

PYTHON

Placement Questions & Answers

Concepts • Theory • Output Questions • Programs • Examples

For: B.Tech / AKTU Examination • Viva • Placement Preparation

NoidaTut
Python Tutorial & Academic Learning Resource

This revised edition uses reliable vector-rendered code boxes so program text remains clearly visible in PDF viewers. Content is arranged to avoid hidden or blank tutorial sections.

Index / Complete Question List

All supplied questions are listed first. Detailed answers follow in the same sequence.

No.	Question
1	Explain the concept of OOPS.
2	Point out the difference between C and Python.
3	Explain the features of OOPS.
4	Differentiate between POP and OOPS.
5	Highlight the features of Python which make it special and different from other programming languages.
6	What will happen if we end our Python statement with a semicolon (;)?
7	Define Python variables. Give an example of assigning int, float and string values.
8	What will be the output of the given Python code using rows, range() and string multiplication?
9	What will be the output of <code>int = 7; print(int)</code> ? Explain if there is an error and correct it if necessary.
10	Write short notes on operator precedence and associativity, and evaluate the given logical expression step by step.
11	Explain Python's built-in data structures with examples.
12	Differentiate break, continue and pass with examples, and write a program to print prime numbers from 1 to 50.
13	Describe Python blocks and conditional statements. Write a program to grade students based on marks using elif.
14	Classify the different types of operators in Python and explain each with an example.
15	Elaborate type conversion and indentation with examples.
16	Discuss for loop with range, strings and lists. Write a program to sum the elements of a list having 10 elements.
17	Implement a program to swap two numbers without a temporary variable, and write a program to find the first even number in the list created.
18	Differentiate keywords and identifiers in Python, and state the rules for naming a variable.
19	Write programs to check whether a year is a leap year and to calculate the factorial of a number.

1. Explain the concept of OOPS

Object-Oriented Programming (OOP) organizes software around objects. An object combines state (attributes/data) and behavior (methods). A class is the blueprint used to create objects.

Basic OOP model



Class: blueprint. Object: instance of a class. Encapsulation: combines data and methods. Abstraction: hides unnecessary implementation details. Inheritance: supports reuse through class relationships. Polymorphism: allows a common interface to behave differently for different objects.

Example:

```
class Student:
    def __init__(self, name, marks):
        self.name = name
        self.marks = marks

    def display(self):
        print(self.name, self.marks)

s1 = Student("Riya", 85)
s1.display()
```

Here, Student is the class, s1 is an object, name and marks are attributes, and display() is a method.

2. Point out the difference between C and Python

Feature	C	Python
Programming style	Primarily procedural	Multi-paradigm: procedural, OOP, functional
Typing	Statically typed	Dynamically typed
Execution	Commonly compiled to machine code	Executed through the Python runtime
Syntax	Braces/semicolons commonly used	Indentation defines blocks; semicolons usually unnecessary
Memory	Manual allocation/deallocation is common	Automatic memory management
Pointers	Explicit pointers and pointer arithmetic	No C-style pointer arithmetic
Typical uses	Systems, embedded, low-level software	Web, automation, data science, AI/ML, scripting

Example syntax:

```
/* C */  
int x = 10;  
printf("%d", x);
```

```
/* Python */  
x = 10  
print(x)
```

3. Explain the features of OOPS

Four core OOP ideas

Data / Structure

- Encapsulation
- Abstraction

Reuse / Behavior

- Inheritance
- Polymorphism

Encapsulation: organizes data and methods together. Abstraction: exposes essential operations while hiding unnecessary details.

Inheritance: a derived class can reuse and extend a base class. Polymorphism: the same interface can have different implementations.

```
class Animal:
    def sound(self):
        print("Some sound")

class Dog(Animal):
    def sound(self):
        print("Bark")

Dog().sound()
```

4. Differentiate between POP and OOPS

Aspect	POP	OOP
Focus	Functions / procedures	Objects and classes
Design	Often top-down	Often based on interacting objects
Data organization	Data passed among functions	Data and methods grouped in objects
Reuse	Functions	Classes, inheritance, composition and objects
Examples	C	Python, Java, C++

The main difference is how the program is decomposed and how data and behavior are organized.

5. Features of Python that make it special

- Readable syntax: programs are generally concise and easy to read.
- High-level: abstracts many low-level details.
- Dynamically typed: explicit variable type declarations are not required.
- Multi-paradigm: supports procedural, OOP and functional styles.
- Large ecosystem: libraries support web, automation, science and AI/ML.
- Cross-platform: available on major operating systems.
- Interactive: convenient interpreter/REPL workflow.
- Automatic memory management: objects are managed by the runtime.

Example:

```
numbers = [1, 2, 3, 4, 5]
squares = [n * n for n in numbers]
print(squares)
```

6. What happens if a Python statement ends with a semicolon (;)?

A semicolon is valid in Python. It is not required at the end of a normal statement, but it can terminate a statement or separate multiple simple statements on one line.

```
x = 10;  
print(x);  
  
a = 1; b = 2  
print(a + b)
```

Output:

```
10  
3
```

Semicolons are generally unnecessary in normal Python code because one simple statement per line is clearer and more idiomatic.

7. Define Python variables; examples of int, float and string

A variable is a name bound to an object/value. Python does not require a separate type declaration; assignment creates the binding.

```
age = 20
price = 99.50
name = "Aadarsh"

print(age)
print(price)
print(name)

print(type(age))
print(type(price))
print(type(name))
```

Variable	Value	Type
age	20	int
price	99.50	float
name	"Aadarsh"	str

8. Output of the range() and string multiplication program

```
rows = 5

for i in range(rows, 0, -1):
    print("*" * i)
```

range(5, 0, -1) produces 5, 4, 3, 2 and 1. String multiplication repeats the asterisk that many times.

Output:

```
*****
****
***
**
*
```

9. `int = 7; print(int)`: output, issue and correction

```
int = 7  
print(int)
```

Output:

```
7
```

There is no syntax error. However, `int` is a built-in Python name. Assigning `int = 7` shadows that name in the current scope.

Better version:

```
number = 7  
print(number)  
  
value = int("25")  
print(value)
```

Avoid using built-in names such as `int`, `str`, `list`, `dict` and `sum` as ordinary variable names.

10. Operator precedence and associativity; step-by-step evaluation

Precedence determines which operators are evaluated first. Associativity determines the evaluation direction for operators at the same precedence level.

Level	Operators
Parentheses	()
Exponentiation	**
Multiplicative	*, /, //, %
Additive	+, -
Comparison	<, <=, >, >=, ==, !=
Logical	not → and → or

Given expression:

```
a = 10
b = 3
c = 2
result = (a + b * c) ** 2 > 100 and a // b == 3 or not (c % 2 == 0)
print(result)
```

Step 1: $b \times c = 6$. Step 2: $a + 6 = 16$. Step 3: $16^2 = 256$.

Step 4: $256 > 100 \rightarrow \text{True}$. Step 5: $10 // 3 = 3$, so $3 == 3 \rightarrow \text{True}$.

Step 6: $2 \% 2 = 0$, therefore $\text{not } (c \% 2 == 0) \rightarrow \text{False}$.

Step 7: $\text{True and True or False} \rightarrow \text{True}$.

Output:

```
True
```

11. Explain Python's built-in data structures

The four core built-in collection types are list, tuple, set and dictionary.

Structure	Mutable?	Main idea	Example
List	Yes	Ordered, indexed; duplicates allowed	[10, 20, 10]
Tuple	No	Immutable sequence	(10, 20, 10)
Set	Yes	Unique elements; no positional indexing	{10, 20}
Dictionary	Yes	Key-value mapping	{'name': 'Riya'}

Example:

```
numbers = [10, 20, 30]
point = (10, 20)
unique = {10, 20, 30}
student = {"name": "Riya", "marks": 85}

print(numbers[0])
print(point[1])
print(student["name"])
```

12. break, continue, pass and prime numbers from 1 to 50

Statement	Meaning
break	Immediately exits the nearest loop.
continue	Skips the rest of the current iteration.
pass	Does nothing; serves as a placeholder.

Examples:

```
for i in range(5):
    if i == 3:
        break
    print(i)

for i in range(5):
    if i == 2:
        continue
    print(i)

if True:
    pass
```

Prime-number program:

```
for num in range(2, 51):
    is_prime = True

    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

    if is_prime:
        print(num, end=" ")
```

Output:

```
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47
```

13. Python blocks and conditional statements; grading program

A Python block is a group of statements belonging to the same control structure, function, class or other compound statement. Python uses indentation to define these blocks.

Typical conditional structure



```
marks = float(input("Enter marks: "))

if marks >= 90:
    grade = "A+"
elif marks >= 80:
    grade = "A"
elif marks >= 70:
    grade = "B"
elif marks >= 60:
    grade = "C"
elif marks >= 50:
    grade = "D"
else:
    grade = "F"

print("Grade:", grade)
```

14. Classify the different types of operators in Python

Category	Operators	Example
Arithmetic	+ - * / // % **	10 + 3
Comparison	== != > < >= <=	10 > 3
Assignment	= += -= *= /= //=	x += 2
Logical	and, or, not	a > 0 and b > 0
Bitwise	& ^ ~ << >>	5 & 3
Membership	in, not in	2 in [1,2,3]
Identity	is, is not	a is b

Example:

```
a = 10
b = 3

print(a + b)      # arithmetic
print(a > b)      # comparison
a += 2            # assignment
print(a > 0 and b > 0) # logical
print(2 in [1, 2, 3]) # membership
```

15. Type conversion and indentation

Type conversion changes a value from one type to another. Common functions include `int()`, `float()`, `str()` and `bool()`.

```
x = "25"
n = int(x)

price = 99
text = str(price)

print(n + 5)
print(text)
```

Indentation is the whitespace used to define Python blocks.

```
age = 20

if age >= 18:
    print("Adult")
    print("Eligible")
else:
    print("Minor")
```

Incorrect indentation can cause an `IndentationError` or place a statement in the wrong block. Four spaces per level is the common convention.

16. for loop with range, strings and lists; sum of 10 elements

A for loop iterates over an iterable such as a range, string or list.

Using range:

```
for i in range(1, 6):  
    print(i)
```

Using a string:

```
for ch in "Python":  
    print(ch)
```

Using a list:

```
colors = ["red", "green", "blue"]  
  
for color in colors:  
    print(color)
```

Sum of 10 elements:

```
numbers = [10, 20, 30, 40, 50,  
           60, 70, 80, 90, 100]  
  
total = 0  
for value in numbers:  
    total += value  
  
print("Sum =", total)
```

Output: Sum = 550

17. Swap two numbers and find the first even number

1. Swap without a temporary variable:

```
a = 10
b = 20

a, b = b, a

print("a =", a)
print("b =", b)
```

Output:

```
a = 20
b = 10
```

2. Find the first even number in the list:

```
numbers = [15, 9, 7, 13, 22, 18, 30]

for value in numbers:
    if value % 2 == 0:
        print("First even number:", value)
        break
```

Output: First even number: 22

18. Keywords, identifiers and variable naming rules

A keyword is a reserved word with a predefined meaning in Python. An identifier is a programmer-defined name for a variable, function, class or other object.

Feature	Keyword	Identifier
Meaning	Reserved by Python	Programmer-defined name
Choice	Cannot be freely chosen	Chosen within naming rules
Examples	if, for, class, return	total, student_name, calculate_sum

Rules for naming variables:

- Start with a letter or underscore.
- After the first character, letters, digits and underscores are allowed.
- Names are case-sensitive.
- Do not use Python keywords.
- Prefer meaningful names.

Valid examples:

```
student_name = "Riya"  
_marks = 85  
subject2 = "Python"
```

Invalid examples:

```
2marks = 85  
student-name = "Riya"  
class = "B.Tech" # class is a keyword
```

19. Leap year and factorial programs

(i) Check whether a year is a leap year

A year is a leap year if it is divisible by 400, or if it is divisible by 4 but not by 100.

```
year = int(input("Enter year: "))

if year % 400 == 0:
    print("Leap year")
elif year % 100 == 0:
    print("Not a leap year")
elif year % 4 == 0:
    print("Leap year")
else:
    print("Not a leap year")
```

Examples: 2024 is a leap year; 1900 is not a leap year; 2000 is a leap year.

(ii) Factorial of a number

For a non-negative integer n , $n! = 1 \times 2 \times \dots \times n$ and $0! = 1$.

```
n = int(input("Enter a number: "))

factorial = 1
for i in range(1, n + 1):
    factorial *= i

print("Factorial =", factorial)
```

For $n = 5$: $1 \times 2 \times 3 \times 4 \times 5 = 120$.

20. Quick Revision Sheet

Topic	Remember
OOP	Objects combine state and behavior; classes are blueprints.
C vs Python	C is commonly statically typed and compiled; Python is dynamically typed and runs through its runtime.
POP vs OOP	POP focuses on procedures; OOP focuses on objects/classes.
Semicolon	Allowed, but normally unnecessary.
Variables	Names are bound to objects; no separate type declaration is required.
int as variable	Shadows the built-in int name.
Precedence	Controls evaluation order; associativity resolves same-level operators.
Data structures	List, tuple, set and dictionary are key built-in collections.
break / continue / pass	Exit / skip iteration / placeholder.
Indentation	Defines blocks.
for loop	Iterates over an iterable.
Keywords / identifiers	Reserved words / programmer-defined names.
Leap year	Divisible by 400, or by 4 but not by 100.
Factorial	Product from 1 to n; $0! = 1$.

End of Python Placement Questions & Answers