Comprehensive Python Programming Guide

Table of Contents

Table of Contents

1. Introduction to Python

Python is a high-level, interpreted programming language known for its simplicity and readability. It was created by Guido van Rossum and first released in 1991.

Key features of Python:

1. Easy to learn and read
2. Dynamically typed
3. Interpreted language
4. Supports multiple programming paradigms (procedural, object-oriented, functional)
5. Extensive standard library and third-party packages

Python is widely used in various domains, including:

1. Web development
2. Data science and machine learning
3. Artificial intelligence
4. Scientific computing
5. Automation and scripting

2. Setting Up Your Python Environment

2.1 Installing Python

1. Visit the official Python website (https://www.python.org)
2. Download the latest version for your operating system
3. Run the installer and follow the instructions

Verify the installation by opening a terminal/command prompt and typing:

```
python --version
```

2.2 Setting Up a Virtual Environment

Virtual environments allow you to create isolated Python environments for your projects.

```
# Create a virtual environment
    python -m venv myenv


# Activate the virtual environment
# On Windows:
    myenv\Scripts\activate
# On macOS and Linux:
```

```
source myenv/bin/activate
```

 # Deactivate the virtual environment

```
deactivate
```

2.3 Installing Packages

Use pip, Python's package installer, to install third-party packages:

```
pip install package_name
```

2.4 Integrated Development Environments (IDEs)

Popular Python IDEs and text editors:

1. PyCharm
2. Visual Studio Code with Python extension
3. Jupyter Notebook (for data science)
4. IDLE (comes with Python installation)

2.5 Your First Python Program

Create a file named `hello_world.py`:

```
print("Hello, World!")
```

Run the program:

```
python hello_world.py
```

3. Variables and Data Types

Python is dynamically typed, meaning you don't need to declare the type of a variable explicitly.

3.1 Basic Data Types

 # Integer

```
age = 25
```

 # Float

```
height = 1.75
```

 # String

```
name = "Alice"
```

 # Boolean

```
is_student = True
```

 # None (null value)

```
data = None
```

 # Check the type of a variable

```
print(type(age)) # <class 'int'>
```

3.2 Complex Data Types

 # List (mutable)

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

 # Tuple (immutable)

```
coordinates = (10, 20)
```

 # Dictionary

```
person = {"name": "Bob", "age": 30}
```

 # Set

```
unique_numbers = {1, 2, 3, 4, 5}
```

3.3 Type Conversion

 # String to int

```
age = int("25")
```

 # Int to string

```
age_str = str(25)
```

 # String to float

```
price = float("19.99")
```

 # List to set

```
unique_fruits = set(["apple", "banana", "apple"])
```

4. Logging to Console and String Interpolation

4.1 Basic Printing

```
print("Hello, World!")
```

```
name = "Alice"
age = 30
print("My name is", name, "and I am", age, "years old.")
```

4.2 String Formatting

```
# Using f-strings (Python 3.6+)
name = "Bob"
age = 25
print(f"My name is {name} and I am {age} years old.")

# Using .format() method
print("My name is {} and I am {} years old.".format(name, age))

# Using % operator (older style)
print("My name is %s and I am %d years old." % (name, age))
```

4.3 Formatted String Literals (f-strings)

```
import math

radius = 5
```

```
area = math.pi * radius ** 2
print(f"The area of a circle with radius {radius} is {area:.2f}")
```

5. User Input

5.1 Basic Input

```
name = input("Enter your name: ")
print(f"Hello, {name}!")
```

5.2 Input with Type Conversion

```
age = int(input("Enter your age: "))
height = float(input("Enter your height in meters: "))
print(f"You are {age} years old and {height} meters tall.")
```

6. Operators and Control Flow Statements

6.1 Arithmetic Operators

```
a = 10
b = 3

print(a + b) # Addition
print(a - b) # Subtraction
print(a * b) # Multiplication
```

```
print(a / b) # Division
print(a // b) # Floor division
print(a % b) # Modulus
print(a ** b) # Exponentiation
```

6.2 Comparison Operators

```
x = 5
y = 10

print(x == y) # Equal to
print(x != y) # Not equal to
print(x < y) # Less than
print(x > y) # Greater than
print(x <= y) # Less than or equal to
print(x >= y) # Greater than or equal to
```

6.3 Logical Operators

```
a = True
b = False

print(a and b) # Logical AND
```

```
print(a or b) # Logical OR
print(not a) # Logical NOT
```

6.4 If-Elif-Else Statements

```
age = 20

if age < 18:
print("You are a minor.")
elif age >= 18 and age < 65:
print("You are an adult.")
else:
print("You are a senior citizen.")
```

6.5 Ternary Operator

```
age = 20
status = "Adult" if age >= 18 else "Minor"
print(status)
```

7. Loops

7.1 For Loop

 # Iterating over a list

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

 # Iterating over a range

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

 # Enumerate

```
for index, fruit in enumerate(fruits):
print(f"{index}: {fruit}")
```

7.2 While Loop

```
count = 0
while count < 5:
print(count)
count += 1
```

7.3 Loop Control Statements

 # Break

```
for i in range(10):
```

```
    if i == 5:
    break
    print(i)
```

 # Continue

```
    for i in range(10):
    if i % 2 == 0:
    continue
    print(i)
```

 # Pass (placeholder)

```
    for i in range(5):
    pass
```

8. Lists and Tuples

8.1 Lists

 # Creating a list

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

 # Accessing elements

```
print(fruits[0]) # First element
print(fruits[-1]) # Last element

# Slicing
print(fruits[1:3]) # ['banana', 'cherry']

# Modifying lists
fruits.append("date")
fruits.insert(1, "blueberry")
fruits.remove("banana")
popped_fruit = fruits.pop()

# List comprehension
squares = [x**2 for x in range(10)]
```

8.2 Tuples

```
# Creating a tuple
coordinates = (10, 20)

# Accessing elements
x, y = coordinates
```

 # Tuples are immutable

 # coordinates[0] = 15 # This will raise an error

 # Tuple with a single element

```
single_element_tuple = (42,)
```

9. Functions

9.1 Defining and Calling Functions

```
def greet(name):
return f"Hello, {name}!"

message = greet("Alice")
print(message)
```

9.2 Default Arguments

```
def power(base, exponent=2):
return base ** exponent

print(power(3)) # 9
print(power(3, 3)) # 27
```

9.3 *args and **kwargs

```
def sum_all(*args):
return sum(args)


print(sum_all(1, 2, 3, 4, 5)) # 15


def print_info(**kwargs):
for key, value in kwargs.items():
print(f"{key}: {value}")


print_info(name="Alice", age=30, city="New York")
```

9.4 Lambda Functions

```
square = lambda x: x ** 2
print(square(5)) # 25


# Using lambda with map
numbers = [1, 2, 3, 4, 5]
squared = list(map(lambda x: x**2, numbers))
print(squared) # [1, 4, 9, 16, 25]
```

10. Dictionaries and Sets

10.1 Dictionaries

 # Creating a dictionary

```
person = {"name": "Alice", "age": 30, "city": "New York"}
```

 # Accessing values

```
print(person["name"])
```

 # Adding or modifying key-value pairs

```
person["job"] = "Engineer"
person["age"] = 31
```

 # Dictionary methods

```
keys = person.keys()
values = person.values()
items = person.items()
```

 # Dictionary comprehension

```
squared_numbers = {x: x**2 for x in range(5)}
```

10.2 Sets

 # Creating a set

```
fruits = {"apple", "banana", "cherry"}
```

 # Adding and removing elements

```
fruits.add("date")

fruits.remove("banana")
```

 # Set operations

```
set1 = {1, 2, 3, 4, 5}

set2 = {4, 5, 6, 7, 8}

union = set1 | set2

intersection = set1 & set2

difference = set1 - set2
```

 # Set comprehension

```
even_numbers = {x for x in range(10) if x % 2 == 0}
```

11. Classes and Objects

11.1 Defining a Class

```
class Dog:
def __init__(self, name, age):
self.name = name
self.age = age

def bark(self):
print(f"{self.name} says Woof!")

# Creating an object
my_dog = Dog("Buddy", 3)
my_dog.bark()
```

11.2 Inheritance

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

def speak(self):
pass

class Cat(Animal):
```

```
def speak(self):
return f"{self.name} says Meow!"

class Dog(Animal):
def speak(self):
return f"{self.name} says Woof!"

cat = Cat("Whiskers")
dog = Dog("Buddy")
print(cat.speak())
print(dog.speak())
```

11.3 Class and Static Methods

```
class MathOperations:
@staticmethod
def add(x, y):
return x + y

@classmethod
def multiply(cls, x, y):
return x * y
```

```
print(MathOperations.add(5, 3))
print(MathOperations.multiply(4, 2))
```

12. String Methods

```
text = "Hello, World!"

# Common string methods
print(text.lower())
print(text.upper())
print(text.capitalize())
print(text.title())
print(text.strip())
print(text.replace("World", "Python"))
print(text.split(", "))

# Checking string properties
print(text.startswith("Hello"))
print(text.endswith("!"))
print("World" in text)
```

 # Formatting

```
name = "Alice"
age = 30
formatted = "My name is {} and I am {} years old".format(name, age)
print(formatted)
```

13. Error Handling, Data Validation, and Type Conversion

13.1 Try-Except Blocks

```
try:
result = 10 / 0
except ZeroDivisionError:
print("Cannot divide by zero")
except Exception as e:
print(f"An error occurred: {e}")
else:
print("No error occurred")
finally:
print("This always executes")
```

13.2 Raising Exceptions

```
def validate_age(age):
if age < 0:
raise ValueError("Age cannot be negative")
return age

try:
validate_age(-5)
except ValueError as e:
print(e)
```

13.3 Data Validation

```
def is_valid_email(email):
import re
pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
return re.match(pattern, email) is not None

print(is_valid_email("user@example.com"))
print(is_valid_email("invalid-email"))
```

13.4 Type Conversion

 # String to int/float

```
age = int("25")
price = float("19.99")
```

 # int/float to string

```
age_str = str(25)
price_str = str(19.99)
```

 # String to list

```
items = "apple,banana,cherry".split(",")
```

 # List to string

```
joined = ", ".join(items)
```

14. Modules and Packages

14.1 Importing Modules

```
import math
print(math.pi)
```

```
from datetime import datetime
```

```
print(datetime.now())


import random as rd
print(rd.randint(1, 10))
```

14.2 Creating Your Own Module

 # mymodule.py

```
def greet(name):
return f"Hello, {name}!"


PI = 3.14159
```

 # Using the module

```
import mymodule
print(mymodule.greet("Alice"))
print(mymodule.PI)
```

14.3 Working with Packages

```
my_package/
__init__.py
module1.py
```

```
        module2.py
# In __init__.py
        from . import module1
        from . import module2


# Using the package
        import my_package
        my_package.module1.function1()
        my_package.module2.function2()
```

15. File I/O

15.1 Reading from a File

```
# Reading an entire file
        with open('example.txt', 'r') as file:
        content = file.read()
        print(content)


# Reading line by line
        with open('example.txt', 'r') as file:
        for line in file:
```

```
print(line.strip())
```

15.2 Writing to a File

```
# Writing to a file
with open('output.txt', 'w') as file:
file.write("Hello, World!\n")
file.write("This is a new line.")

# Appending to a file
with open('output.txt', 'a') as file:
file.write("\nThis line is appended.")
```

15.3 Working with CSV Files

```
import csv

# Reading a CSV file
with open('data.csv', 'r') as file:
csv_reader = csv.reader(file)
for row in csv_reader:
print(row)
```

 # Writing to a CSV file

```
data = [
['Name', 'Age', 'City'],
['Alice', 30, 'New York'],
['Bob', 25, 'Los Angeles']
]
with open('output.csv', 'w', newline='') as file:
csv_writer = csv.writer(file)
csv_writer.writerows(data)
```

15.4 Working with JSON Files

```
import json
```

 # Reading JSON

```
with open('data.json', 'r') as file:
data = json.load(file)
print(data)
```

 # Writing JSON

```
data = {'name': 'Alice', 'age': 30, 'city': 'New York'}
with open('output.json', 'w') as file:
```

```
json.dump(data, file, indent=4)
```

16. Basic Data Structures and Algorithms

16.1 Stacks and Queues

 # Stack using a list

```
stack = []
stack.append(1)
stack.append(2)
stack.append(3)
print(stack.pop()) # 3
```

 # Queue using collections.deque

```
from collections import deque
queue = deque()
queue.append(1)
queue.append(2)
queue.append(3)
print(queue.popleft()) # 1
```

16.2 Linked List

```
class Node:
```

```
def __init__(self, data):
self.data = data
self.next = None

class LinkedList:
def __init__(self):
self.head = None

def append(self, data):
new_node = Node(data)
if not self.head:
self.head = new_node
return
current = self.head
while current.next:
current = current.next
current.next = new_node

def display(self):
current = self.head
```

```
        while current:
        print(current.data, end=" -> ")
        current = current.next
        print("None")


# Usage
        ll = LinkedList()
        ll.append(1)
        ll.append(2)
        ll.append(3)
        ll.display() # 1 -> 2 -> 3 -> None
```

16.3 Binary Search

```
        def binary_search(arr, target):
        left, right = 0, len(arr) - 1
        while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
        return mid
        elif arr[mid] < target:
        left = mid + 1
```

```
    else:
    right = mid - 1
    return -1

# Usage
    sorted_list = [1, 3, 5, 7, 9, 11, 13]
    print(binary_search(sorted_list, 7)) # 3
    print(binary_search(sorted_list, 6)) # -1
```

16.4 Sorting Algorithms

 # Bubble Sort

```
    def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
    for j in range(0, n - i - 1):
    if arr[j] > arr[j + 1]:
    arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

 # Quick Sort

```
    def quick_sort(arr):
```

```
    if len(arr) <= 1:
    return arr
    pivot = arr[len(arr) // 2]
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle + quick_sort(right)


# Usage
    unsorted_list = [64, 34, 25, 12, 22, 11, 90]
    print(bubble_sort(unsorted_list.copy()))
    print(quick_sort(unsorted_list.copy()))
```

17. Multithreading and Multiprocessing

17.1 Threading

```
    import threading
    import time


    def print_numbers():
    for i in range(5):
```

```
        time.sleep(1)
        print(f"Thread {threading.current_thread().name}: {i}")

# Create and start threads
    thread1 = threading.Thread(target=print_numbers, name="Thread 1")
    thread2 = threading.Thread(target=print_numbers, name="Thread 2")

    thread1.start()
    thread2.start()

    thread1.join()
    thread2.join()

    print("All threads completed")
```

17.2 Multiprocessing

```
    import multiprocessing
    import time

    def worker(num):
    print(f"Process {num} started")
```

```
    time.sleep(2)
    print(f"Process {num} finished")

if __name__ == "__main__":
    processes = []
    for i in range(4):
        p = multiprocessing.Process(target=worker, args=(i,))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()

    print("All processes completed")
```

18. Asynchronous Programming

18.1 Asyncio Basics

```
import asyncio

async def say_hello(name, delay):
```

```
await asyncio.sleep(delay)
print(f"Hello, {name}!")

async def main():
await asyncio.gather(
say_hello("Alice", 2),
say_hello("Bob", 1),
say_hello("Charlie", 3)
)

asyncio.run(main())
```

18.2 Asynchronous Context Managers

```
import asyncio

class AsyncResource:
async def __aenter__(self):
print("Acquiring resource")
await asyncio.sleep(1)
return self
```

```
async def __aexit__(self, exc_type, exc_val, exc_tb):
print("Releasing resource")
await asyncio.sleep(1)

async def main():
async with AsyncResource() as resource:
print("Using resource")
await asyncio.sleep(2)

asyncio.run(main())
```

19. Network Programming

19.1 Working with URLs

```
from urllib.parse import urlparse
from urllib.request import urlopen

url = "https://www.example.com/path/to/page?key1=value1&key2=value2"
parsed_url = urlparse(url)
print(f"Scheme: {parsed_url.scheme}")
print(f"Netloc: {parsed_url.netloc}")
```

```
print(f"Path: {parsed_url.path}")
print(f"Query: {parsed_url.query}")


# Fetching content from a URL
with urlopen("https://www.example.com") as response:
html = response.read().decode('utf-8')
print(html[:100]) # Print first 100 characters
```

19.2 Socket Programming

Server

```
import socket


def start_server():
host = '127.0.0.1'
port = 65432


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
s.bind((host, port))
s.listen()
print(f"Server listening on {host}:{port}")
conn, addr = s.accept()
```

```
with conn:
print(f"Connected by {addr}")
while True:
data = conn.recv(1024)
if not data:
break
conn.sendall(data)

if __name__ == "__main__":
start_server()
```

Client

```
import socket

def start_client():
host = '127.0.0.1'
port = 65432

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
s.connect((host, port))
s.sendall(b"Hello, server!")
```

```
    data = s.recv(1024)
    print(f"Received: {data.decode()}")

if __name__ == "__main__":
    start_client()
```

19.3 Making HTTP Requests

```
import requests

# GET request
response = requests.get("https://api.example.com/data")
print(response.status_code)
print(response.json())

# POST request
data = {"key": "value"}
response = requests.post("https://api.example.com/submit", json=data)
print(response.status_code)
print(response.json())
```