Python Interview Script - Complete Guide

Every Python concept explained with: **Interviewer Question → Your Answer → Code Example → VibeVault Project Reference**

---

1. PYTHON BASICS

What are Python's key features?

Interviewer: "What are Python's key features?" **You:** "Python is a high-level, interpreted language known for readability, dynamic typing, and vast library support."

```
# Key Features I used in VibeVault:
# 1. Dynamic Typing - No need to declare types
user_count = 10              # Integer
user_name = "John"           # String

# 2. Interpreted - No compilation step
# 3. Extensive Libraries - I used numpy, transformers, celery, django

# 4. Object-Oriented - I defined custom classes
class AnalyzerPipeline:
    def __init__(self):
        self.embedder = None
```

In VibeVault: "I leveraged Python's extensive ecosystem - Django for web, Celery for async tasks, NumPy for AI computations."

What is the difference between Python 2 and Python 3?

Interviewer: "Difference between Python 2 and 3?" **You:** "Python 3 has Unicode strings by default, print as a function, integer division returns float, and many deprecated features removed."

```
# Python 2 (OLD - don't use)
print "Hello"              # Statement
5 / 2  # Returns 2         # Integer division

# Python 3 (CURRENT - I use this)
print("Hello")             # Function
5 / 2  # Returns 2.5       # True division
5 // 2  # Returns 2        # Floor division
```

In VibeVault: "I use Python 3.10+. All my shebang lines say `#!/usr/bin/env python` which maps to Python 3."

Explain Python memory management.

Interviewer: "How does Python manage memory?" **You:** "Python uses automatic memory management with reference counting and a garbage collector for cyclic references."

```
# Reference counting - each object tracks how many references point to it
a = [1, 2, 3]  # Reference count: 1
b = a          # Reference count: 2
del a          # Reference count: 1
del b          # Reference count: 0 -> Object deleted

# VibeVault example: Large embeddings
embedding = np.array([...])  # Memory allocated
del embedding                # Memory freed when reference count = 0
```

In VibeVault: "I'm careful with large NLP model outputs. I delete embeddings after storing them to free memory."

What are Python namespaces?

Interviewer: "What are namespaces?" **You:** "Namespaces are containers that map names to objects. Python has local, enclosing, global, and built-in namespaces (LEGB rule)."

```
# Global namespace
x = "global"

def outer():
    # Enclosing namespace
    x = "enclosing"

    def inner():
        # Local namespace
        x = "local"
        print(x)  # Prints "local"

    inner()

# Built-in namespace: len(), print(), str()
```

In VibeVault: "I follow the LEGB rule. My Django settings are global, view functions have local scope."

Mutable vs Immutable types?

Interviewer: "What's the difference between mutable and immutable?" **You:** "Mutable objects can be changed after creation (lists, dicts). Immutable objects cannot (strings, tuples, integers)."

```
# Mutable - Can modify in place
my_list = [1, 2, 3]
my_list.append(4)  # Same object, modified

# Immutable - Cannot modify, creates new object
my_string = "hello"
```

```
my_string = my_string + " world"  # New object created

# VibeVault example
memory.metrics_json['sentiment'] = 'positive'  # Dict is mutable
```

In VibeVault: "I use dicts (mutable) for metrics_json so I can update AI results incrementally."

What are Python's built-in data types?

Interviewer: "List Python's built-in data types." **You:** "Numeric (int, float, complex), Sequence (str, list, tuple), Set (set, frozenset), Mapping (dict), Boolean, None."

```
# All types I use in VibeVault:
count = 42                    # int
score = 0.95                  # float
title = "My Memory"           # str
keywords = ['happy', 'joy']   # list
config = ('host', 5432)       # tuple (immutable)
unique_tags = {'ai', 'nlp'}   # set
metrics = {'score': 0.9}      # dict
is_processed = True           # bool
result = None                 # NoneType
```

What are *args and **kwargs?

Interviewer: "Explain *args and **kwargs." **You:** "*args collects positional arguments as a tuple. **kwargs collects keyword arguments as a dictionary."

```
def analyze(*args, **kwargs):
    # args = tuple of positional arguments
    # kwargs = dict of keyword arguments

    for text in args:
        print(f"Analyzing: {text}")

    if kwargs.get('include_sentiment'):
        print("Including sentiment analysis")

# Usage
analyze("text1", "text2", include_sentiment=True, model="bert")
```

In VibeVault: "I use **kwargs in signals: `def create_welcome_memory(sender, instance, created, **kwargs)` "

What are Python decorators?

Interviewer: "What are decorators?" **You:** "Decorators are functions that modify the behavior of other functions. They wrap a function and add functionality."

```python
# Decorator definition
def log_call(func):
    def wrapper(*args, **kwargs):
        print(f"Calling {func.__name__}")
        result = func(*args, **kwargs)
        print(f"Finished {func.__name__}")
        return result
    return wrapper

# Usage with @ syntax
@log_call
def analyze_text(text):
    return "analyzed"

# VibeVault decorators I use:
@receiver(post_save, sender=User)     # Django signal
@shared_task                           # Celery task
@api_view(['GET'])                     # DRF view
@action(detail=False)                  # DRF custom action
```

In VibeVault: "I use `@shared_task` for Celery, `@receiver` for signals, `@api_view` for DRF endpoints."

What are lambda functions?

Interviewer: "What are lambda functions?" **You:** "Lambda functions are anonymous, single-expression functions. Used for short operations, especially with map/filter/sorted."

```python
# Regular function
def square(x):
    return x * 2

# Lambda equivalent
square = lambda x: x * 2

# VibeVault usage - sorting search results
results.sort(key=lambda x: x['score'], reverse=True)

# Filtering memories
pending = filter(lambda m: m.analysis_status == 'pending', memories)
```

In VibeVault: "I use lambda in `results.sort(key=lambda x: x['score'], reverse=True)` for search ranking."

Deep copy vs Shallow copy?

Interviewer: "Difference between deep and shallow copy?" **You:** "Shallow copy creates a new object but references the same nested objects. Deep copy creates completely independent copies."

```python
import copy
```

```python
original = {'user': 'john', 'data': [1, 2, 3]}

# Shallow copy - nested list is still shared
shallow = copy.copy(original)
shallow['data'].append(4)
print(original['data'])  # [1, 2, 3, 4] - MODIFIED!

# Deep copy - completely independent
deep = copy.deepcopy(original)
deep['data'].append(5)
print(original['data'])  # [1, 2, 3, 4] - Unchanged
```

In VibeVault: "When cloning metrics_json, I use deepcopy to avoid accidentally modifying the original."

Garbage collection mechanism?

Interviewer: "Explain Python's garbage collection." **You:** "Python uses reference counting plus a cyclic garbage collector for objects that reference each other."

```python
import gc

# Reference counting handles most cases
def process():
    data = [1, 2, 3]  # ref count = 1
    return sum(data)
# After function: ref count = 0, memory freed

# Cyclic references need GC
a = []
b = []
a.append(b)
b.append(a)  # Circular reference!
del a, b     # Reference count never reaches 0
gc.collect() # GC detects and cleans cycles
```

In VibeVault: "I trust Python's GC for normal operations. For large models, I explicitly `del` to hint early cleanup."

Module vs Package?

Interviewer: "Difference between module and package?" **You:** "A module is a single .py file. A package is a directory with `__init__.py` containing multiple modules."

```
# VibeVault structure:
backend/
├── config/              # Package (has __init__.py)
│   ├── __init__.py
│   ├── settings.py      # Module
│   ├── urls.py          # Module
│   └── middleware.py    # Module
├── users/               # Package
```

```
│   ├── __init__.py
│   ├── models.py        # Module
│   └── views.py         # Module
```

In VibeVault: "Each Django app is a package. `users.models` means the `models.py` module in the `users` package."

How do you import a module?

Interviewer: "How do you import modules?" **You:** "Using `import`, `from...import`, or `import...as` for aliasing."

```python
# Full import
import os
os.getenv('SECRET_KEY')

# Specific import
from django.db import models
class Memory(models.Model): pass

# Aliased import
import numpy as np
embedding = np.array([...])

# Relative import (within package)
from .models import Memory
from ..config import settings
```

In VibeVault: "I use relative imports within apps: `from .serializers import MemorySerializer`"

What are Python docstrings?

Interviewer: "What are docstrings?" **You:** "Docstrings are string literals that document modules, classes, and functions. Accessible via `__doc__`."

```python
def analyze_sentiment(text):
    """
    Analyze sentiment of the given text.

    Args:
        text (str): The text to analyze

    Returns:
        dict: Contains 'label' and 'score' keys

    Example:
        >>> analyze_sentiment("I love this!")
        {'label': 'positive', 'score': 0.95}
```

```
    """
    # Implementation...
```

In VibeVault: "Every function has a docstring explaining purpose, args, and return value."

What is PEP 8?

Interviewer: "What is PEP 8?" **You:** "PEP 8 is Python's official style guide. It covers indentation (4 spaces), naming conventions, line length (79-120 chars), etc."

```
# PEP 8 compliant code:
class MemoryAnalyzer:  # CamelCase for classes

    def analyze_text(self, text):  # snake_case for functions
        MAX_LENGTH = 512  # UPPERCASE for constants

        if not text:
            return None

        # Line length under 120 characters
        result = self.embedder.encode(text)
        return result
```

In VibeVault: "I follow PEP 8 strictly - snake_case for functions, CamelCase for classes."

---

2. DATA STRUCTURES

What is a list in Python?

Interviewer: "What is a list?" **You:** "A list is an ordered, mutable collection that can hold items of different types."

```
# VibeVault example
keywords = ['happy', 'vacation', 'family']

# List operations
keywords.append('beach')          # Add item
keywords.insert(0, 'summer')      # Insert at position
keywords.remove('vacation')       # Remove by value
last = keywords.pop()             # Remove and return last
keywords.reverse()                # Reverse in place
keywords.sort()                   # Sort in place
```

In VibeVault: "I store keywords as a list in metrics_json: `metrics['keywords'] = [...]`"

Difference between list, tuple, and set?

Interviewer: "Compare list, tuple, and set." **You:** "List is ordered+mutable. Tuple is ordered+immutable. Set is unordered+unique items."

```
# List - ordered, mutable, allows duplicates
keywords = ['happy', 'happy', 'joy']

# Tuple - ordered, immutable, allows duplicates
coordinates = (40.7128, -74.0060)  # Can't change after creation

# Set - unordered, mutable, NO duplicates
unique_emotions = {'joy', 'sadness', 'joy'}  # Results in {'joy', 'sadness'}
```

In VibeVault: "I use sets for unique tags: `unique_keywords = set(all_keywords)` "

How do you remove duplicates from a list?

Interviewer: "Remove duplicates from a list?" **You:** "Convert to set and back to list, or use dict.fromkeys() to preserve order."

```
# Method 1: Using set (doesn't preserve order)
keywords = ['happy', 'joy', 'happy', 'peace', 'joy']
unique = list(set(keywords))  # ['peace', 'happy', 'joy']

# Method 2: Preserve order (Python 3.7+)
unique = list(dict.fromkeys(keywords))  # ['happy', 'joy', 'peace']

# Method 3: List comprehension
seen = set()
unique = [x for x in keywords if not (x in seen or seen.add(x))]
```

In VibeVault: "I use `list(set(keywords))` when order doesn't matter for keyword deduplication."

How do you merge two dictionaries?

Interviewer: "How to merge dictionaries?" **You:** "Using `|` operator (Python 3.9+), `{**dict1, **dict2}` , or `.update()` ."

```
# Method 1: | operator (Python 3.9+)
metrics = {'sentiment': 'positive'}
emotion = {'emotion': 'joy'}
combined = metrics | emotion

# Method 2: Unpacking (Python 3.5+)
combined = {**metrics, **emotion}

# Method 3: update() (modifies in place)
metrics.update(emotion)
```

```
# VibeVault example
memory.metrics_json = {**base_metrics, **ai_results}
```

In VibeVault: "I merge AI results: `metrics = {**sentiment_result, **emotion_result}`"

What are list comprehensions?

Interviewer: "Explain list comprehensions." **You:** "A concise way to create lists using a single line with optional conditions."

```
# Traditional loop
squares = []
for x in range(10):
    squares.append(x ** 2)

# List comprehension
squares = [x ** 2 for x in range(10)]

# With condition
even_squares = [x ** 2 for x in range(10) if x % 2 == 0]

# VibeVault example
keywords = [kw['keyword'] for kw in keywords_data if kw['score'] > 0.5]
```

In VibeVault: "I use comprehensions for data extraction: `memory_ids = [m.id for m in memories]`"

What is a generator?

Interviewer: "What is a generator?" **You:** "A generator yields items one at a time instead of returning all at once. It's memory-efficient for large datasets."

```
# Generator function
def get_memories_batch(user, batch_size=100):
    memories = Memory.objects.filter(user=user)
    for i in range(0, memories.count(), batch_size):
        yield memories[i:i + batch_size]

# Usage - processes one batch at a time
for batch in get_memories_batch(user):
    process(batch)  # Memory-efficient!

# Generator expression
squares = (x ** 2 for x in range(1000000))  # No memory allocated yet
```

In VibeVault: "For large memory exports, I'd use generators to avoid loading everything into RAM."

---

3. FUNCTIONS AND OOPs

Function vs Method?

Interviewer: "Difference between function and method?" **You:** "A function is standalone. A method is a function bound to an object/class."

```
# Function - standalone
def analyze_text(text):
    return {'sentiment': 'positive'}

# Method - belongs to a class
class AnalyzerPipeline:
    def analyze_text(self, text):  # self refers to instance
        return {'sentiment': 'positive'}

# Usage
result = analyze_text("hello")           # Function call
analyzer = AnalyzerPipeline()
result = analyzer.analyze_text("hello")  # Method call
```

In VibeVault: "I define methods in classes like `MemoryViewSet.get_queryset()` "

What are default arguments?

Interviewer: "What are default arguments?" **You:** "Parameters with default values that are used if no argument is provided."

```
def search(query, user, top_k=20, use_embedding=True):
    # top_k defaults to 20, use_embedding defaults to True
    pass

# Usage
search("vacation", user)                    # Uses defaults
search("vacation", user, top_k=50)          # Override top_k
search("vacation", user, use_embedding=False)
```

⚠️ **Warning - Mutable default argument trap:**

```
# WRONG - mutable default is shared!
def append_to(item, target=[]):
    target.append(item)
    return target

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

Instance, Class, and Static Methods?

Interviewer: "Explain instance, class, and static methods." **You:** "Instance methods take `self`, class methods take `cls`, static methods take neither."

```python
class AnalyzerPipeline:
    model_name = "bert"  # Class variable

    def __init__(self):
        self.embedder = None  # Instance variable

    # Instance method - accesses instance data
    def analyze(self, text):
        return self.embedder.encode(text)

    # Class method - accesses class data
    @classmethod
    def get_model_name(cls):
        return cls.model_name

    # Static method - utility function, no access to class/instance
    @staticmethod
    def preprocess(text):
        return text.lower().strip()

# Usage
analyzer = AnalyzerPipeline()
analyzer.analyze("hello")              # Instance method
AnalyzerPipeline.get_model_name()      # Class method
AnalyzerPipeline.preprocess("  Hi  ")  # Static method
```

Explain inheritance and its types.

Interviewer: "Explain inheritance." **You:** "Inheritance allows a class to inherit attributes and methods from a parent class."

```python
# Single inheritance
class User(AbstractUser):  # Inherits from AbstractUser
    bio = models.TextField()

# Multiple inheritance
class MemoryViewSet(mixins.CreateModelMixin,
                    mixins.ListModelMixin,
                    GenericViewSet):  # Inherits from multiple
    pass

# Multilevel inheritance
class Animal: pass
```

```
class Mammal(Animal): pass
class Dog(Mammal): pass
```

In VibeVault: "I use inheritance: `class User(AbstractUser)`, `class MemoryViewSet(ModelViewSet)`"

What is method overriding?

Interviewer: "What is method overriding?" **You:** "When a child class provides its own implementation of a parent's method."

```
class ModelViewSet:
    def get_queryset(self):
        return self.queryset

class MemoryViewSet(ModelViewSet):
    # Override parent's method
    def get_queryset(self):
        # Custom logic: filter to current user only
        return Memory.objects.filter(user=self.request.user)

# super() calls parent's method
class MemoryViewSet(ModelViewSet):
    def get_queryset(self):
        base_qs = super().get_queryset()  # Call parent
        return base_qs.filter(user=self.request.user)
```

In VibeVault: "I override `get_queryset()` to ensure users only see their own memories."

What are abstract classes?

Interviewer: "What are abstract classes?" **You:** "Abstract classes can't be instantiated directly. They define a blueprint with abstract methods that children must implement."

```
from abc import ABC, abstractmethod

class BaseAnalyzer(ABC):
    @abstractmethod
    def analyze(self, text):
        pass  # Must be implemented by children

    def preprocess(self, text):
        return text.lower()  # Concrete method

class SentimentAnalyzer(BaseAnalyzer):
    def analyze(self, text):  # Required implementation
        return {'sentiment': 'positive'}

# BaseAnalyzer() -> Error! Can't instantiate abstract class
# SentimentAnalyzer() -> OK
```

In VibeVault: "DRF's BasePermission is abstract - I must implement `has_permission()`"

What is polymorphism?

Interviewer: "What is polymorphism?" **You:** "Polymorphism means the same method name behaves differently for different classes."

```
# Different classes, same method name
class TextMemory:
    def process(self):
        return "Processing text..."

class ImageMemory:
    def process(self):
        return "Processing image..."

# Polymorphic behavior
memories = [TextMemory(), ImageMemory()]
for m in memories:
    print(m.process())  # Calls the right method for each type
```

In VibeVault: "My serializers are polymorphic - `get_serializer_class()` returns different serializers based on action."

---

4. FILE HANDLING AND EXCEPTIONS

How do you open and read files?

Interviewer: "How do you read files in Python?" **You:** "Using `open()` with context manager `with` for automatic cleanup."

```
# Reading a file
with open('data.txt', 'r') as file:
    content = file.read()        # Entire file as string
    # OR
    lines = file.readlines()     # List of lines
    # OR
    for line in file:            # Line by line (memory efficient)
        print(line)

# Writing to a file
with open('output.txt', 'w') as file:
    file.write("Hello World")
```

In VibeVault: "Django handles file uploads, but I use this pattern for reading transcription files."

File opening modes?

Interviewer: "What are the file opening modes?" **You:** "r (read), w (write), a (append), x (create), b (binary), + (read/write)."

```
'r'   # Read (default)
'w'   # Write (overwrites!)
'a'   # Append
'x'   # Create (fails if exists)
'rb'  # Read binary (images, audio)
'wb'  # Write binary
'r+'  # Read and write

# VibeVault - reading uploaded audio
with open(audio_path, 'rb') as f:
    audio_data = f.read()
```

What is the with statement?

Interviewer: "What does `with` do?" **You:** "It's a context manager that ensures resources are properly cleaned up, even if errors occur."

```
# Without 'with' - must manually close
file = open('data.txt')
try:
    content = file.read()
finally:
    file.close()  # Must remember to close!

# With 'with' - automatic cleanup
with open('data.txt') as file:
    content = file.read()
# File automatically closed here

# Works with any context manager
with transaction.atomic():  # Database transaction
    Memory.objects.create(...)
```

In VibeVault: "I use `with transaction.atomic()` for database operations that must succeed or fail together."

How do you handle exceptions?

Interviewer: "How do you handle exceptions?" **You:** "Using try/except blocks to catch and handle errors gracefully."

```
try:
    memory = Memory.objects.get(id=memory_id)
    result = analyzer.analyze(memory.text)
except Memory.DoesNotExist:
```

```
    logger.error(f"Memory {memory_id} not found")
    return {'error': 'Not found'}
except Exception as e:
    logger.error(f"Analysis failed: {e}")
    raise  # Re-raise the exception
else:
    # Runs if NO exception occurred
    memory.metrics_json = result
    memory.save()
finally:
    # ALWAYS runs, even if exception
    logger.info("Analysis attempt completed")
```

In VibeVault: "My Celery tasks use try/except to handle failures gracefully and retry."

How do you raise custom exceptions?

Interviewer: "How do you create custom exceptions?" **You:** "By creating a class that inherits from Exception."

```
# Define custom exception
class AnalysisError(Exception):
    def __init__(self, message, memory_id=None):
        self.message = message
        self.memory_id = memory_id
        super().__init__(self.message)

# Raise it
def analyze(memory):
    if not memory.text:
        raise AnalysisError("Empty text", memory_id=memory.id)

# Catch it
try:
    analyze(memory)
except AnalysisError as e:
    logger.error(f"Analysis failed for {e.memory_id}: {e.message}")
```

---

5. MODULES AND LIBRARIES

How do you install external libraries?

Interviewer: "How do you install libraries?" **You:** "Using pip, Python's package installer."

```
# Install a package
pip install django

# Install specific version
pip install django==4.2.0
```

```
# Install from requirements.txt
pip install -r requirements.txt

# VibeVault requirements.txt
Django>=4.2
djangorestframework>=3.14
celery>=5.3
numpy>=1.24
sentence-transformers>=2.2
```

What are virtual environments?

Interviewer: "What are virtual environments?" **You:** "Isolated Python environments that keep project dependencies separate."

```
# Create virtual environment
python -m venv .venv

# Activate (Windows)
.venv\Scripts\activate

# Activate (Mac/Linux)
source .venv/bin/activate

# Deactivate
deactivate

# Why?
# Project A needs Django 3.2
# Project B needs Django 4.2
# Virtual envs keep them separate
```

In VibeVault: "I have a `.venv` folder for isolated dependencies. It's in `.gitignore`."

---

6. ITERATORS AND GENERATORS

What are iterators?

Interviewer: "What are iterators?" **You:** "Objects that implement `__iter__()` and `__next__()` methods to traverse a sequence."

```
# Any iterable can become an iterator
my_list = [1, 2, 3]
iterator = iter(my_list)

print(next(iterator))  # 1
```

```
print(next(iterator))  # 2
print(next(iterator))  # 3
print(next(iterator))  # StopIteration error

# Custom iterator
class MemoryIterator:
    def __init__(self, memories):
        self.memories = memories
        self.index = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self.index >= len(self.memories):
            raise StopIteration
        memory = self.memories[self.index]
        self.index += 1
        return memory
```

What is the yield keyword?

Interviewer: "What does yield do?" **You:** "Yield pauses a function and returns a value. The function resumes from where it left off on next call."

```
def count_up_to(max):
    count = 1
    while count <= max:
        yield count  # Pause and return value
        count += 1   # Resume here on next call

# Usage
counter = count_up_to(3)
print(next(counter))  # 1
print(next(counter))  # 2
print(next(counter))  # 3

# VibeVault - batch processing
def batch_memories(user, size=100):
    memories = Memory.objects.filter(user=user)
    for i in range(0, len(memories), size):
        yield memories[i:i+size]
```

---

7. ADVANCED PYTHON CONCEPTS

What are context managers?

Interviewer: "What are context managers?" **You:** "Objects that implement `__enter__` and `__exit__` for setup/teardown, used with `with` statement."

```
# Custom context manager using class
class Timer:
    def __enter__(self):
        self.start = time.time()
        return self

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.duration = time.time() - self.start
        print(f"Took {self.duration:.2f}s")

with Timer():
    # Some operation
    results = analyze_memories()

# Using contextlib decorator
from contextlib import contextmanager

@contextmanager
def timer():
    start = time.time()
    yield
    print(f"Took {time.time() - start:.2f}s")
```

Explain Python's GIL.

Interviewer: "What is the GIL?" **You:** "The Global Interpreter Lock prevents multiple threads from executing Python bytecode simultaneously. It affects CPU-bound tasks."

```
# GIL impact:
# - I/O bound tasks (network, file): Threading works fine
# - CPU bound tasks (computation): Use multiprocessing instead

# For I/O bound (GIL released during I/O)
import threading
def fetch_data(url):
    response = requests.get(url)  # GIL released here
    return response

# For CPU bound (use multiprocessing to bypass GIL)
from multiprocessing import Pool
def process_chunk(data):
    return heavy_computation(data)

with Pool(4) as p:
    results = p.map(process_chunk, data_chunks)
```

In VibeVault: "I use Celery (multiprocessing) for CPU-heavy AI tasks to bypass the GIL."

Threading vs Multiprocessing?

Interviewer: "Difference between threading and multiprocessing?" **You:** "Threading shares memory, good for I/O. Multiprocessing uses separate memory, good for CPU tasks."

```
# Threading - shared memory, GIL limits CPU parallelism
import threading
threads = [threading.Thread(target=download, args=(url,)) for url in urls]
for t in threads: t.start()
for t in threads: t.join()

# Multiprocessing - separate processes, true parallelism
from multiprocessing import Pool
with Pool(4) as p:
    results = p.map(analyze, texts)
```

In VibeVault: "Celery workers are separate processes. Each can do full CPU work independently."

Asynchronous programming?

Interviewer: "Explain async programming in Python." **You:** "Using async/await to handle I/O operations concurrently without blocking."

```
import asyncio

async def fetch_data(url):
    # Simulated async I/O
    await asyncio.sleep(1)
    return f"Data from {url}"

async def main():
    # Run multiple tasks concurrently
    results = await asyncio.gather(
        fetch_data("url1"),
        fetch_data("url2"),
        fetch_data("url3"),
    )
    print(results)

asyncio.run(main())
```

In VibeVault: "I use Django Channels with AsyncWebsocketConsumer for real-time notifications."

---

8. DATA HANDLING AND APIs

How do you handle JSON data?

Interviewer: "How do you work with JSON in Python?" **You:** "Using the `json` module for parsing and dumping."

```
import json

# Parse JSON string to Python dict
json_string = '{"emotion": "joy", "score": 0.95}'
data = json.loads(json_string)
print(data['emotion'])  # 'joy'

# Convert Python dict to JSON string
result = {'sentiment': 'positive'}
json_str = json.dumps(result, indent=2)

# Read/write JSON files
with open('config.json') as f:
    config = json.load(f)

with open('output.json', 'w') as f:
    json.dump(data, f, indent=2)
```

In VibeVault: "DRF handles JSON automatically. I store AI results in JSONField."

How do you make API calls?

Interviewer: "How do you make API calls in Python?" **You:** "Using the `requests` library for HTTP requests."

```
import requests

# GET request
response = requests.get('https://api.example.com/data')
data = response.json()

# POST request with JSON body
response = requests.post(
    'https://api.example.com/analyze',
    json={'text': 'Hello world'},
    headers={'Authorization': 'Bearer token123'}
)

# Error handling
if response.status_code == 200:
    result = response.json()
else:
    print(f"Error: {response.status_code}")
```

What is SQLAlchemy and ORM?

Interviewer: "What is ORM?" **You:** "Object-Relational Mapping translates Python classes to database tables. SQLAlchemy is the most popular ORM; Django has its own."

```python
# Django ORM (what I use)
class Memory(models.Model):
    title = models.CharField(max_length=255)
    text = models.TextField()

# ORM translates to SQL:
Memory.objects.filter(user=user)
# SELECT * FROM memories WHERE user_id = 1

Memory.objects.create(title="Test", text="Content")
# INSERT INTO memories (title, text) VALUES ('Test', 'Content')
```

In VibeVault: "I use Django's ORM exclusively. It abstracts SQL and provides security against injection."

---

9. PYTHON WITH WEB DEVELOPMENT

Flask vs Django?

Interviewer: "Difference between Flask and Django?" **You:** "Flask is a micro-framework (minimal, flexible). Django is a full-stack framework (batteries-included)."

```python
# Flask - minimal, you add what you need
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello!'

# Django - everything built-in
# - ORM
# - Admin panel
# - Authentication
# - Forms
# - Templates
# That's why I chose Django for VibeVault
```

In VibeVault: "I use Django because I needed ORM, Admin, Auth, and DRF integration out of the box."

How do you create REST APIs?

Interviewer: "How do you create REST APIs in Python?" **You:** "Using Django REST Framework with ViewSets, Serializers, and Routers."

```python
# 1. Model
class Memory(models.Model):
    title = models.CharField(max_length=255)
```

```python
# 2. Serializer
class MemorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Memory
        fields = '__all__'

# 3. ViewSet
class MemoryViewSet(viewsets.ModelViewSet):
    queryset = Memory.objects.all()
    serializer_class = MemorySerializer

# 4. Router
router = DefaultRouter()
router.register('memories', MemoryViewSet)

# Result: Full CRUD API at /memories/
```

---

10. TESTING AND DEBUGGING

How do you test Python code?

Interviewer: "How do you test code?" **You:** "Using pytest or Django's test framework with assertions."

```python
# pytest example
def test_analyze_sentiment():
    analyzer = AnalyzerPipeline()
    result = analyzer.analyze_sentiment("I love this!")

    assert result['label'] == 'positive'
    assert result['score'] > 0.5

# Django test
class MemoryTestCase(TestCase):
    def setUp(self):
        self.user = User.objects.create_user('test', 'test@test.com', 'pass')

    def test_create_memory(self):
        memory = Memory.objects.create(user=self.user, title='Test')
        self.assertEqual(memory.title, 'Test')
```

In VibeVault: "I have a `tests/` folder with pytest for API endpoints."

What is logging?

Interviewer: "How do you configure logging?" **You:** "Using Python's logging module with handlers and formatters."

```python
import logging

# Configure in settings.py
LOGGING = {
    'version': 1,
    'handlers': {
        'console': {
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        'django': {
            'handlers': ['console'],
            'level': 'INFO',
        },
    },
}

# Usage in code
logger = logging.getLogger(__name__)
logger.info("Starting analysis")
logger.warning("Slow request detected")
logger.error(f"Analysis failed: {error}")
```

In VibeVault: "I log warnings for slow requests in my middleware."

---

11. CODING & LOGICAL QUESTIONS

Reverse a string without slicing.

```python
def reverse_string(s):
    result = ""
    for char in s:
        result = char + result
    return result

# Or using list
def reverse_string(s):
    chars = list(s)
    left, right = 0, len(chars) - 1
    while left < right:
        chars[left], chars[right] = chars[right], chars[left]
        left += 1
        right -= 1
    return ''.join(chars)
```

Factorial using recursion.

```python
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n - 1)

# factorial(5) = 5 * 4 * 3 * 2 * 1 = 120
```

Check if prime.

```python
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True
```

Check if palindrome.

```python
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    return s == s[::-1]

# Without slicing
def is_palindrome(s):
    s = s.lower().replace(" ", "")
    left, right = 0, len(s) - 1
    while left < right:
        if s[left] != s[right]:
            return False
        left += 1
        right -= 1
    return True
```

Count frequency of elements.

```python
def count_frequency(lst):
    freq = {}
    for item in lst:
        freq[item] = freq.get(item, 0) + 1
    return freq
```

```python
# Using Counter
from collections import Counter
freq = Counter(['a', 'b', 'a', 'c', 'a'])
# Counter({'a': 3, 'b': 1, 'c': 1})
```

---

12. HR + PROJECT QUESTIONS

Explain your Python projects.

You: "VibeVault is an AI-powered journaling platform. The backend is Django with Django REST Framework. I used:

- **Celery** for async AI analysis
- **NumPy/Sentence-Transformers** for embeddings
- **NLTK/VADER** for sentiment analysis
- **WebSockets** via Django Channels for real-time notifications"

How did you optimize your Python code?

You: "Several ways:

1. **Lazy loading** - AI models load on first use, not at startup
2. **Database queries** - Used `select_related()` and `prefetch_related()` to reduce N+1 queries
3. **Caching** - Singleton pattern for the analyzer instance
4. **Async processing** - Heavy AI work offloaded to Celery workers"

How do you ensure scalability?

You: "

1. **Stateless API** - JWT auth means no server-side sessions
2. **Celery workers** - Can scale horizontally
3. **Database pooling** - Connection reuse with PostgreSQL
4. **Pagination** - Never load all records at once"

How do you manage version control?

You: "Git with GitHub.

- Feature branches for new work
- Pull requests for code review
- `.gitignore` for secrets and virtual environments
- Meaningful commit messages
- Tags for releases"