Python 50 code snippets

✅ 1–9: Basic Print and Variables

#1

```
print("Hello" + "World")
```

Output:
HelloWorld

#2

```
a = 5
b = "5"
print(str(a) + b)
```

Output:
55

#3

```
print(3 * "7")
```

Output:
777

#4

```
x = "Py"
y = "thon"
print(x + y)
```

Output:
Python

#5

```
x = 10
x = x + 5
print(x)
```

Output:
15

#6

```
x = 4
print(x * x)
```

Output:
16

#7

```
x = True
y = False
print(x + y)
```

Output:
1

#8

```
print("Hello", end=" ")
print("World")
```

Output:
Hello World

#9

```
print("Line1\nLine2")
```

Output:
Line1
Line2

✅ 10–17: Control Flow & Loops

#10

```
for i in range(1, 4):
    print(i * "*")
```

Output
*
**
***

#11. Program

```
x = 10
if x > 5:
    print("A")
else:
    print("B")
```

Output

```
A
```

#12. Program

```
x = 1
while x < 4:
    print(x)
    x += 1
```

Output

```
1
2
3
```

#13. Program

```
for i in range(3):
    print(i + 1)
```

Output

```
1
2
3
```

#14. Program

```
i = 0
while i < 3:
    i += 1
    print(i)
```

Output

```
1
2
3
```

#15. Program

```
for char in "abc":
    print(char * 2)
```

Output
aa
bb
cc

#16. Program

```
for i in range(5, 8):
    print(i)
```

Output
5
6
7

#17. Program

```
x = 3
y = 3
if x == y:
    print("Equal")
```

Output
Equal

🟩 18–27: Lists and Strings

#18. Program

```
lst = [1, 2, 3]
print(lst[::-1])
```

Output
[3, 2, 1]

#19. Program

```
x = [1, 2]
y = x
x.append(3)
print(y)
```

Output
[1, 2, 3]

#20. Program

```
print("apple"[1:4])
```

Output
ppl

#21. Program

```
print("Python".upper())
```

Output
PYTHON

#22. Program

```
print("Data".lower().capitalize())
```

Output
Data

#23. Program

```
nums = [10, 20, 30]
print(sum(nums))
```

Output
60

#24. Program

```
chars = list("code")
print(chars)
```

Output
['c', 'o', 'd', 'e']

#25. Program

```
print(len([i for i in range(10) if i % 2 == 0]))
```

Output
5

#26. Program

```
print(" ".join(["a", "b", "c"]))
```

Output
a b c

#27. Program

```
print([x**2 for x in range(3)])
```

Output
[0, 1, 4]

💡 28–33: Functions & Scope

#28. Program

```
def greet():
    return "Hello"
print(greet())
```

Output
Hello

#29. Program

```
def add(x, y=2):
    return x + y
print(add(3))
```

Output
5

#30. Program

```
x = 10
def show():
    x = 5
    print(x)
show()
print(x)
```

Output
5
10

#31. Program

```
def outer():
    def inner():
        return "Inner"
    return inner()
print(outer())
```

Output
Inner

#32. Program

```
def mult(a, b):
    return a * b
print(mult("Hi", 3))
```

Output

```
HiHiHi
```

#33. Program

```
print(type(lambda x: x))
```

Output

```
<class 'function'>
```

🧠 34–43: Tuples, Sets, Dictionaries

#34. Program

```
t = (1, 2, 3)
print(t[1])
```

Output

```
2
```

#35. Program

```
s = {1, 2, 3, 2}
print(len(s))
```

Output

```
3
```

#36. Program

```
d = {"a": 1, "b": 2}
print(d["b"])
```

Output

```
2
```

#37. Program

```
d["c"] = 3
print(d)
```

Output

```
{'a': 1, 'b': 2, 'c': 3}
```

#38. Program

```
print(set("banana"))
```

Output (order may vary)

```
{'b', 'a', 'n'}
```

#39. Program

```
x = {(1, 2): "value"}
print(x[(1, 2)])
```

Output

```
value
```

#40. Program

```
print("key" in {"key": 1})
```

Output

```
True
```

#41. Program

```
sample = {k: k*k for k in range(3)}
print(sample)
```

Output

```
{0: 0, 1: 1, 2: 4}
```

#42. Program

```
x = {1, 2}
y = {2, 3}
print(x & y)
```

Output

```
{2}
```

#43. Program

```
z = {"a": 1, "b": 2}
z.pop("a")
print(z)
```

Output

```
{'b': 2}
```

Python 100 Interview Questions Theory

🧠 Part 1: Python Basics (Q1–Q35)

1. What is Python and its key features?

Python is a high-level, interpreted language known for readability. It supports OOP, large libraries, cross-platform compatibility, and rapid development.

2. Difference between Python 2 and Python 3?

Python 3 uses Unicode by default, improved syntax, and better library support. Python 2 is outdated and no longer maintained.

3. Is Python interpreted or compiled?

Python is interpreted but first converted into bytecode, then executed by the Python Virtual Machine (PVM).

4. Data types in Python?

Common types: int, float, str, bool, list, tuple, dict, set, bytes. Python also supports custom types via classes.

5. Difference between is and ==?

== checks value equality, is checks memory identity. Two variables may be equal but not identical.

6. What are variables?

Variables store data values. They are created dynamically and can change types during execution.

7. Purpose of None?

None represents the absence of a value. Often used as a default argument or to indicate null results.

8. What is dynamic typing?

Variable types are decided at runtime. The same variable can hold different types at different times.

9. Difference between mutable and immutable types?

Mutable types can be changed in place (list, dict). Immutable types cannot be modified (int, str, tuple).

10. Python immutable types?

int, float, str, tuple, frozenset, bool. Changes create new objects.

11. What is a keyword?

Reserved words that have special meaning in Python (like if, for, class). Cannot be used as variable names.

12. What is PEP8?

Python's style guideline for writing clean, readable code. It covers naming, indentation, spacing, etc.

13. Use of id() function?

Returns the memory address (identity) of an object.

14. How does Python manage memory?

Through private heap space, automatic garbage collection, and reference counting.

15. Rules for naming variables?

Start with letter or underscore, can contain letters, digits, underscores. No spaces or keywords.

16. Difference between script and module?

A script is a standalone program. A module is an importable file containing reusable code.

17. What are comments and docstrings?

Comments (#) explain code. Docstrings (""" """) describe functions, classes, modules.

18. What are literals?

Fixed values in code like numbers, strings, booleans, lists, etc.

19. Magic constant __name__ usage?

It is "__main__" when a file runs directly. Helps in creating reusable modules.

20. Purpose of indentation in Python?

Defines code blocks. Python uses indentation instead of braces.

21. What is a namespace?

A mapping of names to objects. Examples: local, global, built-in namespaces.

22. What is type() function?

Returns the type/class of a given object.

23. What is a dynamic language?

A language where types and bindings are determined at runtime. Python allows flexible variable handling.

24. How does Python handle variable swapping?

Using tuple unpacking: a, b = b, a without needing a temporary variable.

25. What does pass do?

A null statement used as a placeholder for future code.

26. Internal steps in Python execution?

Source → Lexing → Parsing → AST → Bytecode → Executed by PVM.

27. How .pyc files are used?

They store bytecode for faster execution on subsequent runs.

28. Role of Python Virtual Machine (PVM)?

Executes bytecode instructions line by line.

29. What is the Abstract Syntax Tree (AST)?

A tree representation of code structure generated after parsing.

30. Internal memory layout of Python objects?

Objects contain metadata: reference count, type pointer, value/data fields.

31. What is the GIL?

The Global Interpreter Lock allows only one thread to execute Python bytecode at a time in CPython.

32. Garbage collection in Python?

Python uses reference counting and cyclic garbage collection to free unused objects.

33. Reference counting & cyclic GC?

Reference counting deletes objects when count hits zero. Cyclic GC removes reference cycles.

34. Where are functions/variables stored (stack vs heap)?

Objects are stored in heap memory. Function call details are stored in the call stack.

35. How are primitive data types handled in memory?

Primitive types like int and float are immutable and stored as fixed objects in memory.

🔁 Part 2: Control Flow (Q36–Q50)

36. What are control flow statements in Python?

Control flow statements decide the order in which code executes. They include if, elif, else, loops (for, while), and loop controls (break, continue, pass).

37. Difference between if, elif, and else?

if checks the first condition.
elif checks additional conditions when previous ones fail.
else runs when none of the earlier conditions are true.

38. Difference between while and for loops?

while loops run until a condition becomes false.
for loops iterate over a sequence or range.
while is condition-based, for is iterable-based.

39. Use of break and continue?

break exits the loop completely.
continue skips the current iteration and jumps to the next cycle.

40. Use of the else block with loops?

The else block runs only if the loop completes without a break.
Commonly used in search operations.

41. How to loop over a range of numbers?

Using range() with a for loop:
for i in range(1, 10): print(i).

42. What is the use of range()?

It generates a sequence of numbers.
Used commonly for looping, indexing, and generating series.

43. Difference between range() and enumerate()?

range() gives numbers only.
enumerate() gives index + value when looping through iterables.

44. What is zip() used for in loops?

It pairs elements from multiple iterables.
Example: for a, b in zip(list1, list2):.

45. What is a nested loop?

A loop inside another loop.
Used for matrices, patterns, and multiple-level iterables.

46. What are infinite loops?

Loops that never end when the exit condition is never met.
Example: while True: unless a break is used.

47. How to check if a number is even or odd?

Use modulus:
if num % 2 == 0: even else: odd.

48. How does Python evaluate truthy and falsy values?

Falsy: 0, "", [], {}, None, False.
Everything else is truthy.

49. What is a conditional expression (ternary operator)?

A one-line if-else:
x = a if condition else b.

50. What is short-circuiting in Python?

Logical and and or stop evaluating once the result is known.
Useful for safe access: x and x.value.

🔣 Part 3: Operators (Q51–Q60)

51. What are the types of operators in Python?

Python has arithmetic, comparison, logical, bitwise, assignment, membership, and identity operators. Each category performs different operations on operands.

52. Difference between == and is?

== compares values.
is compares memory locations (object identity).
Two objects may be equal but not identical.

53. How does Python handle operator precedence?

Operators follow a defined hierarchy (PEMDAS-like).
Expressions are evaluated based on this order unless parentheses override it.

54. What are logical operators in Python?

and, or, not.
Used to combine conditional statements and return Boolean results.

55. What is identity vs equality?

Identity checks if two variables reference the same object (is).
Equality checks if their values are equal (==).

56. What is membership testing (in, not in)?

Used to check if a value exists in sequences like lists, strings, tuples, sets.
Example: 'a' in 'apple'.

57. Difference between += and =+?

+= is an addition assignment operator (adds then assigns).
=+ assigns a positive value (simply x = +y).

58. How does the not operator work?

not reverses a boolean value.
not True → False, not False → True.

59. Difference between bitwise and logical operators?

Bitwise operators (&, |, ^, <<) work on bits.
Logical operators (and, or, not) work on boolean expressions.

60. Can you overload operators in Python?

Yes, using magic methods like __add__, __sub__, __eq__.
Allows custom behavior for operators in user-defined classes.

🧰 Part 4: Functions (Q61–Q80)

61. What is a function in Python?

A function is a reusable block of code that performs a specific task.
It helps reduce repetition and improves code readability.

62. How do you define a function?

Using the def keyword:

```
def my_func():
    pass
```

Functions may have parameters and a return value.

63. What is the use of return?

return sends a value back to the caller.
It also ends the function execution immediately.

64. What are *args and **kwargs?

*args allows passing variable number of positional arguments.
**kwargs allows passing variable number of keyword arguments.

65. What is a default argument?

A parameter with a predefined value.
Example: def add(x, y=5):.

66. What is a keyword-only argument?

An argument that must be passed using its name.
Defined by placing * before parameters.

67. What is a lambda function?

A small anonymous function created using lambda.
Used for short operations like sorting or filtering.

68. Difference between yield and return?

return exits the function and sends a value.
yield makes the function a generator and returns values one at a time.

69. Scope of variables in functions?

Local variables exist inside the function.
Global variables exist outside and accessible everywhere unless shadowed.

70. What are global and nonlocal keywords?

global allows modifying global variables from inside a function.
nonlocal is used inside nested functions to modify outer-scope variables.

71. How do you pass a list to a function?

Simply pass it like any other argument.
Example: func([1, 2, 3]).

72. What is recursion?

A function calling itself.
Used in problems like factorial, Fibonacci, and tree traversals.

73. What is a higher-order function?

A function that takes another function as argument or returns one.
Examples: map, filter, decorators.

74. What are map(), filter(), and reduce()?

- map() applies a function to each element.
- filter() selects elements that meet a condition.
- reduce() (from functools) reduces a sequence to a single value.
-

75. Use of enumerate() in functions?

It gives index + value when looping over iterables.
Useful for tracking positions in loops.

76. What is a closure in Python?

A closure is a function that remembers variables from its outer scope even after the scope is gone.

77. What is a decorator?

A function that modifies another function's behavior without changing its code.
Uses @decorator_name syntax.

78. Can a function return another function?

Yes, Python supports returning functions (first-class functions).
Useful for closures and decorators.

79. How to define an anonymous function?

Using lambda keyword.
Example: lambda x: x * 2.

80. What is function caching?

Storing function results to speed up repeated calls.
Implemented using functools.lru_cache.

🖌️ Part 5: Data Structures (Q81–Q95)

81. Difference between a list and a tuple?

Lists are mutable and can be changed after creation.
Tuples are immutable and fixed once created.
Lists use [], tuples use ().

82. Key features of lists?

Lists are ordered, mutable, and allow duplicates.
They can store mixed data types and support many built-in methods.

83. What are list comprehensions?

A concise way to create lists in one line.
Example: [x*x for x in range(5)].

84. What is a dictionary?

A collection of key–value pairs.
Keys must be unique and immutable; values can be anything.

85. Difference between a list and a set?

Lists are ordered and allow duplicates.
Sets are unordered and automatically remove duplicates.

86. What is a set? What are its use cases?

A set is an unordered collection of unique items.
Used for membership testing, removing duplicates, and performing set operations.

87. How to remove duplicates from a list?

Convert to a set: list(set(mylist)).
Or use a loop to maintain order.

88. What are nested lists?

Lists that contain other lists.
Used to represent matrices or hierarchical data.

89. How do you merge two dictionaries?

Using {**dict1, **dict2} or dict1.update(dict2).
Python 3.9+ supports dict1 | dict2.

90. How to sort a list or dictionary?

Lists: sorted(list) or list.sort().
Dictionaries: sorted(dict.items()) for key/value sorting.

91. What is a defaultdict?

A dictionary that provides a default value for missing keys.
Defined in collections.

92. What is a Counter?

A subclass of dict used to count occurrences of elements.
Example: Counter("banana").

93. What is a namedtuple?

A lightweight object similar to a tuple but with named fields.
Useful for readable code.

94. Difference between pop(), remove(), and del?

pop() removes by index and returns the value.
remove() deletes by value.
del deletes element or entire variable.

95. What is the use of slicing?

Extracts parts of sequences like lists, tuples, or strings.
Syntax: sequence[start:end:step].

🔐 Part 6: Object-Oriented Programming (Q96–Q110)

96. What is object-oriented programming (OOP)?

OOP is a programming style based on objects and classes.
It focuses on code reusability, modularity, and real-world modeling.

97. What are classes and objects in Python?

A class is a blueprint for creating objects.
An object is an instance of a class containing data and methods.

98. What is __init__()?

It's a constructor method that runs automatically when an object is created.
Used to initialize instance variables.

99. What is self in a class?

self refers to the current instance of the class.
It allows access to instance variables and methods.

100. Difference between instance variables and class variables?

Instance variables are unique to each object.
Class variables are shared by all objects of the class.

101. What is inheritance?

A mechanism where one class (child) acquires properties of another class (parent).
Supports code reuse and extension.

102. What is multiple inheritance?

A class can inherit from more than one parent class.
Python fully supports it through the MRO (Method Resolution Order).

103. What is method overriding?

A child class provides a different implementation of a parent method.
Used to customize or extend behavior.

104. What is super() used for?

It allows calling parent class methods inside a child class.
Useful for inheritance and constructor chaining.

105. What are magic methods?

Special methods that start and end with __.
Examples: __add__, __len__, __str__. They let objects behave like built-ins.

106. Difference between __str__ and __repr__?

__str__ is for user-friendly output.
__repr__ is for developers and should return an unambiguous representation.

107. What is encapsulation?

Wrapping data and methods into a single unit.
Python uses naming conventions like _protected and __private.

108. What is polymorphism?

The ability to use the same method name for different behaviors.
Occurs via method overriding or duck typing.

109. What are static methods and class methods?

Static methods don't use self or class state.
Class methods use cls and modify class-level data.

110. What are access modifiers in Python?

Python uses naming conventions:

- Public: variable
- Protected: _variable
- Private: __variable
  They control how attributes should be accessed.

25 OOPs Theory

🟦 1. Basics of OOP

1. What are the four pillars of OOP in Python?

The four pillars of OOP are **Encapsulation**, **Inheritance**, **Polymorphism**, and **Abstraction**.
They help structure code, reduce duplication, and increase readability.

Cross: Real-world (Car class):

- Encapsulation → private speed
- Inheritance → ElectricCar(Car)
- Polymorphism → same method start() behaves differently
- Abstraction → abstract class Vehicle with abstract method drive()

2. Difference between a class and an object?

A class is a blueprint defining attributes and methods.
An object is an instance created from a class.

Cross: Memory is allocated on the **heap** when you write obj = MyClass().

3. How is OOP different from procedural programming?

Procedural focuses on functions and step-by-step logic.
OOP focuses on objects and behavior around data.

Cross:

- Function version: def area(r): return 3.14*r*r
- Class version: Circle().area().

🟧 2. Encapsulation

4. What is encapsulation in Python?

Encapsulation binds data and methods together and restricts direct access.
Helps protect internal data.

Cross:
_var = protected (convention)
__var = private (name mangling).

5. Why use getters and setters?

They control how attributes are accessed and updated.
Useful for validation and read-only fields.

Cross: Example using @property:

@property

def age(self): return self._age

@age.setter

def age(self, v): self._age = v

🟩 3. Inheritance

6. What is inheritance?

A child class acquires features of a parent class.
Helps in code reuse and modular design.

Cross: Private variables (__x) are *not* inherited directly due to name mangling.

7. Types of inheritance in Python?

Single, Multiple, Multilevel, Hierarchical, Hybrid.
Python supports all these.

Cross: Example of multiple inheritance:

class A: pass

class B: pass

class C(A, B): pass

8. What is the diamond problem?

Occurs when multiple inheritance causes ambiguity in method calling order.

Cross: Python solves it using **MRO (C3 linearization)**.

9. Why do we use inheritance?

To reuse code and extend functionality without rewriting.

Cross: Composition is better when you want "has-a" relation like Car has Engine.

🟦 4. Polymorphism

10. What is polymorphism?

Same function or method name behaves differently depending on the object.

Cross:
Different classes implementing speak() but giving different outputs.

11. Difference between method overloading and overriding?

Overloading = same method name with different parameters (not directly supported).
Overriding = child class redefines parent method.

Cross: Mimicking overloading using default args:

def add(a, b=None): ...

12. What is duck typing?

If an object behaves like expected, it can be used regardless of its type.
"**If it walks like a duck, it's a duck**".

Cross Example: Any object with a .draw() method works in a drawing function.

13. What is operator overloading?

Defining how operators work for custom objects.

Cross: Example for + using __add__:

```
def __add__(self, other):
    return self.value + other.value
```

🟫 5. Abstraction

14. What is abstraction?

Showing essential features while hiding internal details.
Helps reduce complexity.

Cross: Yes, abstraction can be achieved without abc using methods that raise NotImplementedError.

15. Difference between abstract class and interface?

Abstract class can have normal and abstract methods.
Python doesn't have interfaces but we mimic them using abstract classes with only abstract methods.

16. Why use abstraction in large projects?

It enforces structure and ensures all child classes implement required methods.

Cross: Abstract method process_payment() forces all payment classes to implement it.

🟪 6. Constructors & Destructors

17. What is a constructor in Python?

A constructor initializes object attributes during creation.

Cross: Python uses __init__() as a constructor.

18. Difference between constructor and normal method?

Constructor runs automatically when object is created.
Normal methods are called manually.

Cross: Constructors **cannot return** values.

19. What is a destructor in Python?

Destructor cleans up resources before object deletion.
Executed automatically.

Cross: __del__() is called when object is garbage collected.

🟥 7. Access Modifiers

20. What are access modifiers in Python?

Python uses naming conventions:
Public → var
Protected → _var
Private → __var

Cross: Private is not truly private; name-mangling makes it harder but still accessible.

21. Why use public, protected, private attributes?

To protect data and control how variables can be accessed or modified.

Cross: If everything is public, data may be accidentally changed, reducing safety.

🟧 8. Advanced OOP Concepts

22. What is composition?

One class contains an object of another class (has-a relationship).

Cross Example:
Car contains an Engine object.

23. What is aggregation?

A weaker form of composition where the contained object can exist independently.

Difference:
Composition = strong ownership
Aggregation = loose relationship.

24. What is MRO (Method Resolution Order)?

Defines the order in which Python searches for methods in multiple inheritance.
Uses C3 linearization.

Cross: Check MRO using ClassName.mro() or ClassName.__mro__.

25. Difference between static, class, and instance methods?

- **Instance method:** uses self, works on object data.
- **Class method:** uses cls, works on class data.
- **Static method:** no self/cls, behaves like normal function inside class.

Cross Example:

@staticmethod

@classmethod

def instance_method(self)