PYTHON PRACTICE

1. Read a name and age; print `"Alice will be 30 in 5 years"` format using f-strings.
2. Reverse a String (without slicing)
3. Compute area and circumference of a circle from radius (use `pi = 3.14159`).
4. Given a string, print first/last char, middle slice, and reversed string.
5. Parse a comma-separated string of integers into a list of `int` and print sum/mean.
6. Implement a simple calculator reading `a op b` (e.g., `10 * 3`) and output result. Handle `+ - * / // % **`.
7. Dynamic vs static typing in Python—implications for variable assignment?
8. Difference between `/` and `//`? Between == and `is`?
9. Truthy/falsey rules in Python?
10. String immutability: what operations create new objects?
11. How does `input()` return value type; how to safely convert?
12. Classify a number as positive/negative/zero.
13. Sum all numbers from 1..N using a `for` loop.
14. Print all multiples of 3 between A and B (inclusive).
15. Check if a number is prime using loop + break/else.
16. Generate Fibonacci numbers ≤ N using `while`.
17. When does `for ... else` execute the `else`?
18. Differences between `break` and `continue`.
19. When to prefer `while` over `for`?
20. Short-circuiting in boolean expressions.
21. Avoiding infinite loops—patterns.
22. Remove duplicates from a list while preserving order.
23. Count character frequency in a string into a dict.
24. Merge two dicts; for common keys, sum values.
25. Given a list of tuples `(name, score)`, get top 3 by score.
26. Implement set operations (union/intersection) manually using loops.
27. List vs tuple—use cases and performance.
28. Set/dict internal structure (hashing) at a high level.
29. Common list methods and their time complexity (amortized).
30. What is a comprehension; generator vs list comprehension?
31. When do dict keys need to be immutable/hashable?
32. Implement `power(x, n)` recursively.
33. Write `flatten_one_level(lst)` that flattens one nesting level.

34. Implement `memoized_fib(n)` using a dict cache.
35. Write `compose(f, g)` returning a function `h(x)=f(g(x))`.
36. Build `partial_sum(*nums)` returning a closure that remembers cumulative sum.
37. Positional-only, keyword-only parameters (syntax and use cases).
38. Default arg pitfalls with mutable types.
39. Explain first-class functions and closures with examples.
40. Tail recursion in Python—supported? (No optimization.)
41. Uses of `*args` and `**kwargs`.
42. Copy a file line-by-line.
43. Count words and lines in a file.
44. Write CSV writer/reader without using `csv` (then repeat with `csv`).
45. Append logs with timestamps.
46. Safely read a binary file and compute its SHA-256 (`hashlib`).
47. Why use `with` for files?
48. Difference between `w` and `a`.
49. Handling encoding errors.
50. Reading large files efficiently.
51. File descriptors vs file objects (high-level).
52. Implement `Vector2D` with `+`, `-`, and `len()` via dunder methods.
53. Create `Employee` base and `Manager`, `Engineer` subclasses with role-specific methods.
54. Add property validation for email on a `User` class.
55. Implement iterable `Deck` that yields cards and supports `len()`.
56. Convert a simple class to `@dataclass` and compare.
57. Difference between instance, class, and static methods.
58. Method resolution order (MRO) in multiple inheritance.
59. `__repr__` vs `__str__`; importance for debugging.
60. How properties work; computed attributes.
61. When to choose composition over inheritance.
62. Wrap division to handle `ZeroDivisionError`.
63. Validate user input loop until correct int.
64. Implement custom exception in banking example.
65. Use `try/except/else/finally` in file processing.
66. Convert sentinel error codes to exceptions.
67. Flow of `try/except/else/finally`.
68. When to catch broad `Exception` vs specific.
69. Raising vs returning error codes—tradeoffs.

70. Context manager exceptions.
71. Stack traces and debugging.
72. Write `even_numbers(n)` generator.
73. Infinite generator of Fibonacci numbers; stop after first 20.
74. Pipeline generators: read file -> yield stripped lines -> filter non-empty.
75. Re-implement `zip` via a generator.
76. Build a paginated API fetch simulator using a generator (mock data).
77.
78. Iterator protocol in Python.
79. Differences: list vs generator expression.
80. `yield from` usage.
81. When generators improve performance.
82. StopIteration—how it propagates.
83. Decorator to retry a function N times on exception.
84. Decorator to cache results (simple memoize).
85. Context manager that temporarily changes the working directory.
86. Context manager to suppress specified exceptions.
87. Decorator adding role-based access control check.
88. How decorators are applied; preserving metadata (`functools.wraps`).
89. Use cases for context managers beyond files.
90. What does `@classmethod` / `@staticmethod` do?
91. Multiple decorators stacking order.
92. Difference between decorator function and decorator factory.
93. Validate IPv4 address.
94. Extract all hashtags from text.
95. Replace multiple spaces with a single space.
96. Parse `key=value` pairs into a dict.
97. Mask credit card digits except last 4.
98. `match` vs `search`.
99. Greedy vs non-greedy quantifiers.
100. Use of anchors `^` and `$`, word boundary `\b`.
101. Pre-compiling patterns with `re.compile`.
102. Multiline and DOTALL flags.
103. Use threads to download multiple URLs (I/O bound).
104. Use process pool to compute primes in a range.
105. Convert a synchronous API client to `asyncio` using `aiohttp`.

106. Benchmark `threading` vs `multiprocessing` for a CPU task.
107. Implement a producer–consumer with `queue.Queue`.
108. What is the GIL; when does it matter?
109. When to choose threads vs processes vs asyncio?
110. Thread safety; using `Lock`, `Queue`.
111. Cancellation and timeouts in `asyncio`.
112. Pickling constraints in multiprocessing.
113.
114. Use `Counter` to find top-k frequent words.
115. Use `pathlib` to list all `.py` files recursively.
116. Use `deque` to implement fixed-size rolling window average.
117. Use `itertools.groupby` to compress consecutive duplicates.
118. Serialize/deserialize objects with `json`.
119. Differences: `os.path` vs `pathlib`.
120. Use cases for `defaultdict` and `Counter`.
121. Benefits of `lru_cache`.
122. Itertools patterns (`islice`, `groupby`, `accumulate`).
123. Timezone-aware datetimes.
124. Create a new venv and install `requests`, `pytest`.
125. Export and rehydrate environment from `requirements.txt`.
126. Use `pip list --outdated` and update a package safely.
127. Create a minimal `setup.cfg/pyproject.toml` for linters (optional).
128. Use `pip install -e .` for editable local package (optional).
129. Why use virtual environments?
130. Pinning versions: pros/cons.
131. Difference between `requirements.txt` vs `pyproject.toml`.
132. Site-packages vs project-local packages.
133. How to handle dependency conflicts.
134. Write unit tests for a calculator module.
135. Parametrize tests for edge cases (zero, negatives).
136. Use fixtures in pytest to set up temporary files.
137. Add CI-friendly commands (`pytest -q`).
138. Measure coverage with `coverage.py`.
139. Difference: unittest vs pytest.
140. What is a fixture?

141. Mocks and patching (`unittest.mock` basics).
142. Test isolation and determinism.
143. Organizing tests and naming conventions.
144. Implement binary search (iterative & recursive).
145. Merge intervals and return non-overlapping list.
146. Group anagrams.
147. LRU cache (dict + doubly linked list or `OrderedDict`).
148. K-th largest element (heap).
149. Time/space tradeoffs of common data structures.
150. When to use heap vs sort.
151. Detect cycle in linked list (Floyd's algorithm).
152. Complexity of dictionary operations.
153. Designing a rate limiter (conceptual + data structures).