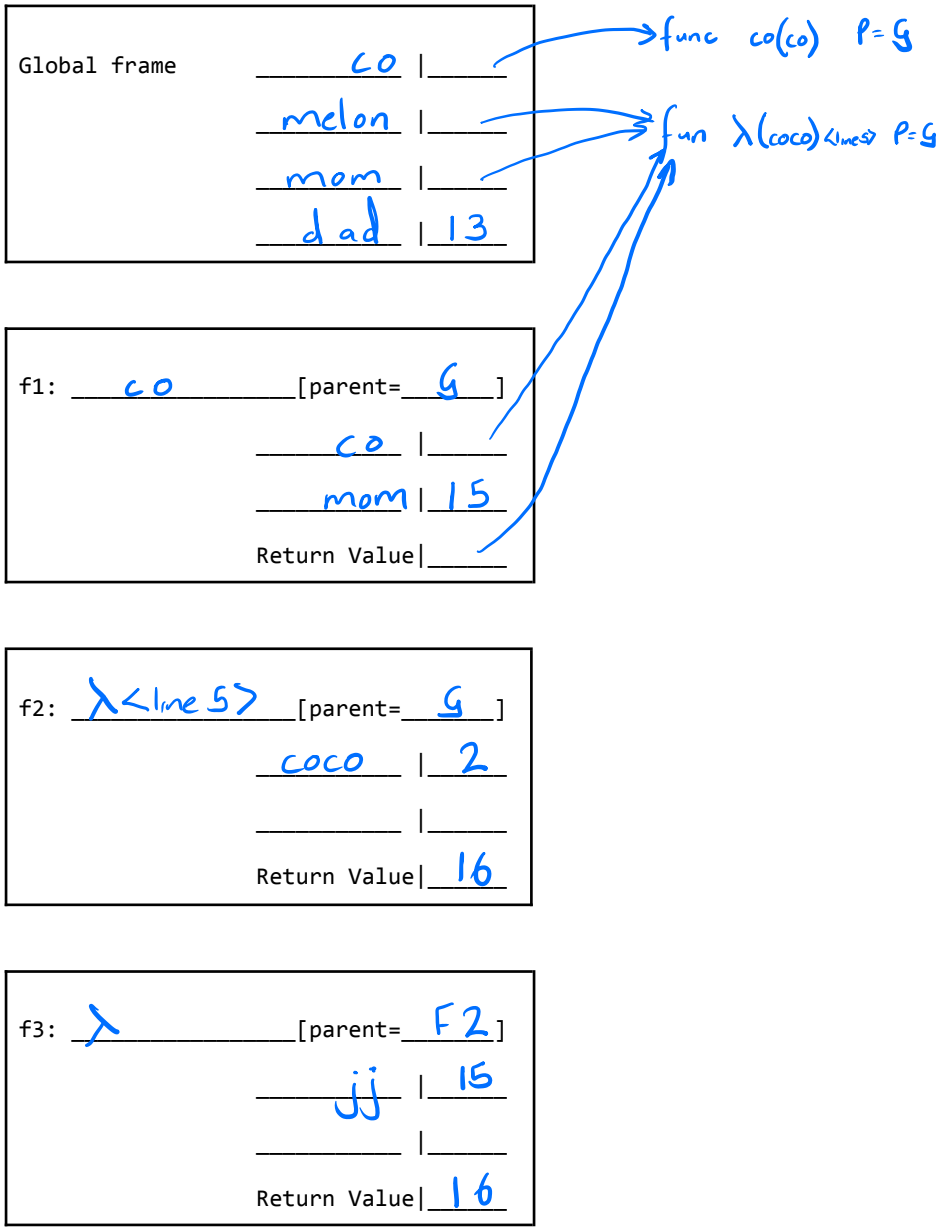
Question	Expression	Interactive Output
	`pow(3, 4)`	`81`
	`print(2, 0)`	`2 0`
A	`print(ndiwe(2+3), print(7))`	5 5 7 None None
B	`janet(1)`	1 1 None 2
C	`janet(joe(7))`	14 14 None 28
D	`usambochema(1)(vana)(min)`	2
E	`vana(print(3))`	3 None 5 None 6
F	`usambochema(0)(joe)(sub)`	0 5 None -1

2. (12 points) Unono ngowami

a. **(6 pt)** 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 or frames.
A complete answer will:

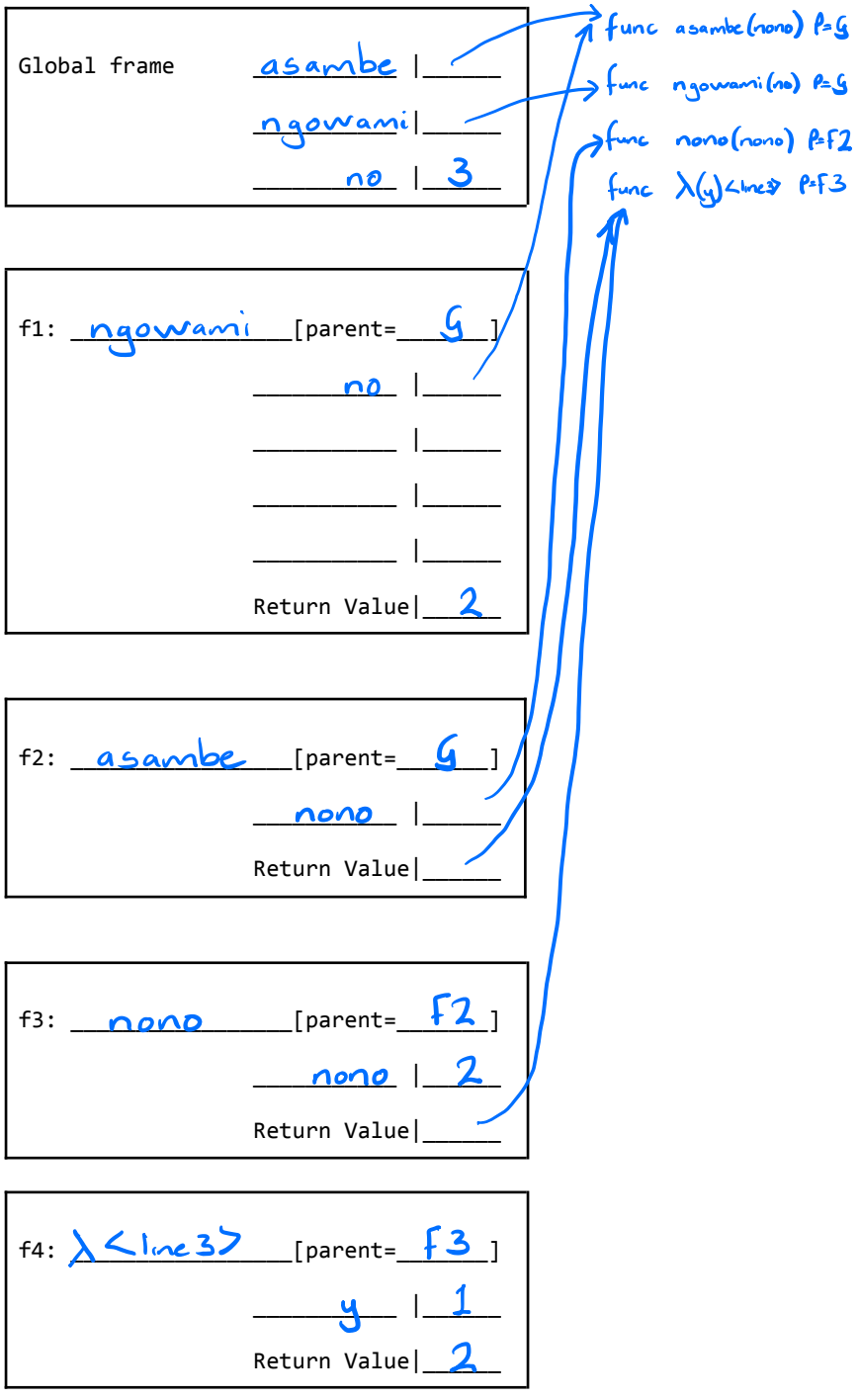
- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
1  def co(co):
2      mom = 15
3      if co(2) > mom:
4          return co
5  melon = lambda coco: (lambda jj: dad+3)(mom+1)
6  mom, dad = 14, 13
7  mom = co(melon)
```



b. **(6 pt)** 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 or frames. The<line ...> annotation in a lambda value gives the line in the Python source of a lambda expression.

```
1  def asambe(nono):
2      def nono(nono):
3          return lambda y: nono
4      return nono(2)
5
6  def ngowami(no):
7      return no(no)(1)
8
9  no = 3
10 ngowami(asambe)
```



3. (10 points) CS 7 : A feast of the Sevens 🧮

a. **(6 pt)** The standard number representation system is the decimal system, where each digit in a number represents a power of ten. The right-most digit is the ones' place, the next digit is the tens' place, etc.

In the septimal system, each digit in a number represents a power of seven. The right-most digit is still the 1's place, but the next digit is the `7`'s place, the next digit is the 49's place, etc. Each digit ranges from `0-6`, so septimal numbers will never contain the digits `7`, `8` or `9`.

To convert a number represented in septimal to a number represented in decimal, each digit must be multiplied by the appropriate power of seven. For example, `123` is actually `(1 * 49) + (2 * 7) + (3 * 1)`, resulting in a decimal representation of `66`.
The diagram visualizes the equivalence between the septimal and decimal numbers:

Septimal				Decimal	
1	2	3	→	6	6
7^2	7^1	7^0		10^1	10^0
49	7	1		10	6

Implement `convert_to_decimal`, which takes a septimal l number and returns the decimal equivalent. The septimal number will always start with a non-`0` digit, and the number will always be positive

```
def convert_to_decimal(septimal):
    """
    >>> convert_to_decimal(3) # (7^0 * 3)
    3
    >>> convert_to_decimal(23) # (7^1 * 2) + (7^0 * 3)
    17
    >>> convert_to_decimal(123) # (7^2 * 1) + (7^1 * 2) + (7^0 * 3)
    66
    """
    decimal = 0
    curr_place = _____1_____
    while septimal>0:
        curr_digit = septimal % 10
        decimal = decimal + (curr_digit * curr_place)
        curr_place = curr_place * 7
        septimal = septimal // 10
    return decimal
```

(4 pt) Implement `forbid_digit`, a higher-order function which takes two arguments, a function `f` and a digit `forbidden`, and returns another function. If the returned function is passed a number where the digit in the 1s

place is equal to the forbidden digit, it should return the result of calling the given function on the number without that final digit. Otherwise, it should return the result of calling the given function on the number.

```
def forbid_digit(f, forbidden):
    """
    >>> g = forbid_digit(lambda y: 200 // (y % 10), 0)
    >>> g(11)
    200
    >>> g(10)
    200
    >>> g = forbid_digit(lambda x: f'{x}a', 6)
    >>> g(61)
    '61a'
    >>> g(66)
    '6a'
    >>> g = forbid_digit(g, 3)
    >>> g(43)
    '4a'
    >>> g(63)
    '0a'
    >>> g(44)
    '44a'
    """
    def forbid_wrapper(n):
        if n % 10 == forbidden:
            return f(n//10)
        else:
            return f(n)
    return forbid_wrapper
```

(7 points) Lava Bending

a. **(6 pt)** Implement `lava_hopper`, a function that "hops" from one number to the next computed number and tries to avoid any number detected as "lava". When it does land on "lava", it steps backwards by one number until it finds a non-lava number and then keeps hopping.

The function takes four arguments: `start_number` (the initial number), `goal_number` (the target number), `next_hop` (a function that computes the next number based on the current), and `is_lava` (a function that returns a boolean indicating if a number is lava), and it returns the minimum number of hops required to get from `start_number` to at least `goal_number`. The number of hops does not include steps backwards. If either the `start_number` or `goal_number` spots are lava, it returns the string 'No lava allowed there!'.

For example, consider this call

```
lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 4)
```

The function starts from the number 1 and then hops to the numbers 2, 4, realizes that's lava, steps back to 3, hops to 6, hops to 12, and returns 4 (the number of hops required to get to/past 8).

Note that depending on the functions passed in for `next_hop` and `is_lava`, it is possible for a correct `lava_hopper` implementation to result in an infinite loop.

```
def lava_hopper(start_number, goal_number, next_hop, is_lava):
    """
    >>> # hops from 1->2, 2->4, 4->8
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: False)
    3
    >>> # hops from 1->2, 2->4, steps to 3, hops 3->6, hops 6->12
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 4)
    4
    >>> # hops from 1->2, 2->4, 4->8, steps to 7, then 6, then 5, hops to 10
    >>> lava_hopper(1, 10, lambda x: x * 2, lambda x: 6 <= x <= 8)
    4
    >>> # hops from 3->6, 6->12, steps to 11, hops 11->22
    >>> lava_hopper(3, 20, lambda x: x * 2, lambda x: x % 10 == 2)
    3
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 1)
    'No lava allowed there!'
    >>> lava_hopper(1, 8, lambda x: x * 2, lambda x: x == 8)
    'No lava allowed there!'
    """
    if is_lava(start_number) or is_lava(goal_number):
        return 'No lava allowed there!'

    num_hops = 0

    while start_number < goal_number:
        while is_lava(start_number):
            start_number -= 1
```

```
        start_number = next_hop(start_number)
        num_hops += 1
    return num_hops
```

NB: C, `start_number` and F have the same indentation

b. **(1 pt)** Write a call to `lava_hopper` that would result in an infinite loop.

`lava_hopper(1, 8, lambda x: x+1, lambda x: x==6)`

c. **(1 pt)** Who was the second strongest person in history after Samson?

Accept any interesting answer.