1. List basics

(Point values: 4 for each part, total 32.) In each part of this problem, you are given a sequence of statements, all beginning the same way, and you are asked to determine what is printed by the final statement. A few things to be aware of:

- All of the code fragments are syntactically correct, but one or more of them will cause run-time errors. When this is the case, you should say that there is an error and describe (in very few words!) what the nature of the error is.
- When the type of an object is printed, it comes out as something like `<class ‘int’>`. It is okay in such instances to just write ‘int’.
- Some expressions can have a value `None` and a type `NoneType`. In such a case you should report this value and type—it should not be considered an error.

(a) \[ \text{t} = [1, 2, [3, 4], ’567’] \]
    \[ \text{print(len(t))} \]

    **Solution:** 4

(b) \[ \text{t} = [1, 2, [3, 4], ’567’] \]
    \[ \text{print(len(t[2]))} \]

    **Solution:** 2 (because `t[2]` is the list `[3,4]`).

(c) \[ \text{t} = [1, 2, [3, 4], ’567’] \]
    \[ \text{print(t[2][2])} \]

    **Solution:** Error, because `t[2]` has only 2 elements, so this will cause an ‘index out of range’ error.

(d) \[ \text{t} = [1, 2, [3, 4], ’567’] \]
    \[ \text{print(type(t[1:3]))} \]

    **Solution:** It will actually print `<class ‘list’>` but I accepted list as correct. `t[1:3]` is the list `[2,[3,4]]`.

(e) \[ \text{t} = [1, 2, [3, 4], ’567’] \]
    \[ \text{print(t+[’eight’])} \]

    **Solution:** `[1,2,[3,4],’567’,’eight’]`
(f) \[ t=[1,2,[3,4],’567’] \\
print(t.append(‘eight’)) \\

Solution: None

(g) \[ t=[1,2,[3,4],’567’] \\
t.append(‘eight’) \\
print(t) \\

Solution: [1,2,[3,4],’567’,’eight’], same as in (e)

(h) \[ t=[1,2,[3,4],’567’] \\
print([s for s in t if len(s)>1]) \\

Solution: [[3, 4], ’567’]

General comment: Most students did well on this problem. Occasionally there were problems getting the indexing right (forgetting that indices begin at 0, not 1). It was common to treat (f) as identical to (e) and (g), but remember that the append method modifies the list but does not return a value. Some students went too far and said that (f) is an error, but it is not (see the instructions concerning this). A few wrote the answer to (e) as \([1, 2, [3, 4], ’567’, [’eight’]]\): this is incorrect because we are concatenating the two lists \(t\) and \([’eight’]\).
2. Transposing an array.

(Point values: 10 for part (a), 12 for part (b), 22 total.) We represent a square array of integers, for example,

\[
\begin{array}{cccc}
8 & 4 & -2 & 7 \\
1 & 6 & 5 & 11 \\
3 & 9 & 2 & 4 \\
2 & 4 & 6 & 8 \\
\end{array}
\]

as a list of lists of integers, so the example above would be represented by the list

\[
m = [[8,4,-2,7],[1,6,5,11],[3,9,2,4],[2,4,6,8]]
\]

The code below was intended to take such an array and return its transpose, which means that each column of the new array is the corresponding row of the original array. In the example above, the result should be:

\[
\begin{array}{cccc}
8 & 1 & 3 & 2 \\
4 & 6 & 9 & 4 \\
-2 & 5 & 2 & 6 \\
7 & 11 & 4 & 8 \\
\end{array}
\]

```python
def transpose(array):
    size=len(array)
    for row in range(size):
        for column in range(size):
            array[row][column]=array[column][row]
    return array
```

Figure 1: Proposed code for transposing a square array. It’s not right!

However, this code does not do the right thing!

(a) What is the value that is actually returned by the function above when applied to the array \( m \)?

Solution: As mentioned in class, the return statement was inadvertently hidden in the printout you saw. The value returned is

\[
[[8,1,3,2],[1,6,9,4],[3,9,2,6],[2,4,6,8]]
\]

or, in rectangular form
(b) Rewrite the function `transpose` so that it does the correct thing (you may assume that the argument represents an array in which the number of rows is the same as the number of columns). You can do this with a nested loop, but you will get an additional point of credit if your function accomplishes this with two lines, using list comprehension, and two additional points if you do it with a single line.

A verbose solution:

```python
def transpose(array):
    newarray=[]
    size=len(array)
    for row in range(size):
        newrow=[]
        for column in range(size):
            newrow.append(array[column][row])
        newarray.append(newrow)
    return array
```
A very compact solution:

def transpose(m):
    return [[m[col][row] for col in range(len(m))] for row in range(len(m))]

Comments: A lot of students had a tough time with this. My hope was that you would see the connection to the image processing assignment, since this illustrates precisely the reason we needed different technique for ‘in-place’ image effects--where each pixel got painted a color that only depended on the original color of that pixel—and effects where the new color of a pixel depends on the original color of different pixels. Trying to do the transpose in place causes you to overwrite pixels before you have the chance to use their original values, something we saw in the bad implementation of the horizontal flip effect as well. If you used the incorrect transpose code above, and applied it to our old friend the clown:

![Clown](image)

then the result is a bit disturbing:
The code for the correct transpose used exactly the method demonstrated in the assignment for creating a brand new array. The result is the original image, reflected about the diagonal line from upper left to lower right.

Some students had very imaginative ideas about what the flawed code would produce; others could not see why it would give an incorrect result at all. A fair number saw that we would get a mirror image like the creepy one above, but got the rows and columns backwards. Some saw that yes, we really would need to make a new array to correct the problem, but fumbled on exactly how to do this, writing something tantamount to

```
newarray=[ ]
newarray[3][4]=5,
```
which will result in a range error.

3. Extracting information from a list of words.

(Point values: 7 for (a), 8 for (b), 15 total.) What does the function below do when applied to a list of strings? Part (a) asks you to answer this question for a special case, and part (b) asks you for a general description.

```python
def g(wlist):
    m=max([len(w) for w in wlist])
    return [w for w in wlist if len(w)==m]
```

Figure 2: What does this function return?

(a) What is the value of

(i) \[ g([‘dog’,’horse’,’bear’]) \]

What is the value of

(ii) \[ g([‘dog’,’horse’,’bear’,’zebra’]) \]

(b) Describe in general what this function returns when applied to a list of strings. The answer to this question should not be a step-by-step description along the lines of ‘The function looks at every word in wlist and then calculates the...’. Instead it should be a succinct description of the end result, along the lines of ‘The function returns the longest string in the list that begins with the letter Q.’

Solution: The first expression has the value [’horse’] and the second [’horse’,’zebra’]. In general, the function returns a list of all the words of maximum length in wlist.

Comments: A bit of a gift, compared to the stressful Problem 2. Almost everyone got it right...

...sort of. I was not as fussy as I might have been, and saw many answers like ‘it returns the words of maximum length in wlist’, even though what it really returns is a list of the words of maximum length. I accepted this, as long as you wrote the solutions to part (a) carefully, but deducted points if you left off the brackets.
4. Representing a family tree with a Python dictionary.

(Point values: 5 each for (a)-(c), 10 each for (d)-(e), 35 points total) The figure on the last page shows the family tree of the powerful Ugg dynasty, which flourished 20,000 years ago in a network of caves in the foothills of the Pyrenees.

Females are marked in red and males in black. So, for example, Mooga is Shocka’s mother and Chooga is his father. Ug is Shocka’s maternal grandfather and Shug is Shocka’s paternal grandmother.

We represent the family tree by a Python dictionary \( d \), which looks, in part, as follows:

\[
\{ \ldots \text{'Rooga'}: (1, \text{'Chug'}, \text{'Wug'}), \text{'Skooga'}: (0, \text{'Chug'}, \text{'Wug'}), \text{'Ug'}: (0,), \text{'Bug'}: (1,\ldots) \ldots \}
\]

That is, every person’s name is a key in the dictionary, and the associated value is a tuple giving, first, the individual’s gender (0 for male, 1 for female) and then the father’s name followed by the mother’s name. Observe that for individuals at the top of the tree, no parents are given, so this tuple contains just a single entry.

For (a)-(c), give the value of the expression. For (c), the precise answer depends on the order in which the items were added to the dictionary, so there are many possible correct answers.

(a) \( \text{len}(d[\text{'Booga']]) \)

Solution: 3 (Booga’s gender and the names of his two parents.)

(b) \( \text{type}(d[\text{'Mooga'][1]]) \)

Solution: str, but ‘string’ was okay, too. The value of the expression is ‘Ug’, the name of Mooga’s father.

(c) \( [\text{person for person in d if len(d[person])==1}] \)

Solution: This will be a list of all the individuals with no parents (i.e., those at the top of the tree). For example, \( ['Ug','Bug','Mug','Shug','Tug','Chug','Wug'] \) is one possibility.

(d) Write an expression involving a variable person whose value is a list of all the children of that person. So for example, if person has the value ‘Mooga’, then the
expression should have the value ['Chocka','Shocka','Rocka'].) This can be done with a single line, but it is also okay to build the expression in several steps. If the person has no children, the resulting list should be empty.

Solution: Simplest way to write it is

[child for child in d if person in d[child]],

but it was okay to give more extensive code for creating this list.

(e) The function on the last page takes a dictionary as an argument and returns another dictionary. What do you get if you apply it to the dictionary d above? Answer this question by giving a few key-value pairs, and a general description of what the new dictionary contains. To see what is going on, try simulating the code assuming the dictionary contains only a few of key-value pairs on the preceding page. Just describe the end result, not what the function is doing at every step.

Solution: The function returns the inverted dictionary. The keys are the same, but now each value in a key-value pair is a 2-tuple giving first the individual’s gender, and second, a list of the individual’s children. For example, two of the key-value pairs are

'Ug':(0,['Ooga','Booga','Mooga'])
'Rooga':(1,[])

Comments: People generally did fine on (a)-(c). A common error in (d) (which cost you only a point) was to write

[child for child in d if d[child][1]==person or d[child][2]==person]

But this will cause an error if child has no parents. Part (e) was more challenging, apparently.

Results: The median was 81.5. The grade distribution is shown in the histogram below and is actually quite extraordinary—I don't think I've ever seen one that looks like this!

1 There is no error in this code: You might be troubled by what x[1:] does when x is an indexed sequence that contains only one element---in that case x[1:] is an empty sequence.
Figure 3. This doesn’t look like the distribution of grades on an exam, but it is!
```
def idict(family):
a={}
    for person in family:
        a[person] = (family[person][0], [])
    for person in family:
        for parent in family[person][1:]:
            a[parent][1].append(person)
    return a
```

Figure 4: What does this function return when applied to the dictionary given in the problem?

Figure 4. Meet the Uggs.