First the function

In Python, functions are organized, reusable blocks of code that perform specific tasks. Functions provide modularity and abstraction, allowing you to break your program into smaller, more manageable parts

1-Function definition:

When you define a function, you are essentially creating a reusable block of code that performs a specific task. This helps organize the code and make it more modular, thus improving readability and maintainability.

```
def function_name(parameters):
    # Function body
    # Code to be executed
    # Return statement (optional)
```

2-**Function Call**:

Once you've defined a function, you can call it from other parts of your code to execute the task it performs. During the function call, you provide the necessary arguments (or inputs) that the function expects.

```
Function_name(Parameters)
```

3-**Return Statement**:

Functions can optionally return a value back to the caller using the `return` statement. This allows the function to produce an output that can be used by other parts of the program

data structure [lists, dictionary, Tuple ,sets]

Data structures are fundamental building blocks in programming, allowing us to organize and manipulate data efficiently. There are several built-in data structures in Python, each with its own properties and use cases. In this article, we will look at four commonly used data structures: lists, sets, tuples, and dictionaries. We discuss their properties and differences, and provide examples to illustrate their use.

First the list

Lists are ordered collections of items that are mutable, meaning they can be modified after creation. They are one of the most versatile data structures in Python and can hold elements of different data types.

```
my_list = [2, 'Aya', True, [0, 1]]
```

Characteristics:

Ordered: Elements in a list are stored in a specific order, and their positions can be accessed using indices.

Mutable: Lists can be modified after creation. You can add, remove, or modify elements.

Allows duplicates: Lists can contain duplicate elements.

Built in methods:

append()	Adds an element at the end of the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position

Sets

Sets are unordered collections of unique elements. They are useful for tasks that involve checking membership or eliminating duplicate values.

Example:

```
my_set = {1, 2, 3, 'apple'}
```

Characteristics:

Unordered: Elements in a set are not stored in any particular order, and their positions cannot be accessed using indices.

- Mutable: Sets can be modified after creation. You can add or remove elements.
- Unique elements: Sets do not allow duplicate values. If you try to add a duplicate element, it will be ignored.

copy()	Returns a copy of the set
difference()	Returns a set containing the difference between two or more sets
difference_update()	Removes the items in this set that are also included in another, specified set
discard()	Remove the specified item
intersection()	Returns a set, that is the intersection of two other sets

Tuples:

Tuples are ordered collections of elements that are immutable, meaning they cannot be changed after creation. They are often used to store related pieces of data together.

```
my_tuple = (1, 'apple', True)
```

- Characteristics:
  - Ordered: Similar to lists, elements in a tuple are stored in a specific order, and their positions can be accessed using indices.
  - Immutable: Once a tuple is created, its elements cannot be modified, added, or removed.
  - Allows duplicates: Like lists, tuples can contain duplicate elements.

Dictionaries: Dictionaries are collections of key-value pairs, where each key is associated with a value. They are useful for mapping one piece of data to another.

Example:

```
my_dict = {'name': 'John', 'age': 30, 'city':
'New York'}
```

- Characteristics:
  - Unordered: Similar to sets, the elements in a dictionary are not stored in any particular order.
  - Mutable: Dictionaries can be modified after creation. You can add, remove, or modify key-value pairs.
  - Keys are unique: Each key in a dictionary must be unique. If you try to add a duplicate key, it will overwrite the existing value.
- **Dictionary Methods**

copy()	Returns a copy of the dictionary
fromkeys()	Returns a dictionary with the specified keys and value
get()	Returns the value of the specified key
items()	Returns a list containing a tuple for each key value pair
keys()	Returns a list containing the dictionary's keys
pop()	Removes the element with the specified key
popitem()	Removes the last inserted key-value pair
setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

Error handling,

exception handling, is a programming technique used to gracefully handle unexpected errors or exceptional conditions that may occur during the execution of a program.

types of error handling

Basic Error Handling:

You can use the `try` and `except` blocks to catch and handle exceptions that may occur within your code .like divided by zero

```
try:
    result = 10 / 0  # This will raise a ZeroDivisionError
except ZeroDivisionError:

    print("Error: Division by zero!")
```

Handling Multiple Exceptions:

use multiple `except` blocks to handle different types of exceptions separately

```
try:
    file = open("nonexistent.txt", "r")
except FileNotFoundError:
    print("Error: File not found!")
except PermissionError:
    print("Error: Permission denied!")
```