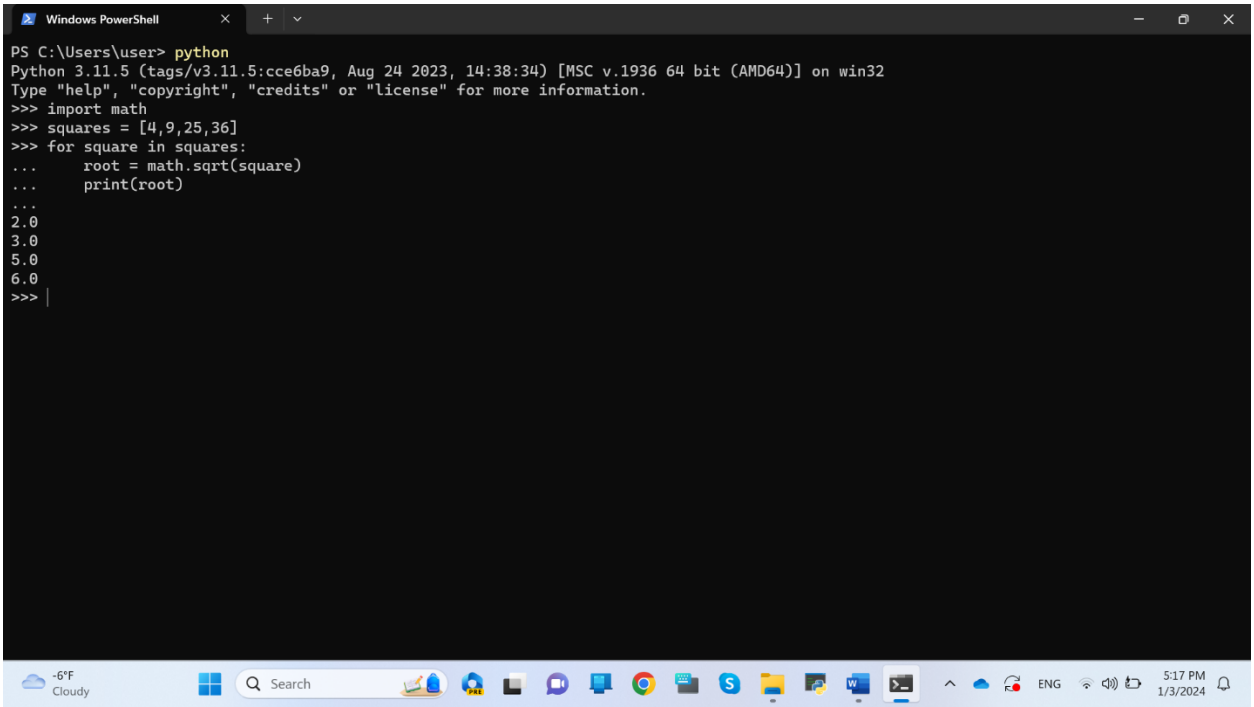
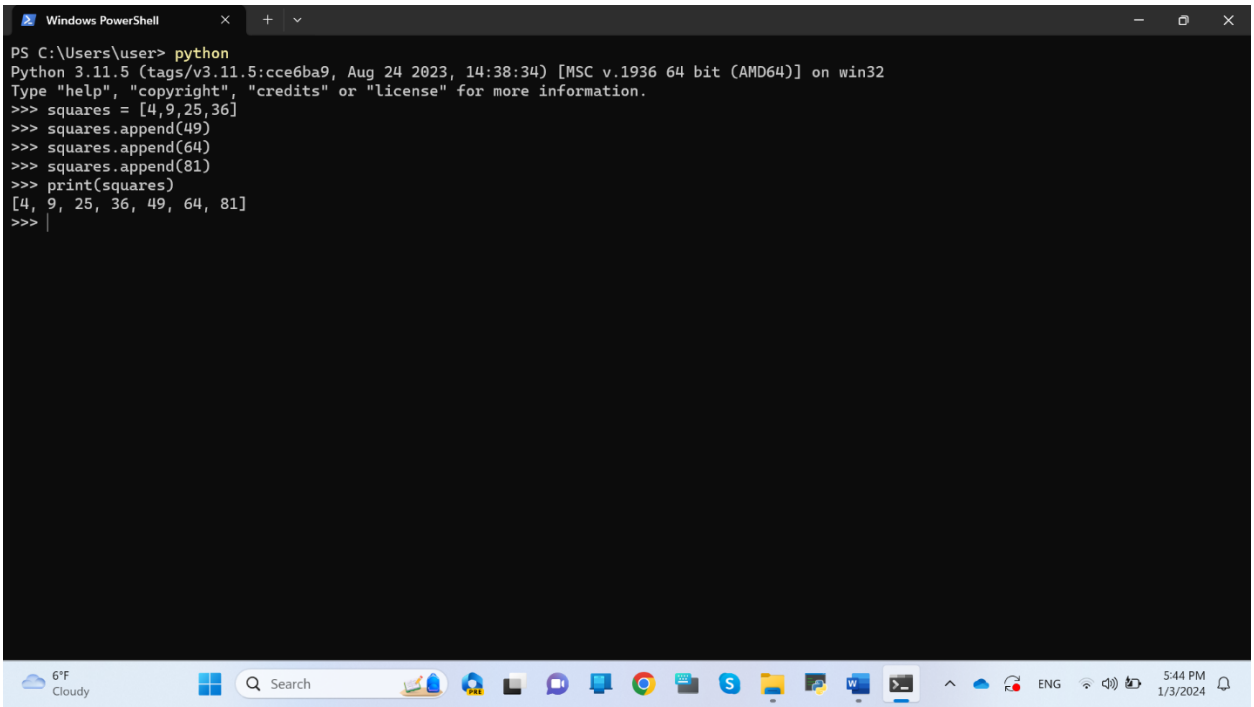
Name = Roshan Chaudhary

TASK: Write a for..in loop that iterates over all the elements of the squares list and prints the square root of each to the screen. Hint: you may want to import a function from the math module to help achieve this.



TASK: Write some code that uses the `append()` method to add the next three square values (49, 64, 81) to the end of the `squares` list.



TASK: Write some code that uses the `extend()` method to add the next three square values, starting at `121` (`11` x `11`), to the end of the `squares` list.

```
PS C:\Users\user> python
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> squares = [4,9,25,36]
>>>
>>> squares.extend([121,144,169])
>>>
>>> print(squares)
[4, 9, 25, 36, 121, 144, 169]
>>>
```

TASK: Write some code that uses the `insert()` method to insert the value `2`, to the very beginning of the `squares` list.

```
PS C:\Users\user> python
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> squares = [4,9,25,36]
>>> squares.extend([121,144,169])
>>> print(squares)
[4, 9, 25, 36, 121, 144, 169]
>>> squares.insert(0,2)
>>> print(squares)
[2, 4, 9, 25, 36, 121, 144, 169]
>>>
```

TASK: Write some code that uses the `remove()` method to remove the value `49` from the `squares` list. Print the list afterwards to ensure the value has indeed been removed.

```
PS C:\Users\user> python
Python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> squares = [4,9,25,36,49]
>>> squares.extend([121,144,169])
>>> print(squares)
[4, 9, 25, 36, 49, 121, 144, 169]
>>> squares.insert(0,2)
>>> print(squares)
[2, 4, 9, 25, 36, 49, 121, 144, 169]
>>> squares.remove(49)
>>> print(squares)
[2, 4, 9, 25, 36, 121, 144, 169]
>>>
```

TASK: Write some code that uses the `remove()` method to remove the value `3` from the `squares` list. Notice how an error is generated since the given value was not present.

```
>>> squares.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'squares' is not defined
>>>
```

TASK: Create a simple list that contains the values `[1, 2, 3, 1, 2]` and then use the `remove ()` method to remove the value `2`. Which value is removed?

```
>>> squares = [1,2,3,1,2]
>>> squares.remove(2)
>>> print(squares)
[1, 3, 1, 2]
```

TASK: Write some code that uses the `pop()` method to remove and display the first value of the `squares` list. Print the list afterwards to ensure the value has been removed.

```
>>> last_letter = squares[-1]
>>> print("last letter of the list is:",squares)
last letter of the list is: [1, 3, 1, 2]
>>>
```

TASK: Write some code that uses the `sort()` method with no arguments, to alphabetically sort the exact list of names shown below. Display the list after the sort has been called.

```
names = [ "Eric", "anna", "Sophie", "sam" ]
```

```
>>> names = ["Eric","anna","sophie","sam"]
>>> names.sort()
>>> print(names)
['Eric', 'anna', 'sam', 'sophie']
>>>
```

TASK: Improve your previous solution so that the list is sorted correctly, ignoring the case used to write the names. To achieve this you will have to include a `key` argument in the form of a *lambda expression* that returns each string as uppercase letters only. Hint: you can use the `str.upper()` method to convert a name to uppercase letters.

```
>>> names = ["Eric","anna","sophie","sam"]
>>> names.sort(key=lambda x: x.upper())
>>> print("the sorted list is: ",names)
the sorted list is:  ['anna', 'Eric', 'sam', 'sophie']
>>>
```

TASK: Write some code that uses the `reverse()` method to reverse the values of the `squares` list. Print the list afterwards to ensure the values have been reversed.

```
>>> squares = [1, 4, 9, 16, 25]
>>> squares.reverse()
>>> print("The reversed list is:", squares)
The reversed list is: [25, 16, 9, 4, 1]
>>> |
```

TASK: Write some code that finds the index of the colour `blue`.

```
>>> colours = ["red", "green", "yellow", "blue", "red"]
>>> print(colours.index("blue"))
3
>>>
>>>
```

TASK: Write some code that makes a copy of the `colours` using the `copy()` method. Then make some changes to the original list. Print the contents of the copied list to ensure these changes have not affected the copy.

```
>>> colours = ["red", "blue", "green", "yellow"]
>>>
>>> colours_copy = colours.copy()
>>>
>>> colours.append("orange")
>>> colours.remove("blue")
>>>
>>> print("Original list:", colours)
Original list: ['red', 'green', 'yellow', 'orange']
>>> print("Copied list:", colours_copy)
Copied list: ['red', 'blue', 'green', 'yellow']
```

TASK: Write some code that uses a list *comprehension* to create a list called `cubes` that contains the cubed values (`x * x * x`) of the numbers from `2` to `20` inclusive

```
>>> cubes = [x ** 3 for x in range(2, 21)]
>>>
>>> print(cubes)
[8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728, 2197, 2744, 3375, 4096, 4913, 5832, 6859, 8000]
>>>
```

```
some_users = [u for u in all_users if len(u) < 8]
```

TASK: Create a tuple called `address` that includes your own “house number”, “street” and, “postcode” as three different values.

address = (123, "kathmandu", "1234")

TASK: Try entering the above examples to create single element tuples. Then use the `type()` function to examine the data-type of the created variables.

Hint: For this reason it is common to include a trailing comma even when one is not needed. For example:

```
Student = ("Griffin, P.", 2, 38.2,)
```

TASK: Use *sequence unpacking* to extract the values you stored within the `address` tuple earlier. Unpack the tuple into variables named `house_num`, `street` and `postcode`

```
>>> address = (123,"kathmandu","1234")
>>> house_num, street, postcode = address
>>> print("house_number:",house_num)
house_number: 123
>>> print("street:",street)
street: kathmandu
>>> print("postcode:",postcode)
postcode: 1234
>>>
```

TASK: Write some code that calls the `print()` function to output the contents of the `address` tuple you created earlier. Ensure you use the '`*`' prefix so that the elements are extracted before being passed to the function. Compare this with a version of the same code that calls the print() function without using the '`*`' prefix,

```
>>> address = (123,"kathmandu","1234")
>>> print(*address)
123 kathmandu 1234
>>>
```

TASK: Look at each of the phrases below and ensure you understand what each of these means. For any that you do not understand, do a little research to find a definition of each term. This research may involve looking back over these notes, or the associated lecture notes. It may also involve searching for these terms on the Internet.

- Method
- List comprehension
- Tuple
- Tuple Packing
- Sequence Unpacking

Method: In Python, a method is a function that is associated with an object. It is called on an object and can access and modify the data within that object.

List Comprehension: List comprehension is a concise way to create lists in Python. It consists of an expression followed by at least one for clause and zero or more if clauses. It provides a more readable and compact syntax for creating lists.

Tuple: A tuple is an ordered and immutable sequence of elements. Tuples are defined using parentheses and can contain elements of different data types.

Tuple Packing: Tuple packing refers to the process of combining multiple values into a single tuple. This happens automatically when you separate values with commas.

Sequence Unpacking: Sequence unpacking is the process of extracting values from a sequence (like a tuple or a list) and assigning them to individual variables. It's often used with multiple assignment statements.