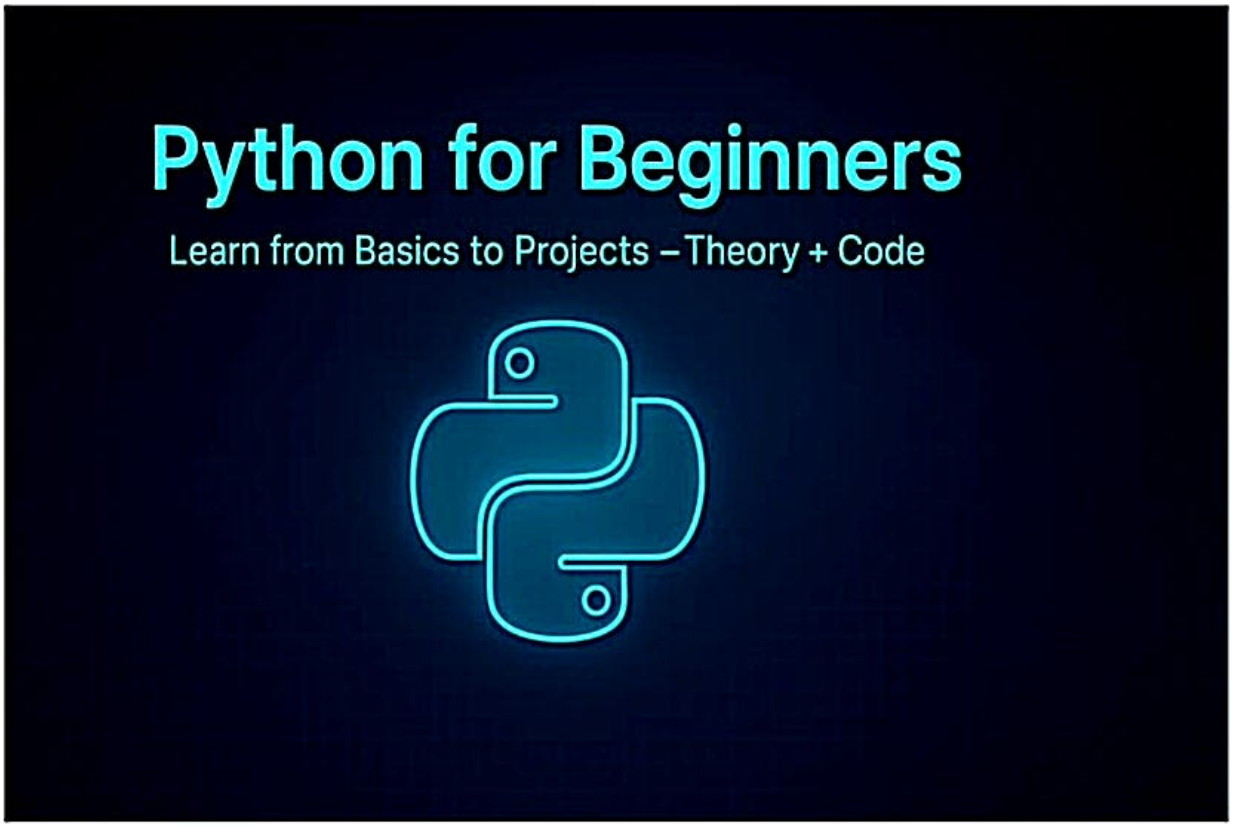
🐍Complete Python for Beginners — Theory + Code (A Practical, Friendly Guide)



> Tip: copy any code blocks into a file (e.g. example.py) and run with python example.py (or run in an interactive REPL or Jupyter notebook).

✅ What is Python?

Python is a high-level, interpreted programming language created by Guido van Rossum (late 1980s). It emphasizes readability and simplicity. Python is dynamically typed (types determined at runtime) and uses indentation to indicate block structure.

Why learners love it

- Clean, readable syntax
- Huge ecosystem (libraries for nearly everything)
- Great for beginners, yet powerful for professionals

✅ Top Features & Real-World Applications

Top features

- Readable, concise syntax
- Dynamic typing & automatic memory management (garbage collection)
- Batteries-included standard library
- Multi-paradigm: procedural, OOP, functional
- Huge third-party ecosystem (`pip`)
- Cross-platform (Windows, macOS, Linux)

Real-world uses

- Web development (Django, Flask)
- Data science & ML (Pandas, NumPy, scikit-learn, TensorFlow)
- Automation & scripting
- DevOps / infrastructure automation
- Desktop apps (Tkinter, PyQt)
- Game development (Pygame)
- APIs and microservices
- Prototyping and research

✅ How to Install Python

Windows / macOS / Linux

1. Download from python.org or use system package manager.
   - Windows: installer from python.org — check "Add Python to PATH".
   - macOS: use Homebrew `brew install python` (or use the official installer).
   - Linux: `sudo apt install python3` (Debian/Ubuntu) or appropriate package.
2. Verify:

```
python --version
# or
python3 --version
```

1. Create isolated environment (recommended):

```
# create venv
python3 -m venv myenv

# activate (macOS/Linux)
source myenv/bin/activate

# activate (Windows PowerShell)
.\myenv\Scripts\Activate.ps1

# install packages
pip install requests
```

✅ Running Your First Program

Create `hello.py`:

```
print("Hello, Python! 👋")
```

Run:

```
python hello.py
```

Or use the REPL:

```
python
>>> print("Hello from REPL")
```

1. Basic Syntax & Variables

Comments

```
# single-line comment
"""
multi-line
comment/string literal
"""
```

Variables and types

```
name = "Deepak"      # str
age = 25             # int
height = 5.9         # float
is_student = True    # bool

print(name, age, height, is_student)
```

Type checking

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

2. Data Types & Built-in Collections

Numbers, strings, booleans

```
a = 10        # int
b = 3.14      # float
s = "python"
flag = False
```

Lists (mutable ordered)

```
fruits = ["apple", "banana", "mango"]
fruits.append("orange")
print(fruits[0])   # apple
```

Tuples (immutable ordered)

```
coords = (10, 20)
# coords[0] = 5  → TypeError
```

Sets (unique unordered)

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

Dictionaries (key-value)

```
person = {"name": "Asha", "age": 30}
print(person["name"])
person["city"] = "Kochi"
```

3. Control Flow — Conditionals & Loops

If / elif / else

```
x = 10
if x > 0:
    print("positive")
elif x == 0:
    print("zero")
else:
    print("negative")
```

For loop

```
for i in range(5):  # 0..4
    print(i)
```

Iterate over list

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

While loop

```
n = 3
while n > 0:
    print(n)
    n -= 1
```

Loop control

```
for i in range(1,6):
    if i == 3:
        continue  # skip
    if i == 5:
        break     # exit loop
```

4. Functions (definition, arguments, return)

Simple function

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

print(greet("Sita"))
```

Default & keyword args

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

print(power(3))       # 9
print(power(2,3))     # 8
```

Variable args

```
def avg(*numbers):
    return sum(numbers)/len(numbers)

print(avg(1,2,3,4))
```

- *Keyword-only args / **kwargs

```
def info(**kwargs):
    for k, v in kwargs.items():
        print(k, v)

info(name="Asha", age=30)
```

5. Files & I/O

Read & write text files

```
# write
with open("notes.txt", "w", encoding="utf-8") as f:
    f.write("Hello\nLine 2\n")

# read
with open("notes.txt", "r", encoding="utf-8") as f:
    data = f.read()
    print(data)
```

CSV example

```
import csv

# write
rows = [["name","age"], ["Asha","30"], ["Ravi","28"]]
with open("people.csv", "w", newline='', encoding="utf-8') as f:
    writer = csv.writer(f)
    writer.writerows(rows)

# read
with open("people.csv","r", encoding="utf-8") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

6. Error Handling / Exceptions

```
try:
    x = int(input("Enter a number: "))
    result = 10 / x
except ValueError:
    print("That's not an integer.")
```

```
except ZeroDivisionError:
    print("Cannot divide by zero.")
except Exception as e:
    print("Unexpected error:", e)
finally:
    print("This runs always.")
```

7. Modules & Packages

Create & use module

- `math_utils.py`

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

- `main.py`

```
import math_utils
print(math_utils.add(2,3))
```

Install package with pip

```
pip install requests
```

Import from package

```
import requests
resp = requests.get("https://example.com")
print(resp.status_code)
```

8. Virtual Environments & `pip`

```
python3 -m venv env
source env/bin/activate    # macOS/Linux
# Windows: .\env\Scripts\activate
pip install requests
pip freeze > requirements.txt
pip install -r requirements.txt
```

Use virtual environments to keep project dependencies isolated.

9. Object-Oriented Programming (OOP)

Class example

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

    def greet(self):
        return f"Hi, I'm {self.name} and I'm {self.age}."

# inherit
class Student(Person):
    def __init__(self, name, age, roll):
        super().__init__(name, age)
        self.roll = roll

s = Student("Maya", 22, "S123")
print(s.greet())
```

Encapsulation and properties

```
class BankAccount:
    def __init__(self, balance=0):
```

```
        self._balance = balance   # convention: protected/private

    @property
    def balance(self):
        return self._balance

    def deposit(self, amt):
        if amt > 0:
            self._balance += amt
```

10. Common Pythonic Patterns

List comprehensions

```
nums = [1,2,3,4]
squares = [x*x for x in nums]
```

Dictionary & set comprehensions

```
names = ["a","b","a"]
counts = {name: names.count(name) for name in set(names)}
```

Generators

```
def fib(n):
    a, b = 0, 1
    for _ in range(n):
        yield a
        a, b = b, a + b
for x in fib(10):
    print(x)
```

Enumerate & zip

```
for i, val in enumerate(['a','b']):
    print(i, val)

for a, b in zip([1,2], ['x','y']):
    print(a,b)
```

11. Functional Tools: `map` , `filter` , `reduce` , `lambda`

```
nums = [1,2,3,4]
doubled = list(map(lambda x: x*2, nums))
evens = list(filter(lambda x: x%2==0, nums))

from functools import reduce
prod = reduce(lambda a,b: a*b, nums)
```

12. Working with JSON & APIs

JSON

```
import json

data = {"name":"Asha","age":30}
s = json.dumps(data)        # to JSON string
obj = json.loads(s)         # back to dict
```

API request

```
import requests
```

```
resp = requests.get("https://api.github.com")
print(resp.json())
```

13. Useful Standard Library Modules (short tour)

- `os` / `pathlib` — filesystem
- `sys` — interpreter interaction
- `datetime` — dates/times
- `collections` — Counter, defaultdict, namedtuple
- `itertools` — combinatoric iterators
- `json`, `csv`, `sqlite3`
- `re` — regular expressions
- `unittest` / `pytest` — testing
- `logging` — logging
- `subprocess` — run external commands

Example: Counter

```
from collections import Counter
words = "the cat and the hat".split()
print(Counter(words))
```

14. Testing & Debugging

Simple unittest

```
# save as test_calc.py
import unittest
from math import sqrt
```

```
class TestMath(unittest.TestCase):
    def test_sqrt(self):
        self.assertEqual(int(sqrt(16)), 4)

if __name__ == "__main__":
    unittest.main()
```

Debugging

- Use `print()` for quick debugging.
- Use `pdb` for step debugging:

```
python -m pdb myscript.py
```

- IDE debuggers (VS Code, PyCharm) are excellent.

15. Packaging & Distribution (brief)

Create `setup.py` or modern `pyproject.toml` to package a project. Use `pip install -e .` to install locally.

16. Asynchronous Python (intro)

Async/await

```
import asyncio

async def say_after(delay, msg):
    await asyncio.sleep(delay)
    print(msg)

async def main():
    await asyncio.gather(
        say_after(1, "hello"),
```

```
        say_after(2, "world")
    )

asyncio.run(main())
```

Useful for IO-bound tasks (HTTP, DB calls).

---

17. Common Beginner Mistakes & Best Practices

Mistakes

- Mutating default args (`def f(x, l=[]): ...`)
- Confusing `is` vs `==`
- Not using virtualenvs
- Overusing global variables

Best practices

- Use meaningful variable names
- Follow PEP8 style (use linters: `flake8`, `black`)
- Write tests for important logic
- Use exceptions properly
- Keep functions small and focused

---

18. Mini Projects (build these to practice)

1. **Todo CLI app** — CRUD tasks stored in JSON file.
2. **Web scraper** — fetch and parse pages with `requests` + `BeautifulSoup`.
3. **Expense tracker** — CSV/SQLite storage, simple CLI.
4. **Simple web app** — Flask app with one form and list view.

5. **Chatbot** — simple rule-based chatbot or use NLP libraries.
6. **Data analysis** — load CSV with Pandas, compute stats, plot with `matplotlib`.
7. **Automation scripts** — rename files in a folder, bulk image resizing.

19. Example: Small End-to-End Project — Todo CLI (complete)

Create `todo.py`:

```
import json
from pathlib import Path

DB = Path("todos.json")

def load_todos():
    if DB.exists():
        return json.loads(DB.read_text())
    return []

def save_todos(todos):
    DB.write_text(json.dumps(todos, indent=2))

def list_todos():
    todos = load_todos()
    if not todos:
        print("No todos.")
    for i, t in enumerate(todos, 1):
        status = "✔" if t["done"] else " "
        print(f"{i}. [{status}] {t['task']}")

def add_todo(task):
    todos = load_todos()
    todos.append({"task": task, "done": False})
    save_todos(todos)
```

```
    print("Added.")

def complete_todo(index):
    todos = load_todos()
    try:
        todos[index-1]["done"] = True
        save_todos(todos)
        print("Marked done.")
    except IndexError:
        print("Invalid index.")

def main():
    import sys
    if len(sys.argv) < 2:
        print("Usage: python todo.py [list|add|done] [task/index]")
        return
    cmd = sys.argv[1]
    if cmd == "list":
        list_todos()
    elif cmd == "add":
        task = " ".join(sys.argv[2:])
        add_todo(task)
    elif cmd == "done":
        idx = int(sys.argv[2])
        complete_todo(idx)
    else:
        print("Unknown command.")

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

Usage:

```
python todo.py add "buy milk"
python todo.py list
```

```
python todo.py done 1
```

20. Exercises (with hints / answers)

Exercise 1: Write a function `is_prime(n)` returns `True` if `n` is prime.

- Hint: check divisibility up to `int(sqrt(n))`

Exercise 2: Given list of dicts `people=[{"name":...,"age":...}]`, print names of people older than 25 using list comprehension.

Exercise 3: Read a CSV of sales and print total sales per product (use `csv` or `pandas`).

(If you want, I can provide full solutions for each — tell me which exercise.)

21. Path Forward — Next Topics to Learn

- Intermediate Python: decorators, context managers, metaclasses (later)
- Web frameworks: Flask → FastAPI → Django
- Data stack: Pandas, NumPy, matplotlib/seaborn, scikit-learn
- Databases: SQL (SQLite/Postgres), ORMs (SQLAlchemy)
- Testing: pytest, mocking
- Packaging & Dockerize Python apps
- Concurrency: threads, multiprocessing, asyncio

22. Quick Reference Cheatsheet (Short)

- `len(x)` — length
- `in` — membership
- `==`, `!=`, `<, >`
- `and`, `or`, `not`