Python cho Khoa học dữ liệu

Bài 02: Các Khái Niệm Cơ Bản

Hà Minh Tuấn

Khoa Toán - Tin học
Trường Đại học Khoa học Tự nhiên, ĐHQG-HCM

Ngày 28 tháng 8 năm 2025

Nội dung bài học

Câu lệnh và Chương trình "Hello, World!"

- Một **câu lệnh (statement)** là một hướng dẫn mà trình thông dịch Python có thể thực hiện.
- Các câu lệnh thường kết thúc bằng một dòng mới, **không cần dấu chấm phẩy (;)**.
- **Hàm `print()`:** Là một hàm tích hợp sẵn được sử dụng để hiển thị đầu ra ra màn hình.

Chương trình Python đầu tiên: "Hello, World!"

```
print("Hello, World!")
```

Giải thích

Hàm `print()` sẽ hiển thị chuỗi ký tự "Hello, World!"ra màn hình.

Giới thiệu về Cú pháp Python

- Python là một ngôn ngữ lập trình rất phổ biến, **dễ học và dễ đọc**.
- Có ít từ khóa, cấu trúc đơn giản và một cú pháp được định nghĩa rõ ràng.
- Mã nguồn của Python khá **dễ bảo trì**.
- Triết lý thiết kế nhấn mạnh **khả năng đọc mã** bằng cách sử dụng thụt lề có ý nghĩa (significant indentation).
- Python là một ngôn ngữ **thông dịch (interpreted)**, không cần biên dịch trước khi thực thi.

Indentation (Thụt lề) – Nền tảng Cú pháp

Thụt lề là một yếu tố cú pháp **quan trọng bậc nhất** trong Python.

- **Xác định các khối mã:** Thay vì dùng dấu ngoặc nhọn "{}", Python dùng thụt lề để nhóm các câu lệnh.
- **Tính nhất quán là chìa khóa:** Tất cả các câu lệnh trong cùng một khối phải có cùng một mức thụt lề (thường là **4 dấu cách**).

Ví dụ về Indentation (Thụt lề)

```
if True:
    print("Dòng này sẽ được in ra.") # Dòng này thụt lề 4 dấu
    ↪ cách
    print("Đây là một phần của khối 'if'.") # Dòng này cũng
    ↪ thụt lề 4 dấu cách
print("Dòng này không thụt lề, thuộc khối ngoài.")
```

Comments (Chú thích)

Mục đích của Chú thích

- **Giải thích mã:** Giúp chương trình dễ hiểu hơn cho bạn và những người khác.
- **Vô hiệu hóa mã tạm thời:** Tạm thời bỏ qua một đoạn mã mà không xóa nó.

Cú pháp

Bắt đầu một dòng chú thích bằng dấu thăng (#). Trình thông dịch Python sẽ bỏ qua mọi thứ sau dấu # trên cùng một dòng.

```
## Đây là một chú thích đơn dòng, giải thích mã bên dưới
print("Học cú pháp Python!") # Chú thích này nằm ở cuối dòng
↪  mã

 # print("Dòng này bị chú thích và sẽ không chạy.")
```

Python Identifiers

Định nghĩa: Tên dùng để nhận diện biến (variable), hàm (function), lớp (class), module, hoặc đối tượng khác.

Quy tắc cơ bản:

- Bắt đầu bằng chữ cái (A–Z, a–z) hoặc _ , theo sau có thể là chữ, số, _.
- Không chứa ký tự đặc biệt (@, $, % ...).
- Phân biệt chữ hoa và chữ thường ("Manpower" $\neq$ "manpower").

Quy ước đặt tên:

- Tên lớp: chữ cái đầu viết hoa.
- Các định danh khác: bắt đầu chữ thường.
- _name : định danh riêng tư.
- __name : định danh riêng tư mạnh.
- __name__ : tên đặc biệt do ngôn ngữ định nghĩa.

Python Reserved Words

Định nghĩa: Các từ khóa trong Python là từ dành riêng, không được dùng làm tên biến, hằng số hay định danh khác. Tất cả đều viết thường (trừ `True`, `False`, `None`).

Danh sách từ khóa:

- and, as, assert
- break, class, continue
- def, del, elif
- else, except, False
- finally, for, from
- global, if, import
- in, is, lambda
- None, nonlocal, not
- or, pass, raise
- return, True, try
- while, with, yield

Python Multi-Line Statements

Quy tắc:

- Mặc định: Mỗi câu lệnh kết thúc bằng dấu xuống dòng.
- Có thể dùng ký tự \ để nối dòng.

Ví dụ với \:

```
total = item_one + \
        item_two + \
        item_three
```

Ngoại lệ: Bên trong [ ], { }, ( ) không cần \.

Ví dụ:

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotations in Python

Quy tắc:

- Python chấp nhận: dấu nháy đơn ' ', nháy kép "", và nháy ba ''' ''' hoặc """""".
- Phải bắt đầu và kết thúc bằng cùng một loại dấu.
- Nháy ba dùng cho chuỗi nhiều dòng.

Ví dụ:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph.
It spans multiple lines."""
```

Multiple Statements on a Single Line

Quy tắc:

- Dùng dấu chấm phẩy ; để viết nhiều câu lệnh trên cùng một dòng.
- Chỉ áp dụng khi không bắt đầu một khối lệnh mới (như `if`, `for`, ...).

Ví dụ:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Multiple Statement Groups as Suites

- Một nhóm nhiều câu lệnh tạo thành một **suite** (khối lệnh).
- Các câu lệnh phức hợp (`if`, `while`, `def`, `class`) cần:
  - **Header line**: bắt đầu bằng từ khóa, kết thúc bằng dấu `:`.
  - **Suite**: một hoặc nhiều dòng thụt lề sau header.

Ví dụ:

```
if expression:
    suite
elif expression:
    suite
else:
    suite
```

Command Line Arguments in Python

Khái niệm:

- Python hỗ trợ chạy chương trình với **tham số dòng lệnh**.
- Có thể dùng `-h` để xem hướng dẫn cách chạy.

Ví dụ:

```
 python3 -h
usage: python3 [option] ... [-c cmd | -m mod | file | -] [arg]
↪   ...

-c cmd : chạy lệnh từ chuỗi
-d     : bật debug
-E     : bỏ qua biến môi trường
-h     : in hướng dẫn và thoát
```

Ghi chú: Có thể lập trình script để chấp nhận nhiều tùy chọn khác nhau. Đây là nội dung nâng cao, nên học sau khi nắm vững các khái niệm Python cơ bản.

Python Variables

Khái niệm:

- Biến trong Python là vị trí bộ nhớ được dùng để lưu trữ giá trị.
- Khi tạo biến, Python cấp phát bộ nhớ tùy theo kiểu dữ liệu.

Đặc điểm:

- Có thể lưu số nguyên, số thực, chuỗi ký tự,...
- Dữ liệu và địa chỉ bộ nhớ đều được lưu dưới dạng nhị phân.

Biến và Địa chỉ trong Python

Quy trình:

- Trình biên dịch/phiên dịch (Python interpreter) chuyển đổi dữ liệu sang mã máy.
- Đối tượng được lưu ở vị trí bộ nhớ ngẫu nhiên.
- Hàm `id()` trả về địa chỉ bộ nhớ của đối tượng.

```
>>> id(18)
140714055169352
```

Creating Python Variables

Khái niệm:

- Python không cần khai báo tường minh để tạo biến.
- Biến được tạo tự động khi gán giá trị bằng toán tử =.
- Toán hạng bên trái = là tên biến, bên phải là giá trị.

Ví dụ:

```
counter = 100          # Integer variable
miles   = 1000.0       # Floating point variable
name    = "Zara Ali" # String variable
```

Print và Delete Python Variables

In biến:

```
counter = 100
miles   = 1000.0
name    = "Zara Ali"

print(counter)  # 100
print(miles)    # 1000.0
print(name)     # Zara Ali
```

Xoá biến với `del`:

```
counter = 100
print(counter)   # 100

del counter
print(counter)   # Lỗi: NameError
```

Python Variables: Casting, Case Sensitivity & Multiple Assignment

Casting: chỉ định kiểu dữ liệu cho biến.

```
x = str(10)     # '10'
y = int(10)     # 10
z = float(10)   # 10.0
print(x, y, z)  # 10 10 10.0
```

Case-Sensitive: biến phân biệt chữ hoa/thường.

```
age = 20
Age = 30
print(age, Age)  # 20 30
```

Multiple Assignment: gán cùng lúc nhiều giá trị

```
a = b = c = 100
print(a, b, c)  # 100 100 100

a, b, c = 1, 2, "Zara Ali"
print(a, b, c)  # 1 2 Zara Ali
```

Python Variables - Naming Convention

Quy tắc đặt tên biến:

- Bắt đầu bằng chữ cái hoặc dấu gạch dưới (_).
- Không bắt đầu bằng số hoặc ký tự đặc biệt ($, %, ...).
- Chỉ chứa chữ cái, số và dấu _ (A–Z, a–z, 0–9, _).
- Phân biệt chữ hoa và chữ thường (`Name` $\neq$ `NAME`).
- Không dùng từ khóa Python.

Phong cách đặt tên:

- **camelCase:** kmPerHour
- **PascalCase:** KmPerHour
- **snake_case:** km_per_hour

Ví dụ hợp lệ:

```
counter = 100
_count  = 100
name1 = "Zara"
name2 = "Nuha"
Age  = 20
zara_salary = 100000
```

Python Variables - Invalid Names & Usage

Tên biến không hợp lệ:

```
1counter = 100       # bắt đầu bằng số
$_count  = 100       # chứa ký tự đặc biệt
zara-salary = 100000 # chứa dấu '-'
```

→

Dẫn đến SyntaxError.

Sử dụng biến trong tính toán:

```
width = 10
height = 20
area = width * height
perimeter = 2 * (width + height)

print("Area =", area)
print("Perimeter =", perimeter)
```

Kết quả:

- Area = 200
- Perimeter = 60

Python Variables - Local, Global & Constants

Local Variables: Được định nghĩa bên trong hàm, chỉ dùng trong hàm đó.

```
def sum(x,y):
    result = x + y
    return result

print(sum(5, 10))    # Kết quả: 15
```

Global Variables: Được định nghĩa bên ngoài hàm, có thể truy cập từ bất kỳ hàm nào.

```
x = 5
y = 10
def sum():
    return x + y

print(sum())    # Kết quả: 15
```

Constants: Python không có hằng số thực sự, nhưng quy ước viết `IN_ALL_CAPS` để biểu thị giá trị không thay đổi, ví dụ: `PI_VALUE = 3.14`.

Python Data Types

Khái niệm:

- Trong Python, *data types* là các lớp, còn biến là các *instance/object*.
- Python **dynamically typed**, kiểu dữ liệu được xác định tại runtime.
- Xác định kiểu dữ liệu giúp biết giá trị có thể lưu trữ và các phép toán áp dụng được.

Các kiểu dữ liệu có sẵn trong Python:

- **Numeric:** `int`, `float`, `complex`
- **String:** `str`
- **Sequence:** `list`, `tuple`, `range`
- **Binary:** `bytes`, `bytearray`, `memoryview`
- **Dictionary:** `dict`
- **Set:** `set`, `frozenset`
- **Boolean:** `bool`
- **None:** `NoneType`

Python Numeric Data Types

Khái niệm:

- Numeric data types dùng để lưu trữ giá trị số.
- Khi gán giá trị cho biến, Python tự động tạo đối tượng số.
- Gồm 4 kiểu số chính: `int`, `bool`, `float`, `complex`.

Ví dụ:

```
var1 = 1        # int
var2 = True     # bool
var3 = 10.023   # float
var4 = 10+3j    # complex
print(type(var4))
```

Complex Number:

- Dạng: `x+yj`, với x là phần thực, y là phần ảo.
- Ví dụ: `5+6j` $\rightarrow$ `<class 'complex'>`.

Python String Data Type

- Chuỗi là tập hợp ký tự Unicode, có thể dùng dấu ‘’‘, ‘”“ hoặc ‘”””.
- Chuỗi trong Python là **immutable** (không thay đổi trực tiếp).
- Thuộc lớp `str`, xác minh với `type()`.
- Hỗ trợ các phép toán: slicing, concatenation (+), repetition (*).

```
str = "Hello World!"
print(str[0])      # 'H'
print(str[2:5])    # 'llo'
print(str*2)       # Lặp 2 lần
print(str+"TEST") # Nối chuỗi
```

Python Sequence Data Types

Sequence là tập hợp có thứ tự, chỉ số bắt đầu từ 0. Ba loại sequence chính trong Python:

- **List** – danh sách, mutable, dùng `[ ]`.
- **Tuple** – tương tự list nhưng immutable, dùng `( )`.
- **Range** – dãy số, thường dùng trong vòng lặp.

Python List Data Type

- Chứa nhiều phần tử, có thể khác kiểu dữ liệu.
- Hỗ trợ nested list.
- Phép toán: slicing, concatenation (+), repetition (*).

```
list = ['abcd', 786, 2.23, 'john']
tinylist = [123, 'john']
print(list[0])        # 'abcd'
print(list[1:3])      # [786, 2.23]
print(list+tinylist)# Nối list
```

Python Tuple Data Type

- Giống list nhưng immutable.
- Dùng ( ) hoặc chỉ dấu phẩy.
- Không thể thay đổi giá trị sau khi tạo.

```
tuple = ('abcd', 786, 2.23, 'john')
print(tuple[0])      # 'abcd'
print(tuple[1:3])    # (786, 2.23)
# tuple[2]=1000 -> Lỗi (immutable)
```

Python Range Data Type

- Đại diện cho dãy số, immutable.
- Cú pháp: `range(start, stop, step)`.

```
for i in range(5): print(i)          # 0..4
for i in range(2,5): print(i)        # 2..4
for i in range(1,5,2): print(i)      # 1,3
```

Python Binary Data Types

Python hỗ trợ ba kiểu dữ liệu nhị phân:

- **bytes** – immutable, lưu chuỗi byte.
- **bytearray** – mutable, có thể thay đổi nội dung.
- **memoryview** – ánh xạ trực tiếp đến vùng nhớ.

Python Dictionary Data Type

- Lưu trữ theo cặp `key:value`.
- Dùng dấu ngoặc nhọn { }.
- Mutable, hỗ trợ thêm, sửa, xóa.

```
dict = {"name":"John", "age":25}
print(dict["name"])    # John
print(dict.keys())     # ['name','age']
print(dict.values())   # ['John',25]
```

Python Set Data Type

- Tập hợp không có thứ tự, không trùng lặp.
- Dùng dấu ngoặc nhọn { }.
- Chỉ chứa immutable object.

```
set1 = {123, 452, 5, 6}
set2 = {"Java", "Python"}
print(set1)
print(set2)
```

Python Boolean & None Type

Boolean

- Có 2 giá trị: `True` và `False`.
- Tương ứng với số 1 và 0.

```
a, b = 2, 4
print(a==b)    # False
print(bool(10)) # True
```

None Type

- Đại diện cho giá trị null, thuộc lớp `NoneType`.

```
x = None
print(type(x))    # <class 'NoneType'>
```

Python Data Type Conversion

- Dùng các hàm dựng sẵn: `int()`, `float()`, `str()`, `list()`, `tuple()`, `dict()`, `set()`, ...
- Cho phép ép kiểu dữ liệu linh hoạt.

```
print(int(2.9))     # 2
print(float("3.3"))# 3.3
print(str(100))     # "100"
print(list((1,2,3))) # [1,2,3]
```

Python Type Casting

- **Type Casting** = chuyển đổi đối tượng từ kiểu dữ liệu này sang kiểu khác.
- Python hỗ trợ 2 dạng:
  - **Implicit Casting** – tự động chuyển đổi (ví dụ: int → float).
  - **Explicit Casting** – ép kiểu thủ công bằng hàm `int()`, `float()`, `str()`...
- Dùng khi cần sử dụng dữ liệu ở dạng khác (VD: input chuỗi nhưng cần số).

Python Implicit Casting

- Python chỉ hỗ trợ **int** → **float** (không gây mất dữ liệu).
- Không cho phép tự động chuyển đổi kiểu không liên quan (VD: str → int).
- Quy tắc: kiểu có kích thước nhỏ hơn được nâng lên kiểu lớn hơn.

```
a = 10      # int
b = 10.5    # float
c = a + b   # int tự động -> float
print(c)    # 20.5
```

Implicit Casting với Boolean

- `True` = 1, `False` = 0.
- Khi cộng với số, Boolean được chuyển thành int rồi float.

```
a = True    # 1
b = 10.5
c = a + b
print(c)    # 11.5
```

Explicit Casting với int()

- Dùng `int()` để chuyển đổi từ:
  - int, float, bool → int
  - string số nguyên hợp lệ → int
  - string nhị phân, bát phân, thập lục phân (dùng tham số base)

```
print(int(10.5))        # 10
print(int(True))        # 1
print(int("100"))       # 100
print(int("110011",2)) # 51
```

Explicit Casting với float()

- Chuyển đổi int, float literal, hoặc string hợp lệ → float.
- Hỗ trợ **scientific notation**.

```
print(float(100))      # 100.0
print(float("9.99"))   # 9.99
print(float("1.00E4"))# 10000.0
```

Explicit Casting với str()

- Chuyển bất kỳ object $\rightarrow$ string.
- Số, boolean, list, tuple, dict... đều được biểu diễn dưới dạng chuỗi.

```
print(str(10))        # '10'
print(str(11.1))      # '11.1'
print(str(True))      # 'True'
print(str([1,2,3]))   # '[1, 2, 3]'
```

Chuyển đổi Sequence Types

- `list()` – chuyển tuple/string → list.
- `tuple()` – chuyển list/string → tuple.
- `str()` – chuyển list/tuple → chuỗi.

```
a = [1,2,3,4,5]
b = (1,2,3,4,5)
c = "Hello"

print(list(c))   # ['H','e','l','l','o']
print(tuple(a))  # (1,2,3,4,5)
print(str(b))    # '(1, 2, 3, 4, 5)'
```

Unicode System

- Ứng dụng phần mềm thường cần hiển thị nhiều ngôn ngữ: English, French, Japanese, Hindi,...
- Python sử dụng chuẩn Unicode để biểu diễn ký tự.
- Một ký tự là thành phần nhỏ nhất của văn bản: 'A', 'B', 'C',...
- Unicode string: dãy các **code point** ($0 \rightarrow$ 0x10FFFF).
- Các code point được ánh xạ thành **code units**, và cuối cùng là **bytes**.

Character Encoding

- **Character Encoding**: luật để dịch Unicode string thành bytes.
- Các dạng mã hoá chính:
  - UTF-8
  - UTF-16
  - UTF-32
- UTF = Unicode Transformation Format.

Python's Unicode Support

- Python 3: hỗ trợ Unicode mặc định.
- Mọi string ('str') đều là Unicode.
- Mặc định mã nguồn Python lưu dưới UTF-8.
- Có thể dùng ký tự Unicode trực tiếp hoặc thông qua mã Unicode: Ví dụ: ký tự $\frac{3}{4}$ có thể viết thành `"3/4"` hoặc `"\u00BE"`.

Ví dụ: Biểu diễn ký tự bằng Unicode

```
var = "3/4"
print(var)
var = "\u00BE"
print(var)
```

Ou

3/4

¾

Ví dụ: Chuỗi Unicode giá trị số

```
var = "\u0031\u0030"
print(var)
```

Ou

```
10
```

Encoding và Decoding

- **Encoding**: chuyển string $\rightarrow$ bytes.
- **Decoding**: chuyển bytes $\rightarrow$ string.
- Lưu ý:
  - `encode()` là phương thức của `str`.
  - `decode()` là phương thức của `bytes`.

Ví dụ: Encode và Decode chuỗi ASCII

```
string = "Hello"
tobytes = string.encode('utf-8')
print(tobytes)
string = tobytes.decode('utf-8')
print(string)
```

Ou

```
b'Hello'
Hello
```

Ví dụ: Unicode ký tự Rupee ( )

```
string = "\u20B9"
print(string)
tobytes = string.encode('utf-8')
print(tobytes)
string = tobytes.decode('utf-8')
print(string)
```

Ou

```
b'\xe2\x82\xb9'
```

Python Literals là gì?

- Literals (hay hằng số) là giá trị cố định trong mã nguồn.
- Ví dụ: `123`, `4.3`, `"Hello"`.
- Khác với biến, literals không thay đổi trong quá trình chạy chương trình.
- Ví dụ:
  - `x = 10` $\rightarrow$ 10 là literal.
  - `y = x*2` $\rightarrow$ kết quả 20 không phải literal trong mã nguồn.

Các loại Python Literals

- Integer Literal
- Float Literal
- Complex Literal
- String Literal
- List Literal
- Tuple Literal
- Dictionary Literal

Integer Literals

- Gồm số nguyên thập phân, bát phân (octal), và thập lục phân (hexadecimal).
- Decimal: `x=10`, `y=-25`, `z=0`
- Octal: bắt đầu bằng `0o` hoặc `00` $\rightarrow$ `x=0034`
- Hexadecimal: bắt đầu bằng `0x` hoặc `0X` $\rightarrow$ `x=0X1C`
- Python vẫn lưu tất cả dưới dạng `int`.

Ví dụ: Integer Literals

```
x = 0034 # Octal
print("0034 =", x, type(x))
# Hexadecimal
x = 0X1c
print("0X1c =", x, type(x))
```

Ou

```
0034 = 28 <class 'int'>
0X1c = 28 <class 'int'>
```

Float Literals

- Gồm phần nguyên và phần thập phân, phân cách bằng dấu chấm.
- Ví dụ: `25.55`, `0.05`, `-12.2345`.
- Có thể dùng **notation khoa học** (E/e):
  - `1.23E5` = 123000.0
  - `1.23e-2` = 0.0123

Ví dụ: Float Literals

```
x = 1.23
print(x, type(x))

x = 1.23E5
print(x, type(x))

x = 1.23e-2
print(x, type(x))
```

Complex Literals

- Số phức có dạng `x + yj`, trong đó:
  - x: phần thực
  - y: phần ảo
- `j` hoặc `J` đại diện cho $\sqrt{-1}$.
- Ví dụ: `2+3j`, `2.5+4.6j`.

Ví dụ: Complex Literals

```
x = 2+3j
print(x, type(x))

y = 2.5+4.6j
print(y, type(y))
```

String Literals

- Chuỗi là dãy ký tự Unicode, bất biến (immutable).
- Viết bằng dấu nháy đơn, nháy kép hoặc ba nháy:
  - `'hello'`
  - `"hello"`
  - `'''hello'''` hoặc `"""hello"""`.
- Có thể chứa dấu nháy kép trong nháy đơn và ngược lại.

Ví dụ: String Literals

```
var1 = 'hello'
var2 = "hello"
var3 = '''hello'''
var4 = """hello"""

print(var1, type(var1))
print(var2, type(var2))
print(var3, type(var3))
print(var4, type(var4))
```

List Literals

- Danh sách có thể chứa nhiều phần tử khác kiểu.
- Cú pháp: dấu ngoặc vuông `[ ]`, phần tử cách nhau bởi dấu phẩy.
- Ví dụ: `[1,"Ravi",75.50, True]`.

Ví dụ: List Literals

```
L1 = [1,"Ravi",75.50, True]
print(L1, type(L1))
```

Tuple Literals

- Tuple là bộ sắp xếp, bất biến.
- Cú pháp: dấu ngoặc tròn ( ).
- Có thể bỏ ngoặc và chỉ dùng dấu phẩy.
- Ví dụ: `(1,"Ravi",75.50, True)` hoặc `1,"Ravi",75.50, True`.

Ví dụ: Tuple Literals

```
T1 = (1,"Ravi",75.50, True)
print(T1, type(T1))

T2 = 1,"Ravi",75.50, True
print(T2, type(T2))
```

Dictionary Literals

- Là tập hợp không theo thứ tự, chứa các cặp `key:value`.
- Viết trong dấu ngoặc nhọn `{ }`.
- Ví dụ:
  - `{"USA":"New York", "France":"Paris"}`
  - `{1:"one", 2:"two"}`
- Key phải bất biến (int, str, tuple), không được trùng lặp.

Ví dụ: Dictionary Literals

```
capitals = {"USA":"New York", "France":"Paris"}
numbers = {1:"one", 2:"two", 3:"three"}
points = {"p1":(10,10), "p2":(20,20)}

print(capitals, type(capitals))
print(numbers, type(numbers))
print(points, type(points))
```