- Python's exception hierarchy

- hierarchy for built-in types in Python

- Python's input() method to create a dynamic decision tree troubleshooter

- `list.pop()` method which is an in-place operation

- Time complexity of `min()` and`max()` on a list of constant size

- mutable object references and shallow vs deep copy

- Python truthy falsy

- asyncio and with async/await syntax

- Python's I/O model: How to leverage blocking, non-blocking, synchronous, and asynchronous I/O

- Buffer Protocol

- OOP: Classes, objects, Inheritance, Polymorphism, and Encapsulation

- Collections Module

- Context Managers in Python
- Sets in Python are unordered collections of unique elements.

- Data Serialization in Python with JSON and XML

- Decorators are very Powerful

- Deep Copy vs. Shallow Copy in Python: A Deep Dive

- Dependency Injection in Python

- Descriptor protocol, the secret behind properties, methods, and static methods,

- `dict_values`object in Python, which is a view into the Python dictionary's entries

- `dict.popitem()` is very useful in multi-thread edenvironments

- function arguments are passed by reference

- exception handling

- event-driven programming using Asyncio

- Functional programming in Python

- initializer in the `reduce()` function

- Function Annotations in Python

- Function Overloading in Python

- `functools` Module

- Generator Expressions and Coroutines

- `__init__` Method in Python

- Iterables vs. Iterator in Python

- logging and debugging, monitoring

- metaclasses

- Python's Method Resolution Order (MRO) and C3 Linearization

- Monkey Patching

- Class attributes and Instance attributes

- shallow and deep copying

- Default Mutable Arguments - A big source of bug

- destructor method

- In Generators expression the `in` clause is evaluated at declaration time

- Generators in Python

- heapq.heapify(pq)

- concatenating 2 lists with the '+' operator (which is a binary operator) creates new list and not append and extend

- slice operation modifies array in place

- `bool` data type is a subclass of the `int` data type

- Tuple Unpacking in Python

- Dynamic Code Execution

- Monkey Patching for Performance Enhancements

- Concurrency and parallelism

- Threading and multiprocessing in Python

- Real-time Data Aggregation with Python using `concurrent.futures` with `ProcessPoolExecutor`

- Overloading Arithmatic Operators

- Python's C API for lightning-fast performance

- Python's reference counting mechanism and garbage collection

- `__slots__`: Memory Efficiency Meets Python!

- Python's `struct` module

- Dynamic keys with Python's `defaultdict`

- Multiplying an Integer with None will raise Type Error

- `Argparse`, the argument parser module

- key hashing mechanism

- String Interning

- slicing tips with Python list

- Parallel processing with multiprocessing

- Shallow copying in Python and the difference between copying with `b = a` and `c = a.copy()`

- Tuple comparison in Python and operator precedence rules

- Ever wondered why two identical Python objects aren't the same in memory?

- *args and **kwargs

- bool(), all(), any()—unlocking logic magic!

- Python's memory with caching decorators

- Chained Comparison mechanism in Python

- Quirky behavior of class-level (or static) attributes

- "Late Binding" in Closures

- Closures capture variables by reference and not by value

- concurrent modification

- 3 of the MOST important Python's decorators

- Default arguments AND its Tricky Behaviour

- Default arguments are evaluated only once when the function is defined

- Definition of identity of an object in Python?

- Python's `==` (equality) operation

- Tricky Behaviour around "==" and "is" operator

- "dunder" method

- Double underscores in Python is a way to obscure attributes and methods in a class and prevent direct access to them

- Tricky Magic method `__index__`

- Ellipsis literal

- `all()` function and "Vacuous Truth" in logic and Programming

- Equality vs identity and customize Python's special methods `__bool__`

- Equality vs Identity, Integer Interning and the `is` operator in Python

- exception handling - Tricky Behaviour

- Forced Garbage Collection with Python `ctypes`

- list comprehension-Efficient use case

- Generator Function, the principle of lazy evaluation

- How `globals()` keyword works in python

- What is Hashable In Python

- implied empty string ('') between each character in Python

- Inheritance model, class-level attributes and the concept of 'shadowing' in Python

- Instance attribute vs class attribute

- `is not` operator in Python

- Function decoration, the `setattr` function and lambda functions

- Python handles list multiplication, by NOT creating multiple separate lists, BUT by creating multiple references to
- the same list.

- "truthy" and "falsy" values

- Priority of logical operators

- Logical `or` vs logical `and` Operator

- How Memory are referenced by Variable

- How Python References Mutable and Immutable objects

- Different behavior of `+` and `+=` operators with mutable objects in Python

- How Python manages memory and Mutable and Non-Mutable objects are referenced.

- Mutable and immutable data types in Python

- Chained Assignment and the Distinction between mutable and immutable objects

- Boolean logic (`not` operator), identity comparison (with `is`), and operator precedence

- Nonlocal Keyword and its use in Inner Function
-
- Optimal way too modify a list in place

- Raise a number to its power in Python

- pass by value vs pass by reference

- Bankers' rounding

- Scope and Namespaces in Python

- Python's slice Notation - a tricky example

- Tricky Ternary operator in Python

- Distinction between `all()` and `any()` and `not` conditionals in Python

- How truthiness is evaluated for different types of objects

- XOR operator

- Async function in Python proper Syntax

- WHY its generally NOT recommended to Iterate over a list while Modifying its size at the same time

- Nesting lambda function in Python

- Python sets are unordered collections

- Why might `a is not b` not be the same as `not a is b`?

- Abstract Base Classes (ABCs) in Python

- Abstract Base Classes (ABCs) are like small building blocks for creating a higher level of abstraction

- How can you avoid cyclical references and potential memory leaks

- Dont just Swallow Exceptions with Bare except Clauses - This can create issues with your code

- Boolean values as a subtype of integers in Python

- Caching with external caching systems like Redis in Python

- Catching and storing exceptions

- How Closures in Python remembers values in the enclosing scope

- Combining multiple scopes with ChainMap

- LONG list of the MOST common, subtle, and tricky sources of bugs

- Python generators and the `yield from` construct

- List Comprehension pitfalls

- Parallel Loop with the Process Class in Python

- Asynchronous and Recursive web scraper with Python.

- Python Asynchronous File Processing

- Common use-case of `asyncio` library

- Difference between `async/await` and traditional threading

- Convert any Python code to run in parallel with dask

- forkserver - while doing multiprocessing

- Parallel Loop with the Pool Class in Python

- Parallel Loop with the ProcessPoolExecutor Class in Python

- ThreadPool class

- Parallel Loop with the ThreadPoolExecutor Class

- Processes, Threads, and Python's Notorious GIL - 10 Key Points

- Parallel Loop with the Thread Class in Python

- Why I should Consider contextlib and with Statements for Reusable try/finally Behavior

- Why I should Consider Generator Expressions for Large List Comprehensions

- The @dataclass decorator in Python is used to decorate classes

- How does `dict` type achieve average-case O(1) lookups

- What's the difference between `type` and `isinstance`?

- Different ways can you remove duplicates in Python

- When working with Python Dictionary, Why you should NOT create instance attributes outside of the `__init__` method

- A broad `except` block can hide genuine errors

- Do not name your variable names as the name of some built_in class

- Hierarchy between fundamental entities in Python: objects, types, and specific instances

- Understand `None` Semantics

- Merge two dictionaries and benchmarking with `perfplot`

- Why 0.1 + 0.1 + 0.1 == 0.3 outputs False

- Function arguments are passed by object reference

- functools` in Python

- `lru_cache` - Unleash its power

- Making Classes Sortable in Python with `@functools.total_ordering

- Asynchronous generators

- The Global Interpreter Lock, or GIL

- How does the `hash` function work with user-defined objects

- How are coroutines different from threads

- How are dictionaries and sets implemented in CPython?

- How are slices represented in memory

- How do you handle memory views in Python?

- How does Python handle integer overflow?

- How does the `__call__` method work

- memory managed in Python

- Infinite Recursion - Forgetting to provide a base case

- Streaming Large Files and Iterators

- Itertools's count function vs range function

- itertools.chain - Manage Multiple iterators

- itertools.combinations - generate all possible pairings in a dataset

- itertools.count

- itertools.cycle — the unsung hero

- itertools.groupby

- itertools.product

- itertools.tee()

- itertools.zip_longest

- itertools.product() - Avoid Nested Python Loops

- itertools.compress

- How are list comprehensions more efficient than traditional for loops

- How exactly the `lru_cache` decorator works under the hood

- `max()` and `min()` - have a dual life. They are both regular function and also higher-order functions

- Misusing or Overusing `__magic__` methods

- Name Shadowing

- Not all errors raise exceptions

- Not Validating External Inputs is BAD

- Differences between the `+=` operator on a list and a tuple

- Python performance profiling

- When we should use pydantic BaseModel instead of @dataclass?

- Reference counting in Python

- Searching through sorted collections using bisect

- Difference between a `set` and a `frozenset`

- Setdefault vs Defaultdict

- Sort by Complex Criteria Using the key Param

- Sorting collections using heapq

- In Python, adding a trailing comma is what differentiates a single value from a single-element tuple

- Why does `(1, 2) < (1, 2, -1)` return `True`?

- In Python, while you cannot modify the tuple itself, you can modify its elements if they’re mutable, like lists inside tuple

- Type annotations to perform structural subtyping (or static duck-typing).

- Understand Memory Usage and Leaks like a Pro with tracemalloc

- unpacking` and `chained assignment

- How closures work in Python, especially when they're inside list comprehensions

- Variable scope, especially in the context of function definitions inside loops

- What happens when you try to extend a tuple?

- Why do you need `__name__` in Python's main block?

- Why is string immutability useful?

- how the `with` statement works

- How to use yield in PyTorch’s getitem()

- Difference between `return` and `yield`

- `__getattr__` vs `__getattribute__`

- Understanding `__init__` and object instantiation is the foundation of all your Python OOP

- `__setitem__` and `__getitem__`

- `__getitem__` method

- cached_property decorator

- Chaining Decorators

- Dispatch decorators in Python

- Find Difference between two lists - BUT which is better for VERY LONG lists

- MASSIVELY FASTER search with bisect module in sorted list

- Parallel Processing on S3: How Python Threads Can Optimize Your Data Operations

- Private methods and name mangling

- dict[key] - Special situations

- Tracebacks in Python