Question 65: )How Many Basic Types of Functions Are Available in Python?

the **two main basic types** are:

1. **Built-in functions**
2. **User-defined functions**

Common built in functions

1.input and output functions

print() → prints to console

input() → takes user input

2.Type and conversion

type() → returns data type

int(), float(), str() → convert between types

bool(), list(), tuple(), set(), dict() → convert objects

3.

**3. Math Functions**

* abs() → absolute value
* pow(x, y) → x raised to power y
* round() → round number
* max(), min() → largest/smallest
* sum() → sum of iterable

**4. Sequence Functions**

* len() → length of sequence
* sorted() → returns sorted list
* reversed() → returns reverse iterator
* enumerate() → returns index + value pair
* zip() → combine iterables

**5. Utility Functions**

* id() → memory address of object
* help() → shows documentation
* dir() → shows attributes/methods of object
* eval() → evaluates string as expression
* isinstance() → checks object type

**6. Any / All**

* any() → True if at least one element is true
* all() → True if all elements are true

**Question 1) What are the types of Applications?**

Applications (or "apps") can be categorized in **different ways** depending on context — for example, based on **platform**, **purpose**, or **deployment model**. Here’s a structured breakdown:

**1. Based on Platform**

* **Desktop Applications** – Run on personal computers (e.g., MS Word, Photoshop).
* **Web Applications** – Run in browsers (e.g., Gmail, Facebook, Google Docs).
* **Mobile Applications** – Designed for smartphones/tablets (e.g., WhatsApp, Instagram).
* **Hybrid Applications** – Work on both web and mobile with one codebase (e.g., PWA apps like Twitter Lite).

**2. Based on Deployment**

* **Cloud Applications** – Hosted on cloud, accessible anywhere (e.g., Dropbox, Google Drive).
* **On-Premise Applications** – Installed and run locally within an organization (e.g., ERP systems in companies).

**3. Based on Usage/Purpose**

* **Business Applications** – ERP, CRM, HRMS, Accounting software.
* **Educational Applications** – e-learning platforms, digital classrooms (Coursera, Duolingo).
* **Entertainment Applications** – Games, video streaming (Netflix, Spotify).
* **Productivity Applications** – Tools like MS Office, Trello, Notion.
* **Social Applications** – Social networking apps (Facebook, LinkedIn, Twitter).
* **Utility Applications** – Antivirus, File managers, Calculators.

**4. Based on Development Type**

* **Native Apps** – Built specifically for one platform (Android/iOS).
* **Cross-Platform Apps** – Built to run on multiple platforms with one codebase (Flutter, React Native).
* **Progressive Web Apps (PWA)** – Web apps that behave like mobile apps (e.g., Twitter Lite).

✅ In summary:  
Applications can be **Desktop, Web, Mobile, Cloud, Business, Educational, Entertainment, Utility, Native, Cross-platform, PWA**, etc.

Question 2)What is programming?

**Programming (also called computer programming) is the process of designing and writing instructions that a computer can understand and execute to perform a specific task.**

Those instructions are written in a **programming language** (like Python, C, Java, JavaScript, etc.).

**🔑 Key Points:**

1. **Definition**:  
   Programming is the act of creating a set of commands (called *code*) that tells a computer *what to do*.
2. **Purpose**:  
   To solve problems, automate tasks, build applications, websites, games, and more.
3. **How it Works**:
   * Programmer writes code using a **programming language**.
   * The code is then **compiled or interpreted** into machine language (binary: 0s and 1s).
   * The computer executes these instructions step by step.

Question 3)What is python?

Python is a **high-level, interpreted, general-purpose programming language** that is widely used for building all kinds of applications — from web apps to AI.

**🔑 Key Features of Python:**

1. **Easy to Learn & Readable** – Python uses simple, English-like syntax.
2. **Interpreted Language** – Code runs line by line (no need to compile).
3. **Cross-platform** – Works on Windows, macOS, Linux, etc.
4. **Versatile** – Used for web development, data science, AI/ML, automation, scripting, game dev, and more.
5. **Huge Community & Libraries** – Rich ecosystem (NumPy, Pandas, TensorFlow, Django, Flask, etc.).

Question 17) What are negative indexes and why are they used?

**Negative Index in Python**

In Python, lists, strings, and other sequence types support **negative indexing**.

* Normally, **positive indexes** start from 0 (first element).
* **Negative indexes** start from -1 (last element), moving backwards.

Question 25) What is List? How will you reverse a list?

**What is a List in Python?**

* A **List** is a **collection of items** in Python.
* It can store **different data types** (integers, strings, floats, even other lists).
* Lists are **ordered, mutable (can be changed), and allow duplicates**.
* Lists are written inside **square brackets [ ]**.

**How to Reverse a List?**

Python provides multiple ways to reverse a list:

**1. Using reverse() method (in-place)**

numbers = [1, 2, 3, 4, 5]

numbers.reverse()

print(numbers) # [5, 4, 3, 2, 1]

28) Differentiate between append () and extend () methods?

a = [1, 2, 3]

a.append([4, 5])

print(a) # [1, 2, 3, [4, 5]]

a = [1, 2, 3]

a.extend([4, 5])

print(a) # [1, 2, 3, 4, 5]

Question 43) What is tuple? Difference between list and tuple.

**What is a Tuple in Python?**

* A **tuple** is a collection in Python, **similar to a list**, but **immutable** (cannot be changed once created).
* Tuples are written inside **parentheses ( )** instead of square brackets [ ].
* They can hold **different data types** and **duplicate values**.
* **Difference Between List and Tuple**

| **Feature** | **List** | **Tuple** |
| --- | --- | --- |
| **Syntax** | [] square brackets | () parentheses |
| **Mutability** | ✅ Mutable (can be changed) | ❌ Immutable (cannot be changed) |
| **Methods** | Has many (append, extend, remove) | Very few (count, index) |
| **Performance** | Slower (because mutable) | Faster (because immutable) |
| **Use Case** | When data needs to change | When data should **not** change |
| **Memory Usage** | More memory | Less memory |

# Question 83)**What is an Error in Python?What is exceptionn in python**

In Python, **errors** are problems in a program that make it stop running (crash).  
There are two main types:

**Syntax Errors**

1.Occur when you write code that Python cannot understand.

Example:print("Hello" # missing closing parenthesis

✅ Output:

SyntaxError: unexpected EOF while parsing

**Exceptions (Runtime Errors)**

1.Occur while the program is running, even though the syntax is correct.

Example:x = 10 / 0

✅ Output:

ZeroDivisionError: division by zero

Some common **built-in exception classes** in Python:

Exception → Base class for all exceptions

ValueError → Raised when a function gets an argument of the right type but inappropriate value

TypeError → Raised when an operation is applied to an object of inappropriate type

ZeroDivisionError → Raised when dividing by zero

IndexError → Raised when accessing an index out of range

KeyError → Raised when a dictionary key is not found

FileNotFoundError → Raised when a file operation fails due to missing file

IOError → Raised when input/output operation fails

AttributeError → Raised when an invalid attribute is accessed

RuntimeError → Raised for runtime errors not covered by other categories

Question 84)How many except statements can a try-except block have? Name Some built-in exception classes:

### Number of except statements in a try-except block

A try block can have **zero or more** except **blocks**.

Practically, you can put **as many** except **clauses as you need**, each handling a different type of exception.

Question 85)When will the else part of try-except-else be executed?

In Python, the else **block** of a try-except-else statement is executed **only if no exception occurs in the** try **block**.

### ✅ Rules:

**If an exception occurs in** try**:**

The matching except block runs.

The else block is skipped.

**If no exception occurs in** try**:**

The else block runs.

**Finally (if present):**

The finally block always runs (whether exception occurs or not).

So the else **part is executed only when the** try **block runs successfully without any exception**.

Questiojn 86))Can one block of except statements handle multiple exception?

**Yes, one** except **block can handle multiple exceptions using a tuple.**

**Question 87)**)When is the finally block executed?

### Rules for finally:

Runs **if no exception occurs**.

Runs **if an exception occurs and is handled**.

Runs **if an exception occurs and is not handled** (before the program terminates).

Runs even if there’s a return**,** break**, or** continue in the try or except block.

The only time finally might not run is if the program is **forcefully killed** (e.g., os.\_exit(), system crash, power failure).

Question 89)How Do You Handle Exceptions with Try/Except/Finally in Python? Explain with coding snippets.

try:

# Code that might raise an exception

except SomeException:

# Code to handle the exception

finally:

# Code that runs no matter what

def safe\_divide(a, b):

try:

result = a / b

except ZeroDivisionError:

print("Error: Division by zero is not allowed.")

result = None

finally:

print("Execution completed.")

return result

print(safe\_divide(10, 2)) # Works fine

print(safe\_divide(10, 0)) # Triggers exception