Python Training Quiz

Instructions:

- For questions 1-30, choose the **single best** option.
- For questions 31-50, choose **all** correct options.

Part 1: Single Correct Answer (30 Questions)

1. Which of the following principles of OOPS refers to the ability of an object to take on many forms?
   a) Encapsulation
   b) Inheritance
   c) Polymorphism
   d) Abstraction
2. What is the purpose of the __init__ method in a Python class?
   a) To destroy an object
   b) To initialize the attributes of an object
   c) To define a static method
   d) To represent the object as a string
3. Which regex special character matches zero or more occurrences of the preceding character?
   a) +
   b) *
   c) ?
   d) .
4. In Python, what module is commonly used for working with regular expressions?
   a) os
   b) sys
   c) re
   d) math
5. What is the primary benefit of using threads in a Python application?
   a) To make the code run sequentially
   b) To improve CPU-bound operation performance
   c) To achieve concurrency for I/O-bound operations
   d) To reduce memory consumption
6. Which of the following is not a standard way to create a thread in Python's threading module?
   a) Subclassing threading.Thread
   b) Passing a target function to threading.Thread
   c) Using os.fork()

d) Using concurrent.futures.ThreadPoolExecutor

7. Which protocol is primarily used for reliable, connection-oriented communication over a network?
   a) UDP
   b) ICMP
   c) TCP
   d) ARP
8. What is the default port for HTTP?
   a) 21
   b) 23
   c) 80
   d) 443
9. Which NumPy function is used to create an array with a specified start, end, and step value?
   a) np.linspace()
   b) np.arange()
   c) np.array()
   d) np.zeros()
10. What is the main data structure in Pandas for storing tabular data with labeled axes (rows and columns)?
    a) Series
    b) DataFrame
    c) Panel
    d) Array
11. Which of the following is a correct syntax for a list comprehension?
    a) [x for x in iterable if condition]
    b) {x for x in iterable if condition}
    c) (x for x in iterable if condition)
    d) x for x in iterable if condition
12. What does a generator comprehension return?
    a) A list
    b) A tuple
    c) A generator object
    d) A set
13. Which of these is an example of an "advanced data structure" in the context of Python's built-in types?
    a) List
    b) Tuple
    c) Dictionary
    d) All of the above
14. What is the key difference between a list and a tuple in Python?
    a) Lists are ordered, tuples are unordered.
    b) Lists are mutable, tuples are immutable.

c) Lists can store heterogeneous data, tuples cannot.
d) Tuples are faster to access than lists.

15. What is the purpose of the yield keyword in Python?
    a) To return a value from a function and terminate its execution.
    b) To pause function execution and return an iterator.
    c) To define a class method.
    d) To raise an exception.
16. Which of the following is an example of an iterator in Python?
    a) A list
    b) A string
    c) A file object
    d) All of the above (when iterated over)
17. What is a decorator in Python primarily used for?
    a) To define a new class.
    b) To modify the behavior of a function or method.
    c) To handle exceptions.
    d) To import modules.
18. What symbol is used to apply a decorator to a function?
    a) &
    b) @
    c) #
    d) $
19. Which of the following is not a benefit of using generators?
    a) Memory efficiency
    b) Lazy evaluation
    c) Infinite sequences
    d) Direct random access to elements
20. What is the output of [i for i in range(5)]?
    a) (0, 1, 2, 3, 4)
    b) [0, 1, 2, 3, 4]
    c) range(0, 5)
    d) An error
21. In OOPS, what is a method?
    a) A variable inside a class.
    b) A function inside a class.
    c) An instance of a class.
    d) A blueprint for creating objects.
22. Which of the following will match "color" and "colour" using regex?
    a) colou?r
    b) colo.?r
    c) colou*r
    d) colo+r
23. When should you use threading.Lock()?

a) To prevent deadlocks.
b) To synchronize access to shared resources by multiple threads.
c) To terminate a thread.
d) To start a new process.

24. What is a socket in network programming?
    a) A physical network interface card.
    b) An endpoint for communication between two programs.
    c) A type of network cable.
    d) A network protocol.
25. How do you select rows in a Pandas DataFrame based on a condition on a column?
    a) df['column_name'].filter(condition)
    b) df[df['column_name'] > value]
    c) df.loc[condition]
    d) df.select_rows(condition)
26. Which of the following NumPy operations performs element-wise multiplication on two arrays of the same shape?
    a) np.dot(arr1, arr2)
    b) arr1 @ arr2
    c) arr1 * arr2
    d) np.multiply(arr1, arr2) (Both c and d are correct, but c is more common and direct)
27. What is the purpose of the super() function in Python?
    a) To call the constructor of the current class.
    b) To call a method from a grandparent class.
    c) To call a method from the parent class.
    d) To create a new instance of a class.
28. Which of the following is true about Python's Global Interpreter Lock (GIL)?
    a) It allows multiple threads to execute Python bytecode concurrently on multiple CPU cores.
    b) It prevents multiple threads from executing Python bytecode at the same time.
    c) It only affects I/O-bound operations.
    d) It is present only in older versions of Python.
29. What is the output of list(map(lambda x: x*2, [1, 2, 3]))?
    a) [1, 2, 3, 1, 2, 3]
    b) [2, 4, 6]
    c) [1, 4, 9]
    d) Error
30. Which decorator is used to make a method a property (allowing it to be accessed like an attribute)?
    a) @staticmethod
    b) @classmethod
    c) @property
    d) @abstractmethod

Part 2: Multiple Correct Answers (20 Questions)

Instructions: Select all options that are correct.

31. Which of the following are pillars of Object-Oriented Programming (OOP)?
    a) Abstraction
    b) Encapsulation
    c) Inheritance
    d) Polymorphism
32. What are valid ways to create an empty Pandas DataFrame?
    a) pd.DataFrame()
    b) pd.DataFrame(columns=['A', 'B'])
    c) pd.DataFrame([])
    d) pd.DataFrame(data={})
33. Which of the following are true about generators in Python?
    a) They use the yield keyword.
    b) They can be iterated over multiple times without resetting.
    c) They are memory efficient for large sequences.
    d) They implement the iterator protocol.
34. Which of these are benefits of using comprehensions (list, set, dict) in Python?
    a) More concise code
    b) Often more readable than traditional loops
    c) Potentially faster execution
    d) They can replace all types of loops
35. Which of the following regular expression patterns would match a sequence of one or more digits?
    a) \d+
    b) [0-9]+
    c) \d*
    d) [0-9]*
36. Which of these are common applications of threading in Python?
    a) Downloading multiple files concurrently
    b) Performing complex mathematical calculations on a single CPU
    c) Running a GUI and background tasks simultaneously
    d) Handling multiple client connections in a server application
37. Which of the following are valid ways to define a class method in Python?
    a) Using @classmethod decorator.
    b) Passing self as the first argument.
    c) Passing cls as the first argument to the method decorated with @classmethod.
    d) Directly calling a method on the class name.
38. Which of these are built-in advanced data structures in Python?
    a) collections.deque

b) heapq
c) collections.Counter
d) graph

39. Which of the following are types of network sockets in Python's socket module?
    a) socket.SOCK_STREAM (for TCP)
    b) socket.SOCK_DGRAM (for UDP)
    c) socket.SOCK_RAW
    d) socket.SOCK_HTTP
40. When should you use numpy.array instead of a Python list?
    a) When performing numerical operations on large datasets.
    b) When memory efficiency is critical for numerical data.
    c) When you need built-in mathematical functions like dot product, element-wise operations.
    d) When you need a highly flexible and heterogeneous data structure.
41. Which of the following are true about Python decorators?
    a) They are syntactic sugar for wrapping functions.
    b) They modify the original function in place.
    c) They take a function as an argument and return a new function.
    d) They must be defined in a separate file.
42. Consider the following code:

```Python
def my_generator():
    yield 1
    yield 2
gen = my_generator()
```

    Which of the following statements would produce 1?
    a) next(gen)
    b) gen.__next__()
    c) list(gen)[0]
    d) gen[0]
43. Which of the following are true regarding the __str__ and __repr__ methods in Python?
    a) __str__ is for creating an "official" string representation for developers.
    b) __repr__ is for creating a "readable" string representation for end-users.
    c) __str__ is called by str() and print().
    d) If __str__ is not defined, __repr__ is used as a fallback for str().
44. Which of these are valid ways to iterate over a Pandas DataFrame?
    a) for index, row in df.iterrows():
    b) for column in df.columns:
    c) for item in df.values:
    d) for (index, col), value in df.stack().items():
45. Which of the following are methods of the threading.Thread class?
    a) start()

b) run()
c) join()
d) terminate()

46. Which of the following are common issues encountered in multithreaded programming?
    a) Race conditions
    b) Deadlocks
    c) Starvation
    d) Syntax errors (specific to threading)
47. Which of the following are true about iterators in Python?
    a) They implement the __iter__ and __next__ methods.
    b) They can be created from any iterable object using iter().
    c) Once exhausted, they cannot be reused.
    d) Lists are iterators.
48. Which of these are typically used for "lazy evaluation" in Python?
    a) Lists
    b) Generators
    c) Iterators
    d) Tuples
49. Which of the following are true about asyncio compared to threading?
    a) asyncio is better for I/O-bound tasks.
    b) threading is better for CPU-bound tasks in Python (despite GIL, due to OS scheduling).
    c) asyncio uses a single thread to manage multiple concurrent operations.
    d) threading creates multiple processes.
50. Which of the following are common use cases for regular expressions?
    a) Validating email addresses
    b) Parsing log files
    c) Searching and replacing text in strings
    d) Performing complex mathematical calculations

Hands-on Lab Assignment: Python Concepts Integration Project

Objective: To apply OOPS, Regex, Threading, Network Programming, Data Science (Pandas, NumPy), Comprehensions, Advanced Data Structures, Iterators, Decorators, and Generators in a practical scenario.

Scenario: You need to build a system that analyzes server log files, extracts specific information, performs some data aggregation, and presents insights. The system should be able to handle large log files efficiently.

Part 1: Log File Generator (Basic I/O & Comprehension)

1. **Create a synthetic log file:** Write a Python script that generates a log file named server.log.
    - The file should contain at least 1000 lines.
    - Each line should simulate a server request with the following format: YYYY-MM-DD HH:MM:SS [LEVEL] - IP_ADDRESS - /path/to/resource - STATUS_CODE - RESPONSE_TIME_MS - USER_AGENT
        - YYYY-MM-DD HH:MM:SS: Random datetime within the last 7 days.
        - [LEVEL]: Randomly choose from [INFO], [WARNING], [ERROR].
        - IP_ADDRESS: Random IP address (e.g., 192.168.X.Y).
        - /path/to/resource: Randomly choose from a small set of paths (e.g., /home, /api/users, /data/report).
        - STATUS_CODE: Randomly choose from 200, 404, 500.
        - RESPONSE_TIME_MS: Random integer between 50 and 1000.
        - USER_AGENT: Simple string like Mozilla/5.0, Chrome/90, Safari/14.
    - Use list/generator comprehensions where appropriate to generate data.

Part 2: Log Analyzer (OOPS, Regex, Generators, Iterators, Decorators)

1. **Define a LogEntry Class (OOPS):**
    - Create a class LogEntry that represents a single parsed log line.
    - The __init__ method should take a raw log line string and parse it using regular expressions.
    - Attributes should include: timestamp, level, ip_address, resource_path, status_code, response_time_ms, user_agent.
    - Implement __str__ and __repr__ methods for better debugging and display.
2. **Create a LogParser Generator (Generators & Iterators):**
    - Write a function parse_log_file(filepath) that acts as a generator.
    - It should open the server.log file, read it line by line, parse each line using the LogEntry class, and yield LogEntry objects.
    - Handle potential FileNotFoundError.
3. **Implement a Decorator for Error Logging (Decorators):**
    - Create a decorator @log_errors that can be applied to methods of your LogAnalyzer class.
    - This decorator should catch any exceptions raised by the decorated method, log the error message (e.g., print to console or write to a separate error log file), and then re-raise the exception or return a default value as appropriate.
4. **Develop a LogAnalyzer Class (OOPS, Decorators, Advanced Data Structures):**
    - Create a class LogAnalyzer.
    - The __init__ method should take the log file path.
    - **Method: get_error_logs():**
        - Uses the parse_log_file generator.
        - Returns a list of LogEntry objects where level is [ERROR].

        - Apply the @log_errors decorator to this method.
    - **Method: get_status_code_counts():**
        - Uses the parse_log_file generator.
        - Returns a collections.Counter object (advanced data structure) mapping status_code to their counts.
        - Apply the @log_errors decorator.
    - **Method: get_average_response_time(resource_path=None):**
        - Uses the parse_log_file generator.
        - Calculates the average response_time_ms.
        - If resource_path is provided, calculate the average only for that specific resource.
        - Apply the @log_errors decorator.
    - **Method: get_unique_ips():**
        - Uses the parse_log_file generator.
        - Returns a set (advanced data structure) of unique IP addresses.
        - Apply the @log_errors decorator.

Part 3: Data Analysis with Pandas & NumPy (Data Science)

1. **Integrate with Pandas:**
    - Modify the LogAnalyzer class or create a new function that takes the list of LogEntry objects (e.g., from parse_log_file) and converts it into a Pandas DataFrame.
    - Ensure columns are correctly typed (e.g., timestamp as datetime, response_time_ms as integer).
2. **Perform Data Analysis using Pandas & NumPy:**
    - Using the DataFrame:
        - Find the top 5 most frequently accessed resources.
        - Calculate the 95th percentile response time for 200 OK requests.
        - Group by level and status_code to see counts of each combination.
        - Identify the IP address with the highest number of [ERROR] requests.
        - Use NumPy functions where appropriate (e.g., for percentile calculation).

Part 4: Concurrent Log Processing (Threading & Network Programming)

1. **Simulate Concurrent Log Processing (Threading):**
    - Imagine you have multiple small log files (e.g., server_part1.log, server_part2.log, etc.).
    - Create a function process_single_log_file(filepath) that takes a log file path, uses your LogAnalyzer to get some statistics (e.g., error count, average response time), and prints them.
    - Use threading.Thread to run process_single_log_file concurrently on 3-5 different small log files (you'll need to generate these small files first, perhaps by splitting your main server.log).
    - Use thread.join() to wait for all threads to complete.

2. **Basic Network Service (Network Programming):**
    - Create a simple TCP server (using Python's socket module) that listens on a specific port (e.g., 12345).
    - When a client connects and sends the message "GET_ERROR_COUNT", the server should respond with the total number of [ERROR] logs from your server.log file.
    - You can integrate your LogAnalyzer within the server to get this count dynamically.
    - Create a simple client script that connects to this server, sends the message, and prints the response.
    - Consider how to handle multiple client connections (briefly mention if you would use threading/asyncio for this in a real-world scenario, but for this lab, a single client at a time is sufficient).

Submission:

- A single Python file containing all the code for the log file generator, LogEntry class, LogParser generator, LogAnalyzer class (with decorator), Pandas integration, and threading example.
- A separate Python file for the basic TCP server.
- A separate Python file for the basic TCP client.
- A brief README.md file explaining how to run the scripts and the insights you gained from the data analysis.