Great! Here's the **full set of 75 MCQs (Easy, Medium, Hard)** for **Chapter 1: Whetting Your Appetite** from the Python Tutorial (Release 3.10.4).

📘 Chapter 1: Whetting Your Appetite — 75 MCQs

🟢 Easy Level (Q1–Q25)

1. Python is a __________ programming language.
   a) low-level
   b) middle-level
   c) high-level
   d) hardware-level
   ✅ c) high-level

2. Python was named after:
   a) a type of snake
   b) a Roman emperor
   c) a TV show
   d) a video game
   ✅ c) a TV show

3. Which of the following is **NOT** a feature of Python?
   a) Object-oriented
   b) Interpreted
   c) Platform-dependent
   d) Easy-to-learn
   ✅ c) Platform-dependent

4. Python is best suited for:
   a) Web browsing
   b) Image scanning
   c) Automation and scripting
   d) Mechanical operations
   ✅ c) Automation and scripting

5. Which symbol is used for comments in Python?
   a) //
   b)
   c) #
   d) %
   ✅ c) #

6. Python’s syntax is mainly influenced by:
   a) C
   b) Pascal
   c) English
   d) Assembly
   ✅ c) English

7. Which of the following is not a valid use of Python?
   a) GUI applications
   b) Compiling Java code
   c) Writing scripts
   d) Data processing
   ✅ b) Compiling Java code

8. Python is an interpreted language, which means:
   a) It compiles first
   b) It translates line by line
   c) It runs only on Linux
   d) It generates binary executables
   ✅ b) It translates line by line

9. One advantage of using Python over C++ is:
   a) Lower-level memory access
   b) Direct hardware control
   c) Rapid development
   d) Slower execution
   ✅ c) Rapid development

10. Python programs are typically:
    a) Longer than C programs
    b) Shorter than C programs
    c) Same length as C programs
    d) Not readable
    ✅ b) Shorter than C programs

11. Python was created by:
    a) Dennis Ritchie
    b) James Gosling
    c) Guido van Rossum
    d) Bjarne Stroustrup
    ✅ c) Guido van Rossum

12. Python code blocks are identified by:
    a) Brackets
    b) Indentation
    c) Semicolons
    d) None of the above

✅ b) Indentation

13. What does Python use instead of curly braces {} for blocks?
    a) Tabs
    b) Comments
    c) Indentation
    d) Markers
    ✅ c) Indentation

14. Which is NOT an advantage of Python?
    a) Large standard library
    b) Compiled execution
    c) Interactive shell
    d) Extensible
    ✅ b) Compiled execution

15. Python code can be embedded in applications written in:
    a) HTML
    b) Assembly
    c) C
    d) CSS
    ✅ c) C

16. Which is a correct extension for Python files?
    a) .pyt
    b) .pt
    c) .py
    d) .python
    ✅ c) .py

17. Python supports which of the following paradigms?
    a) Procedural
    b) Object-oriented
    c) Functional
    d) All of the above
    ✅ d) All of the above

18. What kind of language is Python in terms of type-checking?
    a) Dynamically typed
    b) Statically typed
    c) Untyped
    d) Strongly compiled
    ✅ a) Dynamically typed

19. Python allows the use of modules. What is a module?
    a) A folder
    b) A compressed file

c) A collection of Python definitions
d) A hardware component
✅ c) A collection of Python definitions

20. Python helps avoid which of the following due to indentation?
a) Errors
b) Cluttered code
c) Bugs
d) Semicolons
✅ b) Cluttered code

21. Python is suitable for:
a) AI and ML
b) Robotics
c) Web scraping
d) All of the above
✅ d) All of the above

22. What makes Python “interpreted”?
a) Runs inside Chrome
b) Needs Java runtime
c) Executes code line by line
d) Converts code to HTML
✅ c) Executes code line by line

23. Which of these is a Python advantage?
a) Slow development
b) Fast runtime
c) Large ecosystem
d) Complex syntax
✅ c) Large ecosystem

24. Python syntax emphasizes:
a) Speed
b) Obscurity
c) Readability
d) Compactness only
✅ c) Readability

25. Python standard library is often referred to as:
a) Extra tools
b) Add-on bundle
c) Batteries included
d) Bonus features
✅ c) Batteries included

🟡 Medium Level (Q26–Q50)

26. Python is often preferred over shell scripts because:
    a) Shell scripts are more readable
    b) Python runs only on Linux
    c) Python offers better structure
    d) Python is older
    ✅ c) Python offers better structure

27. Which of these tasks can Python NOT perform directly?
    a) Data analysis
    b) GUI programming
    c) Compiling machine code
    d) Web development
    ✅ c) Compiling machine code

28. Python is faster to develop in than C/C++ because:
    a) It skips logical testing
    b) It has automatic indentation
    c) It requires less boilerplate
    d) It executes faster
    ✅ c) It requires less boilerplate

29. What makes Python suitable for rapid application development?
    a) Binary compilation
    b) Interpreted nature and dynamic typing
    c) Hardware interfacing
    d) Use of macros
    ✅ b) Interpreted nature and dynamic typing

30. Python's standard modules provide:
    a) Only networking utilities
    b) Limited OS access
    c) Extended file and OS operations
    d) GUI-only support
    ✅ c) Extended file and OS operations

Here's the complete set of **75 MCQs for Chapter 1: Whetting Your Appetite**, divided into **Easy (1–25)**, **Medium (26–50)**, and **Hard (51–75)** — all generated below in one go:

📘 Chapter 1: Whetting Your Appetite — Full 75 MCQs

🟢 Easy Level (Q1–Q25)

1. Python is a __________ programming language.
   a) low-level
   b) middle-level
   c) **high-level** ✅
   d) hardware-level

2. Python was named after:
   a) a type of snake
   b) a Roman emperor
   c) **a TV show** ✅
   d) a video game

3. Which of the following is **NOT** a feature of Python?
   a) Object-oriented
   b) Interpreted
   c) **Platform-dependent** ✅
   d) Easy-to-learn

4. Python is best suited for:
   a) Web browsing
   b) Image scanning
   c) **Automation and scripting** ✅
   d) Mechanical operations

5. Which symbol is used for comments in Python?
   a) //
   b)
   c) **#** ✅
   d) %

6. Python's syntax is mainly influenced by:
   a) C
   b) Pascal
   c) **English** ✅
   d) Assembly

7. Which of the following is not a valid use of Python?
   a) GUI applications
   b) **Compiling Java code** ✅
   c) Writing scripts
   d) Data processing

8. Python is an interpreted language, which means:
   a) It compiles first
   b) It translates line by line ✅
   c) It runs only on Linux
   d) It generates binary executables

9. One advantage of using Python over C++ is:
   a) Lower-level memory access
   b) Direct hardware control
   c) **Rapid development** ✅
   d) Slower execution

10. Python programs are typically:
   a) Longer than C programs
   b) **Shorter than C programs** ✅
   c) Same length as C programs
   d) Not readable

11. Python was created by:
   a) Dennis Ritchie
   b) James Gosling
   c) **Guido van Rossum** ✅
   d) Bjarne Stroustrup

12. Python code blocks are identified by:
   a) Brackets
   b) **Indentation** ✅
   c) Semicolons
   d) None of the above

13. What does Python use instead of curly braces {} for blocks?
   a) Tabs
   b) Comments
   c) **Indentation** ✅
   d) Markers

14. Which is NOT an advantage of Python?
   a) Large standard library
   b) **Compiled execution** ✅
   c) Interactive shell
   d) Extensible

15. Python code can be embedded in applications written in:
   a) HTML
   b) Assembly
   c) **C** ✅
   d) CSS

16. Which is a correct extension for Python files?
   a) .pyt
   b) .pt
   c) **.py** ✅

d) .python

17. Python supports which of the following paradigms?
    a) Procedural
    b) Object-oriented
    c) Functional
    d) **All of the above** ✅

18. What kind of language is Python in terms of type-checking?
    a) **Dynamically typed** ✅
    b) Statically typed
    c) Untyped
    d) Strongly compiled

19. Python allows the use of modules. What is a module?
    a) A folder
    b) A compressed file
    c) **A collection of Python definitions** ✅
    d) A hardware component

20. Python helps avoid which of the following due to indentation?
    a) Errors
    b) **Cluttered code** ✅
    c) Bugs
    d) Semicolons

21. Python is suitable for:
    a) AI and ML
    b) Robotics
    c) Web scraping
    d) **All of the above** ✅

22. What makes Python “interpreted”?
    a) Runs inside Chrome
    b) Needs Java runtime
    c) **Executes code line by line** ✅
    d) Converts code to HTML

23. Which of these is a Python advantage?
    a) Slow development
    b) Fast runtime
    c) **Large ecosystem** ✅
    d) Complex syntax

24. Python syntax emphasizes:
    a) Speed
    b) Obscurity
    c) **Readability** ✅

d) Compactness only

25. Python standard library is often referred to as:
    a) Extra tools
    b) Add-on bundle
    c) **Batteries included** ✅
    d) Bonus features

🟡 Medium Level (Q26–Q50)

26. Python is often preferred over shell scripts because:
    a) Shell scripts are more readable
    b) Python runs only on Linux
    c) **Python offers better structure** ✅
    d) Python is older

27. Which of these tasks can Python NOT perform directly?
    a) Data analysis
    b) GUI programming
    c) **Compiling machine code** ✅
    d) Web development

28. Python is faster to develop in than C/C++ because:
    a) It skips logical testing
    b) It has automatic indentation
    c) **It requires less boilerplate** ✅
    d) It executes faster

29. What makes Python suitable for rapid application development?
    a) Binary compilation
    b) **Interpreted nature and dynamic typing** ✅
    c) Hardware interfacing
    d) Use of macros

30. Python's standard modules provide:
    a) Only networking utilities
    b) Limited OS access
    c) **Extended file and OS operations** ✅
    d) GUI-only support

31. Which of the following makes Python good for experimenting interactively?
    a) Batch processing
    b) Shell scripting
    c) **Interactive interpreter** ✅

d) Precompiled headers

32. Python is often used in education because:
    a) It is obscure
    b) It lacks features
    c) **It is readable and beginner-friendly** ✅
    d) It requires compilation

33. Python allows code reuse via:
    a) Java beans
    b) **Modules** ✅
    c) Compilers
    d) Macros

34. Which of the following languages inspired Python?
    a) HTML
    b) Perl
    c) Visual Basic
    d) **C** ✅

35. Python differs from Java in that it:
    a) Is compiled
    b) Has static typing
    c) **Has dynamic typing** ✅
    d) Runs only on Linux

36. Which of these is NOT a common Python use-case?
    a) Data Science
    b) Web Apps
    c) **Operating System Kernel** ✅
    d) Automation

37. Python's error checking is:
    a) Very minimal
    b) Non-existent
    c) **Extensive** ✅
    d) Only during compilation

38. Which Python feature helps break large programs into smaller parts?
    a) Pointers
    b) **Modules** ✅
    c) Bytecode
    d) JVM

39. Which of these is NOT a Python built-in data type?
    a) Dictionary
    b) List
    c) Tuple

d) **SetReference** ✅

40. Python's philosophy includes:
    a) **Explicit is better than implicit** ✅
    b) Confusion is good
    c) Errors should pass silently
    d) All code must compile

41. Which feature of Python allows interactive testing?
    a) Shell scripting
    b) Compiler debugging
    c) **Interpreter** ✅
    d) Assembler

42. Python’s syntax makes heavy use of:
    a) Braces
    b) Keywords
    c) **Indentation** ✅
    d) Semicolons

43. In Python, file I/O can be done using:
    a) import fs
    b) **builtin modules** ✅
    c) system.out
    d) fwrite only

44. Python can be extended with functions written in:
    a) Bash
    b) **C** ✅
    c) HTML
    d) Ruby

45. Python can be used as an extension language for:
    a) Static codebases
    b) Compiled binaries
    c) **Custom applications** ✅
    d) Data-only systems

46. What is a key benefit of using Python over C for scripting?
    a) Faster runtime
    b) Slower syntax
    c) **Higher abstraction** ✅
    d) More verbose

47. Python helps reduce development time by:
    a) Strict typing
    b) Automatic deployment
    c) **No need for compilation** ✅

d) Memory allocation

48. A real-world example of a Python application is:
    a) Linux kernel
    b) Facebook backend
    c) **Photo organizer script** ✅
    d) Windows services

49. Python’s interpreted nature makes it easy to:
    a) Optimize memory
    b) Parallel process
    c) **Experiment and test** ✅
    d) Create compilers

50. Which of these Python features simplifies code grouping?
    a) Labels
    b) Keywords
    c) **Whitespace** ✅
    d) Comments

🔴 Hard Level (Q51–Q75)

51. Why is Python considered more general-purpose than Awk or Perl?
    a) It supports fewer data types
    b) **It has general-purpose libraries** ✅
    c) It has limited syntax
    d) It lacks text processing

52. Which of the following is true about Python and GUI applications?
    a) Not supported
    b) Only on Linux
    c) Limited support
    d) **Supported via toolkits like Tk** ✅

53. Why is no variable declaration needed in Python?
    a) It is compiled
    b) It uses C-style syntax
    c) **It is dynamically typed** ✅
    d) It is interpreted

54. What does Python use to express complex operations in one line?
    a) Macros
    b) **High-level data types** ✅
    c) Low-level functions

d) Binary conversion

55. Which of the following is true about using Python modules?
   a) Modules can't be reused
   b) Modules are limited to one file
   c) **Modules help reuse and organization** ✅
   d) Modules are deprecated

56. How does Python handle code indentation errors?
   a) Silently ignores
   b) Treats as comments
   c) **Throws syntax error** ✅
   d) Executes partial code

57. Which library allows GUI development in Python?
   a) NumPy
   b) **Tkinter** ✅
   c) Pandas
   d) Flask

58. Which of the following is false about Python?
   a) It is statically typed ✅
   b) It is extensible
   c) It can embed C
   d) It supports OOP

59. Python scripts can be run directly because:
   a) They are binaries
   b) **They are interpreted** ✅
   c) They are compiled
   d) They are Java-based

60. What is a common feature between Python and Perl?
   a) Java compatibility
   b) **Dynamic typing** ✅
   c) Lack of functions
   d) Manual memory management

61. What helps Python stand out in rapid development?
   a) Manual garbage collection
   b) Platform dependence
   c) **No linking needed** ✅
   d) Explicit memory types

62. How does Python handle large programs?
   a) Compiles them in blocks
   b) Uses include files
   c) **Encourages modularization** ✅

d) Only supports short scripts

63. Why is Python called an extensible language?
    a) It has a complex parser
    b) It allows macros
    c) **It allows native code integration** ✅
    d) It uses preprocessor

64. Python allows splitting programs using:
    a) Libraries only
    b) Classes only
    c) **Modules** ✅
    d) Projects

65. In what way does Python reduce boilerplate code?
    a) By using macros
    b) By using decorators
    c) **By eliminating variable declaration** ✅
    d) By auto-including libraries

66. What is meant by 'Batteries Included' in Python?
    a) It has hardware drivers
    b) **It ships with standard libraries** ✅
    c) It auto-installs dependencies
    d) It runs without Python

67. Which of the following enables you to try Python code quickly?
    a) VirtualBox
    b) JVM
    c) **Jupyter** ✅
    d) IDLE only

68. Python is said to be platform-independent because:
    a) It's compiled per platform
    b) It doesn't require OS
    c) **Same code runs across OSes** ✅
    d) It only runs on Linux

69. Python scripts can also serve as:
    a) Batch files
    b) Compiled libraries
    c) Executable binaries
    d) **Unix shell replacements** ✅

70. What does dynamic typing mean in Python?
    a) Types change during compilation
    b) No types are used
    c) **Types are assigned at runtime** ✅

d) All variables are strings

71. Python excels in which development phase?
    a) Testing only
    b) Design
    c) **Prototype & scripting** ✅
    d) Benchmarking

72. Which of these is an interpreted language like Python?
    a) C
    b) C++
    c) **Bash** ✅
    d) Assembly

73. In which situation is Python better than Java?
    a) Speed-critical systems
    b) **AI prototyping** ✅
    c) Device drivers
    d) Low-level memory work

74. What does 'extending Python in C' mean?
    a) Using Python inside C
    b) Using C to call Python libraries
    c) **Adding C modules to Python** ✅
    d) Writing Python in C syntax

75. What is a reason Python is easier to maintain?
    a) It has no error handling
    b) It is very verbose
    c) **It has clear, readable syntax** ✅
    d) It uses semicolons

Great! The next chapter is **Chapter 2: Using the Python Interpreter**. Based on its contents from the PDF, this chapter covers:

- **2.1 Invoking the Interpreter**
    - Argument passing
    - Interactive mode
- **2.2 The Interpreter and Its Environment**
    - Source code encoding

    - Environment configuration

I will now generate **75 MCQs (25 Easy, 25 Medium, 25 Hard)** for **Chapter 2: Using the Python Interpreter** in one go, just like Chapter 1.

📘 Chapter 2: Using the Python Interpreter — 75 MCQs

🟢 Easy Level (Q1–Q25)

1. What is the default prompt of the Python interactive interpreter?
   a) `:::`
   b) `>`
   c) **`>>>`** ✅
   d) `///`

2. Python interactive mode is primarily used for:
   a) Debugging HTML
   b) **Trying out small code snippets** ✅
   c) Testing hardware
   d) Writing C extensions

3. The Python interpreter prints the first prompt only after:
   a) System reboot
   b) Keyboard input
   c) **A welcome message** ✅
   d) Code compilation

4. The shebang line in Unix-based systems usually starts with:
   a) `//`
   b) `#!` ✅
   c) `%%`
   d) `--`

5. What is the correct shebang to invoke Python 3?
   a) `#!/bin/bash`
   b) `#!/usr/bin/python`
   c) **`#!/usr/bin/env python3`** ✅

d) `#!/usr/bin/python2`

6. Which module is used to access command-line arguments in Python?
   a) os
   b) cmd
   c) **sys** ✅
   d) arg

7. In the `sys.argv` list, the first element is always:
   a) Program arguments
   b) **Script name** ✅
   c) File path
   d) Python version

8. What does `sys.argv[0]` contain when `-c` is used?
   a) Script file name
   b) Module name
   c) **"-c"** ✅
   d) Directory path

9. What encoding is assumed if none is declared in a Python file?
   a) ASCII
   b) **UTF-8** ✅
   c) ISO-8859-1
   d) cp1252

10. What keyword is used in the encoding declaration comment?
    a) `code`
    b) `filetype`
    c) **`coding`** ✅
    d) `charset`

11. What does `# -*- coding: cp1252 -*-` define?
    a) File type
    b) **Source file encoding** ✅
    c) Interpreter version
    d) Script name

12. The Python interactive mode uses which prompt for continued lines?
    a) `>>`
    b) `:::`
    c) **`...`** ✅
    d) `--`

13. Which of the following is needed to run a `.py` script from terminal?
    a) Python compiler
    b) Bash shell
    c) **Python interpreter** ✅
    d) HTML parser

14. When using Python with the `-m` flag, `sys.argv[0]` contains:
    a) `-m`
    b) `main()`
    c) **Module's full name** ✅
    d) File path

15. What happens if a script does not declare encoding explicitly?
    a) Raises error
    b) **Assumes UTF-8** ✅
    c) Uses Latin-1
    d) Fails execution

16. What function must you use to read command-line arguments?
    a) input()
    b) **import sys** ✅
    c) getargs()
    d) readargs()

17. The interpreter stops execution when it encounters:
    a) Warnings
    b) Print statements
    c) **Syntax errors** ✅
    d) Comments

18. What character starts a Python comment?
    a) `;`
    b) `--`
    c) **`#`** ✅
    d) `//`

19. Which command runs a Python script named `test.py`?
    a) `run test.py`
    b) `bash test.py`
    c) **`python test.py`** ✅
    d) `open test.py`

20. What is the role of the Python shell?
    a) Run OS commands
    b) **Run Python commands interactively** ✅

c) Format JSON
d) Build binaries

21. What is required to run interactive Python from command line?
a) `.py` file
b) **No file, just run `python`** ✅
c) YAML file
d) Text data

22. What does an interactive interpreter help with?
a) Coding in binary
b) **Quick testing** ✅
c) Compiling projects
d) Installing packages

23. What symbol starts encoding declarations in Python?
a) `%`
b) `#` ✅
c) `@`
d) `*`

24. What kind of prompt does Python interpreter show on new line in blocks?
a) `:`
b) `>>>`
c) `...` ✅
d) `//`

25. What happens if an invalid encoding is declared?
a) Python ignores
b) **Script throws error** ✅
c) Uses ASCII
d) Interprets partially

🟡 Medium Level (Q26–Q50)

26. What does `python -c 'print("Hi")'` do?
a) Compiles `print("Hi")`
b) Fails to run
c) **Executes the code as a command** ✅
d) Opens editor

27. When is `sys.argv[0]` set to empty string?
a) When file is missing

b) **When no script or argument is passed** ✅
c) When Python crashes
d) Always

28. In which encoding are Python 3 source files treated by default?
a) Latin-1
b) ASCII
c) **UTF-8** ✅
d) Binary

29. What happens if UTF-8 encoded file is read as ASCII?
a) Interprets correctly
b) **Throws decode error** ✅
c) Reverts to default
d) Ignores differences

30. When is the encoding declaration required as the 2nd line?
a) When using input()
b) **When file starts with shebang** ✅
c) When importing
d) In Windows

31. What module should be imported to work with command-line arguments?
a) argparse
b) os
c) **sys** ✅
d) subprocess

32. What is the default behavior of `sys.argv` if no arguments are passed?
a) Empty list
b) Throws error
c) **List with one item (script name)** ✅
d) None

33. Which encoding is **strongly recommended** for Python source files?
a) cp1252
b) ASCII
c) **UTF-8** ✅
d) ISO-8859-1

34. What is the effect of an incorrect shebang line in Unix?
a) Script runs normally
b) **Interpreter may not be found** ✅
c) Encoding error
d) Prompts for user input

35. How does Python treat `-c` flag?
    a) Reads config
    b) Opens config
    c) **Executes code from the string** ✅
    d) Sets character encoding

36. What happens when you pass `-m` with a module name?
    a) Lists all modules
    b) **Runs the module as a script** ✅
    c) Installs the module
    d) Compiles the module

37. What is returned by `sys.argv[1:]`?
    a) Python version
    b) **List of arguments excluding script name** ✅
    c) Full script
    d) All environment variables

38. When does `...` prompt appear in interactive mode?
    a) On first line
    b) **When a block is continued** ✅
    c) During errors
    d) After print statement

39. The default encoding if not declared in Python 3 is:
    a) ASCII
    b) cp1251
    c) Latin-1
    d) **UTF-8** ✅

40. `#!/usr/bin/env python3` is used for:
    a) Compiling bytecode
    b) **Cross-platform interpreter invocation** ✅
    c) Creating VMs
    d) Replacing OS shell

41. Python script files typically start with:
    a) HTML declaration
    b) JSON header
    c) **Shebang and optional encoding declaration** ✅
    d) XML tag

42. What will `python -m http.server` do?
    a) Fail silently
    b) Compile a module
    c) **Start a simple HTTP server** ✅

d) Run a debugger

43. The second prompt `...` continues until:
   a) Enter key
   b) **Block ends** ✅
   c) Print statement
   d) Compilation

44. When using Python in a script, you write:
   a) `run()`
   b) **Code directly** ✅
   c) Compile header
   d) Interpreter definition

45. Python automatically decodes source using:
   a) OS locale
   b) cp1251
   c) **UTF-8 unless specified otherwise** ✅
   d) Latin-9

46. To test multiple lines in interactive mode:
   a) Use `import`
   b) **Indent properly and press Enter twice** ✅
   c) Add `break`
   d) Use block tags

47. Which tool offers more structured argument parsing than `sys`?
   a) getopt
   b) optparse
   c) **argparse** ✅
   d) os

48. In Python 3, how are scripts encoded by default?
   a) ASCII
   b) cp1252
   c) **UTF-8** ✅
   d) Binary

49. Declaring encoding is especially necessary when:
   a) Using standard library
   b) Running on Linux
   c) **Including non-ASCII characters** ✅
   d) Importing modules

50. The interactive interpreter ends execution when:
   a) print() is used
   b) Comments are written

c) **exit() or Ctrl+D is used** ✅
d) Blank line is entered

🔴 Hard Level (Q51–Q75)

51. Which of these is a valid encoding declaration format?
    a) `// encoding: utf-8`
    b) `## python3`
    c) **`# -*- coding: utf-8 -*-`** ✅
    d) `::encoding=utf8::`

52. The shebang line is mainly used in:
    a) Windows
    b) **Unix/Linux systems** ✅
    c) Python virtual environments
    d) REPL environments

53. What does `#!/usr/bin/env python3` help ensure?
    a) Better error handling
    b) Use of Python 2
    c) **Environment-resolved Python path** ✅
    d) Static typing

54. What is the significance of line 1 and 2 in a Python file?
    a) Versioning
    b) **Shebang and encoding declarations** ✅
    c) Imports
    d) Function definition

55. `python -m timeit` is used to:
    a) Import timer module
    b) **Measure execution time of code** ✅
    c) Run time-related script
    d) Log script start time

56. If your script has non-ASCII characters, what must you do?
    a) Avoid using them
    b) **Declare correct encoding** ✅
    c) Convert to binary
    d) Use encode() function

57. The `-i` option for Python does what?
    a) Imports an image
    b) **Enters interactive mode after script runs** ✅

c) Ignores warnings
d) Inspects output

58. What does `python -m this` do?
a) Shows Python license
b) Lists sys paths
c) **Prints the Zen of Python** ✅
d) Compiles modules

59. If a user enters a line with unclosed parenthesis in interactive mode:
a) Error
b) **`...` prompt continues until closed** ✅
c) Stops execution
d) Restarts shell

60. What tool can help simulate script input for testing?
a) random
b) os
c) **input redirection / piping** ✅
d) threading

61. What causes `UnicodeDecodeError` during script execution?
a) Wrong indentation
b) Syntax error
c) **Mismatched source encoding** ✅
d) Invalid loop

62. What is `PYTHONSTARTUP` environment variable used for?
a) Boot Python faster
b) **Run custom script on interpreter start** ✅
c) Start Python in safe mode
d) Cache pip modules

63. Running `python3 -u script.py` will:
a) Upload script
b) Update modules
c) **Force unbuffered binary stdout/stderr** ✅
d) Undo script

64. Which of these flags starts Python in optimized mode?
a) `-dev`
b) `-perf`
c) **`-O`** ✅
d) `-safe`

65. What is the result of `sys.argv[3]` if only one argument is passed?
- a) Returns 0
- b) **IndexError** ✅
- c) None
- d) True

66. Which is the correct way to run a module as a script?
- a) `python script`
- b) `run script`
- c) **`python -m modulename`** ✅
- d) `./modulename`

67. What is the primary difference between `sys.argv` and `argparse`?
- a) Execution time
- b) Output
- c) **argparse provides better structure/validation** ✅
- d) sys.argv is deprecated

68. Python throws a syntax error in interactive mode when:
- a) String is used
- b) Function is defined
- c) **Indentation is inconsistent** ✅
- d) Comments are added

69. What is the role of `PYTHONPATH`?
- a) Stores function logs
- b) **Specifies additional module paths** ✅
- c) Buffers file access
- d) Defines python version

70. What happens if `# coding:` line is malformed?
- a) UTF-8 used by default
- b) **SyntaxError is raised** ✅
- c) No effect
- d) Python auto-corrects

71. Running `python -c 'import this'` results in:
- a) Empty output
- b) Error
- c) **Zen of Python** ✅
- d) Module info

72. Which version of Python introduced UTF-8 as default?
- a) 2.7
- b) **3.0** ✅
- c) 3.5

d) 3.10

73. Why use `#!/usr/bin/env python3` instead of direct path?
    a) Easier to read
    b) Faster to type
    c) **More portable across environments** ✅
    d) Required for Windows

74. Why is interactive mode helpful for learners?
    a) Offers templates
    b) **Immediate feedback on commands** ✅
    c) Skips errors
    d) Avoids syntax

75. What is printed if `sys.argv` is accessed without import?
    a) Error
    b) Empty list
    c) **NameError** ✅
    d) "argv"

Chapter 3 is titled **"An Informal Introduction to Python"**, covering topics such as numbers, strings, lists, slicing, basic control flow, Fibonacci series, and use of the interpreter.

Now generating **75 MCQs (25 Easy, 25 Medium, 25 Hard)** based on all the topics in Chapter 3.

📘 Chapter 3: An Informal Introduction to Python — Full 75 MCQs

---

🟢 Easy Level (Q1–Q25)

1. What does `2 + 2` evaluate to in Python?
    a) 2
    b) **4** ✅
    c) 6
    d) 22

2. What symbol is used for exponentiation in Python?
    a) ^
    b) *
    c) ** ✅

d) **exp()**

3. The result of `17 / 3` is:
   a) 5
   b) 5.6
   c) **5.666...** ✅
   d) 6

4. Floor division in Python is performed using:
   a) %
   b) /
   c) ^
   d) **//** ✅

5. What is the result of `17 % 3`?
   a) **2** ✅
   b) 3
   c) 5
   d) 1

6. Which operator returns the remainder?
   a) /
   b) **%** ✅
   c) //
   d) *

7. How do you represent strings in Python?
   a) Only single quotes
   b) Only double quotes
   c) **Single or double quotes** ✅
   d) Backticks

8. What does `len('hello')` return?
   a) 6
   b) **5** ✅
   c) 4
   d) Error

9. Strings in Python are:
   a) Mutable
   b) **Immutable** ✅
   c) Executable
   d) Dynamic

10. What does `'Python'[0]` return?
   a) y
   b) n

c) **P** ✅
d) h

11. Which index accesses the last character of a string?
    a) 0
    b) **-1** ✅
    c) -2
    d) last

12. What is the result of `'Python'[1:3]`?
    a) th
    b) Pyt
    c) **yt** ✅
    d) hon

13. What happens when indexing beyond the string length?
    a) Returns empty
    b) **Raises IndexError** ✅
    c) Returns last character
    d) None

14. What does `'Python'[4:]` return?
    a) **on** ✅
    b) thon
    c) ho
    d) h

15. Slicing a string with `word[0:2]` returns:
    a) Py
    b) th
    c) **Py** ✅
    d) Pyt

16. Which symbol is used to concatenate strings?
    a) &
    b) %
    c) **+** ✅
    d) :

17. `word[:2] + word[2:]` is equal to:
    a) Error
    b) Partial string
    c) **word** ✅
    d) Sliced twice

18. What does `'J' + 'ython'` produce?
    a) Jython ✅

b) Python
c) Error
d) String

19. Which of the following types is mutable?
    a) String
    b) Tuple
    c) **List** ✅
    d) Integer

20. A list is denoted by:
    a) {}
    b) ()
    c) **[]** ✅
    d) <>

21. What is the output of `[1, 2, 3][0]`?
    a) 0
    b) **1** ✅
    c) 2
    d) None

22. Which method adds an element to the end of a list?
    a) push()
    b) insert()
    c) **append()** ✅
    d) add()

23. Lists support slicing similar to:
    a) Dictionaries
    b) **Strings** ✅
    c) Tuples only
    d) Sets

24. Lists can contain:
    a) Only integers
    b) Only strings
    c) **Mixed data types** ✅
    d) Only floats

25. `cubes[3] = 64` does what?
    a) Appends 64
    b) **Replaces index 3 with 64** ✅
    c) Removes item
    d) Raises error

🟡 Medium Level (Q26–Q50)

26. What does `cubes.append(343)` do?
    a) Creates new list
    b) **Adds 343 at end** ✅
    c) Sorts list
    d) Adds at start

27. What is the effect of `letters[2:5] = []`?
    a) Replaces with empty list
    b) Appends nothing
    c) **Removes items at indices 2 to 4** ✅
    d) Clears list

28. Which operator is used for power operations?
    a) ^
    b) ** ** ✅
    c) *
    d) sqrt()

29. What is the use of underscore _ in calculator mode?
    a) Placeholder
    b) **Stores last result** ✅
    c) Delimiter
    d) Comment

What is the output of:

a = ['x', 'y']

b = [1, 2]

[a, b]

30. a) Mixed list
    b) **Nested list** ✅
    c) Error
    d) Flat list

31. In nested list `x = [a, b]`, `x[0][1]` refers to:
    a) b
    b) x
    c) **'y'** ✅
    d) 2

32. What’s the output of `len([1,2,3])`?
  a) **3** ✅
  b) 2
  c) Error
  d) 1

33. Can lists be sliced like strings?
  a) **Yes** ✅
  b) No
  c) Only tuples
  d) Only ranges

34. Lists are different from strings because:
  a) Use different syntax
  b) Can store integers
  c) **Are mutable** ✅
  d) Use loop

35. How do you clear a list?
  a) del list
  b) list.clear()
  c) list.remove()
  d) **list[:] = []** ✅

36. What does `range(0,10,2)` generate?
  a) **Even numbers from 0 to 8** ✅
  b) 0 to 10
  c) Odd numbers
  d) Error

37. Fibonacci series is generated using:
  a) for loop
  b) **while loop** ✅
  c) recursion
  d) map()

38. Multiple assignment works like:
  a) a,b = b,a
  b) x = y = z
  c) **a,b = 0,1** ✅
  d) import *

39. What is the result of `round(113.0625, 2)`?
  a) **113.06** ✅
  b) 113.00
  c) 113.05

d) 113

40. What is the role of `end=','` in `print()`?
    a) Error
    b) Adds newline
    c) **Suppresses newline and adds comma** ✅
    d) Formats float

41. Strings can be:
    a) Reassigned
    b) **Sliced** ✅
    c) Mutated
    d) Iterated

42. Tuple syntax uses:
    a) {}
    b) **()** ✅
    c) []
    d) <>

43. `spam = 1 # comment` — what happens?
    a) Error
    b) **1 assigned and comment ignored** ✅
    c) comment assigned
    d) spam deleted

44. Escape character for `'` in string is:
    a) /
    b) ""
    c) \ ✅
    d) *

45. `"doesn't"` is valid because:
    a) In double quotes ✅
    b) Contains escape
    c) Not valid
    d) Not evaluated

46. `'spam' + 'eggs'` = ?
    a) spam
    b) eggs
    c) **spameggs** ✅
    d) spam eggs

47. Strings and lists both support:
    a) Iteration
    b) Indexing

c) Slicing
d) **All of the above** ✅

48. Negative indexing starts from:
a) 0
b) End
c) **-1** ✅
d) -2

49. `print('A', end='')` + `print('B')` gives:
a) AB ✅
b) A\nB
c) A B
d) BA

50. Python treats `#` as:
a) **Comment marker** ✅
b) Hash function
c) Import
d) Escape

Continuing with the **Hard Level (Q51–Q75)** MCQs for:

🔴 Hard Level (Q51–Q75)

51. What does the following code output?

```
a, b = 0, 1

while b < 5:

    print(b, end=',')

    a, b = b, a + b
```

a) 1,2,3,5
b) 0,1,1,2
c) **1,1,2,3,** ✅
d) 1,2,3,4,5

52. Which of the following is valid string concatenation?
a) `'a' + b`
b) `"a" + 1`

c) **`"a" + "b"`** ✅
d) `"a" + 2.0`

53. What will `word[1:100]` return if word is `'hello'`?
    a) Error
    b) 'ello '
    c) **'ello'** ✅
    d) 'el'

54. Which of the following will cause an error?
    a) `'abc' * 3`
    b) `'abc' + 'def'`
    c) **`'abc'[10]`** ✅
    d) `'abc'[0:2]`

55. What does `list(range(5, 1, -1))` return?
    a) [5, 4, 3, 2, 1]
    b) [1, 2, 3, 4, 5]
    c) **[5, 4, 3, 2]** ✅
    d) [5, 4, 3, 2, 1, 0]

56. What does `len(['a', ['b', 'c']])` return?
    a) 3
    b) **2** ✅
    c) 1
    d) Error

57. What is the result of `['a', 'b'] * 2`?
    a) **['a', 'b', 'a', 'b']** ✅
    b) ['a', 'a', 'b', 'b']
    c) ['aa', 'bb']
    d) Error

58. Which of the following modifies a list in place?
    a) `list + [4]`
    b) `newlist = list + [4]`
    c) **`list.append(4)`** ✅
    d) `list = list * 2`

59. What happens if you assign to a slice?

```
letters[1:3] = ['x', 'y']
```

a) Inserts
b) Appends
c) **Replaces items 1 and 2** ✅
d) Deletes list

60. How do you replace the last item in a list `l` with `'end'`?
    a) `l[0] = 'end'`
    b) `l[len(l)] = 'end'`
    c) **`l[-1] = 'end'`** ✅
    d) `l.end = 'end'`

61. Which built-in type supports nesting?
    a) string
    b) int
    c) float
    d) **list** ✅

62. Which slicing technique clones a list?
    a) `list[0:len(list)]`
    b) `list[1:]`
    c) `list[::1]`
    d) **`list[:]`** ✅

63. If `a = [1,2]`, what does `b = [a, a]` do?
    a) Makes deep copy
    b) Makes unrelated copies
    c) **Makes list with two references to same list** ✅
    d) Throws error

64. `a[0] = 100` in above scenario changes:
    a) b only
    b) a only
    c) **both elements in b** ✅
    d) none

65. What happens when slicing a string like `s[5:2]`?
    a) Gives partial
    b) **Returns empty string** ✅
    c) Throws error
    d) Starts from 2

66. What does `list[::2]` do?
    a) Reverses
    b) Skips one item
    c) **Every second item** ✅

d) Last item only

67. `'spam' * 0` evaluates to:
a) spam
b) None
c) **''** ✅
d) Error

68. Which of these cannot be used as list element?
a) int
b) string
c) list
d) **None of the above** ✅

69. Which symbol is used to escape characters in strings?
a) /
b) &
c) \ ✅
d) #

70. What is output of `print('py\nthon')`?
a) py t h o n
b) py\nthon
c) **py
thon** ✅
d) syntax error

71. `'abc'[::-1]` gives:
a) 'abc'
b) 'acb'
c) **'cba'** ✅
d) syntax error

72. What does `'abc' * 2 + 'd'` return?
a) abcabc
b) **abcabcd** ✅
c) abcdabc
d) syntax error

73. Why does slicing not throw errors beyond string bounds?
a) Python is forgiving
b) Error is hidden
c) **Slicing automatically adjusts** ✅
d) Interpreter skips

74. Which of these can be used to generate Fibonacci series?
a) math.fibonacci()

b) random()
c) **loop with tuple unpacking** ✅
d) statistics module

75. What's printed by:

```
a, b = 0, 1

while b < 10:

    print(b)

    a, b = b, a+b
```

a) 1,2,3,...,10
b) **1\n1\n2\n3\n5\n8** ✅
c) 1,1,2,3
d) 0,1,2,3,4

Great! Chapter 4 is **"More Control Flow Tools"**, and it includes:

- `if`, `for`, `while` statements
- `range()` function
- `break`, `continue`, `pass`
- Looping techniques
- `match` statements
- Function definitions, `def`, default values
- `*args`, `**kwargs`, `lambda`, annotations, docstrings

Now generating **75 MCQs (25 Easy, 25 Medium, 25 Hard)** covering all of Chapter 4.

📘 Chapter 4: More Control Flow Tools — Full 75 MCQs

🟢 Easy Level (Q1–Q25)

1. Which statement is used to make decisions in Python?
   a) for
   b) while
   c) **if** ✅
   d) def

2. The `else` block in an `if` statement is executed when:
   a) condition is true
   b) **all conditions are false** ✅
   c) an error occurs
   d) when loop ends

3. Which keyword stands for ‘else if’?
   a) eif
   b) **elif** ✅
   c) elseif
   d) ifelse

4. Which is a valid loop in Python?
   a) if
   b) **for** ✅
   c) when
   d) check

5. What does `range(5)` return?
   a) [1, 2, 3, 4, 5]
   b) **[0, 1, 2, 3, 4]** ✅
   c) (1, 2, 3, 4)
   d) 1 to 5

6. `for i in range(3):` loops how many times?
   a) 4
   b) **3** ✅
   c) 2
   d) infinite

7. What is the output of `range(2, 6)`?
   a) [2, 3, 4, 5, 6]
   b) **[2, 3, 4, 5]** ✅
   c) [1, 2, 3]
   d) [6]

8. What is used to exit a loop early?
   a) stop
   b) exit
   c) **break** ✅

d) halt

9. Which keyword skips to the next loop iteration?
   a) next
   b) continue
   c) **continue** ✅
   d) skip

10. What does `pass` do in a block?
    a) throws error
    b) continues loop
    c) **does nothing** ✅
    d) exits loop

11. What does `enumerate(['a', 'b'])` return?
    a) (0, a), (0, b)
    b) (a, 0), (b, 1)
    c) **(0, 'a'), (1, 'b')** ✅
    d) list

12. Looping over a dictionary uses:
    a) items()
    b) keys()
    c) **items()** ✅
    d) values()

13. Which function gives index and value?
    a) loop()
    b) count()
    c) **enumerate()** ✅
    d) zip()

14. The result of `sum(range(4))` is:
    a) 10
    b) **6** ✅
    c) 4
    d) 3

15. `while` loop executes as long as:
    a) variable is assigned
    b) **condition is True** ✅
    c) range ends
    d) file exists

16. How many times does this run?

```
i = 0

while i < 3:

    i += 1
```

a) **3** ✅

b) 2

c) 4

d) infinite

17. `match` statement in Python is similar to:
a) if
b) for
c) **switch/case** ✅
d) goto

18. `def` is used for:
    a) loop
    b) class
    c) **function definition** ✅
    d) condition

19. What does this return?

```
def add(x, y=5):

    return x + y

add(3)
```

a) 8 ✅

b) 5

c) 3

d) Error

20. `lambda x: x + 1` is:
a) recursive
b) **anonymous function** ✅
c) class
d) exception

21. Default function arguments:
a) Must be passed
b) **Can be omitted** ✅
c) Are errors
d) None

22. What type is `*args`?
a) dict
b) list
c) **tuple** ✅
d) set

23. `**kwargs` is used for:
a) sequences
b) **keyword arguments** ✅
c) files
d) lists

24. The body of an `if` or `def` is:
a) within brackets
b) **indented** ✅
c) comma-separated
d) underlined

25. Python allows function annotations to:
a) Comment code
b) Run faster
c) **Indicate expected argument types** ✅
d) Encrypt function

26. What will this code print?

```
for i in range(1, 4):

   print(i, end=' ')
```

a) 0 1 2
b) **1 2 3** ✅

c) 1 2 3 4
d) 0 1 2 3

27. What happens if `break` is used inside a loop?
    a) Skips current iteration
    b) **Exits the loop immediately** ✅
    c) Restarts loop
    d) Continues next block

28. What does the following code do?

```
for i in [1, 2, 3]:

    if i == 2: continue

    print(i)
```

a) **1 and 3** ✅
b) 1, 2
c) 2, 3
d) Error

29. What is the output?

```
def greet(name='User'):

    print("Hi", name)

greet()
```

a) Hi
b) Error
c) Hi None
d) **Hi User** ✅

30. Which of the following accepts variable number of arguments?
    a) `def fun(*args)` ✅
    b) `def fun(args[])`
    c) `def fun(args*)`
    d) `def fun[*args]`

31. What does `zip(['a','b'], [1,2])` return?
    a) dict

b) list of values
c) **zip object with tuples** ✅
d) Error

32. What is the result of `range(10, 1, -3)`?
    a) [10, 7, 4, 1]
    b) **[10, 7, 4]** ✅
    c) [1, 4, 7]
    d) Error

33. Which is a valid use of `match` in Python 3.10+?

```
match command:

    case "start": print("Running")
```

a) if-else
b) **Pattern matching** ✅
c) Switch
d) Map

34. What is the purpose of function annotations?

```
def add(x: int, y: int) -> int:
```

a) Compile types
b) **Provide type hints** ✅
c) Generate docstring
d) Declare constants

35. What is printed by this?

```
for i in range(3):

    pass

print(i)
```

a) 2 ✅
b) 3

c) Error
d) i

36. What does this do?

```
for key, value in d.items():
```

a) **Unpacks key-value pairs in a dictionary** ✅
b) Reverses dict
c) Loops values
d) Adds values

37. What will this return?

```
def demo(x, y=2, z=3):

    return x + y + z

demo(1, z=5)
```

a) **8** ✅
b) 6
c) 10
d) Error

38. How does Python interpret the following?

```
def foo(*args, **kwargs): pass
```

a) Only positional args
b) Only keyword args
c) **Any number of both positional and keyword args** ✅
d) None allowed

39. What is output of this loop?

```
for i in range(3):

    print(i, end=' ')
```

a) **0 1 2** ✅
b) 1 2 3
c) 0 1 2 3
d) Error

40. What does the `match` statement require to work properly?
    a) Python < 3.9
    b) A loop
    c) **Patterns or literals to match** ✅
    d) Static types

41. In a `match` block, what does `case _:` mean?
    a) Continue
    b) **Default or fallback case** ✅
    c) Pass
    d) Error

42. Which of these supports early return from a function?
    a) yield
    b) **return** ✅
    c) break
    d) stop

43. What happens with this code?

```
def f(x, y): return x + y

print(f(2))
```

a) 2
b) Error ✅
c) 0
d) 4

44. In a `for` loop, `range(len(list))` allows:
    a) direct iteration
    b) **index-based access** ✅
    c) modification
    d) slicing

45. What is the result of `list(reversed([1, 2, 3]))`?
    a) [3, 2, 1] ✅
    b) (1, 2, 3)
    c) error

    d) [1, 2, 3, 3]

46. When are default arguments evaluated in a function?
    a) Each call
    b) **At definition time** ✅
    c) At runtime
    d) During import

47. Which of these is **not** valid syntax?

```
def f(x, y, /): return x+y
```

a) Valid ✅
b) Invalid
c) Only in Python 2
d) Deprecated

48. A lambda function can contain:
    a) statements
    b) multiple lines
    c) **only expressions** ✅
    d) imports

49. What does `globals()` return?
    a) Local vars
    b) Stack
    c) **Global namespace dict** ✅
    d) List

50. A function without return:
    a) Raises error
    b) Returns 0
    c) **Returns None** ✅
    d) Returns false

🔴 Hard Level (Q51–Q75)

51. What will be the output?

```
x = 0

while x < 5:

    if x == 3: break

    x += 1

print(x)
```

a) 5
b) **3** ✅
c) 0
d) 4

52. What happens if you modify a list while iterating over it?
    a) Safe
    b) **May skip elements or behave unexpectedly** ✅
    c) Loops infinitely
    d) Error

53. In function parameters, what does `/`, `*`, mean?
    a) Comment
    b) Deprecated
    c) **Positional-only, keyword-only indicator** ✅
    d) Error

54. What is output of:

```
def f(a, b=2, c=3):

    print(a, b, c)

f(1, c=5)
```

a) **1 2 5** ✅
b) 1 5 3
c) 2 3 5
d) Error

55. Which of these is invalid?

```
def f(x=1, y): pass
```

a) Works
b) Returns tuple
c) **Raises SyntaxError** ✅
d) Calls normally

56. What does `reversed(range(3))` produce?
    a) [1, 2, 3]
    b) Error
    c) **An iterator from 2 to 0** ✅
    d) Infinite loop

57. How does pattern matching handle types?

```
match x:

   case int(): pass
```

a) Checks value
b) **Checks instance type** ✅
c) Converts
d) Ignores

58. When defining `def f(x: int = 10)`, the type hint:
    a) Enforces int
    b) **Is just a suggestion** ✅
    c) Throws type error if wrong
    d) Is mandatory

59. Why might `default` values cause unexpected behavior?

```
def f(x=[]): x.append(1); return x
```

a) Appends randomly
b) Only returns 1
c) **Mutates shared list** ✅
d) Creates new list every time

60. What is the result of:

```
def f(*args): return args
```

f(1,2,3)

a) Error
b) [1,2,3]
c) **(1,2,3)** ✅
d) 1

61. How does `lambda x=1, y=2: x+y` behave when called as `f()`?
    a) Error
    b) 0
    c) **3** ✅
    d) 1

62. What is returned by this code?

```
def f(x, *args): return args

f(1,2,3)
```

a) (1,2)
b) [2,3]
c) **(2,3)** ✅
d) Error

63. A generator can be used with `for` loop because:
    a) It's a list
    b) **It's iterable** ✅
    c) It's cached
    d) It's global

64. What does this print?

```
for i in range(3):

   if i == 1: continue

   print(i)
```

a) 1
b) **0 and 2** ✅
c) 0,1,2
d) 2

65. Which construct helps when function signature is unknown?
    a) def f(x)
    b) def f(x, y)
    c) **def f(*args, **kwargs)** ✅
    d) def f(args)

66. What is the role of _ in `case _:` in pattern matching?
    a) Import
    b) Variable
    c) **Wildcard / default match** ✅
    d) None

67. What happens if you write:

```
for x in range(3):

    pass

else:

    print('done')
```

a) Skips else
b) **Prints 'done'** ✅
c) Loops again
d) Error

68. The keyword `yield` is used to:
    a) End function
    b) **Create generators** ✅
    c) Return and exit
    d) Save memory

69. How many times does this execute?

```
i = 0

while i < 3:

    print(i)

    i += 1
```

a) 4
b) **3** ✅
c) 2
d) Infinite

70. What does `zip(*list)` do?
    a) Reverses list
    b) **Unzips a zipped list** ✅
    c) Compresses
    d) Slices

71. In `for key in sorted(dict):`, what does it do?
    a) Iterate random
    b) Iterate values
    c) **Sorted iteration of keys** ✅
    d) Error

72. Which is a valid `lambda`?
    a) `lambda x: x + 2` ✅
    b) `lambda: x = 2`
    c) `lambda x y: x+y`
    d) `def lambda x`

73. Using `del` on a list slice:
    a) Copies list
    b) **Deletes elements from list** ✅
    c) Clears list
    d) None

74. Which is a safe way to avoid default argument pitfalls?
    a) def f(x=[]): ...
    b) **def f(x=None): if x is None: x = []** ✅
    c) Always use int
    d) Return default

75. Which of these evaluates to `False`?

bool([]), bool(0), bool(None)

a) [], 0
b) Only None
c) **All of them** ✅
d) None

Chapter 5 is titled **"Data Structures"**, covering:

- Lists and list methods
- List comprehensions
- Nested structures and matrix transpose
- `del` statement
- Tuples and sequences
- Sets and set operations
- Dictionaries and their methods

Now generating **75 MCQs (25 Easy, 25 Medium, 25 Hard)** for:

📘 Chapter 5: Data Structures

🟢 Easy Level (Q1–Q25)

1. What is the symbol for a list in Python?
   a) `{}`
   b) `()`
   c) **`[]`** ✅
   d) `<>`

2. What does `len([1, 2, 3])` return?
   a) 2
   b) **3** ✅
   c) 4
   d) None

3. How do you add a single item to the end of a list?
   a) add()
   b) **append()** ✅
   c) push()
   d) insert()

4. Lists in Python are:
   a) Immutable

b) Constant
c) **Mutable** ✅
d) Fixed

5. Which of the following is a tuple?
   a) `[1, 2, 3]`
   b) `{"a": 1}`
   c) **`(1, 2, 3)`** ✅
   d) `{1, 2, 3}`

6. What type is created by `set([1, 2, 2, 3])`?
   a) [1, 2, 2, 3]
   b) {1:2, 3:1}
   c) **{1, 2, 3}** ✅
   d) (1, 2, 3)

7. Which structure does not allow duplicates?
   a) list
   b) tuple
   c) dict
   d) **set** ✅

8. What symbol is used for dictionary key-value pairs?
   a) `:` ✅
   b) `=`
   c) `->`
   d) `,`

9. What does `d = {}` create?
   a) **Empty dictionary** ✅
   b) Empty list
   c) Empty tuple
   d) None

10. Which method removes all items from a list?
    a) remove()
    b) delete()
    c) **clear()** ✅
    d) pop()

11. What is the result of `[1, 2] + [3]`?
    a) [1, 2, 3] ✅
    b) [4, 5]
    c) Error
    d) (1, 2, 3)

12. Which of the following is a valid dictionary key?
    a) []
    b) {}
    c) **(1, 2)** ✅
    d) set([1])

13. What does `list(set([1, 1, 2, 3]))` return?
    a) [1, 1, 2, 3]
    b) **[1, 2, 3]** ✅
    c) [3, 2, 1]
    d) Error

14. Which function can convert a sequence to a list?
    a) convert()
    b) **list()** ✅
    c) array()
    d) dict()

15. Which method removes the last item from a list?
    a) delete()
    b) **pop()** ✅
    c) cut()
    d) shift()

16. What does `a[0]` return when `a = [10, 20, 30]`?
    a) 20
    b) **10** ✅
    c) 30
    d) Error

17. What's printed?

a = [1, 2, 3]

print(a[-1])

a) 1
b) 2
c) **3** ✅
d) Error

18. How is a set defined with curly braces?
    a) `[]`
    b) `()`
    c) **`{1, 2}`** ✅

d) `<>`

19. What happens when you do:

a = [1,2,3]

del a[1]

a) Deletes last element
b) **Deletes element at index 1** ✅
c) Error
d) Removes whole list

20. `a = (1,)` is:
a) int
b) list
c) **tuple** ✅
d) set

21. What is the output of `list('abc')`?
a) abc
b) [‘abc’]
c) **['a', 'b', 'c']** ✅
d) ['a-b-c']

22. Which data type is unordered and indexed by keys?
a) list
b) **dictionary** ✅
c) tuple
d) set

23. Sets are created using which constructor?
a) new()
b) dict()
c) **set()** ✅
d) array()

24. `in` keyword in sets tests for:
a) identity
b) equality
c) **membership** ✅
d) assignment

25. What is the result of `'name' in {'name': 'John'}`?
a) False

b) Error
c) **True** ✅
d) None

🟡 Medium Level (Q26–Q50)

26. What does the `insert()` method do in a list?
    a) Appends to end
    b) Removes item
    c) **Inserts an item at a specific index** ✅
    d) Sorts the list

27. What does `dict.get('key', 'default')` return if key is missing?
    a) Error
    b) None
    c) KeyError
    d) **'default'** ✅

28. Which method adds multiple items to a list?
    a) append()
    b) **extend()** ✅
    c) insert()
    d) update()

29. What will `list(reversed([1, 2, 3]))` return?
    a) **[3, 2, 1]** ✅
    b) [1, 2, 3]
    c) (3, 2, 1)
    d) Error

30. What does the following list comprehension produce?

[x**2 for x in range(3)]

a) [1, 2, 3]
b) **[0, 1, 4]** ✅
c) [2, 4, 6]
d) [1, 4, 9]

31. What does `mylist.remove(2)` do?
    a) Removes index 2
    b) **Removes value 2** ✅

c) Deletes the list
d) Removes all values

32. What is the output?

```
set('banana')
```

a) {'b', 'a', 'n'} is not a) — a) {'banana'}
b) {'b', 'a', 'n'} ✅
c) {'b', 'a'}
d) ['b', 'a', 'n']

33. What does `sorted({'a': 1, 'b': 2})` return?
   a) dict
   b) **List of keys** ✅
   c) List of values
   d) Sorted dict

34. Which data type does not maintain insertion order before Python 3.7?
   a) list
   b) tuple
   c) **dict** ✅
   d) set

35. What does this return?

```
{x for x in range(5) if x % 2 == 0}
```

a) list
b) tuple
c) **set of even numbers** ✅
d) dict

36. Which method removes and returns an arbitrary item from a set?
   a) remove()
   b) discard()
   c) **pop()** ✅
   d) del()

37. What is the result of this?

```
(1, 2) + (3,)
```

a) Error
b) (1, 2, 3) ✅
c) (1, 2)
d) [1, 2, 3]

38. What happens when `del list[:]` is executed?
    a) Deletes some elements
    b) Deletes one element
    c) **Clears the entire list** ✅
    d) None

39. What is the output?

```
d = {'a': 1}

d.update({'b': 2})
```

a) Only 'b': 2
b) Only 'a': 1
c) **{'a': 1, 'b': 2}** ✅
d) Error

40. Which statement checks for common elements in sets A and B?
    a) A + B
    b) A - B
    c) A * B
    d) **A & B** ✅

41. What's the result of:

```
[0] * 4
```

a) 0
b) [0, 0]
c) **[0, 0, 0, 0]** ✅
d) Error

42. How do you reverse a list in place?
    a) reverse(lst)
    b) **lst.reverse()** ✅
    c) lst.reversed()

d) reversed(lst)

43. What does `.keys()` return from a dictionary?
    a) Values
    b) List
    c) **View object of keys** ✅
    d) Tuple

44. What is the output of `dict([('a', 1), ('b', 2)])`?
    a) List
    b) **{'a': 1, 'b': 2}** ✅
    c) [('a', 1), ('b', 2)]
    d) Error

45. What’s the output of `len(set([1,2,2,3]))`?
    a) 4
    b) **3** ✅
    c) 2
    d) Error

46. Which method creates a shallow copy of a list?
    a) list()
    b) copy.deepcopy()
    c) **copy()** ✅
    d) clone()

47. Which operation is not allowed on a set?
    a) add()
    b) update()
    c) **indexing like set[0]** ✅
    d) remove()

48. How do you remove a key from a dict safely?
    a) del
    b) **pop()** ✅
    c) delete()
    d) remove()

49. What is a key property of tuples?
    a) Mutable
    b) Iterable
    c) **Immutable** ✅
    d) Sorted

50. Which is valid for dictionary comprehension?
    a) for i in dict
    b) [k:v for k in range(3)]

c) **{k: k**2 for k in range(3)}** ✅
d) (k: v for k)

🔴 Hard Level (Q51–Q75)

51. What will this list comprehension produce?

[[row[i] for row in matrix] for i in range(3)]

a) Flattened matrix
b) **Transposed matrix** ✅
c) Reversed matrix
d) Identity matrix

52. What is the result of:

list(zip(*[[1,2], [3,4], [5,6]]))

a) **[(1, 3, 5), (2, 4, 6)]** ✅
b) [(1, 2), (3, 4), (5, 6)]
c) [(1, 2, 3), (4, 5, 6)]
d) Error

53. Which expression removes all elements from a list `a` and keeps the same object?
    a) `a = []`
    b) `a.clear()`
    c) `del a[:]`
    d) **Both b and c** ✅

54. What does the following return?

set([1, 2]) == set([2, 1, 2])

a) False
b) Error
c) **True** ✅
d) None

55. What is the purpose of `del a[:]`?
    a) Deletes variable
    b) Removes one element
    c) **Clears all elements in-place** ✅
    d) Copies list

56. What is the output of:

d = {'x': 1}; d.update(y=2)

a) Error
b) **{'x': 1, 'y': 2}** ✅
c) {'x': 1}
d) {'x': 2}

57. How can you safely access a non-existent key in a dictionary?
    a) d[key]
    b) **d.get(key)** ✅
    c) d.pop(key)
    d) d.key

58. In a nested list, how can you access the second item of the second list?

x = [[1, 2], [3, 4]]

a) x[1]
b) x[2][1]
c) **x[1][1]** ✅
d) x[0][2]

59. What will be returned by this comprehension?

{c: c.upper() for c in 'abc'}

a) {‘a’:’a’, ‘b’:’b’, ‘c’:’c’}
b) **{‘a’:’A’, ‘b’:’B’, ‘c’:’C’}** ✅
c) Error
d) [‘A’, ‘B’, ‘C’]

60. Why can’t lists be used as dictionary keys?
    a) Too large

b) Must be strings
c) **They are mutable and unhashable** ✅
d) Only numbers allowed

61. What will happen?

```
a = {}; a['x'] = a
```

a) Error
b) **Creates self-referential dict** ✅
c) Deletes dict
d) Circular ref crash

62. What does the following evaluate to?

```
(1, 2, [3, 4])[2][0]
```

a) 1
b) 3 ✅
c) Error
d) [3]

63. Which method is unavailable for tuples?
a) index()
b) count()
c) **append()** ✅
d) len()

64. Which of these creates a set of squares from a list?

```
{x**2 for x in [1,2,3]}
```

a) list
b) **set** ✅
c) dict
d) tuple

65. What's the result of:

```
[[]] * 3
```

a) 3 separate lists
b) **3 references to the same list** ✅
c) [[], [], []]
d) Error

66. Which of the following modifies only a copy of a list?

a = [1,2]; b = a[:]; b[0] = 10

a) a is changed
b) **a remains [1,2]** ✅
c) Both changed
d) Error

67. What will `frozenset([1,2])` do?
    a) Raise TypeError
    b) **Create immutable set** ✅
    c) Create dict
    d) Create tuple

68. Which operation returns the symmetric difference between sets A and B?
    a) A & B
    b) A | B
    c) A - B
    d) **A ^ B** ✅

69. What is the output?

t = (1,); type(t)

a) int
b) list
c) **tuple** ✅
d) Error

70. What is the type of `dict().fromkeys('abc', 0)`?
    a) list
    b) set
    c) **dict** ✅
    d) tuple

71. What happens when you update a dict with another dict having the same keys?
    a) Ignores new
    b) Merges
    c) **Overwrites existing values** ✅
    d) Error

72. Which code returns all key-value pairs in a dictionary?

```
d = {'x':1}; ?
```

a) d.items() ✅
b) d.all()
c) d.pairs()
d) d.entries()

73. Which method returns the number of times a value appears in a tuple?
    a) count() ✅
    b) total()
    c) sum()
    d) find()

74. What is the result of:

```
{n: n*n for n in range(3)}
```

a) Error
b) List
c) **{0: 0, 1: 1, 2: 4}** ✅
d) [0,1,2]

75. Why are sets faster than lists for membership tests?
    a) They use arrays
    b) **They use hash tables** ✅
    c) They are ordered
    d) They are mutable

Chapter 6 from the Python Tutorial is titled **"Modules"** and includes topics such as:

- Creating and importing modules
- Module search path

- Compiled `.pyc` files
- Standard modules
- `dir()` function
- Packages and subpackages

Now generating **75 MCQs (25 Easy, 25 Medium, 25 Hard)** for:

📘 Chapter 6: Modules

🟢 Easy Level (Q1–Q25)

1. What keyword is used to bring in a module?
   a) include
   b) require
   c) **import** ✅
   d) define

2. What is the file extension of Python modules?
   a) .pyc
   b) .mod
   c) **.py** ✅
   d) .pkg

3. How do you import a module named `math`?
   a) load math
   b) import(math)
   c) **import math** ✅
   d) include math

4. What function lists attributes of a module?
   a) help()
   b) list()
   c) **dir()** ✅
   d) attr()

5. What does `import math as m` allow you to do?
   a) Rename Python
   b) **Use math functions with alias `m`** ✅
   c) Delete module

d) Break scope

6. How do you import only the `sqrt` function?
   a) import math.sqrt
   b) **from math import sqrt** ✅
   c) import sqrt
   d) from sqrt import math

7. What happens when you run a module directly?
   a) **Its code runs as `__main__`** ✅
   b) It imports
   c) It fails
   d) Nothing

8. Which variable identifies module execution context?
   a) mod
   b) sys.path
   c) **name** ✅
   d) main

9. What is `__pycache__`?
   a) Cache for variables
   b) Module backup
   c) **Holds compiled .pyc files** ✅
   d) Debug folder

10. Which statement makes a package?
    a) make dir
    b) file.py
    c) **`__init__.py` file in folder** ✅
    d) setup.py

11. What is the output of `dir(math)`?
    a) Error
    b) **List of math module names** ✅
    c) Compiled code
    d) Class names

12. Which symbol is used for wildcard import?
    a) `.`
    b) `%`
    c) `*` ✅
    d) `#`

13. How do you import all symbols from a module?
    a) `import math.*`

b) `load all math`
c) **`from math import *`** ✅
d) import_all(math)

14. What is the search path for module import?
a) `os.environ`
b) **`sys.path`** ✅
c) `os.path`
d) `mod.path`

15. What does `__init__.py` do in a package?
a) Ignore files
b) **Initializes a Python package** ✅
c) Ends module
d) Creates class

16. Which module provides access to interpreter variables?
a) os
b) time
c) **sys** ✅
d) io

17. How do you execute a file as a script and not a module?
a) Use `sys.run()`
b) `python file.py` ✅
c) import it
d) call it

18. What is the function of `help(module)`?
a) Shows variables
b) Deletes module
c) **Displays documentation** ✅
d) Creates a copy

19. Which module lists all built-in functions?
a) builtins
b) **dir(builtins)** ✅
c) core
d) system

20. How do you list modules in the current package?
a) view()
b) **dir()** ✅
c) tree()
d) modules()

21. What will this print?

```
if __name__ == '__main__': print("Run")
```

a) **Prints “Run” when executed directly** ✅
b) Always prints
c) Never prints
d) Error

22. What does `from math import *` do?
    a) Nothing
    b) **Imports all public names** ✅
    c) Deletes math
    d) Imports only sqrt

23. Which module is loaded by default?
    a) socket
    b) **builtins** ✅
    c) os
    d) random

24. What is a package?
    a) A `.py` file
    b) **A folder with `__init__.py` and modules** ✅
    c) zip file
    d) Class

25. Which function shows module docstrings?
    a) doc()
    b) module.info()
    c) **help(module)** ✅
    d) dir(module)

🟡 Medium Level (Q26–Q50)

26. What is the purpose of `__name__ == '__main__'`?
    a) Prevents import
    b) **Allows conditional execution when run directly** ✅
    c) Compiles module
    d) Loads script as service

27. What does this mean?

import module as m

a) Creates two modules
b) Re-imports builtins
c) **Gives alias `m` to `module`** ✅
d) Shortens code to one line

28. What is `sys.modules`?
a) List of built-ins
b) OS modules
c) **Dictionary of loaded modules** ✅
d) String array

29. What happens when a module is re-imported?
a) Recompiled
b) Reloads
c) **Nothing, cached in memory** ✅
d) Deletes previous

30. What does `__pycache__` contain?
a) Source code
b) Documentation
c) **Compiled bytecode (.pyc)** ✅
d) Error logs

31. How do you create a reusable Python component?
a) zip()
b) **Define a .py file with functions/classes** ✅
c) Make folder
d) Use binary

32. How do you run a module as a script from terminal?
a) open module
b) run module
c) **python module.py** ✅
d) load module

33. Where does Python search for modules?
a) sys.argv
b) os.path
c) **sys.path** ✅
d) PATH env only

34. How can you inspect functions inside a module?
   a) open()
   b) ls(module)
   c) **dir(module)** ✅
   d) sys.path

35. How can you display a module's docstring from the terminal?
   a) man module
   b) **python -m pydoc module** ✅
   c) info(module)
   d) run module

36. Which Python file triggers the creation of a package?
   a) .module
   b) setup.py
   c) **init.py** ✅
   d) loader.py

37. How are `.pyc` files used?
   a) Source
   b) Error handling
   c) **Faster loading (compiled code)** ✅
   d) Debugging

38. What's the effect of:

from math import sqrt as s

a) Error
b) Imports math
c) **Imports sqrt as s** ✅
d) Overwrites sqrt

39. What's returned by `dir(os)`?
   a) Documentation
   b) **List of module attributes/methods** ✅
   c) Current dir
   d) Files

40. Can modules import each other?
   a) Never
   b) **Yes, if accessible in path** ✅
   c) Only from stdlib
   d) Only at runtime

41. Which of the following reloads a module in Python 3?
    a) `reload(module)`
    b) **`importlib.reload(module)`** ✅
    c) import again
    d) run(module)

42. When importing a module, which method is faster on second run?
    a) Manual compile
    b) **Using .pyc in pycache** ✅
    c) Run in shell
    d) Loop import

43. What happens when you `import package.module`?
    a) Imports everything
    b) **Imports the submodule** ✅
    c) Errors if **init**.py missing
    d) Runs **main**

44. What's the difference between package and module?
    a) None
    b) Module is a folder
    c) **Module is a file; package is a folder** ✅
    d) Package has no imports

45. What does `from . import module` indicate?
    a) stdlib
    b) **Relative import inside a package** ✅
    c) Invalid syntax
    d) External install

46. What happens if `__init__.py` is missing in older Python versions?
    a) Works fine
    b) **Package won't be recognized** ✅
    c) Compiles to .pyc
    d) Imports as module

47. What is true about wildcard imports?
    a) Always safe
    b) **Can cause namespace pollution** ✅
    c) Best practice
    d) Only in CLI

48. Why use `import module as shortname`?
    a) Rename permanently
    b) **Convenience or readability** ✅
    c) Save memory

d) Required by Python

49. What happens if a module is not found in `sys.path`?
    a) Auto-create
    b) Skip silently
    c) **ImportError is raised** ✅
    d) Continue

50. What is the use of `__all__` in a module?
    a) Shows errors
    b) Imports everything
    c) **Defines what `from module import *` should import** ✅
    d) Stores globals

🔴 Hard Level (Q51–Q75)

51. What happens when you use this pattern in a module?

```
if __name__ == "__main__":

    main()
```

a) Always runs `main()`
b) **Runs `main()` only when the file is executed directly** ✅
c) Imports `main` from another module
d) Starts threading

52. What is `__file__` in a module?
    a) Path to Python executable
    b) Filename of the `__init__`
    c) **Path to the module's source file** ✅
    d) Variable storing compiled code

53. What will be output when executing a module containing:

```
print(__name__)
```

a) `main`
b) filename

c) `__init__`
d) **`__main__` if run directly, or module name if imported ✅**

54. What is required in a directory to be recognized as a package (pre-3.3)?
    a) setup.cfg
    b) **`__init__.py` file ✅**
    c) run.py
    d) pip install

55. When does Python compile `.py` to `.pyc`?
    a) Always
    b) Only manually
    c) **Automatically when a module is imported ✅**
    d) Never

56. Which Python module helps reload modules dynamically?
    a) import
    b) reload
    c) **importlib ✅**
    d) sys

57. What is the effect of circular imports?
    a) Infinite loop
    b) **May result in ImportError or partial initialization ✅**
    c) Always successful
    d) Caches infinitely

58. What happens if you import a file that contains a syntax error?
    a) Ignores error
    b) Loads partially
    c) **Raises a SyntaxError and fails ✅**
    d) Logs silently

59. How can you prevent certain functions from being imported using `from module import *`?
    a) Hide them
    b) Define in private class
    c) **Omit them from `__all__` list ✅**
    d) Use `nonlocal`

60. What does this code print?

```
import math

print(math.__name__)
```

a) main
b) **init**
c) **file**
d) **math** ✅

61. What is the benefit of `__pycache__` directory?
    a) Caches variables
    b) **Speeds up module loading with compiled bytecode** ✅
    c) Stores test output
    d) Stores logs

62. Which function shows detailed documentation from the terminal?
    a) `dir()`
    b) `info()`
    c) **`pydoc`** ✅
    d) `helpinfo()`

63. What will this output?

import math

print(math.__doc__)

a) File path
b) Function list
c) **Module docstring** ✅
d) **main**

64. Why avoid `from module import *` in real code?
    a) Increases performance
    b) Reduces import time
    c) **Pollutes the namespace unpredictably** ✅
    d) Prevents init

65. How can one structure a large application in Python?
    a) One .py file
    b) List all in sys.path
    c) **Use packages and sub-packages** ✅
    d) Use bash wrapper

66. What is the difference between `import module` and `from module import name`?
    a) None

b) import is private
c) **First imports entire module, second only specific object** ✅
d) Second is deprecated

67. Which file is created during module compilation for caching?
    a) .pkl
    b) .obj
    c) **.pyc** ✅
    d) .zip

68. Which function returns attributes of an object or module?
    a) attr()
    b) props()
    c) **dir()** ✅
    d) type()

69. Why might `from math import *` behave differently inside a function than globally?
    a) Performance
    b) Different scope rules
    c) **Wildcard imports not allowed inside functions in Python 3** ✅
    d) Overrides builtins

70. What is the proper way to expose selective APIs in a module?
    a) Use print()
    b) Comment others
    c) **Define `__all__`** ✅
    d) Only use `import`

71. What does `__package__` contain when a module is run as a script?
    a) Path
    b) True
    c) **None** ✅
    d) Current folder

72. What does this code do?

```
from .sub import mod
```

a) Absolute import
b) **Relative import from sibling subpackage** ✅
c) Error in Python 3
d) Does nothing

73. In Python 3.3+, how are namespace packages created?
   a) **init**.py
   b) sys.path
   c) **No need for init.py** ✅
   d) With zipfile

74. Which module allows you to inspect other modules?
   a) meta
   b) loader
   c) **inspect** ✅
   d) fileinput

75. Which scenario is likely to raise an ImportError?
   a) sys.path exists
   b) module exists
   c) **Module not found in sys.path** ✅
   d) Wildcard import

Great! Chapter 7 from the Python Tutorial is titled **"Input and Output"** and includes:

- Output formatting (f-strings, `format()`, `%` operator)
- File reading/writing (`open()`, `with`, `read()`, `write()`)
- File object methods (`seek()`, `tell()`)
- Structured data via `json`

Now generating **75 MCQs** (25 Easy, 25 Medium, 25 Hard) for:

📘 Chapter 7: Input and Output

🟢 Easy Level (Q1–Q25)

1. Which function is used to display output in Python?
   a) write()
   b) **print()** ✅
   c) out()
   d) echo()

2. What symbol is used in `f-strings`?
   a) %
   b) #
   c) @
   d) **f** ✅

3. Which method opens a file for reading in Python?
   a) openfile()
   b) readfile()
   c) **open()** ✅
   d) load()

4. What mode should you use to write to a file?
   a) 'r'
   b) **'w'** ✅
   c) 'x'
   d) 'rw'

5. What does the `r` mode mean in `open()`?
   a) Write
   b) Append
   c) Execute
   d) **Read** ✅

6. What does `f.read()` return?
   a) Boolean
   b) List
   c) **String of file contents** ✅
   d) Integer

7. What does `with open(...)` ensure?
   a) Delays read
   b) **Closes file automatically** ✅
   c) Clears file
   d) Skips errors

8. What does `f.write("hi")` return?
   a) None
   b) "hi"
   c) Error
   d) **Number of characters written** ✅

9. Which mode is used to append to a file?
   a) 'r'
   b) 'w'
   c) **'a'** ✅

d) 'b'

10. What does `'b'` in mode string signify?
    a) Break
    b) **Binary mode** ✅
    c) Boolean
    d) Big data

11. What is the default mode for `open()`?
    a) w
    b) a
    c) rb
    d) **r** ✅

12. Which method reads one line at a time?
    a) readall()
    b) readchar()
    c) **readline()** ✅
    d) line()

13. What will `f.readlines()` return?
    a) String
    b) Tuple
    c) **List of lines** ✅
    d) Dict

14. What does `f.close()` do?
    a) Clears file
    b) **Releases system resources** ✅
    c) Saves file
    d) Deletes file

15. What is the purpose of `seek()`?
    a) Search a file
    b) **Move cursor in file** ✅
    c) Clear buffer
    d) Encrypt file

16. Which character ends a line in text files on Windows?
    a) \n
    b) \t
    c) **\r\n** ✅
    d) \0

17. What is the function of `str.format()`?
    a) Read file
    b) Compile string

c) **Inject variables in string** ✅
d) Encrypt string

18. Which function gives current file position?
    a) pos()
    b) current()
    c) seek()
    d) **tell()** ✅

19. What is `f = open("test.txt", "rb")` doing?
    a) Text reading
    b) **Binary reading** ✅
    c) Appending
    d) Writing

20. What does `repr()` return?
    a) Nothing
    b) **String representation** ✅
    c) Hash
    d) List

21. What does `json.dumps(obj)` return?
    a) Binary
    b) Error
    c) **JSON string** ✅
    d) Pickle

22. What module handles structured data I/O?
    a) struct
    b) os
    c) **json** ✅
    d) sys

23. What is the output of:

'{:2.2f}'.format(3.14159)

a) 3.14 ✅
b) 3.1
c) 3
d) 3.142

24. Which string method pads zeros left-side?
    a) pad()
    b) **zfill()** ✅

c) fillzero()
d) zpad()

25. What happens when reading past EOF?
    a) Raises error
    b) Loops
    c) **Returns empty string** ✅
    d) Reads backwards

🟡 Medium Level (Q26–Q50)

26. What does this output?

```
f = open('file.txt', 'w')

f.write('Hello\nWorld')

f.close()
```

a) Error
b) **Writes two lines to file.txt** ✅
c) Writes one line
d) File remains empty

27. What is a safe way to open and work with files in Python?
    a) open()
    b) read()
    c) **with open(...) as f:** ✅
    d) os.open()

28. Which operator is used for legacy string formatting?
    a) +
    b) .
    c) **%** ✅
    d) @

29. How do you format a float to 3 decimal places using f-string?
    a) f"{x.3}"
    b) f"{:.3}"
    c) **f"{x:.3f}"** ✅
    d) format(x,3)

30. What does `f.tell()` return?
    a) File name
    b) **Current byte position in file** ✅
    c) File size
    d) Line number

31. What happens if you write to a file opened in read mode `'r'`?
    a) Writes normally
    b) Appends
    c) **Raises an IOError** ✅
    d) Overwrites

32. What is the correct syntax to read file lines as a list?
    a) `file.readlist()`
    b) **`file.readlines()`** ✅
    c) `file.read().list()`
    d) `file.readln()`

33. What does this code output?

```
print('{0} {1}'.format('Hello', 'World'))
```

a) World Hello
b) Hello
c) **Hello World** ✅
d) {0} {1}

34. Which file mode overwrites an existing file or creates a new one?
    a) a
    b) x
    c) **w** ✅
    d) r

35. Which of the following will correctly parse a JSON string?
    a) json.write()
    b) json.read()
    c) **json.loads()** ✅
    d) json.open()

36. What does the `seek(0)` command do?
    a) Closes file
    b) **Moves cursor to beginning of file** ✅
    c) Moves to end

d) Deletes content

37. What is the difference between `read()` and `readline()`?
    a) None
    b) **read() reads entire content, readline() reads one line** ✅
    c) readline() returns int
    d) read() is slower

38. What does the second argument in `open('file.txt', 'r', encoding='utf-8')` specify?
    a) Password
    b) Hash
    c) **Encoding format** ✅
    d) File size

39. Which module allows reading/writing JSON?
    a) struct
    b) io
    c) **json** ✅
    d) os

40. How do you remove newline characters from each line?
    a) trim()
    b) **line.strip()** ✅
    c) del(line)
    d) cut()

41. What does this expression do: `'{:>10}'.format('text')`?
    a) Error
    b) **Right-aligns 'text' in a field of width 10** ✅
    c) Left-aligns
    d) Pads left with zeros

42. What is the default encoding for open() in most systems?
    a) ASCII
    b) cp1252
    c) **UTF-8** ✅
    d) ISO-8859-1

43. Which function serializes a Python object to JSON-formatted string?
    a) json.read()
    b) json.loads()
    c) **json.dumps()** ✅
    d) json.to_json()

44. What will be printed?

```
f = open('test.txt', 'w')

print('hi', file=f)
```

a) **Writes 'hi' to file ✅**
b) Prints to console
c) Error
d) Writes to buffer

45. What is a context manager in file handling?
    a) Function
    b) Import
    c) **The `with` statement ✅**
    d) Error

46. What happens if you call `read()` twice on the same file object without seeking?
    a) Rewinds
    b) **Returns empty string on second call ✅**
    c) Overwrites file
    d) Reads again

47. How do you write a list of strings to a file?
    a) write(list)
    b) writelist()
    c) **writelines(list) ✅**
    d) list.write()

48. How do you pretty-print JSON with indentation?
    a) json.format()
    b) json.load(obj, indent=True)
    c) **json.dumps(obj, indent=4) ✅**
    d) None of the above

49. Which type of file is read using mode `'rb'`?
    a) Text
    b) **Binary ✅**
    c) JSON
    d) CSV

50. What does this code do?

```
json.dump(data, f)
```

a) Loads JSON
b) **Writes JSON data to file object f** ✅
c) Reads file
d) Serializes file

🔴 Hard Level (Q51–Q75)

51. What happens when you write to a file opened in `'a'` mode?
    a) **Appends content to the end without truncating** ✅
    b) Overwrites file
    c) Creates file only if it doesn't exist
    d) Raises an error

52. Which of the following modes opens a file for reading and writing?
    a) 'r+' ✅
    b) 'rw'
    c) 'w'
    d) 'a+'

53. What will the result of `seek(-1, 2)` be on a text file?
    a) Moves to start
    b) **Raises OSError (not allowed in text mode)** ✅
    c) Moves to end
    d) Moves to 1st character

54. What is the result of this code?

'{name} is {age}'.format(age=25, name='Alex')

a) age is Alex
b) **Alex is 25** ✅
c) 25 is Alex
d) Error

55. What does `open('file.txt', 'x')` do?
    a) Opens in binary
    b) **Creates file if it doesn't exist, else raises error** ✅
    c) Deletes file
    d) Opens for execute

56. How do you read a specific number of characters from a file?
    a) `read(num)` ✅
    b) `readchars(num)`

c) `slice(num)`
d) `f[num]`

57. What is a use of `tell()` in binary files?
    a) Closes file
    b) Shows encoding
    c) **Returns current byte offset** ✅
    d) Tells line number

58. What does this code output?

```
f = open('test.txt', 'w+')

f.write("Hello")

f.seek(0)

print(f.read())
```

a) Empty
b) Error
c) Hello ✅
d) 0

59. Why should `with open()` be preferred over `open()`?
    a) Saves memory
    b) Faster
    c) **Automatically closes file even on error** ✅
    d) Opens faster

60. Which of the following is most efficient for reading large files?
    a) read()
    b) **readline() inside a loop** ✅
    c) readlines()
    d) open().all()

61. What's the difference between `repr(obj)` and `str(obj)`?
    a) repr is string
    b) **repr is developer-focused, str is user-friendly** ✅
    c) Same output
    d) None

62. What is the output of:

f"{2*3:.2f}"

a) 6
b) 6.000
c) **6.00** ✅
d) Error

63. Which of the following correctly writes UTF-8 text?
    a) open('f', 'wb')
    b) open('f', 'w')
    c) **open('f', 'w', encoding='utf-8')** ✅
    d) write(utf-8)

64. What does `json.loads('[1, 2, 3]')` return?
    a) String
    b) **Python list [1, 2, 3]** ✅
    c) Set
    d) JSON object

65. What does this code output?

print("{0:>5}".format("x"))

a) x
b) **" x"** ✅
c) Error
d) x

66. What happens when you seek beyond EOF and write?
    a) Error
    b) Writes at end only
    c) **Creates null bytes in between** ✅
    d) Replaces EOF

67. What does the `zfill()` method do?

'42'.zfill(5)

a) '42'
b) **'00042'** ✅

c) '42000'
d) Error

68. Which is the safest way to write structured data to a file?
    a) repr()
    b) str()
    c) **json.dump()** ✅
    d) write()

69. Which of these raises an error on file already existing?
    a) 'w'
    b) 'r'
    c) **'x'** ✅
    d) 'a'

70. How does `writelines()` differ from `write()`?
    a) Writes strings
    b) **Takes a list of strings and writes them without newline** ✅
    c) Writes binary
    d) Auto-inserts newline

71. How to convert a dict to a JSON file?

with open('data.json', 'w') as f:

a) write(dict)
b) f.write(json.load(dict))
c) **json.dump(dict, f)** ✅
d) load(dict, f)

72. What's printed?

print(f"{'hello':^10}")

a) hello
b) **Centered "hello" with padding** ✅
c) Error
d) 'hello'

73. What's the purpose of file buffering?
    a) Faster disk writing
    b) Smaller files
    c) **Reduces I/O operations by caching data** ✅

d) Randomize access

74. Why is binary mode important when dealing with images?
    a) Faster
    b) **Preserves byte data exactly** ✅
    c) Compresses
    d) Converts to ASCII

75. What does this return?

json.loads('{"x":1}')

a) Error
b) String
c) **Dictionary {'x': 1}** ✅
d) List

Chapter 8 is titled **"Errors and Exceptions"** and includes topics such as:

- Basic exception handling (`try`, `except`, `else`, `finally`)
- Built-in exception types (`ZeroDivisionError`, `TypeError`, etc.)
- Raising exceptions
- User-defined exceptions
- Clean-up actions (`finally`, context management)

Now, I'll generate **75 MCQs** for:

📘 Chapter 8: Errors and Exceptions

🟢 Easy Level (Q1–Q25)

1. What keyword starts an exception handling block?
    a) handle
    b) error
    c) **try** ✅

d) catch

2. Which block catches exceptions?
   a) try
   b) finally
   c) **except** ✅
   d) raise

3. What does this raise?

```
x = 10 / 0
```

a) TypeError
b) **ZeroDivisionError** ✅
c) SyntaxError
d) None

4. Which block always runs, even if an exception occurs?
   a) except
   b) **finally** ✅
   c) try
   d) else

5. What is printed?

```
try:

    print(10 / 0)

except ZeroDivisionError:

    print("Error")
```

a) 0
b) Exception
c) **Error** ✅
d) Nothing

6. What exception is raised by `int("abc")`?
   a) KeyError
   b) **ValueError** ✅
   c) IndexError

d) TypeError

7. What is used to manually raise an exception?
   a) throw
   b) **raise** ✅
   c) emit
   d) panic

8. Which clause runs if no exception occurs?
   a) raise
   b) **else** ✅
   c) retry
   d) catch

9. What exception occurs on `lst[100]` if `lst` has 10 elements?
   a) **IndexError** ✅
   b) ValueError
   c) KeyError
   d) NameError

10. How do you handle multiple exception types?
    a) one except
    b) **Multiple except blocks** ✅
    c) use default
    d) catch all

11. What is the output?

```
try:

    print("Hello")

finally:

    print("World")
```

a) **Hello\nWorld** ✅
b) Hello
c) World
d) Error

12. What does this code raise?

```
print(undefined_variable)
```

a) SyntaxError
b) **NameError** ✅
c) TypeError
d) ValueError

13. Which of the following is not a built-in exception?
    a) TypeError
    b) AttributeError
    c) **WrongError** ✅
    d) ImportError

14. Which block is optional in try-except-finally?
    a) try
    b) except
    c) **finally** ✅
    d) All required

15. What happens after an exception is caught?
    a) Terminates
    b) **Continues with next line after handler** ✅
    c) Retries
    d) Restarts script

16. What is the parent of all exceptions?
    a) IOError
    b) Error
    c) **BaseException** ✅
    d) Exception

17. What is the use of `assert`?
    a) Input
    b) Log
    c) **Check condition and raise AssertionError if False** ✅
    d) Compare types

18. How do you define a custom exception?
    a) class X(object)
    b) **class X(Exception):** ✅
    c) def raise:
    d) subclass object

19. What's the result of:

```
try:
    raise ValueError("Invalid")
except:
    print("Handled")
```

a) Invalid
b) **Handled** ✅
c) None
d) Error

20. What does `except Exception as e:` do?
    a) Raises exception
    b) **Assigns exception to `e`** ✅
    c) Ignores `e`
    d) Logs it

21. Which exception type handles attribute access errors?
    a) NameError
    b) **AttributeError** ✅
    c) SyntaxError
    d) ValueError

22. What happens if `finally` has return statement?
    a) Skipped
    b) Uses `try`'s return
    c) **Overrides other returns** ✅
    d) Crashes

23. Which of these is correct syntax?

```
try:
    ...
except ValueError:
    ...
```

a) No try
b) **Correct** ✅
c) Use catch
d) Must add finally

24. What will this raise?

```
"2" + 2
```

a) None
b) **TypeError** ✅
c) ValueError
d) SyntaxError

25. Which block helps clean up resources?
   a) else
   b) try
   c) **finally** ✅
   d) catch

🟡 Medium Level (Q26–Q50)

26. What does this code print?

```
try:

    1 / 0

except ZeroDivisionError as e:

    print(type(e))
```

a) ZeroDivisionError
b) Exception
c) **<class 'ZeroDivisionError'>** ✅
d) <class 'Exception'>

27. Which exception occurs if you access a missing dictionary key?
   a) IndexError
   b) **KeyError** ✅
   c) ValueError
   d) AttributeError

28. What will this output?

```
try:

    pass

except:

    print("Error")

else:

    print("No error")
```

a) Error
b) Nothing
c) **No error** ✅
d) else

29. What does `raise Exception("fail")` do?
    a) Returns "fail"
    b) **Raises an exception with message 'fail'** ✅
    c) Fails silently
    d) Logs

30. Which block will run even after `raise` in `try`?
    a) except
    b) else
    c) **finally** ✅
    d) nothing

31. What happens when you `raise` without exception in an `except` block?
    a) SyntaxError
    b) **Re-raises current exception** ✅
    c) No effect
    d) Returns True

32. How can you raise a `TypeError` manually?
    a) throw TypeError
    b) Error(TypeError)
    c) **raise TypeError("message")** ✅
    d) typeerror("msg")

33. What will happen?

```
try:

    x = 1 / 0
```

```
except:

    pass

print(x)
```

a) 1
b) 0
c) **UnboundLocalError** ✅
d) None

34. What does this output?

```
try:

    raise IndexError

except (KeyError, IndexError):

    print("Handled")
```

a) KeyError
b) IndexError
c) **Handled** ✅
d) Nothing

35. When are `finally` blocks skipped?
    a) After `raise`
    b) After `return`
    c) **Never skipped** ✅
    d) After except

36. How to access the error message in an exception?
    a) e.msg
    b) **str(e)** ✅
    c) e.error
    d) error(e)

37. What is printed?

```
try:

    raise ValueError("Bad value")
```

```
except ValueError as e:

    print(e)
```

a) Bad
b) ValueError
c) **Bad value** ✅
d) 0

38. What happens if `try` raises, `except` doesn't match, and `finally` exists?
    a) nothing
    b) **finally runs, then error is re-raised** ✅
    c) finally skipped
    d) handled

39. How do you define a hierarchy of custom exceptions?
    a) All inherit object
    b) All inherit IOError
    c) **Inherit base class like MyError(Exception)** ✅
    d) Use decorators

40. What is output?

```
try:

    print("A")

    raise

except:

    print("B")
```

a) A
b) B
c) **SyntaxError** ✅
d) None

41. What is a best practice for exception handling?
    a) Catch all errors
    b) Avoid try
    c) **Catch specific exceptions** ✅
    d) Use exit()

42. What will this output?

```
try:
    raise Exception("E")
except Exception:
    raise
```

a) Nothing
b) Prints E
c) **Raises Exception** ✅
d) Pass

43. Which type is raised when slicing beyond length of a list?
   a) KeyError
   b) ValueError
   c) **No exception (Python allows it)** ✅
   d) TypeError

44. How do you define a custom exception with extra attributes?
   a) Pass tuple
   b) Override `__repr__()`
   c) **Define `__init__()` in class** ✅
   d) Inherit from object

45. Which statement is valid?

```
raise Exception from ValueError()
```

a) SyntaxError
b) **Chained exception** ✅
c) Invalid
d) Not allowed

46. Which function halts program execution immediately on error?
   a) pass
   b) except
   c) **raise** ✅
   d) stop()

47. What's the use of `assert` in testing?
    a) Skip
    b) **Ensure expected behavior** ✅
    c) log
    d) silence

48. What happens if `assert False`?
    a) Nothing
    b) **AssertionError raised** ✅
    c) True
    d) return False

49. What happens in this code?

```
try:

   1 / 0

finally:

   print("Finally")
```

a) Exception skipped
b) Finally skipped
c) **Finally prints, then ZeroDivisionError raised** ✅
d) Error silently ignored

50. What happens if exception is raised in except block?
    a) Ignored
    b) Logged
    c) **Propagates further** ✅
    d) Stops silently

🔴 Hard Level (Q51–Q75)

51. What is the result of this code?

```
try:

   1 / 0

except ZeroDivisionError:
    raise ValueError("Invalid math")
```

a) ZeroDivisionError
b) **ValueError with "Invalid math"** ✅
c) None
d) AssertionError

52. What does `raise from` do?

```
raise RuntimeError("New") from KeyError("Missing")
```

a) Merges exceptions
b) Silently ignores KeyError
c) **Creates an exception chain** ✅
d) Restarts execution

53. What is the output?

```
try:

    raise KeyError("x")

except Exception as e:

    print(repr(e))
```

a) KeyError
b) e
c) **KeyError('x')** ✅
d) x

54. What does this code result in?

```
try:

    assert 2 + 2 == 5

except AssertionError:

    print("Assertion failed")
```

a) 4
b) Error
c) **Assertion failed** ✅
d) Nothing

55. What is the correct way to define a custom exception?

```
class MyError(Exception): pass
```

a) Must use init
b) Not valid
c) **Correct** ✅
d) Needs BaseException

56. How to check exception chaining origin?

```
try:
    ...
except Exception as e:
    print(e.__cause__)
```

a) e.message
b) **e.cause** ✅
c) e.source
d) error.cause()

57. What is a potential risk of catching all exceptions like `except:`?
    a) Slower execution
    b) None
    c) **Hides real bugs** ✅
    d) Recompilation

58. What does `__context__` provide in exceptions?
    a) Logs
    b) **The previous unhandled exception** ✅
    c) None
    d) Stack

59. Which of the following keywords is optional in an exception block?
    a) try

b) except
c) **else** ✅
d) raise

60. What does the following output?

```
try:

   raise ValueError("A")

except Exception as e:

   raise TypeError("B") from e
```

a) Only TypeError
b) **TypeError with cause as ValueError** ✅
c) A
d) B

61. What is the best way to re-raise the current exception?
a) raise e
b) throw
c) **raise** ✅
d) pass

62. How do you suppress exceptions intentionally in `with` block?
a) try/except
b) context block
c) **Define `__exit__()` method that returns True** ✅
d) skip

63. What's the risk in writing:

```
except Exception:

   pass
```

a) Skips code
b) Logs all
c) **Silently hides exceptions** ✅
d) Re-raises

64. What is the correct use of `finally` in nested try blocks?
    a) Must be outside
    b) Cannot use
    c) **Valid; each try may have its own finally** ✅
    d) Once only

65. What is the output of this?

```
try:

    raise ValueError

finally:

    print("Cleanup")
```

a) Error
b) **Cleanup, then ValueError raised** ✅
c) None
d) Cleanup only

66. What happens when you `raise` outside of try block?
    a) Valid
    b) **Raises RuntimeError** ✅
    c) Pass
    d) Nothing

67. How can exception info be accessed in custom class?
    a) BaseError
    b) self.msg
    c) **self.args** ✅
    d) e.message

68. What does `try:... finally:` ensure?
    a) Skips code
    b) **Cleanup code is always executed** ✅
    c) Stops loop
    d) Fails silently

69. What does this mean?

```
raise Exception("E") from None
```

a) Raises default
b) **Suppresses context chaining** ✅
c) Raises None
d) Nothing

70. When is `except Exception as e:` preferred?
    a) Avoided
    b) **When you need the exception object for logging or inspection** ✅
    c) Never
    d) Deprecated

71. What is best practice for writing custom exception messages?
    a) Use **doc**
    b) print directly
    c) **Store message in `__str__` or `__init__`** ✅
    d) Avoid messages

72. What happens to return values when both `try` and `finally` have `return`?
    a) try wins
    b) Error
    c) **finally overrides try return** ✅
    d) Both return

73. What is printed?

```
try:

    print("A")

    raise

except:

    print("B")
```

a) A
b) **SyntaxError (missing exception in raise)** ✅
c) B
d) AB

74. Which of these is the base class of all exceptions?
    a) RuntimeError
    b) Exception
    c) **BaseException** ✅
    d) Throwable

75. How can an exception be logged properly?
    a) print()
    b) os.log()
    c) **use logging.exception()** ✅
    d) error()

Chapter 9 of the Python Tutorial PDF is titled **"Classes"** and covers:

- Namespaces and scope
- Class definition syntax
- Object instantiation
- Instance vs class variables
- Method definitions
- Special methods like `__init__`, `__str__`, `__len__`
- Inheritance
- Iterators and generators

Now I'll generate **75 MCQs** for:

📘 Chapter 9: Classes

🟢 Easy Level (Q1–Q25)

1. How do you define a class in Python?
    a) `def ClassName:`
    b) `class = ClassName()`
    c) **`class ClassName:`** ✅
    d) `new class ClassName:`

2. How do you instantiate an object from a class?
    a) `object = class()`
    b) **`object = ClassName()`** ✅
    c) `object = new ClassName()`

d) `create ClassName()`

3. Which method is called automatically during object creation?
   a) `__del__()`
   b) **`__init__()`** ✅
   c) `__str__()`
   d) `__call__()`

4. What does `self` represent in a method?
   a) Global object
   b) Parent class
   c) **Current instance of the class** ✅
   d) None

5. What keyword is used to create a subclass?
   a) inherits
   b) extends
   c) **class Derived(Base):** ✅
   d) super

6. How do you access an attribute of an object?
   a) object("attribute")
   b) get.attribute
   c) **object.attribute** ✅
   d) object->attribute

7. What is printed?

```
class A:
    def __init__(self):
        self.x = 10
a = A()
print(a.x)
```

a) 0
b) 1
c) **10** ✅
d) Error

8. What is a class variable?
   a) **Shared across all instances** ✅

b) Defined in `__init__()`
c) Unique to each object
d) Created using `self.`

9. What will `print(dir(obj))` show?
   a) Imports
   b) **Attributes and methods of the object** ✅
   c) Files
   d) Variables

10. What method provides string representation of an object?
    a) **print**()
    b) **str()** ✅
    c) **desc**()
    d) **init**()

11. What is printed?

```
class A:

    def __len__(self):

        return 5

print(len(A()))
```

a) 0
b) **5** ✅
c) None
d) Error

12. How do you define a class method?
    a) def method():
    b) **@classmethod** ✅
    c) @staticmethod
    d) def class():

13. Which decorator defines a static method?
    a) @init
    b) **@staticmethod** ✅
    c) @self
    d) @method

14. How many arguments does `__init__()` take (at minimum)?
    a) 0

b) 2
c) **1 (self)** ✅
d) It's optional

15. What's true about `__str__()`?
a) Called by `len()`
b) Returns an integer
c) **Returns a string for printing** ✅
d) Destroys object

16. Which of these is a correct class definition?

```
class Dog:

   pass
```

a) Correct ✅
b) Missing parentheses
c) Must have `__init__()`
d) Error

17. What does `isinstance(obj, Class)` return?
a) Type
b) Class
c) **True/False** ✅
d) Method

18. What does `__name__` refer to?
a) Object name
b) File name
c) **Current module name** ✅
d) Parent class

19. What is printed?

```
class A:

   y = 5

print(A.y)
```

a) 0
b) Error

c) **5** ✅
d) None

20. Which of the following is a class attribute?
    a) `self.x = 10`
    b) `x = 10` inside `__init__`
    c) **x = 10 (outside all methods)** ✅
    d) x()

21. How do you check attributes of an object?
    a) type(obj)
    b) repr(obj)
    c) **dir(obj)** ✅
    d) scan(obj)

22. Which of these methods defines an iterator class?
    a) **next**() only
    b) **iter**() only
    c) **iter() and next()** ✅
    d) **gen**()

23. What is the output?

```
class A:

    def __init__(self):

        self.x = 10

a = A()

print(hasattr(a, 'x'))
```

a) **True** ✅
b) False
c) x
d) None

24. How do you make a method private?
    a) Use `private` keyword
    b) **Prefix with double underscore: `__method`** ✅
    c) Use `protected`
    d) Use `hide()`

25. Which built-in function returns the class of an object?
    a) kind()
    b) type() ✅
    c) class()
    d) attr()

🟡 Medium Level (Q26–Q50)

26. What is the purpose of `super()` in Python classes?
    a) Create new objects
    b) **Access parent class methods** ✅
    c) Replace inheritance
    d) Add attributes

27. What is output of:

```
class A:
    def __init__(self):
        print("A")
class B(A):
    def __init__(self):
        super().__init__()
        print("B")
B()
```

a) A
b) B
c) **A\nB** ✅
d) Error

28. What is `__dict__` in a class instance?
    a) Class name
    b) **Dictionary of instance attributes** ✅
    c) Type info
    d) Method name

29. What does `@classmethod` receive as its first parameter?
    a) self

b) obj
c) **cls (class reference)** ✅
d) class

30. What does `@staticmethod` not access?
    a) arguments
    b) class
    c) **instance or class directly** ✅
    d) scope

31. What happens when `__str__()` is not defined in a class?
    a) Error
    b) Uses `__init__()`
    c) **Returns default object representation** ✅
    d) Crashes

32. Which method defines object's truth value in Boolean context?
    a) **int**()
    b) **str**()
    c) **bool()** ✅
    d) **eval**()

33. What is the result?

```
class A:

    pass

a = A()

a.x = 10

print(a.__dict__)
```

a) None
b) x
c) **{'x': 10}** ✅
d) {}

34. How to check if a class inherits from another class?
    a) `super()`
    b) **issubclass(Class1, Class2)** ✅
    c) type()
    d) getattr()

35. What is printed?

```
class A:

    def __len__(self):

        return 10

print(bool(A()))
```

a) True ✅
b) False
c) 10
d) Error

36. What happens if `__bool__()` returns False?
   a) **Object evaluates as False in Boolean context** ✅
   b) Causes error
   c) Becomes 0
   d) Converts to None

37. Which special method returns an iterator?
   a) **call**()
   b) **iter()** ✅
   c) **init**()
   d) **next**()

38. What is true about attributes declared as `ClassName.x`?
   a) Read-only
   b) Only for one instance
   c) **Shared among all instances** ✅
   d) Hidden

39. What happens when `__getattr__()` is defined?
   a) Adds method
   b) **Called when attribute is missing** ✅
   c) Overwrites all
   d) Deletes attr

40. Which function checks if a class has a specific method?
   a) check()
   b) **hasattr()** ✅
   c) call()
   d) getattr()

41. What is printed?

```
class A:

    def __call__(self):

        return "called"

a = A()

print(a())
```

a) Error
b) A
c) called ✅
d) **call**

42. How to customize string representation in print()?
    a) **print**
    b) **doc**
    c) **str** ✅
    d) toString

43. What is `__slots__` used for in classes?
    a) Read-only attributes
    b) **Limit instance attributes to save memory** ✅
    c) Replace **dict**
    d) None

44. What is true about:

```
class A:

    count = 0

    def __init__(self):

        A.count += 1
```

a) Each instance has own count
b) **count is shared across instances** ✅
c) count is private
d) Error

45. What does `__del__()` method do?
    a) Create instance
    b) Convert type
    c) **Called when object is deleted** ✅
    d) Reset

46. What is printed?

class A:

  def __init__(self): self.x = 5

  def __str__(self): return str(self.x)

a = A()

print(a)

a) a
b) 0
c) **5** ✅
d) Error

47. Which of the following overrides `+` operator?
    a) **plus**
    b) **add** ✅
    c) add()
    d) **sum**

48. What does `getattr(obj, 'attr', default)` do?
    a) **Returns value or default if not found** ✅
    b) Raises error
    c) Creates new attr
    d) Removes attr

49. How can you define a read-only property?
    a) readonly()
    b) constant
    c) **@property decorator** ✅
    d) **private**

50. What happens if `__next__()` in an iterator raises `StopIteration`?
    a) Continues
    b) Restarts
    c) **Ends iteration** ✅

d) Loops forever

🔴 Hard Level (Q51–Q75)

51. What is the purpose of using `__slots__` in a class?
    a) Define inheritance
    b) Block attributes
    c) **Optimize memory usage by preventing `__dict__` creation** ✅
    d) Enable multiple inheritance

52. What will this code output?

```
class A:

    def __new__(cls):

        print("Creating")

        return super().__new__(cls)

    def __init__(self):

        print("Initializing")

A()
```

a) Initializing
b) **Creating \n Initializing** ✅
c) Only Creating
d) Nothing

53. What happens when a class has both `__eq__` and `__hash__` methods?
    a) Error
    b) **Objects can be used as dict keys with custom equality** ✅
    c) Only `__eq__` works
    d) Only `__hash__` works

54. What does the `__repr__()` method return?
    a) Callable
    b) Nothing
    c) **Developer-friendly string representation of object** ✅
    d) Float

55. What does `__call__()` allow an object to behave like?
  a) Attribute
  b) Generator
  c) **Function** ✅
  d) Class

56. What is output of this code?

```
class A:

   def __len__(self): return 0

   def __bool__(self): return True

print(bool(A()))
```

a) False
b) 0
c) **True** ✅
d) Error

57. Which method allows a class instance to be used as an iterator?
  a) **next**
  b) **call**
  c) **iter and next** ✅
  d) **len**

58. What is returned by `type(obj)`?
  a) Function
  b) ID
  c) **Class of the object** ✅
  d) Instance

59. When does `__del__()` run?
  a) On **init**
  b) **When object is garbage collected** ✅
  c) When reassigned
  d) On print

60. What is output?

```
class A:

   def __add__(self, other): return "Sum"
```

print(A() + A())

a) Error
b) TypeError
c) **Sum** ✅
d) None

61. What is the effect of:

```
class A:
    def __eq__(self, other): return False
print(A() == A())
```

a) True
b) **False** ✅
c) Error
d) Exception

62. When are class attributes evaluated?
    a) Per object
    b) **Once when class is defined** ✅
    c) On `__init__()`
    d) After first use

63. What's printed?

```
class A:
    x = 5
a1 = A()
a2 = A()
a1.x = 10
print(a2.x)
```

a) 10
b) 0

c) **5** ✅
d) Error

64. What is the effect of overriding `__contains__()`?
    a) Customize `in` operator ✅
    b) Custom sort
    c) Changes `print()`
    d) No use

65. Which method controls `for item in object`?
    a) **next**
    b) **iter** ✅
    c) **getitem**
    d) loop()

66. What happens when `__getitem__()` is defined in a class?
    a) Creates list
    b) **Enables indexing (`obj[index]`)** ✅
    c) Defines str
    d) Disables slicing

67. How can a class support the `len()` function?
    a) **size**
    b) **count**
    c) **len** ✅
    d) **range**

68. Which built-in uses `__contains__()`?
    a) print()
    b) list()
    c) **`in` operator** ✅
    d) str()

69. What does `hasattr(obj, 'attr')` do?
    a) Sets attr
    b) Deletes attr
    c) **Checks if attribute exists** ✅
    d) Compares

70. How to ensure class can't be instantiated directly?
    a) @static
    b) raise
    c) **Use abstract base class (ABC)** ✅
    d) private

71. What will this code print?

```
class A:
    def __init__(self):
        self.x = 10
    def __repr__(self):
        return f"A({self.x})"
print(A())
```

a) 10
b) A
c) A(x)
d) **A(10)** ✅

72. Which class controls iteration?

```
for x in obj:
```

a) Callable
b) Indexable
c) **Iterable (with `__iter__`)** ✅
d) Generator only

73. What's the difference between `__str__` and `__repr__`?
   a) None
   b) str is for devs
   c) **str for users, repr for dev/debug** ✅
   d) repr is unused

74. What is true about Python class inheritance?
   a) Only single
   b) Only standard
   c) **Supports multiple inheritance** ✅
   d) Static only

75. What happens when a subclass does not override a method?
   a) Error
   b) Ignored
   c) **Inherits behavior from base class** ✅

d) Skips class creation

Chapter 10 is titled **"Brief Tour of the Standard Library"** and includes key modules like:

- `os`, `shutil`, `glob`
- `sys`, `argparse`, `re`, `math`, `random`, `statistics`
- `datetime`, `zlib`, `timeit`, `doctest`, `unittest`
- Internet access: `urllib`, `smtplib`
- Data interchange: `json`, `csv`, `xml.etree.ElementTree`, `sqlite3`, `email`

Now, let's generate **75 MCQs** for:

📘 Chapter 10: Brief Tour of the Standard Library

🟢 Easy Level (Q1–Q25)

1. Which module is used for file system operations in Python?
   a) sys
   b) glob
   c) **os** ✅
   d) datetime

2. What does `os.getcwd()` return?
   a) Python version
   b) Current file path
   c) **Current working directory** ✅
   d) Directory name

3. What does `shutil.copyfile()` do?
   a) Compresses a file
   b) **Copies file content to another file** ✅
   c) Reads files
   d) Backs up system

4. Which module helps to find filenames matching a pattern?
   a) os
   b) shutil
   c) **glob** ✅
   d) json

5. What does `glob.glob('*.py')` return?
   a) All folders
   b) Python interpreter
   c) **List of `.py` files** ✅
   d) Nothing

6. What does `sys.argv` contain?
   a) Module names
   b) **Command line arguments** ✅
   c) Environment variables
   d) Current path

7. How to print to stderr?

import sys

sys._____.write("error")

a) stdinput
b) **stderr** ✅
c) stdwrite
d) stdlog

8. What is `sys.exit()` used for?
   a) Clear screen
   b) Logout
   c) **Exit the program** ✅
   d) End a loop

9. Which module is best for handling command line arguments?
   a) os
   b) **argparse** ✅
   c) subprocess
   d) input

10. What does `re.findall(r'\bf[a-z]*', text)` return?
    a) Booleans
    b) Errors
    c) **Words starting with 'f'** ✅

d) None

11. What module supports regular expressions?
    a) str
    b) regex
    c) **re** ✅
    d) parser

12. What does `math.log(1024, 2)` return?
    a) 8
    b) 2
    c) **10.0** ✅
    d) log(1024)

13. What is the output of `random.choice([1, 2, 3])`?
    a) 1
    b) 2
    c) 3
    d) **Any one randomly from the list** ✅

14. What function gives random float between 0–1?
    a) randint()
    b) **random()** ✅
    c) rand()
    d) randfloat()

15. What module is used for statistical calculations?
    a) math
    b) **statistics** ✅
    c) random
    d) scipy

16. Which function gives the average value of a list?
    a) average()
    b) sum()
    c) **statistics.mean()** ✅
    d) stat()

17. What module is used to access URLs?
    a) smtplib
    b) http
    c) **urllib.request** ✅
    d) socket

18. What is `datetime.date.today()` used for?
    a) **Get current date** ✅
    b) Set date

   c) Print year
   d) Format text

19. What is `json.loads()` used for?
   a) Read XML
   b) Decode HTML
   c) **Convert JSON string to Python object** ✅
   d) Parse CSV

20. What is `csv.reader(file)` used for?
   a) Encode
   b) Read JSON
   c) **Read CSV file** ✅
   d) Log

21. What does `timeit()` measure?
   a) Date
   b) **Execution time** ✅
   c) Loops
   d) Memory

22. What does `doctest.testmod()` do?
   a) Run main()
   b) Import
   c) **Check docstring tests** ✅
   d) End module

23. What is `unittest.main()` used for?
   a) Exit test
   b) Run first test
   c) **Run all test cases** ✅
   d) Delete cache

24. Which module allows SQLite interaction?
   a) database
   b) sqlserver
   c) **sqlite3** ✅
   d) dbsql

25. What function is used in `os` to change directories?
   a) goto()
   b) cd()
   c) **chdir()** ✅
   d) move()

🟡 Medium Level (Q26–Q50)

26. What does `shutil.make_archive()` do?
    a) Moves directories
    b) **Creates zip/tar archive from files/folders** ✅
    c) Installs packages
    d) Encrypts data

27. What is the output of `re.split(r'\W+', 'hello, world!')`?
    a) hello world
    b) ['hello', ' ', 'world']
    c) **['hello', 'world', '']** ✅
    d) ['h', 'e', 'l', 'l', 'o']

28. What is `os.path.exists('file.txt')` used for?
    a) Create file
    b) Rename file
    c) **Check if file exists** ✅
    d) Remove file

29. Which module allows you to send emails?
    a) email
    b) inbox
    c) **smtplib** ✅
    d) socket

30. How to list all environment variables in Python?
    a) os.vars()
    b) **os.environ** ✅
    c) os.env()
    d) sys.envs

31. What does `argparse.ArgumentParser()` return?
    a) CLI input
    b) Command output
    c) **Parser object for command-line args** ✅
    d) Dictionary

32. What does `re.sub(r'foo', 'bar', 'foo123')` output?
    a) foo123
    b) barfoo
    c) **bar123** ✅
    d) 123bar

33. Which function in `math` returns factorial?
    a) math.fact()
    b) **math.factorial()** ✅

c) fact()
d) factorial()

34. What does `random.sample(range(100), 10)` return?
a) 100 numbers
b) **10 unique random numbers from range(100)** ✅
c) All numbers
d) Duplicate values

35. What is the output of:

```
import statistics

statistics.median([1, 3, 5])
```

a) 2
b) **3** ✅
c) 5
d) 1

36. What does `urllib.request.urlopen(url)` return?
a) HTML file
b) JSON
c) **Response object** ✅
d) Error

37. What is the format for creating a date object?
a) date("2024-01-01")
b) datetime("now")
c) **datetime.date(YYYY, MM, DD)** ✅
d) timestamp()

38. What does `json.dumps({"a": 1})` return?
a) {'a':1}
b) {"a": 1}
c) **String: '{"a": 1}'** ✅
d) List

39. What is `csv.writer(f).writerow(['a', 'b'])` used for?
a) Reads CSV
b) Converts to string
c) **Writes a row to a CSV file** ✅
d) Exports JSON

40. Which module supports XML parsing?
    a) xmltool
    b) **xml.etree.ElementTree** ✅
    c) treeparser
    d) xslt

41. What does `zlib.compress(b'data')` return?
    a) Encoded string
    b) Plain text
    c) **Compressed byte string** ✅
    d) JSON

42. What does `sqlite3.connect('db.sqlite3')` return?
    a) File object
    b) **Connection object to SQLite DB** ✅
    c) List
    d) Shell

43. How to read email headers in a structured format?
    a) smtplib.headers()
    b) **email.message_from_string()** ✅
    c) socket()
    d) header()

44. What is the output of:

import timeit

timeit.timeit('"-".join(str(n) for n in range(100))')

a) Code result
b) Time in ms
c) **Execution time in seconds** ✅
d) Memory

45. What does `unittest.TestCase` do?
    a) Starts program
    b) **Base class for creating test cases** ✅
    c) Runs CLI
    d) Logs output

46. Which command runs all `doctest` examples?
    a) test()
    b) runall()
    c) **doctest.testmod()** ✅

d) unittest.main()

47. What is `sys.platform` used for?
    a) Get system RAM
    b) **Identify current OS platform** ✅
    c) Python version
    d) File format

48. What does `os.system('ls')` do?
    a) Lists functions
    b) **Executes shell command `ls`** ✅
    c) Imports
    d) Logs

49. Which `random` method gives float in specific range?
    a) randint(a, b)
    b) randrange(a, b)
    c) **uniform(a, b)** ✅
    d) random()

50. What does `os.path.join('a', 'b')` return?
    a) ab
    b) a/b
    c) **Platform-independent path: 'a/b' or 'a\b'** ✅
    d) a:b

🔴 Hard Level (Q51–Q75)

51. What does this code do?

```
import shutil

shutil.copytree('src', 'dest')
```

a) Deletes src
b) Moves src
c) **Copies entire directory tree from `src` to `dest`** ✅
d) Zips directory

52. Which `re` method allows substituting matched patterns?
    a) search()
    b) findall()
    c) **sub()** ✅

d) match()

53. What is the result of:

import re

re.match(r'\d+', '123abc').group()

a) abc
b) **123** ✅
c) \d+
d) None

54. What is true about `argparse.ArgumentParser()`?
   a) Parses Python code
   b) **Provides CLI input parsing with help options** ✅
   c) Validates input type
   d) Compiles strings

55. How do you limit number of random digits in `random.randint()`?
   a) digits=3
   b) **Specify range, e.g. randint(100, 999)** ✅
   c) seed(3)
   d) randint()[:3]

56. What does `statistics.stdev([1, 2, 3])` compute?
   a) Mode
   b) **Standard deviation** ✅
   c) Median
   d) Average

57. What is output?

from datetime import datetime

print(datetime.now().isoformat())

a) Date only
b) Seconds
c) **ISO 8601 formatted timestamp** ✅
d) Object

58. What is used to decompress a zlib-compressed string?
    a) uncompress()
    b) **zlib.decompress()** ✅
    c) unpack()
    d) unzip()

59. What does `csv.DictReader(f)` return?
    a) List
    b) **Iterator of dicts per row** ✅
    c) String
    d) File object

60. What is the effect of `sqlite3.Row` as row_factory?
    a) List output
    b) JSON return
    c) **Row accessible by column name** ✅
    d) Raw bytes

61. How to handle HTTP error in `urllib`?
    a) pass
    b) catch()
    c) **try/except URLError/HTTPError** ✅
    d) request.handle()

62. What is `email.message.EmailMessage()` used for?
    a) Send via smtplib
    b) **Construct a well-formed email message** ✅
    c) Encrypt text
    d) Delete mail

63. What's the purpose of `xml.etree.ElementTree.parse()`?
    a) Format XML
    b) Delete tags
    c) **Read and parse XML file into tree** ✅
    d) Validate JSON

64. What is the role of `doctest` in development?
    a) Generate code
    b) Optimize
    c) **Test code embedded in docstrings** ✅
    d) Time functions

65. What does this test?

```
timeit.timeit("sum(range(100))", number=1000)
```

a) Memory usage
b) **Total time to run 1000 executions** ✅
c) Sum range
d) Line count

66. What does this do?

```
import unittest

class T(unittest.TestCase):

    def test_one(self):

        self.assertEqual(1 + 1, 2)
```

a) **Defines unit test case using assertion** ✅
b) Compiles code
c) Runs directly
d) Errors

67. How do you simulate command-line args in `argparse` during testing?
    a) sys.stdin
    b) input()
    c) **pass list to parse_args()** ✅
    d) mock()

68. What does `os.walk()` return?
    a) Single path
    b) **Generator yielding (dirpath, dirnames, filenames)** ✅
    c) Directory tree
    d) list

69. How does `urllib.parse.urlparse(url)` help?
    a) Downloads site
    b) **Splits URL into components** ✅
    c) Runs query
    d) Validates protocol

70. What does `email.utils.formataddr()` do?
    a) Encrypts
    b) Parses headers
    c) **Formats a tuple into a valid email address** ✅
    d) Sends email

71. What is `zlib.crc32(b'data')` used for?
   a) Compression
   b) **Checksum calculation** ✅
   c) Hashing
   d) Archiving

72. What does `os.stat('file.txt')` return?
   a) File open
   b) File error
   c) **Metadata like size, time, permissions** ✅
   d) Logs

73. Which method writes a dictionary row to CSV?
   a) writer.writerow()
   b) list.write()
   c) **csv.DictWriter().writerow()** ✅
   d) dict.write()

74. What is output?

import math

print(math.isclose(0.1 + 0.2, 0.3))

a) False
b) Error
c) **True** ✅
d) None

75. What does `__name__ == '__main__'` mean in `unittest`?
   a) Unused
   b) **Run tests when script is executed directly** ✅
   c) Check OS
   d) Ignore main

Great! Chapter 11 is titled **"Brief Tour of the Standard Library — Part II"** and includes advanced modules such as:

- `reprlib`, `pprint`, `textwrap`, `locale`, `string.Template`
- `struct`, `threading`, `queue`, `logging`, `weakref`
- `array`, `collections.deque`, `bisect`, `decimal`

Now generating **75 MCQs** across **all levels** for:

📘 Chapter 11: Brief Tour of the Standard Library — Part II

🟢 Easy Level (Q1–Q25)

1. What does `reprlib` module provide?
   a) Format date
   b) Read XML
   c) **Abbreviated `repr()` for long data** ✅
   d) Encrypt files

2. What is `pprint.pprint()` used for?
   a) Run programs
   b) **Pretty-print nested data** ✅
   c) Calculate math
   d) Build charts

3. What does `textwrap.fill()` do?
   a) Fill forms
   b) **Wraps text to fit screen width** ✅
   c) Adds padding
   d) Filters lines

4. Which module supports locale-specific formatting?
   a) text
   b) os
   c) **locale** ✅
   d) parser

5. What does `string.Template` use for variables?
   a) %
   b) `@`
   c) **$** ✅
   d) {}

6. Which method of Template avoids errors on missing placeholders?
   a) fill()
   b) **safe_substitute()** ✅
   c) format()

d) replace()

7. Which module deals with binary record layouts?
   a) zipfile
   b) struct
   c) **struct** ✅
   d) buffer

8. Which method compresses with struct?
   a) decode()
   b) **pack()** ✅
   c) format()
   d) load()

9. What does the threading module allow?
   a) Time travel
   b) **Run parallel tasks in threads** ✅
   c) Clear cache
   d) Increase RAM

10. What is `queue.Queue()` used for?
    a) Parsing
    b) Logging
    c) **Thread-safe task queue** ✅
    d) XML

11. What is `logging.warning()` used for?
    a) Kill process
    b) Print debug info
    c) **Log warning-level message** ✅
    d) Raise error

12. What are the default logging levels?
    a) **DEBUG, INFO, WARNING, ERROR, CRITICAL** ✅
    b) LOW, HIGH
    c) start, stop
    d) info, data

13. Which module handles weak references?
    a) garbage
    b) **weakref** ✅
    c) delref
    d) del()

14. What does `weakref.WeakValueDictionary()` do?
    a) Adds memory
    b) Stores permanent reference

c) **Holds object weakly—removed when not referenced** ✅
d) Deletes attributes

15. What does the `array` module provide?
a) Infinite lists
b) **Efficient typed arrays** ✅
c) Sets
d) Tables

16. What does `collections.deque` provide?
a) Sorting
b) **Fast appends and pops from both ends** ✅
c) Tree
d) Dictionary

17. What is `bisect.insort()` used for?
a) Search
b) Random
c) **Insert into sorted list maintaining order** ✅
d) Filter

18. Which module supports arbitrary-precision decimals?
a) math
b) float
c) **decimal** ✅
d) stats

19. What does `decimal.Decimal('1.0')` return?
a) 1.0
b) **Decimal('1.0')** ✅
c) Float
d) Exponential

20. Which module prints nicely indented structures?
a) pprint
b) **pprint** ✅
c) wrap
d) structure

21. Which of these methods packs binary values?
a) struct.map()
b) compress()
c) **struct.pack()** ✅
d) zip()

22. How do you run a thread using `threading` module?
a) run()

b) start()
c) **.start()** ✅
d) begin()

23. What happens when object in `WeakValueDictionary` is deleted?
a) Nothing
b) Error
c) **Key automatically removed** ✅
d) Overwritten

24. What is the default text-wrapping width for `textwrap`?
a) 50
b) 70
c) **70 characters** ✅
d) 100

25. What does `reprlib.repr()` help with?
a) Execute code
b) Parse JSON
c) **Limit repr() size** ✅
d) Save file

🟡 Medium Level (Q26–Q50)

26. What is the purpose of `string.Template.substitute()`?
a) Define variables
b) Replace `$`
c) **Substitute variables into a string using `$var` syntax** ✅
d) Compile template

27. What does this code print?

```
from collections import deque

d = deque([1, 2, 3])

d.appendleft(0)

print(d)
```

a) [0, 1, 2]
b) **deque([0, 1, 2, 3])** ✅
c) [1, 2, 3, 0]
d) Error

28. What is the default behavior of `queue.Queue()` if full?
  a) Discards old
  b) **Blocks until space is available** ✅
  c) Overwrites
  d) Crashes

29. What does `textwrap.dedent()` do?
  a) Wraps text
  b) Adds indentation
  c) **Removes common leading whitespace** ✅
  d) Prints docstring

30. Which method of `decimal` allows arithmetic with context precision?
  a) float()
  b) round()
  c) **getcontext().prec** ✅
  d) fix()

31. What is the use of `bisect.bisect()`?
  a) **Find insertion index in sorted list** ✅
  b) Delete item
  c) Slice data
  d) Filter values

32. What is the role of `struct.unpack()`?
  a) Return zip
  b) Encrypt
  c) **Convert binary back to Python data** ✅
  d) Save JSON

33. What's the main advantage of `array.array()` over lists?
  a) More methods
  b) Readable
  c) **More efficient memory usage for large numeric data** ✅
  d) Faster string parsing

34. What does `logging.basicConfig()` do?
  a) Start script
  b) **Configure default logging level and format** ✅
  c) Create file
  d) Install modules

35. In `threading.Thread(target=func)`, how is execution started?
  a) func()
  b) **.start()** ✅
  c) .run()

d) call()

36. What is the output of:

```
from decimal import Decimal, getcontext

getcontext().prec = 2

print(Decimal('1') / Decimal('3'))
```

a) 0.33 ✅
b) 0.3
c) 0.3333
d) Error

37. Which class allows template-style string substitution?
    a) Formatter
    b) string.String
    c) **string.Template** ✅
    d) f-string

38. What is a key benefit of using `logging` over `print()`?
    a) Slower
    b) Debug only
    c) **Structured, configurable logging across levels** ✅
    d) Color

39. What is returned by:

```
reprlib.repr(['x'] * 100)
```

a) Full list
b) **Abbreviated list representation** ✅
c) 100
d) None

40. What happens if `queue.get()` is called and queue is empty?
    a) Returns None
    b) Error
    c) **Blocks until item is available (by default)** ✅
    d) Times out

41. What is the result of:

```
from bisect import insort

a = [1, 3, 4]

insort(a, 2)

print(a)
```

a) [1, 2, 3, 4] ✅
b) [2, 1, 3, 4]
c) [1, 3, 4, 2]
d) Error

42. Which of these can use `deque.pop()`?
a) list only
b) dict
c) **deque** ✅
d) set

43. What does `weakref.ref(obj)` return?
a) Object
b) List
c) **Callable weak reference to object** ✅
d) id

44. What does `struct.calcsize('hhl')` return?
a) Header size
b) **Number of bytes the format would occupy** ✅
c) float
d) list

45. What does `pprint.PrettyPrinter(indent=4)` do?
a) Print log
b) Print bar
c) **Create pretty printer with indentation level 4** ✅
d) Formats email

46. What is true about objects stored in `WeakValueDictionary`?
a) They are strongly referenced
b) **They are garbage-collected when not used elsewhere** ✅
c) They raise TypeError
d) They persist forever

47. What does `array('i', [1, 2, 3])` create?
    a) List
    b) Set
    c) **Integer array** ✅
    d) Byte stream

48. Which module would you use for fixed-width binary formats?
    a) base64
    b) marshal
    c) **struct** ✅
    d) os

49. What method from `threading` returns current thread?
    a) thread()
    b) **current_thread()** ✅
    c) get()
    d) active()

50. What happens if `Template.substitute()` is called with missing key?
    a) Skips
    b) Returns None
    c) **Raises KeyError** ✅
    d) Returns empty string

🔴 Hard Level (Q51–Q75)

51. What does this code output?

```
from string import Template

t = Template('$who likes $what')

print(t.substitute(who='Alice', what='Python'))
```

a) $who likes $what
b) Error
c) **Alice likes Python** ✅
d) Python likes Alice

52. What happens if `Template.substitute()` misses a variable?
    a) Ignores
    b) Prints empty
    c) **Raises `KeyError`** ✅

d) Replaces with 'None'

53. What does `Template.safe_substitute()` do?
   a) Requires all args
   b) **Replaces available variables, skips missing** ✅
   c) Skips substitution
   d) Encrypts template

54. What is a practical use of `weakref.ref(obj)`?
   a) Lock object
   b) Copy object
   c) **Track object without preventing garbage collection** ✅
   d) Save reference

55. What is `array.array('d', [1.0, 2.0])`?
   a) List
   b) Integer array
   c) **Double-precision float array** ✅
   d) Set

56. What does `bisect.insort_left(lst, x)` ensure?
   a) Add right
   b) Replace
   c) **Insert `x` before any existing entries of same value** ✅
   d) Sort descending

57. Why is `textwrap.shorten(text, width=15)` useful?
   a) Crop lines
   b) **Truncate and add ellipsis without breaking words** ✅
   c) Encrypt
   d) Justify text

58. What is the benefit of `pprint.pformat()`?
   a) Writes to stderr
   b) **Returns a formatted string instead of printing it** ✅
   c) Logs error
   d) Replaces print()

59. How does `logging.getLogger()` differ from `basicConfig()`?
   a) Same
   b) **Returns a logger instance for modular logging** ✅
   c) Logs to console
   d) Changes OS config

60. What does `decimal.getcontext().prec = 10` affect?
   a) Decimal formatting

b) **Global precision for Decimal arithmetic** ✅
c) Binary storage
d) Rounding type

61. What is a use of `struct.pack('i', 123)`?
    a) Read file
    b) String formatting
    c) **Convert int into 4-byte binary representation** ✅
    d) Decode JSON

62. What happens if you store a large object only in `WeakValueDictionary`?
    a) Increases RAM
    b) **It is deleted when no other reference exists** ✅
    c) Raises MemoryError
    d) Persisted forever

63. Why might you use `deque` over list?
    a) More compact
    b) Sorts better
    c) **Faster append/pop operations from both ends** ✅
    d) Built-in sorting

64. What does `reprlib.repr()` limit?
    a) Code execution
    b) Data compression
    c) **Length and depth of printed representations** ✅
    d) Attribute access

65. What does `queue.put(item, block=False)` do if full?
    a) Waits
    b) **Raises queue.Full exception** ✅
    c) Discards
    d) Skips

66. What is a reason to use `decimal.Decimal` instead of float?
    a) Simpler
    b) **Avoid floating-point rounding issues** ✅
    c) Less RAM
    d) Faster

67. What will this code return?

from array import array

a = array('i', [1, 2, 3])

print(a[1])

a) 1
b) **2** ✅
c) Error
d) None

68. Which logging level will always be displayed unless filtering?
    a) WARNING
    b) DEBUG
    c) INFO
    d) **CRITICAL** ✅

69. Which module would you use for formatting financial values accurately?
    a) float
    b) money
    c) **decimal** ✅
    d) statistics

70. What does `deque.rotate(1)` do?
    a) Reverses
    b) Sorts
    c) **Shifts all elements one step right** ✅
    d) Deletes

71. How to create a queue with maximum size 5?
    a) queue(5)
    b) **queue.Queue(maxsize=5)** ✅
    c) maxqueue(5)
    d) Queue()

72. What is `logging.debug()` used for?
    a) Send email
    b) **Log developer-level debug messages** ✅
    c) Log errors only
    d) Raise exception

73. What happens when you call `q.task_done()` in `queue.Queue()`?
    a) Enqueues item
    b) **Indicates one queued task is complete** ✅
    c) Deletes queue
    d) Stops thread

74. Which `textwrap` method keeps lines inside a max width?
    a) indent()
    b) strip()

c) **wrap()** ✅
d) shorten()

75. Which type of references does `WeakValueDictionary` hold?
a) Strong
b) Callable
c) None
d) **Weak (non-owning)** ✅

Perfect! Chapter 12 is titled **"Virtual Environments and Packages"** and includes the following sub-sections:

- 12.1 Introduction
- 12.2 Creating Virtual Environments
- 12.3 Managing Packages with `pip`

Now generating **75 MCQs (Easy, Medium, Hard)** in one go for:

📘 Chapter 12: Virtual Environments and Packages

🟢 Easy Level (Q1–Q25)

1. What is the purpose of a virtual environment in Python?
a) Run system commands
b) **Isolate dependencies for a project** ✅
c) Upgrade Python
d) Format files

2. Which module is used to create virtual environments?
a) virtualenvtools
b) pyenv
c) **venv** ✅
d) pipenv

3. What is the command to create a virtual environment?
a) `pip init env`
b) **`python -m venv myenv`** ✅
c) `venv install`

d) `mkvirtualenv`

4. What does `source myenv/bin/activate` do?
   a) Deactivates venv
   b) **Activates the virtual environment on Unix/macOS** ✅
   c) Lists packages
   d) Compiles code

5. What is the command to activate a virtual environment on Windows?
   a) `source venv/bin/activate`
   b) `activate venv.sh`
   c) **`myenv\Scripts\activate.bat`** ✅
   d) `python -a`

6. What is the benefit of using a virtual environment?
   a) Faster performance
   b) Auto-formatting
   c) **Avoids conflicts between packages in different projects** ✅
   d) Auto-installation

7. How to check which Python interpreter is being used in a venv?
   a) `python -show`
   b) **`which python` or `where python`** ✅
   c) `pip show python`
   d) `python --list`

8. What does `pip install` do?
   a) Removes package
   b) **Installs Python package** ✅
   c) Runs project
   d) Activates venv

9. What file is commonly used to list dependencies?
   a) config.ini
   b) packages.py
   c) **requirements.txt** ✅
   d) main.json

10. Which command installs all dependencies from a requirements file?
   a) `pip get all`
   b) `install list.txt`
   c) **`pip install -r requirements.txt`** ✅
   d) `pip load`

11. What does `pip list` show?
    a) File paths
    b) Errors
    c) **Installed packages in current environment** ✅
    d) Python docs

12. What does `pip freeze` output?
    a) Binary files
    b) **Installed packages with exact versions** ✅
    c) Environment variables
    d) Logs

13. What is the role of `pip uninstall`?
    a) Run tests
    b) Debug modules
    c) **Remove installed packages** ✅
    d) Restart Python

14. What happens if you run Python outside a venv?
    a) **Global Python interpreter and packages are used** ✅
    b) Nothing runs
    c) Activates local venv
    d) Automatic isolation

15. What command shows detailed info about a package?
    a) `pip info`
    b) **`pip show`** ✅
    c) `pip inspect`
    d) `pip version`

16. What is `pip install requests` an example of?
    a) Uninstall
    b) Versioning
    c) **Installing a package** ✅
    d) Debugging

17. Where is the `site-packages` directory located for venvs?
    a) Globally
    b) System root
    c) **Inside the virtual environment folder** ✅
    d) Not available

18. Which format does `pip freeze` use?
    a) JSON
    b) CSV
    c) **Plain text with version pins (`==`)** ✅

d) YAML

19. How to deactivate a virtual environment?
    a) `exit`
    b) `python exit()`
    c) **`deactivate`** ✅
    d) `venv stop`

20. Which command upgrades a package?
    a) `pip add`
    b) **`pip install --upgrade <pkg>`** ✅
    c) `pip refresh`
    d) `pip redo`

21. What is the primary role of `venv`?
    a) Manage pip
    b) **Create isolated Python environments** ✅
    c) Debug packages
    d) Build GUIs

22. Can multiple virtual environments be created on the same machine?
    a) No
    b) Only one
    c) **Yes, as many as needed** ✅
    d) Only per user

23. What file inside venv shows Python version used?
    a) meta.ini
    b) config.json
    c) **pyvenv.cfg** ✅
    d) settings.py

24. Where are virtual environments typically stored in a project?
    a) root/
    b) **In a folder like `env/` or `venv/`** ✅
    c) `/usr/lib`
    d) system32

25. What does `python -m pip` ensure?
    a) Deactivates
    b) **Runs pip using the current Python interpreter** ✅
    c) Opens GUI
    d) Checks RAM

🟡 Medium Level (Q26–Q50)

26. What does `pip install .` do when run inside a package directory?
    a) Installs from PyPI
    b) Does nothing
    c) **Installs the current package locally (editable mode if setup.py is present)** ✅
    d) Freezes environment

27. What is the function of `--editable` or `-e` in pip?
    a) Uninstalls editable packages
    b) Pins version
    c) **Links the package for development without reinstalling** ✅
    d) Runs in test mode

28. What happens if two virtual environments install different versions of the same package?
    a) Conflict
    b) Overwrites global
    c) **No problem—they are isolated** ✅
    d) Requires venv reset

29. What does this command do?

```
python3 -m venv venv --prompt=project
```

a) Names Python prompt
b) **Creates venv and customizes shell prompt to `project`** ✅
c) Sets path
d) Activates global pip

30. How can you list outdated packages using pip?
    a) pip list
    b) **pip list --outdated** ✅
    c) pip info
    d) pip check

31. Which command checks for broken dependencies?
    a) pip scan
    b) pip freeze
    c) **pip check** ✅
    d) pip show

32. What is the best practice for sharing dependencies with a team?
    a) Email pip version
    b) **Use `requirements.txt` generated by `pip freeze`** ✅
    c) Upload env folder
    d) Share `venv` zip

33. What happens when running `pip install -r requirements.txt` in a new venv?
    a) Deletes old packages
    b) Opens GUI
    c) **Reinstalls all listed dependencies in that environment** ✅
    d) Upgrades Python

34. Why is `python -m pip` preferred over just `pip`?
    a) It's longer
    b) **Ensures the correct pip is used for the Python interpreter** ✅
    c) Prevents install
    d) Works offline

35. Which file can contain metadata for packages in a venv?
    a) pip.ini
    b) `requirements.lock`
    c) **METADATA** ✅
    d) install.cfg

36. What does the `Scripts/activate.bat` file do on Windows?
    a) Opens command prompt
    b) Installs dependencies
    c) **Activates the virtual environment for that terminal session** ✅
    d) Freezes packages

37. Which command upgrades pip itself inside a virtual environment?
    a) pip upgrade
    b) venv update
    c) **python -m pip install --upgrade pip** ✅
    d) pip update all

38. What will happen if `pip install` is run without venv activated?
    a) Error
    b) **Installs to global environment** ✅
    c) Blocks install
    d) Skips version

39. How to install a specific version of a package?
    a) pip install name@2
    b) **pip install name==2.0.1** ✅
    c) pip install name:2

d) pip add --version

40. Why is `pyvenv.cfg` important?
    a) It's a log file
    b) **Stores venv configuration and Python version path** ✅
    c) Starts venv
    d) Freezes modules

41. Where is the Python executable typically located in a Unix-based venv?
    a) /bin/python
    b) **venv/bin/python** ✅
    c) /usr/bin/python
    d) ~/.pyenv/python

42. What command removes a package from the environment?
    a) pip drop
    b) delete module
    c) **pip uninstall** ✅
    d) pip remove

43. Why might one recreate a venv after pulling from GitHub?
    a) Change shell
    b) Reinstall OS
    c) **To install fresh dependencies from requirements.txt** ✅
    d) Replace pip

44. What does `pip list --format=freeze` output?
    a) Sorted list
    b) CSV
    c) **Same format as `pip freeze`** ✅
    d) Binary

45. Can venvs be nested inside each other?
    a) Yes, always
    b) **Technically possible, but not recommended** ✅
    c) No
    d) Only in Docker

46. What does running `deactivate` do in venv context?
    a) Deletes pip
    b) Removes all packages
    c) **Restores the shell to global environment** ✅
    d) Logs out

47. Which pip command generates a lock-style output with hashes?
    a) pip freeze
    b) pip list --locked

c) **pip-compile (from `pip-tools`)** ✅
d) pip lock

48. How can you recreate the exact environment from `pip freeze`?
a) pip load requirements
b) **pip install -r requirements.txt** ✅
c) pip show all
d) venv --reset

49. What is the primary difference between `pip install` and `python setup.py install`?
a) No difference
b) **`pip` uses wheels and dependency resolution** ✅
c) setup.py is only for pip
d) pip compiles binary

50. What happens if a package listed in `requirements.txt` is not available?
a) Pip skips
b) Warning
c) **pip throws an error and exits** ✅
d) Uses default version

🔴 Hard Level (Q51–Q75)

51. What happens if you try to install a package that has C extensions but your system lacks a C compiler?
a) It skips compilation
b) Installs anyway
c) **Installation fails with a build error** ✅
d) Switches to global

52. What is the purpose of using `--no-cache-dir` with pip?
a) Caches install
b) Blocks dependencies
c) **Avoids using pip's local cache when installing** ✅
d) Installs faster

53. What does `pip install --user` do?
a) Global install
b) Only works in venv
c) **Installs packages in the user's home directory** ✅
d) Requires admin

54. How does pip resolve conflicting dependencies?
a) Chooses lowest version

b) Ignores conflicts
c) **Raises `ResolutionImpossible` or conflict warning** ✅
d) Deletes others

55. What is a virtual environment technically?
a) Docker
b) **A directory with its own Python binary and `site-packages`** ✅
c) Shared Python folder
d) Hidden script

56. Why should you avoid checking `venv/` into version control?
a) It's too small
b) Insecure
c) **It’s system-specific and can be easily recreated** ✅
d) Pip ignores it

57. What does `pip install -e .` require in the project directory?
a) pyproject.toml
b) **init**.py
c) **setup.py or pyproject.toml (depending on backend)** ✅
d) config.txt

58. What is a limitation of `pip freeze`?
a) Doesn't show versions
b) **Includes transitive (indirect) dependencies, not just direct** ✅
c) Only runs in root
d) Sorts randomly

59. Why is `pipx` used alongside venvs?
a) For machine learning
b) Debug mode
c) **To run Python CLI tools in isolated environments** ✅
d) For unit tests

60. What is `pyproject.toml` used for?
a) venv management
b) pip logs
c) **Declares build system and metadata for modern Python packaging** ✅
d) Only for poetry

61. What happens if you delete the venv folder without deactivating?
a) Error
b) **Your shell remains broken until you restart it** ✅
c) Global env resets
d) System hangs

62. Which pip command generates hashes for reproducible installs?
   a) pip secure
   b) **pip hash (or via `pip-compile --generate-hashes`)** ✅
   c) pip sign
   d) pip crypt

63. What is an advantage of `pip install --upgrade-strategy eager`?
   a) Slower
   b) **Upgrades all dependencies to latest possible** ✅
   c) Blocks pip
   d) Downgrades first

64. What's the difference between `pip install -r requirements.txt` and `pip-sync`?
   a) Same
   b) pip-sync installs only new
   c) **`pip-sync` removes unlisted packages, `pip install` does not** ✅
   d) pip-sync disables caching

65. What file signals that a directory is a Python package (pre-PEP 420)?
   a) **pycache**
   b) build.py
   c) **init.py** ✅
   d) venv.ini

66. What is an editable install used for?
   a) Production deployment
   b) **Allow local development with live changes without reinstalling** ✅
   c) Compression
   d) Testing only

67. What is `pipdeptree` used for?
   a) Zip venv
   b) Audit tools
   c) **Visualize dependency tree of installed packages** ✅
   d) Install tests

68. How can one share both code and dependencies effectively?
   a) Copy env/
   b) Email zip
   c) **Package code + `requirements.txt` or `pyproject.toml`** ✅
   d) Share binaries

69. How do you specify an environment marker in requirements.txt?
   a) pip marker
   b) --os
   c) **platform_system=="Windows"** ✅

d) version=>

70. What does this requirement mean?

```
package_name ; python_version < "3.10"
```

a) Version pin
b) **Install only if Python < 3.10** ✅
c) Always installs
d) Skips Linux

71. Why are virtual environments preferred in CI/CD pipelines?
a) Faster execution
b) **Reproducibility and environment isolation** ✅
c) No need for testing
d) Replaces Git

72. What is a drawback of using global pip installations?
a) Faster
b) Simpler logs
c) **Risk of version conflicts between unrelated projects** ✅
d) Easier to debug

73. Which tool helps manage multiple Python environments and versions?
a) venv
b) pytools
c) **pyenv** ✅
d) pipx

74. How do `pipenv` and `venv` differ?
a) pipenv is only for Linux
b) venv supports poetry
c) **pipenv integrates venv creation and dependency management** ✅
d) pipenv is deprecated

75. What is the best practice before committing `requirements.txt`?
a) Remove comments
b) **Regenerate with `pip freeze` after install or updates** ✅
c) Rename it
d) Minify it

Chapter 13 is titled **"What Now?"**, and it's a short concluding chapter that provides learning resources, community links, and guidance on continuing the Python journey.

Now generating 75 MCQs (Easy, Medium, Hard) for:

📘 Chapter 13: What Now?

🟢 Easy Level (Q1–Q25)

1. What is the primary focus of Chapter 13 "What Now?"
   a) Functions
   b) Classes
   c) **Further learning resources and community support** ✅
   d) Installation

2. Which website hosts Python's official documentation?
   a) pypi.com
   b) **docs.python.org** ✅
   c) pythonhub.io
   d) pydocs.dev

3. What is the purpose of the Python Package Index (PyPI)?
   a) Python compiler
   b) Debug tool
   c) **Repository for Python packages** ✅
   d) Logging website

4. Which resource is referred to as the “Cheese Shop”?
   a) piptools.org
   b) **PyPI** ✅
   c) Python.org
   d) pythonclub

5. What is https://www.python.org used for?
   a) Hosting Git repos
   b) Paid courses
   c) **Python downloads and resources** ✅
   d) File storage

6. Where can you find Python video tutorials and conference recordings?
   a) PyTV
   b) PyVids.io
   c) **http://www.pyvideo.org** ✅

d) PyFilms

7. What is the “Python Cookbook”?
   a) Food recipes
   b) IDE
   c) **Collection of code examples and scripts** ✅
   d) Framework

8. What type of content does the Python FAQ contain?
   a) Source code
   b) Paid tutorials
   c) **Answers to common questions about Python** ✅
   d) Ads

9. What is the mailing list for Python discussions?
   a) pydiscuss.org
   b) **python-list@python.org** ✅
   c) help@python.io
   d) dev@pythonhub

10. Where can users post questions and suggestions about Python?
    a) pip.org
    b) **comp.lang.python newsgroup** ✅
    c) code.python.com
    d) python-post.com

11. Which website contains books and scripts contributed by users?
    a) PyPI
    b) PythonMail
    c) **code.activestate.com/recipes/langs/python** ✅
    d) PyBooks

12. What should you consult before posting a question to the Python mailing list?
    a) Newsletter
    b) **FAQ page** ✅
    c) Book
    d) GitHub issues

13. What does Python’s community value in questions?
    a) Humor
    b) **Clarity and research effort** ✅
    c) Long messages
    d) Automation

14. Who contributes to the Python documentation?
    a) AI tools
    b) Paid employees only
    c) **Python contributors and users** ✅

d) Developers only

15. Is Python documentation freely accessible?
    a) Paid tier needed
    b) No
    c) **Yes** ✅
    d) Trial only

16. What can you find in Python's reference index?
    a) Games
    b) **Formal definition of syntax and semantics** ✅
    c) Random articles
    d) Code samples

17. What is the purpose of the Python Glossary?
    a) Build HTML
    b) **Explain Python terminology** ✅
    c) Log definitions
    d) Print output

18. What's encouraged after finishing the Python Tutorial?
    a) Build a website
    b) Contribute to C++
    c) **Explore libraries and advanced topics** ✅
    d) Only review

19. What is one of the best ways to continue learning Python?
    a) **Start building projects** ✅
    b) Watch movies
    c) Use spreadsheets
    d) Avoid docs

20. Which group is most active in Python Q&A?
    a) Twitter
    b) Reddit
    c) **comp.lang.python** ✅
    d) pip-group

21. What should you use to search for Python packages?
    a) Stack Overflow
    b) **pypi.org** ✅
    c) PythonBot
    d) pipsearch

22. What kind of license does Python use?
    a) MIT
    b) Proprietary
    c) **Open Source** ✅

d) Oracle

23. Which resource lists common Python modules and tools?
    a) **init**.py
    b) sys.modules
    c) **python.org "Modules" section** ✅
    d) os.py

24. What is the best use of the Python mailing list?
    a) Python job search
    b) GitHub support
    c) **Ask language and library questions** ✅
    d) Share memes

25. What kind of questions should you avoid posting in the mailing list?
    a) Beginner
    b) Library usage
    c) **Questions already answered in the FAQ** ✅
    d) Syntax

🟡 Medium Level (Q26–Q50)

26. What is a recommended next step after completing the tutorial?
    a) Stop using Python
    b) **Explore the standard library and community projects** ✅
    c) Switch to Java
    d) Learn HTML only

27. Which resource offers Python tips and how-tos?
    a) PyChat
    b) **ActiveState's Python recipes** ✅
    c) CPython Dev Blog
    d) Python AI Hub

28. What is the benefit of subscribing to `python-list@python.org`?
    a) Get discount coupons
    b) **Receive Python Q&A and discussions via email** ✅
    c) Watch movies
    d) Auto-install packages

29. What should be included when posting to a Python mailing list?
    a) Meme
    b) Binary file
    c) **Detailed explanation and minimal example** ✅
    d) URL only

30. What is one reason the Python FAQ is valuable?
    a) Provides tutorials
    b) **Answers common errors and misconceptions** ✅
    c) Updates packages
    d) Installs pip

31. Which format is used to submit documentation patches to Python?
    a) .txt
    b) CSV
    c) **reStructuredText (`.rst`)** ✅
    d) HTML

32. What is the style guide for Python documentation?
    a) DocuRules
    b) XML DTD
    c) **PEP 257** ✅
    d) ISO 9001

33. What does the comp.lang.python newsgroup mirror?
    a) Twitter
    b) GitHub
    c) **python-list@python.org** ✅
    d) IRC

34. What etiquette should be followed on Python's mailing list?
    a) All caps
    b) **Be concise, respectful, and clear** ✅
    c) Use emojis
    d) Cross-post to 10 groups

35. Which resource contains up-to-date Python changes and news?
    a) pip.org
    b) **https://www.python.org/blogs/** ✅
    c) stackoverflow.dev
    d) pychangelog.net

36. What is the Python Software Foundation (PSF)?
    a) Software tool
    b) Compiler
    c) **Non-profit organization managing Python development** ✅
    d) Debug tool

37. How can one contribute to Python documentation?
    a) Send email to Guido
    b) **Use GitHub and follow dev guide** ✅
    c) Push to master

d) Submit CSV

38. What section of Python Docs provides extensive standard library info?
    a) PEP list
    b) **Library Reference** ✅
    c) Tutorial only
    d) Install guide

39. What is one way to improve Python skills long term?
    a) Watch memes
    b) **Contribute to open-source Python projects** ✅
    c) Avoid writing code
    d) Only read books

40. Which search engine is commonly used to search for Python errors?
    a) AskJeeves
    b) Wolfram
    c) **Google (with Stack Overflow links)** ✅
    d) PythonBot

41. What can be found at http://code.activestate.com/recipes/langs/python/?
    a) Job listings
    b) **User-contributed Python code examples** ✅
    c) Full books
    d) Syntax only

42. What is the advantage of using IRC or Discord Python communities?
    a) Paid help
    b) **Real-time help and feedback** ✅
    c) Logs only
    d) Just news

43. What is a “cookbook” in programming communities?
    a) Recipe for snacks
    b) **A collection of ready-to-use code snippets** ✅
    c) Only for chefs
    d) Syntax checker

44. Why is PyPI nicknamed the “Cheese Shop”?
    a) It stores dairy data
    b) **It’s a Monty Python reference** ✅
    c) It’s about cheese algorithms
    d) It was originally named Cheese.py

45. What does Python's glossary contain?
    a) Code only
    b) **Definitions of Python-specific terms and jargon** ✅
    c) Log history

d) Class listings

46. What is a “minimal reproducible example” in Python help requests?
    a) Test case
    b) Code editor
    c) **Simplified code that replicates the issue clearly** ✅
    d) A short program

47. How is the Python documentation built and maintained?
    a) Manually updated weekly
    b) Paid contributors
    c) **Built automatically from reStructuredText using Sphinx** ✅
    d) Exported from Word

48. Where can new contributors find guidance on improving docs?
    a) **init**.py
    b) Python IRC
    c) **https://devguide.python.org** ✅
    d) PyPI front page

49. What is `https://planetpython.org`?
    a) Hosting site
    b) **Blog aggregator for Python news** ✅
    c) Module repo
    d) Game site

50. Which mailing list is used for discussing core development?
    a) pip-dev@
    b) python-projects@
    c) **python-dev@python.org** ✅
    d) python-newbies@

🔴 Hard Level (Q51–Q75)

51. What is the key benefit of reading Python’s PEPs after the tutorial?
    a) Entertainment
    b) **Understanding language design decisions** ✅
    c) Upgrading pip
    d) Creating GUIs

52. Why should users not cross-post the same question to multiple forums?
    a) It reduces visibility
    b) **It clutters discussion and wastes others’ time** ✅
    c) Makes Python crash
    d) It is filtered automatically

53. How can you help improve Python even as a beginner?
    a) Hack CPython
    b) **Report bugs or suggest doc improvements** ✅
    c) Build compilers
    d) Modify core

54. What is the difference between the Python Tutorial and the Library Reference?
    a) One is for Java
    b) Same content
    c) **Tutorial is hands-on, Library Reference is comprehensive API doc** ✅
    d) One is outdated

55. Which guideline ensures clarity and formatting in Python documentation contributions?
    a) PEP 5
    b) PEP 20
    c) **PEP 257 (Docstring Conventions)** ✅
    d) PEP 101

56. Which resource explains Python's syntax formally?
    a) Cookbook
    b) **Language Reference** ✅
    c) FAQ
    d) Glossary

57. Why is it better to ask for help on `python-list@python.org` than personal email?
    a) Faster
    b) Private
    c) **Enables community discussion and archives answers** ✅
    d) Avoids replies

58. Which module documents all standard built-ins?
    a) stdlib
    b) help.py
    c) **`__builtin__` (Python 2) or `builtins` (Python 3)** ✅
    d) runtime

59. What is the typical tone of official Python documentation?
    a) Casual
    b) Slang-heavy
    c) **Formal but accessible** ✅
    d) In-code comments

60. Why should you not rely solely on tutorials for learning Python?
    a) Tutorials are fake
    b) **They don't cover deep details or edge cases** ✅
    c) They break pip

d) They cost money

61. What is `https://devguide.python.org/docquality/` for?
  a) File cleaner
  b) Compiler settings
  c) **Guidelines to improve Python documentation** ✅
  d) GitHub issues

62. Which format is Python documentation written in?
  a) Markdown
  b) **reStructuredText (reST)** ✅
  c) HTML
  d) LaTeX

63. What happens when a PEP is accepted?
  a) Added to FAQ
  b) **It may lead to language or standard library changes** ✅
  c) Ignored
  d) Made a blog

64. How is the glossary different from the FAQ?
  a) It's visual
  b) Has source code
  c) **Defines terms, while FAQ answers common user questions** ✅
  d) They are the same

65. Why are FAQs important for contributors?
  a) Skip reading
  b) **Avoid redundant questions and understand historical context** ✅
  c) Check updates
  d) Install pip

66. What is `https://discuss.python.org` used for?
  a) Installing Python
  b) **Forum for Python development, help, and ideas** ✅
  c) Errors only
  d) Python errors

67. What is the advantage of asking questions in public forums?
  a) Higher costs
  b) Faster uninstall
  c) **Answers benefit others with the same issue** ✅
  d) Removes bugs

68. How can one contribute code to Python?
  a) Email files
  b) **Fork the repo on GitHub and submit a pull request** ✅

c) Use pip
d) Modify `.pyc`

69. What is the mailing list used for discussion of proposed Python changes?
a) python-dev
b) py-core
c) **python-ideas@python.org** ✅
d) pip-changes

70. Why should posts to Python mailing lists be plain text?
a) Easier to read
b) Prevent code execution
c) **For compatibility across platforms and tools** ✅
d) Markdown is blocked

71. What tool generates Python's official documentation site?
a) Jupyter
b) Doxygen
c) **Sphinx** ✅
d) Pandoc

72. What is Planet Python?
a) Compiler
b) **Community blog aggregator for Python posts** ✅
c) PEP linter
d) Package manager

73. What does the term "community-driven language" imply for Python?
a) Has bugs
b) **Users contribute to its evolution** ✅
c) Has ads
d) Closed source

74. What makes Python documentation globally useful?
a) Only online
b) Ads support
c) **Free access, open contributions, and localization** ✅
d) Flash format

75. What is the final recommendation in the Python Tutorial's last chapter?
a) Quit coding
b) Learn C++
c) **Explore Python further and engage with the community** ✅
d) Pay for Python license

Great! Continuing with:

📘 Chapter 14: Interactive Input Editing and History Substitution

---

🟢 Easy Level (Q1–Q25)

1. What is the main focus of Chapter 14?
   a) Functions
   b) Data types
   c) **Interactive command-line editing and history features** ✅
   d) Web frameworks

2. Which module enables interactive editing features in Python?
   a) editor
   b) pyinput
   c) **readline** ✅
   d) promptlib

3. What does the `readline` module support?
   a) File reading
   b) **Line editing and command history** ✅
   c) Audio input
   d) GUI design

4. What key enables command-line history in most Unix Python shells?
   a) Esc
   b) **Up arrow key** ✅
   c) Ctrl + Q
   d) F5

5. How can you navigate through previous Python commands interactively?
   a) Alt+P
   b) **Arrow keys (↑ / ↓)** ✅
   c) Ctrl+Z
   d) Tab

6. What does the Tab key provide in interactive mode (on supported systems)?
   a) Breakpoint
   b) Restart
   c) **Auto-completion of identifiers and keywords** ✅
   d) Exit

7. What is `.python_history`?
   a) Package manager
   b) **File storing history of Python shell commands** ✅
   c) HTML logs
   d) GUI theme file

8. Where is `.python_history` typically stored?
   a) Desktop
   b) **User's home directory** ✅
   c) Python site-packages
   d) Logs folder

9. Which built-in function lets you check previous commands in REPL?
   a) last()
   b) hist()
   c) **readline.get_history_item()** ✅
   d) input()

10. What does `readline.get_current_history_length()` return?
    a) Memory size
    b) Filename
    c) **Number of items in history** ✅
    d) Error count

11. What platform does `readline` primarily support?
    a) **Unix-like systems** ✅
    b) Windows
    c) Android
    d) Jupyter

12. What is an alternative to `readline` on Windows systems?
    a) Bash
    b) **pyreadline** ✅
    c) pipline
    d) zshell

13. What happens when you press Ctrl+R in the Python shell (if supported)?
    a) Refresh
    b) **Reverse search through history** ✅
    c) Save file
    d) Run setup

14. Which Python shell typically supports tab completion by default?
    a) Default shell
    b) **IPython** ✅
    c) Bash

d) Windows Command Prompt

15. What Python feature helps you recall previous inputs?
    a) history()
    b) input()
    c) **Command history buffer** ✅
    d) old()

16. What is the purpose of command history?
    a) Save variables
    b) **Reuse previous commands without retyping** ✅
    c) Exit shell
    d) Lock script

17. What is one advantage of using IPython over the default interpreter?
    a) GUI
    b) Code size
    c) **Enhanced editing, auto-completion, and history** ✅
    d) RAM usage

18. What is the default shortcut to autocomplete a variable in Python shell?
    a) Shift
    b) **Tab** ✅
    c) Enter
    d) Ctrl+A

19. What does `readline.clear_history()` do?
    a) Restarts shell
    b) Logs error
    c) **Clears stored command history** ✅
    d) Breaks loop

20. What is required to enable `readline` features in Python?
    a) pip install autoedit
    b) Custom keyboard
    c) **System support and `import readline`** ✅
    d) VS Code

21. Can Python command-line editing features be customized?
    a) No
    b) Partially
    c) **Yes, through key bindings and readline settings** ✅
    d) Only on Windows

22. Which keybinding is common for deleting a whole line in history?
    a) Shift+Del
    b) Esc

c) **Ctrl+U** ✅
d) Alt+D

23. What's the purpose of an interactive shell in Python?
a) Build packages
b) **Quickly test and experiment with Python code** ✅
c) Install updates
d) Configure hardware

24. Which alternative Python shell supports rich outputs and features?
a) Bash
b) Notepad
c) **IPython** ✅
d) Eclipse

25. What happens to your shell history after exiting Python REPL?
a) Cleared
b) **Saved in `.python_history` (if supported)** ✅
c) Sent to cloud
d) Compiled to .pyc

🟡 Medium Level (Q26–Q50)

26. What's the result of this snippet (on systems supporting `readline`)?

import readline

readline.get_history_item(1)

a) Error
b) **Returns the first command in history** ✅
c) Clears history
d) Disables shell

27. How can `.python_history` be viewed manually?
a) It's encrypted
b) Through Python only
c) **By opening it in a text editor** ✅
d) With pip

28. What's the benefit of `readline.parse_and_bind("tab: complete")`?
a) Installs modules

b) Breaks interpreter
c) **Enables tab completion in supported terminals** ✅
d) Edits functions

29. Why is the `readline` module less useful on Windows?
a) Python blocks it
b) **Native Windows shell doesn't support GNU readline** ✅
c) Requires GUI
d) Python doesn't install it

30. What does `readline.write_history_file(filename)` do?
a) Installs pip
b) **Saves current history to a specific file** ✅
c) Opens IDE
d) Removes functions

31. What is required to reverse search through history?
a) Ctrl+P
b) Esc
c) **Ctrl+R** ✅
d) Shift+R

32. What happens when using `readline.read_history_file()`?
a) Opens new shell
b) **Loads previously saved command history** ✅
c) Clears REPL
d) Deletes `.python_history`

33. What does `readline.set_history_length(10)` do?
a) Saves last 10 variables
b) Locks terminal
c) **Limits history buffer to 10 entries** ✅
d) Kills process

34. How does IPython differ from the standard Python shell?
a) Smaller memory
b) **Supports advanced features like history, magic commands, and rich output** ✅
c) Less readable
d) Fewer modules

35. How can users make `readline` features permanent?
a) CLI only
b) **Use a `.pythonrc.py` startup script** ✅
c) Add to .bashrc

d) Place in pip.ini

36. What does `readline.get_history_item(-1)` return?
    a) Last variable
    b) Error
    c) **Returns last command (in some implementations)** ✅
    d) Clears buffer

37. Why does Python use GNU readline on Unix?
    a) Compatibility
    b) **For enhanced command-line editing and history** ✅
    c) Network support
    d) Compilation

38. What does `readline.get_line_buffer()` do during an input session?
    a) Clears RAM
    b) **Returns the current input buffer** ✅
    c) Rebuilds shell
    d) Prints version

39. Which function appends a command to the current history list?
    a) read_last()
    b) push_history()
    c) **readline.add_history("command")** ✅
    d) insert()

40. How can Python history persist across sessions?
    a) Save manually
    b) **Use `readline.write_history_file()` on exit and `read_history_file()` on start** ✅
    c) Use pip cache
    d) It's automatic

41. What can be a disadvantage of large command history?
    a) Causes syntax errors
    b) **Increased memory usage** ✅
    c) Slower compilation
    d) Command lock

42. What does this line do?

readline.clear_history()

a) Erases Python variables
b) **Removes all stored input history in the session** ✅

c) Clears memory cache
d) Logs session

43. Why might tab-completion not work in some Python shells?
    a) Typo
    b) Broken code
    c) **Shell doesn't support readline** ✅
    d) Wrong Python version

44. Which platform might require installing `pyreadline` separately for readline support?
    a) Linux
    b) Unix
    c) **Windows** ✅
    d) MacOS

45. What does `readline.set_completer()` do?
    a) Loads shell
    b) **Defines a custom tab-completion function** ✅
    c) Saves history
    d) Prints keywords

46. How does `readline.get_history_length()` differ from `get_current_history_length()`?
    a) They are aliases
    b) **First returns file length, second returns session length** ✅
    c) One resets data
    d) One prints

47. Which startup hook can be used to import `readline` automatically?
    a) setup.py
    b) pipenv
    c) **PYTHONSTARTUP environment variable** ✅
    d) **init**.py

48. What should be included in a `.pythonrc.py` file to enable readline features?
    a) Only imports
    b) **Imports and tab binding** ✅
    c) print()
    d) logging

49. What happens if readline history file is not found during load?
    a) Error
    b) Shell crashes
    c) **Nothing; it continues silently or creates one** ✅
    d) Python exits

50. What makes IPython more powerful than the standard REPL for daily use?
    a) Smaller size
    b) Fewer logs
    c) **Enhanced editing, history, introspection, and integration features** ✅
    d) No need for Python

🔴 Hard Level (Q51–Q75)

51. What happens if you use `readline.write_history_file()` without arguments?
    a) Error
    b) **It attempts to write to the default history file, typically `.python_history`** ✅
    c) It writes to stdout
    d) It saves to system logs

52. What does `readline.set_pre_input_hook()` allow?
    a) Save history
    b) **Execute a function before each prompt input** ✅
    c) Run Python startup file
    d) Skip history

53. What is a primary reason to manage history with `readline` programmatically?
    a) Disable editing
    b) **Implement custom REPL or enhanced interactive features** ✅
    c) Reduce performance
    d) Remove auto-complete

54. In interactive shells, which signal may disrupt `readline`-based input?
    a) SIGKILL
    b) SIGSTOP
    c) **SIGINT (Ctrl+C)** ✅
    d) SIGLOG

55. What would you use `readline.insert_text('print("Hi")')` for?
    a) Compile script
    b) **Pre-fill the current input buffer with custom text** ✅
    c) Show history
    d) Save script

56. When using `readline.set_completer_delims()`, what is being set?
    a) Prompt text
    b) File path
    c) **Characters used to separate words for tab-completion** ✅

d) History index

57. Which method allows saving history up to a specific number of lines?
    a) history_cut()
    b) **readline.set_history_length(n)** ✅
    c) history_freeze(n)
    d) readline.crop(n)

58. Why is using `PYTHONSTARTUP` discouraged for complex startup scripts?
    a) Too slow
    b) **It only works in interactive sessions and may break tools** ✅
    c) Requires admin
    d) Blocks pip

59. What is one limitation of `readline` on cross-platform Python scripts?
    a) No library
    b) **Inconsistent availability and behavior across OSes** ✅
    c) Too verbose
    d) Doesn't save

60. What is the output of:

readline.get_history_item(readline.get_current_history_length())

a) 0
b) None
c) **The last command in history** ✅
d) Index error

61. Why might command-line editing not work in embedded interpreters?
    a) Import error
    b) **They may lack proper terminal capabilities or readline bindings** ✅
    c) Shell is slow
    d) Too many packages

62. What does a negative argument to `readline.remove_history_item(index)` cause?
    a) Nothing
    b) Deletes all
    c) **Raises an IndexError** ✅
    d) Clears buffer

63. Which function lets you override the behavior of tab-completion?
    a) readline.override_tab()
    b) set_auto()

c) **readline.set_completer(func)** ✅
d) replace_tab()

64. What is a good practice for extending shell functionality via readline?
a) Change config.py
b) **Define custom hooks and completers in `.pythonrc.py`** ✅
c) Use subprocess
d) Use IDLE

65. What's a use case for combining `readline` with `atexit`?
a) Load pandas
b) **Automatically save command history on exit** ✅
c) Delete logs
d) Reboot terminal

66. What does `readline.redisplay()` do during session?
a) Clears screen
b) Saves history
c) **Refreshes the current input line display** ✅
d) Logs output

67. How can one avoid history persistence between Python sessions?
a) Use `clear()`
b) **Avoid calling `write_history_file()`** ✅
c) Delete sys
d) Reset terminal

68. What's the primary difference between `input()` and `readline.get_line_buffer()`?
a) Input is for files
b) Same function
c) **`input()` gets full user input; `get_line_buffer()` shows typed buffer** ✅
d) One is deprecated

69. What type of applications often use `readline` programmatically?
a) Compilers
b) **Custom REPLs and shells** ✅
c) GUIs
d) Games

70. Which function returns the current position in the input line buffer?
a) cursor_get()
b) **readline.get_endidx()** ✅
c) readline.index()
d) get_cursor()

71. How can readline be extended to support path auto-completion?
    a) os.path.bind()
    b) autocomplete.py
    c) **Use `readline.set_completer()` with a custom path-aware function** ✅
    d) readline.paths()

72. Why doesn't tab-completion always trigger in Python's default REPL?
    a) Python bug
    b) **The interpreter must explicitly enable and configure readline support** ✅
    c) Windows restriction
    d) Too many modules

73. What is a security consideration when logging `.python_history`?
    a) Limited RAM
    b) **Sensitive data like credentials might be stored** ✅
    c) Slows shell
    d) Creates duplicates

74. What's the difference between `write_history_file()` and `append_history_file()`?
    a) append is faster
    b) **write overwrites; append adds to existing file** ✅
    c) same output
    d) append is deprecated

75. Which type of shell might disable readline features by default?
    a) IDLE
    b) IPython
    c) Bash
    d) **Embedded interpreters or restricted consoles** ✅

Perfect! Let's now continue with:

📘 Chapter 15: Floating Point Arithmetic — Issues and Limitations

(Section 15.1: Representation Error)

🟢 Easy Level (Q1–Q25)

1. What does Chapter 15 primarily discuss?
   a) Numbers in algebra
   b) **Issues with floating-point arithmetic in Python** ✅
   c) Lists and tuples
   d) Conditional logic

2. What is a floating-point number?
   a) Integer
   b) Character
   c) **A number with a decimal part** ✅
   d) String

3. Why are floating-point numbers sometimes inaccurate in Python?
   a) Bad compiler
   b) Slow processor
   c) **Due to binary representation limitations** ✅
   d) Incorrect code

What does this expression return?

0.1 + 0.2

4. a) 0.3
   b) **0.30000000000000004** ✅
   c) Error
   d) 1.0

5. Which standard do most modern computers follow for floating-point numbers?
   a) ISO-8859
   b) **IEEE 754** ✅
   c) UTF-8
   d) ASCII

6. Which built-in type does Python use for floating-point numbers?
   a) float64
   b) **float** ✅
   c) decimal
   d) real

7. What can help avoid floating-point inaccuracies in financial applications?
   a) integers
   b) **decimal module** ✅
   c) tuple
   d) for loops

8. Why does Python show `0.1 + 0.2` as `0.30000000000000004`?
   a) Bug in interpreter

b) Misuse of syntax
c) **0.1 and 0.2 can't be represented exactly in binary** ✅
d) Logic error

9. How does IEEE 754 store numbers?
a) String form
b) Hexadecimal
c) **As binary fractions (base 2)** ✅
d) Decimal only

10. What is the recommended way to compare floating-point numbers?
a) `==`
b) `!=`
c) **Use a tolerance with `math.isclose()`** ✅
d) Convert to string

11. What does the `decimal` module offer?
a) Matplotlib graphs
b) **Precise decimal arithmetic** ✅
c) List formatting
d) Hex to float

12. Which module provides better precision with decimals?
a) math
b) time
c) **decimal** ✅
d) input

13. Is `0.1 + 0.2 == 0.3` true in Python?
a) Yes
b) **No** ✅
c) Always
d) Depends on OS

14. What causes tiny inaccuracies in floating point calculations?
a) Human error
b) Poor IDE
c) **Binary fraction limitations** ✅
d) RAM overload

15. What does `round(0.1 + 0.2, 1)` return?
a) 0.1
b) **0.3** ✅
c) 0.2
d) 0.4

16. Which function is designed for comparing floats reliably?
    a) compare()
    b) eq()
    c) **math.isclose()** ✅
    d) floatcmp()

17. Why doesn't Python always show the exact value stored in memory for floats?
    a) Hidden bug
    b) **It tries to display the shortest decimal equivalent** ✅
    c) Low precision
    d) Memory error

18. What's the goal of `math.isclose(a, b)`?
    a) True/False for integers
    b) **Return True if a and b are nearly equal** ✅
    c) Round numbers
    d) Convert types

19. Which import is required for `isclose`?
    a) `from decimal import isclose`
    b) `import os`
    c) **`import math`** ✅
    d) `import sys`

20. What does the term "representation error" refer to?
    a) GUI bug
    b) **Inability to represent decimal values exactly in binary** ✅
    c) Formatting issue
    d) Input mismatch

21. Can all decimal fractions be represented precisely in binary?
    a) Yes
    b) **No** ✅
    c) Only integers
    d) Only even numbers

22. How many bits are typically used to store a Python float on modern systems?
    a) 8
    b) 16
    c) **64** ✅
    d) 128

23. What happens when you use too many decimal digits in a float?
    a) Program crash
    b) Zero output
    c) **Loss of precision** ✅

d) Round-off

24. How does the `decimal` module avoid representation errors?
    a) Using binary
    b) **Using exact base-10 arithmetic** ✅
    c) Using strings
    d) Disabling floats

25. Why is `format(0.1, '.20f')` useful?
    a) Converts to int
    b) **Displays the full binary approximation in decimal** ✅
    c) Rounds to 1
    d) Shows memory

🟡 Medium Level (Q26–Q50)

26. What is the output of the following?

```
format(0.1 + 0.2, '.17f')
```

a) 0.30000000000000000
b) **0.30000000000000004** ✅
c) 0.3
d) 0.1

27. Why are floats imprecise in binary systems?
    a) Compiler limitation
    b) Incorrect libraries
    c) **Some decimal values have repeating binary representations** ✅
    d) Python bug

28. Which two values are used by `math.isclose()` to determine closeness?
    a) Tolerance and accuracy
    b) Margin and average
    c) **Relative tolerance (`rel_tol`) and absolute tolerance (`abs_tol`)** ✅
    d) Precision and float

29. What's a key feature of the `decimal.Decimal` type?
    a) Integer only
    b) Uses base 2
    c) **Preserves decimal exactness** ✅

d) Truncates automatically

30. What's the result of:

from decimal import Decimal

Decimal('0.1') + Decimal('0.2')

a) 0.30000000000000004
b) 0.31
c) **Decimal('0.3')** ✅
d) Error

31. When does `float('nan') == float('nan')` evaluate to True?
    a) Always
    b) **Never — NaN is not equal to anything, including itself** ✅
    c) When rounded
    d) On Linux only

32. What causes loss of significance in floating-point arithmetic?
    a) Type errors
    b) Small memory
    c) **Subtracting two nearly equal floating-point numbers** ✅
    d) Division

33. Why is rounding necessary in financial computations?
    a) Saves memory
    b) **Avoids errors due to floating point inaccuracy** ✅
    c) Improves performance
    d) Converts strings

34. What is the default relative tolerance in `math.isclose()`?
    a) 0.0001
    b) **1e-09** ✅
    c) 1e-06
    d) 0

35. What is a denormal (or subnormal) float?
    a) Normal integer
    b) **Very small number closer to zero than can be represented normally** ✅
    c) Truncated binary
    d) Hexadecimal

36. What does `round(2.675, 2)` return and why is it surprising?
    a) **2.67 — due to binary floating point approximation** ✅

b) 2.68 — always correct
c) 2.675 — no rounding
d) Error

37. What kind of error arises due to binary-to-decimal mismatch?
a) OverflowError
b) LogicError
c) **Representation error** ✅
d) UnicodeError

38. Which format specifier can help highlight floating-point inaccuracies?
a) `.0f`
b) `.1g`
c) **`.20f`** ✅
d) `.2x`

39. How does Python mitigate confusion about floating-point outputs?
a) It raises warning
b) **It rounds results when printed for readability** ✅
c) Uses string
d) Uses 32-bit float

40. Why does the decimal module outperform floats in precision?
a) Less space
b) **Performs arithmetic in base 10** ✅
c) Binary optimization
d) Compiler trick

41. Which method in the decimal module sets global precision?
a) setup()
b) get_decimals()
c) **getcontext().prec =** ✅
d) round()

42. What is the output?

```
from decimal import Decimal, getcontext

getcontext().prec = 3

print(Decimal('1') / Decimal('7'))
```

a) 0.142857
b) **0.143** ✅

c) 0.1
d) Error

43. Why should you avoid comparing floats directly with `==`?
    a) Raises exceptions
    b) **Tiny representation errors can cause comparison failure** ✅
    c) Slower performance
    d) Converts to int

44. What happens when you exceed the float range?
    a) Zero
    b) **Returns `inf` or `-inf`** ✅
    c) Segfault
    d) Raises KeyError

45. What is the IEEE representation of a float made of?
    a) Exponent only
    b) String
    c) **Sign bit, exponent, and mantissa** ✅
    d) Digits

46. What is a practical workaround for float inaccuracies in money?
    a) Use string
    b) Bitwise ops
    c) **Use `decimal` or represent in smallest currency units (e.g. cents)** ✅
    d) Integer math only

47. What does `float('1e400')` return?
    a) Error
    b) 1.0
    c) **inf** ✅
    d) 0

48. What does `float('nan') != float('nan')` return?
    a) False
    b) Error
    c) **True — NaN is not equal to anything** ✅
    d) None

49. When would you use `sys.float_info`?
    a) System name
    b) RAM check
    c) **To inspect float limits and precision details** ✅
    d) Check pip version

50. Why should testing code involving floats use `assertAlmostEqual` in unit tests?
    a) Because floats change

b) Fast comparison
c) **To tolerate tiny differences in float values** ✅
d) Ignores exceptions

🔴 Hard Level (Q51–Q75)

51. Which of the following decimal values can be exactly represented in binary floating point?
a) **0.5** ✅
b) 0.1
c) 0.3
d) 0.7

52. What is a correct way to compare floating-point numbers in critical applications?
a) Using `==`
b) Converting to strings
c) **Using tolerances with `math.isclose()` or `assertAlmostEqual()`** ✅
d) Substring match

53. Which module would you use to examine the IEEE representation of floats in Python?
a) floatinfo
b) ieee754
c) **struct** ✅
d) decimal

54. What would be the result of:

```
from decimal import Decimal

Decimal('0.1') + Decimal('0.2') == Decimal('0.3')
```

a) False
b) Error
c) **True** ✅
d) None

55. What kind of arithmetic is implemented by the `decimal` module?
a) IEEE 754
b) Randomized
c) **Base-10 exact arithmetic** ✅
d) Approximate math

56. What does `float('nan') + 1` return?
   a) 1
   b) Error
   c) **nan** ✅
   d) inf

57. What type of rounding errors may result from floating-point division?
   a) None
   b) **Cumulative round-off error** ✅
   c) Integer overflow
   d) RecursionError

58. Which floating-point values are used to represent overflow?
   a) 0
   b) -1
   c) **inf and -inf** ✅
   d) floatmax

59. What behavior would you expect from:

```
0.1 + 0.1 + 0.1 == 0.3
```

a) **False — due to binary rounding error** ✅
b) True
c) Syntax error
d) Always exception

60. Why is `Decimal('0.1')` more accurate than `float(0.1)`?
   a) Faster
   b) Converts to int
   c) **Stores exactly 0.1 as entered** ✅
   d) Limits float

61. What is the role of `context.prec` in the decimal module?
   a) Import checker
   b) File writer
   c) **Defines number of significant digits used in calculations** ✅
   d) Memory limit

62. What's the output of:

```
from math import isclose

isclose(1.0000001, 1.0000002, rel_tol=1e-7)
```

a) False
b) **True** ✅
c) None
d) Error

63. Which is **NOT** a typical cause of floating-point errors?
    a) Finite precision
    b) Binary fraction limitations
    c) **Garbage collection** ✅
    d) Rounding error

64. Why does Python represent `0.1` internally as a longer decimal?
    a) For accuracy
    b) **Because 0.1 cannot be exactly stored in binary** ✅
    c) Due to Python3 changes
    d) To confuse users

65. What effect does subtraction of nearly equal floats have?
    a) Doubling
    b) No effect
    c) **Loss of significant digits (catastrophic cancellation)** ✅
    d) Converts to int

66. What is the difference between `math.isclose()` and `decimal.Decimal.compare()`?
    a) Both are identical
    b) **`isclose()` is for floats, `compare()` is for Decimals** ✅
    c) One is deprecated
    d) Both use relative error

67. What causes `float('nan') != float('nan')` to be True?
    a) Float mismatch
    b) Library bug
    c) **IEEE 754 specifies NaN != NaN by design** ✅
    d) Decimal rounding

68. What does `Decimal('NaN')` return in arithmetic?
    a) Zero
    b) Raises error
    c) **NaN stays in results — it's contagious** ✅
    d) Rounds to zero

69. What is the most common base used in floating-point internal representation?
    a) Base-10
    b) Base-12

c) **Base-2 (binary)** ✅
d) Base-16

70. What value is returned by `Decimal('Infinity') * 0`?
   a) Infinity
   b) Zero
   c) **NaN** ✅
   d) Error

71. Which module provides context managers for decimal operations?
   a) math
   b) io
   c) **decimal** ✅
   d) struct

72. What is a downside of using the `decimal` module over `float`?
   a) Inaccuracy
   b) **Slower performance** ✅
   c) Less readable
   d) Float errors

73. Which term describes the inability to exactly represent some decimals in binary?
   a) Buffer underflow
   b) **Representation error** ✅
   c) Compiler mismatch
   d) Float recursion

74. What is the format used to print the exact internal representation of a float?
   a) str()
   b) repr()
   c) **format(x, '.20f')** ✅
   d) round(x, 5)

75. How do you safely perform precise currency calculations in Python?
   a) Use float()
   b) Multiply strings
   c) **Use `decimal.Decimal` and fixed precision** ✅
   d) Use `math.floor`
