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

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

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

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

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

(f) \[t=[1,2,[3,4],’567’]\]
\[\text{print}(t.\text{append}(’eight’))\]

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

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

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

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

\[m=\begin{bmatrix}
[8,4,-2,7],[1,6,5,11],[3,9,2,4],[2,4,6,8]
\end{bmatrix}\]

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:

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

```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 \(m\)?

(b) Rewrite the function \texttt{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.
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) $g([\text{'dog'},\text{'horse'},\text{'bear'}])$ ?

What is the value of

(ii) $g([\text{'dog'},\text{'horse'},\text{'bear'},\text{'zebra'}])$ ?

(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.’
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:

```python
{...'Rooga': (1, 'Chug', 'Wug'), 'Skooga': (0, 'Chug', 'Wug'), 'Ug': (0,),'Bug':(1,...}
```

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) len(d[ ‘Booga’])
(b) type(d[‘Mooga’][1])
(c) [person for person in d if len(d[person])==1]

(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.

(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.1 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.

---

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.
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 3: What does this function return when applied to the dictionary given in the problem?

Figure 4. Meet the Uggs.