

Name: _____

Rules and Hints

- You may use one handwritten 8.5 x 11" cheat sheet (front and back). This is the only additional resource you may consult during this exam. *No calculators.*
- When you write code, be sure that the indentation level of each statement is clear.
- Explain/show work if you want to receive partial credit for wrong answers.
- As long as your code is correct, you will get full credit. No points for style.
- As always, the SSU rules on academic integrity are in effect.

Problem	Max Score	Your Score
<i>Problem 1:</i> Trace Code	20	
<i>Problem 2:</i> Trace Functions	20	
<i>Problem 3:</i> Define functions	30	
<i>Problem 4:</i> Write a complete program	30	
Total	100	

Problem 1: Trace Code (20 points)

Write what will be printed to the screen when each of the following snippets of code is executed in PyCharm or in the Online Python Tutor.

Write your final solution in the box provided

Do not write any scratch-work in the solution box.

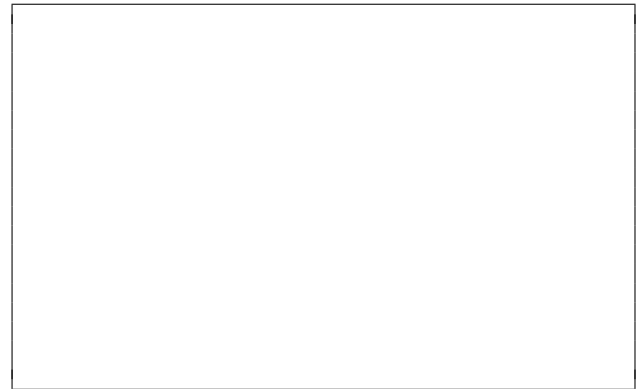
In your solution, be very precise with spacing, line breaks, etc.

Treat each sub-problem as an independent question.

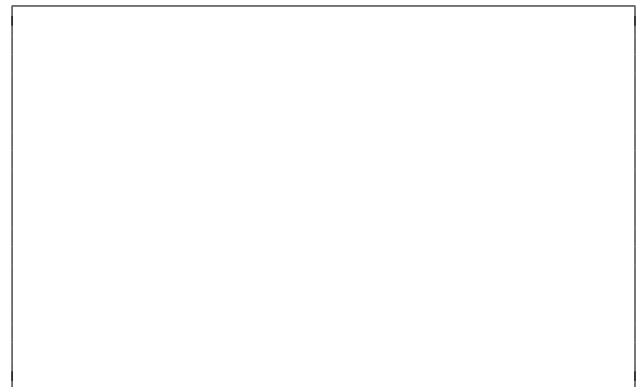
All questions in this section are worth 4 points.

Problem 1A

```
c = 12
while c > 5:
    c = c - 3
    print(c)
```

**Problem 1B**

```
s = 'Russian River'
print(len(s))
print(s[2] + s[-2])
print(s[5:10])
print(s[-5:-2])
```



Problem 1C

```
dna = ['CAGT', 'TTC',  
       'GCC', 'ATTG']  
print(len(dna))  
print(len(dna[1]))  
print(dna[3])  
print(dna[1][2])
```

Problem 1D

```
A = ['Suki', 'Peppy',  
     'Poppy']  
B = A  
C = A[:]  
B[1] = 'Guy'  
C[2] = 'Chef'  
print(A)  
print(C)
```

Problem 1E

```
L = []  
i = 2  
while i <= 8:  
    i = i * 2  
    L.append(i)  
    print(L)  
L.reverse()  
print(L)
```

Problem 2: Trace Functions (20 points)

Write what will be printed to the screen when each of the following snippets of code is executed in PyCharm or in the Online Python Tutor.

Write your solution in the box provided.

Do not write any scratch-work in the solution box.

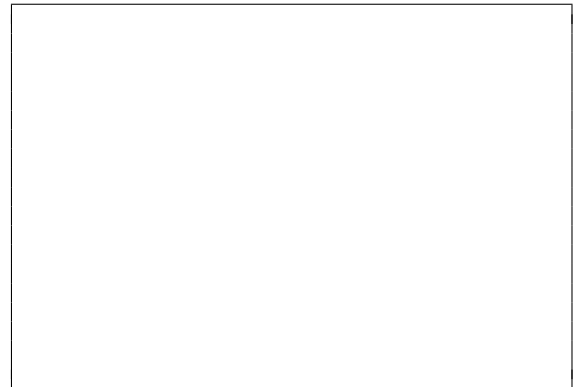
In your solution, be very precise with spacing, line breaks, etc.

Treat each sub-problem as an independent question.

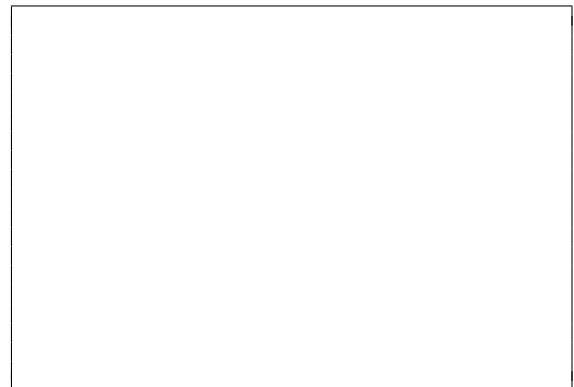
All questions in this section are worth 5 points.

Problem 2A

```
def add2Angle(angle, val):  
    angle = angle + val  
    print(angle)  
  
angle = 45  
val = 5  
add2Angle(angle, val)  
print(angle)
```

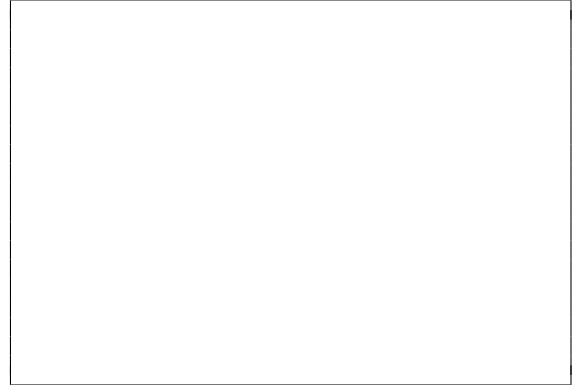
**Problem 2B**

```
def mystery(s):  
    half = len(s) // 2  
    fHalf = s[:half]  
    sHalf = s[half:]  
    print(fHalf, sHalf)  
    if half % 2 == 1:  
        return fHalf.upper()  
    else:  
        return sHalf.upper()  
  
answer = mystery('BiggiE')  
print(answer)
```

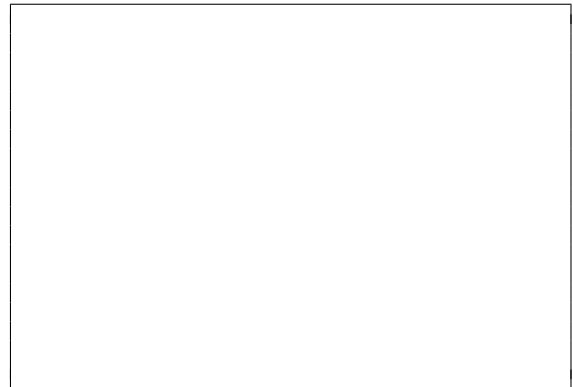


Problem 2C

```
def getElem(A):  
    return A[len(A)-1]  
  
A = [1, 3, 2, 4, 8,  
     7, 6, 5]  
a_elem = getElem(A)  
print(len(A), a_elem)  
  
B = [[1], [2, 3, 6],  
     [4, 5, 8], [0, 7]]  
b_elem = getElem(B)  
print(len(B), b_elem)
```

**Problem 2D**

```
def add1(L):  
    for i in range(len(L)):  
        L[i] = L[i] + L[0]  
    print(L)  
  
A = [2, 3, 4]  
add1(A)  
print(A)
```



Problem 3: Define functions (30 points)

Define functions to perform the following tasks, obeying the stated requirements, including:

- Assume that the math library has been imported for you.
- Do not ask the user for input unless the specification explicitly requires it.
- Do not print anything unless the specification explicitly requires it.
- Do not call any function unless the specification explicitly requires it.

Problem 3A (7 points)

Define a function named `averageGrade` that:

- Takes 2 parameters: a float representing total grade of students and an integer representing number of students
- Returns a float representing the average grade.

For example, `averageGrade(8978.5, 100)` returns `89.785`

Problem 3B (8 points)

Define a function named `list_length` that:

- Takes 1 parameter: a list of strings
- Returns a list containing length of each string

Example of calling this function:

```
list_length(['sonoma', 'state', 'university']) returns [6, 5, 10]
```

Problem 3C (8 points)

Define a function named `getColMax` that:

- Takes 2 parameters: a nested list (i.e., a 2-dimensional list) and an integer `j`
- Returns the maximum of all the elements in column `j`

You may assume that column `j` exists.

Example of calling this function:

```
Let L = [ [5,7,10],  
          [3,2,1],  
          [6,9,4],  
          [8,5,2]], then
```

`getColMax(L,1)` returns 9 because column 1 has the elements 7, 2, 9 and 5 which has max value 9.

Problem 3D (7 points)

Define a function named `sumColMax` that:

- Takes 1 parameter: a nested list (i.e., a 2-dimensional list)
- Returns the sum of maximum of each column of the 2D list
- Calls the function `getColMax` (which was defined in Problem 3C) to compute maximum of an individual column

Example of calling this function:

Let `L = [[5,7,10],
 [3,2,1],
 [6,9,4],
 [8,5,2]]`, then

`sumColMax(L)` returns 27 because sum of maximum element in column 0 (8), maximum element in column 1 (9), and maximum element in column 2 (10) is equal to 27.

Problem 4: Write a complete program (30 points)

For this problem, you must write a complete program. This includes writing a docstring, logic in `def main()`, a call to `main()`, any necessary library imports, etc.

If you get stuck, try to maximize your partial credit. To get full credit, your functions should call each other where appropriate and avoid duplicating code. In all examples, user input is underlined.

Your program should contain the following functions:

- `get_toppings`:
 - Takes no parameters. Prompts the user for pizza toppings until the user enters a blank line. Each line of user input should be considered a topping. Returns a list of toppings the user entered (not including the blank line).

Example: A call to `get_toppings` with the below sample user input

```
Enter a name: red onions (v)
Enter a name: tomatoes (Veg)
Enter a name: Anchovies (sea)
Enter a name:
```

should return the list `['red onions (v)', 'tomatoes (Veg)', 'Anchovies (sea)']`

- `is_vegetarian`:
 - Takes 1 parameter: a string holding a topping. Returns `True` if the last word of the string contains `V` (in lower or uppercase), otherwise it returns `False`.

Example: For `'red onions (v)'`, the function returns `True`

Example: For `'tomatoes (Veg)'`, the function returns `True`

Example: For `'Anchovies (sea)'`, the function returns `False`

- `main`:
 - Calls `get_toppings` to get a list of toppings from the user. Uses `is_vegetarian` to determine total number of vegetarian toppings.
 - If all toppings are vegetarian, it prints `Pizza is vegetarian` otherwise it prints `Pizza is not vegetarian`.

Sample input/output:

```
Enter a name: red onions (v)
Enter a name: tomatoes (Veg)
Enter a name: Anchovies (sea)
Enter a name:
```

```
Pizza has 2 vegetarian toppings
```

```
Pizza is not vegetarian
```

Start your solution on the next page...

Problem 4, continued ...

Problem 4, continued ...