25 Python Interview Questions & Answers

Complete Technical Interview Guide for Python Developers

About This Guide

Python is the world's fastest-growing programming language, powering everything from web development and data science to artificial intelligence and automation. This comprehensive guide covers essential Python concepts frequently tested in technical interviews.

Last Updated: December 2024 | **Python Version Coverage:** Python 3.8 to Python 3.12

Table of Contents

💡 **Pro Tip:** Python interviews often focus on practical problem-solving. Practice coding daily on platforms like LeetCode and HackerRank.

1. Python Fundamentals

1 What is Python and its key features?

Python is a high-level, interpreted, dynamically-typed programming language known for its simplicity and readability.

Key Features:

- **Interpreted:** No compilation needed
- **Dynamically Typed:** No explicit type declarations
- **Garbage Collected:** Automatic memory management
- **Multi-paradigm:** Supports OOP, functional, procedural
- **Extensive Libraries:** Rich standard library
- **Cross-platform:** Runs on Windows, Linux, macOS
- **Open Source:** Free to use and distribute

```
 "python-keyword">class="python-keyword"># Simple Python Program
"python-keyword">class="python-keyword">print("python-keyword">class="

"python-keyword">class="python-keyword"># Dynamic typing example
x = "python-keyword">class="python-string">"Hello"      "python-keywor
x = "python-keyword">class="python-number">42           "python-keywor
x = ["python-keyword">class="python-number">1, "python-keyword">class=
```

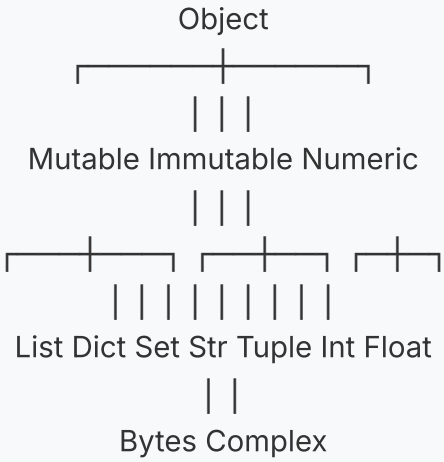
2 Python vs Other Languages

Feature	Python	Java	JavaScript
Typing	Dynamic	Static	Dynamic
Paradigm	Multi-paradigm	OOP-focused	Multi-paradigm
Performance	Slower (interpreted)	Faster (compiled)	Fast (JIT compiled)
Syntax	Simple, readable	Verbose	C-style
Use Cases	Data Science, ML, Web	Enterprise, Android	Web, Frontend

Python's Philosophy: "Readability counts" (Zen of Python). Python prioritizes clean, readable code over micro-optimizations.

3 What are Python's Data Types?

PYTHON DATA TYPE HIERARCHY



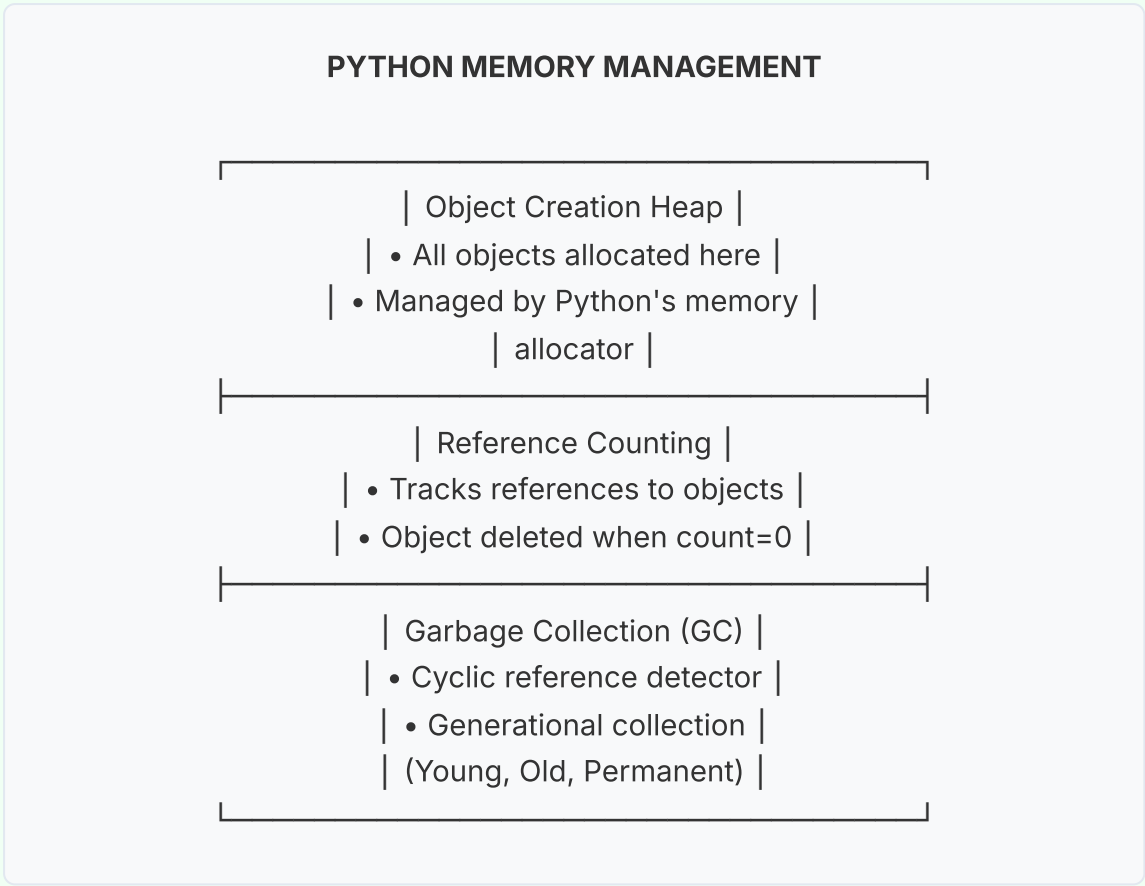

```
 "python-keyword">class="python-keyword"># Python Data Types Examples
"python-keyword">class="python-comment"># Immutable Types
num = "python-keyword">class="python-number">42                    "py
pi = "python-keyword">class="python-number">3.14159                "pyt
name = "python-keyword">class="python-string">"Python"             "py
flag = "python-keyword">class="python-keyword">"python-keyword">True
coordinates = ("python-keyword">class="python-number">10, "python-keyw

"python-keyword">class="python-comment"># Mutable Types
numbers = ["python-keyword">class="python-number">1, "python-keyword">
person = {"python-keyword">class="python-string">"name": "python-keywo
unique_nums = {"python-keyword">class="python-number">1, "python-keywo
```

4 Mutable vs Immutable Types

Mutable Types	Immutable Types
Can be changed after creation	Cannot be changed after creation
Lists, Dictionaries, Sets	Integers, Floats, Strings, Tuples
Stored by reference	Stored by value
Can be modified in-place	Operations create new objects

```
 "python-keyword">class="python-keyword"># Mutable Example (List)
list1 = ["python-keyword">class="python-number">1, "python-keyword">cl
list2 = list1          "python-keyword">class="python-comment"># Both
list1.append("python-keyword">class="python-number">4)
"python-keyword">class="python-keyword">print(list2)          "python-

"python-keyword">class="python-keyword">"># Immutable Example (String)
str1 = "python-keyword">class="python-string">"hello"
str2 = str1           "python-keyword">class="python-comment"># Both r
str1 = str1 + "python-keyword">class="python-string">" world"  "pythor
"python-keyword">class="python-keyword">print(str2)          "python-k
```

5 Memory Management in Python




```
 "python-keyword">class="python-keyword">"python-keyword">import sys
"python-keyword">class="python-keyword">"python-keyword">import gc

"python-keyword">class="python-comment"># Reference counting example
x = []              "python-keyword">class="python-comment"># Ref cour
y = x               "python-keyword">class="python-comment"># Ref cour
"python-keyword">class="python-keyword">"python-keyword">del x
"python-keyword">class="python-keyword">"python-keyword">del y

"python-keyword">class="python-comment"># Check reference count
obj = ["python-keyword">class="python-number">1, "python-keyword">clas
"python-keyword">class="python-keyword">print(sys.getrefcount(obj))  '

"python-keyword">class="python-comment"># Manual garbage collection
gc.collect()        "python-keyword">class="python-comment"># Force ga
```

Key Points:

- Python uses automatic memory management
- Reference counting + garbage collection
- Memory is allocated on the heap
- GIL (Global Interpreter Lock) affects threading

2. Data Structures & Collections

6 Lists vs Tuples vs Sets vs Dictionaries

Structure	Mutable	Ordered	Duplicate Items	Use Case
List	Yes	Yes	Allowed	Collection of items, sequencing
Tuple	No	Yes	Allowed	Fixed collection, hashable keys
Set	Yes	No	No	Unique items, membership tests
Dictionary	Yes	Yes (Python 3.7+)	Unique keys only	Key-value pairs, mappings

```
 "python-keyword">class="python-comment"># Lists - Ordered, Mutable
fruits = ["python-keyword">class="python-string">"apple", "python-keyw
fruits.append("python-keyword">class="python-string">"orange")
fruits["python-keyword">class="python-number">0] = "python-keyword">cl

"python-keyword">class="python-comment"># Tuples - Ordered, Immutable
coordinates = ("python-keyword">class="python-number">10.5, "python-ke
"python-keyword">class="python-comment"># coordinates[0] = 15.0  # Err

"python-keyword">class="python-comment"># Sets - Unordered, Unique ele
unique_numbers = {"python-keyword">class="python-number">1, "python-ke
unique_numbers.add("python-keyword">class="python-number">4)

"python-keyword">class="python-comment"># Dictionaries - Key-Value pai
person = {
    "python-keyword">class="python-string">"name": "python-keyword">cl
    "python-keyword">class="python-string">"age": "python-keyword">cla
    "python-keyword">class="python-string">"city": "python-keyword">cl
}
person["python-keyword">class="python-string">"email"] = "python-keywo
```

7 List Comprehensions vs Generator Expressions

```
 "python-keyword">class="python-comment"># List Comprehension - Create
squares = [x**"python-keyword">class="python-number">2 "python-keyword
"python-keyword">class="python-comment"># Output: [0, 1, 4, 9, 16, 25,

"python-keyword">class="python-comment"># Generator Expression - Lazy
squares_gen = (x**"python-keyword">class="python-number">2 "python-key
"python-keyword">class="python-comment"># squares_gen "python-keyword"
"python-keyword">class="python-keyword">"python-keyword">for square "p
    "python-keyword">class="python-keyword">print(square)  "python-key

"python-keyword">class="python-comment"># Dictionary Comprehension
square_dict = {x: x**"python-keyword">class="python-number">2 "python-
"python-keyword">class="python-comment"># Output: {0: 0, 1: 1, 2: 4, 3

"python-keyword">class="python-comment"># Set Comprehension
unique_squares = {x**"python-keyword">class="python-number">2 "python-
"python-keyword">class="python-comment"># Output: {0, 1, 4}
```

Memory Efficiency: Use generator expressions for large datasets. They produce items one at a time and don't store the entire sequence in memory.

8 String Manipulation in Python

```
 "python-keyword">class="python-comment"># String Basics
text = "python-keyword">class="python-string">"  Hello, Python World!

"python-keyword">class="python-comment"># Common string operations
"python-keyword">class="python-keyword">print(text.strip())
"python-keyword">class="python-keyword">print(text.upper())
"python-keyword">class="python-keyword">print(text.lower())
"python-keyword">class="python-keyword">print(text.replace("python-key
"python-keyword">class="python-keyword">print(text.split("python-keywo

"python-keyword">class="python-comment"># String formatting (3 ways)
name = "python-keyword">class="python-string">"Alice"
age = "python-keyword">class="python-number">30

"python-keyword">class="python-comment"># 1. f-strings (Python 3.6+) -
message = "python-keyword">class="python-string">f"Hello, {name}. You

"python-keyword">class="python-comment"># 2. format() method
message = "python-keyword">class="python-string">"Hello, {}. You are {

"python-keyword">class="python-comment"># 3. % formatting (legacy)
message = "python-keyword">class="python-string">"Hello, %s. You are %

"python-keyword">class="python-comment"># String slicing
text = "python-keyword">class="python-string">"Python Programming"
"python-keyword">class="python-keyword">print(text["python-keyword">cl
"python-keyword">class="python-keyword">print(text["python-keyword">cl
"python-keyword">class="python-keyword">print(text[::"python-keyword">
```

9 Collections Module - Advanced Data Structures

```
 "python-keyword">class="python-keyword">"python-keyword">from collect

"python-keyword">class="python-comment"># 1. defaultdict - Dictionary
word_count = defaultdict(int)
"python-keyword">class="python-keyword">"python-keyword">for word "pyt
    word_count[word] += "python-keyword">class="python-number">1
"python-keyword">class="python-comment"># No KeyError "python-keyword"

"python-keyword">class="python-comment"># 2. Counter - Count hashable
colors = ["python-keyword">class="python-string">"red", "python-keywor
color_count = Counter(colors)
"python-keyword">class="python-keyword">print(color_count.most_common(

"python-keyword">class="python-comment"># 3. deque - Double-ended queu
queue = deque(["python-keyword">class="python-number">1, "python-keywo
queue.append("python-keyword">class="python-number">4)       "python-k
queue.appendleft("python-keyword">class="python-number">0)   "python-k
queue.pop()          "python-keyword">class="python-comment"># Remove
queue.popleft()      "python-keyword">class="python-comment"># Remove

"python-keyword">class="python-comment"># 4. namedtuple - Tuple "pytho
Point = namedtuple("python-keyword">class="python-string">"Point", ["p
p = Point("python-keyword">class="python-number">10, "python-keyword">
"python-keyword">class="python-keyword">print(p.x, p.y)     "python-ke
"python-keyword">class="python-keyword">print(p["python-keyword">class
```

10 Sorting in Python

```
 "python-keyword">class="python-comment"># Basic sorting
numbers = ["python-keyword">class="python-number">3, "python-keyword">

"python-keyword">class="python-comment"># sorted() - returns new sorte
sorted_numbers = sorted(numbers)
"python-keyword">class="python-comment"># Original list unchanged

"python-keyword">class="python-comment"># list.sort() - sorts "python-
numbers.sort()  "python-keyword">class="python-comment"># Original lis

"python-keyword">class="python-comment"># Sorting "python-keyword">wit
words = ["python-keyword">class="python-string">"banana", "python-keyw
sorted_words = sorted(words, key=len)  "python-keyword">class="python-
"python-keyword">class="python-comment"># Output: ['date', 'apple', 'b

"python-keyword">class="python-comment"># Sorting dictionaries
scores = {"python-keyword">class="python-string">"Alice": "python-keyw

"python-keyword">class="python-comment"># Sort by key
sorted_by_name = dict(sorted(scores.items()))
"python-keyword">class="python-comment"># Output: {'Alice': 95, 'Bob':

"python-keyword">class="python-comment"># Sort by value
sorted_by_score = dict(sorted(scores.items(), key="python-keyword">cla
"python-keyword">class="python-comment"># Output: {'Bob': 87, 'Charlie

"python-keyword">class="python-comment"># Reverse sort
sorted_reverse = sorted(numbers, reverse="python-keyword">class="pytho
```

3. Functions & OOP

11 Functions in Python - *args, **kwargs, Decorators

```
 "python-keyword">class="python-comment"># Basic function
"python-keyword">class="python-keyword">"python-keyword">def greet(nam
    "python-keyword">class="python-keyword">"python-keyword">return "p

"python-keyword">class="python-comment"># *args - Variable positional
"python-keyword">class="python-keyword">"python-keyword">def sum_all(*
    "python-keyword">class="python-keyword">"python-keyword">return "p

"python-keyword">class="python-keyword">print(sum_all("python-keyword"
"python-keyword">class="python-keyword">print(sum_all("python-keyword"

"python-keyword">class="python-comment"># **kwargs - Variable keyword
"python-keyword">class="python-keyword">"python-keyword">def print_inf
    "python-keyword">class="python-keyword">"python-keyword">for key,
        "python-keyword">class="python-keyword">print("python-keyword"

print_info(name="python-keyword">class="python-string">"Alice", age="p

"python-keyword">class="python-comment"># Decorators - Modify functior
"python-keyword">class="python-keyword">"python-keyword">def timer_dec
    "python-keyword">class="python-keyword">"python-keyword">import ti
    "python-keyword">class="python-keyword">"python-keyword">def wrapp
        start = time.time()
        result = func(*args, **kwargs)
        end = time.time()
        "python-keyword">class="python-keyword">print("python-keyword"
        "python-keyword">class="python-keyword">"python-keyword">retur
    "python-keyword">class="python-keyword">"python-keyword">return wr

"python-keyword">class="python-comment"># Using decorator "python-keyw
@timer_decorator
"python-keyword">class="python-keyword">"python-keyword">def slow_func
    time.sleep("python-keyword">class="python-number">1)
    "python-keyword">class="python-keyword">"python-keyword">return "p

slow_function()  "python-keyword">class="python-comment"># Prints: "sl
```

12 Lambda Functions

```
 "python-keyword">class="python-comment"># Lambda syntax: "python-keyw

"python-keyword">class="python-comment"># Simple "python-keyword">lamb
square = "python-keyword">class="python-keyword">"python-keyword">lamb
"python-keyword">class="python-keyword">print(square("python-keyword">

"python-keyword">class="python-comment"># Lambda "python-keyword">with
add = "python-keyword">class="python-keyword">"python-keyword">lambda
"python-keyword">class="python-keyword">print(add("python-keyword">cla

"python-keyword">class="python-comment"># Using "python-keyword">lambd
numbers = ["python-keyword">class="python-number">1, "python-keyword">

"python-keyword">class="python-comment"># map() "python-keyword">with
squared = "python-keyword">class="python-keyword">list("python-keyword
"python-keyword">class="python-comment"># [1, 4, 9, 16, 25]

"python-keyword">class="python-comment"># filter() "python-keyword">w
even_numbers = "python-keyword">class="python-keyword">list("python-ke
"python-keyword">class="python-comment"># [2, 4]

"python-keyword">class="python-comment"># sorted() "python-keyword">w
words = ["python-keyword">class="python-string">"apple", "python-keywo
sorted_by_length = sorted(words, key="python-keyword">class="python-ke
"python-keyword">class="python-comment"># ['date', 'apple', 'banana',

"python-keyword">class="python-comment"># Lambda "python-keyword">in
operations = [
    ("python-keyword">class="python-string">"add", "python-keyword">cl
    ("python-keyword">class="python-string">"subtract", "python-keywor
    ("python-keyword">class="python-string">"multiply", "python-keywor
]

"python-keyword">class="python-keyword">"python-keyword">for name, fun
    "python-keyword">class="python-keyword">print("python-keyword">cla
```

13 Object-Oriented Programming in Python

```
"python-keyword">class="python-keyword">"python-keyword">class Person
    "python-keyword">class="python-comment"># Class variable (shared b
    species = "python-keyword">class="python-string">"Homo sapiens"

    "python-keyword">class="python-comment"># Constructor (__init__ me
    "python-keyword">class="python-keyword">"python-keyword">def __ini
        "python-keyword">class="python-comment"># Instance variables
        self.name = name
        self.age = age

    "python-keyword">class="python-comment"># Instance method
    "python-keyword">class="python-keyword">"python-keyword">def greet
        "python-keyword">class="python-keyword">"python-keyword">retur

    ">class="python-comment"># Class method
    @classmethod
    ">class="python-keyword">">def from_birth_year(cls, name, birth_ye
        age = ">class="python-number">2024 - birth_year
        ">class="python-keyword">">return cls(name, age)

    ">class="python-comment"># Static method (no access to self ">or c
    @staticmethod
    ">class="python-keyword">">def is_adult(age):
        ">class="python-keyword">">return age >= ">class="python-numbe

    ">class="python-comment"># Special methods (dunder methods)
    ">class="python-keyword">">def __str__(self):
        ">class="python-keyword">">return ">class="python-string">f"Pe

    ">class="python-keyword">">def __eq__(self, other):
        ">class="python-keyword">">return self.name == other.name ">cl

">class="python-comment"># Creating objects
person1 = Person(">class="python-string">"Alice", ">class="python-numb
person2 = Person.from_birth_year(">class="python-string">"Bob", ">clas

">class="python-keyword">print(person1.greet())
">class="python-keyword">print(person2)
">class="python-keyword">print(Person.is_adult(">class="python-number'
">class="python-keyword">print(person1.species)  ">class="python-comme
```

14 Inheritance & Polymorphism

```
 "python-keyword">class="python-keyword">"python-keyword">class Animal
    "python-keyword">class="python-keyword">"python-keyword">def __in
        self.name = name

    "python-keyword">class="python-keyword">"python-keyword">def speak
        "python-keyword">class="python-keyword">"python-keyword">raise

"python-keyword">class="python-keyword">"python-keyword">class Dog(An
    "python-keyword">class="python-keyword">"python-keyword">def speak
        "python-keyword">class="python-keyword">"python-keyword">retur

    "python-keyword">class="python-keyword">"python-keyword">def wag_t
        "python-keyword">class="python-keyword">"python-keyword">retur

"python-keyword">class="python-keyword">"python-keyword">class Cat(An
    "python-keyword">class="python-keyword">"python-keyword">def speak
        "python-keyword">class="python-keyword">"python-keyword">retur

    "python-keyword">class="python-keyword">"python-keyword">def purr(
        "python-keyword">class="python-keyword">"python-keyword">retur

"python-keyword">class="python-comment"># Multiple inheritance
"python-keyword">class="python-keyword">"python-keyword">class Pet:
    "python-keyword">class="python-keyword">"python-keyword">def __in
        self.owner = owner

    "python-keyword">class="python-keyword">"python-keyword">def owner
        "python-keyword">class="python-keyword">"python-keyword">retur

"python-keyword">class="python-keyword">"python-keyword">class PetDog(
    "python-keyword">class="python-keyword">"python-keyword">def __in
        Dog.__init__(self, name)
        Pet.__init__(self, owner)

"python-keyword">class="python-comment"># Using polymorphism
"python-keyword">class="python-keyword">"python-keyword">def animal_sc
    "python-keyword">class="python-keyword">"python-keyword">for anima
        "python-keyword">class="python-keyword">print(animal.speak())

"python-keyword">class="python-comment"># Create animals
dog = Dog("python-keyword">class="python-string">"Buddy")
cat = Cat("python-keyword">class="python-string">"Whiskers")
pet_dog = PetDog("python-keyword">class="python-string">"Max", "python

"python-keyword">class="python-keyword">print(dog.speak())      "pytho
"python-keyword">class="python-keyword">print(cat.speak())      "pytho
"python-keyword">class="python-keyword">print(pet_dog.owner_info())  "
```

```
"python-keyword">class="python-comment"># Method Resolution Order (MRO
"python-keyword">class="python-keyword">print(PetDog.__mro__)
"python-keyword">class="python-comment"># Output: (<"python-keyword">c
#          <"python-keyword">class '__main__.Animal'>, <"python-keywor
#          <"python-keyword">class 'object'>)
```

15 Encapsulation & Properties

```
"python-keyword">class="python-keyword">"python-keyword">class BankAc
    "python-keyword">class="python-keyword">"python-keyword">def __in
        self.owner = owner
        self._balance = initial_balance  "python-keyword">class="pytho

    "python-keyword">class="python-comment"># Getter property
    @property
    "python-keyword">class="python-keyword">"python-keyword">def balar
        "python-keyword">class="python-keyword">"python-keyword">retur

    "python-keyword">class="python-comment"># Setter property "python-
    @balance.setter
    "python-keyword">class="python-keyword">"python-keyword">def balar
        "python-keyword">class="python-keyword">"python-keyword">if am
            "python-keyword">class="python-keyword">"python-keyword">r
        self._balance = amount

    "python-keyword">class="python-keyword">"python-keyword">def depos
        "python-keyword">class="python-keyword">"python-keyword">if am
            "python-keyword">class="python-keyword">"python-keyword">r
        self.balance += amount  "python-keyword">class="python-comment

    "python-keyword">class="python-keyword">"python-keyword">def withd
        "python-keyword">class="python-keyword">"python-keyword">if am
            "python-keyword">class="python-keyword">"python-keyword">r
        "python-keyword">class="python-keyword">"python-keyword">if am
            "python-keyword">class="python-keyword">"python-keyword">r
        self.balance -= amount

"python-keyword">class="python-comment"># Using the "python-keyword">c
account = BankAccount("python-keyword">class="python-string">"Alice",

"python-keyword">class="python-keyword">print(account.balance)      "p
account.deposit("python-keyword">class="python-number">500)
"python-keyword">class="python-keyword">print(account.balance)      "p
account.withdraw("python-keyword">class="python-number">200)
"python-keyword">class="python-keyword">print(account.balance)      "p

"python-keyword">class="python-comment"># Direct access to protected a
"python-keyword">class="python-keyword">print(account._balance)     "p

">class="python-comment"># Trying to set negative balance through prop
">class="python-keyword">">try:
    account.balance = -">class="python-number">100    ">class="python-
```

```
">class="python-keyword">">except ValueError ">class="python-keyword">
    ">class="python-keyword">print(">class="python-string">f"Error: {e
```

Python Naming Conventions:

- `_single_leading_underscore`: "Internal use" (protected)
- `__double_leading_underscore`: Name mangling (private)
- `__double_underscore__`: Magic/dunder methods
- `single_trailing_underscore_`: Avoid naming conflicts

4. Advanced Python Concepts

16 Iterators, Generators, and yield

```
 "python-keyword">class="python-comment"># Iterator - object "python-k
"python-keyword">class="python-keyword">"python-keyword">class CountDo
    "python-keyword">class="python-keyword">"python-keyword">def __ini
        self.current = start

    "python-keyword">class="python-keyword">"python-keyword">def __ite
        "python-keyword">class="python-keyword">"python-keyword">retur

    "python-keyword">class="python-keyword">"python-keyword">def __nex
        "python-keyword">class="python-keyword">"python-keyword">if se
            "python-keyword">class="python-keyword">"python-keyword">r
        num = self.current
        self.current -= "python-keyword">class="python-number">1
        "python-keyword">class="python-keyword">"python-keyword">retur

"python-keyword">class="python-comment"># Using iterator
"python-keyword">class="python-keyword">"python-keyword">for num "pyth
    "python-keyword">class="python-keyword">print(num)  "python-keywor

"python-keyword">class="python-comment"># Generator function - uses "p
"python-keyword">class="python-keyword">"python-keyword">def fibonacci
    a, b = "python-keyword">class="python-number">0, "python-keyword">
    count = "python-keyword">class="python-number">0
    "python-keyword">class="python-keyword">"python-keyword">while cou
        "python-keyword">class="python-keyword">"python-keyword">yield
        a, b = b, a + b
        count += "python-keyword">class="python-number">1

"python-keyword">class="python-comment"># Using generator
"python-keyword">class="python-keyword">"python-keyword">for num "pyth
    "python-keyword">class="python-keyword">print(num)  "python-keywor

"python-keyword">class="python-comment"># Generator expression (simila
squares_gen = (x**"python-keyword">class="python-number">2 "python-key
"python-keyword">class="python-keyword">print("python-keyword">class="
"python-keyword">class="python-keyword">print("python-keyword">class="

"python-keyword">class="python-comment"># "python-keyword">yield "pyth
"python-keyword">class="python-keyword">"python-keyword">def chain(*it
    "python-keyword">class="python-keyword">"python-keyword">for it "p
        "python-keyword">class="python-keyword">"python-keyword">yield
```

```
"python-keyword">class="python-keyword">"python-keyword">for char "pyt
    "python-keyword">class="python-keyword">print(char)  "python-keywo
```

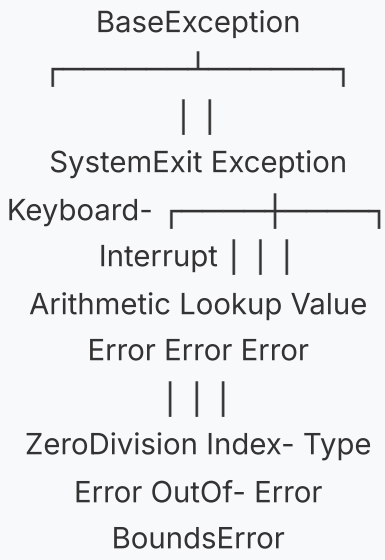
17 Context Managers & with statement

```
 "python-keyword">class="python-comment"># Using context manager "pyth
"python-keyword">class="python-keyword">"python-keyword">with "python-
    file.write("python-keyword">class="python-string">"Hello, World!")
"python-keyword">class="python-comment"># File automatically closed

"python-keyword">class="python-comment"># Creating custom context mana
"python-keyword">class="python-keyword">"python-keyword">class Timer:
    "python-keyword">class="python-keyword">"python-keyword">def __in
        self.name = name

    "python-keyword">class="python-keyword">"python-keyword">def __ent
        self.start = time.time()
        "python-keyword">class="python-keyword">"python-keyword">retur

    "python-keyword">class="python-keyword">"python-keyword">def __ex
        self.end = time.time()
        "python-keyword">class="python-keyword">print("python-keyword"
        "python-keyword">class="python-keyword">"python-keyword">retur

">class="python-comment"># Using custom context manager
">class="python-keyword">">with Timer(">class="python-string">"Process
    time.sleep(">class="python-number">1)
    ">class="python-comment"># Prints: "Process took 1.00 seconds"

">class="python-comment"># Creating context manager using contextlib
">class="python-keyword">">from contextlib ">class="python-keyword">">

@contextmanager
">class="python-keyword">">def temp_change_dir(new_dir):
    old_dir = os.getcwd()
    os.chdir(new_dir)
    ">class="python-keyword">">try:
        ">class="python-keyword">">yield
    ">class="python-keyword">">finally:
        os.chdir(old_dir)

">class="python-comment"># Using the context manager
">class="python-keyword">">with temp_change_dir(">class="python-string
    ">class="python-keyword">print(">class="python-string">f"Current d
">class="python-comment"># Directory automatically restored

">class="python-comment"># Multiple context managers
">class="python-keyword">">with ">class="python-keyword">open(">class=
```

```
 ">class="python-keyword">open(">class="python-string">"output.txt
output_file.write(input_file.read())
```

18 Error Handling & Exceptions

PYTHON EXCEPTION HIERARCHY



```
 "python-keyword">class="python-comment"># Basic "python-keyword">try-
"python-keyword">class="python-keyword">"python-keyword">try:
    result = "python-keyword">class="python-number">10 / "python-keywo
"python-keyword">class="python-keyword">"python-keyword">except ZeroD
    "python-keyword">class="python-keyword">print("python-keyword">cla

"python-keyword">class="python-comment"># Multiple exceptions
"python-keyword">class="python-keyword">"python-keyword">try:
    value = int("python-keyword">class="python-string">"not_a_number")
    result = "python-keyword">class="python-number">10 / value
"python-keyword">class="python-keyword">"python-keyword">except (Value
    "python-keyword">class="python-keyword">print("python-keyword">cla

"python-keyword">class="python-comment"># "python-keyword">else "pytho
"python-keyword">class="python-keyword">"python-keyword">try:
    file = "python-keyword">class="python-keyword">open("python-keywor
    data = file.read()
"python-keyword">class="python-keyword">"python-keyword">except FileNo
    "python-keyword">class="python-keyword">print("python-keyword">cla
"python-keyword">class="python-keyword">"python-keyword">else:
    "python-keyword">class="python-keyword">print("python-keyword">cla
    "python-keyword">class="python-comment"># "python-keyword">else ru
"python-keyword">class="python-keyword">"python-keyword">finally:
    "python-keyword">class="python-keyword">print("python-keyword">cla
    "python-keyword">class="python-comment"># "python-keyword">finally

"python-keyword">class="python-comment"># Custom exceptions
"python-keyword">class="python-keyword">"python-keyword">class Insuff
    "python-keyword">class="python-keyword">"python-keyword">def __in
        self.balance = balance
        self.amount = amount
        message = "python-keyword">class="python-string">f"Insufficien
        "python-keyword">class="python-keyword">super().__init__(messa

"python-keyword">class="python-keyword">"python-keyword">class BankAcc
    "python-keyword">class="python-keyword">"python-keyword">def witho
        "python-keyword">class="python-keyword">"python-keyword">if am
            "python-keyword">class="python-keyword">"python-keyword">r
        self.balance -= amount

"python-keyword">class="python-comment"># Raising exceptions
"python-keyword">class="python-keyword">"python-keyword">def validate_
    "python-keyword">class="python-keyword">"python-keyword">if age <
        "python-keyword">class="python-keyword">"python-keyword">raise
    "python-keyword">class="python-keyword">"python-keyword">if age >
```

```
        "python-keyword">class="python-keyword">"python-keyword">raise
    "python-keyword">class="python-keyword">"python-keyword">return "p
```

19 Multithreading vs Multiprocessing

Aspect	Threading	Multiprocessing
Memory	Shared memory	Separate memory
GIL	Affected by GIL	Bypasses GIL
Use Case	I/O-bound tasks	CPU-bound tasks
Overhead	Low	High
Communication	Easy (shared memory)	Complex (IPC needed)

```
 "python-keyword">class="python-keyword">"python-keyword">import threa
"python-keyword">class="python-keyword">"python-keyword">import multip
"python-keyword">class="python-keyword">"python-keyword">import time

"python-keyword">class="python-comment"># Threading example (I/O-bound
"python-keyword">class="python-keyword">"python-keyword">def download_
    "python-keyword">class="python-keyword">print("python-keyword">cla
    time.sleep("python-keyword">class="python-number">2)  "python-keyw
    "python-keyword">class="python-keyword">print("python-keyword">cla

"python-keyword">class="python-comment"># Create threads
threads = []
"python-keyword">class="python-keyword">"python-keyword">for i "python
    thread = threading.Thread(target=download_file, args=("python-keyw
    threads.append(thread)
    thread.start()

"python-keyword">class="python-comment"># Wait "python-keyword">for al
"python-keyword">class="python-keyword">"python-keyword">for thread "p
    thread.join()

"python-keyword">class="python-comment"># Multiprocessing example (CPU
"python-keyword">class="python-keyword">"python-keyword">def calculate
    "python-keyword">class="python-comment"># CPU-intensive task
    result[index] = [n**"python-keyword">class="python-number">2 "pyth

"python-keyword">class="python-comment"># Create processes
numbers = ["python-keyword">class="python-number">1, "python-keyword">
result = multiprocessing.Array("python-keyword">class="python-string">
processes = []

"python-keyword">class="python-keyword">"python-keyword">for i "python
    p = multiprocessing.Process(target=calculate_square,
                                args=(numbers[i::"python-keyword">class
    processes.append(p)
    p.start()

"python-keyword">class="python-keyword">"python-keyword">for p "python
    p.join()

"python-keyword">class="python-keyword">print("python-keyword">class="
```

GIL (Global Interpreter Lock): Python's GIL allows only one thread to execute Python bytecode at a time. Use multiprocessing for CPU-bound parallel tasks.

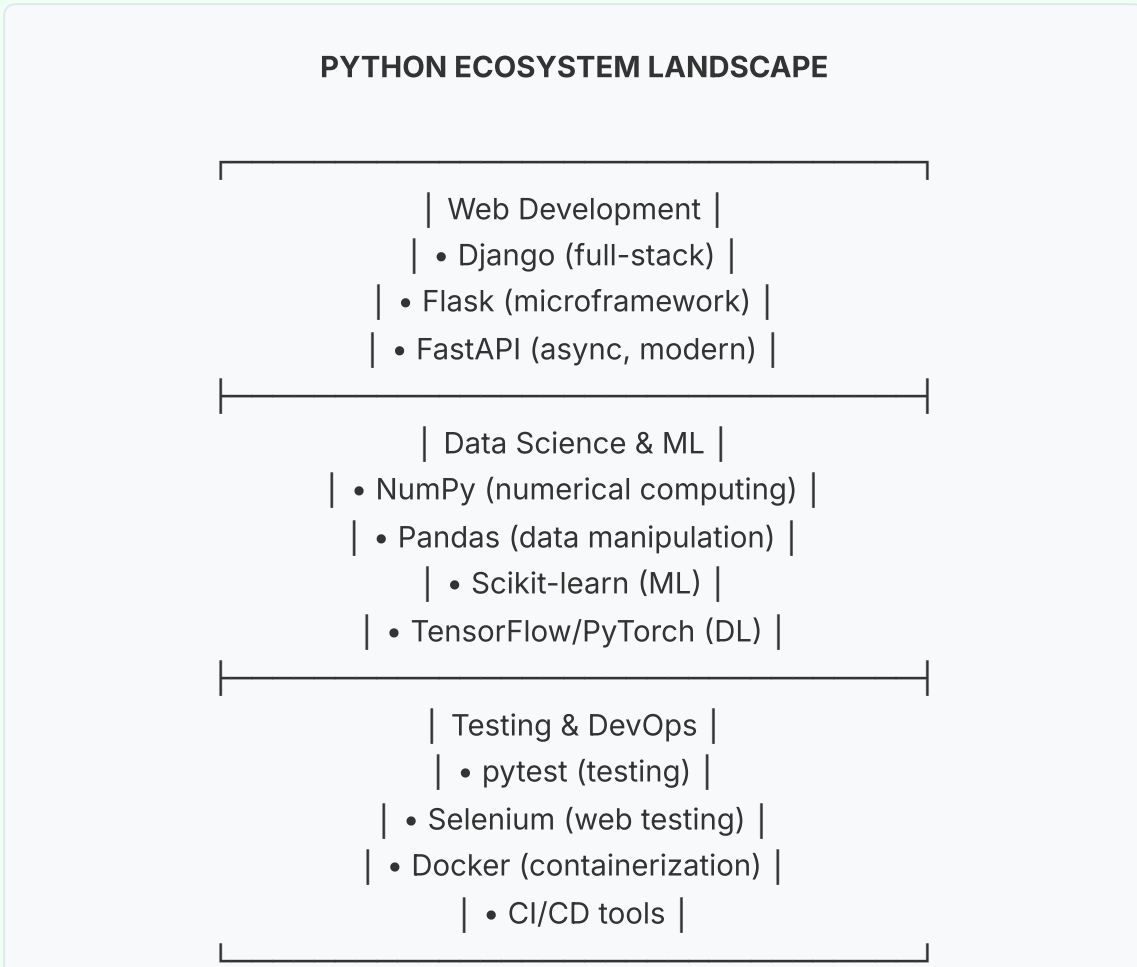
20 Metaprogramming & Decorators Advanced

```
 "python-keyword">class="python-comment"># Class decorators
"python-keyword">class="python-keyword">"python-keyword">def add_metho
    "python-keyword">class="python-keyword">"python-keyword">def new_m
        "python-keyword">class="python-keyword">"python-keyword">retur
    cls.dynamic_method = new_method
    ">class="python-keyword">">return cls

@add_method
">class="python-keyword">">class MyClass:
    ">class="python-keyword">">pass

obj = MyClass()
">class="python-keyword">print(obj.dynamic_method())  ">class="python-

"python-keyword">class="python-comment"># Property decorators "python-
"python-keyword">class="python-keyword">"python-keyword">class Circle:
    "python-keyword">class="python-keyword">"python-keyword">def __ini
        self.radius = radius
        self._area = "python-keyword">class="python-keyword">"python-k

    @property
    "python-keyword">class="python-keyword">"python-keyword">def area(
        "python-keyword">class="python-keyword">"python-keyword">if se
            "python-keyword">class="python-keyword">print("python-keyw
            self._area = "python-keyword">class="python-number">3.1415
        "python-keyword">class="python-keyword">"python-keyword">retur

circle = Circle("python-keyword">class="python-number">5)
"python-keyword">class="python-keyword">print(circle.area)  "python-ke
"python-keyword">class="python-keyword">print(circle.area)  "python-ke

"python-keyword">class="python-comment"># Decorator "python-keyword">w
"python-keyword">class="python-keyword">"python-keyword">def repeat(n)
    "python-keyword">class="python-keyword">"python-keyword">def decor
        "python-keyword">class="python-keyword">"python-keyword">def w
            "python-keyword">class="python-keyword">"python-keyword">f
                result = func(*args, **kwargs)
            "python-keyword">class="python-keyword">"python-keyword">r
        "python-keyword">class="python-keyword">"python-keyword">retur
    "python-keyword">class="python-keyword">"python-keyword">return de

@repeat("python-keyword">class="python-number">3)
"python-keyword">class="python-keyword">"python-keyword">def say_hello
    "python-keyword">class="python-keyword">print("python-keyword">cla

say_hello("python-keyword">class="python-string">"Alice")  "python-key
```

```
"python-keyword">class="python-comment"># Using functools.wraps to pre
"python-keyword">class="python-keyword">"python-keyword">from functool

"python-keyword">class="python-keyword">"python-keyword">def log_calls
    @wraps(func)  "python-keyword">class="python-comment"># Preserves
    "python-keyword">class="python-keyword">"python-keyword">def wrapp
        "python-keyword">class="python-keyword">print("python-keyword"
        "python-keyword">class="python-keyword">"python-keyword">retur
    "python-keyword">class="python-keyword">"python-keyword">return wr
```

5. Python Ecosystem & Tools

21 Popular Python Libraries & Frameworks



Essential Libraries:

- **Web:** Django, Flask, FastAPI, Requests
- **Data Science:** NumPy, Pandas, Matplotlib, Seaborn
- **Machine Learning:** Scikit-learn, TensorFlow, PyTorch
- **Database:** SQLAlchemy, Psycopg2, SQLite3
- **Testing:** pytest, unittest, coverage.py
- **Utilities:** Click (CLI), Celery (tasks), Redis (cache)

22 Virtual Environments & Package Management

```
 "python-keyword">class="python-comment"># Creating virtual environmen
"python-keyword">class="python-comment"># Terminal commands:
python -m venv myenv          "python-keyword">class="python-comment">
"python-keyword">class="python-comment"># On Windows:
myenv\Scripts\activate        "python-keyword">class="python-comment">
"python-keyword">class="python-comment"># On Linux/Mac:
source myenv/bin/activate     "python-keyword">class="python-comment">
deactivate                    "python-keyword">class="python-comment">

"python-keyword">class="python-comment"># Using pip (Python package ma
pip install package_name      "python-keyword">class="python-comment">
pip install package_name==1.0.0  "python-keyword">class="python-commer
pip install -r requirements.txt  "python-keyword">class="python-commer
pip list                     "python-keyword">class="python-comment">#
pip freeze > requirements.txt  "python-keyword">class="python-comment"

"python-keyword">class="python-comment"># requirements.txt example
"python-keyword">class="python-comment"># Flask==2.3.2
"python-keyword">class="python-comment"># pandas>=1.5.0
"python-keyword">class="python-comment"># numpy
"python-keyword">class="python-comment"># requests

"python-keyword">class="python-comment"># Using pipenv (alternative)
pip install pipenv           "python-keyword">class="python-comment">#
pipenv install package_name  "python-keyword">class="python-comment">#
pipenv shell                "python-keyword">class="python-comment"># 
pipenv lock                "python-keyword">class="python-comment"># C

"python-keyword">class="python-comment"># Using poetry (modern alterna
poetry new project_name      "python-keyword">class="python-comment">#
poetry add package_name      "python-keyword">class="python-comment">#
poetry install              "python-keyword">class="python-comment"># 
poetry update               "python-keyword">class="python-comment"># 
```

23 Testing in Python

```
 "python-keyword">class="python-keyword">"python-keyword">import unitt
"python-keyword">class="python-keyword">"python-keyword">import pytest

"python-keyword">class="python-comment"># Using unittest (built-"pytho
"python-keyword">class="python-keyword">"python-keyword">class TestMat
    "python-keyword">class="python-keyword">"python-keyword">def setUp
        "python-keyword">class="python-comment"># Runs before each tes
        self.calculator = Calculator()

    "python-keyword">class="python-keyword">"python-keyword">def tearD
        "python-keyword">class="python-comment"># Runs after each test
        "python-keyword">class="python-keyword">"python-keyword">del s

    "python-keyword">class="python-keyword">"python-keyword">def test_
        self.assertEqual(self.calculator.add("python-keyword">class="p

    "python-keyword">class="python-keyword">"python-keyword">def test_
        "python-keyword">class="python-keyword">"python-keyword">with
            self.calculator.divide("python-keyword">class="python-numb

    @unittest.skip("python-keyword">class="python-string">"Temporarily
    "python-keyword">class="python-keyword">"python-keyword">def test_
        self.fail("python-keyword">class="python-string">"This won't r

">class="python-comment"># Using pytest (popular third-party)
">class="python-keyword">">def test_addition():
    ">assert ">class="python-number">2 + ">class="python-number">3 ==

">class="python-keyword">">def test_list_contains():
    items = [">class="python-number">1, ">class="python-number">2, ">c
    ">assert ">class="python-number">2 ">class="python-keyword">">in

">class="python-comment"># pytest fixtures
@pytest.fixture
">class="python-keyword">">def sample_data():
    ">class="python-keyword">">return {">class="python-string">"name":

">class="python-keyword">">def test_data_processing(sample_data):
    ">assert sample_data[">class="python-string">"name"] == ">class="p

">class="python-comment"># Mocking ">with unittest.mock
">class="python-keyword">">from unittest.mock ">class="python-keyword"

">class="python-keyword">">def test_api_call():
    ">class="python-keyword">">with patch(">class="python-string">'rec
        mock_get.return_value.status_code = ">class="python-number">20
```

```
        response = make_api_call()
        ">assert response.status_code == ">class="python-number">200
```

24 Python Performance Optimization

Optimization Techniques:

Technique	When to Use	Example
Use built-in functions	Always	`sum()` instead of loop
List comprehensions	Simple transformations	Faster than for loops
Generator expressions	Large datasets	Memory efficient
Local variables	Inside loops	Access is faster
String joining	Multiple concatenations	`''.join()` vs `+`
Use sets for membership	Frequent lookups	O(1) vs O(n) for lists

```
 "python-keyword">class="python-comment"># ❌ Inefficient code
result = "python-keyword">class="python-string">""
"python-keyword">class="python-keyword">"python-keyword">for word "pyt
    result += word  "python-keyword">class="python-comment"># Creates

"python-keyword">class="python-comment"># ✅ Efficient code
result = "python-keyword">class="python-string">"".join(words)  "pytho

"python-keyword">class="python-comment"># ❌ Inefficient membership t
"python-keyword">class="python-keyword">"python-keyword">if item "pyth
    "python-keyword">class="python-keyword">"python-keyword">pass

"python-keyword">class="python-comment"># ✅ Efficient membership tes
my_set = "python-keyword">class="python-keyword">set(my_list)
"python-keyword">class="python-keyword">"python-keyword">if item "pyth
    "python-keyword">class="python-keyword">"python-keyword">pass

"python-keyword">class="python-comment"># Using timeit "python-keyword
"python-keyword">class="python-keyword">"python-keyword">import timeit

code1 = "python-keyword">class="python-string">"""
result = []
">for i ">in range(1000):
    result.append(i**2)
"""

code2 = "python-keyword">class="python-string">"""
result = [i**2 ">for i ">in range(1000)]
"""

time1 = timeit.timeit(code1, number="python-keyword">class="python-num
time2 = timeit.timeit(code2, number="python-keyword">class="python-num
"python-keyword">class="python-keyword">print("python-keyword">class="
```

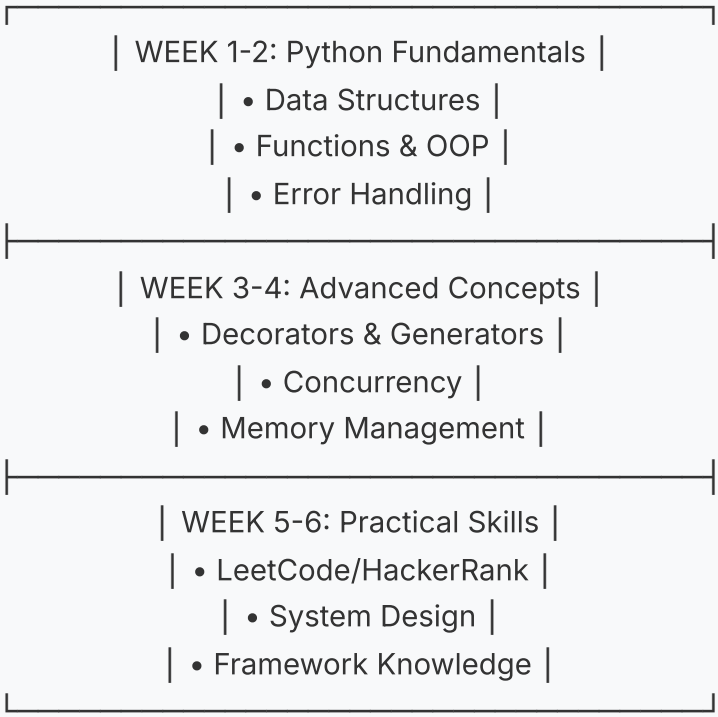
25 Modern Python Features (3.8-3.12)

```
 "python-keyword">class="python-comment"># Python 3.8: Walrus operator
"python-keyword">class="python-keyword">"python-keyword">while (line :
    "python-keyword">class="python-keyword">print("python-keyword">cla

"python-keyword">class="python-comment"># Python 3.9: Dictionary unior
dict1 = {"python-keyword">class="python-string">"a": "python-keyword">
dict2 = {"python-keyword">class="python-string">"c": "python-keyword">
combined = dict1 | dict2  "python-keyword">class="python-comment"># Me
dict1 |= dict2            "python-keyword">class="python-comment"># Up

"python-keyword">class="python-comment"># Python 3.10: Structural Patt
"python-keyword">class="python-keyword">"python-keyword">def handle_co
    "python-keyword">class="python-keyword">match command.split():
        "python-keyword">class="python-keyword">case ["python-keyword"
            "python-keyword">class="python-keyword">print("python-keyw
        "python-keyword">class="python-keyword">case ["python-keyword"
            "python-keyword">class="python-keyword">print("python-keyw
        "python-keyword">class="python-keyword">case ["python-keyword"
            "python-keyword">class="python-keyword">print("python-keyw
        "python-keyword">class="python-keyword">case _:
            "python-keyword">class="python-keyword">print("python-keyw

"python-keyword">class="python-comment"># Python 3.11: Exception group
"python-keyword">class="python-keyword">"python-keyword">try:
    "python-keyword">class="python-keyword">"python-keyword">raise Exc
        "python-keyword">class="python-string">"Multiple errors",
        [ValueError("python-keyword">class="python-string">"Bad value"
    )
"python-keyword">class="python-keyword">"python-keyword">except* Value
    "python-keyword">class="python-keyword">print("python-keyword">cla
"python-keyword">class="python-keyword">"python-keyword">except* TypeE
    "python-keyword">class="python-keyword">print("python-keyword">cla

"python-keyword">class="python-comment"># Python 3.12: Type parameter
"python-keyword">class="python-keyword">"python-keyword">def max[T](a:
    "python-keyword">class="python-keyword">"python-keyword">return a

"python-keyword">class="python-comment"># Python 3.12: F-strings enhan
name = "python-keyword">class="python-string">"Alice"
"python-keyword">class="python-keyword">print("python-keyword">class="
```

6. Interview Preparation Strategy

26 How to Prepare for Python Interviews?

6-WEEK PREPARATION ROADMAP



Common Coding Problem Categories:

- **Strings:** Palindrome, Anagrams, String manipulation
- **Arrays/Lists:** Two Sum, Maximum Subarray, Rotations
- **Linked Lists:** Reverse, Detect Cycle, Merge
- **Trees:** Traversals, BST operations, Path Sum
- **Graphs:** BFS/DFS, Shortest Path, Connected Components
- **Dynamic Programming:** Fibonacci, Knapsack, LCS
- **System Design:** Design URL shortener, Cache, API

> **Behavioral Questions:** Prepare STAR method answers (Situation, Task, Action, Result) for: • Biggest technical challenge
> • Team conflict resolution
> • Project you're proud of
> • Handling tight deadlines

7. Questions to Ask Interviewers

Technical Questions:

- "What Python version does the team use, and what's the migration plan for newer versions?"
- "How does the team handle dependency management and virtual environments?"
- "What's the typical development workflow and CI/CD pipeline?"
- "How are code reviews conducted, and what coding standards are followed?"
- "What testing frameworks and coverage expectations are there?"

Team & Project Questions:

- "What does the onboarding process look like for new Python developers?"
- "What's the team's approach to technical debt and refactoring?"
- "How are architectural decisions made in the team?"
- "What opportunities are there for learning and conference attendance?"
- "What's the career growth path for Python developers here?"

8. Additional Resources

Recommended Learning Resources:

- **Official Documentation:**
  - Python.org Documentation
  - PEP Index (Python Enhancement Proposals)
  - Python Standard Library Reference
- **Online Platforms:**
  - LeetCode (coding practice)
  - HackerRank (Python challenges)
  - Real Python (tutorials & articles)
  - Coursera/edX (Python courses)
- **Books:**
  - "Fluent Python" - Luciano Ramalho
  - "Python Cookbook" - David Beazley
  - "Effective Python" - Brett Slatkin
  - "Clean Code in Python" - Mariano Anaya
- **Communities:**
  - Stack Overflow (python tag)
  - Python Discord Server
  - Reddit r/Python, r/learnpython
  - Local Python User Groups

Certification Paths:

- **PCAP:** Certified Associate in Python Programming
- **PCPP:** Certified Professional in Python Programming
- **Specialized:** Django Certification, Data Science with Python
- **Vendor:** AWS Certified Developer, Google Cloud Python

9. Final Checklist Before Interview

- ✓ Review all Python fundamentals in this guide
- ✓ Practice 2-3 coding problems daily (30-60 minutes)
- ✓ Prepare 3-5 real project examples to discuss
- ✓ Research the company's tech stack and projects
- ✓ Prepare questions for the interviewer
- ✓ Test your coding environment if interview is remote
- ✓ Review common algorithms and data structures
- ✓ Practice explaining your thought process aloud
- ✓ Get good rest before the interview day
- ✓ Arrive early (virtual or in-person)

💡 **Python-Specific Tips:**

1. Know when to use list vs tuple vs set vs dict
2. Understand Python's memory model and GIL
3. Be comfortable with list/dict comprehensions
4. Know common standard library modules
5. Understand decorators and context managers
6. Be ready to discuss Python 3.x features

⚠️ **Common Pitfalls to Avoid:**

- Don't forget about Python's dynamic typing
- Remember that default arguments are evaluated once
- Understand shallow vs deep copy
- Know when to use `is` vs `==`
- Remember that Python uses reference counting + GC

Conclusion

Python's popularity continues to grow across domains from web development to data science and artificial intelligence. Mastering these concepts will not only help you succeed in interviews but also build a strong foundation for your career as a Python developer.

Key Takeaways:

- Understand Python's philosophy and design choices
- Master both basic and advanced language features
- Practice problem-solving with Python's unique capabilities
- Stay updated with modern Python features (3.8+)
- Build real projects to apply your knowledge
- Contribute to open-source Python projects

Remember: Python values readability and simplicity. Write clean, maintainable code, and be prepared to explain your design choices during interviews.

THE PYTHON ZEN (Tim Peters)

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
...