CS 88
Spring 2019

Computational Structures in Data Science

MIDTERM

INSTRUCTIONS

Solutions

- You have 2 hours to complete the exam.
- The exam is closed book, closed notes, closed computer, closed calculator, except one 8.5” × 11” cheat sheet of your own creation and the official CS 88 midterm reference sheet (attached to the back of the exam).
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper. Check that you have 5 double-sided pages (including cover) for 6 problems. Put your name and SID on every page.

Last name	
First name	
Student ID number	
Berkeley 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

- You may use built-in Python functions that do not require import, such as `min`, `max`, `pow`, and `abs`. You may not use functions defined on your study guide unless clearly specified in the question.
- 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. Your solution must fit within the number of lines provided, but may not require all of the lines.
- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.

[This page is purposely left blank. Use it as scratch space.]

1. Evaluators Gonna Evaluate

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".** You answers must fit within the boxes provided. Work outside the boxes will not be graded.

Hint: No answer requires more than 4 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:

```
x = 3
y = 2
z = 1
a = 'cs88'
def fun_hof(f, lst):
      return [f(x, x) for x in lst if f(x, x)]
def combine(a, b):
      return a+b if a+b else 'combined'
def mooz(m, n):
      if m * n > 9:
            return m
      else:
            return None
```

Expression	Interactive Output
`x * y`	`6`

`def free(s):`
`    print(s)`
`    return 'free ' + s`
`free('points')`
	`points`
	`'free points'`

`z / y`	`0.5`
`a / y`	`TypeError`
`z if z // y else x`	`3`
`a * 2`	`'cs88cs88'`

Name and SID: ______________________________

`a or x`	‘cs88’
`print(mooz(2, 3))`	None

`mooz(3, 2)`
`mooz(4, 3)`
	4

`mooz(7, 7)`
`mooz(3, 3) + mooz(4, 4)`
	7
	TypeError

`mooz(5, 5) or mooz(4, 4)`	5
`fun_hof(combine, a)`	[‘cc’, ‘ss’, ‘88’, ‘88’]
`fun_hof(combine, [0, 1, 2, 3])`	['combined', 2, 4, 6]
`fun_hof(mooz, [-2, -1, 0]) and x == 3`	[]

`grades(100)`
`def grades(score):`
`    if score >= 90:`
`        return ‘Awesome!’`
`    elif score < 70:`
`        return ‘Come to OH!’`
	NameError

`def get_element(k):`
`    def f(lst):`
`        if k < len(lst):`
`            return lst[k]`
`            print(‘Found it!’)`
`        print(‘Let me see...’)`
`    return f`
`get_5 = get_element(5)`
`get_5([1, 2, 3, 4, 5])`
`get_5(range(1, 10))`
	Let me see...
	6

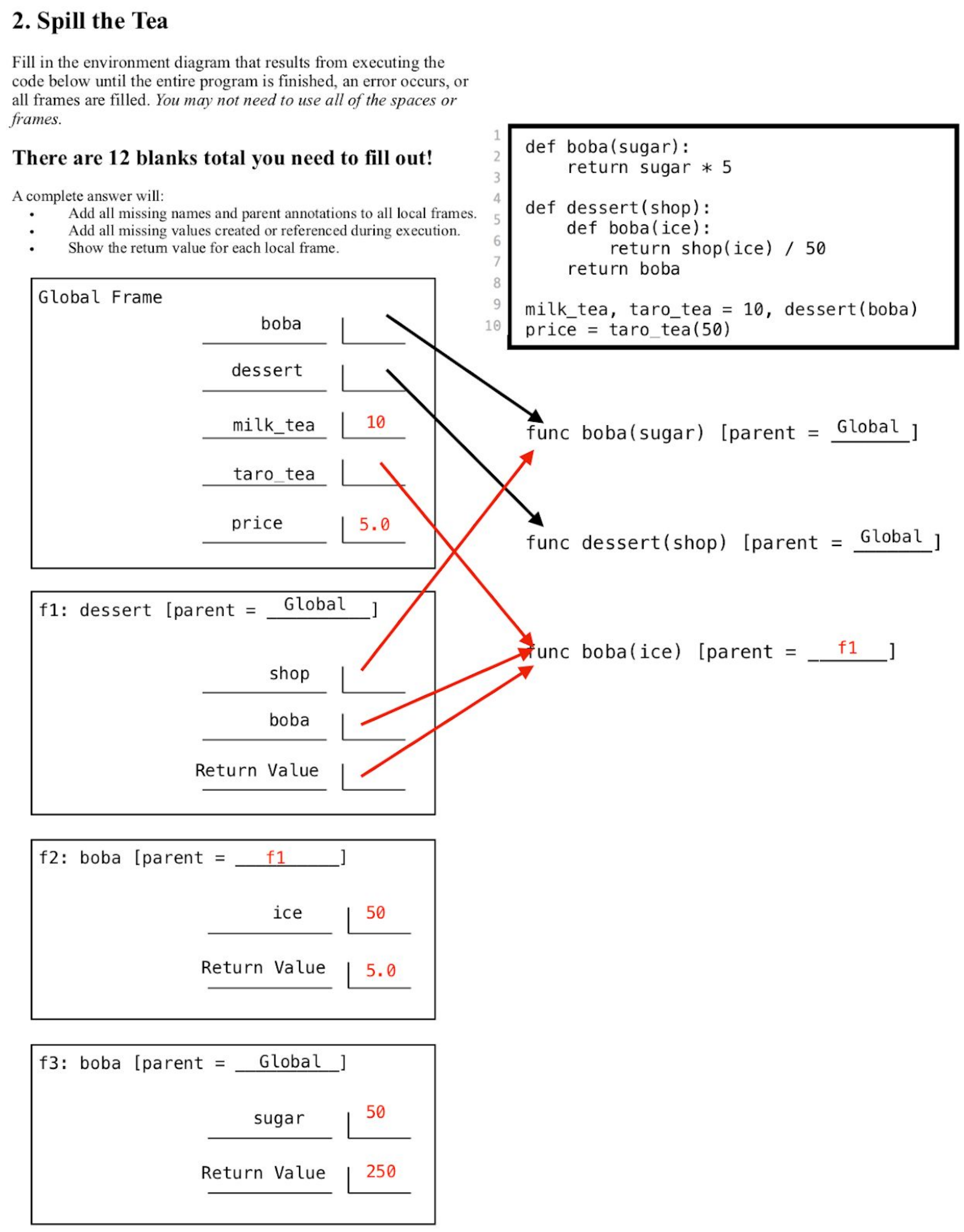
2. Spill the Tea

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.

Solution: https://goo.gl/ZGcJQk



2. Spill the Tea

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.*

There are 12 blanks total you need to fill out!

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.

```
def boba(sugar):
    return sugar * 5

def dessert(shop):
    def boba(ice):
        return shop(ice) / 50
    return boba

milk_tea, taro_tea = 10, dessert(boba)
price = taro_tea(50)
```

Global Frame

boba	
dessert	
milk_tea	10
taro_tea	
price	5.0

func boba(sugar) [parent = Global]

func dessert(shop) [parent = Global]

func boba(ice) [parent = f1]

f1: dessert [parent = Global]

shop	
boba	
Return Value	

f2: boba [parent = f1]

ice	50
Return Value	5.0

f3: boba [parent = Global]

sugar	50
Return Value	250

3. Fibonacci Sequence

You define a function that calculates the nth item of the Fibonacci sequence to be this:

```
def fib(n):
    if n <= 2:
       return 1
    else:
       return fib(n-1) + fib(n-2)
```

You call the `fib` function with the input 6, i.e. `fib(6)`. How many calls to **`fib`** will the function make? Include the original call to `fib` in your solution, i.e. `fib(1)` produces 1 call to `fib`.

`Solution: 15`

4. Harshad Number

In math, a Harshad (or Niven) number is defined as a number that is divisible by the sum of its digits. For example, the number 18 is a Harshad number, because the sum of the digits 1 and 8 is 9, and 18 is divisible by 9. The number, 15, is not a Harshad number because the sum of the digits 1 and 5 is 6, and 15 is not divisible by 6.

Given a positive integer input, write an iterative program that returns True if the input is a Harshad number, and False otherwise.

```
def is_harshad(n):
    """Returns True if n is a Harshad number. False otherwise.

    >>> is_harshad(18) # 18 is divisible by (1+8)
    True
    >>> is_harshad(15) # 15 is NOT divisible by (1+5)
    False
    >>> is_harshad(111) # 111 is divisible by (1+1+1)
    True
    """
    sum_digits = 0
    temp = n
    while temp > 0:
        sum_digits = sum_digits + temp % 10
        temp = temp // 10
    return n % sum_digits == 0
```

5. NBA Players

Andrew is a big basketball fan and wants to know the average height of every team in the NBA. He is given a list of NBA players represented as tuples: (Name, Height, Team). Before writing a function to find the average height of each team, he first defines some helper functions.

For each of these problems, assume that the following list has already been initialized:

```
nba = [
    ("Lebron James",     80, "Los Angeles Lakers"),
    ("Stephen Curry",    75, "Golden State Warriors"),
    ("Klay Thompson",    79, "Golden State Warriors"),
    ("Michael Jordan",   78, "Chicago Bulls"),
    ("Kobe Bryant",      78, "Los Angeles Lakers"),
    ("Shaquille O'Neal", 85, "Los Angeles Lakers")
]
```

You may further assume the following functions:

- `sum(s): returns the sum of a sequence s`
- `map(f, s): maps function f over sequence s`
- `filter(f, s): filters sequence s by function f`

a. team_filter takes in a team and returns a function that checks if a player is on that team.

```
def team_filter(team):
    """Generates a function that takes in a player and checks if the
    player is on the team

    >>> is_warrior = team_filter("Golden State Warriors")
    >>> is_warrior(nba[0])
    False # Lebron James is not on the Golden State Warriors
    >>> is_warrior(nba[1])
    True # Stephen Curry is on the Golden State Warriors
    >>> is_warrior( ("Andrew Tan", 76, "Golden State Warriors") )
    True # Andrew Tan wishes he was on the Golden State Warriors
    """
    def is_team(player):
        return player[2] == team
    return is_team
```

b. teammates takes in a list of players and team, and returns a list of players on that team.

```
def teammates(players, team):
    """Return a list of players who are on the specified team

    >>> teammates(nba, "Golden State Warriors")
    [("Stephen Curry", 75, "Golden State Warriors"), ("Klay Thompson",
    79, "Golden State Warriors")]
    >>> teammates(nba, "Chicago Bulls")
    [("Michael Jordan", 78, "Chicago Bulls")]
    """
    return filter( team_filter(team), players )
```

c. get_player_height takes in a player and returns the height of that player in inches.

```
def get_player_height(player):
    """Return the height of the given player. Assume player is
    represented as a tuple (name, height, team).

    >>> get_player_height(nba[0])
    80 # Lebron James is 80 inches tall
    >>> get_player_height(nba[1])
    75 # Stephen Curry is 75 inches tall
    >>> get_player_height(("Andrew Tan", 76, "Golden State Warriors"))
    76 # Andrew Tan wishes he was 76 inches tall
    """
    return player[1]
```

d. team_average produces the average height of players on a team

```
def team_average(players, team):
    """Takes in a list of players (Name, Height, Team) and team, and returns
    the average height of the players on that team

    >>> team_average(nba, "Los Angeles Lakers")
    81 # (80 + 78 + 85) / 3
    >>> team_average(nba, "Golden State Warriors")
    77 # (75 + 79) / 2
    """
    team_players = teammates(players, team)
    team_heights = map(get_player_height, team_players)
    return sum(team_heights) / len(team_heights)
```

6. Subtraction Game

In game theory, a subtraction game is a simple game with two players, player 0 and player 1. At the beginning, there is a pile of n cookies. The players alternate turns; each turn, a player can take anywhere from 1 to 3 cookies. The player who takes the last cookie wins. Fill in the function can_win, which returns True if it is guaranteed to win starting at the given number of cookies for the current player. Assume the players play with perfect logic (i.e. always make the optimal move). It uses the following ideas:

- If the number of cookies is nonpositive, it is impossible to win, so return False.
- Otherwise, the current player can choose to take 1, 2, or 3 cookies.
  - Evaluate each action: if that action forces the opponent to lose, then return True (since we can win).
  - If none of the actions can guarantee a win, then return False.

```
def can_win(number):
    """Returns True if the current player is guaranteed a win
    starting from the given state. It is impossible to win a game
    from an invalid game state.

    >>> can_win(-1) # invalid game state
    False
    >>> can_win(3) # take all three!
    True
    >>> can_win(4) # you take 1-3 cookies, opponent takes remaining
    False
    """
    if number <= 0:
        return False
    for action in range(1, 4):
        new_state = number - action
        if not can_win ( new_state ):
            return True
    return False

    [or]

    if number <= 0:
        return False
    if number % 4 == 0:
        return False
    else:
        return True
```