Python Programming

Short notes for B.Tech CSE / IT · BCA · MCA · M.Tech

Uncodemy Handwritten Notes · free for students

I. Introduction

- **Python** = a high-level, interpreted, dynamically typed, general-purpose language.
- Created by Guido van Rossum, first released in 1991.
- Code is first compiled to **bytecode** (`.pyc`), then run by the **Python Virtual Machine (PVM)**.
- Uses indentation (spaces) to mark blocks, not braces.

➤ Features

1. Simple, readable syntax
2. Interpreted and interactive (REPL)
3. Dynamically typed: no need to declare types
4. Object-oriented and also supports functional style
5. Large standard library ("batteries included")
6. Portable: same code runs on Windows, Linux, macOS

➤ Identifiers, keywords, comments

- Identifier: starts with a letter or `_`, case-sensitive, cannot be a keyword.
- Keywords: `if`, `else`, `for`, `while`, `def`, `class`, `return`, `import`, `lambda`, `None`, `True`, `False`, etc.
- Comment: `# single line`. Docstring: `"""text"""` as the first line of a function or class.

EXAM POINT Python is called interpreted, but it actually compiles source to bytecode first. The PVM then interprets the bytecode.

2. Data Types and Operators

Type	Example	Mutable?
int	10	No
float	3.14	No
complex	2+3j	No
bool	True	No
str	"hello"	No
list	[1, 2, 3]	Yes
tuple	(1, 2, 3)	No
set	{1, 2, 3}	Yes
dict	{"a": 1}	Yes

- `type(x)` gives the type. `id(x)` gives the identity (memory reference).
- Type conversion: `int("5")`, `float(3)`, `str(10)`, `list("abc")`.

➤ Operators

- Arithmetic: `+ - * / // % **`. Here `/` gives float, `//` gives floor division.
- Comparison: `== != < > <= >=`. Logical: `and or not`.
- Identity: `is`, `is not` (same object?). Membership: `in`, `not in`.
- Bitwise: `& | ^ ~ << >>`.

EXAMPLE `7 // 2 = 3`, `7 % 2 = 1`, `-7 // 2 = -4` (floor goes down), `2 ** 3 = 8`.

REMEMBER `==` checks value, `is` checks identity (same object in memory).

3. Control Flow

```
if marks >= 60:
    print("First")
elif marks >= 45:
    print("Second")
else:
    print("Fail")
```

- `for` loops over any iterable: `for i in range(1, 6):` gives 1 to 5.
- `range(start, stop, step)`: stop is not included.
- `while` loop runs while the condition is true.

- `break` exits the loop, `continue` skips to the next round, `pass` does nothing.
- Loop `else` runs only when the loop ends without break.

EXAMPLE `range(10, 0, -3)` gives 10, 7, 4, 1.

4. Strings and Collections

Strings

- Immutable sequence of characters. Indexing `s[0]`, negative indexing `s[-1]` = last char.
- Slicing `s[start:stop:step]`. `s[::-1]` reverses the string.
- Common methods: `upper()`, `lower()`, `strip()`, `split()`, `join()`, `replace()`, `find()`.

EXAMPLE `s = "PYTHON"`. `s[1:4] = "YTH"`, `s[-2:] = "ON"`, `s[::2] = "PTO"`.

List, tuple, set, dict

Feature	List	Tuple	Set	Dict
Brackets	`[ ]`	`( )`	`{ }`	`{k: v}`
Ordered	Yes	Yes	No	Yes (3.7+)
Duplicates	Allowed	Allowed	Not allowed	Keys unique
Mutable	Yes	No	Yes	Yes

- List methods: `append()`, `extend()`, `insert()`, `remove()`, `pop()`, `sort()`, `reverse()`.
- Dict methods: `keys()`, `values()`, `items()`, `get()`, `update()`.
- Set operations: `|` union, `&` intersection, `-` difference, `^` symmetric difference.

Comprehensions

```
squares = [x*x for x in range(5)]        # [0, 1, 4, 9, 16]
evens = [x for x in range(10) if x % 2 == 0]
d = {x: x*x for x in range(3)}           # {0: 0, 1: 1, 2: 4}
```

TIP A one-element tuple needs a comma: `(5,)`. `(5)` is just the int 5.

5. Functions

- Defined with `def`. Returns `None` if there is no `return`.
- Argument types:
  - **Positional**: `f(1, 2)`
  - **Keyword**: `f(b=2, a=1)`
  - **Default**: `def f(a, b=10)`
  - **Variable length**: `*args` (tuple), `**kwargs` (dict)
- **Lambda** = small anonymous function: `sq = lambda x: x * x`.
- `map(f, list)`, `filter(f, list)`, `reduce(f, list)` (from `functools`).
- **Recursion**: a function calling itself; needs a base case.

```
def fact(n):
    if n <= 1:
        return 1
    return n * fact(n - 1)
print(fact(5))   # 120
```

➤ Scope (LEGB rule)

- Python looks for a name in order: Local, Enclosing, Global, Built-in.
- `global x` lets a function change a global variable. `nonlocal` is for an enclosing function.

EXAM POINT Mutable default arguments are a trap. `def f(a=[])` shares the same list across calls. Use `a=None` instead.

6. Object-Oriented Programming

- **Class** = blueprint. **Object** = instance of a class.
- `__init__` = constructor. `self` = reference to the current object.

```
class Student:
    college = "ABC"              # class variable
    def __init__(self, name):
        self.name = name         # instance variable
    def show(self):
        print(self.name, Student.college)

s = Student("Riya")
s.show()
```

Four pillars

Pillar	Meaning	In Python
Encapsulation	Wrap data + methods together	`_x` protected, `__x` private (name mangling)
Inheritance	Child reuses parent code	`class B(A):`, `super()`
Polymorphism	Same name, different behaviour	Method overriding, operator overloading
Abstraction	Show only essentials	`abc` module, `@abstractmethod`

- Inheritance types: single, multiple, multilevel, hierarchical, hybrid.
- **MRO (Method Resolution Order)**: order Python searches parent classes. See it with `ClassName.mro()`.
- Magic (dunder) methods: `__str__`, `__len__`, `__add__`, `__eq__`.
- `@staticmethod` needs no `self`. `@classmethod` takes `cls`.

REMEMBER Python does not support method overloading by signature. The last definition wins. Use default arguments instead.

7. Exception Handling

```
try:
    x = int(input())
    print(10 / x)
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Not a number")
else:
    print("No error")
finally:
    print("Always runs")
```

- `else` runs only if no exception. `finally` always runs.
- `raise ValueError("msg")` raises an exception yourself.
- Custom exception: `class MyError(Exception): pass`.
- Common: `IndexError`, `KeyError`, `TypeError`, `NameError`, `FileNotFoundError`.

EXAM POINT Syntax error = caught before running. Exception = error while running (runtime).

8. File Handling

- `open(name, mode)`. Modes: `r` read, `w` write (overwrites), `a` append, `r+` read+write, `b` binary.
- Methods: `read()`, `readline()`, `readlines()`, `write()`, `close()`.
- Best practice: `with open("a.txt") as f:` closes the file automatically.

```
with open("data.txt", "w") as f:
    f.write("Hello\n")
with open("data.txt") as f:
    for line in f:
        print(line.strip())
```

- `seek(n)` moves the file pointer, `tell()` gives its position.
- `pickle` module stores Python objects in binary form (serialization).

9. Modules, Packages and Advanced Topics

- **Module** = a `.py` file. **Package** = a folder of modules (traditionally with `__init__.py`).
- Import styles: `import math`, `from math import sqrt`, `import numpy as np`.
- `if __name__ == "__main__":` runs code only when the file is run directly.
- Useful modules: `math`, `random`, `os`, `sys`, `datetime`, `re`, `json`, `collections`.
- `pip install name` installs third-party packages.

➤ Iterators and generators

- **Iterator**: object with `__iter__()` and `__next__()`.
- **Generator**: function with `yield`; produces values one by one, saves memory.

```
def count_up(n):
    i = 1
    while i <= n:
        yield i
        i += 1
print(list(count_up(3)))   # [1, 2, 3]
```

➤ Decorators

- A function that takes a function and returns a new function, adding behaviour.
- Applied with `@decorator_name` above a function.

➤ Shallow vs deep copy

- `copy.copy()` = shallow (inner objects shared). `copy.deepcopy()` = full independent copy.

> **TIP** Libraries to know by name: NumPy (arrays), Pandas (tables), Matplotlib (plots), Tkinter (GUI), Flask / Django (web).

10. Most Asked Questions (practice)

1. Features of Python. Why is it called an interpreted language?
2. Difference between list, tuple, set and dictionary.
3. Explain mutable vs immutable types with examples.
4. What are `*args` and `**kwargs`? Explain lambda, map and filter.
5. Explain the LEGB scope rule with a program.
6. Explain the four pillars of OOP in Python with code.
7. Explain try, except, else and finally with an example.
8. Write a program to read a file and count words and lines.
9. What is a generator? Difference between `yield` and `return`.
10. What is a decorator? Write a simple one.

> **TIP** Exams love short programs: factorial, Fibonacci, prime check, palindrome, string reverse and file word count. Practise writing them without errors.