1. Giới thiệu về Python

→ Python là gì?

- Python là ngôn ngữ lập trình bậc cao, thông dịch (interpreted) và đa mục đích (general-purpose). Python dễ học, dễ đọc, phù hợp cho người mới bắt đầu nhưng vẫn đủ mạnh mẽ cho các chuyên gia.

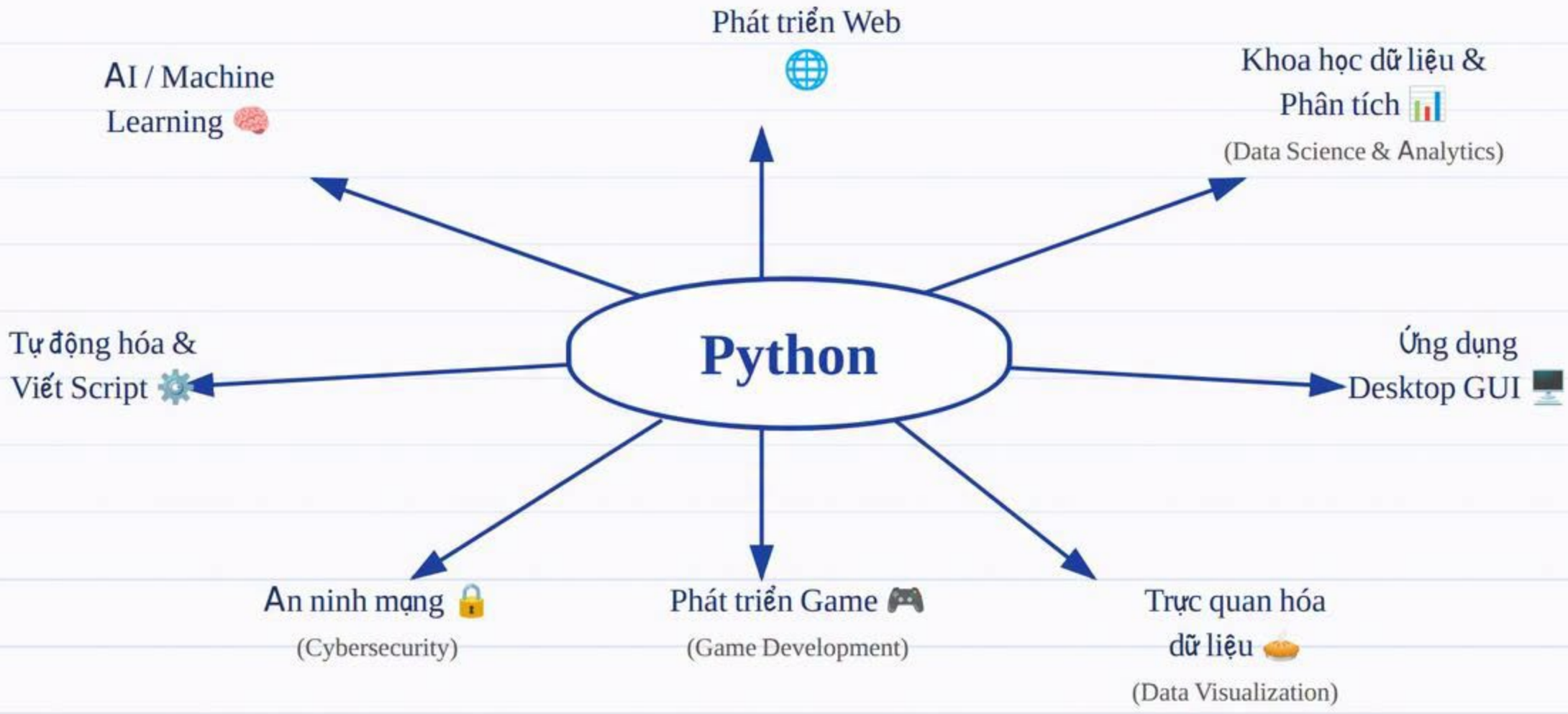
→ Tại sao chọn Python?

- Cú pháp đơn giản, dễ hiểu (giống tiếng Anh)
- Ngôn ngữ bậc cao (High-level language)
- Kiểu dữ liệu động (Dynamically typed)
- Ngôn ngữ thông dịch (Interpreted language)
- Đa nền tảng (Windows, Mac, Linux)
- Thư viện chuẩn phong phú
- Cộng đồng hỗ trợ đông đảo

→ Đặc điểm của Python

- Dễ học
- Cú pháp rõ ràng, dễ đọc
- Mã nguồn mở (Open source)
- Có thể mang đi dễ dàng (Portable)
- Có thể mở rộng (Extensible)
- Tích hợp tốt (Integrated)
- Quản lý bộ nhớ tự động
- Hỗ trợ hướng đối tượng (Object-oriented)

→ Ứng dụng của Python



→ Lịch sử (tóm tắt)

Python được tạo ra bởi Guido van Rossum và phát hành lần đầu vào năm 1991. Tên gọi Python được lấy cảm hứng từ chương trình truyền hình "Monty Python's Flying Circus".

→ Chương trình Python đầu tiên của bạn

```
# Chương trình này in ra Hello World
print("Hello World!")
```

↖ Chạy chương trình này và xem điều kỳ diệu của Python 🙂

2. Biến & Kiểu Dữ Liệu

→ Biến (Variable) là gì?

Biến (variable) là một vùng chứa (container) dùng để lưu trữ giá trị dữ liệu trong chương trình.

Mẹo ghi nhớ
Hãy hình dung biến (variable) như một chiếc hộp có tên và giá trị.

→ Quy tắc đặt tên biến

- Có thể chứa chữ cái (a-z, A-Z), chữ số (0-9) và dấu gạch dưới (_)
- Phải bắt đầu bằng chữ cái hoặc dấu gạch dưới
- Không được bắt đầu bằng chữ số
- Phân biệt chữ hoa/thường (name, Name và NAME là khác nhau)
- Không được dùng từ khóa (keyword) của Python (như if, else, for, class, ...)

→ Các kiểu dữ liệu có sẵn (Built-in Data Types)

Kiểu dữ liệu	Mô tả	Ví dụ	Kết quả type()
int	Số nguyên	10 , -25 , 1000	<class 'int'>
float	Số thập phân	10.5 , -3.14 , 0.001	<class 'float'>
str	Chuỗi văn bản (Text / String)	"Hello" , 'Python'	<class 'str'>
bool	Kiểu Boolean (True/False)	True , False	<class 'bool'>
NoneType	Kiểu đặc biệt (không có giá trị)	None	<class 'NoneType'>

→ Ví dụ (Chương trình)

```
name        = "Abhi"          →   # String
age         = 22              →   # Integer
height      = 5.9             →   # Float
is_student  = True            →   # Boolean
marks       = None            →   # NoneType
```

→ Sử dụng hàm type()

Chúng ta có thể dùng type() để tìm kiểu dữ liệu của bất kỳ giá trị nào.

```
type(name)          # <class 'str'>
type(age)           # <class 'int'>
type(height)        # <class 'float'>
type(is_student)    # <class 'bool'>
type(marks)         # <class 'NoneType'>
```

⭐ Python có kiểu dữ liệu động (dynamically typed),

3. Toán Tử (Operators)

Operators (toán tử) được dùng để thực hiện các phép toán trên biến và giá trị.

1. Toán tử số học (Arithmetic Operators)

+	-	*	/	//	%	**
Cộng	Trừ	Nhân	Chia	Chia lấy nguyên	Chia lấy dư	Lũy thừa

Ví dụ: 7 / 2 = 3.5 , 7 // 2 = 3 , 7 % 2 = 1

2 ** 3 = 8

2. Toán tử so sánh (Comparison Operators)

==	!=	>	<	>=	<=
Bằng	Khác	Lớn hơn	Nhỏ hơn	Lớn hơn hoặc bằng	Nhỏ hơn hoặc bằng

3. Toán tử gán (Assignment Operators)

=	+=	-=	*=	/=	//=	%=	**=
Gán	Gán cộng	Gán trừ	Gán nhân	Gán chia	Gán chia nguyên	Gán chia dư	Gán lũy thừa

Ví dụ: a = 5 a += 3 (a = a + 3)

4. Toán tử logic (Logical Operators)

and	or	not
Trả về True nếu cả hai đều True	Trả về True nếu một trong hai True	Đảo ngược kết quả

5. Toán tử thành viên (Membership Operators)

in	not in
Trả về True nếu giá trị có trong dãy (sequence)	Trả về True nếu giá trị không có trong dãy

6. Toán tử định danh (Identity Operators)

is	is not
Trả về True nếu cả hai biến trỏ đến cùng một object	Trả về True nếu hai biến không trỏ đến cùng object

7. Toán tử bitwise (dùng với số nguyên)

&	|	^	~	<<	>>
AND	OR	XOR	NOT	Dịch trái	Dịch phải

⭐ **Mẹo phỏng vấn**

Sự khác biệt giữa == và **is**

== → kiểm tra giá trị có bằng nhau không (nội dung giống nhau hay không)

is → kiểm tra định danh (identity) (cả hai có trỏ đến cùng một object trong bộ nhớ không)

Ví dụ nhanh

1) Số học (Arithmetic)

a = 10, b = 3

a + b = 13

a // b = 3

a % b = 1

a ** b = 1000

2) So sánh (Comparison)

5 > 3 → True

5 == 5 → True

5 != 3 → True

3) Logic (Logical)

True and False → False

True or False → True

not True → False

4) Thành viên (Membership)

'a' in 'python' → True

'z' not in 'python' → True

5) Định danh (Identity)

x = 10

y = 10

x is y → True

💡 Toán tử (operators) là nền tảng của mọi chương trình. Hãy nắm vững chúng! 🙂

4. Nhập Liệu & Câu Lệnh Điều Kiện

1. Nhập dữ liệu (Input) trong Python

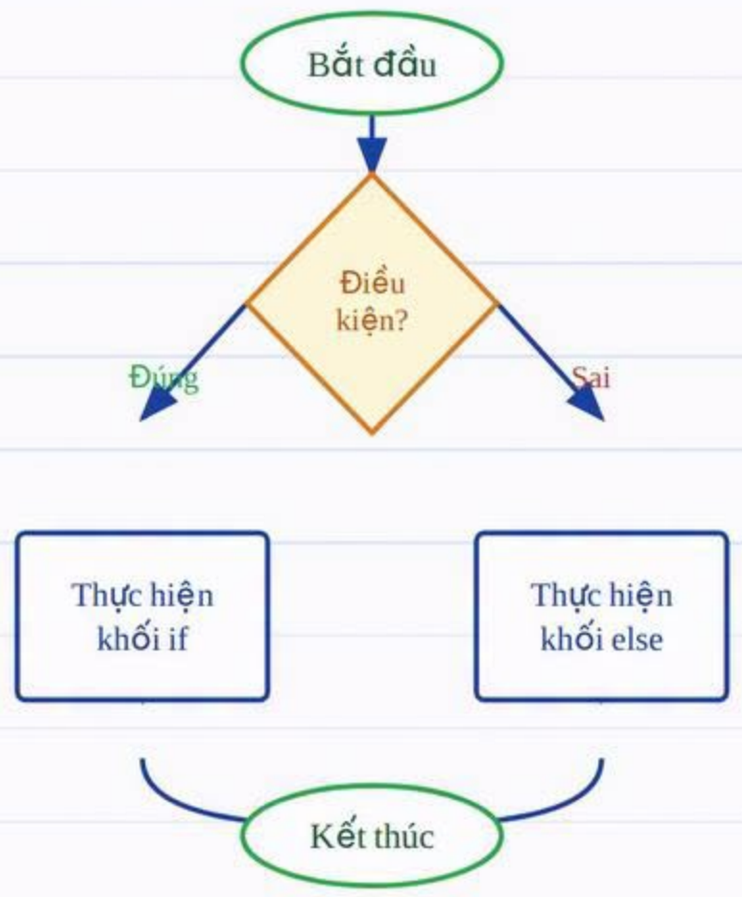
- Hàm input() được dùng để nhận dữ liệu nhập từ người dùng.
- Mặc định, input() trả về dữ liệu dạng chuỗi (string).

Cú pháp: name = input("Nhập gì đó:")

Ví dụ:

```
name = input("Nhập tên của bạn: ")
age = int(input("Nhập tuổi của bạn: "))
# int() dùng để chuyển chuỗi thành số nguyên
```

2. Sơ đồ if...else



3. if, elif, else

- if → thực hiện khối lệnh nếu điều kiện đúng (True)
- elif → kiểm tra điều kiện khác nếu điều kiện trước là sai (False)
- else → thực hiện nếu tất cả điều kiện đều sai

```
if condition1:
    statement1
elif condition2:
    statement2
elif condition3:
    statement3
else:
    statement4
```

→ kiểm tra đầu tiên

→ kiểm tra nếu condition1 sai

→ kiểm tra nếu condition2 sai

→ nếu không điều kiện nào đúng

4. Ví dụ thực tế

Chương trình kiểm tra đủ điều kiện bầu cử dựa trên độ tuổi.

```
age = int(input("Nhập tuổi của bạn: "))
if age >= 18:
    print("Bạn đủ điều kiện bầu cử.")
elif age > 0:
    print("Bạn chưa đủ tuổi bầu cử.")
else:
    print("Tuổi nhập không hợp lệ.")
```

Ghi chú

★ Trong Python, thụt lề (indentation) rất quan trọng.
★ Dùng 4 dấu cách cho mỗi khối lệnh.
★ Mọi giá trị khác 0 đều là True trong Python.
★ 0, False, None, "", [], {} được coi là False.

5. Câu lệnh if lồng nhau (Nested if)

- Một câu lệnh if bên trong một câu lệnh if khác.

```
if condition1:
    if condition2:
        statement1
    else:
        statement2
else:
    statement3
```

if bên trong chỉ thực thi khi if bên ngoài đúng (True)

Mẹo →

```
Luôn tự hỏi: Điều gì xảy ra nếu điều kiện đúng (True)?
Và điều gì xảy ra nếu nó sai (False)?
```

5. Vòng Lặp trong Python

Vòng lặp (loops) được dùng để lặp lại một khối lệnh nhiều lần.

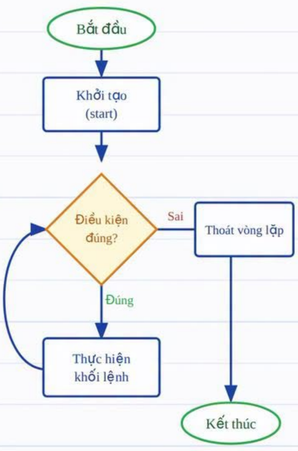
1. Vòng lặp for

- Dùng để duyệt qua một dãy (list, tuple, string, range, ...)
- Biết trước số lần lặp.

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

Kết quả: 0 1 2 3 4

Sơ đồ luồng (vòng lặp for)



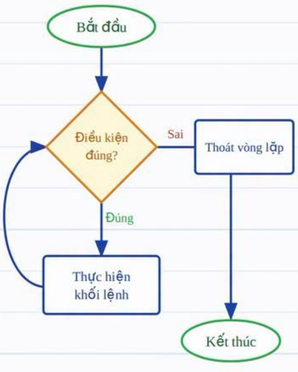
2. Vòng lặp while

- Lặp lại một khối lệnh khi điều kiện còn đúng (True).
- Dùng khi chưa biết trước số lần lặp.

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

Kết quả:
1
2
3
4
5

Sơ đồ luồng (vòng lặp while)



3. Các câu lệnh điều khiển vòng lặp

★ **break**: Dừng vòng lặp hoàn toàn.

★ **continue**: Bỏ qua lần lặp hiện tại và tiếp tục vòng lặp kế tiếp.

★ **pass**: Không làm gì cả. Dùng làm placeholder cho code sẽ viết sau.

★ **else**: Thực thi khi vòng lặp hoàn tất bình thường (không gặp break).

```
for i in range(5):
    if i == 3:
        break
    print(i)
```

Kết quả: 0 1 2

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Kết quả: 0 1 3 4

```
for i in range(3):
    pass  # không lỗi
```

```
for i in range(3):
    print(i)
else:
    print("Vòng lặp hoàn tất")
```

Kết quả: 0 1 2
Vòng lặp hoàn tất

4. Hàm range()

- Tạo ra một dãy các số.

Cú pháp: range(start, stop, step)

- start bao gồm, stop không bao gồm.

range()	Kết quả
range(5)	0 1 2 3 4
range(1, 6)	1 2 3 4 5
range(2, 10, 2)	2 4 6 8
range(10, 0, -2)	10 8 6 4 2

Lộ trình trang tiếp theo (Trang 6/15)

6. Hàm trong Python

Hàm (functions) là các khối lệnh thực hiện một tác vụ cụ thể. Chúng giúp tái sử dụng code (code reusability) và làm chương trình gọn gàng, có tổ chức hơn.

💡 **Nguyên tắc DRY**
Don't Repeat Yourself
(Đừng lặp lại chính mình)

1. Định nghĩa một hàm

```
def function_name(parameters):
    # thân của hàm
    return value
```

Ví dụ:

```
def add(a, b):          # tham số
    sum = a + b          # câu lệnh
    return sum           # câu lệnh return

result = add(10, 20)
print(result)    # Kết quả: 30
```

2. Gọi hàm (Function Call)

Hàm chỉ được thực thi khi được gọi.

```
add(5, 7)    # lời gọi hàm
```

Lưu ý:
Một hàm không được thực thi cho đến khi nó được gọi.
🙂

3. Tham số & Đối số

- **Tham số (Parameters)**: Biến được liệt kê trong định nghĩa hàm.
- **Đối số (Arguments)**: Giá trị được truyền vào hàm khi gọi.

Ví dụ:

```
def greet(name):
    print("Hello", name)
        ↑ tham số       ↑ đối số

greet("Abhi")    # lời gọi hàm
```

4. Câu lệnh Return

★ Câu lệnh return trả kết quả về cho nơi gọi hàm.

★ Một hàm chỉ có thể trả về một giá trị.

Ví dụ:

```
def square(x):
    return x * x

num = square(6)
print(num)    # Kết quả: 36
```

5. Đối số mặc định (Default Arguments)

★ Ta có thể cung cấp giá trị mặc định cho tham số.

Ví dụ:

```
def greet(name, msg="Hello"):
    print(msg, name)
```

greet("Abhi") # Hello Abhi

greet("Abhi", "Hi") # Hi Abhi

6. Đối số từ khóa (Keyword Arguments)

★ Ta có thể truyền đối số bằng tên tham số.

Ví dụ:

```
def info(name, age):
    print(name, age)
```

info(age=22, name="Abhi") # Abhi 22

7. Hàm Lambda (hàm ẩn danh)

★ Hàm nhỏ, viết trên một dòng, dùng từ khóa lambda.

Cú pháp:

```
lambda arguments: expression
```

Ví dụ:

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

8. Trả về nhiều giá trị

★ Một hàm có thể trả về nhiều giá trị dưới dạng tuple.

Ví dụ:

```
def operations(a, b):
    return a + b, a - b, a * b

x, y, z = operations(5, 3)
print(x, y, z)    # 8 2 15
```

⭐ Hàm giúp code của bạn module hóa, dễ tái sử dụng và dễ bảo trì.

Lộ trình trang tiếp theo (Trang 7/15)

7. Chuỗi (Strings) trong Python

1. Chuỗi (String) là gì?

Chuỗi (string) là dãy ký tự bất biến (immutable), được đặt trong dấu ' ' hoặc " " 🙂

- Chuỗi (string) là một dãy ký tự.
- Trong Python, ta có thể dùng dấu nháy đơn ('), dấu nháy kép (" ") hoặc dấu nháy ba ('" "').
- Chuỗi là bất biến (immutable - không thể thay đổi).

Ví dụ:

```
s1 = 'Hello'
s2 = "Python"
s3 = '''Multi
Line
String'''
```

2. Chỉ số trong Chuỗi (Indexing)

s = "HELLO"					
Chỉ số	0	1	2	3	4
Ký tự	H	E	L	L	O
Chỉ số âm	-5	-4	-3	-2	-1

- Chỉ số bắt đầu từ 0.
- Chỉ số âm bắt đầu từ -1 (từ phải sang trái).
- Truy cập bằng s[chỉ số]

Ví dụ:

```
s = "HELLO"
print(s[0])     # H
print(s[4])     # O
print(s[-1])    # O
print(s[-5])    # H
```

3. Cắt chuỗi (Slicing)

- Trích một phần của chuỗi bằng s[start : stop : step]

s =	P	Y	T	H	O	N
	0	1	2	3	4	5
	-6	-5	-4	-3	-2	-1

Ví dụ:

```
s = "PYTHON"
s[0:6]    → PYTHON  # cả chuỗi
s[0:3]    → PYT     # từ đầu đến 3-1
s[2:6]    → THON    # từ 2 đến hết
s[1:5:2]  → YH       # bước = 2
s[::-1]   → NOHTYP  # đảo ngược chuỗi
```

4. Các phương thức chuỗi thông dụng

Phương thức	Mô tả	Ví dụ	Kết quả
len(s)	Độ dài chuỗi	len("Hello")	5
s.upper()	Chuyển thành chữ hoa	"hello".upper()	"HELLO"
s.lower()	Chuyển thành chữ thường	"HELLO".lower()	"hello"
s.capitalize()	Viết hoa ký tự đầu, còn lại thường	"python".capitalize()	"Python"
s.title()	Viết hoa ký tự đầu mỗi từ	"python is fun".title()	"Python Is Fun"
s.strip()	Xóa khoảng trắng đầu & cuối	" abhi ".strip()	"abhi"
s.replace(a,b)	Thay a bằng b	"one two one".replace("one","1")	"1 two 1"
s.split()	Tách chuỗi thành list	"a,b,c".split(",")	["a", "b", "c"]
"sep".join(list)	Nối list thành chuỗi	"-".join(["a","b","c"])	"a-b-c"

5. Định dạng chuỗi (String Formatting)

- Kiểu cũ

```
name = "Abhi"
age = 22
print("My name is %s and age is %d" % (name, age))
```

- f-string (Khuyên dùng)

```
print(f"My name is {name} and age is {age}")
```

- format()

```
print("My name is {} and age is {}".format(name, age))
```

6. Ký tự thoát (Escape Characters)

\n : Xuống dòng
\t : Tab
\' : Nháy đơn
\" : Nháy kép
\\ : Gạch chéo ngược
\r : Về đầu dòng
\b : Xóa lùi
\f : Sang trang

Ví dụ:
print("Hello\nPython")
↓
Hello
Python

⭐ Chuỗi (Strings) có ở khắp mọi nơi!

Lộ trình trang tiếp theo (Trang 8/15)
• List, Tuple và Set

8. Collections trong Python

Python có bốn kiểu dữ liệu tập hợp (collection) chính: List, Tuple, Set và Dictionary.

Collections được dùng để lưu trữ nhiều phần tử trong một biến duy nhất. 🙂

Bảng so sánh

Đặc điểm	List	Tuple	Set	Dictionary
Cú pháp	[ ]	( )	{ }	{ key : value }
Có thứ tự (Ordered)	Có	Có	Không	Có (Python 3.7+)
Có thể thay đổi (Mutable)	Có	Không	Không* (nhưng phần tử là duy nhất)	Có
Cho phép trùng lặp	Có	Có	Không	Key - Không, Value - Có
Indexing / Slicing	Có	Có	Không	Key - Không, Value - Có
Dữ liệu đa kiểu	Có	Có	Có	Có
Trường hợp dùng	Lưu tập hợp có thứ tự	Lưu tập hợp cố định (không đổi được)	Lưu các phần tử duy nhất (không thứ tự)	Lưu cặp key-value (ánh xạ)

1. List (Có thể đổi, có thứ tự)

- Định nghĩa bằng dấu ngoặc vuông [ ].
- Có thể chứa giá trị trùng lặp.

Ví dụ:

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

Phương thức thông dụng:

- append() → thêm phần tử vào cuối
- insert(i, x) → chèn tại vị trí i
- remove(x) → xóa lần xuất hiện đầu tiên
- pop(i) → xóa & trả về phần tử tại i
- index(x) → trả về chỉ số xuất hiện đầu
- count(x) → đếm số lần xuất hiện
- sort() → sắp xếp list
- reverse() → đảo ngược list

2. Tuple (Không đổi, có thứ tự)

- Định nghĩa bằng dấu ngoặc tròn ( ).
- Sau khi tạo, không thể thay đổi.

Ví dụ:

```
days = ("Mon", "Tue", "Wed", "Thu")
```

Phương thức thông dụng:

- count(x) → đếm số lần xuất hiện
- index(x) → trả về chỉ số

Indexing của List fruits

A	B	C	D
0	1	2	3
-4	-3	-2	-1

Tuple nhanh hơn list. 🙂

3. Set (Không trùng lặp, không thứ tự)

- Định nghĩa bằng dấu ngoặc nhọn { }.
- Chỉ lưu các phần tử duy nhất.

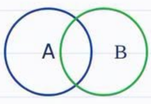
Ví dụ:

```
nums = {1, 2, 3, 3, 4, 5}  → {1, 2, 3, 4,
5}
```

Phương thức thông dụng:

- add(x) → thêm phần tử
- remove(x) → xóa phần tử (lỗi nếu không có)
- discard(x) → xóa phần tử (không lỗi nếu không có)
- union(s) → hợp hai tập hợp
- intersection(s) → phần tử chung
- difference(s) → phần tử chỉ có ở tập đầu

Sơ đồ Venn



A ∪ B → hợp (union)
A ∩ B → giao (intersection)
A - B → hiệu (difference)

4. Dictionary (Có thể đổi, có thứ tự)

- Lưu dữ liệu dưới dạng cặp key : value.
- Key phải là duy nhất và bất biến (string, number, tuple).

Ví dụ:

```
student = {"name": "Abhi",
  "age": 22, "city": "Pune"}
```

Phương thức thông dụng:

- keys() → trả về tất cả key
- values() → trả về tất cả value
- items() → trả về các cặp key-value
- get(key) → trả về giá trị theo key
- update(d) → cập nhật dictionary với d
- pop(key) → xóa key & trả về giá trị
- clear() → xóa toàn bộ phần tử

Cấu trúc Dictionary

Key	Value
name	Abhi
age	22
city	Pune

⭐ Chọn đúng loại collection cho đúng công việc!

Lộ trình trang tiếp theo (Trang 9/15)
• Dictionary trong Python

9. Dictionary trong Python

1. Dictionary là gì?

- Dictionary là một tập hợp các cặp key : value.
- Key phải là duy nhất và bất biến (string, number, tuple).
- Value có thể là bất kỳ kiểu dữ liệu nào.
- Dictionary có thể thay đổi (mutable).

Dictionary lưu dữ liệu theo cặp key : value. 🙂

Ví dụ:

```
student = {
  "name" : "Abhi",
  "age"  : 22,
  "city" : "Pune"
}
```

Cấu trúc Key : Value

Key	Value
name	Abhi
age	22
city	Pune

2. Tạo Dictionary

- Dùng dấu ngoặc nhọn { }.
- Dùng hàm khởi tạo dict()

```
d1 = { }
d2 = {"a": 1, "b": 2}
d3 = dict(a=1, b=2)
d4 = dict([("a",1), ("b",2)])
```

3. Truy cập phần tử

- Dùng key
- Dùng phương thức get() (an toàn hơn)

```
d = {"name":"Abhi", "age":22,
"city":"Pune"}
print(d["name"])            # Abhi
print(d.get("age"))         # 22
print(d.get("country", "Not Found"))
                            # Not Found
```

4. Cập nhật Dictionary

- Cập nhật key đã tồn tại
- Thêm key mới

```
d["age"] = 23          # cập nhật
d["course"] = "Python"  # thêm mới
print(d)
```

5. Xóa phần tử

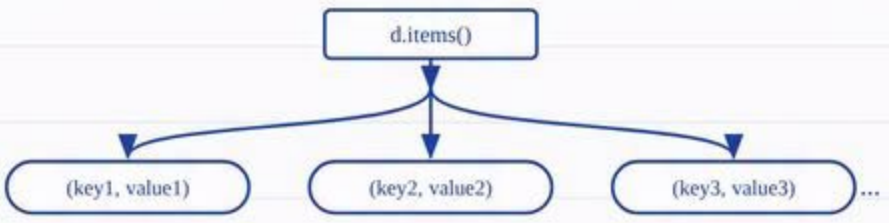
- del → xóa cặp key-value
- pop() → xóa key và trả về value
- popitem() → xóa phần tử được thêm sau cùng
- clear() → xóa toàn bộ phần tử

```
del d["city"]
age = d.pop("age")
item = d.popitem()
d.clear()
```

6. Duyệt Dictionary

- Có thể duyệt bằng keys(), values() hoặc items()

```
for k in d.keys():   print(k)
for v in d.values(): print(v)
for k, v in d.items(): print(k, "→", v)
```



7. Các phương thức Dictionary

keys() → trả về tất cả key
values() → trả về tất cả value
items() → trả về các cặp key-value
get(key) → trả về giá trị theo key
update(d) → cập nhật dictionary với d
pop(key) → xóa key & trả về giá trị
popitem() → xóa phần tử thêm sau cùng
clear() → xóa toàn bộ phần tử
copy() → trả về bản sao dictionary

8. Ghi chú quan trọng

★ Key phải là duy nhất.
★ Dictionary không có thứ tự (trước Python 3.7).
★ Từ Python 3.7+, dictionary giữ nguyên thứ tự chèn (insertion order).
★ Truy cập theo key rất nhanh → trung bình O(1).
★ Hữu ích cho tra cứu nhanh và ánh xạ dữ liệu. 🙂

Chương trình ví dụ

1. Đếm tần suất ký tự

```
s = "abacaba"
freq = {}
for ch in s:
    freq[ch] = freq.get(ch, 0)
+ 1
print(freq)  # {'a':4, 'b':2,
'c':1}
```

2. Điểm học sinh

```
marks = {"Math":95, "Eng":88,
"Sci":90}
total = sum(marks.values())
avg = total / len(marks)
print("Average =", avg)
```

3. Đảo ngược Dictionary

```
d = {"a":1, "b":2, "c":3}
inv = {v:k for k,v in
d.items()}
print(inv)  # {1:'a', 2:'b',
3:'c'}
```

10. File Handling trong Python

File giúp chúng ta lưu trữ dữ liệu vĩnh viễn. 🙂

1. Mở một File

- Ta dùng hàm open() để mở một file.

Cú pháp: open(filename, mode)

Các chế độ (mode) thông dụng:

'r'	Đọc (mặc định)	file phải tồn tại
'w'	Ghi	tạo file mới hoặc ghi đè file cũ
'a'	Nối thêm	thêm dữ liệu vào cuối file
'x'	Tạo mới	tạo file, lỗi nếu đã tồn tại
'b'	Chế độ nhị phân	dùng cùng r, w, a
't'	Chế độ văn bản (mặc định)	dùng cùng r, w, a

2. Đọc từ một File

read() → đọc toàn bộ file
readline() → đọc một dòng
readlines() → đọc tất cả dòng thành list

Ví dụ:

```
f = open("hello.txt", "r")
content = f.read()     # toàn bộ file
print(content)
f.close()
```

★ Đừng quên đóng file bằng phương thức close()! 🙂

3. Ghi vào một File

write() → ghi một chuỗi vào file
writelines() → ghi một list các dòng

Ví dụ:

```
f = open("hello.txt", "w")
f.write("Hello Python!\n")
f.write("File handling is easy.")
f.close()
```

4. Nối thêm vào File

- Thêm dữ liệu mới vào cuối mà không xóa dữ liệu cũ.

Ví dụ:

```
f = open("hello.txt", "a")
f.write("\nThis line is appended.")
f.close()
```

5. Đọc từng dòng

readline() đọc từng dòng một.
readlines() đọc tất cả dòng và trả về list.

Ví dụ:

```
f = open("hello.txt", "r")
line1 = f.readline()
line2 = f.readline()
all_lines = f.readlines()
f.close()
```

6. Dùng câu lệnh with (khuyên dùng)

- Tự động đóng file.
- Gọn gàng và an toàn hơn.

Ví dụ:

```
with open("hello.txt", "r") as f:
    content = f.read()
    print(content)
```

File tự đóng sau khi khối lệnh kết thúc. 🙂

7. Tổng hợp các phương thức File

Phương thức	Mục đích	Dùng cho
read()	Đọc toàn bộ file	File nhỏ
readline()	Đọc một dòng	Đọc từng dòng
readlines()	Đọc tất cả dòng thành list	Xử lý tất cả dòng
write()	Ghi chuỗi vào file	Ghi đè (với 'w')
writelines()	Ghi list các dòng	Ghi nhiều dòng
close()	Đóng file	Giải phóng tài nguyên hệ thống

Chương trình ví dụ:

```
# Tạo, ghi và đọc file
with open("demo.txt", "w") as f:
    f.write("Python is awesome!\n")
    f.write("File handling rocks!")
# Đọc file
with open("demo.txt", "r") as f:
    for line in f:
        print(line.strip())
```

⭐ Luyện tập file handling mỗi ngày - điều này rất cần thiết khi làm việc với dữ liệu thực!

Lộ trình trang tiếp theo (Trang 11/15)

11. Exception Handling trong Python

Exception giúp xử lý lỗi một cách nhẹ nhàng, không làm crash chương trình! 🙂

1. Exception là gì?

- Exception là lỗi xảy ra trong quá trình chương trình thực thi.
- Python cung cấp cách xử lý các lỗi này để chương trình không bị crash.

2. Cú pháp cơ bản

```
try:
    # code có thể gây ra lỗi
except ExceptionType:
    # xử lý lỗi
else:
    # chạy nếu không có lỗi
finally:
    # luôn luôn chạy
```

3. Ví dụ đơn giản

```
try:
    num = int(input("Nhập một số: "))
    result = 10 / num
except ZeroDivisionError:
    print("Không thể chia cho 0!")
else:
    print("Kết quả = ", result)
finally:
    print("Đã thực thi xong.")
```

khối finally luôn luôn chạy! 🙂

4. Else và Finally khi hoạt động

- else chỉ chạy khi khối try không có lỗi.
- finally luôn luôn chạy, dù có lỗi hay không.

```
try:
    x = int(input("Nhập một số: "))
    print("Số là ", x)
except ValueError:
    print("Dữ liệu nhập không hợp lệ!")
else:
    print("Không có lỗi xảy ra.")
finally:
    print("Lệnh này luôn được chạy.")
```

5. Nhiều khối Except

```
try:
    a = int(input("Nhập tử số: "))
    b = int(input("Nhập mẫu số: "))
    print("Kết quả = ", a / b)
except ZeroDivisionError:
    print("Mẫu số không được bằng 0!")
except ValueError:
    print("Vui lòng nhập số nguyên hợp lệ!")
except Exception as e:
    print("Đã xảy ra lỗi khác:", e)
```

6. Các Exception có sẵn thường gặp

Exception	Mô tả
ZeroDivisionError	Xảy ra khi chia cho 0
ValueError	Hàm nhận đối số đúng kiểu nhưng sai giá trị
TypeError	Thao tác/hàm áp dụng sai kiểu dữ liệu
FileNotFoundError	Không tìm thấy file
IndexError	Chỉ số nằm ngoài phạm vi
KeyError	Không tìm thấy key trong dictionary

7. Chủ động ném ra Exception

- Ta có thể chủ động ném ra exception bằng raise.
- Hữu ích để áp đặt điều kiện trong code.

```
def check_age(age):
    if age < 18:
        raise ValueError("Tuổi phải từ 18 trở lên!")
    print("Được phép truy cập")

check_age(16)    # Ném ra ValueError
```

8. Exception tùy chỉnh

- Ta có thể tự tạo exception riêng bằng cách tạo một class mới.
- Class này nên kế thừa từ class Exception.

```
class InvalidAgeError(Exception):
    pass

def validate_age(age):
    if age < 0 or age > 120:
        raise InvalidAgeError("Tuổi không hợp lệ!")

try:
    validate_age(200)
except InvalidAgeError as e:
    print("Custom Exception:", e)
```

⭐ Hãy xử lý exception, đừng bỏ qua chúng.

Lộ trình trang tiếp theo (Trang 12/15)
• Modules trong Python • Import, from ... import

12. Modules trong Python

Module giúp chúng ta tổ chức code thành các file có thể tái sử dụng! 🙂

1. Module là gì?

- Module là một file chứa code Python (hàm, class, biến).
- Ta có thể import và tái sử dụng code từ các module khác.
- Mỗi file Python đều là một module.

📄 mymath.py

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

def sub(a, b):
    return a - b
```

Ta có thể import module này trong file khác.

main.py

```
import mymath
result = mymath.add(5, 3)
print(result)
```

2. Tạo một Module

- Tạo một file Python với đuôi .py
- Viết hàm, class, biến trong đó.

Ví dụ: mymath.py

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

def mul(a, b):
    return a * b

PI = 3.14159
```

↖ File này giờ là một module tên là mymath

3. Import Module

- Ta dùng câu lệnh import.
- Có 2 cách import:

(a) Import cả module

```
import mymath
print(mymath.add(2, 3))
print(mymath.PI)
```

(b) Import mục cụ thể

```
from mymath import add, mul
print(add(2, 3))
print(mul(2, 3))
```

★ Import cụ thể giúp code ngắn gọn và dễ đọc hơn. 🙂

4. Dùng từ khóa 'as'

- Ta có thể dùng as để đặt bí danh (alias) cho module.

```
import mymath as mm
print(mm.add(4, 5))
print(mm.PI)
```

↖ mm là bí danh (alias) của mymath

5. Các Module có sẵn (Built-in)

- Python có rất nhiều module có sẵn.
- Một số module thông dụng:

Module	Chức năng
math	Các hàm toán học
random	Sinh số ngẫu nhiên
datetime	Làm việc với ngày & giờ
os	Tương tác với hệ điều hành
sys	Tham số hệ thống

6. Ví dụ - Dùng Module có sẵn

- Dùng module math

```
import math
print(math.sqrt(16))    # 4.0
print(math.pi)   # 3.141592653589793
```

7. Biến __name__

- Mỗi module Python có một biến đặc biệt __name__.
- Khi module được chạy trực tiếp, __name__ được gán là "__main__".
- Ngược lại, nó được gán bằng tên module.

Ví dụ: mymodule.py

```
def greet():
    print("Hello from module!")

if __name__ == "__main__":
    greet()   # chỉ chạy khi file được
thực thi trực tiếp
```

8. Lợi ích của Module

- ☑ Cải thiện khả năng đọc và tổ chức code.
- ☑ Khuyến khích tái sử dụng code.
- ☑ Giúp quản lý chương trình lớn dễ dàng hơn.
- ☑ Thúc đẩy làm việc nhóm (nhiều lập trình viên có thể làm trên các module khác nhau).

Mẹo
★ Đặt tên module ngắn gọn và có ý nghĩa.
• Tránh đặt tên module trùng với module có sẵn.
• Dùng đường dẫn tương đối hoặc package cho dự án lớn. 🙂

Lộ trình trang tiếp theo (Trang 13/15)

13. Packages trong Python

Package là tập hợp các module trong một cấu trúc thư mục. 🙂

1. Package là gì?

- Package là cách tổ chức các module liên quan trong một thư mục.
- Nó chứa một file đặc biệt tên __init__.py
- Package giúp tái sử dụng code và mở rộng dự án dễ dàng.

Ví dụ cấu trúc Package

```
mypackage/
    ├── __init__.py
    ├── math_utils.py
    └── string_utils.py
main.py
```

↖ mypackage là một package. Nó chứa các module.

2. Tạo một Package

- Tạo một thư mục với tên package.
- Bên trong, thêm một file tên __init__.py (có thể để trống).
- Thêm các file module vào cùng thư mục đó.

Ví dụ:

```
mkdir mypackage
cd mypackage
touch __init__.py     # tạo file trống
touch math_utils.py
touch string_utils.py
```

3. Import từ một Package

- Ta có thể import module hoặc mục cụ thể từ một package.

(a) Import Module

```
from mypackage import math_utils
result = math_utils.add(5, 3)
print(result)
```

(b) Import mục cụ thể

```
from mypackage.math_utils import add
result = add(5, 3)
print(result)
```

★ Luôn dùng toán tử chấm (.) để truy cập module và hàm. 🙂

4. File __init__.py

- File này được thực thi khi package được import.
- Ta có thể dùng nó để khởi tạo package.
- Nó cũng có thể chứa biến cấp package hoặc các câu lệnh import.

Ví dụ: __init__.py

```
print("Initializing mypackage...")
__version__ = "1.0.0"
from .math_utils import add
```

↖ Được thực thi khi package được import.

5. Import tương đối và tuyệt đối

Import tuyệt đối	Import tương đối
from mypackage import math_utils	from . import math_utils
✓ Bắt đầu từ package cấp cao nhất.	✓ Bắt đầu từ package hiện tại.
✓ Rõ ràng và được khuyên dùng.	✓ Dùng bên trong package.

Dùng import tuyệt đối trong chương trình chính.
Import tương đối hữu ích khi dùng bên trong package. 🙂

6. Package có sẵn / thư viện chuẩn

- Python đi kèm với rất nhiều package có sẵn trong thư viện chuẩn.
- Một số hữu ích thường dùng:

★ math - Các hàm toán học
★ random - Sinh số ngẫu nhiên
★ datetime - Làm việc với ngày & giờ
★ os - Tương tác hệ điều hành
★ sys - Tham số hệ thống
★ json - Làm việc với dữ liệu JSON

Không cần tạo các module này, chúng đã sẵn sàng để dùng! 🙂

7. Cài đặt Package bên ngoài

- Package bên ngoài không được tích hợp sẵn trong Python.
- Cài đặt bằng pip.

```
pip install package_name       # cài bản mới nhất
pip install package_name==1.2.0  # phiên bản cụ thể
```

Ví dụ:

```
pip install requests
pip install numpy==1.24.0
```

8. Liệt kê Package đã cài

- Xem tất cả package đã cài bằng pip.

```
pip list      # liệt kê package đã cài
pip freeze    # liệt kê kèm phiên bản
```

Gỡ cài đặt một package:

```
pip uninstall package_name
```

Lộ trình trang tiếp theo (Trang 14/15)

14. Virtual Environments trong Python

Virtual environment giúp các dự án độc lập và tránh xung đột thư viện! 🙂

1. Virtual Environment là gì?

- Virtual environment (venv) là một môi trường Python biệt lập.
- Nó có trình thông dịch Python và các package riêng.
- Giúp tránh xung đột phụ thuộc (dependency) giữa các dự án.
- Giữ cho bản cài đặt Python toàn cục luôn sạch.

Hãy hình dung nó như một không gian làm việc riêng cho dự án của bạn. 💻

2. Tại sao dùng Virtual Environment?

★ Các dự án khác nhau có thể cần phiên bản khác nhau của cùng một package.

★ Ngăn ngừa lỗi "chạy được trên máy tôi nhưng không chạy trên máy khác".

★ Dễ dàng chia sẻ yêu cầu (requirements) của dự án.

★ Giữ Python hệ thống sạch & an toàn.

★ Là thói quen tốt cho công việc phát triển chuyên nghiệp. 🙂

3. Tạo Virtual Environment

- Python có sẵn module venv.
- Tạo một virtual environment mới bằng:

```
# Windows
python -m venv myenv

# macOS / Linux
python3 -m venv myenv
```

↖ Lệnh này tạo một thư mục "myenv" với Python và package riêng.

4. Kích hoạt Virtual Environment

- Kích hoạt trước khi sử dụng.

Windows (Command Prompt)

```
myenv\Scripts\activate
```

macOS / Linux (Terminal)

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

★ Sau khi kích hoạt, dấu nhắc lệnh của bạn sẽ trông như:
(myenv) C:\Users\Abhi> (myenv) abhi@mac ~ $

5. Tắt Environment

- Khi xong, hãy tắt nó:

```
deactivate
```

↖ Hoạt động trên mọi nền tảng.

6. Cài Package trong Environment

- Tất cả package cài lúc này sẽ nằm trong environment này.
- Dùng pip như bình thường.

```
pip install numpy
pip install requests
```

↖ Chỉ được cài trong virtual environment này.

7. Xem Package đã cài

- Liệt kê tất cả package trong environment hiện tại.

```
pip list
```

↖ Hiển thị tất cả package đã cài.

Hoặc xem chi tiết hơn:

```
pip freeze
```

↖ Hiển thị phiên bản chính xác. Hữu ích khi chia sẻ.

8. Lưu Dependencies

- Lưu các package đã cài vào file requirements.

```
pip freeze > requirements.txt
```

↖ Tạo một file chứa tất cả package và phiên bản.

9. Cài đặt từ File Requirements

- Tái tạo cùng environment trên máy khác.

```
pip install -r requirements.txt
```

↖ Cài tất cả package được liệt kê trong file.

10. Thực hành tốt nhất

☑ Luôn dùng virtual environment cho mọi dự án.

☑ Đừng cài package vào môi trường toàn cục.

☑ Thêm .venv/ hoặc myenv/ vào .gitignore.

☑ Chia sẻ requirements.txt, không chia sẻ thư mục environment.

☑ Cập nhật dependencies thường xuyên.

Lộ trình trang tiếp theo (Trang 15/15)

15. Tổng Kết & Ghi Chú Cuối

Bạn đã hoàn thành! Hãy tiếp tục luyện tập và xây dựng dự án nhé. 🙂

1. File Handling - Điểm chính

- Python giúp việc xử lý file trở nên đơn giản và mạnh mẽ.
- Luôn đóng file (dùng with open() để an toàn).
- Chọn đúng chế độ (mode) cho từng công việc.
- Xử lý exception để code chạy ổn định.

★ Quy tắc vàng
Luôn đóng file.
Một file để mở có thể gây ra vấn đề! 🙂

2. Exception Handling - Điểm chính

- Exception giúp chương trình xử lý lỗi một cách nhẹ nhàng.
- Dùng try - except để bắt và xử lý lỗi.
- Dùng else và finally để điều khiển luồng chạy.
- Ném ra exception tùy chỉnh để kiểm tra dữ liệu tốt hơn.

💡 Ghi nhớ
Một chương trình tốt không tránh lỗi,
nó xử lý lỗi một cách khôn ngoan. 🙂

3. Modules - Điểm chính

- Module giúp code gọn gàng và có thể tái sử dụng.
- Chỉ import những gì bạn cần.
- Ưu tiên dùng module có sẵn trước khi tự viết.
- Tuân theo cấu trúc rõ ràng và đặt tên có ý nghĩa.

Module có sẵn thông dụng	
math - Hàm toán học	sys - Tham số hệ thống
random - Số ngẫu nhiên	json - Xử lý JSON
datetime - Ngày & giờ	re - Regular expressions
os - Tương tác hệ điều hành	collections - Container đặc biệt

4. Virtual Environments - Điểm chính

- Cô lập dependency cho từng dự án.
- Ngăn lỗi "chạy được trên máy tôi".
- Giữ bản cài đặt Python toàn cục sạch sẽ.
- Dễ dàng tạo, kích hoạt và chia sẻ.

Quy trình cơ bản

Tạo venv → Kích hoạt venv → Cài package → Làm việc trên dự án

5. Tham khảo nhanh - Các lệnh

Tác vụ	Lệnh / Code
Mở file	open("file.txt", "r")
Đọc toàn bộ nội dung	f.read()
Ghi vào file	f.write("text")
Nối thêm vào file	open("file.txt", "a")
Xử lý exception	try: ... except Exception as e: ...
Import module	import module_name
Import mục cụ thể	from module import item
Tạo venv (Windows)	python -m venv myenv
Kích hoạt venv (Win)	myenv\Scripts\activate
Kích hoạt venv (macOS/Linux)	source myenv/bin/activate
Cài package	pip install package_name

6. Checklist thực hành tốt nhất

- ☑ Luôn dùng with open() khi xử lý file.
- ☑ Xử lý các exception có thể xảy ra trong code.
- ☑ Giữ hàm và module nhỏ gọn, tập trung.
- ☑ Dùng tên biến, hàm, module có ý nghĩa.
- ☑ Dùng virtual environment cho mọi dự án.
- ☑ Cài chỉ những package bạn cần.
- ☑ Cập nhật dependencies thường xuyên.
- ☑ Viết code sạch, dễ đọc và có chú thích.
- ☑ Kiểm thử code với nhiều tình huống khác nhau.
- ☑ Tiếp tục học và xây dựng dự án! 🙂

🏆

Chúc mừng!

Bạn đã hoàn thành Cẩm nang Python.
Hãy luyện tập mỗi ngày, xây dựng dự án và áp dụng những gì đã học.

Tiếp theo là gì?

- Xây dựng các dự án thực tế
- Khám phá các chủ đề Python nâng cao
- Đóng góp cho mã nguồn mở (open source)