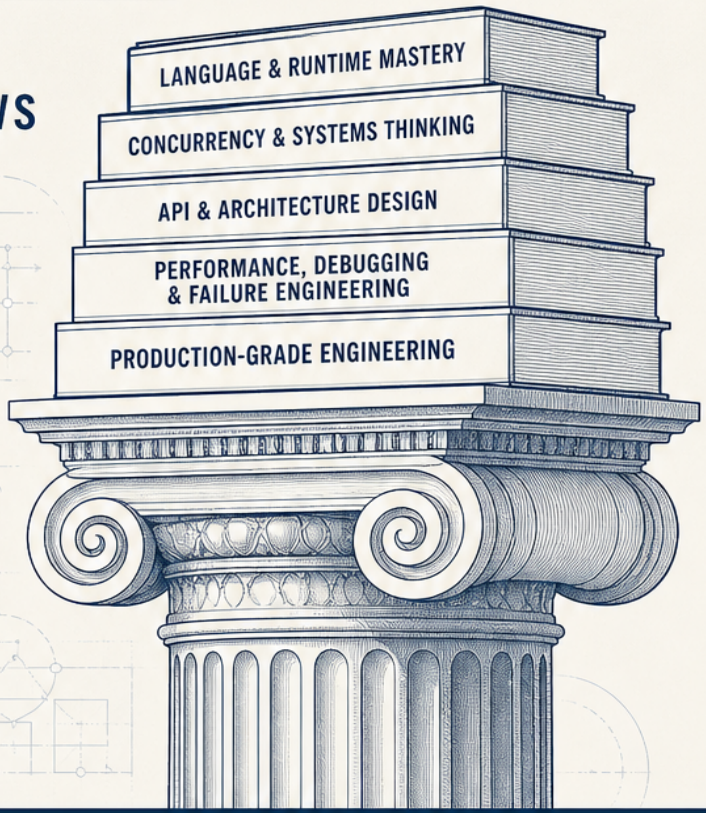
#STAFF-LEVEL PYTHON MASTERY

"Master Python at the depth expected of staff engineers."

A REFERENCE TEXTBOOK

FOR SENIOR & STAFF ENGINEERING INTERVIEWS



BUILT FOR REAL-WORLD ENGINEERS.

BACKED BY EXPERIENCE.

```
ob_refcnt
ob_type
```

```
def func(x: int) -> list[int]:
    for i in range(x):
        yield i
```

DEEP LANGUAGE KNOWLEDGE | CONCURRENCY & SYSTEMS MASTERY | ARCHITECTURE & API DESIGN | PERFORMANCE & DEBUGGING | RELIABILITY & SECURITY

FROM FOUNDATIONS TO IMPACT. FROM CODE TO SYSTEMS.

Staff Level Python Mastery - A Reference Guide

For Senior & Staff Engineering Interviews

Samyak Shah

This book is available at https://leanpub.com/python-mastery

This version was published on 2026-05-03

Contents

Chapter 1 — The Python Object Model

1.1 Everything Is an Object

In CPython, every entity the language can name — integers, strings, functions, classes, modules, `None`, even `type` itself — is a heap-allocated C struct of type `PyObject`. That struct contains at minimum two fields: a reference count (`ob_refcnt`) and a pointer to the object's type (`ob_type`). All Python-level behavior is dispatched through the type object.

```
>>> type(42)
<class 'int'>
>>> type(int)
<class 'type'>
>>> type(type)
<class 'type'>          # type is its own metaclass
>>> isinstance(int, type)
True
>>> isinstance(int, object)
True
```

The universal base: every class implicitly inherits from `object`. `type` itself inherits from `object`, and `object`'s type is `type`. This circularity is bootstrapped at interpreter startup and is a known intentional design.

Practical implication: You can attach attributes to instances of user-defined classes arbitrarily, because every such instance has a `__dict__` (unless `__slots__` is used). Functions are objects with `__name__`, `__doc__`, `__defaults__`, `__annotations__`, `__code__`, and `__closure__`. You can inspect, modify, and pass them freely.

* * *

1.2 Attribute Lookup: The Full Resolution Order

When Python evaluates `obj.attr`, it does not simply look in a dictionary. It follows a precise chain, delegating to the type system at each step.

1.2.1 The MRO (Method Resolution Order)

For inheritance, Python uses the **C3 Linearization** algorithm to compute a consistent, deterministic MRO. The MRO is stored in `ClassName.__mro__` as a tuple of classes.

```
class A: pass
class B(A): pass
class C(A): pass
class D(B, C): pass

print(D.__mro__)
# (<class 'D'>, <class 'B'>, <class 'C'>, <class 'A'>, <class 'object'>)
```

C3 guarantees: (1) a class always precedes its parents; (2) the order among parents is preserved; (3) monotonicity – if C comes before D in any class's MRO, that ordering holds throughout.

An inconsistent hierarchy raises `TypeError` at class definition time:

```
class X(A, object): pass   # fine
class Y(object, A): pass   # fine
class Z(X, Y): pass        # TypeError: Cannot create a consistent MRO
```

1.2.2 The Descriptor Protocol — The Core of Attribute Lookup

The full attribute lookup algorithm for `obj.attr` (where `type(obj)` is `T`):

1. Search `T.__mro__` for `attr`. Call the first class that defines it `B`, and the found object `descriptor`.
2. If `descriptor` is a **data descriptor** (defines both `__get__` and `__set__`, or `__get__` and `__delete__`): return `descriptor.__get__(obj, T)`. Data descriptors take priority over the instance `__dict__`.
3. If `attr` is in `obj.__dict__`: return `obj.__dict__['attr']`. Instance dictionary beats non-data descriptors.
4. If `descriptor` is a **non-data descriptor** (defines `__get__` only): return `descriptor.__get__(obj, T)`.
5. If `descriptor` exists but defines neither `__get__` nor `__set__`: return `descriptor` itself.

6. Raise `AttributeError`.

Why this matters:

- `@property` is a data descriptor (it defines `__get__`, `__set__`, and `__delete__`), which is why it takes priority over the instance `__dict__`. You cannot shadow a property by setting `self.x = ...` unless the property's `__set__` allows it.
- Functions are non-data descriptors (they define `__get__` but not `__set__`). That is how method binding works: `obj.method` calls `function.__get__(obj, type(obj))`, which returns a bound method that prepends `obj` as the first argument.
- `staticmethod` and `classmethod` are also descriptors – they override `__get__` to return the raw function or a class-bound callable respectively.

1.2.3 Writing a Descriptor

```python
class Validated:
    """Data descriptor that enforces a type constraint."""

    def __set_name__(self, owner, name):
        # Called at class creation time; `name` is the attribute name.
        self.public_name = name
        self.private_name = '_' + name

    def __get__(self, obj, objtype=None):
        if obj is None:
            # Accessed on the class itself, return the descriptor.
            return self
        return getattr(obj, self.private_name, None)

    def __set__(self, obj, value):
        self.validate(value)
        setattr(obj, self.private_name, value)

    def validate(self, value):
        raise NotImplementedError


class PositiveInt(Validated):
    def validate(self, value):
        if not isinstance(value, int) or value <= 0:
            raise ValueError(f"Expected positive int, got {value!r}")
```

```python


class Connection:
    max_retries = PositiveInt()
    timeout = PositiveInt()

    def __init__(self, max_retries, timeout):
        self.max_retries = max_retries   # calls PositiveInt.__set__
        self.timeout = timeout


c = Connection(3, 30)
c.max_retries = -1   # raises ValueError
```

Key points:

- `__set_name__` is called by `type.__new__` during class body execution; it grants the descriptor knowledge of the attribute name without manual configuration.
- When accessed on the class (`Connection.max_retries`), `obj` is `None` inside `__get__`. Returning `self` lets you inspect the descriptor itself.
- The private name convention (`_max_retries`) avoids infinite recursion: `setattr(obj, self.private_name, value)` writes directly to `obj.__dict__`, bypassing the descriptor.

* * *

1.3 Special Methods (Dunder Methods)

Special methods are looked up on the **type**, not the instance. This is a critical distinction:

```
class Weird:
    def __len__(self):
        return 42

w = Weird()
w.__len__ = lambda: 0      # patching the instance
len(w)                     # still returns 42 — type lookup, not instance
```

This prevents user-defined objects from accidentally or maliciously bypassing the protocol by patching individual instances.

1.3.1 Construction and Lifecycle

`__new__(cls, *args, **kwargs)` – allocates and returns the new instance. It is a static method implicitly. It runs before `__init__`. If it returns something that is not an instance of `cls`, `__init__` is not called.

```
class Singleton:
    _instance = None

    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super().__new__(cls)
        return cls._instance
```

`__init__(self, *args, **kwargs)` – initialises the already-allocated instance. Return value must be `None`; returning anything else raises `TypeError`.

`__del__(self)` – finalizer, called when the reference count drops to zero (or by the cyclic GC). Unreliable for resource cleanup; do not depend on it for correctness. Prefer `__exit__` via context managers.

1.3.2 Representation

`__repr__(self)` – unambiguous machine-readable string. Should ideally return a string that, when `eval`'d, reconstructs the object. Used in the REPL, `repr()`, logging, and as a fallback for `__str__`.

`__str__(self)` – human-readable string. Used by `str()`, `print()`, and f-string `{obj}`. Falls back to `__repr__` if not defined.

`__format__(self, format_spec)` – called by `format()` and f-string `{obj:spec}`. Allows custom format mini-languages (see `datetime`, `Decimal`).

```python
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return f"Vector({self.x!r}, {self.y!r})"

    def __str__(self):
        return f"({self.x}, {self.y})"

    def __format__(self, spec):
        if spec == 'polar':
            r = (self.x**2 + self.y**2) ** 0.5
            return f"|{r:.3f}|"
        return str(self)
```

1.3.3 Comparison and Hashing

The **hash/equality contract**: objects that compare equal must have the same hash. Formally: if `a == b` then `hash(a) == hash(b)`. The converse is not required.

In Python:

- If you define `__eq__`, Python sets `__hash__` to `None` by default (making the class unhashable), because Python cannot know if your equality relation is compatible with any hash function.
- You must explicitly define `__hash__` alongside `__eq__`.
- If you inherit `__eq__` from a parent and want the inherited `__hash__`, you must explicitly restore it: `__hash__ = ParentClass.__hash__`.

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __eq__(self, other):
        if not isinstance(other, Point):
            return NotImplemented
        return self.x == other.x and self.y == other.y

    def __hash__(self):
        return hash((self.x, self.y))
```

Returning `NotImplemented` (not `NotImplementedError`) from comparison operators tells Python to try the reflected operation on the other operand. This is required for correct interoperability:

```python
class MyNum:
    def __eq__(self, other):
        if not isinstance(other, MyNum):
            return NotImplemented   # let `other.__eq__(self)` be tried
        ...
```

`functools.total_ordering` – if you define `__eq__` and one of `__lt__`, `__le__`, `__gt__`, `__ge__`, it fills in the rest. Use with care: it has a performance overhead and is disabled in some internal CPython types. Acceptable in interview code.

```python
from functools import total_ordering

@total_ordering
class Priority:
    def __init__(self, value):
        self.value = value

    def __eq__(self, other):
        return self.value == other.value

    def __lt__(self, other):
        return self.value < other.value

    # __le__, __gt__, __ge__ are auto-generated
```

1.3.4 Container Emulation

```python
class RingBuffer:
    """Fixed-size circular buffer demonstrating container protocol."""

    def __init__(self, capacity):
        self._capacity = capacity
        self._buffer = [None] * capacity
        self._head = 0
        self._size = 0

    def append(self, item):
        idx = (self._head + self._size) % self._capacity
        if self._size < self._capacity:
            self._buffer[idx] = item
            self._size += 1
        else:
            # Overwrite oldest
            self._buffer[self._head] = item
            self._head = (self._head + 1) % self._capacity

    def __len__(self):
        return self._size

    def __getitem__(self, index):
        if index < 0:
            index += self._size
        if not (0 <= index < self._size):
            raise IndexError("index out of range")
        return self._buffer[(self._head + index) % self._capacity]

    def __contains__(self, item):
        # O(n); override for custom membership semantics
        return any(self[i] == item for i in range(self._size))

    def __iter__(self):
        for i in range(self._size):
            yield self[i]

    def __repr__(self):
        return f"RingBuffer({list(self)!r})"
```

Notes:

- `__len__` must return a non-negative integer. `bool(obj)` falls back to `__len__` if `__bool__` is not defined; empty containers are falsy.
- `__contains__` is optional; Python falls back to iterating via `__iter__` if absent.
- `__iter__` returning a generator is idiomatic and sufficient for most

use cases. For persistent iterators (multiple independent iterations), implement `__iter__` to return a new iterator object each time.

- `__getitem__` with integer indices and raising `IndexError` is sufficient for `for` loops even without `__iter__` (the legacy iteration protocol).

1.3.5 Callable Objects

```
class Memoize:
    """Callable object wrapping a function with a result cache."""

    def __init__(self, func):
        self._func = func
        self._cache = {}
        # Preserve metadata for introspection
        import functools
        functools.update_wrapper(self, func)

    def __call__(self, *args):
        if args not in self._cache:
            self._cache[args] = self._func(*args)
        return self._cache[args]

    def cache_clear(self):
        self._cache.clear()
```

1.3.6 Context Managers

The context manager protocol: `__enter__` runs on entry to the `with` block and its return value is bound to the `as` target; `__exit__(self, exc_type, exc_val, exc_tb)` runs on exit. If `__exit__` returns a truthy value, any exception is suppressed.

```python
class ManagedConnection:
    def __init__(self, dsn):
        self.dsn = dsn
        self.conn = None

    def __enter__(self):
        self.conn = connect(self.dsn)   # hypothetical
        return self.conn                # bound to `as` target

    def __exit__(self, exc_type, exc_val, exc_tb):
        if exc_type is not None:
            self.conn.rollback()
        else:
            self.conn.commit()
        self.conn.close()
        return False   # do not suppress exceptions
```

Suppression example — useful in tests or resilient loops:

```python
def __exit__(self, exc_type, exc_val, exc_tb):
    if exc_type is KeyError:
        return True    # suppress only KeyError
    return False
```

`contextlib.contextmanager` transforms a generator function into a context manager without a class:

```python
from contextlib import contextmanager

@contextmanager
def temp_directory():
    import tempfile, shutil
    path = tempfile.mkdtemp()
    try:
        yield path
    finally:
        shutil.rmtree(path)
```

The code before `yield` is `__enter__`; the `yield` value is the `as` target; the code after is `__exit__`. The `finally` ensures cleanup even on exception.

* * *

1.4 `__slots__`

By default, each instance of a user-defined class has a `__dict__` – a hash table allocated per instance. For classes with many instances and a fixed set of attributes, this wastes significant memory.

`__slots__` replaces `__dict__` with a fixed set of slot descriptors stored at the class level:

```
class Point:
    __slots__ = ('x', 'y')

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Consequences of `__slots__`:

- No `__dict__` on instances (unless `'__dict__'` is listed in `__slots__`).
- No `__weakref__` unless `'__weakref__'` is listed.
- Cannot add arbitrary attributes at runtime.
- Memory reduction is approximately 40–50% for small objects in CPython (varies by attribute count).
- **Inheritance caveat:** If a parent class does not use `__slots__`, the subclass still has `__dict__` (inherited from the parent). `__slots__` only blocks `__dict__` creation when the entire MRO uses it.

```
import sys

class WithDict:
    def __init__(self):
        self.x = 1
        self.y = 2

class WithSlots:
    __slots__ = ('x', 'y')
    def __init__(self):
        self.x = 1
        self.y = 2

print(sys.getsizeof(WithDict()))    # ~48 bytes + dict overhead (~232 bytes)
print(sys.getsizeof(WithSlots()))   # ~56 bytes (slots stored compactly)
```

* * *

1.5 Function Objects in Depth

Functions in Python are first-class objects of type `function`. Their key attributes:

Attribute	Contents
`__name__`	Unqualified function name
`__qualname__`	Qualified name (includes enclosing scope, e.g., `Class.method`)
`__doc__`	Docstring
`__module__`	Module in which the function was defined
`__defaults__`	Tuple of default values for positional params; `None` if none
`__kwdefaults__`	Dict of default values for keyword-only params
`__annotations__`	Dict of parameter/return annotations
`__code__`	Code object (contains bytecode, constants, local variable names)
`__closure__`	Tuple of cell objects for free variables; `None` if no closure
`__globals__`	Reference to the module's global namespace dict

1.5.1 The Mutable Default Argument Trap

Default values are evaluated **once**, at function definition time, and stored in `__defaults__`. Mutable defaults are shared across all calls:

```python
def append_to(element, target=[]):   # BAD
    target.append(element)
    return target

append_to(1)   # [1]
append_to(2)   # [1, 2]  — same list!

def append_to(element, target=None):  # CORRECT
    if target is None:
        target = []
    target.append(element)
    return target
```

The fix uses `None` as the sentinel. This pattern is idiomatic Python.

1.5.2 Closures

A closure captures references to variables in the enclosing scope via cell objects. The reference is live — mutations in the enclosing scope are visible:

```python
def make_counter():
    count = 0
    def increment():
        nonlocal count
        count += 1
        return count
    return increment

c = make_counter()
c()   # 1
c()   # 2
```

`nonlocal` is required to rebind a variable in the enclosing scope. Without it, an assignment inside the inner function creates a new local variable (shadowing), and reading before assignment raises `UnboundLocalError`.

Classic closure-in-loop bug:

```python
fns = [lambda: i for i in range(3)]
[f() for f in fns]   # [2, 2, 2] — all capture the same `i`

# Fix: bind current value as a default argument
fns = [lambda i=i: i for i in range(3)]
[f() for f in fns]   # [0, 1, 2]
```

* * *

1.6 Attribute Access Hooks

1.6.1 `__getattr__` vs `__getattribute__`

- **`__getattribute__`** – called on *every* attribute access. Overriding it incorrectly causes infinite recursion. Call `super().__getattribute__(name)` to use the default.
- **`__getattr__`** – called only when the *normal lookup fails* (i.e., after `__getattribute__` raises `AttributeError`). This is the safe hook for computed or lazy attributes.

```python
class LazyLoader:
    """Loads a module attribute on first access."""

    def __init__(self, module_name):
        object.__setattr__(self, '_module_name', module_name)
        object.__setattr__(self, '_module', None)

    def _load(self):
        import importlib
        mod = importlib.import_module(self._module_name)
        object.__setattr__(self, '_module', mod)
        return mod

    def __getattr__(self, name):
        # Only called if `name` not found normally
        mod = self._module or self._load()
        return getattr(mod, name)
```

1.6.2 `__setattr__` and `__delattr__`

`__setattr__` is called on *every* `self.x = value`. Must use `object.__setattr__(self, name, value)` internally to avoid infinite recursion.

```python
class Immutable:
    def __init__(self, **kwargs):
        for k, v in kwargs.items():
            object.__setattr__(self, k, v)

    def __setattr__(self, name, value):
        raise AttributeError("Immutable object")

    def __delattr__(self, name):
        raise AttributeError("Immutable object")
```

* * *

1.7 `@property`, `@staticmethod`, `@classmethod`

All three are implemented as descriptors.

`@property`

```python
class Circle:
    def __init__(self, radius):
        self._radius = radius

    @property
    def radius(self):
        return self._radius

    @radius.setter
    def radius(self, value):
        if value < 0:
            raise ValueError("Radius must be non-negative")
        self._radius = value

    @radius.deleter
    def radius(self):
        del self._radius
```

```
18
19      @property
20      def area(self):
21          import math
22          return math.pi * self._radius ** 2
```

`@property` creates a data descriptor. The getter, setter, and deleter are stored as attributes of the property object. The `@radius.setter` syntax returns a *new* property object with the setter attached – it does not mutate the original.

`@classmethod`

Receives the class as first argument (conventionally `cls`). Used for alternative constructors:

```
class Date:
    def __init__(self, year, month, day):
        self.year, self.month, self.day = year, month, day

    @classmethod
    def from_iso_string(cls, s):
        # Works correctly on subclasses too – `cls` is the actual class
        year, month, day = map(int, s.split('-'))
        return cls(year, month, day)
```

`@staticmethod`

Receives neither the instance nor the class. A plain function namespaced to the class:

```
class Validator:
    @staticmethod
    def is_valid_email(email):
        return '@' in email and '.' in email.split('@')[1]
```

Choose `@staticmethod` when the method does not need access to class or instance state. Choose `@classmethod` when it needs the class (especially for subclass-aware construction). Choose a module-level function when neither is needed – avoid over-encapsulating into a class.

* * *

1.8 `__init_subclass__` and Class Hooks

Introduced in Python 3.6, `__init_subclass__` runs when a class is subclassed:

```
class Plugin:
    _registry = {}

    def __init_subclass__(cls, plugin_name=None, **kwargs):
        super().__init_subclass__(**kwargs)
        if plugin_name:
            Plugin._registry[plugin_name] = cls


class CSVPlugin(Plugin, plugin_name='csv'):
    pass

class JSONPlugin(Plugin, plugin_name='json'):
    pass

print(Plugin._registry)
# {'csv': <class 'CSVPlugin'>, 'json': <class 'JSONPlugin'>}
```

This is a lightweight alternative to metaclasses for registration patterns. The keyword argument `plugin_name` is passed at class definition time, not at instantiation.

* * *

1.9 The `object.__class__` and `type.__instancecheck__`

`isinstance(obj, cls)` is not simply `type(obj) is cls`. It calls `cls.__instancecheck__(obj)`, which is defined on the metaclass. This is how Abstract Base Classes implement virtual subclassing:

```python
from abc import ABCMeta

class Sequence(metaclass=ABCMeta):
    @classmethod
    def __subclasshook__(cls, C):
        # Return True if C should be considered a Sequence
        # even without inheriting from Sequence
        if '__getitem__' in C.__dict__ and '__len__' in C.__dict__:
            return True
        return NotImplemented
```

`__subclasshook__` is called by `ABCMeta.__subclasscheck__`. If it returns `True`, `issubclass(C, Sequence)` is `True` without explicit inheritance. This enables structural subtyping in Python's ABCs.

* * *

1.10 Object Copying

Shallow copy (`copy.copy`) – creates a new object of the same type with the same top-level attribute values. Nested objects are *not* duplicated – both copies point to the same nested objects.

Deep copy (`copy.deepcopy`) – recursively duplicates the entire object graph. Handles circular references using a memo dictionary. Expensive; avoid for large graphs.

```python
import copy

class Graph:
    def __init__(self, nodes):
        self.nodes = nodes   # list of Node objects

g1 = Graph([Node(1), Node(2)])
g2 = copy.copy(g1)      # g2.nodes is the same list object as g1.nodes
g3 = copy.deepcopy(g1)  # g3.nodes is a new list with new Node copies
```

Customising:

```python
class Node:
    def __copy__(self):
        # Custom shallow copy
        new = Node.__new__(Node)
        new.__dict__.update(self.__dict__)
        return new

    def __deepcopy__(self, memo):
        # memo maps id(original) -> copy; prevents infinite loops on cycles
        new = Node.__new__(Node)
        memo[id(self)] = new
        for k, v in self.__dict__.items():
            setattr(new, k, copy.deepcopy(v, memo))
        return new
```

* * *

1.11 Identity, Equality, and Interning

- `is` tests object identity (`id(a) == id(b)`), not value equality.
- `==` calls `__eq__`.
- CPython interns small integers (-5 to 256) and many string literals; do not rely on this behavior in production code.

```python
a = 256
b = 256
a is b   # True  — interned

a = 257
b = 257
a is b   # False — not guaranteed (though often True in same code block)

# String interning
s1 = 'hello'
s2 = 'hello'
s1 is s2   # True — interned at compile time

s1 = ''.join(['h','e','l','l','o'])
s2 = 'hello'
s1 is s2   # False — runtime-constructed string
```

The only safe uses of `is`: testing against singletons (`None`, `True`, `False`, `NotImplemented`, `Ellipsis`).

* * *

Summary of Key Principles

Concept	Rule
Attribute lookup	Data descriptor > instance `__dict__` > non-data descriptor
`__eq__` and `__hash__`	Always define both together; return `NotImplemented` not `False` for unknown types
`__repr__`	Should be unambiguous and ideally eval-able
`__getattr__`	Only fires on failure; use for lazy/dynamic attributes
`__getattribute__`	Fires on every access; override with extreme care
`__slots__`	Reduces memory; requires all classes in MRO to use it
Default arguments	Use `None` sentinel, not mutable objects
`is` operator	Only valid against singletons
Closures	Capture by reference; use default-argument trick for loop variables

* * *

Chapter 2 — The Type System & Type Hints

* * *

2.1 The Purpose of Type Hints in Python

Python's type system is **gradual**: type annotations are optional, carry no runtime enforcement by default, and coexist with fully untyped code. The type checker (mypy, pyright, pytype) operates as a separate offline pass — it reads the annotations and the code's structure, then reports inconsistencies without ever running the program.

This design means:

- Annotations can be added incrementally to existing codebases.
- You can annotate some functions and leave others as `Any`.
- Runtime behavior is completely unaffected by annotations (with minor exceptions noted below).

The primary value of type hints:

1. **Documentation** — a function signature is a contract that tools can verify.
2. **IDE support** — auto-complete, refactoring, and inline error detection.
3. **Correctness** — catching bugs at the boundary of system components before they manifest at runtime.
4. **Maintainability** — large codebases become navigable without reading every implementation.

At a staff level you are expected to write fully annotated code, reason about type correctness, and understand the limits of the type system.

* * *

2.2 Annotation Mechanics

2.2.1 How Annotations Are Stored

Function annotations are stored in `__annotations__` as a dict:

```
def fetch(url: str, timeout: float = 5.0) -> bytes:
    ...

fetch.__annotations__
# {'url': <class 'str'>, 'timeout': <class 'float'>, 'return': <class 'bytes'>}
```

Variable annotations at module or class scope are stored in the module or class `__annotations__`. Annotations inside function bodies are evaluated but **not** stored anywhere — they exist only for the type checker.

```
class Config:
    host: str
    port: int = 8080

Config.__annotations__
# {'host': <class 'str'>, 'port': <class 'int'>}
```

2.2.2 `from __future__ import annotations`

By default, annotations are evaluated eagerly at definition time. This causes problems with forward references (a class referencing itself) and with annotations that import symbols not yet defined.

```
# Without future annotations — fails at runtime
class Node:
    def children(self) -> list[Node]:  # NameError: Node not yet defined
        ...
```

The fix before Python 3.10 was to quote the annotation: `-> 'list[Node]'`. From Python 3.10+ with `from __future__ import annotations` (or natively in 3.11+), all annotations are stored as **strings** (lazy evaluation):

```python
from __future__ import annotations

class Node:
    def children(self) -> list[Node]:  # fine – stored as the string
    ↪ "list[Node]"
        ...
```

`typing.get_type_hints(obj)` resolves these strings back to types when needed, using the appropriate namespace. `__annotations__` itself returns raw strings.

Practical rule: Always use `from __future__ import annotations` in modules with any non-trivial type hints. It also improves import performance by deferring annotation evaluation.

* * *

2.3 Core Typing Constructs

2.3.1 Basic Builtins vs `typing` Aliases

From Python 3.9+, built-in collection types are directly parameterizable:

```python
# Python 3.9+
def process(items: list[int]) -> dict[str, list[int]]:
    ...

# Pre-3.9 (still works everywhere, but verbose)
from typing import List, Dict
def process(items: List[int]) -> Dict[str, List[int]]:
    ...
```

The capitalized `typing.List`, `typing.Dict`, etc. are aliases that exist purely for backward compatibility. Prefer the lowercase builtins in new code targeting 3.9+.

2.3.2 `Optional` and `Union`

`Optional[X]` is precisely `Union[X, None]`. Nothing more.

```python
from typing import Optional, Union

def find(key: str) -> Optional[str]:      # may return None
    ...

def parse(val: Union[str, int]) -> float: # accepts str or int
    ...

# Python 3.10+ — cleaner union syntax using |
def find(key: str) -> str | None:
    ...

def parse(val: str | int) -> float:
    ...
```

Design note: `Optional` return types force callers to handle `None`. This is correct when `None` is a meaningful absence. If `None` signals an error, raise an exception instead. Proliferating `Optional` through a codebase creates chains of `if x is not None` checks; consider a `Result` type or exceptions.

2.3.3 Any

`Any` is a special form that is simultaneously a subtype and supertype of every type. It is the escape hatch from the type system:

```python
from typing import Any

def legacy_function(data: Any) -> Any:
    ...
```

A value typed as `Any`:

- Can be passed where any type is expected.
- Can have any attribute accessed without error.
- Can be called with any arguments.

`Any` is contagious: operations on `Any` values return `Any`. This is by design – once you leave the type system, the checker cannot reason further. Minimize `Any` to integration boundaries with untyped libraries.

`object` is not `Any`. `object` is the actual root base class; the type checker will only permit operations defined on `object`. `Any` disables checking entirely.

2.3.4 `Union` Narrowing

The type checker tracks which branch of a union is active through control flow analysis:

```
def process(val: str | int) -> str:
    if isinstance(val, str):
        return val.upper()   # checker knows val is str here
    else:
        return str(val * 2)  # checker knows val is int here
```

This is called **type narrowing**. Supported narrowing forms:

- `isinstance(x, T)` – narrows to `T`
- `x is None` / `x is not None` – narrows `Optional[T]`
- `assert isinstance(x, T)` – narrows after the assertion
- **`TypeGuard`** – user-defined narrowing functions

```
from typing import TypeGuard

def is_str_list(val: list[object]) -> TypeGuard[list[str]]:
    return all(isinstance(x, str) for x in val)

def process(val: list[object]) -> None:
    if is_str_list(val):
        # val is narrowed to list[str] here
        print(val[0].upper())
```

2.3.5 `Literal`

Constrains a value to a specific set of literal values:

```python
from typing import Literal

def open_file(path: str, mode: Literal['r', 'w', 'a', 'rb', 'wb']) -> None:
    ...

open_file('data.txt', 'r')    # ok
open_file('data.txt', 'x')    # type error
```

Useful for discriminated unions, configuration options, and any argument where the set of valid values is small and enumerable. Prefer `Literal` over raw `str` for such parameters.

2.3.6 `Final` and `ClassVar`

```python
from typing import Final, ClassVar

MAX_CONNECTIONS: Final = 100        # cannot be reassigned
MAX_CONNECTIONS = 200               # type error

class Config:
    instance_count: ClassVar[int] = 0   # class variable, not instance
    name: str                           # instance variable
```

`Final` prevents reassignment. It does not make the object immutable – a `Final` list can still be mutated. For immutability, use `tuple`, `frozenset`, or `dataclasses.dataclass(frozen=True)`.

`ClassVar` tells the type checker that an annotation describes a class-level variable, preventing accidental assignment through an instance.

* * *

2.4 Generics

2.4.1 `TypeVar`

A `TypeVar` is a placeholder for a type that is determined at call time. It links types across a function signature:

```python
from typing import TypeVar

T = TypeVar('T')

def first(seq: list[T]) -> T:
    return seq[0]

x: int   = first([1, 2, 3])     # T is inferred as int
y: str   = first(['a', 'b'])    # T is inferred as str
z: float = first([1, 2, 3])     # type error: int is not float
```

The string name passed to `TypeVar` should match the variable name – this is a convention the type checker uses in error messages.

Constrained TypeVar – restricts T to a specific set of types:

```python
Numeric = TypeVar('Numeric', int, float, complex)

def square(x: Numeric) -> Numeric:
    return x * x
```

Bounded TypeVar – T must be a subtype of the bound:

```python
from typing import TypeVar
from collections.abc import Hashable

H = TypeVar('H', bound=Hashable)

def add_to_set(s: set[H], item: H) -> set[H]:
    return s | {item}
```

Constrained means "exactly one of these types." Bounded means "this type or any subtype."

2.4.2 Generic Classes

```python
from typing import TypeVar, Generic

T = TypeVar('T')
E = TypeVar('E', bound=Exception)

class Result(Generic[T, E]):
    """
    Explicit Result type: either a value of type T or an error of type E.
    More explicit than Optional; useful for APIs that must not raise.
    """

    def __init__(self, value: T | None = None, error: E | None = None):
        self._value = value
        self._error = error

    @classmethod
    def ok(cls, value: T) -> 'Result[T, E]':
        return cls(value=value)

    @classmethod
    def err(cls, error: E) -> 'Result[T, E]':
        return cls(error=error)

    def is_ok(self) -> bool:
        return self._error is None

    def unwrap(self) -> T:
        if self._error is not None:
            raise self._error
        return self._value  # type: ignore[return-value]

    def unwrap_or(self, default: T) -> T:
        return self._value if self.is_ok() else default
```

When subclassing a generic class, you can specialize or remain generic:

```python
class IntResult(Result[int, ValueError]):
    # Specialized: T=int, E=ValueError
    pass

class AnyResult(Result[T, Exception]):  # Still generic in T
    pass
```

2.4.3 `ParamSpec` and `Concatenate` (Python 3.10+)

`TypeVar` cannot capture an entire parameter list. `ParamSpec` fills this gap – it is essential for correctly annotating decorators:

```python
from typing import ParamSpec, Callable, TypeVar
import functools

P = ParamSpec('P')
R = TypeVar('R')

def logged(func: Callable[P, R]) -> Callable[P, R]:
    @functools.wraps(func)
    def wrapper(*args: P.args, **kwargs: P.kwargs) -> R:
        print(f"Calling {func.__name__}")
        result = func(*args, **kwargs)
        print(f"Done {func.__name__}")
        return result
    return wrapper

@logged
def fetch(url: str, timeout: float) -> bytes:
    ...

fetch('http://example.com', 5.0)   # type checker knows full signature
```

Without `ParamSpec`, the inner `wrapper` would be typed as `(*args: Any, **kwargs: Any) -> R`, losing all parameter type information.

2.4.4 `TypeVarTuple` (Python 3.11+)

Enables variadic generics – typing heterogeneous tuples of arbitrary length (used internally by `tuple`, `zip`, etc.):

```python
from typing import TypeVarTuple, Unpack

Ts = TypeVarTuple('Ts')

def broadcast(func: Callable[[Unpack[Ts]], None], *args: Unpack[Ts]) -> None:
    func(*args)
```

This is advanced and rarely needed in application code, but appears in library internals.

* * *

2.5 Structural Typing with `Protocol`

`Protocol` enables **structural subtyping** (duck typing made explicit). A class satisfies a `Protocol` if it implements the required methods and attributes, regardless of inheritance:

```
from typing import Protocol, runtime_checkable

class Closeable(Protocol):
    def close(self) -> None:
        ...

class Serializable(Protocol):
    def to_bytes(self) -> bytes:
        ...
    def from_bytes(self, data: bytes) -> None:
        ...

@runtime_checkable
class Drawable(Protocol):
    def draw(self) -> None:
        ...

class Circle:
    def draw(self) -> None:
        print("Drawing circle")

# No inheritance — structurally satisfies Drawable
def render(obj: Drawable) -> None:
    obj.draw()

render(Circle())   # type checker: ok
```

`@runtime_checkable` allows `isinstance(obj, Drawable)` at runtime, but only checks for method presence (not signatures). Without it, `Protocol` is purely a static artifact.

Protocol vs ABC:

- `Protocol` — structural; no explicit registration or inheritance required; better for functions that accept anything with a given shape.
- `ABC` — nominal; requires `register()` or explicit inheritance; better when you want to enforce a full contract via `abstractmethod`.

Protocol with attributes:

```python
class HasName(Protocol):
    name: str   # Structural: any object with a `name: str` attribute qualifies
```

* * *

2.6 `TypedDict`

A `TypedDict` defines the expected structure of a dict without converting it to a class:

```python
from typing import TypedDict, Required, NotRequired

class UserRecord(TypedDict):
    id: int
    name: str
    email: str
    role: NotRequired[str]   # optional key

def create_user(record: UserRecord) -> None:
    print(record['name'])

create_user({'id': 1, 'name': 'Alice', 'email': 'a@b.com'})   # ok
create_user({'id': 1, 'name': 'Alice'})   # type error: missing 'email'
```

`TypedDict` is useful at I/O boundaries – JSON payloads, API responses, config files – where the data is genuinely a dict and converting it to a dataclass would be overhead. It does not provide runtime validation.

Inheritance – `TypedDict` supports inheritance to compose schemas:

```python
class BaseRecord(TypedDict):
    id: int
    created_at: str

class UserRecord(BaseRecord):
    name: str
    email: str
```

* * *

2.7 `dataclass` vs `NamedTuple` vs `TypedDict`

Choosing the right tool:

	`dataclass`	`NamedTuple`	`TypedDict`
Type	Class instance	Tuple subclass	Plain dict
Mutable	Yes (default)	No	Yes
Hashable	No (unless `frozen=True` + no mutable fields)	Yes	No
Inheritance	Full	Limited	Yes
Runtime overhead	Low	Very low	None
Positional unpacking	No	Yes	No
`__slots__`	Via `slots=True` (3.10+)	Built-in	N/A
Use case	Mutable domain objects	Lightweight value objects	Dict schema at I/O boundaries

```python
from dataclasses import dataclass, field
from typing import NamedTuple

@dataclass(frozen=True, slots=True)
class Point:
    x: float
    y: float

class Coordinate(NamedTuple):
    latitude: float
    longitude: float

# NamedTuple: positional access, tuple semantics
lat, lon = Coordinate(51.5, -0.1)   # tuple unpacking
```

2.7.1 `dataclass` in Depth

```python
from dataclasses import dataclass, field, KW_ONLY

@dataclass(order=True, frozen=True, slots=True)
class Version:
    major: int
    minor: int
    patch: int

    def __str__(self) -> str:
        return f"{self.major}.{self.minor}.{self.patch}"

@dataclass
class Pipeline:
    name: str
    _: KW_ONLY                          # everything after this is keyword-only
    stages: list[str] = field(default_factory=list)
    metadata: dict[str, str] = field(default_factory=dict, repr=False)
    _id: int = field(default=0, init=False, repr=False)  # not in __init__

    def __post_init__(self):
        # Runs after __init__; use for validation and derived fields
        if not self.name:
            raise ValueError("Pipeline name cannot be empty")
        object.__setattr__(self, '_id', id(self))  # needed if frozen=True
```

`field()` parameters:

- `default` / `default_factory` – use `default_factory` for mutable defaults.
- `init=False` – exclude from `__init__`; set in `__post_init__`.
- `repr=False` – exclude from `__repr__`.
- `compare=False` – exclude from `__eq__` and ordering methods.
- `hash=None` – follows `compare` by default.

* * *

2.8 `@overload`

When a function has multiple valid signatures with different return types depending on argument types, use `@overload` to provide the checker with each signature. The implementation uses a final non-decorated definition:

```python
from typing import overload

@overload
def parse(data: str) -> dict: ...
@overload
def parse(data: bytes) -> dict: ...
@overload
def parse(data: dict) -> dict: ...

def parse(data: str | bytes | dict) -> dict:
    if isinstance(data, dict):
        return data
    if isinstance(data, bytes):
        data = data.decode()
    import json
    return json.loads(data)
```

The `@overload` variants are invisible at runtime — they are erased and only the final implementation exists. The type checker uses the overloads to resolve calls.

When to use `@overload`:

- Return type varies based on argument type.
- Combinations of optional arguments that are mutually exclusive.
- When `Union` return types would be too permissive.

* * *

2.9 Abstract Base Classes and `typing.abc`

`collections.abc` provides abstract base classes for container protocols. These interact with the type system:

```python
from collections.abc import (
    Iterable, Iterator, Generator,
    Sequence, MutableSequence,
    Mapping, MutableMapping,
    Callable, Awaitable, Coroutine,
    AsyncIterable, AsyncIterator
)
```

Prefer these over concrete types in function signatures:

```python
# BAD – overly restrictive; rejects tuples, generators, etc.
def process(items: list[int]) -> None: ...

# GOOD – accepts anything iterable
def process(items: Iterable[int]) -> None: ...

# GOOD – accepts any sequence (supports len and indexing)
def process(items: Sequence[int]) -> None: ...
```

The rule: **accept the most general type your implementation actually needs; return the most specific type your implementation actually produces.**

2.9.1 Annotating Iterators and Generators

```python
from collections.abc import Iterator, Generator

def count_up(start: int) -> Iterator[int]:
    n = start
    while True:
        yield n
        n += 1

# Generator[YieldType, SendType, ReturnType]
def accumulator() -> Generator[float, float, str]:
    total = 0.0
    while True:
        value = yield total   # yields float, receives float via send()
        if value is None:
            return f"Final total: {total}"
        total += value
```

`Iterator[T]` is equivalent to `Generator[T, None, None]`. Use `Iterator[T]` when you only yield. Use `Generator[Y, S, R]` only when `send()` or `return` values matter.

* * *

2.10 Annotating Callables

```python
from collections.abc import Callable

# Callable[[arg_types...], return_type]
def apply(func: Callable[[int, int], int], x: int, y: int) -> int:
    return func(x, y)

# Callable with no arguments
def run(task: Callable[[], None]) -> None:
    task()

# Callable with unknown signature — Callable[..., R]
def wrap(func: Callable[..., int]) -> Callable[..., int]:
    ...
```

`Callable[..., R]` means "callable returning R, any parameters." Use this when you genuinely don't care about parameter types (e.g., a retry decorator that wraps any function).

* * *

2.11 Runtime Type Checking

Type annotations do not enforce types at runtime. For runtime validation:

Option 1: Manual assertions (fine for internal code):

```python
def connect(host: str, port: int) -> None:
    assert isinstance(host, str), f"Expected str, got {type(host)}"
    assert isinstance(port, int) and 1 <= port <= 65535
```

Option 2: `dataclass` with `__post_init__` (good for domain objects):

```python
@dataclass
class Config:
    host: str
    port: int

    def __post_init__(self):
        if not isinstance(self.host, str):
            raise TypeError(f"host must be str, got {type(self.host)}")
        if not (1 <= self.port <= 65535):
            raise ValueError(f"port must be 1-65535, got {self.port}")
```

Option 3: `pydantic` (the de facto standard for API-boundary validation):

```python
from pydantic import BaseModel, Field

class Config(BaseModel):
    host: str
    port: int = Field(ge=1, le=65535)
    timeout: float = 5.0

cfg = Config(host='localhost', port='8080')  # port coerced to int
cfg = Config(host='localhost', port=99999)   # ValidationError
```

Pydantic coerces types where possible and raises `ValidationError` with detailed field-level messages. It is the correct tool for parsing and validating external data (HTTP bodies, config files, CLI args).

* * *

2.12 `mypy` Configuration and Usage Patterns

2.12.1 Key `mypy` Flags

```ini
# mypy.ini
[mypy]
python_version = 3.11
strict = True          # enables all strictness flags

# Individual flags (subset of what --strict enables)
warn_return_any = True          # warn when returning Any
warn_unused_ignores = True      # warn about unnecessary # type: ignore
disallow_untyped_defs = True    # require annotations on all functions
disallow_any_generics = True    # disallow bare List, Dict without params
check_untyped_defs = True      # type-check function bodies without annotations
no_implicit_optional = True     # Optional[X] must be explicit; X | None = ...
↪  no longer allowed as default
```

2.12.2 `# type: ignore` and `cast`

```python
from typing import cast

# Suppress a single error — use sparingly, document why
value = legacy_function()   # type: ignore[return-value]

# Inform the type checker of a type it cannot infer
result = cast(list[str], some_dynamic_function())
```

`cast(T, x)` is a no-op at runtime — it returns `x` unchanged. It is a type-checker-only assertion. Use it at well-understood boundaries where you know more than the checker. Never use it to mask a genuine type error.

2.12.3 Stubs and `py.typed`

For packages you distribute, signal that you support type checking by:

1. Including a `py.typed` marker file (PEP 561).
2. Providing inline type annotations in your source.

For untyped third-party libraries, use stub files (`.pyi`) from `typeshed` or from `types-<library>` packages on PyPI.

* * *

2.13 Common Typing Mistakes and Misconceptions

Mistake 1: Conflating `Optional[X]` with "optional parameter"

```python
# WRONG interpretation: this does NOT mean `name` is optional
def greet(name: Optional[str]) -> str:
    return f"Hello, {name}"    # mypy: name could be None

# Correct: optional parameter with a default
def greet(name: str = 'World') -> str:
    return f"Hello, {name}"
```

Mistake 2: Using mutable containers as TypeVar constraints

```python
T = TypeVar('T')

def first_or_none(seq: list[T]) -> T | None:
    return seq[0] if seq else None

# T is bound per-call; the list[T] annotation is fine.
# Problem arises if you try to mutate the list assuming a specific type.
```

Mistake 3: Annotating `self` incorrectly in classmethods

```python
from typing import Self   # Python 3.11+

class Builder:
    def set_name(self, name: str) -> Self:
        self.name = name
        return self     # Self preserves the actual subclass type

class AdvancedBuilder(Builder):
    pass

b: AdvancedBuilder = AdvancedBuilder().set_name("x")  # correctly typed
```

Without `Self`, the return type would be `Builder`, losing the subclass information.

Mistake 4: Using `List` (capital) from `typing` in new code

In Python 3.9+, `list[int]` is fully supported. `typing.List` is deprecated. Use the lowercase builtins.

Mistake 5: Thinking `isinstance` is slow

It is not. `isinstance` is a C-level operation in CPython, O(depth of MRO), and fully appropriate in hot paths. The performance concern applies to deeply nested class hierarchies, not typical user code.

* * *

2.14 Typing Patterns for Interview Code

At a staff level, interviewers expect to see type annotations used confidently. The following patterns cover the most common scenarios:

```python
from __future__ import annotations
from typing import TypeVar, Generic, Protocol, overload, Final
from collections.abc import Iterator, Callable, Iterable, Sequence
from dataclasses import dataclass, field

# 1. Annotating a generic container
T = TypeVar('T')

@dataclass
class Stack(Generic[T]):
    _items: list[T] = field(default_factory=list)

    def push(self, item: T) -> None:
        self._items.append(item)

    def pop(self) -> T:
        if not self._items:
            raise IndexError("pop from empty stack")
        return self._items.pop()

    def peek(self) -> T | None:
        return self._items[-1] if self._items else None

    def __len__(self) -> int:
        return len(self._items)

    def __iter__(self) -> Iterator[T]:
        return iter(reversed(self._items))

# 2. Protocol for dependency injection
class HtmlParser(Protocol):
```

```python
    def get_urls(self, url: str) -> list[str]: ...

# 3. TypedDict for structured external data
from typing import TypedDict

class CrawlResult(TypedDict):
    url: str
    status: int
    links: list[str]
    error: str | None

# 4. Overload for multi-signature function
@overload
def load_config(path: str) -> dict[str, str]: ...
@overload
def load_config(path: None) -> None: ...

def load_config(path: str | None) -> dict[str, str] | None:
    if path is None:
        return None
    with open(path) as f:
        import json
        return json.load(f)

# 5. Callable with ParamSpec for decorators
from typing import ParamSpec
P = ParamSpec('P')
R = TypeVar('R')

def retry(max_attempts: int) -> Callable[[Callable[P, R]], Callable[P, R]]:
    def decorator(func: Callable[P, R]) -> Callable[P, R]:
        import functools
        @functools.wraps(func)
        def wrapper(*args: P.args, **kwargs: P.kwargs) -> R:
            for attempt in range(max_attempts):
                try:
                    return func(*args, **kwargs)
                except Exception:
                    if attempt == max_attempts - 1:
                        raise
        return wrapper
    return decorator
```

* * *

Summary

Construct	Purpose	When to use
`Optional[X]` / `X | None`	Nullable value	Return value may be absent
`Union[A, B]` / `A | B`	Multiple valid types	Arguments accepting heterogeneous types
`Any`	Opt out of type checking	Integration with untyped code
`TypeVar`	Generic type parameter	Linking arg/return types
`Generic[T]`	Generic class	Reusable typed containers
`Protocol`	Structural interface	Duck-typed dependencies
`TypedDict`	Dict with shape	JSON / config at boundaries
`dataclass`	Typed mutable object	Domain entities
`NamedTuple`	Typed immutable tuple	Lightweight value objects
`Literal`	Specific value set	Enum-like string params
`Final`	Non-reassignable	Constants
`ClassVar`	Class-level variable	Distinguishing class vs instance
`@overload`	Multiple signatures	Conditional return types
`ParamSpec`	Capture param list	Decorators preserving signatures
`Self`	Return subclass type	Fluent/builder patterns

* * *

Chapter 3 — Functional Programming: Comprehensions, `functools`, `itertools`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.1 What "Functional" Means in Python

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.2 Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.2.1 List Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.2.2 Dict and Set Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.2.3 Generator Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.2.4 `any` and `all` with Generator Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.2.5 Comprehension Anti-Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3 `functools`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3.1 `functools.wraps`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3.2 `functools.lru_cache` and `functools.cache`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3.3 `functools.partial`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3.4 `functools.reduce`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3.5 `functools.total_ordering`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.3.6 `functools.singledispatch`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.4 `itertools`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.4.1 Infinite Iterators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.4.2 Finite Iterators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.4.3 `groupby` — Group Consecutive Elements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.4.4 Combinatoric Iterators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.4.5 `itertools.tee`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.5 Building an Iterator Algebra

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.5.1 Windowed Iteration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.5.2 `itertools.batched` (Python 3.12+)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.5.3 A Complete Pipeline Example

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.6 `operator` Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.7 Higher-Order Function Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.7.1 Currying and Partial Application

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.7.2 Function Composition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.7.3 Memoization vs Caching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.8 `map` and `filter`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.9 Performance Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

3.10 Common Interview Applications

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Pattern: Flatten a Nested List

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Pattern: Running Statistics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Pattern: Round-Robin Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Pattern: Lazy CSV Processing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 4 — Iterators, Generators & Coroutines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.1 The Iterator Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.1.1 The Legacy Iteration Protocol

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.1.2 Separating Iterable from Iterator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.2 Generator Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.2.1 Execution Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.2.2 `send()`, `throw()`, and `close()`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.2.3 Generator Return Values

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.3 `yield from`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.3.1 Simple Delegation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.3.2 Full `yield from` Semantics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.3.3 `yield from` for Return Value Chaining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.4 Practical Generator Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.4.1 Infinite Sequences with Early Termination

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.4.2 Coroutine-Style Data Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.4.3 Context Manager Generator (revisited)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.4.4 Generator-Based State Machine

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.5 Async Generators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.6 `contextlib` Utilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.6.1 `contextlib.suppress`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.6.2 `contextlib.ExitStack`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.6.3 `contextlib.asynccontextmanager`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.6.4 `contextlib.closing`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.6.5 `contextlib.redirect_stdout` / `redirect_stderr`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.7 Advanced Generator Internals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.7.1 Frame Objects and Generator State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.7.2 Generator Memory Footprint

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.7.3 `gi_yieldfrom`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.8 Typing Generators and Iterators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.9 Common Mistakes and Pitfalls

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Mistake 1: Reusing an Exhausted Generator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Mistake 2: Sending to an Unprimed Generator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Mistake 3: Ignoring `StopIteration` Propagation Inside Generators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Mistake 4: Mutable State Shared Across Generator Instances

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Mistake 5: `yield from` on a Non-Generator Iterable in an Async Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

4.10 Complete Example: Lazy Streaming Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 5 — Decorators & Metaclasses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.1 What a Decorator Is

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.2 Function Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.2.1 The Minimal Wrapper Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.2.2 Decorator Factories (Decorators with Arguments)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.2.3 Optional Arguments — Single Decorator Usable With or Without Parentheses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.2.4 Preserving Decorated Function Signatures for Introspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.3 Class Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.3.1 Adding Methods or Attributes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.3.2 Singleton via Class Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.3.3 Class Decorator vs Metaclass

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.4 Stacking Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.5 Descriptors as Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.6 The `type` Metaclass

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.6.1 Metaclass Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.7 Writing Metaclasses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.7.1 Enforcing Interface Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.7.2 Auto-Registration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.7.3 `__prepare__` for Ordered or Validated Namespaces

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.8 Abstract Base Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.8.1 `@abstractmethod` Combinations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.8.2 Virtual Subclasses

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.8.3 `__subclasshook__`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.9 `__init_subclass__` — Metaclass-Lite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.10 Decorator Patterns Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.10.1 Caching Property

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.10.2 Class-Level Validation Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.10.3 Rate-Limiting Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.10.4 Context-Injecting Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.10.5 Timing and Profiling Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.11 Decorator Anti-Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Anti-pattern 1: Forgetting `functools.wraps`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Anti-pattern 2: Mutable Default in Decorator Factory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Anti-pattern 3: Decorator that breaks subclassing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Anti-pattern 4: Metaclass Without Calling `super()`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.12 Choosing the Right Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

5.13 Complete Example: A Production-Grade Decorator Stack

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 6 — Memory Management & CPython Internals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.1 Why CPython Internals Matter at Staff Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.2 Reference Counting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.2.1 Where References Are Created

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.2.2 Strengths and Weaknesses of Reference Counting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.2.3 `weakref` — Non-Owning References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.3 The Cyclic Garbage Collector

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.3.1 What It Collects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.3.2 Generational Collection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.3.3 Finalizers and `__del__`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.3.4 Controlling GC for Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.4 CPython Memory Allocator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.4.1 Three Allocation Tiers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.4.2 Free Lists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.4.3 Measuring Object Size

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.5 String Interning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.6 Integer Interning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.7 The `dis` Module — Bytecode Inspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.7.1 Why Local Variables Are Faster Than Globals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.7.2 Constants and Compile-Time Folding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.7.3 The `code` Object

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.8 The Import System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.8.1 `sys.modules` — The Import Cache

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.8.2 Import Resolution Order

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.8.3 Circular Imports

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.8.4 Custom Import Hooks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.9 The Global Interpreter Lock (Preview)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.10 Profiling and Memory Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.10.1 `cProfile` — CPU Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.10.2 `line_profiler` — Line-Level CPU Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.10.3 `tracemalloc` — Memory Allocation Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.10.4 `timeit` — Micro-benchmarking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.10.5 `memory_profiler` — Line-Level Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.11 Object Lifecycle Patterns and Leak Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.11.1 Common Causes of Memory Leaks in Python

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.11.2 Diagnosing a Memory Leak

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.12 `__del__`, `weakref.finalize`, and Safe Cleanup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

6.13 Slots Revisited — Memory Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 7 — The GIL: A Precise Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.1 What the GIL Is

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.2 Why the GIL Exists

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.2.1 Reference Counting Is Not Thread-Safe

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.2.2 CPython's Internal Data Structures Are Not Thread-Safe

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.2.3 C Extension Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.3 The Execution Model: When the GIL Is Held and Released

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.3.1 The Check Interval

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.3.2 Voluntary GIL Release Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.3.3 What Does NOT Release the GIL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.4 Threading with the GIL: What Actually Happens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.4.1 I/O-Bound Workload — Threading Helps

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.4.2 CPU-Bound Workload — Threading Does Not Help (and May Hurt)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.4.3 The GIL Contention Problem in Detail

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.5 The GIL and C Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.5.1 NumPy and the GIL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.5.2 Writing GIL-Releasing C Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.5.3 `ctypes` and `cffi` — Automatic GIL Release

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.6 Alternatives to the GIL for CPU-Bound Parallelism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.6.1 `multiprocessing` — Process-Based Parallelism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.6.2 `concurrent.futures.ProcessPoolExecutor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.6.3 NumPy/SciPy — Bypass at the Library Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.6.4 Free-Threaded CPython (Python 3.13, Experimental)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.7 Atomicity Under the GIL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.8 The GIL and `asyncio`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.9 Diagnosing GIL-Related Performance Problems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.9.1 Identifying GIL Contention

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.9.2 Profiling a Multi-Threaded Program

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.10 The GIL in Practice: Decision Framework

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.10.1 Practical Thread Count Guidelines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

7.11 Common Misconceptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 8 — Threading: Correct Concurrent Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.1 The `threading` Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.2 `Thread` — Lifecycle and Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.2.1 Creating and Starting Threads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.2.2 Daemon Threads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.2.3 `join()` with Timeout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.2.4 Thread Identity and Introspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.3 `Lock` — Mutual Exclusion

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.3.1 Lock Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.3.2 Non-Blocking Lock Acquisition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.4 `RLock` — Reentrant Lock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.5 `Semaphore` and `BoundedSemaphore`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.5.1 Semaphore as Rate Limiter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.6 `Event` — Thread Signalling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.6.1 Stop Event Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.7 `Condition` — Monitor Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.7.1 The `while` Loop Invariant

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.7.2 `notify()` vs `notify_all()`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.7.3 `Condition` with External Lock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.8 `Barrier` — Phase Synchronisation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.9 `threading.local` — Thread-Local Storage

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.10 Deadlock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.10.1 Classic Deadlock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.10.2 Deadlock Prevention: Lock Ordering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.10.3 Deadlock Prevention: Timeout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.10.4 Deadlock Detection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.11 Livelock and Starvation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.12 Correct Thread-Safe Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.12.1 Thread-Safe Counter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.12.2 Thread-Safe Singleton

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.12.3 Thread-Safe Visited Set (Web Crawler Pattern)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.12.4 Lock Striping for High-Throughput Structures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.13 `threading.Timer`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.14 `threading.excepthook` — Unhandled Thread Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

8.15 Complete Example: Thread-Safe Web Crawler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 9 — `concurrent.futures` & Queue-Based Concurrency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.1 The Abstraction Hierarchy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.2 `concurrent.futures` — Core Concepts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.2.1 Executors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.2.2 `submit()` — Single Work Item

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.2.3 `map()` — Parallel Map

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.2.4 `as_completed()` — Results in Completion Order

9.2.5 `wait()` — Conditional Waiting

9.3 `ThreadPoolExecutor` in Depth

9.3.1 Pool Sizing

9.3.2 Thread Names and Initializers

9.3.3 Executor Shutdown

9.3.4 Internal Architecture

9.4 `ProcessPoolExecutor` in Depth

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.4.1 Pickling Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.4.2 `initializer` for Process Pools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.4.3 `mp_context` — Choosing Start Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.4.4 Shared Memory (Python 3.8+)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.5 `queue` Module — Production Queue Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.5.1 `queue.Queue` — Thread-Safe FIFO

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.5.2 `queue.LifoQueue` and `queue.PriorityQueue`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.5.3 `queue.SimpleQueue` — Lock-Free Fast Queue

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.6 Producer-Consumer Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.6.1 Single Producer, Multiple Consumers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.6.2 Multiple Producers, Multiple Consumers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.6.3 Pipeline Stages

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.7 Backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.8 `concurrent.futures` vs `queue` — Decision Guide

9.9 Error Handling in Futures

9.9.1 Per-Future Error Handling

9.9.2 Cancellation

9.9.3 `BrokenProcessPool` and `BrokenThreadPool`

9.10 Monitoring and Observability

9.10.1 Tracking Pool Utilisation

9.10.2 Queue Depth Monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

9.11 Complete Example: Parallel Crawler with `ProcessPoolExecutor`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 10 — Asyncio: Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.1 The Concurrency Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.2 Coroutines, Tasks, and the Event Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.2.1 Coroutines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.2.2 The Event Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.2.3 `asyncio.get_event_loop()` vs `asyncio.get_running_loop()`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.2.4 Tasks — Scheduled Coroutines

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.2.5 Task Object Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.3 `gather`, `wait`, and `as_completed`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.3.1 `asyncio.gather`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.3.2 `asyncio.wait`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.3.3 `asyncio.as_completed`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.4 Cancellation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.4.1 How Cancellation Works

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.4.2 `asyncio.shield`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.4.3 `asyncio.timeout` (Python 3.11+)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.5 Synchronisation Primitives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.5.1 `asyncio.Lock`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.5.2 `asyncio.Semaphore`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.5.3 `asyncio.Event`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.5.4 `asyncio.Condition`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.5.5 `asyncio.Queue`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.6 `run_in_executor` — Bridging Sync and Async

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.7 Event Loop Internals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.7.1 The Event Loop Cycle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.7.2 `loop.time()`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.7.3 Low-Level Transports and Protocols

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.7.4 `asyncio.StreamReader` / `StreamWriter`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.8 Structured Concurrency (Python 3.11+ `TaskGroup`)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.9 `asyncio` Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.9.1 Async Context Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.9.2 Async Iterator

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.9.3 Semaphore-Bounded Concurrent Fetcher

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.9.4 Periodic Background Task

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.9.5 Fan-Out / Fan-In with Queue

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.10 Mixing `asyncio` with Threads

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.10.1 Calling Async Code from a Thread

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.10.2 Calling Sync Code from the Event Loop

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.10.3 `asyncio.to_thread` (Python 3.9+)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.11 Error Handling and Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.11.1 Exception Handler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.11.2 Debug Mode

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.11.3 `asyncio.TaskGroup` and Exception Groups in Tests

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.11.4 Detecting Blocking Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

10.12 `asyncio` and `aiohttp` — HTTP Client Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 11 — Multiprocessing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.1 When Multiprocessing Is the Right Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.2 Start Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.2.1 `spawn` (Default on Windows and macOS)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.2.2 `fork` (Default on Linux)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.2.3 `forkserver`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.3 `Process` — Individual Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.3.1 `Process` Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.3.2 Subclassing `Process`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.4 `Pool` — Worker Process Pools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.4.1 Pool Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.4.2 `AsyncResult` Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.4.3 `chunksize` Tuning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.4.4 Pool Initializer

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.5 Inter-Process Communication

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.5.1 `mp.Queue`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.5.2 `mp.Pipe`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.5.3 `mp.Value` and `mp.Array` — Shared Memory Primitives

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.6 `multiprocessing.shared_memory` — Zero-Copy Shared Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.6.1 `SharedMemory` Lifecycle Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.6.2 `ShareableList`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.7 `mp.Manager` — Managed Shared Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.8 Synchronisation Across Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.9 `ProcessPoolExecutor` vs `mp.Pool`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.10 Pickling Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.10.1 What Is and Is Not Picklable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.10.2 Making Custom Objects Picklable

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.10.3 `cloudpickle` for Lambdas and Closures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.11 Common Failure Modes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.11.1 The `if __name__ == '__main__':` Guard

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.11.2 Deadlock from `mp.Queue` and `fork`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.11.3 Zombie Processes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.11.4 Broken Pipe on `mp.Queue.put()`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

11.12 Complete Example: Parallel Document Processor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 12 — Designing Pythonic APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.1 What Makes an API Pythonic

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.2 Function Signatures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.2.1 Positional, Keyword, and Positional-Only Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.2.2 Argument Defaults — Correctness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.2.3 `*args` and `**kwargs` — When to Use

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.2.4 Fluent Interface and Method Chaining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.3 Return Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.3.1 Return the Most Specific Type You Can Guarantee

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.3.2 Iterator vs List vs Sequence

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.3.3 `None` vs Exception vs `Optional` Return

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.3.4 Result Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.4 Lazy vs Eager APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.4.1 The Default Should Match the Common Case

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.4.2 Generator Functions as Public API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.5 Context Managers as API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.5.1 Making Resources Context Managers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.5.2 Context Manager Protocols and `ExitStack`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.6 Error Handling in APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.6.1 Exception Hierarchy Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.6.2 Exception Chaining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.6.3 When to Use Assertions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.7 Naming and Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.7.1 PEP 8 Naming Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.7.2 Naming Principles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.7.3 The _ and __ Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.8 Versioning and Backward Compatibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.8.1 Semantic Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.8.2 Deprecation Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.8.3 Adding Parameters Without Breaking Callers

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.9 Configuration and Dependency Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.9.1 Configuration Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.9.2 Dependency Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.9.3 Factory Functions and Class Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.10 Operator Overloading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.11 Introspection and `__repr__`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

12.12 Complete Example: A Pythonic HTTP Client API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 13 — Object-Oriented Design & Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.1 Design Patterns in a Python Context

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.2 SOLID Principles in Python

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.2.1 Single Responsibility Principle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.2.2 Open/Closed Principle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.2.3 Liskov Substitution Principle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.2.4 Interface Segregation Principle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.2.5 Dependency Inversion Principle

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.3 Creational Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.3.1 Factory Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.3.2 Abstract Factory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.3.3 Builder

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.3.4 Singleton

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.4 Structural Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.4.1 Adapter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.4.2 Decorator Pattern (Structural, not the @ Syntax)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.4.3 Facade

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.4.4 Proxy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.4.5 Composite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5 Behavioural Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.1 Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.2 Observer / Event System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.3 Command

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.4 Template Method

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.5 Iterator (Pattern)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.6 Chain of Responsibility

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.5.7 State

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.6 Composition vs Inheritance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.6.1 The Rule

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.6.2 Mixin Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.6.3 Composition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

13.7 Complete Example: A Plugin-Based Processing System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 14 — Designing for Change & Extension

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.1 The Core Problem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.2 Identifying and Isolating Change Axes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.3 Extension Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.3.1 Protocol-Based Extension Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.3.2 Hook Methods

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.3.3 Callable Extension Points (Higher-Order Functions)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.4 Configuration Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.4.1 Configuration Hierarchy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.4.2 Immutable Configuration Objects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.4.3 Secrets Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.5 Feature Flags

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.5.1 Simple In-Process Feature Flags

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.5.2 Gradual Rollout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.6 Dependency Management and Import Discipline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.6.1 Dependency Direction

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.6.2 Dependency Inversion at Module Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.6.3 Deferred Imports

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.6.4 `__all__` — Explicit Public Interface

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.7 Versioning API Contracts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.7.1 Versioning Strategy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.7.2 Multi-Version Support

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.7.3 Changelog and Deprecation Timeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.8 Module and Package Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.8.1 Package Layout for Extensible Systems

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.8.2 Top-Level `__init__.py` as Facade

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.9 Testing as a Design Signal

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.9.1 Testability Indicators

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.9.2 Seams

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.10 Backward Compatibility Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

14.11 Complete Example: Extensible Middleware System

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 15 — Packaging & Distribution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.1 Why Packaging Matters at Staff Level

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.2 The Modern Python Package Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.2.1 Key Standards

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.2.2 Build Backends

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.3 `pyproject.toml` — The Complete Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.3.1 Minimal Pure-Python Package

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.3.2 Tool Configuration in `pyproject.toml`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.4 Version Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.4.1 PEP 440 Version Scheme

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.4.2 Single Source of Truth for Version

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.4.3 Version Bumping

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.5 Dependency Specification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.5.1 Version Constraints

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.5.2 Extras (Optional Dependencies)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.5.3 Dependency Groups (PEP 735, Python 3.12+)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.5.4 Lockfiles

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.6 Project Layout

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.6.1 `src` Layout (Recommended)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.6.2 Flat Layout (Simpler Projects)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.7 Entry Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.7.1 Console Scripts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.7.2 Plugin Entry Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.7.3 `importlib.metadata` vs `pkg_resources`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.8 Distribution Formats

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.8.1 Source Distribution (sdist)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.8.2 Wheel (`.whl`)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.8.3 Publishing to PyPI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.9 Type Information Distribution (PEP 561)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.10 Virtual Environments and Isolation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.10.1 venv — Built-in Virtual Environments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.10.2 Editable Installs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.10.3 uv — Modern Package Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.11 CI/CD Packaging Pipeline

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.11.1 GitHub Actions Workflow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.11.2 Testing Matrix

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.12 CLI Design

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.12.1 `argparse` — Standard Library CLI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.12.2 Testing the CLI

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.13 Reproducible Builds and Supply Chain Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.13.1 Hash Verification

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.13.2 Trusted Publishing (PyPI OIDC)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.13.3 `SBOM` and Dependency Auditing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.14 Internal Packages and Private Registries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.14.1 Private PyPI with `devpi` or Artifactory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.14.2 Namespace Packages (PEP 420)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

15.15 Complete `pyproject.toml` Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 16 — Performance Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.1 The Performance Engineering Mindset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.2 Understanding Python's Performance Characteristics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.2.1 The Cost Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.2.2 Where Python Is Slow

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3 Profiling Tools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.1 `cProfile` — Deterministic CPU Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.2 `line_profiler` — Line-Level CPU Profiling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.3 `py-spy` — Sampling Profiler (No Code Changes)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.4 `tracemalloc` — Memory Allocation Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.5 `memray` — Line-Level Memory Profiler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.6 `timeit` — Micro-benchmarking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.3.7 `perf_counter` for Wall Time

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4 Common Bottlenecks and Their Fixes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.1 Attribute Lookup in Tight Loops

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.2 String Concatenation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.3 Repeated `isinstance` and `type` Checks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.4 Repeated Dictionary Lookups

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.5 List vs `deque` for Queue Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.6 Unnecessary Object Creation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.4.7 Avoiding Python Loops: NumPy Vectorisation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.5 Data Structure Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.5.1 Time Complexity Reference

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.5.2 Choosing the Right Structure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.6 Caching Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.6.1 `functools.lru_cache` — Function Memoization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.6.2 Manual Caching with TTL

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.6.3 Bloom Filters for Probabilistic Membership

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.7 Algorithmic Complexity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.7.1 Common Complexity Improvements

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.7.2 Amortised Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.8 Python-Specific Optimisation Techniques

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.8.1 Local Variable Hoisting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.8.2 `__slots__` for Memory-Dense Classes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.8.3 `array` Module for Homogeneous Sequences

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.8.4 `bytearray` vs `bytes`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.8.5 `map` with Built-in Functions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.9 Concurrency as a Performance Tool

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.9.1 Throughput vs Latency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.10 Cython, Numba, and C Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.10.1 Cython

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.10.2 Numba JIT

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.10.3 ctypes and cffi — Calling C Libraries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.11 Benchmarking Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.11.1 Avoiding Benchmarking Errors

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.11.2 Regression Testing for Performance

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

16.12 Complete Example: Profiling and Optimising a URL Processor

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 17 — Debugging Under Pressure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.1 The Debugging Mindset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.2 Reading Tracebacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.2.1 Traceback Anatomy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.2.2 Chained Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.2.3 `traceback` Module

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.3 `pdb` — The Python Debugger

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.3.1 Entering `pdb`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.3.2 Core `pdb` Commands

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.3.3 Post-Mortem Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.3.4 Debugging Async Code

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.4 `logging` — Structured Observability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.4.1 Logger Setup

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.4.2 Application-Level Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.4.3 Logging Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.4.4 Structured Logging with `structlog`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.5 `sys._current_frames()` and Stack Inspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.6 `faulthandler` — Segfault and Signal Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.7 Debugging Concurrency Bugs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.7.1 Identifying a Race Condition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.7.2 Diagnosing a Deadlock

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.7.3 Reproducing Non-Deterministic Bugs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.7.4 `threading.settrace` for Execution Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.8 `inspect` Module — Runtime Introspection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.8.1 Caller Inspection for Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.9 `dis` Module — Bytecode Inspection for Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.10 Remote Debugging and Production Diagnostics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.10.1 `debugpy` — VS Code Remote Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.10.2 `icecream` — Improved print Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.10.3 `objgraph` — Object Reference Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.11 Binary Search Debugging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.12 Debugging Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

17.13 Complete Example: Debugging a Production Race Condition

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 18 — Failure Mode Thinking & Resilience Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.1 The Failure Mode Mindset

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.2 Timeout Everywhere

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.2.1 Network Timeouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.2.2 Lock and Queue Timeouts

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.2.3 Timeout Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.3 Retry with Exponential Backoff and Jitter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.3.1 Exponential Backoff

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.3.2 Jitter Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.3.3 Idempotency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.4 Circuit Breaker

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.4.1 Circuit Breaker Integration Points

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.5 Rate Limiting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.5.1 Token Bucket

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.5.2 Sliding Window Counter

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.6 Bulkhead Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.7 Graceful Degradation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.7.1 Cache-Aside with Stale Data

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.8 Load Shedding

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.9 Backpressure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.10 Graceful Shutdown

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.10.1 Graceful Shutdown in `asyncio`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.11 Health Checks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.12 Failure Mode Analysis: The Web Crawler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

18.13 Complete Example: Resilient HTTP Client

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Part V — Production-Grade Engineering

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 19 — Security Awareness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.1 Security as a Design Property

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.2 Injection Vulnerabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.2.1 SQL Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.2.2 Shell Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.2.3 `eval` and `exec`

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.3 `pickle` — Arbitrary Code Execution

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.4 Path Traversal

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.5 Server-Side Request Forgery (SSRF)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.6 Secrets Management

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.6.1 Never Hardcode Secrets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.6.2 Secrets in Memory

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.6.3 Timing Attacks on Secret Comparison

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.7 Input Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.7.1 Validate at Trust Boundaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.7.2 File Upload Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.8 Cryptography

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.8.1 Password Hashing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.8.2 Generating Secrets and Tokens

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.8.3 Symmetric Encryption

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.8.4 HMAC for Data Integrity

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.9 Dependency Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.9.1 Scanning for Known Vulnerabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.9.2 Supply Chain Attacks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.9.3 `bandit` — Static Security Analysis

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.10 Web Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.10.1 Cross-Site Request Forgery (CSRF)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.10.2 Content Security Policy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.10.3 Safe URL Redirects

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.11 Logging Security

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.11.1 Never Log Secrets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.11.2 Log Injection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.12 Security Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

19.13 Complete Example: Secure Web Crawler Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 20 — Observability & Monitoring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.1 The Three Pillars of Observability

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.2 Structured Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.2.1 JSON Structured Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.2.2 Correlation IDs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.2.3 Log Levels and Volume Control

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.3 Metrics

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.3.1 Metric Types

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.3.2 `prometheus_client` — Metrics in Python

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.3.3 Label Cardinality

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.3.4 Custom Metrics Registry for Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.4 Distributed Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.4.1 OpenTelemetry — The Standard

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.4.2 Span Attributes and Semantic Conventions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.4.3 Context Propagation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.4.4 Async Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.5 Health Checks and Readiness Probes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.5.1 Instrumenting Health Checks

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.5.2 Kubernetes Probe Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.6 Alerting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.6.1 Prometheus Alerting Rules

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.6.2 The Four Golden Signals

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.7 SLIs, SLOs, and Error Budgets

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.7.1 Service Level Indicators (SLIs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.7.2 Service Level Objectives (SLOs)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.7.3 Error Budget

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.8 `structlog` — Production Structured Logging

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.9 Runbooks and On-Call Readiness

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

20.10 Complete Example: Fully Observable Crawler

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Chapter 21 — Reliability Concepts & Distributed Patterns

21.1 From Single-Process to Distributed Systems

21.2 The CAP Theorem and Its Practical Meaning

21.3 Consistency Models

21.3.1 Strong Consistency (Linearisability)

21.3.2 Sequential Consistency

21.3.3 Eventual Consistency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.3.4 Read-Your-Writes Consistency

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.4 Distributed Transactions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.4.1 Two-Phase Commit (2PC)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.4.2 Saga Pattern

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.5 Consistent Hashing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.6 Message Queues and Event-Driven Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.6.1 At-Most-Once, At-Least-Once, Exactly-Once

21.6.2 Outbox Pattern

21.7 Leader Election

21.7.1 Redis-Based Leader Election

21.8 Service Discovery and Configuration

21.8.1 DNS-Based Service Discovery

21.8.2 Environment-Based Configuration

21.9 Horizontal Scaling Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.9.1 Stateless Services

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.9.2 Work Partitioning

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.10 Data Replication and Read Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.10.1 Read Replicas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.10.2 CQRS (Command Query Responsibility Segregation)

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.11 Distributed Rate Limiting

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.12 The Twelve-Factor App

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

21.13 Complete Example: Distributed Crawler Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.

Summary

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/python-mastery.