Python

Basics to Advanced with examples, in simple words

Python Notes · easy to understand · free for students

1. Start with Python

- **Python** is an easy, readable, general purpose language. It is used for web apps, data science, AI, automation and scripting.
- It is **interpreted** (runs line by line) and **dynamically typed** (no need to declare types). Blocks of code are made with indentation (4 spaces), not braces.
- Run a file: save it as `hello.py` and type `python hello.py`. Or type `python` alone to try code line by line.

EXAMPLE Your first program.

```
# this is a comment
print("Hello, World!")
```

Area	Popular tools
Web	Django, Flask, FastAPI
Data	NumPy, Pandas, Matplotlib
AI and ML	scikit-learn, PyTorch, TensorFlow
Automation	os, requests, Selenium

2. Variables and Data Types

- A **variable** is a name that points to a value. No type is written. Names use letters, digits and _, cannot start with a digit, and are case-sensitive.

Type	Example	Meaning
int	10	Whole number
float	3.14	Decimal number
str	"hi"	Text
bool	True, False	Yes or no
list	[1, 2, 3]	Ordered, changeable collection
tuple	(1, 2)	Ordered, fixed collection

Type	Example	Meaning
set	{1, 2}	Unique items only
dict	{"a": 1}	Key and value pairs
NoneType	None	Means no value

EXAMPLE Create variables and check type.

```
age = 20
price = 99.5
name = "Asha"
print(type(age))      # <class 'int'>
```

- Convert types with `int('5')`, `float('2.5')`, `str(10)`. The function `input()` always gives a string.

EXAMPLE Take a number from the user.

```
n = int(input("Enter a number: "))
print(n * 2)
```

REMEMBER **Immutable** (cannot change after creation): int, float, str, tuple. **Mutable** (can change): list, dict, set.

3. Operators

Type	Operators
Arithmetic	+ - * / // % **
Comparison	== != > < >= <=
Logical	and, or, not
Assignment	= += -= *=
Membership	in, not in
Identity	is, is not

EXAMPLE Division types.

```
print(7 / 2)    # 3.5  normal division
print(7 // 2)   # 3    floor division
print(7 % 2)    # 1    remainder
print(2 ** 3)   # 8    power
```

- `==` checks if values are **equal**. `is` checks if two names point to the **same object**.

4. Strings

- A **string** is text in quotes. It is immutable. Index starts at `0`, and `-1` is the last character.

EXAMPLE Indexing and slicing.

```
s = "Python"
print(s[0], s[-1])    # P n
print(s[0:3])         # Pyt
print(s.upper(), len(s))   # PYTHON 6
```

Method	What it does
upper() / lower()	Change the case
strip()	Remove spaces at both ends
split() / join()	Break into a list / join a list into text
replace(a, b)	Replace a with b
find(x)	Position of x, or -1 if missing
startswith(x)	True if text begins with x

SYNTAX f-string puts variables inside text.

```
name = "Asha"
print(f"Hello {name}, next year you are {20 + 1}")
```

5. Lists, Tuples, Sets, Dicts

Type	Ordered	Changeable	Duplicates
list	Yes	Yes	Allowed
tuple	Yes	No	Allowed
set	No	Yes	Not allowed
dict	Yes (insertion order)	Yes	Keys are unique

EXAMPLE List and set.

```
nums = [3, 1, 2]
nums.append(4)
nums.sort()
print(nums)           # [1, 2, 3, 4]
print({1, 2, 2, 3})   # {1, 2, 3}
```

EXAMPLE Dictionary.

```
d = {"name": "Asha", "age": 20}
d["age"] = 21
for key, value in d.items():
    print(key, value)
```

SYNTAX List comprehension builds a list in one line.

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

6. Conditions and Loops

EXAMPLE if, elif, else.

```
marks = 72
if marks >= 90:
    print("A")
elif marks >= 60:
    print("B")
else:
    print("C")
```

EXAMPLE for and while loops.

```
for i in range(1, 4):     # 1 2 3
    print(i)

count = 0
while count < 3:
    count += 1
```

Keyword	Meaning
break	Stop the loop completely
continue	Skip to the next round
pass	Do nothing, a placeholder

- ***match-case*** (Python 3.10+) is a cleaner way to write many conditions: `match day:` then `case 'Sat':` and `case _:` for default.

7. Functions

- A ***function*** is a reusable block of code. Define with `def`, send values back with `return`.

EXAMPLE Default parameter.

```
def add(a, b=0):
    return a + b

print(add(2, 3))   # 5
print(add(2))      # 2
```

EXAMPLE *args, **kwargs and lambda.

```
def total(*nums):          # any number of values
    return sum(nums)

def info(**data):          # any number of name=value
    print(data)

square = lambda x: x * x   # small one-line function
```

REMEMBER A function with no `return` gives `None`. Never use a list as a default value like `def f(x, items=[])`, because it is shared between calls.

- ***Scope***: a variable made inside a function is ***local***. Use `global x` only when you really must.

8. Modules and Files

- A ***module*** is a Python file you can import. Install outside libraries with `pip install requests`.

EXAMPLE Import styles.

```
import math
from random import randint
print(math.sqrt(16))    # 4.0
print(randint(1, 6))    # random from 1 to 6
```

- Use `with open(...)`. It closes the file for you. Modes: `r` read, `w` write (erases old), `a` append.

EXAMPLE Write and read a file.

```
with open("notes.txt", "w") as f:
    f.write("Hello\n")

with open("notes.txt") as f:
    for line in f:
        print(line.strip())
```

Library	Use
math, random	Maths and random numbers
datetime	Dates and time
os, pathlib	Files and folders
json, csv	Read and write data files
re	Pattern matching (regex)
collections	Counter, defaultdict, deque

9. OOP (Classes)

- A **class** is a blueprint. An **object** is one thing made from it. `__init__` runs when the object is created and `self` means the object itself.

EXAMPLE Class and object.

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

    def hello(self):
        print("Hi", self.name)

s = Student("Asha")
s.hello()              # Hi Asha
```

EXAMPLE Inheritance.

```
class Animal:
    def speak(self):
        return "..."

class Dog(Animal):
    def speak(self):       # overrides the parent method
        return "Woof"

print(Dog().speak())       # Woof
```

Pillar	Simple meaning
Encapsulation	Keep data and methods together, protect data (_name)
Inheritance	Child class reuses the parent class
Polymorphism	Same method name, different behaviour
Abstraction	Show only what is needed, hide the rest

- `__str__` decides what `print(obj)` shows. Methods with double underscores are called **dunder** methods.

10. Error Handling

- Use **try / except** so the program does not crash when something goes wrong.

EXAMPLE Handle errors.

```
try:
    n = int(input("Number: "))
    print(10 / n)
except ZeroDivisionError:
    print("Cannot divide by zero")
except ValueError:
    print("Enter digits only")
else:
    print("No error")        # runs if no error
finally:
    print("Done")            # always runs
```

- Raise your own error with `raise ValueError('bad input')`. Make a custom one with `class MyError(Exception): pass`.

Error	When it happens
SyntaxError	Code is written wrongly
NameError	Variable name not defined
TypeError	Wrong type, like 5 + "a"
ValueError	Right type, wrong value, like int("abc")
IndexError	List index out of range
KeyError	Key not found in dict

11. Advanced Python

➤ Generators

- A **generator** uses `yield` to give values one at a time. It saves memory because it does not build the whole list.

EXAMPLE Generator function.

```
def count_up(n):
    i = 1
    while i <= n:
        yield i
        i += 1

for x in count_up(3):
    print(x)              # 1 2 3
```

Decorators

- A ***decorator*** wraps a function to add extra work, like logging, without changing the function.

EXAMPLE Simple decorator.

```
def log(func):
    def wrapper(*args, **kwargs):
        print("Running", func.__name__)
        return func(*args, **kwargs)
    return wrapper

@log
def hello():
    print("Hi")
```

More advanced ideas

Topic	Simple meaning
Context manager	The with statement. Sets up and cleans up for you
Type hints	`def add(a: int, b: int) -> int:` helps readers and tools
dataclass	Short way to make classes that mainly hold data
Virtual env	`python -m venv venv` keeps each project's libraries separate
async / await	Run many waiting tasks (network calls) together with asyncio
GIL	One thread runs Python code at a time. Use multiprocessing for heavy CPU work

EXAMPLE dataclass saves typing.

```
from dataclasses import dataclass

@dataclass
class Point:
    x: int
    y: int

print(Point(1, 2))     # Point(x=1, y=2)
```

12. Practice and Projects

Project	What you practice
1. Calculator	Functions, conditions, input
2. Number guessing game	Loops, random, conditions
3. To-do list in terminal	Lists, functions, file saving
4. Password generator	Strings, random, functions
5. Expense tracker	Dicts, json or csv files, OOP
6. Web scraper	requests, BeautifulSoup, error handling
7. Mini API	Flask or FastAPI, json, modules

➤ Common mistakes

Mistake	Fix
Wrong indentation	Use 4 spaces everywhere, never mix tabs
= instead of == in if	Use == to compare
Changing a list while looping it	Loop over a copy or build a new list
List as default argument	Use `None` and create the list inside
Forgetting `self` in methods	First parameter must be self

➤ Practice questions

1. What is the difference between list, tuple, set and dict?
2. Mutable vs immutable. Give examples.
3. What are *args and **kwargs?
4. Explain list comprehension with an example.
5. What is a decorator? What is a generator?
6. Explain the four pillars of OOP.
7. How does try, except, else, finally work?
8. Write a program to count the words in a text file.

- Practice daily and build one small project every week.