100 PYTHON INTERVIEW QUESTIONS

1. **Python Overview**: Python is a high-level, interpreted language known for simplicity and readability. It emphasizes clean code and supports multiple paradigms like procedural and object-oriented programming.

2. **Key Features**: Python's features include readability, dynamic typing, extensive library support, automatic memory management, and versatility across domains like web development and data analysis.

3. **Distinctive Aspects**: Python's simplicity, readability, extensive community, and diverse libraries set it apart from other languages. Its usage spans web development, data science, and automation.

4. **PEP 8 Definition**: PEP 8 serves as Python's style guide, promoting code readability and consistency. It offers guidelines for formatting, naming conventions, and code organization.

5. **Python Modules**: Modules contain reusable code like functions and variables. They enhance project organization and facilitate code reuse through import statements.

6. **Python Packages**: Packages organize related modules into directories, aiding project management. They offer a structured approach to code organization and distribution.

7. **Commenting in Python**: Python comments start with '#' and aid code clarity. They're ignored during execution, serving as inline documentation.

8. **Data Types**: Python supports diverse data types, including integers, floats, strings, lists, tuples, dictionaries, and booleans, each serving specific purposes in programming.

9. **Type Conversion**: Type conversion transforms data from one type to another using functions like int(), float(), and str().

10. **String Interpolation**: String interpolation embeds expressions or variables within strings for dynamic content. It's done via f-strings or the format() method.

11. **Conditional Statements**: if, elif, and else control program flow based on conditions, enabling decision-making.

12. **Loops in Python**: Loops like for and while repeat code execution. They iterate over sequences or execute until conditions are met.

13. **Range vs. Xrange (Python 2)**: range() generates a list, while xrange() returns an iterator, conserving memory for large ranges.

14. **Functions in Python**: Functions are reusable code blocks, aiding modularity and organization.

15. **Function vs. Method**: Functions stand alone; methods belong to classes or objects, accessing their data.

16. **Function Definition**: Functions are defined with 'def', specifying name, parameters, and body.

17. **__init__ Method**: Initializes class instances, setting up attributes.

18. **Object-Oriented Programming (OOP)**: Organizes code around objects, emphasizing encapsulation, inheritance, and polymorphism.

19. **Classes and Objects**: Classes define object blueprints; objects are instances of classes.

20. **Object Creation**: Objects are created by calling class constructors.

21. **Inheritance**: Subclasses inherit properties and methods from superclasses, promoting code reuse.

22. **Method Overriding**: Subclasses redefine superclass methods.

23. **Method Overloading**: Achieved via default arguments or variable-length arguments.

24. **Encapsulation**: Bundles data and methods within classes, controlling access.

25. **Polymorphism**: Objects exhibit multiple forms or behaviors, supporting method overriding.

26. **Generators**: Functions producing iterable sequences, conserving memory.

27. **Decorators**: Modify function or class behavior, enhancing functionality.

28. **Lambda Functions**: Anonymous functions for concise one-liners.

29. **Modules in Python**: Files containing code for reuse via imports.

30. **Importing Modules**: Done using 'import' keyword.

31. **Virtual Environments**: Isolates project dependencies, managing environments.

32. **Exceptions**: Handle runtime errors gracefully using try-except blocks.

33. **Error Handling**: Gracefully manage errors using try-except blocks.

34. **try-except-else-finally**: Structure for exception handling.

35. **Built-in Data Structures**: Lists, tuples, dictionaries, sets, strings offer diverse data storage.

36. **Lists**: Ordered, mutable collections.

37. **Tuples**: Immutable ordered collections.

38. **Dictionaries**: Unordered key-value pairs.

39. **Sets**: Unordered unique elements.

40. **Strings**: Immutable sequences of characters.

41. **String Concatenation**: Done with '+' or '.join()' methods.

42. **String Formatting**: Utilizes '%', str.format(), or f-strings.

43. **File Handling**: Operations for reading and writing files.

44. **Opening and Closing Files**: Use open() and close() methods.

45. **File Modes**: Specify read, write, append, or exclusive creation modes.

46. **Exception Handling in File Operations**: Gracefully manage file-related errors.

47. **Context Managers**: Facilitate resource management, like automatic file closure.

48. **Generator Functions**: Produce iterators for on-demand value generation.

49. **List Comprehensions**: Concise list creation from existing iterables.

50. **The pass Statement**: Placeholder for no-operation situations.

51. **self Parameter**: References current class instance.

52. **Shallow vs. Deep Copy**: Differences in copying data structures.

53. **Advantages of Python in Web Development**: Versatility, frameworks, community support.

54. **Global Interpreter Lock (GIL)**: Restricts concurrent Python thread execution.

55. **Metaclasses**: Define class behaviors and structures.

56. **File I/O Error Handling**: Gracefully manage file-related errors.

57. **Purpose of __name__ Variable**: Indicates module execution context.

58. **Shallow vs. Deep Comparison**: Contrasts in comparing objects.

59. **Advantages of Virtual Environments**: Dependency isolation, version management.

60. **Purpose of __main__ Block**: Defines script entry point.

61. **Purpose of __str__ Method**: Provides human-readable object representation.

62. **Purpose of __repr__ Method**: Offers unambiguous object representation.

63. **Difference Between __str__ and __repr__**: Distinctions in object string representation.

64. **Purpose of super() Function**: Calls superclass methods.

65. **Purpose of __getitem__ Method**: Enables custom indexing/slicing behavior.

66. **Purpose of __setitem__ Method**: Facilitates item assignment customization.

67. **Purpose of __len__ Method**: Returns object length.

68. **Purpose of __iter__ Method**: Makes objects iterable.

69. **Purpose of __next__ Method**: Provides next iterator item.

70. **Purpose of @property Decorator**: Defines getter method for attributes.

71. **Purpose of @staticmethod Decorator**: Defines static methods in classes.

72. **Purpose of @classmethod Decorator**: Defines class methods.

73. **Purpose of __call__ Method**: Enables object invocation.

74. **Purpose of *args and **kwargs**: Handle variable arguments.

75. **Decorators in Python**: Modify function/class behavior.

76. **Purpose of @classmethod Decorator**: Defines class methods.

77. **Lambda Functions in Python**: Anonymous function shorthand.

78. **Modules in Python**: Encapsulate reusable code.

79. **Packages in Python**: Organize modules hierarchically.

80. **Purpose of __init__.py File**: Marks directory as Python package.

81. **Purpose of sys Module**: Provides system-specific functions.

82. **Purpose of os Module**: Offers OS interaction capabilities.

83. **Purpose of datetime Module**: Manipulates dates and times.

84. **Decorators in Python**: Enhance function/class behavior.

85. **Purpose of @property Decorator**: Defines attribute getter method.

86

. **Purpose of @staticmethod Decorator**: Defines static methods.

87. **Purpose of @classmethod Decorator**: Defines class methods.

88. **Lambda Functions in Python**: Concise anonymous functions.

89. **Modules in Python**: Encapsulate reusable code.

90. **Packages in Python**: Hierarchical module organization.

91. **Purpose of __init__.py File**: Indicates Python package directory.

92. **Purpose of sys Module**: Provides system-specific functionality.

93. **Purpose of os Module**: Facilitates OS interaction.

94. **Purpose of datetime Module**: Manipulates date and time.

95. **Purpose of random Module**: Generates random numbers.

96. **Purpose of json Module**: Handles JSON data.

97. **Purpose of pickle Module**: Serializes/deserializes Python objects.

98. **Generators in Python**: Efficient iterable value producers.

99. **Purpose of yield Keyword**: Pauses generator functions, yielding values.

100. **Purpose of zip() Function**: Combines iterables into tuples.