1. What is Python?

Python is a high-level, interpreted programming language known for its simplicity and readability. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely used in web development, data science, artificial intelligence, and automation due to its vast ecosystem of libraries.

---

2. Is Python an interpreted language? If yes, explain why.

Yes, Python is an interpreted language. This means that Python code is not compiled to machine language before execution. Instead, the Python interpreter reads the code line by line, converts it into intermediate bytecode, and executes it. This makes debugging and testing easier and more interactive.

---

3. What is the difference between interpreter and compiler?

Interpreter	Compiler
Translates code line by line	Translates entire code at once
Slower execution	Faster execution after compilation
Good for debugging	Errors shown after full compilation
No separate output file is created	Creates a separate executable file

---

4. What is data? What are the data types in Python?

Data refers to information stored or processed by a computer. In Python, data types define the kind of value a variable holds. Common types include:

- `int` (integers)
- `float` (decimals)
- `str` (strings)
- `bool` (True/False)
- `list`, `tuple`, `dict`, `set` (collections)

---

5. What is a list? Give an example.

A list is an ordered, mutable collection of items. It can contain elements of any data type.

```
my_list = [1, "apple", 3.5, True]
```

You can change items, add, or remove elements.

---

6. What is a dictionary? Give an example.

A dictionary is an unordered, mutable collection of key-value pairs. Keys must be unique.

```
my_dict = {"name": "Ram", "age": 25}
```

You access values using keys.

---

7. What is a tuple? Give an example.

A tuple is an ordered, immutable collection. Once created, elements cannot be changed.

```
my_tuple = (1, "apple", 3.5)
```

---

8. What is the difference between mutable & immutable variables?

Mutable	**Immutable**
Can be changed after creation	Cannot be changed after creation
Examples: list, dict, set	Examples: int, float, str, tuple
Uses less memory if reused	New object is created for every change
Useful when you need editable data	Useful for hashable or constant data

---

9. What is the difference between tuple and list?

Tuple	List
Immutable	Mutable
Uses less memory	Uses more memory
Faster than lists	Slower compared to tuples
Cannot change elements	Can add, remove, change elements

---

10. How can we mutate the list?

You can mutate a list using:

- `append()` – adds item at end
- `insert()` – inserts at a specific index
- `remove()` – removes an item
- `pop()` – removes by index or last item
- Indexing – to directly assign new values

---

11. What is the difference between append() and insert()?

append()	insert()
Adds item at the end	Adds item at a specific position
Syntax: list.append(x)	Syntax: list.insert(i, x)
Takes one argument	Takes index and item
More efficient and faster	May shift elements, slower

---

12. What is the difference between pop() and pop(index)?

pop()	**pop(index)**
Removes last element	Removes element at given index
No argument needed	Requires index argument
Acts like stack pop	Allows specific item removal
Raises IndexError if list is empty	Raises IndexError if index invalid

---

13. How can you mutate a dictionary in Python?

You can:

- Add new key-value pair: `d['new'] = value`
- Update existing: `d['key'] = new_value`
- Delete: `del d['key']`
- Example:

```
d = {'name': 'Ram'}
d['age'] = 25
d['name'] = 'Ravi'
del d['age']
```

---

14. Write nested dictionaries for electronics product.

```
electronics = {
  'laptop': {'brand': 'HP', 'price': 50000},
  'phone': {'brand': 'Samsung', 'price': 20000}
}
```

---

15. Write a list of dictionaries.

```
products = [
  {'name': 'Phone', 'price': 15000},
  {'name': 'Tablet', 'price': 25000}
]
```

16. What are logical operators?

Logical operators combine multiple conditions:

- `and`: True if both are True
- `or`: True if at least one is True
- `not`: Inverts the condition

17. Difference between logical and & logical or

and	**or**
Returns True if both are True	Returns True if any is True
Short-circuits on first False	Short-circuits on first True
Used for strict conditions	Used for optional conditions
Example: a > 0 and b > 0	Example: a > 0 or b > 0

18. What are membership operators?

They check presence of elements in a sequence:

- `in` – True if item exists
- `not in` – True if item does not exist

19. Difference between `in` and `not in`

in	not in
Returns True if value found	Returns True if value not found
Used in loops and conditions	Used for negative filtering
Checks membership	Checks absence
Example: 'a' in 'apple'	Example: 'x' not in [1, 2, 3]

---

20. Difference between == and !=

== (equal to)	!= (not equal to)
True if values are the same	True if values are different
Used to compare equality	Used to check inequality
Example: x == 5	Example: x != 10
Used in conditionals and logic	Used in loops, filters, decisions

---

21. What are conditional statements in Python?

They control program flow based on conditions:

```
if condition:
    # do something
elif other_condition:
    # do something else
else:
    # fallback
```

---

22. Write a program using if-else.

```
age = 18
if age >= 18:
    print("Adult")
else:
    print("Minor")
```

---

23. Write if-elif-else ladder.

```
score = 75
if score > 90:
    print("A")
elif score > 70:
    print("B")
else:
    print("C")
```

---

24. Program showing nested conditions.

```
x = 10
if x > 5:
    if x < 20:
        print("x is between 5 and 20")
    else:
        print("x is more than 20")
else:
    print("x is 5 or less")
```

---

25. What is indentation? Why is it important?

Indentation defines code blocks in Python. It replaces curly braces and makes code readable and structured. Improper indentation leads to errors.

```
if True:
    print("Indented correctly")
```

---

26. What is an error? Types in Python?

An error is a problem in code that stops execution. Types:

- Syntax Error
- Runtime Error (e.g. ZeroDivisionError)
- Logical Error (wrong output)

---

27. Examples of system error, name error, key error

```
# NameError
print(x)  # x not defined

# KeyError
d = {'a': 1}
print(d['b'])  # 'b' not found

# SystemError (usually internal; rare in beginner code)
```

---

28. What is loop? Types in Python?

Loops repeat code:

- `for` loop – iterate over sequence
- `while` loop – based on condition

---

29. For loop using list

```
nums = [1, 2, 3]
for num in nums:
    print(num)
```

---

30. For loop using str, dict, tuple

```
# String
for ch in "abc":
    print(ch)

# Dictionary
d = {'a': 1, 'b': 2}
for k in d:
    print(k, d[k])

# Tuple
t = (10, 20)
for i in t:
    print(i)
```