CHƯƠNG 2: BIẾN, BIỂU THỨC & CÂU LỆNH

1.HẰNG SỐ – BIẾN – TÊN BIẾN

-Hằng số (constant): giá trị cố định (123, 98.6, "Hello")

-Biến (variable): nơi lưu trữ dữ liệu, có thể thay đổi.

-Tên biến hợp lệ: chữ, số, _ ; không bắt đầu bằng số; phân biệt hoa-thường.

-Ví dụ tốt: spam, total_sum, _index

Không hợp lệ: 23spam, a.b, #x

2. TỪ KHÓA (reserved words)

Không dùng làm tên biến:and, or, not, if, else, for, while, break, return, def, class, import,…

3. CÂU LỆNH GÁN (Assignment)

biến = biểu_thức

Python tính bên phải trước, gán kết quả vào biến bên trái.

```
x = 3.9 * x * (1 - x)
```

4. BIỂU THỨC SỐ HỌC

Toán tử:

+ cộng

- trừ

* nhân

/ chia (float)

% chia dư

** lũy thừa

- Thứ tự ưu tiên:

()

*

**,/,%

+, -

5. KIỂU DỮ LIỆU

int → số nguyên

float → số thực

str → chuỗi

bool → True / False

list, tuple → cấu trúc dữ liệu

Python là dynamic typing:

```
a = 5
a = "Hello"
```

6. ÉP KIỂU

(type conversion)int("123") → 123

float("99.5") → 99.5

Lỗi nếu chuỗi không phải số.

7. NHẬP DỮ LIỆU

(input)(Python 2: raw_input() — Python 3: input())

```
inp = input("Europe floor? ")
us = int(inp) + 1
```

8. COMMENT (chú thích)

Dùng dấu #, Python bỏ qua phần sau dấu này.

```
# This is a comment
```

9. TOÁN TỬ CHUỖI

"abc" + "123" → "abc123“

"Hi" * 5 → "HiHiHiHiHi"

10. ĐẶT TÊN BIẾN MNEMONIC (gợi nhớ)

-Tốt:

```
hours = 35
rate = 12.5
pay = hours * rate
```

-Không nên dùng: x1q3z9, a, b, …

1.Hàm là gì?

-Một đoạn mã có thể tái sử dụng, thực hiện một nhiệm vụ.
-Có 2 loại:
Built-in functions: print(), input(), type(), int(), max(), min()
User-defined functions: lập trình viên tự khai báo.

2. Cú pháp định nghĩa hàm

```
def tên_hàm(tham_số):
    # thân hàm (thụt lề)
    câu_lệnh
```

Dùng từ khóa def.
Không chạy thân hàm ngay, chỉ định nghĩa.
Gọi hàm bằng: tên_hàm().

3. Gọi hàm (Function Call)

```
thing()
print("Hello")
```

Khi gọi hàm → mã trong hàm sẽ chạy.
Có thể gọi nhiều lần (tái sử dụng).

4. Tham số (parameters) & Đối số (arguments)

Tham số: biến trong khai báo hàm
Đối số: giá trị truyền vào khi gọi hàm

```
def greet(lang):  # parameter
    ...

greet('es')  # argument
```

5. Hàm có giá trị trả về (fruitful function)

Dùng return để trả kết quả.

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

x = add(3, 5)  # x = 8
```

return kết thúc hàm.

6. Hàm không trả về (void function)

Không dùng return → trả về None.

```
def say():
    print("Hello")
```

7. Lợi ích khi dùng hàm

Chia nhỏ chương trình → dễ đọc & dễ hiểu
Tránh lặp code (Don't Repeat Yourself – DRY)
Dễ quản lý, sửa lỗi, mở rộng
Có thể tạo thư viện hàm để tái dùng

Hàm xử lý bằng điều kiện

```
def greet(lang):
    if lang == 'es':
        return 'Hola'
    elif lang == 'fr':
        return 'Bonjour'
    else:
        return 'Hello'
```

Type Conversion trong hàm

Chuyển kiểu:
int()
float()
str()
Chuỗi không phải số → lỗi khi chuyển bằng int()

```
int("123")  # OK
int("hello")  # lỗi
```

1.Boolean & Comparison Operators (Toán tử so sánh)

Dùng để tạo biểu thức điều kiện (True/False).

Toán tử	Ý nghĩa
`<`	nhỏ hơn
`<=`	nhỏ hơn hoặc bằng
`>`	lớn hơn
`>=`	lớn hơn hoặc bằng
`==`	bằng
`!=`	khác

2. Câu lệnh if – One-way decision

Chạy một khối lệnh nếu điều kiện đúng.

```
x = 5
if x < 10:
    print("Smaller")
print("Finis")
```

3. Indentation (thụt lề trong Python)

Khối lệnh sau : phải thụt vào (4 spaces).

Không được dùng tabs lẫn với spaces, dễ gây lỗi.

Python quyết định phạm vi code dựa vào thụt lề.

4. Two-way decision – if/else

Chọn 1 trong 2 nhánh.

```
x = 4
if x > 2:
    print("Bigger")
else:
    print("Smaller")
print("Done")
```

5. Multi-way decision – if/elif/else

Dùng nhiều điều kiện:

```
if x < 2:
    print("small")
elif x < 10:
    print("medium")
else:
    print("large")
```

6. Nested decisions – If lồng nhau

```
if x > 1:
    if x < 100:
        print("Between 1 and 100")
```

7. Các lỗi Multi-way puzzles thường gặp

✕ Lỗi logic do sắp xếp sai thứ tự:

```
if x < 2:
    print("Below 2")
elif x < 20:
    print("Below 20")
elif x < 10:  # điều này sẽ KHÔNG BAO GIỜ chạy
    print("Below 10")
```

Lý do: mọi số < 10 đều đã bị bắt bởi x < 20.

8. try / except – xử lý lỗi

Dùng khi đoạn code "nguy hiểm" có thể lỗi (như ép kiểu).

```
try:
    value = int(text)
except:
    value = -1
```

CHƯƠNG 5 - ITERATION

1. Cập nhật biến (Updating Variables)
Trước khi lặp, ta thường cần khởi tạo biến.

```
x = 0
x = x + 1  # Tăng giá trị x lên 1 (Increment)
```

2. Vòng lặp while (While Loops)
Chạy liên tục chừng nào điều kiện còn là True.
-Cấu trúc cơ bản:

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Hết giờ!')
```

-Vòng lặp vô tận (Infinite Loop): Xảy ra nếu điều kiện luôn **True** và không có lệnh dừng.
-break: Nhảy ra khỏi vòng lặp ngay lập tức.
-continue: Bỏ qua phần còn lại của thân vòng lặp hiện tại, quay lại kiểm tra điều kiện.

3. Vòng lặp for (Definite Loops)
Dùng để duyệt qua một tập hợp (list, chuỗi, file) đã biết trước số lượng phần tử.

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends:
    print('Happy New Year:', friend)
# friend là biến chạy (iteration variable)
```

4. Các mẫu vòng lặp thông dụng (Loop Idioms)

a. Đếm số lượng (Counting)

```
count = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    count = count + 1
print('Tổng số phần tử:', count)
```

b. Tính tổng (Summing)

```
total = 0
for itervar in [3, 41, 12, 9, 74, 15]:
    total = total + itervar
print('Tổng cộng:', total)
```

c. Tìm số lớn nhất (Finding the Largest)

```
largest = -1
for the_num in [9, 41, 12, 3, 74, 15]:
    if the_num > largest:
        largest = the_num
print('Số lớn nhất:', largest)
```

d. Tìm số nhỏ nhất (Finding the Smallest)
Lưu ý: Dùng **None** (giá trị rỗng) để khởi tạo cờ hiệu (flag).

```
smallest = None
for value in [9, 41, 12, 3, 74, 15]:
    if smallest is None:
        smallest = value  # Gán giá trị đầu tiên làm mốc
    elif value < smallest:
        smallest = value
print('Số nhỏ nhất:', smallest)
```

5. is và is not:
- **is** Kiểm tra xem hai biến có trỏ đến cùng một đối tượng trong bộ nhớ không (mạnh hơn **==**).
- Khuyên dùng is chỉ cho các giá trị **Boolean** hoặc **None.**
- **0 == 0.0** là **True**, nhưng **0 is 0.0** là **False**.

CHƯƠNG 6 - STRINGS

1. Cấu trúc chuỗi (String Anatomy)

Chuỗi là một dãy các ký tự.

```
fruit = 'banana'
letter = fruit[1]  # Kết quả: 'a'
```

-Index (Chỉ số): Bắt đầu từ **0**.
-Lỗi: Truy cập chỉ số vượt quá độ dài chuỗi sẽ báo lỗi.
-Độ dài: **len(fruit)** trả về 6.

2. Duyệt chuỗi (Traversal)

Đi qua từng ký tự trong chuỗi.

```
# Cách 1: Dùng while (cần quản lý index)
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(letter)
    index = index + 1

# Cách 2: Dùng for (Pythonic - Khuyên dùng)
for char in fruit:
    print(char)
```

3. Cắt chuỗi (Slicing)

Lấy một đoạn con của chuỗi.
Cú pháp: **s[start:end]** (lấy từ *start* đến sát *end*, không lấy *end*).

```
s = 'Monty Python'
print(s[0:4])   # 'Mont'
print(s[6:7])   # 'P'
print(s[6:20])  # 'Python' (không lỗi dù index quá giới hạn)
print(s[:2])    # 'Mo' (mặc định bắt đầu từ 0)
print(s[8:])    # 'thon' (mặc định đến hết chuỗi)
print(s[:])     # 'Monty Python' (toàn bộ chuỗi)
```

4. Thao tác chuỗi

Nối chuỗi (+): **'a' + 'b'** -> **'ab'**
Toán tử **in**: Kiểm tra sự tồn tại.

```
'n' in 'banana'      # True
'nan' in 'banana'    # True
```

So sánh chuỗi: Dựa trên thứ tự bảng chữ cái (Lưu ý: Chữ Hoa luôn nhỏ hơn chữ thường trong máy tính).

5. Thư viện String (String Methods)

Chuỗi là đối tượng có sẵn các phương thức. **str.method()**. Lưu ý: Chuỗi là bất biến (Immutable), các hàm này trả về chuỗi mới, không đổi chuỗi gốc.
type(x): Kiểm tra kiểu dữ liệu.
dir(x): Xem tất cả phương thức có sẵn.Các phương thức phổ biến:

```
greet = 'Hello Bob'
zap = greet.lower()          # 'hello bob'
print(greet.upper())         # 'HELLO BOB'
print(greet.replace('Bob','Jane')) # 'Hello Jane'
print(greet.replace('o','X'))      # 'HellX BXb'

# Loại bỏ khoảng trắng (trắng, tab, xuống dòng)
line = '   Hello   '
print(line.strip())          # 'Hello' (cắt 2 đầu)
print(line.lstrip())         # Cắt bên trái
print(line.rstrip())         # Cắt bên phải (rất hay dùng khi đọc file)

# Tiền tố
line.startswith('He')        # True
```

6. Tìm kiếm và Trích xuất (Parsing/Extracting)Ví dụ tìm vị trí và cắt chuỗi: data = 'From stephen.marquard@uct.ac.za Sat Jan 5'

```
atpos = data.find('@')           # Tìm vị trí '@' -> 21
sppos = data.find(' ', atpos)    # Tìm khoảng trắng đầu tiên SAU vị trí atpos -> 31
host = data[atpos+1 : sppos]     # Cắt từ sau @ đến trước khoảng trắng
print(host)                      # uct.ac.za
```

CHƯƠNG 7 - FILES

1. Khái niệm cơ bản (File Handle)

File text được xem như một chuỗi các dòng, mỗi dòng kết thúc bằng ký tự xuống dòng **\n** (newline).
open(): Hàm mở file, trả về một "handle" (tay cầm) để thao tác, không chứa toàn bộ dữ liệu file.

```
fhand = open('mbox.txt', 'r')
# 'r': read (đọc - mặc định)
# 'w': write (ghi - xóa nội dung cũ nếu có)
```

2. Ký tự xuống dòng (Newline)

\n được tính là 1 ký tự.

```
stuff = 'Hello\nWorld!'
print(len(stuff))  # 12 ký tự
```

3. Đọc toàn bộ file (Reading the whole file)

Dùng khi file nhỏ, có thể chứa hết trong RAM.

```
fhand = open('mbox-short.txt')
inp = fhand.read()
print(len(inp))      # In ra tổng số ký tự
print(inp[:20])      # In ra 20 ký tự đầu
```

4. Duyệt file từng dòng (Searching through a file)

Đây là cách phổ biến nhất, tiết kiệm bộ nhớ cho file lớn. Vòng lặp **for** coi file handle như một danh sách các dòng.
Mẫu 1: In dòng có điều kiện Lưu ý: Hàm **print()** tự xuống dòng, và trong file cũng có **\n**, nên cần dùng **rstrip()** để tránh bị dòng trống.

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip() # Loại bỏ \n ở cuối
    if line.startswith('From:'):
        print(line)
```

Mẫu 2: Bỏ qua dòng (Skipping lines with continue)

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From:'):
        continue # Bỏ qua, quay lại đầu vòng lặp
    print(line) # Chỉ in dòng bắt đầu bằng From:
```

Mẫu 3: Sử dụng toán tử in để tìm kiếm

```
for line in fhand:
    line = line.rstrip()
    if '@uct.ac.za' in line:
        print(line)
```

5. Xử lý tên file người dùng nhập (User Input & Try/Except)

Người dùng có thể nhập sai tên file, cần dùng **try/except** để tránh chương trình bị "sập" (traceback).

```
fname = input('Nhập tên file: ')
try:
    fhand = open(fname)
except:
    print('File không tồn tại:', fname)
    quit() # Thoát chương trình an toàn

count = 0
for line in fhand:
    if line.startswith('Subject:'):
        count = count + 1
print('Có', count, 'dòng Subject trong file', fname)
```

6. Ghi file (Writing files)

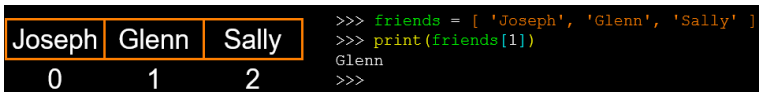
Mở với mode **'w'**. Cảnh báo: Nếu file đã tồn tại, nó sẽ bị xóa trắng ngay lập tức.
Phương thức **.write()** không tự động thêm xuống dòng, bạn phải tự thêm **\n**.

```
fout = open('output.txt', 'w')
line1 = "Dòng này nằm ở đây,\n"
fout.write(line1)
line2 = "Dòng này nằm ở dưới.\n"
fout.write(line2)
fout.close() # Rất quan trọng: Phải đóng file để lưu dữ liệu vào ổ cứng
```

List là một danh sách các giá trị có thể là bất kì các phần tử nào trong Python, trong đó ta có thể dùng nhiều thao tác như: thêm, bớt, tham chiếu, thay đổi giá trị,…

Mỗi giá trị trong List đều có 1 chỉ số Index, tất cả đều bắt đầu từ 0, ta có thể dùng chỉ số này để làm các việc như đã nói ở trên

Ví dụ tham chiếu



Thay đổi 1 phần tử trong List

```
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

Ta có thể dùng hàm range để trả về 1 dãy các chỉ số (Index) của List thể thuận tiện dùng cho vòng lặp,…

```
>>> print(range(4))
[0, 1, 2, 3]
>>> friends = ['Joseph', 'Glenn', 'Sally']
>>> print(len(friends))
3
>>> print(range(len(friends)))
[0, 1, 2]
```

Tách List bằng cú pháp "chỉ số đầu : chỉ số cuối"

- Lưu ý: cắt từ "chỉ số đầu" đến "chỉ số cuối" – 1
- Thiếu chỉ số nào thì Python sẽ hiểu là lấy hết phần tử theo hướng chỉ số đó

```
>>> t = [9, 41, 12, 3, 74, 15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

Hàm min, max, sum dùng để xác định giá trị lớn nhất, nhỏ nhất và tổng các giá trị trong List

- Lưu ý chỉ có tác dụng khi các giá trị là number

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

Tương tự Strings, List cũng dùng hàm len() để trả về độ dài

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>>
```

Nối List bằng toàn tử "+"

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

Vài lệnh thường dùng cho List:

'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort'

Toán tử logic True, False để kiểm tra có hay không giá trị đó trong List

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
```

Cắt String thành List ta dùng hàm split(), đây là hàm cắt chuỗi và loại bỏ tất cả dấu "space" nếu có

Và ta cần nhập điều kiện để cắt vào trong () nếu không muốn hàm bị nhầm lẫn

```
>>> line = 'A lot                of spaces'
>>> etc = line.split()
>>> print(etc)
['A', 'lot', 'of', 'spaces']
>>>
>>> line = 'first:second:third'
>>> thing = line.split()
>>> print(thing)
['first:second:third']
>>> print(len(thing))
1
>>> thing = line.split(':')
>>> print(thing)
['first', 'second', 'third']
>>> print(len(thing))
3
```

- Numpy là một thư viện lõi phục vụ cho khoa học máy tính của Python, hỗ trợ cho việc tính toán các mảng nhiều chiều, có kích thước lớn với các hàm đã được tối ưu áp dụng lên các mảng nhiều chiều đó. Numpy đặc biệt hữu ích khi thực hiện các hàm liên quan tới Đại Số Tuyến Tính.

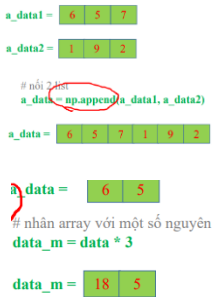
- Khai báo thư viện numpy: import numpy as np

array_name = np.array([element-1, …, element-n])

- Cú pháp: as np có thể có hay không tùy ý, đây là cú pháp rút gọn tên thư viện cho dễ dùng

- Chỉ số tham chiếu (Index), các hàm len(), min(), max() của List và Numpy Array giống nhau nên ta sẽ lượt bỏ phần này

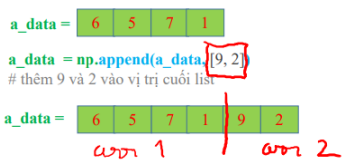
- Toán tử "+" và "*"



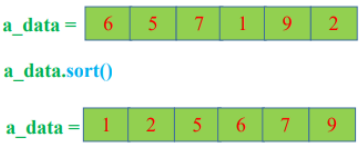
- Thêm giá trị vào mảng np

- Thay đổi giá trị trong np

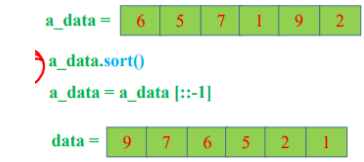
- Thêm 1 mảng giá trị



- Sắp xếp các phần tử tăng dần

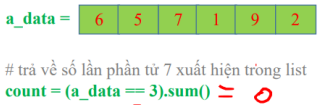


- Sắp xếp các phần tử giảm dần



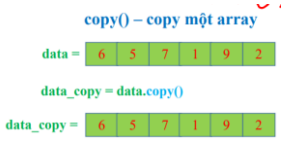
- Xóa 1 phần tử

- Đếm số lần phần tử xuất hiện




```
# trả về số lần phần tử 7 xuất hiện trong list
count = (a_data == 3).sum()
```

- Copy 1 mảng



- Nối 2 mảng khác biến được gán

```
a = np.array([1, 2])
b = np.array([3, 4])
print(np.concatenate((a, b)))  # [1 2 3 4]
```

- Liệt kê phần tử

```
import numpy as np

# Example arrays
arr1 = np.array([6, 1, 7])

for index,value in np.ndenumerate(arr1):
    print(index,value)
```

Output

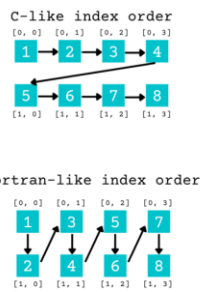
```
(0,) 6
(1,) 1
(2,) 7
```

- Thay đổi hình dạng mảng

```
Python
>>> import numpy as np
>>> numbers = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> numbers
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> numbers.shape
(8,)
>>> numbers.ndim
1

>>> numbers.reshape((2, 4), order="C")
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])

>>> numbers.reshape((2, 4), order="F")
array([[1, 3, 5, 7],
       [2, 4, 6, 8]])
```



- Nối 2 mảng chung biến được gán

```
arr = np.array([[1, 2], [3, 4]])
print(arr.flatten())  # [1 2 3 4]
```

- Cộng mảng 1 chiều với đa chiều

```
import numpy as np
a = np.array([[1, 2], [3, 4], [5, 6]])
b = np.array([1, 2])

# Broadcasting in action: adding the 1D array 'b' to each row of the 2D array
result = a + b
# Outputs:
# [[2 4]
#  [4 6]
#  [6 8]]
```

- Nhân 2 mảng cùng chiều

```
import numpy as np
a = np.array([1, 2, 3, 4])
b = np.array([5, 6, 7, 8])

# Vectorized multiplication of the two arrays
result = a * b
# Outputs: [5 12 21 32]
```

- Mảng 2D

```
import numpy as np

# Create a 2D NumPy array
array_2d = np.array([
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
])

# Accessing elements in a 2D NumPy array
print("\n2D NumPy Array:")
print(array_2d)
```

Output

```
2D NumPy Array:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

- Nhân ma trận

```
import numpy as np

# Define the matrices
A = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

B = np.array([
    [7, 8],
    [9, 10],
    [11, 12]
])

# Multiply the matrices using np.dot
result = np.dot(A, B)

# Alternatively, you can use the @ operator
# result = A @ B

# Print the result
print(result)
```

- Hình dạng (loại) mảng

```
Python
>>> import numpy as np
>>> numbers = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
>>> numbers
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
```

```
>>> numbers.shape
(2, 4)
```

```
Python
>>> numbers.ndim
2
```