

1) **TABLE COMPLETION** [24 pts] (3 points each) Pretend you are the python interpreter. Evaluate each of the expressions below. Write down the value that each evaluates to. If your answer is a string include quotes around your answer (i.e "hello"). If your answer is a float make sure you include the decimal (i.e 5.0). Write the word error in both columns if the expression causes an error.

Table 1: Expressions

Expression	Return Value of Expression (2 pts)	Data Type of Expression (1 pt)
("sp00ky",)[0][3:0:-1]	"00p"	str
[[8, 1], 4][0][1] + 3	4	int
["a","b","c"].append("d")	None	None Type
len([0, 1, [2, [3]], 4])	4	int
[[1],[2],[8],[3],[5]][-2][0]	3	int
("sweet","", "treat") - ("",)	Error	Error
list(range(len([(1,2),(3,4)])))	[0,1]	list
[9,3] * 2	[9,3,9,3]	list

2) **TABLE COMPLETION** [6 pts] (2 points each) List **all** possible values that could be generated when the following statements involving random numbers are executed. You can assume that the random module has been imported as shown.

import random

Table 2: Random Range

Command	Possible Outcomes
random.randrange(2,7,2)	2,4,6
random.randrange(4)	0,1,2,3
random.randrange(3,-1,-1)	3,2,1,0

3) **MULTIPLE CHOICE** [30 pts] (3 points each) For each of the following multiple choice questions, indicate the best answer by circling it.

a) Which of the following would correctly change a tuple named aTup from ("snickers", "hersheys") to ("m&m", "snickers", "hersheys")

- ☒ A: aTup.append("m&m")
- ☒ B: aTup += ("m&m")
- ☒ C: aTup = ("m&m",) + aTup
- ☒ D: B and C
- ☒ E: All of the above

b) Given aDict = {"Trick" : {"Or" : ("candy", 5), "Treat": ["9", "boo"]}}, what is aDict["Trick"]["Treat"][0]*2

- ☒ A: "99"
- ☒ B: 18 X not int
- ☒ C: "18" X
- ☒ D: "candycandy" X
- ☒ E: aDict is not a valid dictionary

c) What is the proper way to ensure that the contents of a file are saved after writing to the file? Assume that the handle for the file is a variable named aFile.

- ☒ A: close(aFile)
- ☒ B: aFile.close()
- ☒ C: aFile.abort()
- ☒ D: A and B
- ☒ E: None of the above

d) Which of the following is a valid dictionary?

- ☒ A: `its_spooky_season = {'boo',): 3, 'trick': 'or treat'}`
- ☐ B: `its_spooky_season = {'Oct': 31.0, ['candy']: True}`
- ☒ C: `its_spooky_season = {'witches': {1.0: 'bats'}, 'skeletons': None}`
- ☒ D: A and C
- ☐ E: All of the above

e) Which of the following lines of code would remove the space from a variable named `scary` that contains the value 'spooky time'? *string*

- ☒ A: `scary = scary.split()[0] + scary.split()[1]`
- ☐ B: `scary = scary.strip()` \Rightarrow 'spooky time'
- ☐ C: `scary[6] = ""` *immutable*
- ☐ D: A and C
- ☐ E: None of the above

f) The `eat_candy()` function is contained in a module named `halloween`. Which of the following import methods would require referencing the function as `halloween.eat_candy()` when used in code?

I. `import halloween`

II. `from halloween import *`

III. `from halloween import eat_candy`

} from \rightarrow No need to reference.

- ☒ A: I only
- ☐ B: II only
- ☐ C: III only
- ☐ D: II and III only
- ☐ E: I, II, and III

g) What would be printed after running the following code?

```
def func(dictList):
    newDict = {}
    for d in dictList:
        for item in d.items():
            a, b = item
            newDict[b] = a
            newDict[a] = b
    return newDict
```

```
dictList = [{'candy': 'corn'}, {'cotton': 'candy'}, {'pop': 'corn'}]
print(func(dictList))
```

- ☐ A: {'corn': 'pop', 'candy': 'cotton'}
- ☒ B: {'corn': 'pop', 'candy': 'cotton', 'cotton': 'candy', 'pop': 'corn'}
- ☐ C: {'corn': 'candy', 'candy': 'cotton', 'corn': 'pop'}
- ☐ D: {'cotton': 'candy', 'pop': 'corn'}
- ☐ E: None of the above

corn: candy candy: corn cotton: candy pop: corn
 corn: candy candy: cotton cotton: candy pop: corn
 corn: pop candy: cotton cotton: candy pop: corn

Use the following code to answer parts h, i, and j.

```
alist = [['jack'], 'o', 'lantern', [3, 1]]
blist = alist[:]
blist.append('spooky season')
clist = blist
clist[0] = "jack!!!"
blist = clist.append(0)
```

Handwritten notes:
alist = [['jack'], 'o', 'lantern', [3, 1]]
clist blist = [['jack'], 'o', 'lantern', [3, 1], 'spooky season']
jack!!! 10
b -> None

h) What is the value of alist after the code above is run?

- ☐ A: [['jack', 'jack'], 'o', 'lantern', [3, 1]]
- ☒ B: [['jack'], 'o', 'lantern', [3, 1]]
- ☐ C: [['jack'], 'o', 'lantern', [3, 1], 0]
- ☐ D: None (NoneType)
- ☐ E: None of the above

i) What is the value of blist after the code above is run?

- ☐ A: [['jack'], 'o', 'lantern', [3, 1], 'spooky season', 0]
- ☐ B: ['jack!!!', 'o', 'lantern', [3, 1], 'spooky season', 0]
- ☐ C: ['jack!!!', 'o', 'lantern', [3, 1]]
- ☒ D: None (NoneType)
- ☐ E: None of the above

j) What is the value of clist after the code above is run?

- ☐ A: [['jack'], 'o', 'lantern', [3, 1]]
- ☐ B: ['jack!!!', 'o', 'lantern', [3, 1], 'spooky season']
- ☒ C: ['jack!!!', 'o', 'lantern', [3, 1], 'spooky season', 0]
- ☐ D: None (NoneType)
- ☐ E: None of the above

4. Tracing [16 pts] (4 points each) Show exactly what would be printed out when each of the following segments of code are executed. None of these code segments will cause an error. They all have at least partial output that would be shown.

`['pumpkin', 'picking', 'yay!']`

```
def traceMe(myStr):
    myList = myStr.split()
    first = open("aFile.txt", "w")
    first.write(myList[0])
    first.close()
    second = open("aFile.txt", "w")
    second.write(myList[2])
    second.close()
    third = open("aFile.txt")
    print(third.read())
    third.close()
```

`traceMe("pumpkin picking yay!")`

yay!

`[6, 12]`

```
def trace2(aList, bList):
    cList = bList
    for i, j in enumerate(aList):
        if aList[i] % 3 == 0:
            cList.append(j)
            aList[i] = ["candy"]
            aList[i].append("boo")
    print(aList)
    print(bList)
```

`trace2([6, 12], [1, 3])`

`[["candy", "boo"], ["candy", "boo"]]`
`[1, 3, 6, 12]`

```

def tupMagic(aTup, bTup):
    try:
        (happy, halloween) = aTup
        message = aTup[0] + halloween
        print(message)
        print(happy)
    except:
        print("too scary")
    finally:
        print("good night")
tupMagic(("Merry", 5), (3, 5))

```

Handwritten notes:
 - "Merry" is underlined.
 - "5" is underlined.
 - "happy" is underlined.
 - "halloween" is underlined.
 - "message" is underlined.
 - "print(message)" is underlined.
 - "print(happy)" is underlined.
 - "Error" is written next to the except block.
 - "str + int" is written below the except block.

too scary
good night

```

def foo():
    aDict = {'costume': ['ghost']}
    aDict['costume'] = ['harry potter']
    for v in aDict:
        print(aDict[v])
    aDict['costume'][0] = 'itsa me'
    print(aDict)

foo()

```

['harry potter']
{ 'costume': ['itsa me'] }

5. **CODING** [10 pts] Write a function called `fallFest` that takes in one parameter, a list of tuples where each tuple is a pair of words (strings). The function should create and return a list of tuples from the original list. However, a tuple should only be included in the new list if both strings in it start with the same letter. Note that case sensitivity should be ignored, so 't' and 'T' should be considered the same letter. You may assume that the tuples will always contain two strings.

Example Output #1:

```
>>> wordPairs = [("trick", "Treat"), ("chocolate", "gummy"), ("costume",  
"CAT"), ("Treat", "candy"), ("scary", "Mask")]  
>>> print(fallFest(wordPairs))  
[("trick", "Treat"), ("costume", "CAT")]
```

```
def fallFest(wordPairs):  
    new-list = []  
    for worda, wordb in wordPairs:  
        if worda[0].lower() == wordb[0].lower():  
            new-list.append((worda, wordb))  
    return new-list
```


6. **CODING** [14 pts] Write a function called **candyTracker** that takes in two parameters, a list of candies and a dictionary to keep track of the number of certain candies. If the candy doesn't exist in the dictionary, add it to the dictionary. Return an updated version of the dictionary that maps each candy name to the number of times that the candy appears in the list. You may assume that the candies' names will always be in all lowercase letters.

Example Output #1:

```
>>> candy = ["m&m", "snickers", "kitkat", "nerds", "m&m", "nerds", "m&m"]
>>> inventory = {"snickers": 0, "m&m": 0}
>>> print(candyTracker(candy, inventory))
{"snickers": 1, "m&m": 3, "kitkat": 1, "nerds": 2}
```

```
def candyTracker(candy, inventory):
    for name in candy:
        if name not in inventory.keys():
            inventory[name] = 1
        else:
            inventory[name] += 1
    return inventory
```