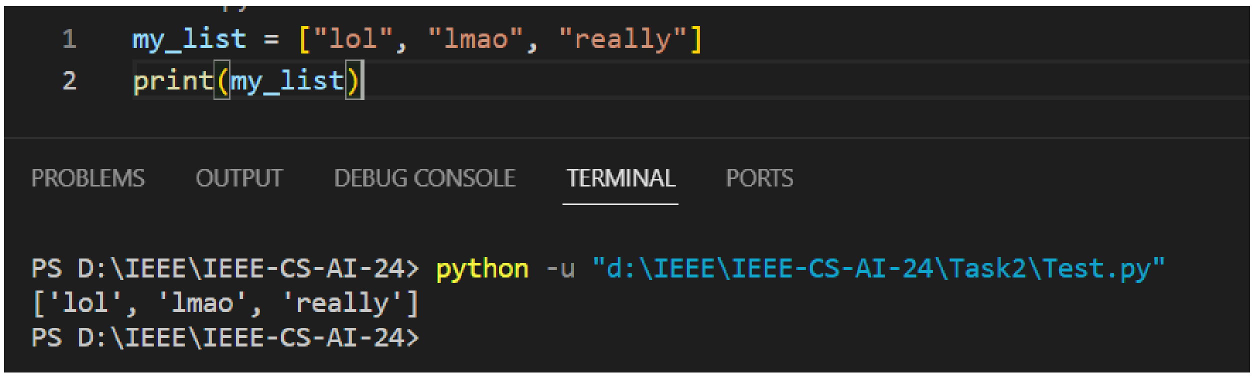
Python summary

Collections:

- List
- Set
- Tuple

1. List
   Declaration ⇒ mylist = list() or mylist = [ ]
   lists are ordered and changeable, duplicates are okay
   Ex:



```python
my_list = ["lol", "lmao", "really"]
print(my_list)
```

```
PS D:\IEEE\IEEE-CS-AI-24> python -u "d:\IEEE\IEEE-CS-AI-24\Task2\Test.py"
['lol', 'lmao', 'really']
PS D:\IEEE\IEEE-CS-AI-24>
```

   Functions ⇒
   - You can check if an element exist in a list or not by using the keyword “in” → “no” in my_list → if the string no exist in my_list it will return true if not it will return false
   - len(my_list) → return the length of the list which is the number of elements in it
   - You can use indexing as in strings → my_list[0] = “anything” → you can change the element or just access it using indexing
   - my_list.append(“oh no”) → it will add the string “oh no” to the list
   - my_list.remove(“really”) → it removes the argument given from the list

- my_list.insert(0, “really”) → it also add an element to the list but the difference between it and append is that it insert it at the required index not at the end
- my_list.sort() → it sorts the list alphabitcally
- my_list.reverse() → it reverse the order of the list
- my_list.clear() → it clears every element in the list and become empty
- my_list.index(“oh no”) → it will return the index of the value given
- my_list.count (“really”) → it counts how many of the value given does the list have

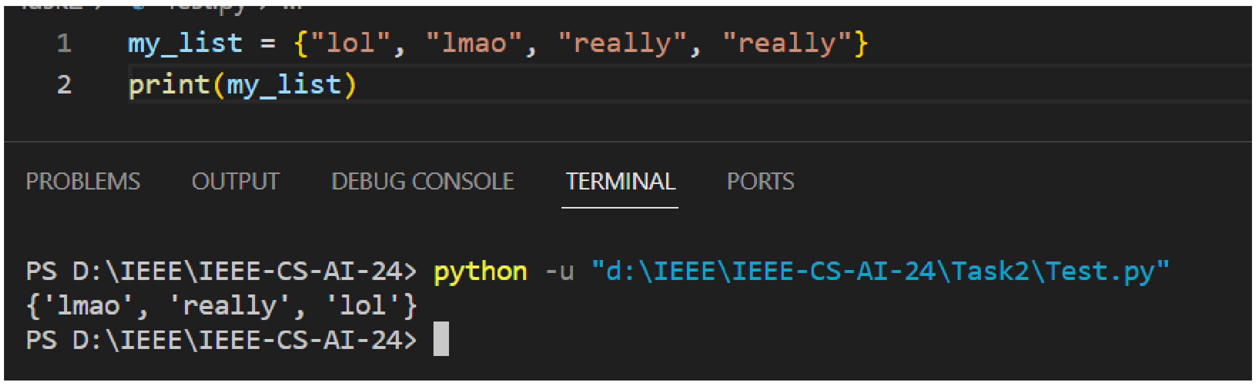
2. set

   Declaration ⇒ my_set = set() or my_set = { }

   Sets are unordered and immutable, but add/remove are ok.

   No duplicates

   Ex:



```
my_list = {"lol", "lmao", "really", "really"}
print(my_list)
```

```
PS D:\IEEE\IEEE-CS-AI-24> python -u "d:\IEEE\IEEE-CS-AI-24\Task2\Test.py"
{'lmao', 'really', 'lol'}
PS D:\IEEE\IEEE-CS-AI-24>
```

   Functions ⇒

   - No indexing is allowed because the elements are unordered
   - my_list.add(“looool”) → it adds elements to the set
   - my_list.remove(“lol”) → it removes an element
   - my_list.pop() → it removes a random element
   - my_list.clear() → it removes all the elements

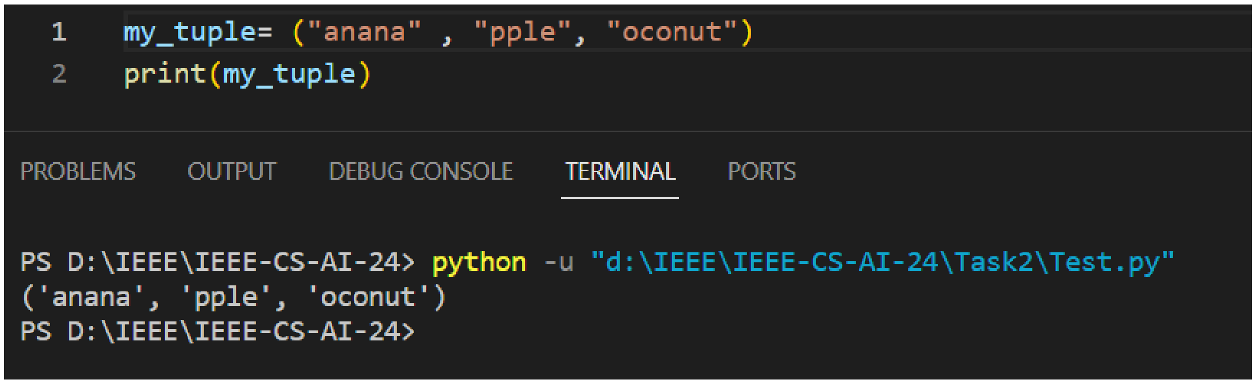
3. Tuples

   Declaration ⇒ my_tuple = tuple() or my_tuple = ( )

   Tuples are ordered and unchangeable. Duplicates are ok.

   It is faster than lists

Ex:



```python
my_tuple= ("anana" , "pple", "oconut")
print(my_tuple)
```

```
PS D:\IEEE\IEEE-CS-AI-24> python -u "d:\IEEE\IEEE-CS-AI-24\Task2\Test.py"
('anana', 'pple', 'oconut')
PS D:\IEEE\IEEE-CS-AI-24>
```

Functions ⇒

- my_tuple.index(“anana”) → it will return the index of the value in the tuple, if it doesn’t exist it will return -1
- my_tuple.count(“pple”) → it return the count of the given value in the tuple

2d collections

It’s just a collection made up of more collections
Like a list of lists
Children = [‘mohamed’, ‘nadeen’, ‘ola’]
Grandchildren = [‘soha’, ‘aya’, ‘ali’]
Parents = [‘amr’, hossam’, ‘layla’, ‘phope’]
Family = [childern, grandchildren, parents]
And that’s simply a collection made up of more collections

Dictionary

It’s a collection of {key: value} pair, ordered and changeable, duplicates are not allowed
Declaration ⇒ my_dict = dict() or my_dict = {name : ‘anis’} you need to assign a key value pair to be regocnized as a dictionary

Functions ⇒

- my_dict.get(key) → it returns the value associated with this key

Ex: my_dict.get(name) → will return “anis”
If it doesn’t find a key it will return None

- my_dict.update({age: 21}) → it will add this pair to the dictionary
  Using the update method you can insert a new key value pair or update an existing one
- my_dict.pop(name) → it removes the pair with the key in the argument
- my_dict.popitem() → it removes the last inserted pair
- My_dict.clear → that will clear the dictionary
- my_dict.keys() → it returns the keys only of the the dictionary
- my_dict.values() → it returns all the values of the dictionary
- my_dict.items() → it returns 2d list of tuples

Random numbers

You have to import random library
Import random
number = random.randint(1, 6) → this will generate a random number from 1 to 6
You can place variables instead of numbers as long as they have integer values in it
number = random.random() → that will return a random floating number between zero and one

options = (“rock”, “paper”, “scissors”)
option = random.choice(options) → this will return a random choice form a sequence of elements

options.suffle() → it will shuffle the order of the elements in options assuming it’s a list

Function

Declaration ⇒ def name_of_the_function(arguments, argument2):
Body_of_the_function

```
Ex: def say_hello(name):
            print(f"hello {name}")
say_hello("anis")
```
⇒ hello anis

Return → functions can return value if needed or end the function with a return

```
Ex: def add(num1, num2):
        num3 = num1 + num2
        return num3
```
This function will return the summation of the two numbers we give it

Default arguments

You can assign a value to an argument so if the use didn’t enter it you can use the default one

```
Ex: def net_pric(list_price, discount=0,tax=0.05):
            return list_price * (1 - discount) * (1 + tax)
print(net_price(500)) → 525.0 it used the default arguments
```

Keyword arguments

You can use it assign values to variables in the function right away

```
Ex: def say_nonsense(first, last):
            print(f"{first} {last}")
say_nonsense(last="oh no", first = "oh yes")
```
The order really doesn’t matter here but if you used keyword argument on one argument and the other is positional you need to do the positional first

Arbitary arguments

We use *args with the unpacking operator to allow multiple arguments and **kwargs to allow keyword arguments

```
Ex: def add(*args):
        sum = 0
        for arg in args:
            sum += arg
        return sum
```
This function can sum any number of arguments

add(1, 2, 3) → 6, add(5, 6) → 11, add(1) → 1

```
Ex: def print_address(**kwargs):
        for key, value in kwargs.items():
                print(f"{key} : {value}")
```

You can treat kwargs as a dictionary

```
print_address(street = "al nasser streat", city = "faqous")
street : al nasser streat
city : faqous
```

Error handling

We use try and except to avoid crashing the program due to an error

Ex:

```
try:
        num = int(input())
        num2 = int(input())
        result = num / num2
        print(result)
except Exception:
        print("oh no something is wrong but I won't tell you what!")
```

Here if an error occurred in the try block the except block will catch it and the program continue

- It's not good practice to use general exception as you need what went wrong

```
try:
        num = int(input())
        num2 = int(input())
        result = num / num2
        print(result)
except zerodivisonerror:
        print("oh no you can't divide by zero!")
except ValueError:
        print("you can only insert integers or floats")
```

except Ecxeption:
        print(“okay I don’t know what happened but you did something wrong”)
Here we specified the type of the error so we can handle it correctly

- You can also insert an else statement at the end if no exception or error raised the else statement will be executed

Finally → this will always execute at the end even if an error raised

Files

File dedication
First you need to import the os module
import os
path = the path of the file
if os.path.exists(path) → to check if the location really exists
if os.path.isfile(path) → to check if the path leads to a file or not
if os.path.isdir(path) → to check it the path is a folder or not
Reading files
with open(“path of the file”) as file:
        print(file.read)
here we read the file in the path included and print it
if the file doesn’t exist it will result in an error
The with statement open and close the file automatically so you don’t have to do anything else
Writing files

```
text = "this is the texttttttttttt"
with open('test.tx', 'w') as file:
        file.write(text)
```

- if the file does not exist it will create one with the name included
- If the file exist and not empty it will overwrite it
- To append and not overwrite the existing text use the ‘a’ argument instead of ‘w’

Copying a file
We will use the shutil

We have 3 functions

1. copyfile() ⇒ copies content of file
2. copy() ⇒ copyfile() + permission mode + destination can be a dircotory
3. copy2() ⇒ copy() + copies metadata

Ex:

```
import shutil

shutil.copyfile("test.txt", "copy.txt")
```

It's the same with the other two

Moving a file

Here is an example to explain how it's done:

```
import os

source = "test.txt"
destination = "D:\\test.txt"

try:
    if os.path.exists(destination):
        print("there is  already a file there")
    else:
        os.replace(source, destination)
        print(source + " was moved")
except FileNotFoundError:
    print(source + " was not found")
```

- We used try and except to handle errors
- We checked if there is already a file where we want to move our file or not
- We used os.replace method to do the moving part as we need the source of file and the destination we need

Deleting files

```
import os
import shutil

```

```
path = "text.txt"
path2 = "folder"
os.remove(path)
os.rmdir(path2)
shutil.rmtree(path2)
```

- To remove a file we use → os.remove(path of the file)
- To remove an empty folder → os.rmdir(path of the folder)
- To remove folder with files in it we have to include the shutil module and use → shutli.rmtree(path of the file)