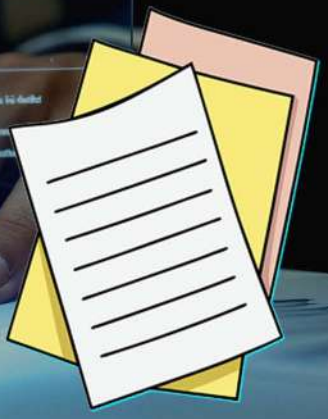
GeeksforGeeks

GATE फर्स्ट

CSE

PYTHON PROGRAMMING

SHORT NOTES



ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

PYTHON PROGRAMMING

GATE फर्रे

Python is a high-level, interpreted, object-oriented programming language used for easy coding, automation, data science, web development, AI, and more.

Input function: input( ) → takes input from user as a string.

```
name = input("Enter your name: ")
```

Output function: print( ) → displays output.

```
print("Hello", name)
```

List of operators in Python

1. Arithmetic Operators

```
a, b = 10, 3
print(a + b)   # 13 (Addition)
print(a - b)   # 7  (Subtraction)
print(a * b)   # 30 (Multiplication)
print(a / b)   # 3.33 (Division)
print(a % b)   # 1  (Modulus)
print(a ** b)  # 1000 (Exponent)
print(a // b)  # 3 (Floor Division)
```

2. Comparison Operators

```
python

print(a > b)    # True
print(a == b)   # False
```

3. Logical Operators

```
x, y = True, False
print(x and y)   # False
print(x or y)    # True
print(not x)     # False
```

4. Assignment Operators

```
c = 5
c += 2    # c = c + 2 → 7
```

5. Bitwise Operators

```
print(a & b)   # 2 (AND)
print(a | b)   # 11 (OR)
```

6. Membership Operators

```
print(3 in [1,2,3])    # True
print(4 not in [1,2,3]) # True
```

7. Identity Operators

```
x = [1,2]
y = [1,2]
print(x is y)       # False
print(x is not y)   # True
```


GeeksforGeeks

Python, short-circuiting happens with and / or operators:

```
# AND short-circuit (stops if first is False)
print(False and (10/0))    # False (does not check 10/0)

# OR short-circuit (stops if first is True)
print(True or (10/0))      # True (does not check 10/0)
```

Python, flow control statements decide the order of execution:

1. Conditional (if, elif, else):

```
x = 10
if x > 0:
    print("Positive")
elif x == 0:
    print("Zero")
else:
    print("Negative")
```

2. Looping (for, while):

```
for i in range(3):
    print(i)      # 0 1 2

count = 3
while count > 0:
    print(count) # 3 2 1
    count -= 1
```

3. Jump (break, continue, pass):

```
for i in range(5):
    if i == 2:
        continue    # skips 2
    if i == 4:
        break       # stops at 4
    print(i)        # 0 1 3

pass    # does nothing, placeholder
```

1. break → exits loop immediately

```
for i in range(5):
    if i == 3:
        break
    print(i)
# Output: 0 1 2
```

2. continue → skips current iteration

```
for i in range(5):
    if i == 3:
        continue
    print(i)
# Output: 0 1 2 4
```

range() → generates a sequence of numbers.

Syntax:

```
range(start, stop, step)
```

- `start` → starting number (default 0)
- `stop` → end (excluded)
- `step` → increment (default 1)

Examples:

```
print(list(range(5)))         # [0,1,2,3,4]
print(list(range(2, 7)))      # [2,3,4,5,6]
print(list(range(1, 10, 2))) # [1,3,5,7,9]
```

Nested loop → a loop inside another loop.

Example:

```
for i in range(3):          # outer loop
    for j in range(2):      # inner loop
        print(i, j)
```

Output:

```
0 0
0 1
1 0
1 1
2 0
2 1
```

Python, data types define the type of values stored in variables.

1. Numeric Types

- int → whole numbers

```
x = 10
print(type(x))    # <class 'int'>
```

- **float** → decimal numbers

```
y = 3.14
print(type(y))    # <class 'float'>
```

- **complex** → numbers with real + imaginary part

```
z = 2 + 3j
print(type(z))    # <class 'complex'>
```

2. Sequence Types

- **str (String) → text in quotes**

```
name = "Python"
print(type(name))    # <class 'str'>
```

- **list** → ordered, mutable collection

```
lst = [1, 2, 3]
print(type(lst))    # <class 'list'>
```

- **tuple** → ordered, immutable collection

```
tup = (1, 2, 3)
print(type(tup))    # <class 'tuple'>
```

3. Set Types

- **set** → unordered, unique elements

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

- **frozenset** → immutable set

```
fs = frozenset({1, 2, 3})
print(type(fs))    # <class 'frozenset'>
```

4. Mapping Type

- dict → key-value pairs

```
d = {"a": 1, "b": 2}
print(type(d))    # <class 'dict'>
```

5. Boolean Type

```
flag = True
print(type(flag))    # <class 'bool'>
```

6. None Type

```
x = None
print(type(x))    # <class 'NoneType'>
```

1. What is a String?

A string is a sequence of characters enclosed in single (' '), double (" "), or triple quotes (''' ''' or """ """).

Example:

```
s = "Hello Python"
print(type(s))    # <class 'str'>
```

2. What is String Slicing?

Slicing means extracting part of a string using indexing.

Syntax: string[start:end:step]

```
s = "Python"
print(s[0:4])     # "Pyth"   (from index 0 to 3)
print(s[:3])      # "Pyt"    (start default 0)
print(s[2:])      # "thon"   (till end)
print(s[::-1])    # "nohtyP" (reverse)
```

3. String Methods (Commonly Used)

Method	Syntax	Example	Output
upper()	s.upper()	"hello".upper()	"HELLO"
lower()	s.lower()	"HELLO".lower()	"hello"
title()	s.title()	"hello world".title()	"Hello World"
strip()	s.strip()	" hi ".strip()	"hi"
replace()	s.replace(old, new)	"hello".replace("h","H")	"Hello"
split()	s.split(sep)	"a,b,c".split(",")	['a','b','c']
join()	sep.join(list)	"-".join(["a","b"])	"a-b"
find()	s.find(sub)	"hello".find("e")	1
count()	s.count(sub)	"banana".count("a")	3
startswith()	s.startswith(val)	"hello".startswith("he")	True
endswith()	s.endswith(val)	"hello".endswith("lo")	True

What is a List?

- A list is an ordered, mutable (changeable), and indexed collection in Python.
- Written with square brackets [ ].
- Can store different data types.

Example:

```
my_list = [10, "hello", 3.14]
print(my_list)        # [10, 'hello', 3.14]
print(type(my_list)) # <class 'list'>
```

2. Common List Methods

Method	Syntax	Example	Output
append()	list.append(x)	l=[1,2]; l.append(3)	[1,2,3]
insert()	list.insert(i,x)	l=[1,2]; l.insert(1,5)	[1,5,2]
extend()	list.extend(iterable)	l=[1]; l.extend([2,3])	[1,2,3]
remove()	list.remove(x)	l=[1,2]; l.remove(1)	[2]
pop()	list.pop([i])	l=[1,2]; l.pop()	returns 2, list=[1]
clear()	list.clear()	l=[1,2]; l.clear()	[]
index()	list.index(x)	[1,2,3].index(2)	1
count()	list.count(x)	[1,2,2].count(2)	2
sort()	list.sort()	l=[3,1,2]; l.sort()	[1,2,3]
reverse()	list.reverse()	l=[1,2]; l.reverse()	[2,1]
copy()	list.copy()	l=[1,2]; m=l.copy()	m=[1,2]

Method	Syntax	Example	Output
count()	tuple.count(x)	(1,2,2,3).count(2)	2
index()	tuple.index(x)	(1,2,3).index(2)	1

1. What is a Tuple?

- A tuple is an ordered, immutable (unchangeable) collection in Python.
- Written with parentheses ( ).
- Faster than lists.

Example:

```
python

my_tuple = (10, "hello", 3.14)
print(my_tuple)          # (10, 'hello', 3.14)
print(type(my_tuple))    # <class 'tuple'>
```

2. Tuple Methods (very few because tuples are immutable)

3. Tuple Operations

Although methods are limited, you can still:

```
python

t = (1, 2, 3, 4)

print(t[1])       # Indexing → 2
print(t[1:3])     # Slicing → (2,3)
print(len(t))     # Length → 4
print(max(t))     # Max → 4
print(min(t))     # Min → 1
print(sum(t))     # Sum → 10
```

1. What is a Set?

- A set is an unordered collection of unique elements in Python.
- Written with curly braces { }.
- No duplicates, no indexing.

Example:

```
python

my_set = {1, 2, 3, 2}
print(my_set)          # {1, 2, 3}
print(type(my_set))    # <class 'set'>
```

2. Set Methods

Method	Syntax	Example	Output
add()	set.add(x)	s={1,2}; s.add(3)	{1,2,3}
update()	set.update(iterable)	s={1}; s.update([2,3])	{1,2,3}
remove()	set.remove(x)	s={1,2}; s.remove(1)	{2}
discard()	set.discard(x)	s={1,2}; s.discard(3)	{1,2} (no error)
pop()	set.pop()	s={1,2,3}; s.pop()	removes random element
clear()	set.clear()	s={1,2}; s.clear()	set()
union()	A.union(B)	{1,2}.union({2,3})	{1,2,3}
intersection()	A.intersection(B)	{1,2}.intersection({2,3})	{2}
difference()	A.difference(B)	{1,2,3}.difference({2,3})	{1}
symmetric_difference()	A.symmetric_difference(B)	{1,2}.symmetric_difference({2,3})	{1,3}
issubset()	A.issubset(B)	{1,2}.issubset({1,2,3})	True
issuperset()	A.issuperset(B)	{1,2,3}.issuperset({2})	True
isdisjoint()	A.isdisjoint(B)	{1,2}.isdisjoint({3})	True
copy()	set.copy()	s={1,2}; t=s.copy()	{1,2}

1. What is a Dictionary?

- A dictionary is an unordered collection of key–value pairs.
- Keys must be unique & immutable (like strings, numbers, tuples).
- Values can be anything.
- Written with curly braces { }.

Example:

```
python

my_dict = {"name": "John", "age": 25, "city": "Delhi"}
print(my_dict["name"])    # John
print(type(my_dict))      # <class 'dict'>
```

2. Dictionary Methods

Method	Syntax	Example	Output
get()	dict.get(key[, default])	d={"a":1}; d.get("a")	1
keys()	dict.keys()	d={"a":1}; d.keys()	dict_keys(['a'])
values()	dict.values()	d={"a":1}; d.values()	dict_values([1])
items()	dict.items()	d={"a":1}; d.items()	dict_items([('a',1)])
update()	dict.update(other)	d={"a":1}; d.update({"b":2})	{'a':1,'b':2}
pop()	dict.pop(key[,default])	d={"a":1,"b":2}; d.pop("a")	returns 1, dict={'b':2}
popitem()	dict.popitem()	d={"a":1,"b":2}; d.popitem()	removes last → ('b',2)
setdefault()	dict.setdefault(key[,default])	d={"a":1}; d.setdefault("b",2)	{'a':1,'b':2}
fromkeys()	dict.fromkeys(keys[,value])	dict.fromkeys([1,2],0)	{1:0,2:0}
copy()	dict.copy()	d={"a":1}; x=d.copy()	{'a':1}
clear()	dict.clear()	d={"a":1}; d.clear()	{}

Mutability vs Immutability in Python

1. Mutable Objects

- Can be changed (modified) after creation.
- **Examples** → list, dict, set

Example:

```
python

lst = [1, 2, 3]
lst[0] = 100
print(lst)    # [100, 2, 3]  ✓ changed
```

2. Immutable Objects

- **Cannot be changed after creation.**
- Any modification creates a new object.
- **Examples** → int, float, str, tuple, frozenset

Example:

```
python

s = "hello"
s = s + " world"
print(s)    # "hello world" (new string created)
```

- **Mutable** → changeable (list, dict, set)
- **Immutable** → unchangeable (int, float, str, tuple, frozenset)

1. Shallow Copy

- Creates a new object, but copies references of inner objects.
- Changes in nested objects affect both copies.
- Done using: copy.copy() or list.copy( ) or slicing.

Example:

```
python

import copy

list1 = [[1, 2], [3, 4]]
shallow = copy.copy(list1)

shallow[0][0] = 99
print(list1)      # [[99, 2], [3, 4]]  ✓ affected
print(shallow)    # [[99, 2], [3, 4]]
```

2. Deep Copy

- Creates a completely independent copy, including nested objects.
- Changes in one don't affect the other.
- Done using: copy.deepcopy().

Example:

```
python

import copy

list1 = [[1, 2], [3, 4]]
deep = copy.deepcopy(list1)

deep[0][0] = 99
print(list1)    # [[1, 2], [3, 4]]    ✓ not affected
print(deep)     # [[99, 2], [3, 4]]
```

Shallow Copy → Copies outer object, inner references shared.

Deep Copy → Full independent clone (outer + inner).

Function in Python

A function is a block of reusable code that performs a specific task.
It helps in code reusability, modularity, and readability.

Syntax:

```
python

def function_name(parameters):
    """docstring (optional)"""
    # code block
    return value    # optional
```

Example:

```
python

def add(a, b):
    return a + b

print(add(5, 3))    # 8
```

1. Default Argument

- A parameter with a default value.
- If no value is passed, the default is used.

Syntax & Example:

```
python

def greet(name="Guest"):
    print("Hello", name)

greet("John")    # Hello John
greet()          # Hello Guest
```

2. Keyword Argument

- We pass arguments using parameter names.
- Order does not matter.

Syntax & Example:

```
python

def student(name, age):
    print(name, age)

student(age=21, name="Alice")  # Alice 21
```

Default Argument → assigns a default value.

Keyword Argument → specify by name, order-free.

Recursion in Python

- Recursion is a process where a function calls itself to solve a problem.

- Every recursive function must have a base case (stopping condition), otherwise it runs infinitely.

Syntax:

```
python

def function_name(parameters):
    if condition:           # base case
        return value
    else:
        return
function_name(modified_parameters)
```

Example: Factorial using Recursion

```
python

def factorial(n):
    if n == 0:            # base case
        return 1
    else:
        return n * factorial(n-1)   # recursive call

print(factorial(5))   # 120
```

Lambda Function in Python

- A lambda function is a small, anonymous (nameless) function.
- Defined using the keyword lambda.
- Can take any number of arguments but has only one expression.

Syntax:

```
lambda arguments : expression
```

Examples:

```
python

# Add two numbers
add = lambda a, b: a + b
print(add(5, 3))    # 8

# Square of a number
square = lambda x: x * x
print(square(4))    # 16

# Using with sort()
nums = [(1, "b"), (3, "a"), (2, "c")]
nums.sort(key=lambda x: x[1])
print(nums)    # [(3, 'a'), (1, 'b'), (2, 'c')]
```

GeeksforGeeks

GATE CSE BATCH

KEY HIGHLIGHTS:

- 300+ HOURS OF RECORDED CONTENT
- 900+ HOURS OF LIVE CONTENT
- SKILL ASSESSMENT CONTESTS
- 6 MONTHS OF 24/7 ONE-ON-ONE AI DOUBT ASSISTANCE
- SUPPORTING NOTES/DOCUMENTATION AND DPPS FOR EVERY LECTURE

COURSE COVERAGE:

- ENGINEERING MATHEMATICS
- GENERAL APTITUDE
- DISCRETE MATHEMATICS
- DIGITAL LOGIC
- COMPUTER ORGANIZATION AND ARCHITECTURE
- C PROGRAMMING
- DATA STRUCTURES
- ALGORITHMS
- THEORY OF COMPUTATION
- COMPILER DESIGN
- OPERATING SYSTEM
- DATABASE MANAGEMENT SYSTEM
- COMPUTER NETWORKS

LEARNING BENEFIT:

- GUIDANCE FROM EXPERT MENTORS
- COMPREHENSIVE GATE SYLLABUS COVERAGE
- EXCLUSIVE ACCESS TO E-STUDY MATERIALS
- ONLINE DOUBT-SOLVING WITH AI
- QUIZZES, DPPS AND PREVIOUS YEAR QUESTIONS SOLUTIONS

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

STAR MENTOR CS/DA

KHALEEL SIR
ALGORITHM & OS
29 YEARS OF TEACHING EXPERIENCE

CHANDAN SIR
DIGITAL LOGIC
GATE AIR 23 & 26 / EX-ISRO

SATISH SIR
DISCRETE MATHEMATICS
BE in IT from MUMBAI UNIVERSITY

MALLESHAM SIR
M.TECH FROM IIT BOMBAY
AIR – 114, 119, 210 in GATE
(CRACKED GATE 8 TIMES)
14+ YEARS EXPERIENCE

VIJAY SIR
DBMS & COA
M. TECH FROM NIT
14+ YEARS EXPERIENCE

PARTH SIR
DA
IIIT BANGALORE ALUMNUS
FORMER ASSISTANT PROFESSOR

SAKSHI MA'AM
ENGINEERING MATHEMATICS
IIT ROORKEE ALUMNUS

SHAILENDER SIR
C PROGRAMMING & DATA STRUCTURE
M.TECH in Computer Science
15+ YEARS EXPERIENCE

AVINASH SIR
APTITUDE
10+ YEARS OF TEACHING EXPERIENCE

AJAY SIR
PH.D. IN COMPUTER SCIENCE
12+ YEARS EXPERIENCE