What is python?

Python is a high-level, easy-to-learn programming language used for building websites, apps, games, data analysis, and more. It uses simple and readable syntax, which makes it great for beginners. Python supports many libraries and tools, making it powerful for various tasks like AI, automation, and web development. It works on different platforms like Windows, macOS, and Linux.

Is python a interpreter language? If yes explain?

Yes, Python is an interpreted language. This means that Python code is executed line by line by an interpreter, instead of being compiled into machine code all at once. When you run a Python program, the interpreter reads each line, checks for errors, and then executes it. This makes debugging easier and allows quick testing of code.

What is data? what are the different data types in python?

What is Data?
Data is any information that can be collected, stored, and processed. It can be numbers, text, images, or anything that a computer can use. In programming, data is used to perform operations and solve problems.

Different Data Types in Python:
Python has several built-in data types to store different kinds of data:

1. **int** – for whole numbers (e.g., 5, -2)
2. **float** – for decimal numbers (e.g., 3.14, -0.5)
3. **str** – for text or strings (e.g., "hello")
4. **bool** – for boolean values (True or False)
5. **list** – for ordered collections (e.g., [1, 2, 3])
6. **tuple** – for fixed ordered collections (e.g., (1, 2))
7. **dict** – for key-value pairs (e.g., {"name": "John"})
8. **set** – for unordered unique items (e.g., {1, 2, 3})

What is list? Give an example of any list?

A **list** in Python is a collection of items that are ordered and changeable (mutable). It can hold different types of data like numbers, strings, or even other lists. Lists are written using square brackets [], and items are separated by commas.

Example of a list:

fruits = ["apple", "banana", "mango", "orange"]

In this example, fruits is a list that contains four string items. You can add, remove, or change items in a list.

What is dictionary? Give an example of any dictionary?

A **dictionary** in Python is a collection of key-value pairs. Each item has a key and a value, and you can use the key to access its value. Dictionaries are written using curly braces {} and the key and value are separated by a colon :.

Example of a dictionary:

student = {"name": "Alice", "age": 20, "grade": "A"}

In this example, student is a dictionary with keys like "name", "age", and "grade", and their corresponding values. You can add, update, or remove items in a dictionary.

What is tuple? Give an example for that?

A **tuple** in Python is a collection of items that are ordered but **cannot be changed** (immutable). Tuples are similar to lists, but once created, you cannot add, remove, or change the items inside a tuple. They are written using **round brackets** ().

Example of a tuple:

colors = ("red", "green", "blue")

In this example, colors is a tuple containing three string items. Tuples are often used when you want to store data that should not be modified.

What is the difference between mutable and immutable? Give a single example for each of them to demonstrate?

Mutable means the object can be changed after it is created.
Immutable means the object cannot be changed once it is created.

Example of mutable (List):

fruits = ["apple", "banana"]

fruits.append("mango") # List is changed

print(fruits) # Output: ['apple', 'banana', 'mango']

Example of immutable (Tuple):

colors = ("red", "green")

colors[0] = "blue" # This will give an error because tuples can't be changed

So, **lists are mutable**, and **tuples are immutable**

.

What is the difference between tuple and list?

The main difference between **tuple** and **list** in Python is:

1. **Mutability**:
    - **List** is **mutable** – you can change, add, or remove items.
    - **Tuple** is **immutable** – you cannot change, add, or remove items once created.
2. **Syntax**:
    - **List** uses square brackets []
      Example: fruits = ["apple", "banana"]
    - **Tuple** uses round brackets ()
      Example: colors = ("red", "green")
3. **Performance**:
    - Tuples are faster and use less memory than lists.
4. **Use Case**:
    - Use **lists** when you need to change data.
    - Use **tuples** when data should stay constant.
    -

How can we mutate the list? Explain with example?

We can **mutate** a list in Python by **changing**, **adding**, or **removing** its elements. Since lists are **mutable**, their contents can be updated after creation.

Ways to mutate a list:

1. **Changing an item:**

```
fruits = ["apple", "banana", "cherry"]

fruits[1] = "mango"  # Changes "banana" to "mango"

print(fruits)  # Output: ['apple', 'mango', 'cherry']
```

2. **Adding an item:**

```
fruits.append("orange")  # Adds "orange" to the end

print(fruits)  # Output: ['apple', 'mango', 'cherry', 'orange']
```

3. **Removing an item:**

```
fruits.remove("mango")  # Removes "mango"

print(fruits)  # Output: ['apple', 'cherry', 'orange']
```

These operations show how you can change the content of a list after it's created.

What is the difference between append and insert?

The difference between **append()** and **insert()** in Python is how and where they add elements to a list:

append()

- Adds an item **at the end** of the list.
- Syntax: list.append(item)

Example:

```
fruits = ["apple", "banana"]

fruits.append("mango")

print(fruits)  # Output: ['apple', 'banana', 'mango']
```

insert()

- Adds an item **at a specific position** in the list.
- Syntax: list.insert(index, item)

Example:

```
fruits = ["apple", "banana"]

fruits.insert(1, "mango")  # Inserts "mango" at index 1

print(fruits)  # Output: ['apple', 'mango', 'banana']
```

So, use append() to add at the end, and insert() to add at a specific index.

What is the difference between pop and pop index?

In Python, **pop()** and **pop(index)** are both used to **remove and return** items from a list, but they work differently:

pop()

- Removes and returns the **last item** in the list by default.
- Syntax: list.pop()

Example:

```
fruits = ["apple", "banana", "mango"]
fruits.pop()  # Removes "mango"
print(fruits)  # Output: ['apple', 'banana']
```

pop(index)

- Removes and returns the item at the **given index**.
- Syntax: list.pop(index)

Example:

```
fruits = ["apple", "banana", "mango"]
fruits.pop(1)  # Removes "banana" (index 1)
print(fruits)  # Output: ['apple', 'mango']
```

Summary:

- pop() = removes last item
- pop(index) = removes item at specific index
-

How can you mutate the dictionary in python? Give an example?

You can **mutate a dictionary** in Python by **adding**, **changing**, or **removing** key-value pairs. Dictionaries are **mutable**, so their contents can be updated after creation.

Example:

```
student = {"name": "Alice", "age": 20}

# 1. Changing a value
student["age"] = 21

# 2. Adding a new key-value pair
student["grade"] = "A"

# 3. Removing a key-value pair
del student["name"]

print(student)  # Output: {'age': 21, 'grade': 'A'}
```

This shows how you can easily update the contents of a dictionary using keys.

Write nested dictionary for electronic products?

Here's a simple **nested dictionary** for electronic products:

```
electronics = {
   "laptop": {
      "brand": "Dell",
      "price": 50000,
      "features": ["i5 processor", "8GB RAM", "512GB SSD"]
   },
   "mobile": {
      "brand": "Samsung",
      "price": 20000,
      "features": ["5G", "128GB Storage", "Dual Camera"]
   },
   "tv": {
      "brand": "Sony",
      "price": 40000,
      "features": ["Smart TV", "4K", "43 inch"]
   }
}
```

```
print(electronics["mobile"]["features"])  # Output: ['5G', '128GB Storage', 'Dual Camera']
```

In this example, each product (like laptop, mobile, tv) has its own dictionary with details like brand, price, and features.

What is operator?

An **operator** in Python is a symbol that performs a specific operation on values or variables. Operators are used to do tasks like addition, comparison, or logic checking. For example, + is used to add numbers, and == is used to compare two values. Python has different types of

operators such as **arithmetic**, **comparison**, **logical**, **assignment**, and **bitwise** operators. They help in building expressions and performing operations in programs.

What are different types of operators in python?

Python has several types of operators that are used to perform different kinds of operations:

1. **Arithmetic Operators** – Used for basic math operations: +, -, *, /, %, **, //
   Example: 5 + 2 = 7
2. **Comparison Operators** – Compare two values: ==, !=, >, <, >=, <=
   Example: 5 > 3 returns True
3. **Assignment Operators** – Assign values to variables: =, +=, -=, *=, etc.
   Example: x += 2 means x = x + 2
4. **Logical Operators** – Used for logical conditions: and, or, not
   Example: (x > 5 and x < 10)
5. **Bitwise Operators** – Work on binary numbers: &, |, ^, ~, <<, >>
   Example: 5 & 3 gives 1
6. **Membership Operators** – Check if a value is in a sequence: in, not in
   Example: "a" in "apple" returns True
7. **Identity Operators** – Check if two variables point to the same object: is, is not
   Example: x is y returns True if x and y are the same object.
8.

What are arithmetical operators? Why we use them?

Arithmetical operators in Python are used to perform basic mathematical operations on numbers. These operators help us do calculations like addition, subtraction, multiplication, etc.

Common arithmetical operators:

- + : Addition
- - : Subtraction
- * : Multiplication
- / : Division
- // : Floor Division (removes decimal part)
- % : Modulus (gives remainder)
- ** : Exponentiation (power)

Why we use them?

We use arithmetical operators to perform calculations in programs, such as finding totals, averages, or solving math-related problems. They are very useful in building logic for finance, games, data analysis, and more.

What are compare operators? Why we use them?

Comparison operators in Python are used to **compare two values**. They return **True** or **False** based on the result of the comparison.

Common comparison operators:

- == : Equal to
- != : Not equal to
- > : Greater than
- < : Less than
- >= : Greater than or equal to
- <= : Less than or equal to

Why we use them?

We use comparison operators to make decisions in programs. They are often used in **conditions**, such as if statements, to check values and control the flow of the program based on whether something is true or false.