Question J - Daniyar Nazarbayev [H00204990].

1.

1. { 0:"zero", 1:"one", 2:"two" }[1]

The collection in particular is a dictionary – an array that has names for its values.

[1] should access the second index of the array – 'one'.

2. range(1000)[0]

range(1000) should create a list from 0 to 999 – 1000 values.

But it needs to be wrapped around list() first.

Having [0] should just return an int. In this case – '0'.

3. for x in range(1,1000): print(list(enumerate(range(x)))[0][0])

This is a loop that should iterate 999 times – because it starts at 1 but ends at 999.

During each iteration a new list is created – a list of tuples.

How big it is depends on range(x).

Then enumerate function is applied, which indexes every value in the range, making tuples consisting of 2 values each.

list function converts it all into a list, and then index [0][0] is the result that gets printed.

It is sort of like a multidimensional array. [0][0] chooses the first index of the list and the first index in the tuple – which is the value that was created by the enumerate function.

Which should be '0' no matter what, since the count starts from 0.

The range will get bigger making a bigger list, but the bottom line is that the count on enumerate will always start from 0.

That should repeat 998 more times.

4. "hello world"[:-2]

This should return a string, except without the last 2 characters - "hello wor".

2.

x=[""] - is a list with an empty string value.

for i in x: x.append("") – this is a for each loop that should loop for every element in the list.

Originally it should only loop once since there is only one element, but during each loop an extra element is appended to the list.

So basically this is an infinite loop, and it keep going until it runs out of memory for storing the list x.

```
>>> x = [""]
>>> print(x)
['']
>>> for i in x:
        print(x)
        x.append("")


['']
['', '']
['', '', '']
['', '', '', '']
['', '', '', '', '']
['', '', '', '', '', '']
['', '', '', '', '', '', '']
['', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '']
['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
 '']
```

In the second snippet, x[:] is used. [:] makes a deep copy of the list. This means that a temp variable is created somewhere in the memory and contents of x is copied there. The list x still changes, after the first loop it will become [“”,””], but now the for each loop checks for elements in the temp variable rather than x. It is a deep copy, meaning that the temp variable is not pointing at x at all, it’s only the values that are copied.

```
>>> x = [""]
>>> for i in x[:]:
        print(x)
        x.append("")


['']
>>> print(x)
['', '']
>>>
```

3.

x = [[]] – x is a list that stores an empty list.

x[0], which is [] is changed by x[0].extend(x).

Extend adds content of one list to the other, meaning it should combine lists x and x[0] into one and store them in x[0].

But x is [[]], and when extending the outer brackets will be removed, only adding [] to x[0].

The result of x should be something like this [[[ ]]]. This is the new x, but now x[0] has changed, becoming [[ ]].

I assume that x is a recursive list of some sort.

After researching, I found the name of this thing – ellipsis. It is said that it is used for slicing multidimensional lists.

In the loop, the code tries to break down the list, by making x = x[0] in a while True loop, but no matter how hard it will try, it won't be able to, since the list is recursive. The x after the first iteration equals to [[…]], and this value will be printed infinitely.

4.

{[0]:0}[[0]]

from the looks of it, this is a dictionary, that has a key of type list and a value of type int.

When trying to initialize a dictionary alone, like this: {[0]:0}

It does not work and gives an error - unhashable type: 'list'.

After googling for a minute I found this:

> https://stackoverflow.com/questions/7257588/why-cant-i-use-a-list-as-a-dict-key-in-python
>
> and this
>
> https://wiki.python.org/moin/DictionaryKeys

It says that lists cannot be used as keys because they cannot provide a proper hash function.

Why? Because lists are mutable, meaning that a new hash must be made every time a change in the list is made.

Tuples on other hand can be used as keys, since they cannot be altered.

Like in here:

{(0):0}[(0)]

For exercises 5-9, check the code in the zip file.

8.

Just want to give a short explanation on 8th exercise. There we are asked to get numbers that are both perfect squares and are pentagonal. As I was making it, I made 4 answers to this exercise.

Originally I only had 2nd/3rd functions (they are pretty much identical), but they had a very high complexity. I simply took the pentagonal formula and added another loop to it. In that loop, it should look for the number that is supposed to be its square. $O(n^2)$ complexity.

Then I made the 1st function, where I use the list comprehensions provided in the exercise. I turn one of those lists into a set, and then I use intersection function to check which elements are in both the lists. Originally I used "is in" to check both lists – it was very slow. As I was going through the internet, I saw a lot of suggestions to use set to check for the same values.

Set made this function much faster – actually it is pretty much the fastest function here. That said it has one drawback. The function relies on first creating the 2 lists, and then picking out identical values. The program runs out of memory when making both lists create 100,000,000 values *each*.

Then I created the 4th function, which is basically the pentagonal formula provided, plus a filter. For the filter, I imported a math module, which has sqrt() function. The sqrt() returns the exact root, so I cannot check whether it is a perfect root with that function alone. To finish the job, I used is_integer() which should give true for all integers, and even floats/doubles like 1.0, 2.0 etc.

9.

As for my 9th question, everything is pretty straight forward. For my multiplication I decided to just add the number a set amount of times. I am not sure whether that's how ALU does the multiplication, but my abstraction level isn't that high to implement multiplication with nested brackets just like that.

This question had my scratching my head for a few hours. My multiplication function would not work, to the point where I had to make print statements everywhere in the function to check whether everything went wrong. In the end, the cause of that was the add function. Every time I would do something like this:

```
list2.pop()
```

or

```
list2 = list2[0]
```

I would be losing my list2, bit by bit. I implemented a temp variable that would store a copy, and would assign it back at the end of the function.
