

Basics

- **Functions**

- **Definition:** A reusable block of code that performs a specific task. Can be built-in (like `print`), imported from modules, or custom-written.
- **Example:**

```
print("hello world") # Output: hello world
x = 5
print("x is", x)      # Output: x is 5
print(f"x is {x}")    # Output: x is 5 (f-string format)
```

- **Common DevOps Interview Q:** “How would you write a Python function to check if a service is running?”
 - **Answer:** Use `subprocess` module to execute shell commands and check return codes or output

- **Literals**

- **Definition:** Fixed values that are directly written in code (actual data, not variables).
- **Example:**

```
20          # Integer literal
0.1         # Float literal
"Abhishek" # String literal
True        # Boolean literal (True or False)
```

- **Common DevOps Interview Q:** “What’s the difference between `0`, `'0'`, and `0.0` as literals?”
 - **Answer:** `0` is an integer, `'0'` is a string, `0.0` is a float. Type matters in DevOps automation for comparison and data processing

- **Operators**

- **Definition:** Symbols that perform operations on values (arithmetic, comparison, logical, etc.)
- **Arithmetic Operators:**

```
2 ** 3       # Exponentiation: 2^3 = 8
2 * 3        # Multiplication: 6
10 / 2        # Division: 5.0 (always returns float)
5 // 2        # Floor Division: 2 (returns integer)
5 % 2         # Modulo (remainder): 1
```

```
10 + 5      # Addition: 15
10 - 5      # Subtraction: 5
```

- **Common DevOps Interview Q:** “Why use floor division (//) instead of regular division (/) in DevOps scripts?”
 - Answer: Floor division returns an integer, which is useful for port numbers, counts, or any integer-only calculations. Regular division returns float which may cause type errors

• Variables

- **Definition:** Named containers that store values in memory. You can change the value they hold.
- **Example:**

```
name = "Abhishek"          # String variable
port = 8080                # Integer variable
is_active = True            # Boolean variable
name = "Gawade"             # Can reassign to new value
```

- **Common DevOps Interview Q:** “Why avoid hardcoding values like IP addresses or ports directly in scripts?”
 - Answer: Use variables to store configuration values for reusability, maintainability, and easy updates across environments

• Comments

- **Definition:** Text in code that is ignored by Python. Used to explain code for developers.
- **Example:**

```
# This is a single line comment
name = "Abhishek" # Comments can be inline too
```

- **Common DevOps Interview Q:** “Why is commenting important in DevOps automation scripts?”
 - Answer: Comments document why decisions were made, making scripts maintainable for future troubleshooting and understanding complex logic

• Input

- **Definition:** Taking user input from keyboard or command line to use inside a program.
- **Example:**

```
name = input("Enter your name: ")      # Takes user input
age = int(input("Enter your age: "))    # Convert input to integer
```

- **Common DevOps Interview Q:** “How would you safely accept user input in a production DevOps script?”

- Answer: Validate and sanitize all input, use type conversion with error handling, and avoid using eval() for security reasons

• String

- **Definition:** A sequence of characters (text) enclosed in quotes. Can be single ', double ", or triple quotes.
- **Example:**

```
print("ha" * 10)                      # Output: hahahaha
message = "Hello DevOps"
print(message[0])                      # Output: H (index
print(message[6:])                     # Output: DevOps (
```

- **Common DevOps Interview Q:** “How would you parse a configuration string from a log file?”
 - Answer: Use string methods like .split(), .strip(), .replace() to extract and manipulate configuration values

• Comparison Operators:

- **Definition:** Operators that compare two values and return True or False.
- **Example:**

```
5 == 5          # Equal to: True
5 != 3          # Not equal to: True
5 > 3           # Greater than: True
5 < 3           # Less than: False
5 >= 5          # Greater or equal: True
5 <= 5          # Less or equal: True
```

- **Common DevOps Interview Q:** “How would you check if a service port is responding in a health check script?”
 - Answer: Use comparison operators to verify status codes (response_code == 200) or response times (latency < threshold)

Conditional Statements

• if/elif/else

- **Definition:** Execute different code blocks based on whether conditions are true or false.
- **Example:**

```
age = 25
if age < 18:
    print("Age is below 18")
```

```

        elif age >= 18 and age <= 50:
            print("Age is middle age")
        else:
            print("Old Age")
    
```

- **Common DevOps Interview Q:** “How would you use if/elif/else to validate environment configurations?”
 - **Answer:** Check environment variables or config values; execute different deployment strategies based on the environment (dev/staging/prod)

• Loop - While

- **Definition:** Repeat a block of code as long as a condition is True.
- **Example:**

```

secret_number = 10
guessed_number = int(input("Enter a number: "))

while guessed_number != secret_number:
    guessed_number = int(input("Guess Again: "))
else:
    print("You have finally guessed the number right")
    
```

- **Common DevOps Interview Q:** “How would you use a while loop to retry a failed API call?”
 - **Answer:** Loop while retry count < max_retries, with backoff delays to handle transient failures gracefully

• Loop - For

- **Definition:** Iterate over a sequence (list, string, range) a fixed number of times.
- **Example:**

```

for i in range(0, 10):
    print(i)
# Output: 0 1 2 3 4 5 6 7 8 9
    
```

- **Common DevOps Interview Q:** “How would you use a for loop to process multiple servers or log files?”
 - **Answer:** Iterate through a list of server IPs or file paths, execute commands on each, and aggregate results

Logic & Bitwise Operators

- **Definition:** Operators used to combine multiple conditions (and, or) or invert conditions (not).
- **Example:**

```
is_running = True
is_healthy = False

if is_running and is_healthy:          # Both must be True
    print("Service OK")

if is_running or is_healthy:          # At least one must be
    print("Something is working")

if not is_running:                   # Opposite of condition
    print("Service is down")
```

- **Common DevOps Interview Q:** “How would you check multiple conditions in a health check script?”
 - Answer: Use and to ensure all critical checks pass (CPU < threshold AND memory < threshold), use or for fallback checks, use not for error conditions

Lists

- **Definition:** An ordered, mutable collection that can store multiple values of any type. Values can be added, removed, or modified.
 - **Example:**

```
countries = ["US", "UK", "IN"]
print(countries[0])                      # Output: US (indexing)
countries[0] = "CN"
print(countries)                         # Modify element
print(countries[-1])                    # Output: IN (last elemen
del countries[1]
print(countries)                         # Remove element
# Output: ['CN', 'IN']
```

- **Common DevOps Interview Q:** “How would you use lists in deployment automation?”
 - Answer: Store lists of servers to deploy to, collect log lines, gather metrics, iterate and execute commands on each item

List Methods

- **Definition:** Built-in functions you can call on lists to manipulate them.
 - **Common Methods:**

```
list = [3, 1, 4, 1, 5]
list.append(9)
list.insert(0, 2)
list.remove(1)
list.pop()
list.sort()
list.reverse()

# Add element: [3, 1, 4, 1, 5,
# Insert at position: [2, 3, 1
# Remove first occurrence: [2,
# Remove and return last: [2,
# Sort in place: [1, 2, 3, 4,
# Reverse order: [5, 4, 3, 2,
```

- **Common DevOps Interview Q:** “How would you remove duplicate servers from a deployment list?”
 - **Answer:** Use list methods like `.remove()`, or convert to a `set()` to automatically eliminate duplicates, then convert back to list
-

Iterating Through For Loop

- **Definition:** Loop through each element in a list one by one.
- **Example:**

```
ages = [10, 20, 30, 40]
total = 0

for age in ages:
    total = total + age

print(total) # Output: 100
```

- **Common DevOps Interview Q:** “How would you iterate through a list of configuration files and validate each?”
 - **Answer:** Use a for loop to process each file, read its contents, validate against schema, and log results

Enumerate

- **Definition:** Get both the index (position) and value while iterating through a list.
- **Example:**

```
fruits = ["apple", "banana", "cherry"]

for index, fruit in enumerate(fruits):
    print(index, fruit)

# Output:
# 0 apple
# 1 banana
# 2 cherry
```

- **Common DevOps Interview Q:** “When would you use enumerate instead of just iterating?”
 - **Answer:** When you need the position number, like logging which server (index 0, 1, 2) failed, or processing files in order

Slicing

- **Definition:** Extract a portion of a list by specifying start and end positions.
- **Example:**

```
letters = [1, 2, 3, 4, 5]
new_letters = letters[0:2] # Get elements at index 0 and
```

```
recent_logs = logs[-10:]      # Get last 10 elements
```

- **Common DevOps Interview Q:** “How would you get the latest N log entries?”
 - Answer: Use negative indexing with slicing like `logs[-100:]` to get the last 100 entries

Check in List

- **Definition:** Quickly check if a value exists in a list.
- **Example:**

```
servers = ['web-1', 'web-2', 'db-1', 'cache-1']
print('web-1' in servers)          # Output: True
print('web-3' not in servers)      # Output: True
```

- **Common DevOps Interview Q:** “How would you verify if a server is in the allowed deployment list?”
 - Answer: Use `if server in allowed_servers:` to validate before deployment

Find Index of Element

- **Definition:** Get the position (index) of an element in a list.
- **Example:**

```
my_list = [0, 3, 4, 1, 2]
print(my_list.index(1))          # Output: 3 (element 1 is at index 1)
```

- **Common DevOps Interview Q:** “How would you find which server in the list needs to be restarted?”
 - Answer: Use `.index()` to find its position, then use that index to access or manipulate that specific server

Function

- **Definition:** A reusable block of code that performs a specific task, accepts inputs (parameters), and returns outputs.
- **Example:**

```
def sum(num1, num2):
    return num1 + num2

result = sum(5, 10)  # Output: 15
```

- **Common DevOps Interview Q:** “Why is writing reusable functions important in DevOps scripts?”
 - Answer: Functions reduce code duplication, make testing easier, improve maintainability, and allow you to deploy the same logic across multiple environments

scripts

Variable Scope

- **Definition:** The region of code where a variable can be accessed. Variables defined in functions are local; outside are global.
- **Example - Local Scope:**

```
num = 10

def square():
    print(num) # Can access outer num

square() # Output: 10
```

- **Example - Function Scope Override:**

```
num = 10

def square():
    num = 20 # Creates new local variable
    print(num)

square()      # Output: 20
print(num)    # Output: 10 (outer num unchanged)
```

- **Example - Global Keyword:**

```
num = 10

def change_global():
    global num
    num = 20 # Modifies global variable

change_global()
print(num) # Output: 20
```

- **Common DevOps Interview Q:** “Why should you be careful with global variables in DevOps automation?”
 - **Answer:** Global variables can cause unexpected side effects, make debugging difficult, and cause state management issues in parallel execution

Tuples

- **Definition:** An ordered, immutable (unchangeable) collection. Once created, elements cannot be added, removed, or modified.
- **Example:**

```

tuple1 = (1, 2, 3)      # Using parentheses
tuple2 = 1, 2, 3        # Without parentheses (also valid)

print(tuple1[0])        # Output: 1
# tuple1[0] = 99 # ERROR! Cannot modify tuples

coordinates = (40.7128, -74.0060) # Lat, Long immutable

```

- **Use Cases:**
 - Storing constant data like configuration values, coordinates, or fixed options
 - Using as dictionary keys (lists can't be keys because they're mutable)
 - Function return values that shouldn't be accidentally modified
 - **Common DevOps Interview Q:** “Why would you use tuples for server coordinates or configuration constants?”
 - **Answer:** Tuples prevent accidental modification of critical configuration data, can be used as dictionary keys, and signal to other developers that data is fixed
-

Dictionaries

- **Definition:** An unordered collection of key-value pairs. Each key is unique and maps to a value. Mutable and flexible.
- **Example:**

```

usernames = {
    'Abhishek': 'abhishek123',
    'Gawade': 'gawade123'
}

print(usernames["Abhishek"])      # Output: abhishek123
print(usernames.keys())          # Output: dict_keys(['Al
print(usernames.values())        # Output: dict_values([
print(usernames.items())         # Output: dict_items([('A

```

- **Common DevOps Interview Q:** “How would you use dictionaries to store configuration data?”
 - **Answer:** Use dicts for config files (YAML/JSON), store server metadata (IP, port, status), or map environment variables to values

Modifying Dictionaries

- **Definition:** Ways to change, add, or remove key-value pairs from a dictionary.
- **Example:**

```

usernames = {'Abhishek': 'abhishek123', 'Gawade': 'gawade123'}

usernames["Abhishek"] = "abhishekNew123"      # Modify existing key
usernames.update({"prajakta": "prajakta123"})  # Add new key-value pair
del usernames["Gawade"]                         # Delete specific key

```

```

usernames.clear()                                # Delete all items
usernames.popitem()                             # Delete last item

```

- **Common DevOps Interview Q:** “How would you dynamically update a server configuration dictionary?”
 - Answer: Use `.update()` to merge new config, check with `.keys()` before updating to avoid overwrites, use `.pop()` to safely remove deprecated entries
-

List vs Tuples vs Set

- **Definition:** Three different collection types with different properties.
- **Quick Reference** (Ordered means insertion order, not sorting):

Feature	List	Tuple	Set
Syntax	<code>[1, 2, 3]</code>	<code>(1, 2, 3)</code>	<code>{1, 2, 3}</code>
Ordered	✓ Yes	✓ Yes	✗ No (unordered)
Mutable	✓ Yes (can change)	✗ No (immutable)	✓ Yes (can add/remove)
Allows duplicates	✓ Yes	✓ Yes	✗ No (unique elements)
Indexing	✓ Supported	✓ Supported	✗ Not supported
Use case	General collection	Fixed collection	Unique items, set ops

- **DevOps Example Usage:**

```

# List: Track logs in order
error_logs = ["error1", "error2", "error1", "error3"]

# Tuple: Immutable server config
server_config = ("prod-server", 8080, "active")

# Set: Unique server names (remove duplicates)
deployed_servers = {"web-1", "web-2", "db-1"}
unique_servers = set(error_logs)  # Removes duplicates

```

- **Common DevOps Interview Q:** “How would you efficiently find unique errors in a log file?”
 - Answer: Read logs into a list, convert to set to eliminate duplicates, then iterate for analysis
-

Errors and Exceptions

- **Definition:** Errors occur when code doesn't work as expected. Exceptions are errors that Python detects at runtime. Handle them to prevent crashes.
- **Example:**

```

try:
    result = 10 / 0 # This will cause an error
except ZeroDivisionError as e:
    print(f"Error caught: {e}")
    # Output: Error caught: division by zero

```

- **Common Exception Types:**

- ZeroDivisionError: Division by zero
- ValueError: Invalid value (e.g., int("abc"))
- FileNotFoundError: File doesn't exist
- KeyError: Dictionary key doesn't exist
- IndexError: List index out of range
- TypeError: Wrong data type operation

- **Example - Multiple Exceptions:**

```

try:
    print("abc")
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Invalid value provided")
except Exception as e:
    print(f"Unexpected error: {e}")

```

- **Raising Exceptions:**

```

def validate_port(port):
    if port < 1 or port > 65535:
        raise ValueError("Port must be between 1 and 65535")

validate_port(70000) # Will raise ValueError

```

- **Common DevOps Interview Q:** “Why is exception handling critical in DevOps scripts?”

- Answer: Gracefully handle failures (network issues, missing files, API timeouts) to prevent entire deployments from failing; log errors for debugging; implement retry logic

Internals

- **Definition:** Understanding how Python works internally - how it processes code.

- **Key Concepts:**

- **Python is an Interpreted Language:** Executes code line-by-line (not compiled upfront like Java/C++)

- Interpreter reads code and converts to machine instructions immediately
 - Slower but more flexible and easier to debug
 - **CPython:** Python written in C language
 - Most common implementation, default Python you download
 - Good for general-purpose use
 - **Cython:** Translates Python to C for faster execution
 - Use when performance is critical (data processing)
 - **Jython:** Python written in Java
 - Runs on Java Virtual Machine (JVM)
 - Useful for Java ecosystem integration
 - **Common DevOps Interview Q:** “Why is Python suitable for DevOps automation despite being interpreted?”
 - **Answer:** Interpreted nature allows quick development and testing, huge standard library for system operations, easy cross-platform compatibility, and sufficient performance for most infrastructure tasks. Use Cython only if profiling shows bottlenecks
-

Some Common Python Packages/Modules for DevOps

- **Definition:** Pre-written code libraries that add functionality. Some are built-in (standard library), others need installation.
- **Common DevOps Packages:**
 - **ansible:** Configuration management and infrastructure automation
 - **pytest:** Testing framework for writing and running tests
 - **docker:** Docker SDK to interact with Docker containers programmatically
 - **requests:** HTTP library for making API calls to services
 - **paramiko:** SSH & SFTP client for remote server access
 - **tensorflow / pytorch:** Machine learning (for anomaly detection, predictions)
 - **transformers:** NLP models (for log analysis, text processing)
 - **openai:** Integrate AI capabilities into automation scripts
 - **os:** Built-in module for operating system operations
- **Example - OS Module:**

```
import os
print(os.getcwd())                                # Show current directory
os.chdir("/path/to/your/folder")                  # Change to desired directory
print(os.getcwd())                                # Verify change
```

```
# DevOps example: Process files in a directory
for file in os.listdir("/var/log"):
    if file.endswith(".log"):
        print(f"Processing: {file}")
```

- **Common DevOps Interview Q:** “Which Python packages would you use to automate server provisioning?”
 - *Answer:* ansible for configuration management, paramiko for SSH operations, requests for API calls to cloud providers, docker to manage containers
-

Modules & Custom Packages

- **Definition:** Modules are Python files containing code. You can import them to reuse code across multiple scripts.
- **Example - Creating and Using a Module:**

```
# my_module.py
def greet(name):
    return f"Hello, {name}!"

# main.py
import my_module

print(my_module.greet("Alice")) # Output: Hello, Alice!
```

- **Example - Importing Specific Functions:**

```
# main.py
from my_module import greet

print(greet("Bob")) # Output: Hello, Bob! (works directly)
```

- **Common DevOps Interview Q:** “How would you structure a DevOps automation project with multiple scripts?”
 - *Answer:* Create reusable modules for common functions (logging, API calls, validation), import them into main scripts, version control everything, write tests for each module
-

Courses

Free

- <https://www.youtube.com/watch?v=rfscVS0vtbw>

Paid

- <https://learn.kodekloud.com/courses/python-basics>