CS 88 Fall 2019 Computational Structures in Data Science

Mock Midterm Solutions

INSTRUCTIONS

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5" × 11" crib sheet of your own creation
- The exam is out of **40 points**. We wish you all the best of luck!

Last name	
First name	
Student ID number	
BearFacts email (`_@berkeley.edu`)	
TA	
Name of the person to your left	
Name of the person to your right	
By my signature, I certify that all the work on this exam is my own, and I will not discuss it with anyone until final session is over. **(please sign)**	

1. (9 points) What Would Python Print

Answer the sequence of "What would Python Print" questions. For each, assume that the additional sequence of statements is executed. If the result is an error, write Error. Likewise, if the result is a function, write Function.

```
c, s, m = 4123, 3, 3

sub, add = lambda x, y: x - y, lambda x, y: x + y

def why(n, z):
    b = 0
    for i in range(z):
        n, b = n // 10, b * 10 + n % 10
    if n:
        while z:
            z = z - 1
            high = pow(10, z)
            n = n * 10 + b // high
            b = b % high
        return n

def a(b):
    return lambda f, g: f(s, m) or g(b, s)

def circle():
    c = True
    while (lambda: c)():
        circle = lambda: s + m
        c = not c
    return circle
```

Expression	Interactive Output
`>>> 1 + 2`	3
`>>> warmup = lambda p: p * 2` `>>> warmup(warmup(3))`	12
`>>> c or not c`	4123

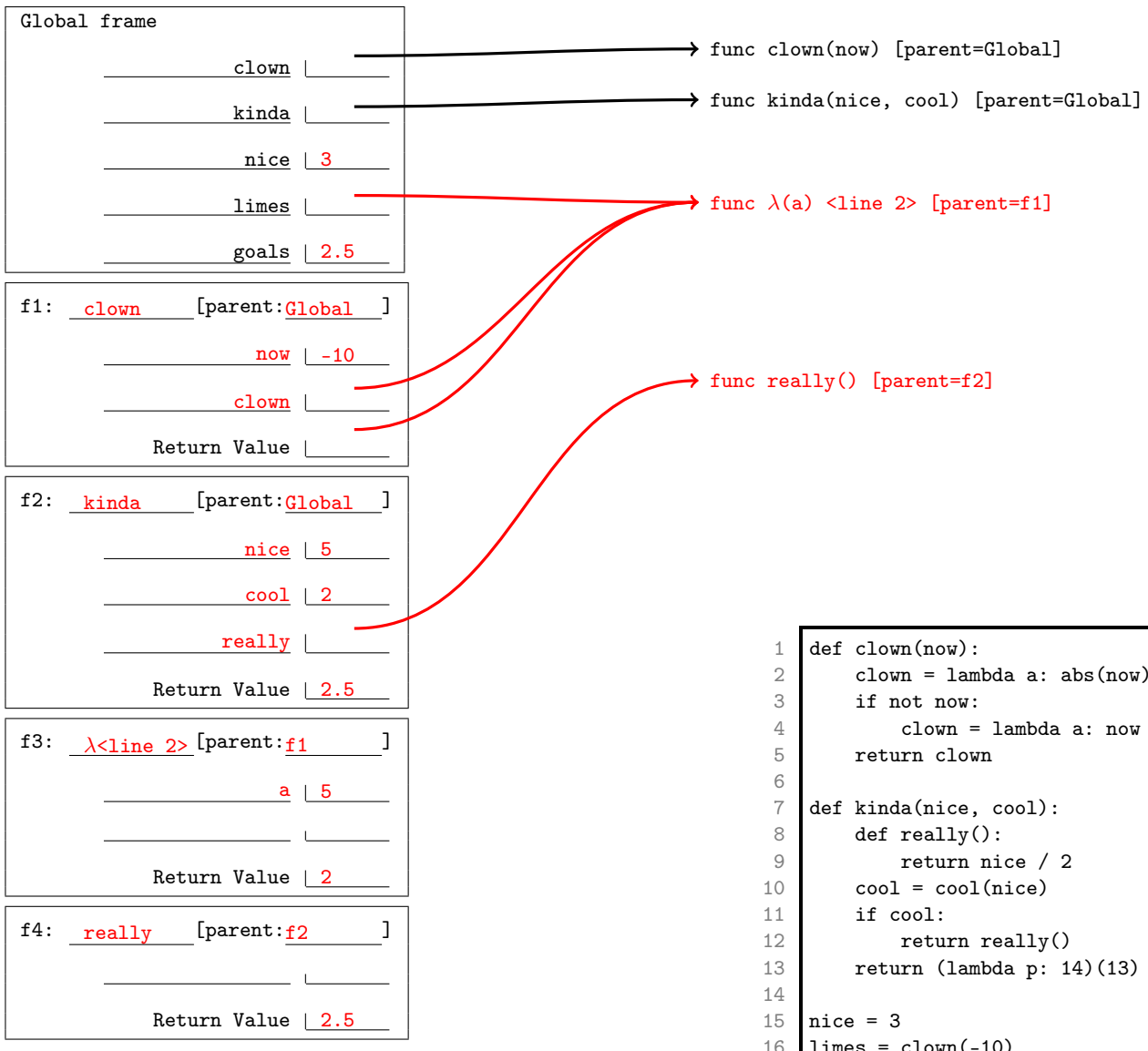
Expression	**Interactive Output**
`>>> not c and c`	False
`>>> a_bit_harder = a(5)` `>>> a_bit_harder`	Function
`>>> stilts = a_bit_harder(sub, add)` `>>> stilts`	8
`>>> cherry = a_bit_harder(add, sub)` `>>> cherry`	6
`>>> a_bit_harder(cherry, stilts)`	Error
`>>> sunny = a(c)` `>>> sunny(why, why)`	4321
`>>> jag = circle()` `>>> jag()`	6

2. (8 points) Environment Diagram

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 clown(now):
2      clown = lambda a: abs(now) // a
3      if not now:
4          clown = lambda a: now + a
5      return clown
6
7  def kinda(nice, cool):
8      def really():
9          return nice / 2
10     cool = cool(nice)
11     if cool:
12         return really()
13     return (lambda p: 14)(13)
14
15 nice = 3
16 limes = clown(-10)
17 goals = kinda(5, limes)
```

3. (3 points) Artwork

Anna (our protagonist) needs to find a job! She decides to dabble in artwork. Help Anna create the function `triangle` which **prints** out a triangle of height k.

For a triangle of height 3, the first level has two spaces followed by one asterisk, the second level has one space followed by three asterisks, and the third level has no spaces followed by five asterisks.

Hint: `print(" ", end="")` print the string " " without a newline afterwards.

```
def triangle(k):
    """ Print out a triangle using '*' of height k
    >>> triangle(3)
      *
     ***
    *****
    >>> triangle(4)
       *
      ***
     *****
    *******
    """
    for i in range(k):

        x = " " * (k - i - 1) + "*" * (2 * i + 1)

        print(x)
```

4. (0 points) Free Space

Enjoy!

5. (7 points) Math

(a) (3 pt) After realizing that artwork is far too difficult, Anna has decided to pursue mathematics. **Recursively** implement `many_factorial` and `multiply_squares`.

For the entirety of this problem, you may use `factorial` defined below and assume n will always be an integer greater than or equal to one.

```
def factorial(n):
    """ Returns the factorial of n
    >>> factorial(3) # 3 * 2 * 1
    6
    """
    if n == 1:
        return 1
    return n * factorial(n - 1)

def many_factorial(n):
    """ Returns the product of the first n factorials
    >>> many_factorial(3) # 3! * 2! * 1!
    12
    """
    if n == 1:

        return 1

    return factorial(n) * many_factorial(n - 1)


def multiply_squares(n):
    """ Returns the product of the first n squares
    >>> multiply_squares(3) # 9 * 4 * 1
    36
    """
    if n == 1:

        return 1

    return n * n * multiply_squares(n - 1)
```

(b) (2 pt) Anna realizes that much of her code above is repeated! She decides to make her code more concise by combining all three of the functions above into one function — `general_solver` — which takes in a number `n` and a one argument function `func` and returns the product of the first `n` calls to `func`.

```
def general_solver(n, func):
    """ Returns the product of the first n calls to func
    >>> general_solver(3, factorial) # 6 * 2 * 1
    12
    """
    if n == 1:

        return 1

    return func(n) * general_solver(n - 1, func)
```

(c) (2 pt) Next, fill in the two functions calls below so the first is equal to `factorial(n)` and the second is equal to `multiply_squares(n)`.

```
1. general_solver(n, lambda x: x) # equivalent of factorial(n)

2. general_solver(n, lambda x: x * x) # equivalent of multiply_squares(n)
```

6. (7 points) Encryption

(a) (1 pt) After getting bored of math, Anna decides encryption is her calling. First, implement `list_to_string`, which takes a `lst` — a list of characters — and returns a string of the characters in `lst`.

```
def turn_to_string(lst):
    """Given a list of characters, turn them into a string
    >>> turn_to_string(['f', 'r', 'i', 's', 'b', 'e', 'e'])
    'frisbee'
    """
    answer = ""

    for x in lst:

        answer += x

    return answer
```

(b) (3 pt) Anna is now ready to code her first caesar cipher! Implement `caesar_cipher` which takes in a string `message` and a number `shift`.

A **caesar cipher** adds the given `shift` (a number between 0 and 25) to each letter in the string, circling around the alphabet if the shifted letter overshoots z.
For example, 'zoo' shifted by 2 produces 'bqq' ('z' shifted 2 is 'b', 'o' shifted 2 is 'q').

You may use `turn_to_string` (assuming it works) as well as the `index_of` and `char_at` functions defined below.

```
def char_at(x):
    """Returns the character at the given index
    >>> char_at(2) # a -> 0, b -> 1, c -> 2 ...
    'c'
    """
    ...

def index_of(y):
    """Returns the index of the given character
    >>> index_of('c') # a -> 0, b -> 1, c -> 2 ...
    2
    """
    ...
```

Hints: You can iterate through strings. For example: `[x for x in 'max']` → `['m', 'a', 'x']`. In addition, the modulo operation can be very helpful here!

```
def caesar_cipher(message, shift):
    """Apply the designated caesar cipher shift on the given message
    >>> caesar_cipher("any", 2)
    'cpa'
    >>> caesar_cipher("hello", 1)
    'ifmmp'
    """
    return turn_to_string([char_at((index_of(m) + shift) % 26) for m in message])
```

(c) (3 pt) Anna wants to expand her encryption services! Instead of only encrypting messages, she now wants to decrypt them. Assist Anna in implementing `guess`, which takes in two strings — `encrypted` and `possibility` — and returns whether `possibility` can be shifted to produce `encrypted`.

You may use `caesar_cipher` (assuming it works).

```
def guess(encrypted, possibility):

    """Returns whether possibility can be the encrypted message, i.e., return if
    possibility can be shifted to produce the message
    >>> guess("hello", "ifmmp") #Shift of 25
    True
    >>> guess("hello", "izmmp")
    False
    >>> guess("ultimate", "ultimate") #Shift of zero
    True
    """
    return encrypted in [caesar_cipher(possibility, i) for i in range(26)]
```

7. (6 points) Sorted, Increasing Numbers

(a) (3 pt) Anna has decided to give up on finding a job. Instead, she has started exploring her curiosity in sorted, increasing numbers. Without using lists, help Anna implement the `unique_digits` function, which takes an sorted, increasing number `a` and returns the number of unique digits in `a`.

```
def unique_digits(a):
    """Given a number a that is in sorted, increasing order,
    return the number of unique digits in a.
    >>> unique_digits(11123)
    3
    >>> unique_digits(12345)
    5
    >>> unique_digits(9)
    1
    >>> unique_digits(0)
    1
    """
    if a < 10:

        return 1

    last, rest = a % 10, a // 10

    if last == rest % 10:

        return unique_digits(rest)

    return 1 + unique_digits(rest)
```

(b) (3 pt) Implement `missing_digits`, which takes a sorted, increasing number `a` and returns the number of missing digits in `a`. A missing digit is a number between the first and last digit of `a` that is not in `a`. Assume `a` has *no* repeated numbers, i.e. all the digits are unique.

```
def missing_digits(a):
    """Given a number a that is in sorted, increasing order,
    return the number of missing digits in a. A missing digit is
    a number between the first and last digit of a that is not in a.
    >>> missing_digits(1248) #3, 5, 6, 7
    4
    >>> missing_digits(123456) # No missing numbers
    0
    """
    if a < 10:

        return 0

    last, rest = a % 10, a // 10

    return last - rest % 10 - 1 + missing_digits(rest)
```