Computer Science 1 — CSci 1100 — Fall 2022
Exam 1
September 29, 2022

SOLUTIONS

1. (**10 points total; 2 points each**) Assume each of the following is a complete, separate program. For each program find the **first** error that prevents the program from generating output, or decide there is no error. If there is an error, write in the solution box that there is an error, write the line number **and** very briefly describe the error. If the program would run and generate output, write **No Error**.

Part a:

```
s=('abc','def')  #1
x,y_z,t = s      #2
print(s[1])      #3
print([y_z])     #4
```

Solution:

```
Line #2 - LHS has 3 values
RHS has 2 values
```

Part b:

```
s1='hello "python" '    #1
s2='Is fun'             #2
new_s=s1+s2             #3
s3=s1 s2                #4
print(s1*3)             #5
```

Solution:

```
line #4, Need + for
variable concatenation
```

Part c:

```
s1='Good\tmorning\n'    #1
s2='Have\ta\nnice\nday' #2
s3=('\\\\n'*3)          #3
s4= s1*s3               #4
print(s4)               #5
print(s1+s2)            #6
```

Solution:

```
Line number 4, cannot multiply
string with non-integer type
```

Part d:

```
x = 9                              #1
y = 9 / 3                          #2
a = "abab"                         #3
b = int(y) * a                     #4
print(b)                           #5
```

Solution:

```
No error
```

Part e:

```
from math import *                #1
import math                       #2
def new(s):                       #3
       math.pi=3                  #4
       return sqrt(s) *math.pi    #5
print(new(math.pi))               #6
```

Solution:

```
No error
```

2. (**15 points total; 3 points each**) Assume each of the following is an individual program. Write the **exact** output of these programs in the area provided. Use the scratch area as you wish. There are **no syntax errors** in the code provided in this question.

Part a:

```
print('*\n'*3,end='*')
print('-','+','\\',sep='*')
```

Solution:

```
*
*
*
*-*+*\
```

Scratch area:

Part b:

```
def t(x):
    n=int(x[0])
    m=x[1]
    return n*m,str(n)+m

print(t(('2','2')))
print(t('25'))
```

Solution:

```
('22', '22')
('55', '25')
```

Scratch area:

Part c:

```
def _find(s):
    x=s.count('*')
    y=s*x
    return y.replace('*','*'*x)

s='1*2*'
print(_find(s))

t='12*31'
print(_find(t))
```

Solution:

```
1**2**1**2**
12*31
```

Scratch area:

Part d:

```
a='apple'
b='Cat'
c='Zoo'

print(a<b or b<c)
print(a>c and a<b)
print(a==c or a!=b)
```

Solution:

```
True
False
True
```

Scratch area:

Part e:

```
import math as m
x='helloworld'
x1=x.count('o')
x2=m.sqrt(x.find('o'))
x.upper()
x3=x.count('O')
print(x*x1)
print(x2)
print(x3)
```

Solution:

```
helloworldhelloworld
2.0
0
```

Scratch area:

3. (**20 points total; 4 points each**) Give a single line of code to accomplish each of the following. You can assume you are typing into the Python interpreter. Make sure your solution is only one line. Answers of more than one line will result in a loss of credit. Do not use **if**s or **loops**.

Part a:

Given a tuple variable `t` with exactly 2 string type values, write a single line of code to print average number of letter `a's` in the tuple. For example, with the values `t=('aaAAaeroplane','aheAd')`, your expression would print:

```
4.0
```

Here there are 6 `a's` in the first string and 2 `a's` in the second. Therefore the average per word is `4.0`

Solution:

```
print((t[0].lower().count('a')+\
t[1].lower().count('a'))/2)
```

Part b:

Given a string `s` of of 4 digits, write a single line of code to output the larger of either the first 2 digits or the last 2 digits. For example if `s = '1234'`, the output should be the integer (not string) 34. If `s = '9934'`, the output should be the integer (not string) 99.

Solution:

```
print(max(int(s)%(10**(len(s)-2)),\
int(s)//(10**(2))))
```

or

```
print(max(int(s[0]+s[1]),int(s[2]+s[3])))
```

or

```
print(int(max((s[0]+s[1]),s[2]+s[3])))
```

Part c:

Given a string `a`, write an expression that prints the string repeated on three consecutive lines in all capitals. Note that this will require a print(). For `a='hello world, this is python'` this would be:

```
HELLO WORLD, THIS IS PYTHON
HELLO WORLD, THIS IS PYTHON
HELLO WORLD, THIS IS PYTHON
```

Solution:

```
print(3*(a.upper()+'\n'),end="")
```

or

```
print((3*(a.upper()+'\n')).strip())
```

Part d:	**Solution:**

Given a point, `(x,y)` and a rectangle with corner coordinates `(x0, y0)`, `(x0, y1)`, `(x1, y0)` and `(x1, y1)` where `x0<x1` and `y0<y1`, write a **boolean** expression using the variables `x, y, x0, x1, y0, y1` that evaluates to `True` if the point is strictly within the rectangle. Otherwise, it evaluates to `False`.	`x0<x<x1 and y0<y<y1`
	
	or
	
	`x0<x and x<x1 and y0<y and y<y1`

Part e:	**Solution:**

Given a string **a**, write an expression that evaluates to the string with all spaces, tabs, and back slashes replaced with commas, and with the initial character of the string in upper case. For example, if:
 	
`a = "what is your quest?"`
 	
For **a**, this would give:
 	
`'What,is,your,quest?'`
	`a.capitalize().replace(' ',',').\`
	`replace('\t',',').replace('\\',',')`

This page is left blank for scratch work. Do not put your solutions here. Put them in the boxes provided for each question.

‘

4. (**16 points**) You visit the supermarket to buy groceries. In this problem you will write code to calculate the change the store should give back depending on the amount due. Assume, you can only pay in one dollar bills. Your change can be returned only in one dollar bills (100 cents), quarters (25 cents), dimes (10 cents),nickels (5 cents) and pennies (1 cent) (no other denominations). Assume your code takes the amount due and amount paid (by you) both as input variables.

   Depending on the amount due and the money you paid, you should print appropriate output messages.

   You can use **if** constructs including **elif** and **else** clauses, but do not use **loops**.

   Here are some example runs :

```
>>>Amount due=>5
>>>Dollar bills paid=>5
No change due

>>>Amount due=>5.75
>>>Dollar bills paid=>7
Collect 1 one dollar bills
Collect 1 quarters
Collect 0 dimes
Collect 0 nickels
Collect 0 pennies

>>>Amount due=>5
>>>Dollar bills paid=>3
Paid 2.0 dollars less than amount due

>>>Amount due=>8.82
>>>Dollar bills paid=>11
Collect 2 one dollar bills
Collect 0 quarters
Collect 1 dimes
Collect 1 nickels
Collect 3 pennies
```

Write your answer on the next page.

Solution:

```
amount=float(input('Amount due=>'))
money=int(input('Dollar bills paid=>'))
if money > amount:
    ones=max(0,int(((money-amount)*100)//100))
    due=abs(ones*100-round((money-amount)*100))
    quarters_due=due//25
    dimes_due=(due%25)//10
    nickels_due=((due%25)%10)//5
    pennies_due=due-(quarters_due*25+dimes_due*10+nickels_due*5)
    print("Collect {} one dollar bills".format(ones))
    print("Collect {} quarters".format(int(quarters_due)))
    print("Collect {} dimes".format(dimes_due))
    print("Collect {} nickels".format(nickels_due))
    print("Collect {} pennies".format(pennies_due))
elif money==amount:
    print('No change due')
else:
    print('Paid {} dollars less than amount due'.format(amount-money))
```

5. (**13 points**) Two users play a game by entering strings with English characters only. Write a function `score_calc(word)` that takes a string type as input and returns a score for the word based on the following rules.

Rules for scoring:

```
    1. Even length word with the same first and last letter gets 5 points
    2. Odd length word with the same first , last and middle letter gets 8 points
    3. All a's and e's count towards the score with 1 point for every occurence.
```

The player whose string scores more, wins. Then finish up the program by asking each player for their word, and calling `score_calc()` to determine the winner. Your code should handle both upper and lower case. If the user enters nothing or an empty string, then the score should be zero. See a few sample runs below.

```
>>>Player 1 word==>
>>>Player 2 word==>ab
Player 1 score= 0  and Player 2 score= 1
Player 2 won!

>>>Player 1 word==>APala
>>>Player 2 word==>eeae
Player 1 score= 11  and Player 2 score= 9
Player 1 won!

>>>Player 1 word==>eerie
>>>Player 2 your word==>aaria
Player 1 score= 3  and Player 2 score= 3
Its a tie!
```

Write your answer on the next page.

Solution:

```python
def score_calc(word):
    score=0
    if len(word)==0:
        return score
    word=word.lower()
    if len(word)%2==0 and word[0]==word[-1]:
        score+=5
    if len(word)%2!=0 and word[0]==word[-1]==word[len(word)//2]:
        score+=8
    score+=word.count('a')
    score+=word.count('e')
    return score
```

or

```python
def score_calc(w):
    if len(w) == 0:
        return 0
    w = w.upper()
    even = len(w) % 2 == 0
    score = w.count('A') + w.count('E')
    if even:
        if w[0] == w[-1]:
            score += 5
    else:
        if w[0] == w[-1] == w[len(w)//2]:
            score += 8
    return score
```

```python
w1=input('Player 1 word==>').strip()
w2=input('Player 2 word==>').strip()
Player1=score_calc(w1)
Player2=score_calc(w2)
print('Player 1 score=',Player1,' and Player 2 score=', Player2)
if Player1>Player2:
    print('Player 1 won!')
elif Player1<Player2:
    print('Player 2 won!')
else:
    print('Its a tie!')
```

‘

6. (**14 points**) This question is in 2 parts.

   Part a (6/14 points): In the space below write the contents of the file **line.py**. The file has a function `midpoint`. The function takes a 4-tuple `(x1,y1,x2,y2)` as input with first 2 elements as the `(x1,y1)` coordinates of a first point and last 2 elements `(x2,y2)` as the coordinates of a second point. The function returns a 2-Tuple representing the midpoint of the line-segment formed by these points, `((x1+x2)/2,(y1+y2)/2)`

   Here are a few sample calls for your function:

```
>>> print(midpoint((5,1,6,3)))
(5.5, 2.0)
>>> print(midpoint((10,10,30,40)))
(20.0, 25.0)
```

   Solution:

```python
def midpoint(t):
    return (t[0]+t[2])/2 , (t[1]+t[3])/2
```

Part b: (8/14 points) Now write code in a separate file called main.py. Assume you are given 2 points `(x0, y0)` and `(x1, y1)` that are the 2 opposite corners of a rectangle (bottom left and top right). You can assume these are already available to you and you do not need to do input to get the values. Your program should print the midpoints of the 4 sides of the rectangle. For full credit you must use the function written in the `line.py` file. Assume the input is a valid rectangle that has sides parallel to the x and y axes. NOTE: The order in which mid-points are printed does NOT matter. Here is some sample output:

```
>>>x0=0
>>>y0=0
>>>x1=5
>>>y1=10

First midpoint = (0.0, 5.0)
Second midpoint = (2.5, 0.0)
Third midpoint = (2.5, 10.0)
Fourth midpoint = (5.0, 5.0)
```

Solution:

```python
import line
x0=0    # No deduction if this is missing
y0=0    # No deduction if this is missing
x1=5    # No deduction if this is missing
y1=10   # No deduction if this is missing

first=line.midpoint((x0,y0,x0,y1))
second=line.midpoint((x0,y0,x1,y0))
third=line.midpoint((x0,y1,x1,y1))
fourth=line.midpoint((x1,y1,x1,y0))

print('First point =',first)
print('Second point =',second)
print('Third point =',third)
print('First point =',fourth)
```

‘

7. (**12 points**) Given a 3-Tuple `tup` comprised of three strings, write 3 different single print statement arragements to print each component of the Tuple on a separate line as follows. For example, if `tup=("Line 1". "Line 2", "Line 3")`, then the output would be

```
Line 1
Line 2
Line 3
```

All solutions must use the elements of `tup`.

Part a (3/12 points): Use 1 print statement with 3 value arguments. Control the new line using the `sep` parameter of print.

Solution:

```
print(tup[0], tup[1], tup[2], sep='\n')
```

Part b (3/12 points): Use 1 print statement with a single format string. The `format()` should take the 3 elements of `tup` as arguments.

Solution:

```
print('{}\n{}\n{}'.format(tup[0], tup[1], tup[2]))
```

Part c (3/12 points): Use 1 print statement with a single value argument. Use the concatenation operator and the new line escape sequence to compose a single string.

Solution:

```
print(tup[0]+\n+tup[1]+\n+tup[2])
```

Part d (3/12 points): Use 3 different print statements.

Solution:

```
print(tup[0])
print(tup[1])
print(tup[2])
```

This page is left blank for scratch work. Do not put your solutions here. Put them in the boxes provided for each question.

This page is left blank for scratch work. Do not put your solutions here. Put them in the boxes provided for each question.