1.**Boilerplate** – Standardized code that is commonly included in multiple places with minimal modification.

2.**Dynamic Typing** – Python allows variable types to be determined at runtime instead of being explicitly declared.

3.**Variables** – Named storage locations that hold data and can change during program execution.

4.**Constants** – Fixed values that do not change throughout the execution of a program (by convention, written in uppercase).

5.**Identifiers** – The names used to identify variables, functions, classes, and other objects.

6.**Immutable** – Data types that cannot be modified after creation, such as tuples and strings.

7.**Mutable** – Data types that can be changed after creation, such as lists and dictionaries.

8.**Indentation** – Python uses indentation to define code blocks instead of curly braces {}.

9. **Interpreter** – The Python program that reads and executes Python code line by line.

10.**Script –** A Python file (.py) containing executable code.

11.**Module –** A file containing Python code that can be imported and reused.

12.**Package** – A collection of modules organized in directories containing an __init__.py file.

13.**Library –** A collection of modules and functions that extend Python’s capabilities.

14.**PIP (Package Installer for Python)** – The standard tool for installing Python packages from the Python Package Index (PyPI).

15.**List Comprehension –** A concise way to create lists using a single line of code.

16.**Tuple –** An immutable sequence type.

17.**Dictionary –** A key-value data structure (dict).

18.**Set –** An unordered collection of unique elements.

19.**Loop –** A construct used for iteration (for and while loops).

20. **Function –** A reusable block of code defined using def.

21.**Lambda Function –** A small anonymous function defined using lambda.

22.**Class –** A blueprint for creating objects.

23.**Object –** An instance of a class.

24.**Method –** A function defined inside a class.

25.**Inheritance –** A mechanism allowing one class to derive from another.

26.**Encapsulation –** The practice of keeping data and methods safe from outside interference.

27.**Polymorphism –** The ability to use the same interface for different data types.

28.**Decorator –** A special function that modifies the behavior of another function or class.

29.**Generator –** A function that yields values lazily using yield instead of return.

30.**Iterator –** An object that can be traversed using next().

31.**Exception Handling –** The use of try, except, and finally to manage runtime errors.

32.**Docstring –** A string used to document a function, class, or module.

33.**Global Scope –** Variables that are accessible throughout the script.

34.**Local Scope –** Variables that are accessible only within a function.

35.**Virtual Environment** – An isolated Python environment for managing dependencies.

36.**Garbage Collection** – Python's automatic memory management to reclaim unused memory.

37.**Threading** – Running multiple tasks concurrently using threads.

38.**Asynchronous Programming –** Using async and await to handle tasks that run independently.

39.**Type Hinting** – The practice of annotating function arguments and return types (def add(x: int, y: int) -> int).

40.**List Slicing** – Extracting portions of lists using indices (list[start:end:step]).

41.**Regex (Regular Expressions) –** A sequence of characters defining a search pattern.

42.**PEP (Python Enhancement Proposal) –** A design document providing guidelines and features for Python.

43.**Zen of Python –** A set of guiding principles for Python programming (import this).

44.**Duck Typing –** Python's principle of treating objects based on their behavior rather than their class.

45.**Comprehensions –** Syntax constructs like list, dictionary, and set comprehensions.

46.**F-Strings –** A modern way of formatting strings (f"Hello, {name}").

47.**Metaclass –** A class of a class that defines how classes behave.

48.**Magic Methods (Dunder Methods) –** Special methods prefixed and suffixed by double underscores (__init__, __str__, etc.).

49.**Hashable –** Objects that have a fixed hash value and can be used as dictionary keys.

50.**Shallow Copy vs Deep Copy –** Cloning objects where a shallow copy copies references, and a deep copy creates a new independent object.