Fundamental Programming with Python

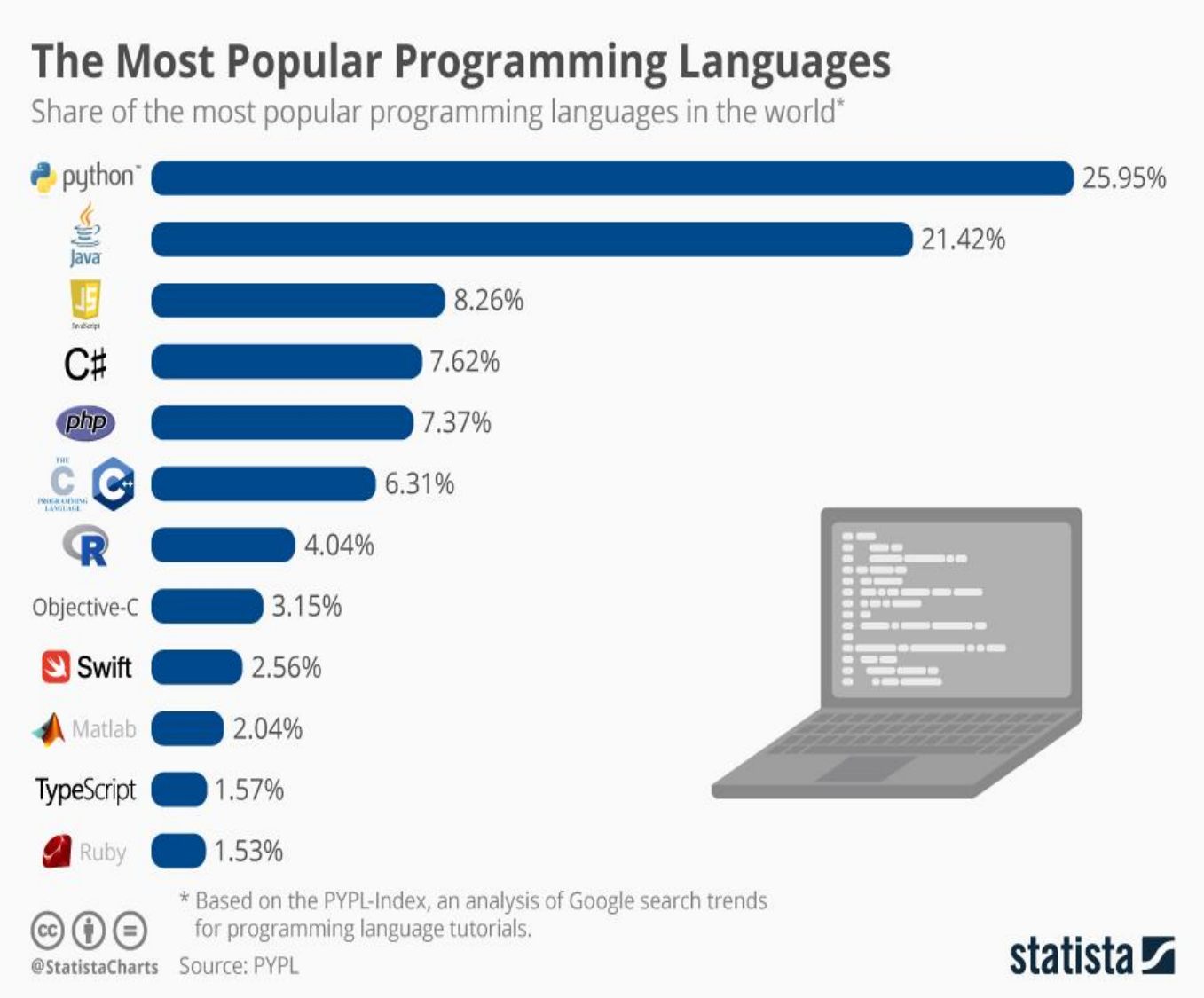
Nội dung

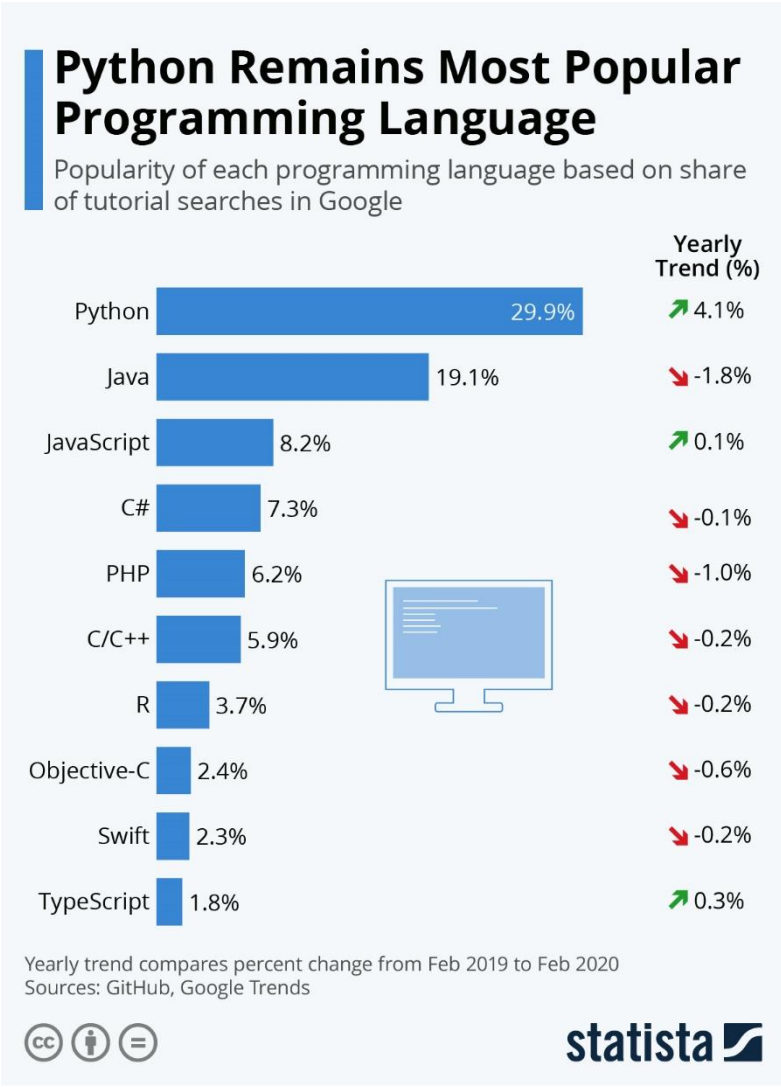
- **Giới thiệu**
- Làm việc với Python

  Setup, running, package,…
- Lập trình Python

  Data types, Control flows, Classes, functions, modules,…
- Phân tích dữ liệu với NumPy, Pandas
- Bài tập

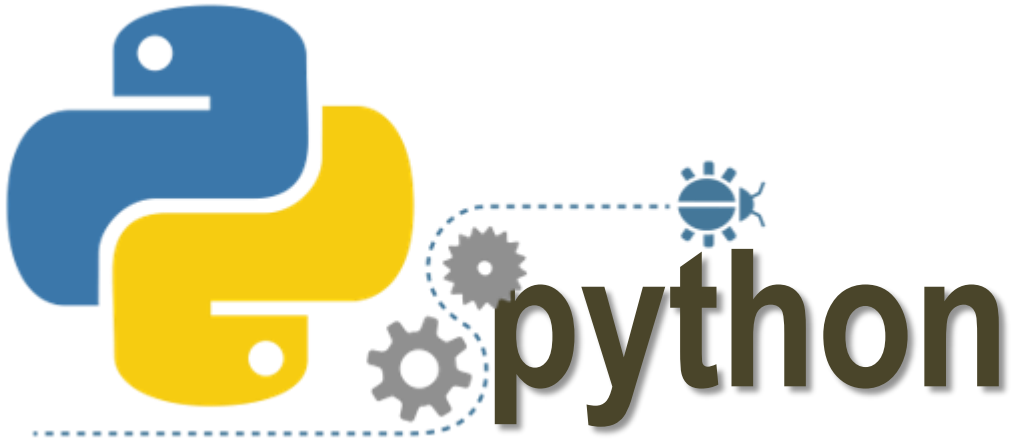
Python là gì?


The Most Popular Programming Languages
Share of the most popular programming languages in the world*
python 25.95%
Java 21.42%
8.26%
C# 7.62%
php 7.37%
6.31%
4.04%
Objective-C 3.15%
Swift 2.56%
Matlab 2.04%
TypeScript 1.57%
Ruby 1.53%
* Based on the PYPL-Index, an analysis of Google search trends for programming language tutorials.
@StatistaCharts
Source: PYPL
statista


Python Remains Most Popular Programming Language
Popularity of each programming language based on share of tutorial searches in Google
Yearly Trend (%)
Python 29.9% 4.1%
Java 19.1% -1.8%
JavaScript 8.2% 0.1%
C# 7.3% -0.1%
PHP 6.2% -1.0%
C/C++ 5.9% -0.2%
R 3.7% -0.2%
Objective-C 2.4% -0.6%
Swift 2.3% -0.2%
TypeScript 1.8% 0.3%
Yearly trend compares percent change from Feb 2019 to Feb 2020
Sources: GitHub, Google Trends
statista

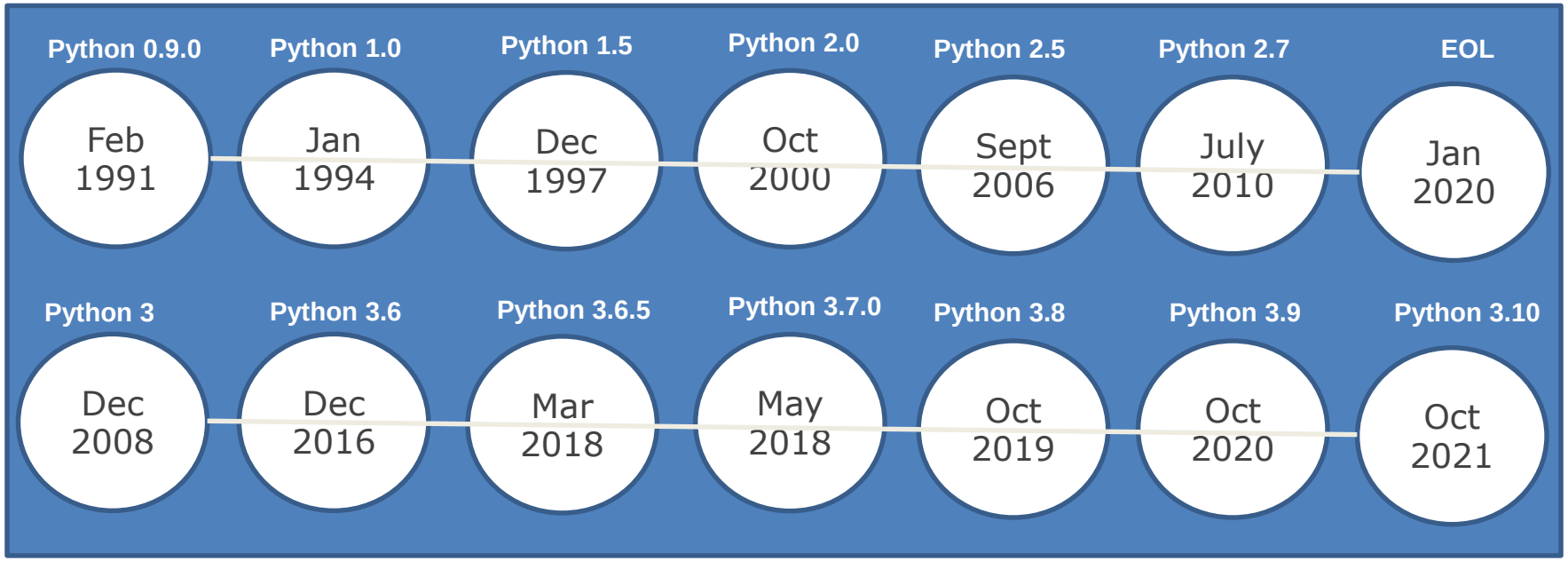
Python là gì?

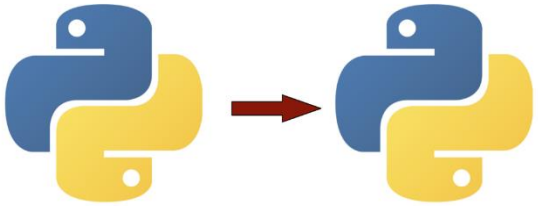
- Ngôn ngữ hướng đối tượng
- Ngôn ngữ thông dịch
- Hỗ trợ kiểu dữ liệu động
- Độc lập với nền tảng
- Tập trung vào thời gian phát triển
- Ngữ pháp đơn giản và dễ hiểu
- Kiểu dữ liệu đối tượng nội bộ cấp cao
- Quản lý bộ nhớ tự động
- Miễn phí (mã nguồn mở)!



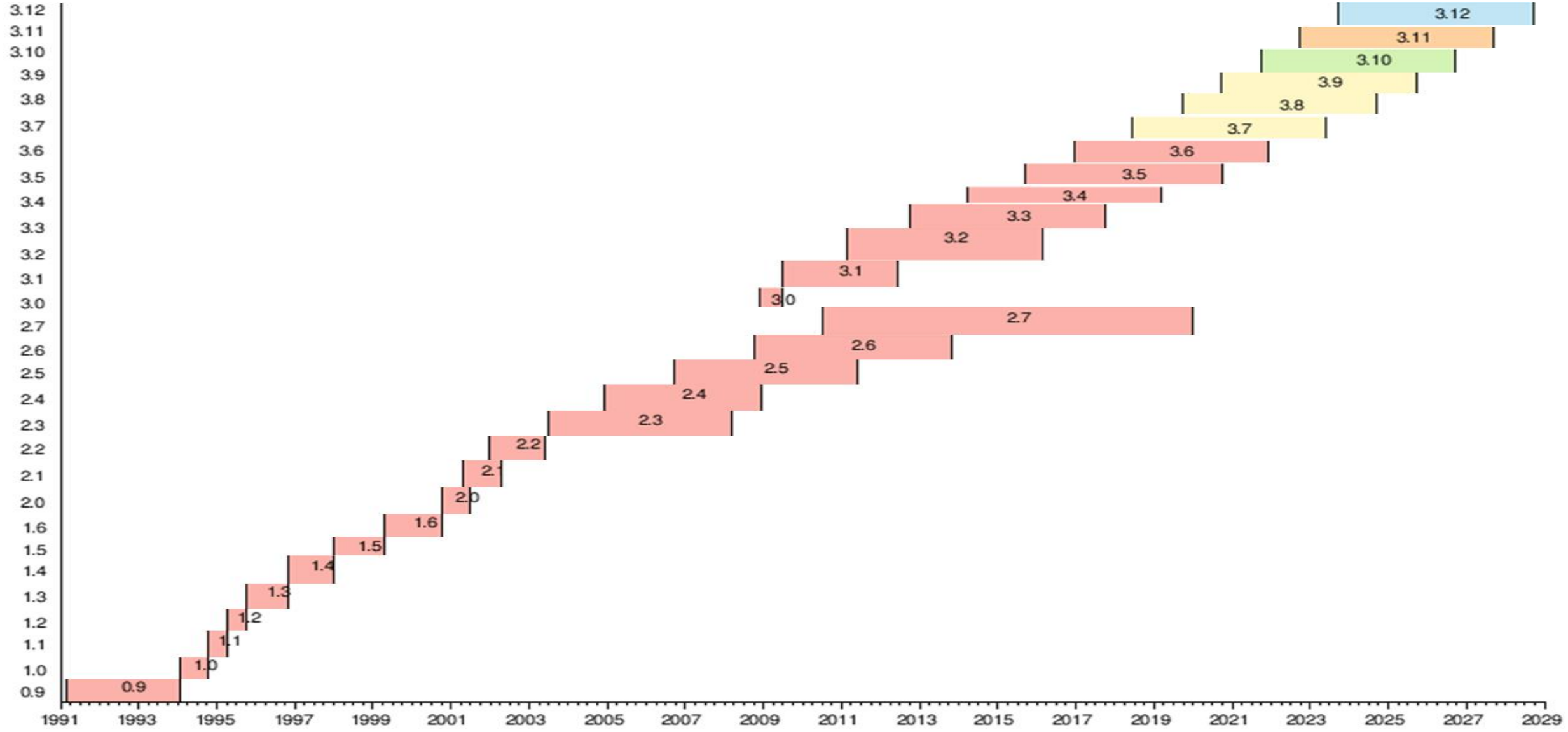
Lịch sử của Python

- Python born, name picked - Dec 1989
  - By Guido van Rossum, now at GOOGLE
- First public release (USENET) - Feb 1991





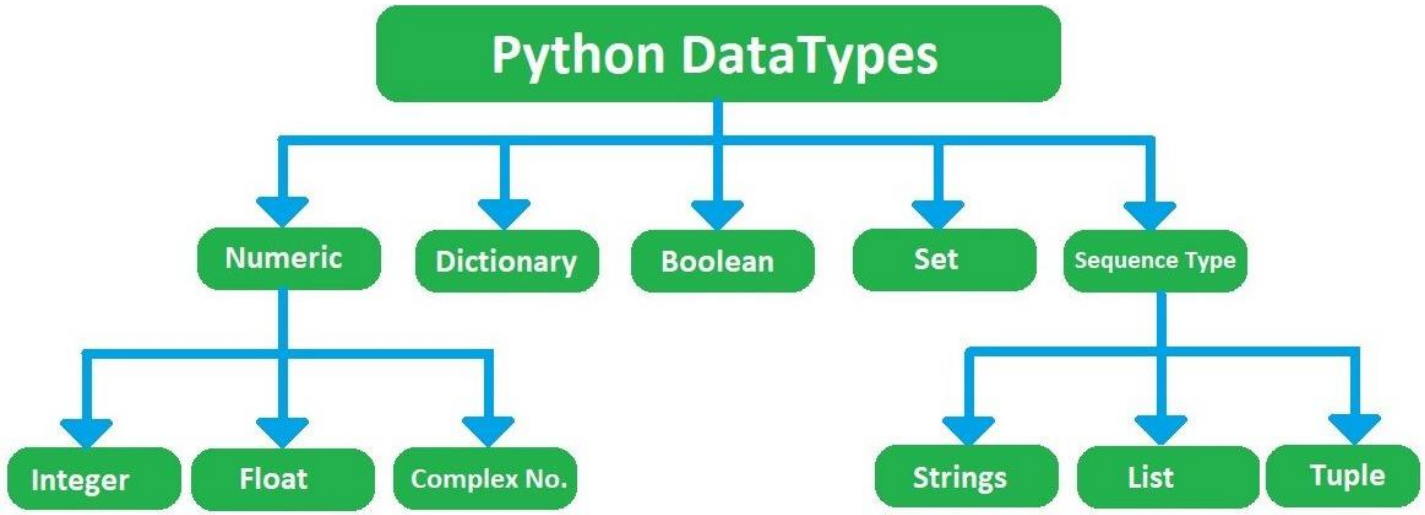
Lịch sử của Python



Các đặc trưng

- Mọi thứ đều là đối tượng
- Mô-đun, lớp, hàm
- Xử lý ngoại lệ
- Kiểu động, đa hình
- Phạm vi tĩnh
- Thụt lề cho cấu trúc khối

- Các kiểu dữ liệu cấp cao:
  - Số: int, long, float, complex
  - Chuỗi: không thay đổi
  - Danh sách và từ điển: container
  - Các kiểu khác, ví dụ như dữ liệu nhị phân, biểu thức chính quy, Mô-đun mở rộng có thể định nghĩa các kiểu dữ liệu "tích hợp sẵn" mới



Tại sao lại học Python?

- "Ngôn ngữ kịch bản" dễ sử dụng
- Hướng đối tượng
  - Có tính giáo dục cao
- Rất dễ học
- Mạnh mẽ, có khả năng mở rộng, dễ bảo trì
  - Năng suất cao
  - Nhiều thư viện

- Lợi ích:
  - Giảm thời gian phát triển
  - Giảm độ dài mã
  - Dễ học và sử dụng với tư cách là nhà phát triển
  - Dễ hiểu mã
  - Dễ thực hiện các dự án nhóm
  - Dễ mở rộng sang các ngôn ngữ khác

Nội dung

- **Giới thiệu**
- Làm việc với Python

  Setup, running, package,…
- Lập trình Python

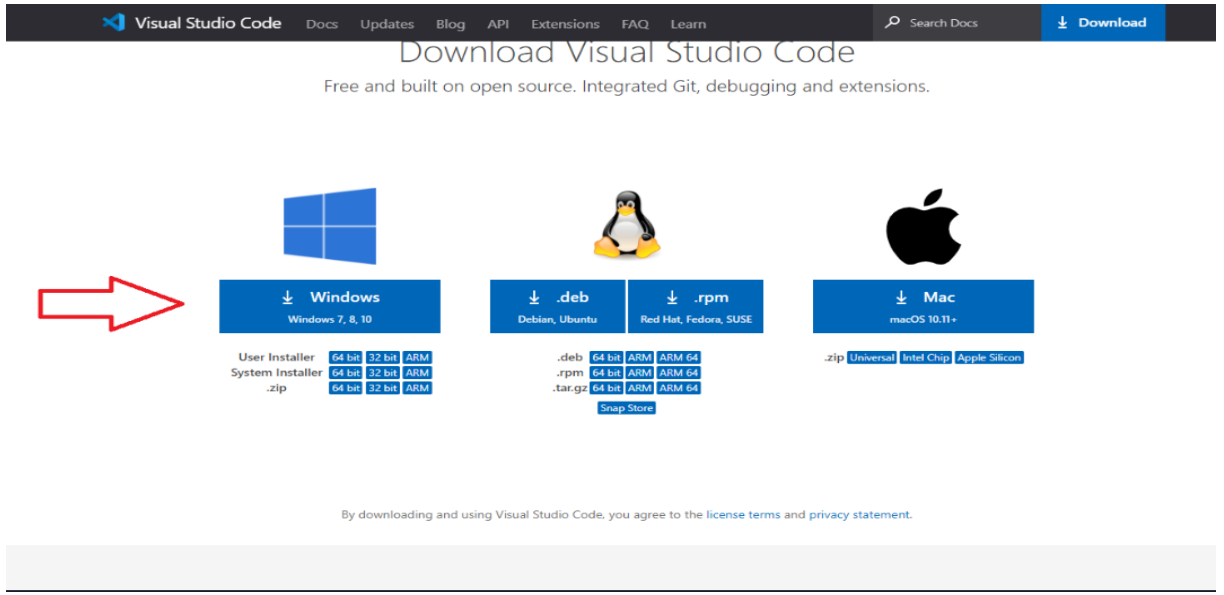
  Data types, Control flows, Classes, functions, modules,…
- Phân tích dữ liệu với NumPy, Pandas
- Bài tập

Cài đặt Python

- Some options to install python

https://www.python.org/downloads

https://code.visualstudio.com/download

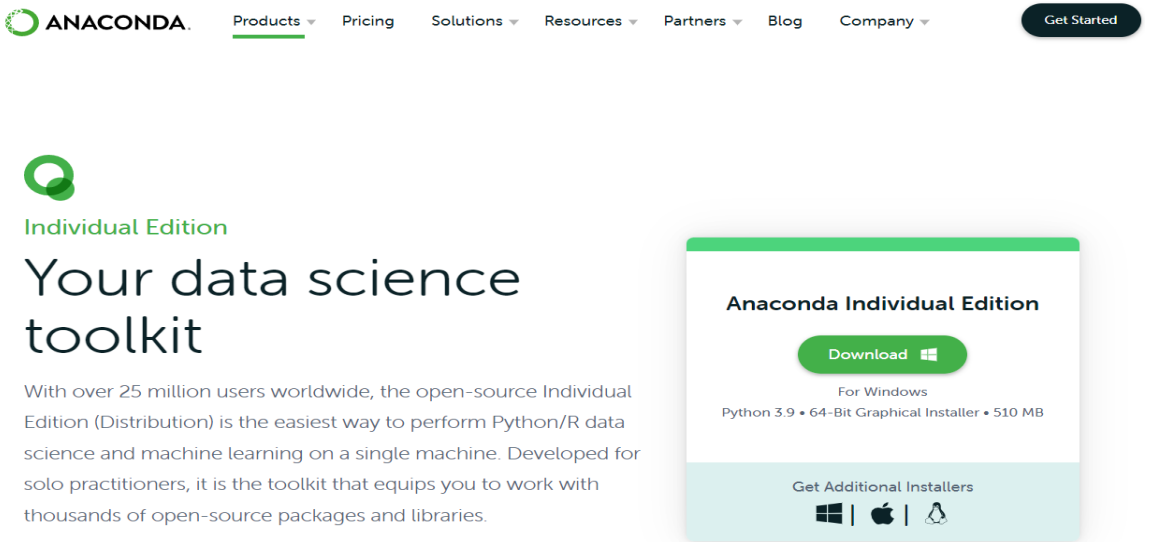


Cài đặt Python

- Some options to install python

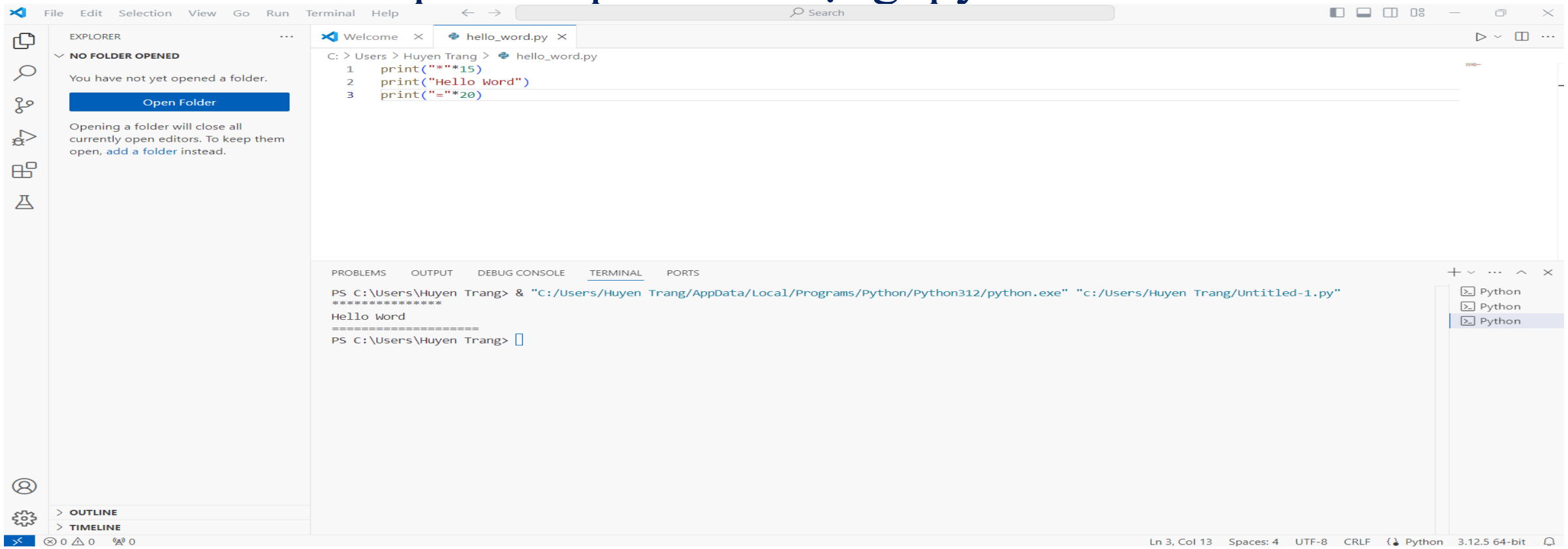
https://www.python.org/downloads

https://www.anaconda.com/products/individual



IDE for Python

- các tệp python thường kết thúc bằng hậu tố .py
- nhưng các tệp thực thi thường không có phần mở rộng .py
- các mô-đun luôn phải có phần mở rộng .py



Running Python Programs Interactively

Suppose the file `script.py` contains the following lines:

```
print ('Hello world')
x = [0,1,2]
```

Let's run this script in each of the ways described on the last slide:

-
```
python -i script.py
Hello world
>>> x
[0,1,2]
```

-
```
$ python
>>> execfile('script.py')
>>> x
[0,1,2]
```

File name:

- các tệp python thường kết thúc bằng hậu tố .py
- nhưng các tệp thực thi thường không có phần mở rộng .py
- các mô-đun phải luôn có phần mở rộng .py
- Notebook =>.ipynb

Comments

- Bắt đầu bằng # và đi đến cuối dòng
- Còn chú thích theo kiểu C, C++ thì sao?
  - KHÔNG được hỗ trợ!

Nội dung

- **Giới thiệu**
- Làm việc với Python

  Setup, running, package,…
- Lập trình Python

  Data types, Control flows, Classes, functions, modules,…
- Phân tích dữ liệu với NumPy, Pandas
- Bài tập

Python Syntax

- Phần lớn tương tự như cú pháp C
- Ngoại lệ:
  - toán tử bị thiếu: ++, --
  - không có dấu ngoặc nhọn,{}, cho các khối;
  - sử dụng khoảng trắng
  - các từ khóa khác nhau
  - nhiều tính năng bổ sung
  - không có khai báo kiểu!

Simple data types

- Numbers
  - Integer, floating-point, complex!
- Strings
  - characters are strings of length 1
- Booleans are **`False`** or **`True`**

Type	Example	Description
int	x = 1	Integers (i.e., whole numbers)
float	x = 1.0	Floating-point numbers (i.e., real numbers)
complex	x = 1 + 2j	Complex numbers
bool	x = True	Boolean: True/False values
str	x = 'abc'	String: characters or text
NoneType	x = None	Special object indicating nulls

Ép kiểu dữ liệu:

Dùng các hàm str(); int(); float

Ví dụ:

```
C:\Users\huyen>python
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> x = 5.0
>>> type(x) # kiểm tra kiểu dữ liệu của x
<class 'float'>
>>> x = int(x) # Ép kiểu dữ liệu của x từ float sang int
>>> x
5
>>>
```

kiểm tra kiểu dữ liệu của x

⇒ Đây là dòng ghi chú

⇒ Sẽ có 2 loại ghi chú sau:

⇒ Ghi chú theo từng dòng: bắt đầu bằng ký tự #

⇒ Ghi chú cho nhiều dòng: bắt đầu bằng ’’’ và kết thúc bằng ’’’

Variables

- Không cần phải khai báo
- Cần gán (khởi tạo)
- Các biến có thể được gán dễ dàng
  - Ví dụ: gán số nguyên 10 cho x
  - x = 10
  - Nghĩa là xác định một con trỏ có tên x trỏ đến số nguyên 10.
  - Chúng ta có thể thay đổi giá trị x bất kỳ lúc nào.
- Mọi thứ đều là biến: functions, modules, classes

```
>>> x= 10           # x là số nguyên 10
>>> x=10.0          # x là số thực 10.0
>>> x='ten'         # x là một string
>>> x=(1,2,3)       # x là một tuple
>>> x=[1,2,4]       # x là một list
```

Tất cả các biến đều là các tham chiếu

Nghĩa của tham chiếu:

- Phép gán thao tác với các tham chiếu

  x = y **không tạo ra một copy của** y

  x = y là làm cho x **tham chiếu đến đối tượng mà** y tham chiếu

- Rất hữu ích nhưng phải cẩn thận!
- Ví dụ:

```
>>> x=[1,2,4]
>>> y=x
>>> y[0]=4
>>> print(y)
[4, 2, 4]
>>> print(x)
[4, 2, 4]
```

Simple data types: toán tử toán học

Toán tử	Tên	Mô tả
a + b	Phép cộng	Tính tổng của a và b
A-b	Phép trừ	Tính hiệu của a và b
A * b	Phép nhân	Tính tích của a và b
a/b	Phép chia	Tính thương của a và b
a//b	Phép chia lấy phần nguyên	Thương của a và b , loại bỏ phần phân số
A % b	Phép chia lấy phần dư	Số dư sau khi chia a cho b
A ** b	Phép mũ	a được nâng lên lũy thừa của b

Simple data types: toán tử so sánh

Toán tử	Mô tả
A == b	A bằng b
A != b	A không bằng b
A < b	A nhỏ hơn b
A > b	A lớn hơn b
A <=b	A nhỏ hơn hoặc bang b
A >=b	A lớn hơn hoặc bằng b

- Kết quả trả về là: True/False

Methods in string

- upper()
- lower()
- capitalize()
- find(s)
- rfind(s)
- index(s)

- strip(), lstrip(), rstrip()
- replace(a, b)
- expandtabs()
- split()
- join()
- center(), ljust(), rjust()

Compound Data Type

Tên loại dữ liệu	Ví dụ	Mô tả
List	[1,2,3]	Tập hợp các phần tử có thứ tự
Tuple	(1,2,3)	Tập hợp các phần tử có thứ tự bất biến
Dict	{‘a’:1, ‘b’:2, ‘c’:3}	Ánh xạ không có thứ tự (khóa, giá trị)
Set	{1,2,3}	Tập hợp các phần tử có giá trị duy nhất không có thứ tự

- Bất biến:
  - Không thể thay đổi hoặc điều chỉnh

Compound Data Type: List

- List:
  - bộ sưu tập dữ liệu có thứ tự và có thể thay đổi cơ bản
  - danh sách có thể có hình dạng bất kỳ và chứa bất kỳ loại dữ liệu nào
  - có thể có lists, floats, integers, tuple, dictionaries, set trong danh sách
  - có thể thêm float 12.0 vào danh sách y bằng y.append(12.0)
  - danh sách y có thể được sắp xếp theo y.sort()
  - số phần tử trong danh sách y có thể được tìm thấy bằng len(y)

Compound Data Type: List

- Ví dụ:
  - y = [] # khởi tạo một danh sách trống
  - y.append(0.2) # nối số float 0,2 vào y
  - y.append(2) # nối số nguyên 2 vào y
  - y.append([4,5,'hello',(11,10)]) # nối thêm một danh sách có tuple vào y
  - y.sort() # sắp xếp y từ thấp đến cao
  - m = len(y) # đếm số phần tử trong danh sách y
  - print('Co tat ca ',m,' phan tu trong danh sach y')
  - print(x)

Note: y.sort() không hoạt động trong trường hợp này với Python 3 vì bạn sẽ cần so sánh các loại dữ liệu khác nhau!!!

Compound Data Type: List

- Indexing:
  - x = [2, 10, 11, 1]
  - Phần tử đầu tiên trong danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[0] => Đầu ra: 2
  - Phần tử thứ hai của danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[1] => Đầu ra: 10
  - Phần tử thứ ba của danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[2] => Đầu ra: 11
  - Phần tử cuối cùng của danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[3] => Đầu ra: 1

Compound Data Type: List

- Indexing:
  - x = [2, 10, 11, 1]
  - Phần tử cuối cùng trong danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[-1] => Đầu ra: 1
  - Phần tử cuối cùng thứ hai của danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[-2] => Đầu ra: 11
  - Phần tử cuối cùng thứ ba của danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[-3] => Đầu ra: 10
  - Phần tử đầu tiên của danh sách có thể được gọi bằng cách sử dụng
    - Nhập vào: x[-4] => Đầu ra: 2

Compound Data Type: List

- Slicing:
  - Syntax: [startPoint : endPoint]
  - x = [2, 10, 11, 1]
  - Nếu muốn lấy phần tử thứ nhất và thứ hai trong danh sách x:
    - Nhập lệnh: x[0:2] => Đầu ra: [2,10]
  - Nếu muốn lấy phần từ thứ 2 đến phần tử thứ 4 trong danh sách x:
    - Nhập lệnh: x[1:4] => Đầu ra: [10,11,1]

Compound Data Type: List

- Slicing with step size:
  - Syntax: [startPoint:endPoint:stepSize]
  - x = [19,18,17,16,15,14,13,12,11,10,9,8,7,6,5,4,3,2,1,0]
  - Nếu:
    - Nhập lệnh: x[0:20:2]
    - Đầu ra: [19,17,15,13,11,9,7,5,3,1]
  - Nếu:
    - Nhập lệnh: x[0:20:5]
    - Đầu ra: [19,14,9,4]

Compound Data Type: List

More list operations

```
>>> a.append(5)          # [0,1,2,3,4,5]
>>> a.pop()              # [0,1,2,3,4]
5
>>> a.insert(0, 5.5)     # [5.5,0,1,2,3,4]
>>> a.pop(0)             # [0,1,2,3,4]
5.5
>>> a.reverse()          # [4,3,2,1,0]
>>> a.sort()             # [0,1,2,3,4]
>>> b = range(2)         # [0,1]
>>> a + b                # [0,1,2,3,4,0,1]
```

Operations in List

- Indexing e.g., L[i]
- Slicing e.g., L[1:5]
- Concatenation e.g., L + L
- Repetition e.g., L * 5
- Membership test e.g., ‘a’ in L
- Length e.g., len(L)

- append
- insert
- index
- count
- sort
- reverse
- remove
- pop
- extend

Nested List

- List in a list
- E.g.,

```
>>> s = [1,2,3]
>>> t = [‘begin’, s, ‘end’]
>>> t
[‘begin’, [1, 2, 3], ‘end’]
>>> t[1][1]
2
```

Compound Data Type: Tuples

- Tuples cũng giống như lists, ngoại trừ việc bạn xác định tuple bằng dấu ngoặc đơn thay vì dấu ngoặc vuông.
- Việc lập chỉ mục danh sách cho một phần của Tuple hoạt động giống hệt như list và bạn sẽ sử dụng dấu ngoặc vuông để gọi các mục của Tuple.
- Các tuple là bất biến, có nghĩa là một khi chúng được tạo ra, chúng không thể sửa đổi theo bất kỳ cách nào.
- Hãy tạo một tuple gồm 1, 3, 5.
  - Y = (1,3,5)

Compound Data Type: Tuples

Operations in Tuple:

Indexing	e.g., T[i]
Slicing	e.g., T[1:5]
Concatenation	e.g., T + T
Repetition	e.g., T * 5
Membership test	e.g., ‘a’ in T
Length	e.g., len(T)

List vs. Tuple

- Đặc điểm chung là gì?
  - Cả hai đều lưu trữ các đối tượng dữ liệu tùy ý
  - Cả hai đều thuộc kiểu dữ liệu tuần tự
- Sự khác biệt là gì?
  - Tuple không cho phép sửa đổi
  - Tuple không có phương thức
  - Tuple hỗ trợ chuỗi định dạng
  - Tuple hỗ trợ tham số có độ dài thay đổi trong lệnh gọi hàm.
  - Tuples nhanh hơn một chút

Dictionaries

- Dictionaries:
  - What is dictionary?
    - Tham chiếu value thông qua key; “mảng kết hợp”
    - Không có chỉ mục với từ điển,thay vào đó là một key.
  - Giống như một array được lập chỉ mục bởi một string
  - Một tập hợp các cặp key: value không có thứ tự
  - *Values thuộc bất kỳ loại nào; hầu hết các key thuộc mọi loại*
    - {"name":"Guido", "age":43, ("hello","world"):1,
      42:"yes", "flag": ["red","white","blue"]}

```
d = { "foo" : 1, "bar" : 2 }
print (d["bar"])   # 2
some_dict = {}
some_dict["foo"] = "yow!"
print (some_dict.keys()) # ["foo"]
```

Dictionary: Methods

Method	Mô tả
clear()	Xóa tất cả các phần tử
copy()	Trả về một bản sao của từ điển
fromkeys()	Trả về một từ điển có các khóa và giá trị được chỉ định
get()	Trả về giá trị của khóa được chỉ định
items()	Trả về một danh sách chứa một bộ cho mỗi cặp giá trị khóa
keys()	Trả về một danh sách chứa các khóa của từ điển
pop()	Xóa phần tử có khóa được chỉ định
popitem()	Xóa cặp khóa-giá trị được chèn cuối cùng
setdefault()	Trả về giá trị của khóa được chỉ định. Nếu khóa không tồn tại: chèn khóa, có giá trị được chỉ định
update()	Cập nhật từ điển với các cặp khóa-giá trị được chỉ định
values()	Trả về danh sách tất cả các giá trị trong từ điển

Dictionary: Methods

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

phone.clear()

print(phone)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

x = phone.copy()

print(x)
```

```
x = ('key1', 'key2', 'key3')
y = 0

thisdict = dict.fromkeys(x, y)


print(thisdict)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

x=phone.get(“model”)

print(x)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

x=phone.items()

print(x)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

x=phone.keys()

print(x)
```

Dictionary: Methods

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

phone.pop(“model”)

print(phone)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

phone.popitem()

print(phone)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

x=phone.setdefault("model", "Bronco")

print(x)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

phone.update{"color": "White"})

print(phone)
```

```
phone = {
  "brand": “samsung",
  "model": “zflip",
  "year": 2020
}

x=phone.values()

print(x)
```

Files: Input/Output

iFile = open('data', 'r')	Open the file for input
S = iFile.read()	Read whole file into one String
S = iFile.read(n)	Reads n bytes (chars) (n >= 1)
L = iFile.readlines()	Returns a list of line strings

oFile = open('data', 'w')	Open the file for writing
oFile.write(S)	Writes the string S to file
oFile.writelines(L)	Writes each of the strings in list L to file
oFile.close()	Manual close

open() and file()

- These are identical:

```
f = open(filename, "r")
f = file(filename, "r")
```

(The `file()` version is the recommended way to open a file now)

```
"r"=> Read existing content
"w"=> Write - will overwrite any existing content
"a"=> Append - will append to the end of the file
```

```
f = open("demo1.txt", "a")
f.write("Write some contents to the file!")
f.close()
```

```
f = open("demo1.txt", "r")
line = f.readline()
print (line)
f.close()
```

OOP Terminology

Một mô hình lập trình trong đó mã liên quan đến một đối tượng cụ thể được nhóm lại với nhau.

- class – định nghĩa về một object
- object – một instance của một Lớp
- method -- một hàm là một phần của đối tượng và hoạt động trực tiếp trên các instance
- constructor -- "phương thức" đặc biệt tạo ra các instance mới
- mọi thứ trong Python đều là đối tượng
- ưu điểm - ít mã trùng lặp
- nhược điểm - vấn đề về hiệu suất với OOP

Objects

- Hãy tưởng tượng bạn phải viết một chương trình thú y về mèo. Tất cả các con mèo đều có chức năng và thuộc tính gì chung?
- Objects:
  - Một đối tượng là gì?
    - cấu trúc dữ liệu
    - các functions (methods) hoạt động trên nó

```
class cat:
# Definition of the class here


t = cat()
t.method()
print (t.field)
```

Defining a class

```
class tk_nganhang:
    """ Lớp này lưu trữ một đối tượng tùy ý."""
```

```
def __init__(self, sodu_dau=0.0):
    """Khởi tạo một tài khoản ngân hàng."""
    self.sodu = sodu_dau
```

constructor

```
def withdraw(self, sotien):
    self.sodu -= sotien
    print('Số dư mới của bạn là', self.sodu)
```

method

Defining a class

```
class tk_nganhang:
    """ Lớp này lưu trữ một đối tượng tùy ý."""
    def __init__(self, sodu_dau=0.0):
        """Khởi tạo một tài khoản ngân hàng."""
        self.sodu = sodu_dau
```

constructor

```
    def withdraw(self, sotien):
        self.sodu -= sotien
        print('Số dư mới của bạn là', self.sodu)
```

method

```
    def deposit(self, sotien):
        self.sodu += sotien
        print('Số dư mới của bạn là ', self.sodu)
```

method

Using a class

```
in : an = tk_nganhang(100.0)    # calls __init__ method
in : an.withdraw(2.77)           # calls withdraw method
out: Số dư mới của bạn là 97.23

in : an.deposit(10.0)            # calls deposit method
out: Số dư mới của bạn là 107.23
```

- an là một instance của class **`tk_nganhang`**
- **`Withdraw và deposit`** là các method of class **`bank_account`**

Ví dụ

```
Nhập lệnh:
   alice = bank_account(100, 0)

Nhập lệnh: alice.withdraw(20)
Đầu ra: Your account balance is ….
        Your own the bank ….

Nhập lệnh: alice.deposit(200)
Đầu ra: Your account balance is ….
        Your own the bank ….

Nhập lệnh: alice.get_loan(1000)
Đầu ra: Your account balance is ….
        Your own the bank ….
```

```
Nhập lệnh: alice.pay_debt(200)
Đầu ra: Your account balance is ….
        Your own the bank ….
```

Control flow: while loops

`while` loops

Example

```
a = 10
while a > 0:
    print (a)
    a =a- 1
```

- Common `while` loop idiom:

```
f = open(filename, "r")
while True:
    line = f.readline()
    if not line:
        break
    # do something with line
```

`for` loops

Example

```
for a in range(10): #0->9
    print (a)
```

- Common `for` loop idiom:

```
a = [3, 1, 4, 1, 5, 9]
for i in range(len(a)):
    print (a[i])
```

Control flow: while loops

Duyệt trên một danh sách

Ví dụ:

```
name = ['trang', 'an', 'trinh', 'long', 'linh', 'lan']
for i in name:
    print(i)
```

```
trang
an
trinh
long
linh
lan
```

Control flow: while loops

Duyệt đồng thời trên hai danh sách

Ví dụ:

```
name = ['trang', 'an', 'trinh', 'long', 'linh', 'lan']
age = [36,23,12,65,45,43]
for n,a in zip(name,age):
    print(i)
```

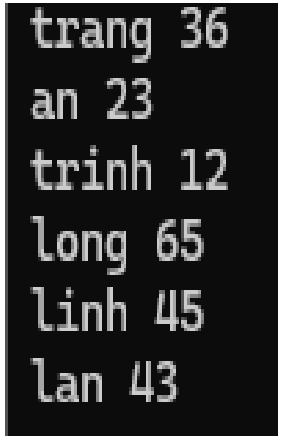
```
trang 36
an 23
trinh 12
long 65
linh 45
lan 43
```

Control flow: while loops

Duyệt đồng thời trên hai danh sách

Ví dụ:

```
name = ['trang', 'an', 'trinh', 'long', 'linh', 'lan']
age = [36,23,12,65,45,43]
for i in range(len(name)):
    print(name[i], age[i])
```



Control flow: for loops

Duyệt các phần tử với range()

Ví dụ:

```
name = ['trang', 'an', 'trinh', 'long', 'linh', 'lan']
sum_name = ''
for i in range(len(name)):
    sum_name = sum_name + name[i]
print(sum_name)
```

```
trangantrinhlonglinhlan
```

Control flow: for loops

Cách range()hoạt động

- Cú pháp: range(startPoint, endPoint, stepSize)
- range(100): - lặp từ 0 đến 99 (100 lần)
- range(0,100) - lặp từ 0 đến 99 (100 lần)
- range(0,100,2) - lặp 0, 2, 4, ..., 98 (50 lần)

Control flow

Sử dụng break để ngắt vòng lặp

• Ví dụ:

```
for i in range(-10,11):
    if i == 0:
        if i == 0:
    print(i)
```

```
i=0
n=0
while i == 0:
    n += 1
    if n > 100:
        if n > 100:
            break
```

Control flow: for loops

Vòng lặp lồng nhau:

• Ví dụ:

```
for i in range(0,4):
    for i in range(0,4):
        print(i)
```

```
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
3 0
3 1
3 2
3 3
```

phẳng tốt hơn lồng nhau

- Sử dụng các hàm với các tác vụ cụ thể để tránh số lượng lớn các lớp lồng nhau
- Rất khó để gỡ lỗi mã có nhiều lớp - nhưng lại dễ kiểm tra các chức năng
- Vị trí ngắt của bạn trong mã lồng nhau rất quan trọng. Bạn đang cố gắng phá vỡ vòng lặp nào ???

Control flow: for loops

Vòng lặp có index và value- enumerate() :

- Thường thì tôi cần tạo một vòng lặp thông qua một mảng, trong đó tôi sẽ cần chỉ mục và giá trị. Tôi sử dụng enumerate() để làm điều này.

Ví dụ:

```
x = [8,181,129,10,'hi']
for index, value in enumerate(x):
    print(index, value)
```

```
0 8
1 181
2 129
3 10
4 hi
```

- Nếu không rõ ràng, chỉ mục và giá trị có thể được đặt tên bất cứ thứ gì bạn muốn.

Ví dụ:

```
x = [8,181,129,10,'hi']
for i,j in enumerate(x):
    print(i, j)
```

Control flow

- **if, if/else, if/elif/else**

```
if a == 0:
    print ("zero!")
elif a < 0:
    print ("negative!")
else:
    print ("positive!")
```

- **continue** statement like in C
- **pass** keyword:

```
if a == 0:
    pass  # do nothing
else:
    # whatever
```

➢ **Chú ý:**

- khối được phân cách bằng cách thụt lề!
- dấu hai chấm (:) được sử dụng ở cuối dòng chứa từ khóa luồng điều khiển

Functions: Một vài điều cơ bản

Định nghĩa hàm:

```
def hiword():
  print('Hellow Python Function')

def foo(x):
   y = 10 * x + 2
   return y
```

Thực thi:

```
def foo(x):
    y = 10 * x + 2
    return y

print (foo(10))  # 102
```

Câu lệnh def được sử dụng để định nghĩa một hàm.

- Tất cả các biến đều là cục bộ trừ khi được chỉ định là toàn cục (global)
- Đối số được truyền theo giá trị (value)

• Tên hàm ở đây là foo.
• Các hàm được sử dụng () để truyền đầu vào.
• Đầu vào cho foo là x.
• Đầu vào là tùy chọn, xem hiworld().
• Một lần nữa : được sử dụng ở cuối câu lệnh.
• Thụt lề biểu thị khối mã của hàm.

Functions: Đối số tùy chọn và đầu vào - đầu ra

Định nghĩa hàm:

```
def foo(x):
    y = 10 * x + 2
    return y

def foo(x,c):
    y = 10 * x + c
    return y
```

- Đối số là tùy chọn, báo nhiêu đối số thì truyền bấy nhiêu giá trị
- Đầu vào có thể là bất cứ gì objects, lists, strings, floats, etc.
- Đầu ra cũng có thể là bất cứ gì objects, lists, strings, floats, etc.
- Return được dùng để truyền đầu ra

Functions: multiple returns

Định nghĩa hàm:

```
def computation(x,y):
  y1 = x+y
  y2=x-y
  y3=x*y
  y4=x/y
  return y1,y2,y3,y4

def computation(x,y):
  return x+y, x-y, x*y, x/y
```

```
def computation(x,y):
    y1=x+y
    y2=x-y
    y3=x*y
    y4=x/y
    return y1,y2,y3,y4

print(computation(10,5))
```

```
(15, 5, 50, 2.0)
```

```
def computation(x,y):
    return x+y, x-y, x*y, x/y

print(computation(10,5))
```

```
(15, 5, 50, 2.0)
```

- return có thể trả lại nhiều đầu ra
- Phân cách đầu ra bằng dấu phẩy

Functions: Đối số linh hoạt

Định nghĩa hàm:

```
def catchAll(*args, **kwargs):
    print('args', args)
    print('kwargs', kwargs)
```

```
>>> def catchAll(*args, **kwargs):
...     print('args', args)
...     print('kwargs', kwargs)
...
>>> print(catchAll(1,2,3,4,a=7.0,b=3.7))
args (1, 2, 3, 4)
kwargs {'a': 7.0, 'b': 3.7}
```

- Hàm catchAll thể hiện các đối số và từ khóa
- args là một bộ các đối số được truyền cho hàm
- kwargs là một từ điển chứa các từ khóa được truyền

Modules

Module là gì?

- Một module chỉ đơn giản là một không gian tên.
- Có thể nghĩ về các tập tin bạn viết có đuôi .py là module.
- Module là các hàm và biến được định nghĩa trong các tệp riêng biệt
- Khi một mô-đun được nhập, mã trong tệp đó sẽ được chạy và bất kỳ tên nào được xác định trong tệp đó giờ sẽ có sẵn

Tại sao sử dụng mô-đun?

- Tái sử dụng mã
  - Các quy trình có thể được gọi nhiều lần trong một chương trình
  - Các quy trình có thể được sử dụng từ nhiều chương trình
- Phân vùng không gian tên
  - Nhóm dữ liệu cùng với các hàm được sử dụng cho dữ liệu đó
- Triển khai các dịch vụ hoặc dữ liệu được chia sẻ
  - Có thể cung cấp cấu trúc dữ liệu toàn cục được nhiều chương trình con truy cập

Modules

Ví dụ: Ta có một tệp python có tên tinh.py như sau:

```
x=1
y=9
z=6
def phepcong(x,y,z):
        return print(x+y+z)
```

- Những tên nào sẽ được định nghĩa trong mô-đun đó?
- Bạn có thể truy cập vào những tên đó bằng cách nào?

What/How use modules

- Các mục được nhập bằng from or import
  - from module import function
  - function()
  - import module
  - module.function()

- import tinh
  - tinh.x
  - tinh.y
  - tinh.z
  - tinh.phepcong(x,y,z)
- From tinh import phepcong
  - phepcong(x,y,z)

Modules

- Access other code by importing modules

```
import math
print (math.sqrt(2.0))
```

or:

```
from math import sqrt
print (sqrt(2.0))
```

or:

```
from math import *
print (sqrt(2.0))
```

- Can import multiple modules on one line:

```
import sys, string, math
```

- Only one "`from x import y`" per line