50 Python Interview Questions & Answers

Learn Python, Crack Interviews!

SLIDE 1 : PYTHON BASICS

1 What is Python?

→ Python is a high-level, interpreted, object-oriented programming language known for its simple and readable syntax.

2 What are Python's key features?

→
- Easy to learn and use
- Interpreted language
- Dynamically typed
- Object-oriented
- Large standard library
- Cross-platform support

Why Python?

- ☆ Beginner friendly
- ☆ Versatile
- ☆ Huge community support
- ☆ Used in Web, AI, Data Science, Automation, and more!

3 What is the difference between interpreted and compiled languages?

→ Interpreted languages execute code line by line, while compiled languages convert the entire program into machine code before execution.

Interpreted	Compiled
• Executes line by line	• Compiles entire program first
• No separate compilation step	• Faster execution
• Examples: Python, JavaScript	• Examples: C, C++, Java

4 What is PEP 8?

→ PEP 8 is Python's style guide that provides conventions for writing clean, readable and consistent code.

PEP 8 helps in:

- ✓ Improving code readability
- ✓ Maintaining consistency
- ✓ Collaborating better in teams

5 What are variables in Python?

→ Variables are names used to store data values in memory.

Example :

```
x = 10                  # x is integer
name = "Python"         # name is string
is_fun = True           # is_fun is boolean
print(x, name, is_fun)
```

Practice Today, Code Tomorrow, Succeed Always!

50 Python Interview Questions & Answers

Learn Python, Crack Interviews!

SLIDE 2 : DATA TYPES

6 What are Python's built-in data types?

→ Python has the following built-in data types:

int, float, complex, str, list, tuple, set, dict, bool, NoneType

7 What is the difference between a list and a tuple?

→ List is mutable (can be changed).

→ Tuple is immutable (cannot be changed after creation).

List	Tuple
• Uses [ ]	• Uses ( )
• Mutable	• Immutable
• Slower	• Faster
• More methods	• Fewer methods
Example:	Example:
lst = [1, 2, 3]	tup = (1, 2, 3)
lst[0] = 10 ✓	tup[0] = 10 ✗

8 What is a dictionary?

→ A dictionary stores data in key-value pairs.

Sample Code

```
student = {"name": "John", "age": 22}
print(student["name"])    # John
student["age"] = 23
print(student)   # {'name': 'John', 'age': 23}
```

9 What is a set?

→ A set is an unordered collection of unique elements.

Sample Code

```
numbers = {1, 2, 2, 3, 3, 4}
print(numbers)  # {1, 2, 3, 4}
numbers.add(5)
numbers.remove(2)
print(numbers)  # {1, 3, 4, 5}
```

1,2,3

10 What is type casting?

→ Type casting is the process of converting one data type into another.

Sample Code

```
x = "10"              # string
y = int(x)            # type casting to int
z = float(y)          # type casting to float
print(x, type(x))     # 10 <class 'str'>
print(y, type(y))     # 10 <class 'int'>
print(z, type(z))     # 10.0 <class 'float'>
```

Common Type Conversions

→ int()	→ to integer
→ float()	→ to float
→ str()	→ to string
→ list()	→ to list
→ tuple()	→ to tuple
→ set()	→ to set

Understand Data Types Well → Write Better Code!

50 Python Interview Questions & Answers

Keep Practicing, Keep Growing!

SLIDE 3 : OPERATORS

11 What are arithmetic operators?

→ Used to perform mathematical operations.

Operator	Name	Example	Result
+	Addition	5 + 3	8
-	Subtraction	5 - 3	2
*	Multiplication	5 * 3	15
/	Division	5 / 2	2.5
%	Modulus	5 % 2	1
**	Exponent	2 ** 3	8
//	Floor Division	5 // 2	2

12 What is the difference between = and == ?

→ = is an assignment operator.

→ == is a comparison operator.

Sample Code

```
x = 10          # assignment (stores value)
y = 10          # assignment
print(x == y)   # True (comparison)
print(x = y)    # ✗ Syntax Error
```

13 What are logical operators?

→ They are used to combine conditional statements.

Operator	Meaning	Example	Result
and	True if both are True	True and False	False
or	True if any one is True	True or False	True
not	Reverses the result	not True	False

14 What is the modulus operator?

→ The % operator returns the remainder after division.

Sample Code

```
print(10 % 3)      # 1
print(25 % 4)      # 1
print(-10 % 3)     # 2 (remainder is always non-negative)
```

15 What is operator precedence?

→ Operator precedence determines the order in which operations are evaluated.

→ Higher precedence operators are evaluated first.

Precedence Order (High → Low)

1. ( ) Parentheses
2. ** Exponent
3. +x, -x, ~x Unary plus, minus, bitwise NOT
4. *, /, //, % Multiplication, Division, Floor Division, Modulus
5. +, - Addition, Subtraction
6. <, <=, >, >=, !=, == Comparison
7. not Logical NOT
8. and Logical AND
9. or Logical OR
10. =, +=, -=, *=, /= Assignment operators

Sample Expression

```
x = 2 + 3 * 4 ** 2
        ①
  = 2 + 3 * 16   (Step 1: 4 ** 2 = 16)
        ②
  = 2 + 48       (Step 2: 3 * 16 = 48)
  = 50           (Step 3: 2 + 48 = 50)
```

Understanding operators well helps you write efficient and bug-free code! ☺

50 Python Interview Questions & Answers

Control the Flow, Control the Program!

SLIDE 4 : CONTROL FLOW

16 What is an if statement?

→ An if statement executes a block of code only when the condition is True.

Sample Code

```
age = 18
if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are not eligible.")
```

17 What is the difference between for loop and while loop?

→ for loop is used to iterate over a sequence (list, tuple, string, etc.).

→ while loop is used to repeat a block of code as long as a condition is True.

for Loop	while Loop
• Used with a sequence	• Used with a condition
• Number of iterations is known	• Number of iterations is unknown
• Example: iterate over list, string, range	• Example: user input, condition-based tasks

18 What does break do?

→ break statement is used to exit the loop immediately.

Sample Code

```
for i in range(1, 6):
    if i == 3:
        break   # loop terminates here
    print(i)
```

Output: 1 2

19 What does continue do?

→ continue statement skips the current iteration and moves to the next iteration.

Sample Code

```
for i in range(1, 6):
    if i == 3:
        continue   # skip the rest and go next
    print(i)
```

Output: 1 2 4 5

20 What is a nested loop?

→ A loop inside another loop is called a nested loop. It is useful for working with multi-dimensional data.

Sample Code

```
for i in range(1, 4):       # Outer loop
    for j in range(1, 4):   # Inner loop
        print(f"i={i}, j={j}")
```

Output

```
i=1, j=1
i=1, j=2
i=1, j=3
i=2, j=1
i=2, j=2
i=2, j=3
i=3, j=1
i=3, j=2
i=3, j=3
```

Quick Tips

- Use if to make decisions.
- Use for when you know how many times to loop.
- Use while when the number of iterations is unknown.
- Use break to exit early.
- Use continue to skip and move ahead.
- Nested loops are powerful! Use them wisely.

Master Control Flow → Write Smart & Efficient Python Code! ☺

50 Python Interview Questions & Answers

Functions Make Code Reusable & Better!

SLIDE 5 : FUNCTIONS

21 What is a function?

→ A function is a reusable block of code that performs a specific task.

Sample Code

```
def greet():
    print("Hello, Python!")

greet()   # Function call
```

22 What is the difference between parameters and arguments?

→ Parameters are variables in the function definition.

→ Arguments are the values passed to the function when it is called.

Sample Code

```
def add(a, b):      # a, b are parameters
    return a + b
result = add(5, 7)   # 5, 7 are arguments
print(result)        # Output: 12
```

23 What is a return statement?

→ A return statement sends a value back from the function to the caller.

Sample Code

```
def square(x):
    return x * x
y = square(4)
print(y)   # Output: 16
```

24 What are default arguments?

→ Arguments with predefined values that are used if no value is passed.

Sample Code

```
def greet(name = "Guest"):
    return f"Hello, {name}!"
print(greet())          # Hello, Guest!
print(greet("Alice"))   # Hello, Alice!
```

25 What is a lambda function?

→ A lambda function is a small anonymous function defined using the lambda keyword.

Lambda is used for short, one-time functions!

Sample Code

```
# Example 1
square = lambda x: x * x
print(square(5))   # Output: 25
# Example 2
add = lambda a, b: a + b
print(add(3, 7))   # Output: 10
```

Why Functions?

- ✓ Makes code modular
- ✓ Improves reusability
- ✓ Easy to debug and maintain
- ✓ Enhances readability

Good Functions = Clean Code + Happy Programmer ☺

50 Python Interview Questions & Answers

OOP Makes Code Organized, Scalable & Powerful! ♡

SLIDE 6 : OBJECT ORIENTED PROGRAMMING

26 What is OOP?

→ Object Oriented Programming (OOP) is a programming paradigm that organizes code using objects and classes.

Key Principles of OOP

- ✓ Encapsulation
- ✓ Inheritance
- ✓ Polymorphism
- ✓ Abstraction

27 What is a class?

→ A class is a blueprint or template for creating objects.

Sample Code

```
class Car:                  # Class definition
    def __init__(self, brand, model):
        self.brand = brand      # attribute
        self.model = model      # attribute
    def info(self):
        return f"{self.brand} {self.model}"
```

28 What is an object?

→ An object is an instance of a class.

Sample Code

```
c1 = Car("Toyota", "Innova")  # object 1
c2 = Car("Honda", "City")     # object 2
print(c1.info())    # Toyota Innova
print(c2.info())    # Honda City
```

29 What is inheritance?

→ Inheritance allows a new class (child) to inherit attributes and methods from an existing class (parent).

Sample Code

```
class Animal:               # Parent class
    def speak(self):
        return "Some sound"
class Dog(Animal):          # Child class
    def speak(self):        # Method Overriding
        return "Bark"

d = Dog()
print(d.speak())    # Bark
```

30 What is polymorphism?

→ Polymorphism allows different classes to respond to the same method call in different ways.

Sample Code

```
class Cat:
    def sound(self):
        return "Meow"
class Dog:
    def sound(self):
        return "Bark"
for animal in [Cat(), Dog()]:
    print(animal.sound())    # Meow  Bark
```

Why OOP?

- ☆ Makes code modular and easy to maintain.
- ☆ Reusability of code through inheritance.
- ☆ Improves scalability for large projects.
- ☆ Enhances readability and organization.
- ☆ Reduces complexity.

Think in Objects, Code with Clarity, Build with Confidence! ♡

50 Python Interview Questions & Answers

Go Beyond Basics, Think Advanced!

SLIDE 7 : ADVANCED PYTHON

31 What is a decorator?

→ A decorator is a function that takes another function and extends or modifies its behavior without changing it.

Sample Code

```
def my_decorator(func):
    def wrapper():
        print("Before function call")
        func()
        print("After function call")
    return wrapper
@my_decorator
def say_hello():
    print("Hello!")
say_hello()
```

Output

```
Before
function call
Hello!
After
function call
```

32 What is a generator?

→ A generator is a function that returns an iterator using the yield keyword.

Sample Code

```
def count():
    yield 1
    yield 2
    yield 3
for num in count():
    print(num)
```

Output

```
1
2
3
```

33 What is list comprehension?

→ A concise way to create lists in a single line of code.

Sample Code

```
squares = [x*x for x in range(6)]
print(squares)  # [0, 1, 4, 9, 16, 25]
```

Output

```
[0, 1, 4,
9, 16, 25]
```

34 What is the difference between copy and deepcopy?

→ copy creates a shallow copy, while deepcopy creates a completely independent copy.

copy (shallow)	deepcopy (deep)
• Copies the outer object only. • Nested objects are referenced. • Changes in nested objects affect both.	• Copies the outer object and all nested objects. • Fully independent copy. • Changes in one do not affect the other.

35 What is *args and **kwargs?

→ *args allows a function to accept any number of positional arguments.

→ **kwargs allows a function to accept any number of keyword arguments.

Sample Code

```
def demo(*args, **kwargs):
    print("Positional args:", args)
    print("Keyword args:", kwargs)
demo(1, 2, 3, name="John", age=25, city="Pune")
```

Output

```
Positional args: (1, 2, 3)
Keyword args: {'name': 'John',
'age': 25, 'city': 'Pune'}
```

Why These Concepts Matter?

- ✓ Decorators → For logging, authentication, timing, etc.
- ✓ Generators → For memory-efficient iteration.
- ✓ List Comprehension → For cleaner and faster code.
- ✓ Copy vs Deepcopy → To avoid unexpected side effects.
- ✓ *args & **kwargs → For flexible and reusable functions.

Master Advanced Concepts → Write Smarter, Cleaner & More Efficient Code! ☺

50 Python Interview Questions & Answers

Handle Errors Gracefully, Build Robust Code! ☺

SLIDE 8 : EXCEPTION HANDLING

36 What is an exception?

→ An exception is an error that occurs during the execution of a program and disrupts the normal flow.

Common Built-in Exceptions

- ZeroDivisionError – Division by zero
- ValueError – Invalid value
- TypeError – Wrong data type
- FileNotFoundError – File not found
- IndexError – Index out of range
- KeyError – Key not found in dict

37 What is try-except block?

→ try-except block is used to catch and handle exceptions so that the program doesn't crash.

Sample Code

```
try:
    x = 10 / 0
    print(x)
except ZeroDivisionError:
    print("Cannot divide by zero!")
print("Program continues...")
```

Output

```
Cannot divide by zero!
Program continues...
```

38 What is the purpose of finally block?

→ finally block contains code that executes no matter what – whether an exception occurred or not.

Sample Code

```
try:
    f = open("data.txt")
    content = f.read()
except FileNotFoundError:
    print("File not found!")
finally:
    print("This will always execute.")
```

Output

```
File not found!
This will always execute.
```

39 What is the use of raise keyword?

→ raise keyword is used to manually throw an exception.

Sample Code

```
def check_age(age):
    if age < 18:
        raise ValueError("Age must be 18 or above!")
    else:
        print("Access granted")

check_age(16)      # Raises ValueError
```

40 What is the difference between Syntax Errors and Exceptions?

Syntax Errors	Exceptions
• Occur when the code violates Python grammar rules. • Detected by the interpreter before execution. • Prevent the program from running. • Example: missing colon, unmatched parentheses, indentation error.	• Occur during the execution of a valid program. • Caused by runtime conditions. • Can be handled using try-except. • Example: division by zero, file not found, invalid input.
Example: `if True` ← Missing colon	Example: `x = 10 / 0` ← Runtime exception

Quick Tips

- ✓ Use try-except to handle errors gracefully.
- ✓ Always add finally to clean up resources.
- ✓ Never ignore exceptions – handle them!

Exceptions are not failures, they are opportunities to make your code stronger! ☺

50 Python Interview Questions & Answers

Think Smart,
Code Clean,
Stay Consistent,
Get Hired! ☺

SLIDE 9 : MODULES & PACKAGES

41 What is a module?

→ A module is a file containing Python definitions and statements. It can include functions, classes, and variables.

Sample Code

```
# mymodule.py
def add(a, b):
    return a + b
PI = 3.14159
```

We can import and use this module in another program.

42 How do you import a module?

→ We use import keyword to import a module.

Sample Code

```
import mymodule
result = mymodule.add(5, 7)
print(result)      # Output: 12
```

You can also import specific items from a module using from...import statement.

43 What is a package?

→ A package is a collection of modules in a directory, with a special __init__.py file.

Example Structure

```
mypackage/
├── __init__.py
├── module1.py
├── module2.py
└── subpkg/
    ├── __init__.py
    └── module3.py
```

The __init__.py file tells Python that the directory is a package.

44 How do you import from a package?

→ We can import modules or specific items from a package.

Sample Code

```
# Import entire module
from mypackage import module1
module1.some_function()

# Import specific item
from mypackage.module2 import some_variable
print(some_variable)

# Import with alias
import mypackage.module1 as m1
m1.some_function()
```

45 What is the difference between module and package?

Module	Package
Single Python file (.py).	Collection of modules in a directory.
Contains functions, classes, variables.	Contains multiple modules and a __init__.py file.
Imported using import module_name	Imported using import package_name or from package import module.
Example: math.py	Example: mypackage/

Quick Tips

- ✓ Use modules to organize code.
- ✓ Use packages for larger projects.
- ✓ Always use meaningful names.
- ✓ Keep code reusable and maintainable.
- ✓ Follow PEP 8 naming conventions.

♡ Well Organized Code Today, Better Opportunities Tomorrow! ☺

50 Python Interview Questions & Answers

Small Habits, Clean Code, Big Impact! ☺

SLIDE 10 : BUILT-IN FUNCTIONS & BEST PRACTICES

46 What are some commonly used built-in functions in Python?

→ Python has many useful built-in functions.

Examples

- `len()` - Returns length of an object
- `type()` - Returns type of an object
- `range()` - Returns a sequence of numbers
- `max()` - Returns largest item
- `min()` - Returns smallest item
- `sum()` - Returns sum of items
- `sorted()` - Returns sorted list
- `enumerate()` - Returns index and value

47 Explain enumerate() with example.

→ enumerate() adds a counter to an iterable and returns it (index, value) pair.

Sample Code

```
fruits = ["apple", "banana", "cherry"]
for index, fruit in enumerate(fruits):
    print(index, fruit)
```

Output

```
0 apple
1 banana
2 cherry
```

48 What is the difference between list() and tuple()?

list()	tuple()
• Mutable (can be changed)	• Immutable (cannot be changed)
• Uses square brackets [ ]	• Uses parentheses ( )
• Slower than tuples	• Faster than lists
• Example: `a = [1, 2, 3]`	• Example: `b = (1, 2, 3)`

49 How do you use map(), filter(), reduce()?

→ These are higher-order functions from the functools module.

Examples

```
map() - Applies a function to all items
nums = [1, 2, 3]
squares = list(map(lambda x: x*x, nums))
print(squares)      # Output: [1, 4, 9]

filter() - Filters items based on condition
nums = [1, 2, 3, 4, 5]
evens = list(filter(lambda x: x%2==0, nums))
print(evens)      # Output: [2, 4]

reduce() - Applies function cumulatively
from functools import reduce
nums = [1, 2, 3, 4]
product = reduce(lambda x, y: x*y, nums)
print(product)      # Output: 24
```

Note: Don't forget to import reduce from functools. ☺

50 What are some Python best practices?

→ Following best practices makes your code clean, efficient and professional.

- ✓ Follow PEP 8 style guide.
- ✓ Use meaningful variable and function names.
- ✓ Write modular and reusable code.
- ✓ Add comments and docstrings.
- ✓ Handle exceptions properly.
- ✓ Avoid global variables.
- ✓ Keep functions small and focused.
- ✓ Test your code regularly.
- ✓ Use list comprehensions & built-in functions when possible.

GOOD CODE ✓ ✓

Key Takeaways

- ✓ Functions help in organizing and reusing code.
- ✓ OOP makes code scalable and easier to maintain.
- ✓ Modules & Packages keep large projects well-structured.
- ✓ Built-in functions & best practices improve code quality.

Practice Consistently, Code Confidently, Crack Your Python Interview! ☺

Keep Learning, Keep Coding, Keep Growing! ♡