

LẬP TRÌNH PYTHON

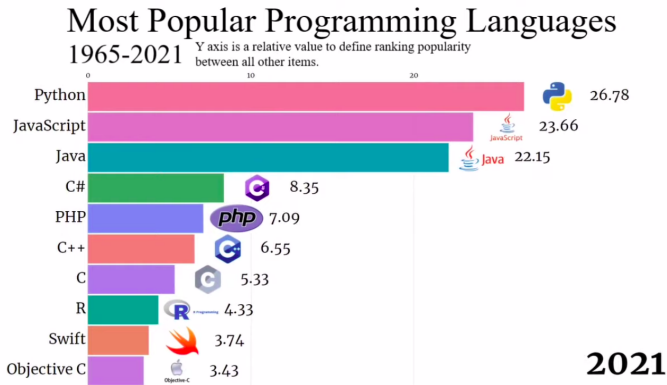
TỔNG QUAN VỀ PYTHON

NGUYỄN HẢI TRIỀU¹

¹Bộ môn Kỹ thuật phần mềm,
Khoa Công nghệ thông tin, Trường ĐH Nha Trang

NhaTrang, February 2022

Tại sao sv cần phải biết Python?



Hình 1: Most Popular Programming Languages

Tại sao sv cần phải biết Python?

Ưu điểm

Python là một ngôn ngữ **phù hợp cho tất cả mọi lứa tuổi**:

- câu lệnh và cấu trúc của Python thật sự rất đơn giản, gần gũi với ngôn ngữ tự nhiên
- hệ thống thư viện mã nguồn mở đồ sộ, cộng đồng hỗ trợ rất lớn
- áp dụng được trong nhiều lĩnh vực như AI, web-backend, game, IoT...

Tại sao sv cần phải biết Python?

Nhược điểm

Vì Python là một ngôn ngữ LT **đa nền tảng**, nên:

- có một số nền tảng chưa được tối ưu
- tốc độ thực thi còn chậm so với C/C++, Java
- hệ thống thư viện công kênh, chiếm dụng bộ nhớ lớn

Tổng quan

- 1 Cài đặt môi trường và sử dụng Python
- 2 Lập trình Python căn bản
- 3 Hàm

- 1 Cài đặt môi trường và sử dụng Python
- 2 Lập trình Python căn bản
- 3 Hàm

Cài đặt môi trường

Có rất nhiều cách cài đặt môi trường để lập trình Python, tuy nhiên trong phạm vi môn học này, ta có 3 phương pháp phổ biến để lập trình Python

- Phương pháp 1: sử dụng [Jupyter notebook trực tuyến](#) như *Google Colab*.
- Phương pháp 2: sử dụng [Jupyter notebook trực tiếp trên máy](#). *Khuyến khích sinh viên sử dụng cách này.*
- Phương pháp 3: code trực tiếp không thông qua Jupyter notebook.

Yêu cầu OS

Tất cả các hướng dẫn trong môn học này được thực hiện bằng [Python 3.x](#) trên Ubuntu 20.04. *Khuyến khích sv sử dụng Linux.*

Google Colab

Để sử dụng được google Colab, sinh viên truy cập link <https://colab.research.google.com/>. Ưu điểm:

- được miễn phí sử dụng các tài nguyên của google như **CPU**, **GPU**, **RAM**... trong vòng *12h*
- sử dụng kết hợp các dịch vụ khác của google như **drive** để lưu trữ

Để kết nối Colab với Drive của bạn, sử dụng các lệnh sau:

```
1 from google.colab import drive
2 drive.mount('/content/drive')
3 %cd /content/drive/'My Drive'/NTU/bert_clustering # đi đến thư
   mục NTU/bert_clustering trên drive của bạn
```

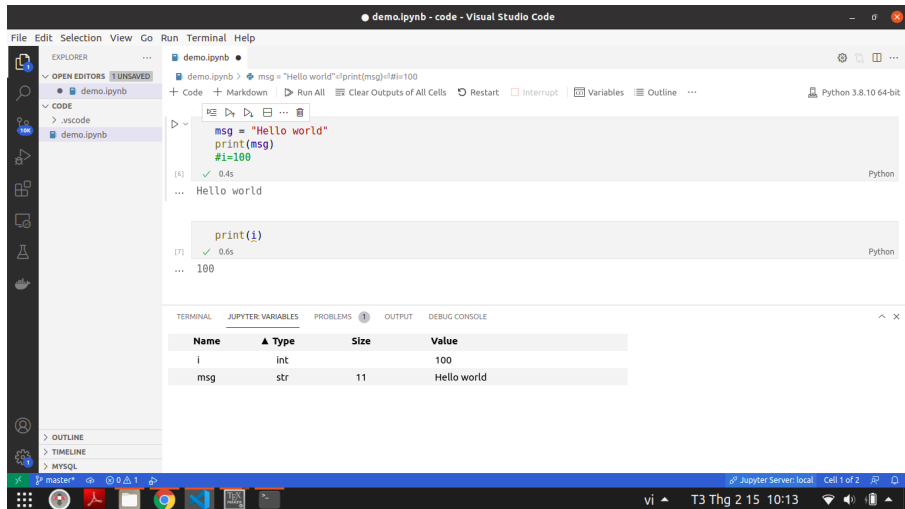

Jupyter notebook trực tiếp trên máy



Cài đặt

- Python tại <https://www.python.org/downloads/>. Lưu ý, nên chọn version 3.6 hoặc 3.8.
- VScode tại <https://code.visualstudio.com/>
- Jupyter notebook thông qua pip. Bật terminal và gõ lệnh:
`pip install jupyter`

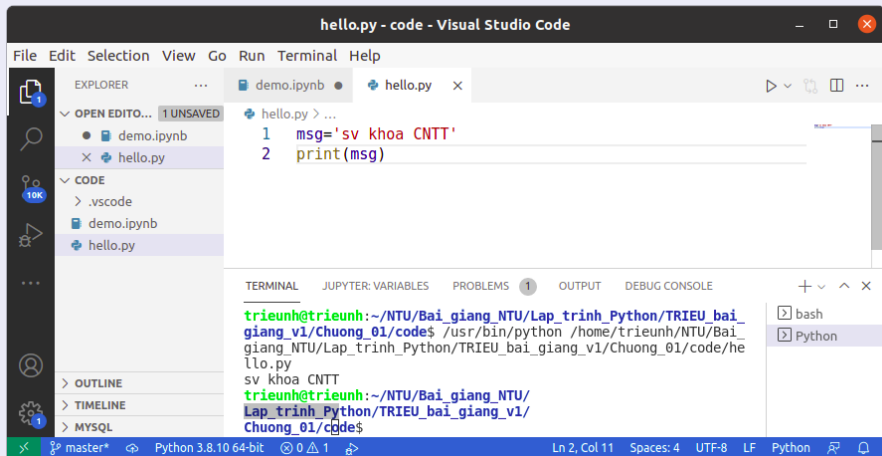
Jupyter notebook trực tiếp trên máy



Hình 2: Kết quả sau khi cài đặt

Code trực tiếp

Sau khi đã hoàn thành cài đặt Python và VScode. SV có thể tạo các file *.py để code chương trình bằng Python.



The screenshot displays the Visual Studio Code interface with a file named `hello.py` open. The code in the editor is:

```
1 msg='sv khoa CNTT'  
2 print(msg)
```

The terminal at the bottom shows the command to run the file and the output:

```
trieunh@trieunh:~/NTU/Bai_giang_NTU/Lap_trinh_Python/TRIEU_bai_giang_v1/Chuong_01/code$ /usr/bin/python /home/trieunh/NTU/Bai_giang_NTU/Lap_trinh_Python/TRIEU_bai_giang_v1/Chuong_01/code/hello.py  
sv khoa CNTT  
trieunh@trieunh:~/NTU/Bai_giang_NTU/Lap_trinh_Python/TRIEU_bai_giang_v1/Chuong_01/code$
```

The status bar at the bottom indicates the file is named `hello.py`, the Python version is 3.8.10 64-bit, and the current line is Ln 2, Col 11.

Môi trường ảo Virtualenv

Sử dụng Virtual Environments (Virtualenv)

- để quản lý môi trường làm việc độc lập cho từng dự án (project)
- đồng bộ môi trường làm việc giữa các lập trình viên
- đồng bộ môi trường khi triển khai (deployment)

Đặc biệt, để tránh xung đột giữa các phiên bản Python, giữa các thư viện giữa các project khác nhau. SV cần phải sử dụng **Virtualenv**

Môi trường ảo Virtualenv

- Để cài đặt môi trường ảo, SV có thể chạy lệnh sau đây trong terminal (hoặc Cmd trên Windows): `pip install virtualenv`
- Kiểm tra xem Virtualenv đã được cài đặt chưa:
`virtualenv --version`
- Đi đến thư mục của project và *tạo môi trường ảo cho project* bằng lệnh:
`virtualenv thư_mục_chứa_môi_trường_ảo` . Thư mục đó chứa các file thực thi của Python và module pip
- Nếu có nhiều phiên bản Python ở local, thì có thể chỉ định rõ phiên bản Python mà bạn muốn sử dụng:
`virtualenv -p /usr/bin/python3.8 my_project`

Môi trường ảo Virtualenv

- Tạo một Virtualenv sạch, không có các gói, module đã được cài đặt sẵn từ trước:
`virtualenv --no-site-packages my_project_env`
- Tạo một Virtualenv kế thừa lại các thư viện đã cài trong hệ thống `virtualenv --system-site-packages my_project_env`

Kích hoạt sử dụng VirtualEnv

Kích hoạt virtualenv đã tạo:

```
source my_project_env/bin/activate
```

Thoát khỏi VirtualEnv

Thoát khỏi virtualenv đang làm việc:

```
deactivate
```

Môi trường ảo Virtualenv

Đồng bộ môi trường ảo VirtualEnv

- Để đồng bộ môi trường ảo của dự án giữa các lập trình viên hay triển khai mã nguồn lên máy mới. Chúng ta sử dụng lệnh **freeze** để xuất danh sách các gói (packages), modules của môi trường hiện tại cùng với phiên bản của chúng
`pip freeze > requirements.txt`
- Sử dụng file requirements.txt này để tạo lại môi trường dự án thông qua lệnh sau:
`pip install -r requirements.txt`

Ngoài ra, chúng ta có thể tạo môi trường ảo và code bằng **Anaconda** hoặc **PyCharm** rất tiện lợi và đơn giản.

- 1 Cài đặt môi trường và sử dụng Python
- 2 Lập trình Python căn bản**
- 3 Hàm

Hello world

Python là một ngôn ngữ có cú pháp rất đơn giản. Bắt đầu với chương trình in ra màn hình chữ *Hello world*. Lệnh đơn giản nhất để in là `print`.

```
1 print("Hello world")
```

Lưu ý có sự khác biệt giữa Python version 2 và 3

Ở Python 2 chỉ thị `print` không phải là hàm nên không có dấu ngoặc `()`

print()

Escape sequence

Escape sequence giúp in kí tự đặc biệt ra màn hình

- `\\`: `print('\\\\')` -> in ra kí tự `\`
- `\'`: `print('\')` -> in ra kí tự `'`
- `\n`: `print('\n')` -> in ra kí tự xuống dòng
- `\t`: `print('\t')` -> in ra kí tự dấu tab

Ví dụ 2.1

Hãy viết chương trình in các dòng sau đây lên màn hình:

```
1 C:\\some\\name
```

print()

Escape sequence

Escape sequence giúp in kí tự đặc biệt ra màn hình

- `\\`: `print('\\\\')` -> in ra kí tự `\`
- `\'`: `print('\')` -> in ra kí tự `'`
- `\n`: `print('\n')` -> in ra kí tự xuống dòng
- `\t`: `print('\t')` -> in ra kí tự dấu tab

Ví dụ 2.1

Hãy viết chương trình in các dòng sau đây lên màn hình:

```
1 C:\\some\\name
```

đáp án:

```
1 print("C:\\\\some\\\\name")
```

```
2 #or
```

```
3 print(r"C:\\some\\name")
```

print()

Ví dụ 2.2

Hãy viết chương trình in các dòng sau đây lên màn hình:

```
1 Usage: thingy [OPTIONS]
2     -h                                Display this usage message
3     -H hostname                       Hostname to connect to
```

print()

Ví dụ 2.2

Hãy viết chương trình in các dòng sau đây lên màn hình:

```
1 Usage: thingy [OPTIONS]
2     -h                               Display this usage message
3     -H hostname                       Hostname to connect to
```

đáp án:

```
1 print("""\
2 Usage: thingy [OPTIONS]
3     -h                               Display this usage message
4     -H hostname                       Hostname to connect to
5 """)
```

Khi chuỗi kí tự kéo dài nhiều dòng, chúng ta có thể sử dụng triple-quotes

```
1 """...""" or '''...'''
```

Comments

Để ghi chú, bình luận lại những thông tin và muốn trình thông dịch Python bỏ qua phần nội dung đó, ta sử dụng ký tự `#`

Ví dụ 2.3

```
1 print("Hello world") # chương trình in ra dòng Hello world
```

Indentation

Ví dụ 2.4

```
1 x = 1
2 if x == True:
3     # indented four spaces
4     print("x is 1.")
5 else: print("x is 0.") # in case x=0
```

Lưu ý

- Python sử dụng thụt đầu dòng cho các khối, thay cho dấu ngoặc nhọn `{}`. Có thể sử dụng cả tab và dấu cách nhưng tiêu chuẩn sử dụng bốn dấu cách.
- Sử dụng dấu `:` phía sau tên hàm, vòng lặp, cấu trúc điều khiển
- **True/False** viết hoa chữ cái đầu
- Không có dấu `;` sau các lệnh, chỉ thị

Biến và kiểu dữ liệu

Mọi biến trong Python là một đối tượng \Rightarrow không cần phải khai báo biến hay kiểu dữ liệu của biến trước khi sử dụng. Python có hai loại biến: toàn cục và cục bộ (sẽ nhắc lại trong phần Hàm).

	Chuỗi kí tự (str)	Số nguyên (int)	số thực (float)
Định nghĩa	các kí tự được đặt trong cặp dấu nháy đơn ' ' hoặc nháy kép "	gồm các số nguyên dương, nguyên âm và 0	gồm phần nguyên và phần thập phân
Ví dụ	'ký tự' "he's" '1234'	123 -123	123.4 -123.0
Hàm chuyển đổi về kiểu dữ liệu	str(123) -> '123' str(23.5) -> '23.5'	int('123') -> 123 int(12.3) -> 123	float(123) -> 123.0 float('12.3') -> 12.3

Bảng 1: Các kiểu dữ liệu cơ bản trong Python

Toán tử $+$ trong Python

Đối với các kiểu dữ liệu số nguyên, số thực toán tử cộng chỉ đơn giản là thực hiện phép toán cộng. Đối với kiểu dữ liệu chuỗi, **toán tử cộng** sẽ thực hiện nối các chuỗi lại với nhau.

Ví dụ 2.5

```
1 hello = "hello"
2 world = "world"
3 helloworld = hello + " " + world
4 print(helloworld)
```

TypeError

Traceback (most recent call last)

Input In [14], in <module>
1 hello = "hello"
----> 2 print(hello+1)

TypeError: can only concatenate str (not "int") to str

Thao tác với chuỗi

Python cung cấp cho chúng ta một loạt phương thức (method) thao tác với chuỗi kí tự. Truy xuất kí tự trong chuỗi dựa vào chỉ số theo cả hướng bắt đầu và kết thúc chuỗi. Chỉ số chuỗi bắt đầu bằng 0.

0	1	2	3	4	5
p	y	t	h	o	n
-6	-5	-4	-3	-2	-1

Ví dụ 2.6

```
1 text="python"
2 print(text[-6]) #->p
3 print(text[2]) #->t
4 print(text[2:]) #->thon
```

Thao tác với chuỗi

Ví dụ 2.7

```
1 text="python"  
2 text[0]='c'
```

Thao tác với chuỗi

Ví dụ 2.7

```
1 text="python"  
2 text[0]='c'
```

-> *TypeError: 'str' object does not support item assignment*

Chuỗi trong Python là bất biến, vì vậy không thể thay đổi được ký tự riêng lẻ trong một vị trí bộ nhớ khi đã khởi tạo.

Giải pháp cho ví dụ 2.7

Thao tác với chuỗi

Ví dụ 2.7

```
1 text="python"  
2 text[0]='c'
```

-> *TypeError: 'str' object does not support item assignment*

Chuỗi trong Python là bất biến, vì vậy không thể thay đổi được ký tự riêng lẻ trong một vị trí bộ nhớ khi đã khởi tạo.

Giải pháp cho ví dụ 2.7

Tạo ra một chuỗi khác.

```
1 text="python"  
2 'c'+text[1:] #-> cython
```

Thao tác với chuỗi

Định dạng chuỗi

- sử dụng **f-strings**: đặt tiền tố **f** ở trước chuỗi, các biến đặt trong dấu **{}**

```
1 name = 'trieu'
2 age = 22
3 print(f"Hello, My name is {name} and I'm {age} years old.")
```

- sử dụng **str.format()**, tương tự như **f-strings**

```
1 print("Hello, My name is {} and I'm {} years old.".format(
    name, age))
```

Thao tác với chuỗi

Định dạng chuỗi

- có thể định dạng chuỗi theo kiểu ngôn ngữ C bằng toán tử %.

```
1 name = "John"
2 age = 23
3 print("%s is %d years old." % (name, age))
```

- ngoài ra, bất kì đối tượng nào không phải là chuỗi thì đều có thể sử dụng %s để chuyển định dạng kiểu chuỗi khi in.

```
1 KDL_list = [1,2,3]
2 print("kieu du lieu danh sach: %s" % KDL_list)
```

Bài tập

Hãy in ra chuỗi có định dạng như sau: **Hello John Doe. Your current balance is \$53.44.** Với `data = ("John", "Doe", 53.44)`

Thao tác với chuỗi

Method	Ý nghĩa	Ví dụ
<code>.strip()</code>	loại bỏ tất cả khoảng trắng ở đầu và cuối của chuỗi	<code>s1.strip()</code>
<code>.upper()</code>	trả về một chuỗi hoa	<code>s.upper()</code>
<code>.lower()</code>	trả về một chuỗi thường	<code>s.lower()</code>
<code>.replace()</code>	thay thế một chuỗi cho chuỗi khác	<pre>s='Khoa CNTT' s1 = 'Khoa'; s2 = 'Lop' print(s.replace(s1,s2))</pre>
<code>.split()</code>	dùng để tách chuỗi thành chuỗi con	<pre>s='Khoa-CNTT' print(s.split("-")) #kq: ['Khoa', 'CNTT']</pre>
<code>.count()</code>	đếm số lần xuất hiện một chuỗi con trong chuỗi	<code>s.count('K')</code>

Bảng 2: Một vài phương thức thao tác với chuỗi

Thao tác với chuỗi

Để lặp lại các chuỗi, ta có thể sử dụng toán tử `*` với số lần lặp mong muốn.

Ví dụ 2.8

```
1 seq = "xin chao\n" * 3
2 print(seq)
3 #dap an
4 xin chao
5 xin chao
6 xin chao
```

Gán giá trị cho biến

Có thể gán nhiều giá trị cho từng biến trên một dòng như sau:

```
1 a, b = 3, 4
2 print(a, b)
```

Quy tắc đặt tên biến

- chỉ được chứa chữ cái từ a->z, A->Z, các số 0->9 và dấu gạch dưới _
- tên biến không nên bắt đầu bằng kí tự số
- đặt tên biến gợi nhớ, khoa học

Hàm `type()`

Trong một số trường hợp muốn kiểm tra kiểu dữ liệu của biến, ta sử dụng hàm `type(tên_biến)`.

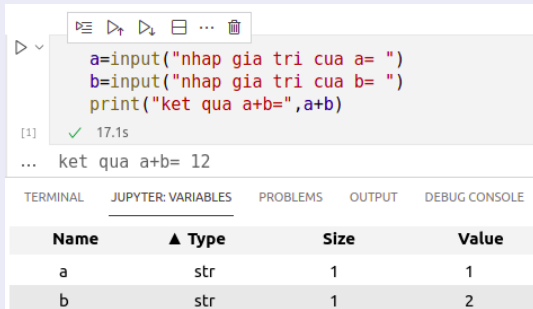
Ví dụ 2.9

```
1 h = "hello"  
2 w = 'world'  
3 c=h+w  
4 type(c) #->str
```

Hàm Nhập dữ liệu từ bàn phím

Để lấy được dữ liệu nhập từ bàn phím, chúng ta dùng lệnh **input()**. Kết quả trả về của lệnh này là một giá trị thuộc kiểu chuỗi (string).

Ví dụ 2.10



```
a=input("nhap gia tri cua a= ")
b=input("nhap gia tri cua b= ")
print("ket qua a+b=",a+b)
```

[1] ✓ 17.1s

... ket qua a+b= 12

JUPYTER: VARIABLES			
Name	Type	Size	Value
a	str	1	1
b	str	1	2

Thư viện tính toán số: `import math`

Các phép toán cơ bản $+$, $-$, $*$, $/$, $\%$ đều giống với các ngôn ngữ lập trình khác. Phép chia lấy nguyên ta có thể dùng toán tử `//` (kết quả được làm tròn xuống). Tuy nhiên để thao tác được với nhiều hàm toán học hơn, ta sử dụng thư viện **`math`**.

Khai báo thư viện trước khi sử dụng

```
import math
```

```
1 #ví dụ sử dụng thư viện math
2 import math
3 x=4
4 math.pi
5 math.sin(x)
6 math.tan(x)
7 math.sqrt(x)
8 math.ceil(x)
9 math.floor(x)
```

Lists

list giúp ta có thể khai báo, lưu trữ, truy xuất một dãy gồm các đối tượng thuộc bất kì kiểu dữ liệu nào. Bản chất của list rất giống array. Khai báo danh sách như sau:

Cú pháp 2.1

$$< ten_danh_sach > = [< pt_1 >, < pt_2 >, ..., < pt_n >]$$

Ví dụ 2.11

$$a = ['hello', 1, 3, 'hi', False, 3.14]$$

Truy xuất các phần tử trong danh sách thông qua chỉ số (giống mảng)

$$a[-6] \longrightarrow 'hello'; a[2] \longrightarrow 3; a[4:] \longrightarrow [False, 3.14]$$

Các thao tác với danh sách

Cú pháp 2.2

- *thay đổi giá trị của một phần tử*
`< ten_danh_sach > [< chi_so >] = < gia_tri_moi >`
- *thêm phần tử vào cuối danh sách, sử dụng hàm append()*
`< ten_danh_sach > .append(< gia_tri >)`
- *thêm phần tử vào vị trí bất kì của danh sách*
`< ten_danh_sach > .insert(< vi_tri >, < gia_tri >)`
- *xóa phần tử ở cuối danh sách*
`< ten_danh_sach > .pop()`
- *xóa phần tử ở vị trí bất kì của danh sách*
`del < ten_danh_sach > [< vi_tri >]`

Các thao tác với danh sách

Cú pháp 2.3

- xóa phần tử đầu tiên trùng với giá trị cho trước
`<ten_danh_sach>.remove(<gia_tri>)`
- đếm số lần xuất hiện của một giá trị trong danh sách
`<ten_danh_sach>.count(<gia_tri>)`
- để tạo một list khác độc lập và giống với hết với list hiện có
`<ten_danh_sach>.copy()`
- nối hai danh sách thông qua toán tử `+`
`<ten_danh_sach1> + <ten_danh_sach2>`
- kiểm tra 1 phần tử có thuộc list, trả về True/False
`<phan_tu> in <ten_danh_sach>`

Các thao tác với danh sách

Lưu ý với phép gán danh sách

Việc gán một danh sách đã cho vào một danh sách mới *không thực sự tạo ra một danh sách mới, độc lập với danh sách cũ mà chỉ đơn thuần là tạo ra một biến danh sách mới tham chiếu đến danh sách cũ.*

Ví dụ 2.12

```
1 a=['hello', 1, 3, "hi", False, 3.14]
2 b=a; b[0]='xin_chao'
3 print(a[0])
```

Các thao tác với danh sách

Lưu ý với phép gán danh sách

Việc gán một danh sách đã cho vào một danh sách mới *không thực sự tạo ra một danh sách mới, độc lập với danh sách cũ mà chỉ đơn thuần là tạo ra một biến danh sách mới tham chiếu đến danh sách cũ.*

Ví dụ 2.12

```
1 a=['hello', 1, 3, "hi", False, 3.14]
2 b=a; b[0]='xin_chao'
3 print(a[0])
```

Đáp án: xin_chao

Các thao tác với danh sách

Sử dụng toán tử $+$ với các danh sách

Với KDL danh sách, toán tử $+$ có vai trò nối các danh sách lại với nhau.

Ví dụ 2.13

```
1 even_numbers = [2,4,6,8]
2 odd_numbers = [1,3,5,7]
3 all_numbers = odd_numbers + even_numbers
4 print(all_numbers)
```

Đáp án: *[1, 3, 5, 7, 2, 4, 6, 8]*

Các thao tác với danh sách

Sử dụng toán tử `*` với các danh sách

Giống như chuỗi, Python hỗ trợ tạo danh sách mới bằng cách sử dụng toán tử nhân với một danh sách đã có:

Ví dụ 2.14

```
1 print([1,2,3] * 3)
```

Đáp án: `[1, 2, 3, 1, 2, 3, 1, 2, 3]`

Các thao tác với danh sách

Sử dụng toán tử **in**, **not in** với các danh sách

Các toán tử này dùng để kiểm tra một phần tử có ở trong list hay không, kết quả trả về **True** hoặc **False**.

Ví dụ 2.15

```
1 a1=["b",2,3]
2 print(100 in a1)
```

Đáp án: *False*

Các thao tác với danh sách

Vì lists có thể chứa các đối tượng thuộc bất kì kiểu dữ liệu nào nên các lists có thể lồng được vào nhau. Truy xuất đến các phần tử thông qua chỉ số tương tự mảng.

Ví dụ 2.16

2 lists lồng vào nhau:

```
1 a1=["b",2,3]
2 a2=[1,1.4,"ws",a1]
3 print(a2[3][0])
4 #-> b
```

Tuples

Cú pháp 2.4

tên_biến_tuple = (item1, item2,...,itemn)

- Kiểu dữ liệu tuple là một danh sách có kích thước cố định, các phần tử trong tuple được sắp xếp theo thứ tự và không thể thay đổi \Rightarrow không được thêm, sửa hoặc xóa các phần tử sau khi tuple được khởi tạo.
- Thường được dùng để lưu trữ các đối tượng không thay đổi
- Tuples được viết bằng dấu ngoặc tròn
- Giống như lists, tuples có thể lồng vào nhau.

```
1 a1=( "b",2,3)
2 a2=(a1,1,'c',a1)
3 print(a2[0])
```

Tuples

Các hàm thao tác với tuple

- Hàm `tuple()` dùng để khởi tạo tuple hoặc dùng để chuyển list thành tuple

```
1 a1=["b",2,3]
2 a2=(tuple(a1),1,'c',a1)
3 print(type(a2[0]))
4 #-> <class 'tuple'>
```

- bên cạnh đó còn có: hàm `len()` trả về số phần tử trong tuple; hàm `max()`, `min()` trả về giá trị lớn nhất, nhỏ nhất; phương thức `.index()` trả về vị trí xuất hiện đầu tiên của phần tử cần tìm.
- các toán tử `+`, `*`, `in`, `not in` tương tự như list

Sets

Cú pháp 2.5

tên_biến_set = {*item1*, *item2*, ..., *itemn*}

- Kiểu dữ liệu set là một tập hợp các phần tử không có thứ tự, không thể thay đổi giá trị các phần tử và không tồn tại chỉ số.
- Mỗi giá trị của phần tử trong set là duy nhất (các phần tử trùng sẽ bị lược bỏ).

```
1 a1={"b",2,3,2}
2 print(a1)
3 #-> {2, 3, 'b'}
```

- set được viết bằng dấu ngoặc {}
- các sets không thể lồng vào nhau.
- có thể thêm (phương thức `.add()`), cập nhật (phương thức `.update()`), xóa phần tử trong set (phương thức `.remove()`)

Dictionary

Cú pháp 2.6

*Từ điển được sử dụng để lưu trữ các giá trị dữ liệu trong các cặp **key:value**. Từ điển được viết trong dấu ngoặc nhọn và có các khóa và giá trị*

```
1 ten_tu_dien={key1:value1, key2:value2,..., keyn:valuen}
```

- từ điển là một tập hợp được sắp xếp theo thứ tự (*lưu ý: Python 3.6 trở về trước, từ điển không có thứ tự*), có thể thay đổi và các phần tử không được phép trùng lặp.

```
1 a1={1:'hello',2:'hi',2:'xinchao','hi':'xinchao'}  
2 print(a1)#->{1: 'hello', 2: 'xinchao', 'hi': 'xinchao'}
```

- các khóa phải là bất biến: số hoặc chuỗi

Dictionary

- để truy cập vào từ điển, ta sử dụng dấu ngoặc `[]` hoặc phương thức `.get()`

```
1 a1={1:'hello',2:'hi',2:'xinchao','hi':'xinchao'}
2 print(a1[1])#->hello
3 print(a1.get('hi'))#->xinchao
```

- xóa một mục của từ điển dùng từ khóa `del ten_tu_dien[key]`; hàm `len(ten_tu_dien)` trả về độ dài của từ điển.
- để lấy một danh sách các cặp giá trị **key:value**, ta sử dụng phương thức `.items()`

```
1 a1={1:'hello',2:'hi',2:'xinchao','hi':'xinchao'}
2 print(a1.items())
3 #->dict_items([(1, 'hello'), (2, 'xinchao'), ('hi', 'xinchao')])
```

- có thể sắp xếp từ điển theo key hoặc value.

So sánh 4 tập kiểu dữ liệu trong Python

	ordered	changeable	Allows duplicate	indexed
List	✓	✓	✓	✓
Tuple	✓	×	✓	✓
Set	×	×	×	×
Dictionary	✓ (Python version ≥ 3.7)	✓	×	✓

Bảng 3: Four collection data types in the Python

- 1 Cài đặt môi trường và sử dụng Python
- 2 Lập trình Python căn bản
- 3 Hàm

Hàm trong Python

Cú pháp 3.1

Hàm trong Python có cú pháp như sau

```
1 def <ten_ham> (<tham_so_1>,<tham_so_2>,...,<tham_so_n>):  
2     <cau_lenh> #cac cau lenh tinh toan xu ly ben trong ham  
3     return <ket_qua_tra_ve>
```

Lưu ý:

- hàm không bắt buộc phải có tham số truyền vào hay kết quả trả ra.
- hàm không được gọi thì không được thực thi

Ví dụ 3.1

```
1 def msg_hello():  
2     print("xin chào")  
3 msg_hello()
```

Hàm trong Python

Ví dụ 3.2

Viết hàm tính tổng hai số nguyên a , b . Trong đó a, b là tham số truyền vào.

Hàm trong Python

Ví dụ 3.2

Viết hàm tính tổng hai số nguyên a, b . Trong đó a, b là tham số truyền vào.

```
1 def tong(a,b):  
2     return a+b  
3 a=1;b=2  
4 c=tong(a,b) # gọi ham để tính tổng 2 số a,b  
5 print("tong {}+{}={} ".format(a,b,c))
```

Trong trường hợp chưa xác định số lượng tham số cần truyền vào có thể sử dụng dấu `*` trước tham số đầu vào.

```
1 def tong(*n):  
2     return n[0]+n[1]  
3 a=1;b=2  
4 c=tong(a,b) # gọi ham để tính tổng 2 số a,b  
5 print("tong {}+{}={} ".format(a,b,c))
```


Hàm main trong Python

Python packages

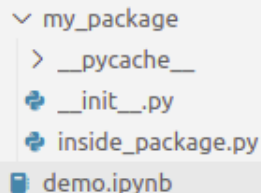
Tạo mới một package có tên là *my_package* theo cấu trúc như sau
inside_package.py

```
1 def msg_hello():  
2     print("in thông báo bên trong package")  
3 msg_hello()
```

demo.ipynb

```
1 from my_package import inside_package  
2 def msg_hello():  
3     print("in thông báo bên NGOÀI package")  
4 msg_hello()
```

Chú thích: câu lệnh `from my_package import inside_package` trong file **demo.ipynb** sẽ liên kết đến file **inside_package.py** để sử dụng được các hàm, biến của **inside_package.py**



Hàm main trong Python

Ở lần đầu chạy notebook **demo.ipynb** thu được kết quả
in thông bao bên trong package
in thông bao bên NGOÀI package

Vì sao thu được kết quả này?

Hàm main trong Python

Ở lần đầu chạy notebook **demo.ipynb** thu được kết quả
in thong bao ben trong package
in thong bao ben NGOAI package

Vì sao thu được kết quả này?

Khi chạy file **demo.ipynb**, đầu tiên trình thông dịch của Python sẽ thực thi lệnh `import inside_package` nên lời gọi hàm `msg_hello()` ở file `inside_package` cũng được thực thi. Sau đó lời gọi hàm `msg_hello()` ở file **demo.ipynb** sẽ được thực thi tiếp theo.

Hàm main trong Python

Giải pháp

Để tránh trường hợp này, chúng ta sử dụng thuộc tính

`__name__` có giá trị là `__main__` như sau:

inside_package.py

```
1 def msg_hello():
2     print("in thông báo bên trong package")
3 if __name__ == '__main__':
4     msg_hello()
```

demo.ipynb

```
1 from my_package import inside_package
2 def msg_hello():
3     print("in thông báo bên NGOÀI package")
4 if __name__ == '__main__':
5     msg_hello()
```

-> khi chạy trực tiếp **demo.ipynb**, chỉ hàm `msg_hello()` bên trong nó mới được thực thi.

Phạm vi hoạt động của biến

Giống như các ngôn ngữ hướng đối tượng khác, biến toàn cục và biến cục bộ có phạm vi và thời gian hoạt động.

- Biến toàn cục: có tác dụng lên toàn bộ chương trình.

```
1 #vi du bien toan cuc
2 def vd_bien():
3     print('gia tri cua bien cuc bo la: %d'%x)
4 x=1
5 vd_bien()
6 #-> gia tri cua bien cuc bo la: 1
```

- Biến toàn cục với từ khóa `global`





```
1 #vi du bien toan cuc voi tu khoa global
2 def vd_bien():
3     global x #x la bien toan cuc.
4     x=100
5 vd_bien()
6 print("gia tri cua x la: %d"%x)
7 #-> gia tri cua bien cuc bo la: 1
```

Phạm vi hoạt động của biến

- Biến cục bộ: chỉ bên trong hàm hoặc trong khối (sau dấu :) và chỉ có tác dụng trong bản thân hàm hoặc khối đó. Biến cục bộ sẽ bị xóa khỏi bộ nhớ khi kết thúc khối đó.

```
1 #vi du bien cuc bo
2 def vd_bien():
3     x=100
4     return x
5 vd_bien()
6 print("gia tri cua x la: %d"%x)
7 #-> NameError: name 'x' is not defined
```

Tài liệu tham khảo

-  V.H. Quân, C.X. Nam, H.T. Hiếu, N.H. Triều, V.C. Tài
Tự học lập trình Python căn bản. *NXB DHQG Tp.HCM, 2019.*
-  Mark Lutz
Learning Python (5th Edition). *O'Reilly Media, Inc, 2013.*
-  Luciano Ramalho
Fluent Python (2nd Edition). *O'Reilly Media, Inc, 2021.*
-  Python Software Foundation
<https://docs.python.org/3/tutorial/>