

# Ngôn ngữ Python

## 1. Python là gì:

### 1.1 Lịch sử ra đời:

- Python là ngôn ngữ kịch bản (scripting) ra đời năm 1990 do **Guido van Rossum** tạo ra năm 1990.
- Cú pháp rất tường minh và dễ đọc, vì thế rất thích hợp cho người mới bước vào nghề lập trình.
- Dữ liệu động ở mức cao.
- Còn rất nhiều ưu điểm khác khiến cho python là một trong những ngôn ngữ được yêu thích nhất hiện nay.

# Ngôn ngữ Python

## 1. Python là gì:

### 1.2 Cú pháp python:

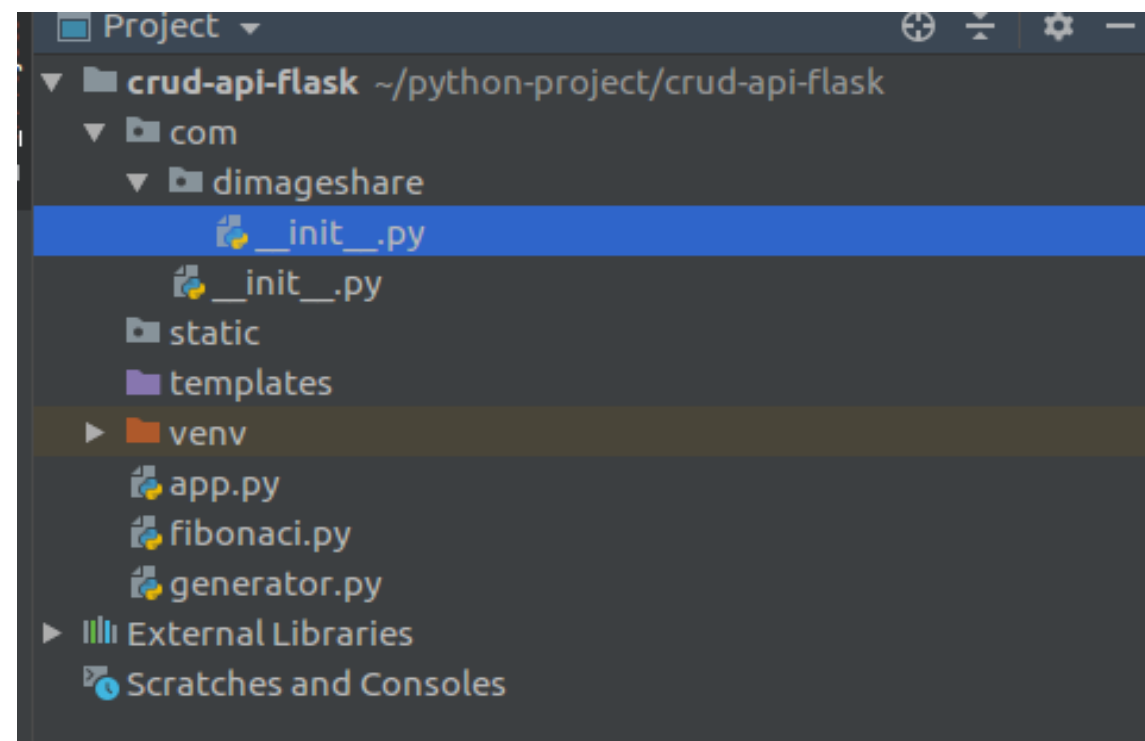
- Python sử dụng thụt đầu dòng, không sử dụng cặp dấu {} như các ngôn ngữ khác.
- Sử dụng từ khóa : **class** để tạo 1 class, **def** để tạo method.
- Python là ngôn ngữ Script tuy nhiên cũng hỗ trợ OOP cực kỳ mạnh mẽ.
- Ví dụ OOP như ở slide tiếp theo:

# Ngôn ngữ Python

## 1. Python là gì:

### 1.2 Cú pháp python:

```
class Account(object):  
    def __init__(self, username, email, password):  
        self.username = username  
        self.email = email  
        self.password = password  
  
    def get_username(self):  
        return self.username  
  
    def get_email(self):  
        return self.email  
  
    def get_password(self):  
        self.password  
  
if __name__ == '__main__':  
    account = Account('duybac', 'bac93.it@gmail.com', '123456789')  
    print(account.get_username())  
    print(account.get_email())  
    print(account.get_password())
```



# Ngôn ngữ Python

## 1. Python là gì:

### 1.2 Cú pháp python:

- Từ khóa **class** định nghĩa 1 lớp. Các method được định nghĩa bởi từ khóa **def**
  - Hàm **\_\_init\_\_** là hàm khởi tạo.
  - Khởi tạo 1 package, python cũng dùng file khởi tạo là **\_\_init\_\_.py**
  - Từ khóa **self** dùng để tham chiếu đến các biến của lớp Account. Vai trò của nó giống từ khóa **this** trong java
  - Khi Python chạy "tệp chứa mã nguồn" dưới vai trò là chương trình chính, nó sẽ gán giá trị ("\_\_main\_\_") cho biến (**\_\_name\_\_**).
  - Khi bạn thực thi hàm main, nó sẽ đọc câu lệnh "if" và kiểm tra xem biến **\_\_name\_\_** có mang giá trị **\_\_main\_\_** không.
- => Trong Python, "if **\_\_name\_\_** == **\_\_main\_\_** " cho phép bạn chạy các tệp chứa mã nguồn Python dưới dạng các mô-đun có thể tái sử dụng hoặc các chương trình độc lập.

# Ngôn ngữ Python

## 1. Python là gì:

### 1.2 Cú pháp python:

- Về các từ khóa được dùng trong python, bạn đọc có thể vào trang của python để tìm hiểu:

<https://docs.python.org/3/tutorial/> . (Dành cho python version 3.x). <https://docs.python.org/2/tutorial/> . (Dành cho python version 2.x).

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.1 Kiểu dữ liệu **list** và **tuple**:

List	Tuple
Khai báo: list= ['A', 'B', 'C']	Khai báo: tuple =('A', 'B', 'C')
Thao tác: 1. lấy ra: list[0] = 'A', list[1] = 'B'... dùng for: ví dụ: for el in list: print(el)  2. xóa: list.remove(index) 3. thêm: list.append(element) (thêm vào vị trí cuối cùng), list.insert(index, element) (thêm một phần tử vào vị trí chỉ định) ...	Thao tác: 1. lấy ra: tuple [0] = 'A', list[1] = 'B'... dùng for: ví dụ: for el in tuple: print(el)  2. remove: không có. 3. thêm : không có. ...

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.1 Kiểu dữ liệu **list** và **tuple**:

- Như vậy, về cơ bản việc lưu dữ liệu của tuple giống list, tuy nhiên **list** là kiểu dữ liệu có thể biến đổi (mutable) tức là có thêm xóa... còn tuple là kiểu dữ liệu không thể thay đổi (immutable).
- Tùy vào bài toán mà ta sử dụng **tuple** hay **list** cho phù hợp.

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

##### 1. Đối tượng iterator:

- Iterator là một khái niệm chỉ các lớp dạng danh sách cho phép chúng ta duyệt qua (for loop...) chúng mà không cần quan tâm đến kiểu dữ liệu đó là gì. Ví dụ list, tuple, string, dictionary...đều là 1 iterator.

- Một đối tượng là iterator khi nó có 2 phương thức: `__iter()` Phương thức này đơn giản là trả về một đối tượng tham chiếu tới chính nó. `__next()` Phương thức này trả về phần tử tiếp theo có trong danh sách cho đến phần tử cuối cùng thì giải phóng exception `StopIteration`.

- Xem ví dụ ở slide sau:



# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

##### 1. Đối tượng iterator:

```
str = "DimageShareVn"  
it = iter(str)  
  
print(it.__next__())  
print(it.__next__())  
print(it.__next__())  
print(it.__next__())  
print(it.__next__())
```



```
Run: iterator_generator x  
/usr/bin/python3.6 /home/entropy/python_workspace/flask-python-tutorial/com/dimageshare/entity/iterator_generator.py  
D  
i  
m  
a  
g  
  
Process finished with exit code 0
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

- Generator là cách đơn giản để tạo ra iterator. Một generator là một hàm trả kết quả về là một chuỗi kết quả thay vì một giá trị duy nhất.
- Generator sử dụng từ khóa **yield**. Để hiểu hơn xem ví dụ sau:

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

```
generator.py x
1 def generator(n):
2     index = 0
3     for i in range(n):
4         index += 5
5         yield i, index
6
7
8 def calculator(n):
9     index = 0
10    for i in range(n):
11        index += 5
12        return i, index
13
14
15 if __name__ == '__main__':
16     print(generator(5))
17     print(calculator(5))
```



```
Run: generator
/home/entropy/python-project/crud-api-flask/venv/bin/python /home/entropy/python-project/crud-api-flask/generator.py
<generator object generator at 0x7f376190f360>
(0, 5)
Process finished with exit code 0
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

- Sự khác nhau duy nhất giữa 2 hàm là một hàm dùng từ khóa **yield** trả về một generator object, một hàm dùng từ khóa **return**, trả về giá trị.
- Do vậy function với **yield** sẽ trả về 1 generator object thay vì 1 giá trị cụ thể như **return**.
- Tiếp tục với hàm generator:

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

```
generator.py x
1 def generator(n):
2     index = 0
3     for i in range(n):
4         index += 5
5         yield i, index
6
7
8 def calculator(n):
9     index = 0
10    for i in range(n):
11        index += 5
12        return i, index
13
14
15 if __name__ == '__main__':
16     gen_obj = generator(5)
17     for returned_obj in gen_obj:
18         print(returned_obj)
19
```



```
Run: generator x
/home/entropy/python-project/crud-api-flask/venv/bin/python /home/entropy/python-project/crud-api-flask/generator.py
(0, 5)
(1, 10)
(2, 15)
(3, 20)
(4, 25)
Process finished with exit code 0
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

- Như vậy, chúng ta có thể duyệt qua **generator obj** và nhận giá trị mỗi lần **yield** được gọi.
- Tiếp tục test với hàm `iter()`, `next()`, :

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

```
generator.py x
1 def generator(n):
2     index = 0
3     for i in range(n):
4         index += 5
5         yield i, index
6
7
8 def calculator(n):
9     index = 0
10    for i in range(n):
11        index += 5
12        return i, index
13
14
15 if __name__ == '__main__':
16     gen_obj = generator(5)
17     print(iter(gen_obj))
18     print(next(gen_obj))
19     print(len(gen_obj))
```



```
Run: generator x
/home/entropy/python-project/crud-api-flask/venv/bin/python /home/entropy/python-project/crud-api-flask/generator.py
<generator object generator at 0x7ff2dd1f5360>
Traceback (most recent call last):
(0, 5)
File "/home/entropy/python-project/crud-api-flask/generator.py", line 19, in <module>
    print(len(gen_obj))
TypeError: object of type 'generator' has no len()
Process finished with exit code 1
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

- Ta thấy rằng generator hỗ trợ hàm `iter()` và `next()` nhưng không hỗ trợ hàm `len()` (bắn ra exception). Do đó, ta có các kết luận sau đây:

-> Generator là Iterator (vì có hỗ trợ các hàm `iter ()` và `next ()`) tuy nhiên chiều ngược lại thì không đúng. Tức là các data structure như list, dictionary...có hỗ trợ hàm `len()` chúng là 1 iterator nhưng không là 1 generator.

-> Generator không phải là 1 loại data structure (list, dict, set, etc.), mà là 1 loại function (lazy evaluation). Function này sẽ yield giá trị cần trả lại tại thời điểm khi `next()` được gọi.



# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

- Nói ngắn gọn, generator là 1 cách để sinh ra iterator.
- Việc sử dụng generator có thể nâng cao hiệu suất bởi vì generator chỉ thực sự sinh kết quả khi được gọi. Do đó, nó sẽ sử dụng ít bộ nhớ hơn. Ngoài ra, chúng ta không cần phải chờ tất cả các phần tử của nó được sinh ra hết mới có thể sử dụng. Chúng sẽ được sinh trong quá trình chúng ta gọi generator.
- Ví dụ về việc áp dụng generator trong thuật toán tìm dãy số Fibonacci:

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

- Nói ngắn gọn, generator là 1 cách để sinh ra iterator.
- Việc sử dụng generator có thể nâng cao hiệu suất bởi vì generator chỉ thực sự sinh kết quả khi được gọi. Do đó, nó sẽ sử dụng ít bộ nhớ hơn. Ngoài ra, chúng ta không cần phải chờ tất cả các phần tử của nó được sinh ra hết mới có thể sử dụng. Chúng sẽ được sinh trong quá trình chúng ta gọi generator.
- Ví dụ về việc áp dụng generator trong thuật toán tìm dãy số Fibonacci:

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

```
fibonacci.py x
1 def gen_fibonacci(n):
2     star_val = 0
3     next_val = 1
4     for i in range(n):
5         yield star_val
6         star_val, next_val = next_val, star_val + next_val
7
8 if __name__ == '__main__':
9     fibonacci_arr = []
10
11     for fn in gen_fibonacci(20):
12         fibonacci_arr.append(fn)
13     print(fibonacci_arr)
```



```
Run: fibonacci x
/home/entropy/python-project/crud-api-flask/venv/bin/python /home/entropy/python-project/crud-api-flask/fibonacci.py
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
Process finished with exit code 0
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

1.3.2 Đối tượng **iterator** và **generator** trong python:

## 2. Đối tượng generator:

-> Nhờ việc sử dụng **generator** đã tiết kiệm được memory vì không cần phải lưu toàn bộ mảng (fibonaci\_arr []) trong bộ nhớ. Và generator function của chỉ cần có 2 variables là *start\_val* và *next\_val*. Kỹ thuật này gọi là [lazy evaluation](#) (Chỉ tính toán khi gọi đến nó).

-> Tóm lại: Hãy cân nhắc khi sử dụng 2 từ khóa: **return**, **yield**.

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.3 Hàm **decorator**:

1. Trong python, chúng ta có thể định nghĩa một hàm bên trong một hàm khác.
2. Trong python, một hàm có thể được coi như là một tham số của một hàm khác (một hàm có thể cũng có thể trả lại một hàm).

-> Đây chính là hàm decorator, đặc trưng bởi dấu @ + một cái tên nào đó. (Giống annotation trong java).

- Ví dụ:

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.3 Hàm decorator:

```
def return_welcome(str):  
    def welcome():  
        return "Hello world"  
  
    return welcome() + " " + str  
  
def return_name(name):  
    return name  
  
print(return_welcome(return_name("DimageShareVn")))
```



```
Run: decorator x  
/usr/bin/python3.6 /home/entropy/python-project/advance-python-tutorial/decorator.py  
Hello world DimageShareVn  
  
Process finished with exit code 0
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.3 Hàm decorator:

- Cách viết kia có thể viết lại với @ như sau:

```
#-----Rewrite-----  
  
def return_welcome(fun):  
    def welcome(name):  
        return "Hello world " + fun(name)  
    return welcome  
  
@return_welcome  
def return_name(name):  
    return name  
  
print(return_name("GeeksforGeeks"))
```



```
Run: decorator x  
/usr/bin/python3.6 /home/entropy/python-project/advance-python-tutorial/decorator.py  
Hello world GeeksforGeeks  
  
Process finished with exit code 0
```

# Ngôn ngữ Python

## 1. Python là gì:

### 1.3 Cú pháp python - nâng cao:

#### 1.3.3 Hàm **decorator**:

- Tóm lại, decorator là một công cụ mạnh mẽ để loại bỏ dư thừa, cho phép tái sử dụng code. Mở rộng các hàm hoặc lớp mà không cần thay đổi code có sẵn.
- Do đó, nó thể hiện sự pro của bạn so với người khác, hãy tận dụng :D.



# Ngôn ngữ Python

## 1. Python là gì:

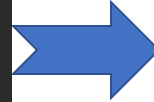
### 1.3 Cú pháp python - nâng cao:

#### 1.3.4 Kế thừa và đa hình trong python:

```
class Animal(object):  
    def __init__(self, name):  
        self.name = name  
  
    def show_info(self):  
        print("It's " + self.name)  
  
    def run(self):  
        print("runing...")
```



```
from animal import Animal  
  
class Chickend(Animal):  
    def __init__(self, name, age, height):  
        super().__init__(name)  
        self.age = age  
        self.height = height  
  
    def show_info(self):  
        print("It's " + self.name)  
        print(" age " + str(self.age))  
        print(" height " + str(self.height))  
  
if __name__ == '__main__':  
    chickend = Chickend("Kentucky", 2, 20)  
    chickend.show_info()  
    chickend.run()
```



```
Run: chicken  
/usr/bin/python3.6 /home/entropy/python-project/advance-python-tutorial/chicken.py  
It's Kentucky  
age 2  
height 20  
runing...  
  
Process finished with exit code 0
```

# Ngôn ngữ Python

## 2. Cài đặt môi trường cho Python:

### 2.1 Cài đặt môi trường cho python:

- Việc cài đặt môi trường cho python, có thể làm theo 2 cách:

1. Cách thứ nhất là cài riêng rẽ python. Sau đó chúng ta cài riêng các tool để sử dụng (pip, anaconda, miniconda...)
2. Cách thứ hai, cài một gói có sẵn. Ví dụ anaconda, miniconda. Việc cài gói sẽ tiện lợi hơn vì ngoài việc chứa môi trường python, bản thân các gói này là một tool để cho phép quản lí các lib khác.

-> Riêng môi trường window, các bạn nên dùng cách 2 để cài đặt python một cách nhanh chóng hơn thay vì cài riêng rẽ do việc can thiệp vào sâu vào window khó hơn phía linux, mac.

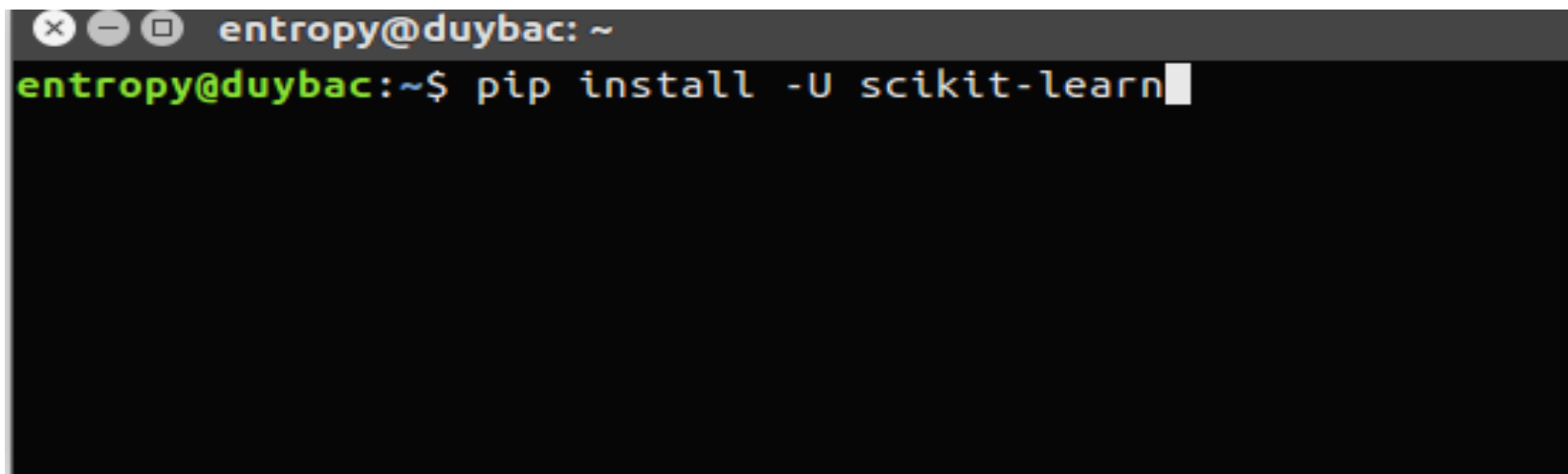
# Ngôn ngữ Python

## 2. Cài đặt môi trường cho Python:

### 2.1 Cài đặt môi trường cho python:

- Thêm thư viện:

Trong project, chúng ta sẽ cần nhiều lib, thư viện bên ngoài để hỗ trợ, cài đặt chúng bằng các tool quản lý. Ví dụ dùng pip để install lib.



```
entropy@duybac: ~  
entropy@duybac:~$ pip install -U scikit-learn
```

# Ngôn ngữ Python

## 3. Python với machine learning:

- Python là một trong những ngôn ngữ được dùng nhiều nhất để viết các thuật toán vì tính đơn giản nhưng mạnh mẽ của nó.
- Được support rất lớn từ cộng đồng, python cùng với R, C, C++ là các ngôn ngữ hàng đầu nếu có ý định chuyên sâu về data scientiest.
- Dưới đây là 2 bài toán rất hay được dùng trong hệ thống recommend. Bài toán phân loại log (classfify), bài toán phân cụm (clustering).

# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

- Bài toán phân cụm user (Clustering user). Giả sử chúng ta có một chuỗi siêu thị. Chúng ta đã có các thông tin về user như sau : mã, tuổi, giới tính, mức lương, số điểm tích lũy mua sắm (từ 1 - 100). Chúng ta cần phân loại user thành 5 tập để tìm ra các tập user tích cực mua sắm tại siêu thị chúng ta, và các tập user còn lại với tần suất nhỏ hơn, từ đó có những chiến lược chăm sóc phù hợp cho từng loại

-> Đây chính là bài toán phân cụm, một bài toán rất quan trọng trong bigdata.

# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

-> Ý tưởng thuật toán:

1. Ý tưởng đơn giản nhất về cluster (cụm) là tập hợp các điểm ở *gần nhau trong một không gian nào đó* (không gian này có thể có rất nhiều chiều trong trường hợp thông tin về một điểm dữ liệu là rất lớn).
2. Về phần công thức toán học, khá lằng nhằng, trong khuôn khổ bài viết không thể nêu hết, bạn đọc vui lòng tự tìm hiểu.

# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

-> Ý tưởng thuật toán:

3. Các bước thuật toán như sau:

**Đầu vào:** Dữ liệu  $X$  và số lượng cluster cần tìm  $K$ .

**Đầu ra:** Các center  $M$  và label vector cho từng điểm dữ liệu  $Y$ .

# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

- 1. Khởi tạo

- Chọn ngẫu nhiên  $k$  điểm bất kì làm điểm trung tâm

$$\mathbb{C}^{(0)} = \{m_1^{(0)}, m_2^{(0)}, \dots, m_k^{(0)}\}$$

- 2. Nhóm dữ liệu

- Nhóm mỗi điểm dữ liệu vào 1 cụm có điểm trung tâm gần nhất với nó

$$\mathbb{S}_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2\} \quad , \forall j, 1 \leq j \leq k$$

- Nếu các cụm sau khi nhóm không thay đổi so với trước khi nhóm thì ta dừng giải thuật

- 3. Cập nhật trung tâm

- Với mỗi cụm sau khi nhóm lại, ta cập nhật lại điểm trung tâm của chúng bằng cách lấy trung bình cộng

$$m_i^{(t+1)} = \frac{1}{|\mathbb{S}_i^{(t)}|} \sum_{x_j \in \mathbb{S}_i^{(t)}} x_j$$

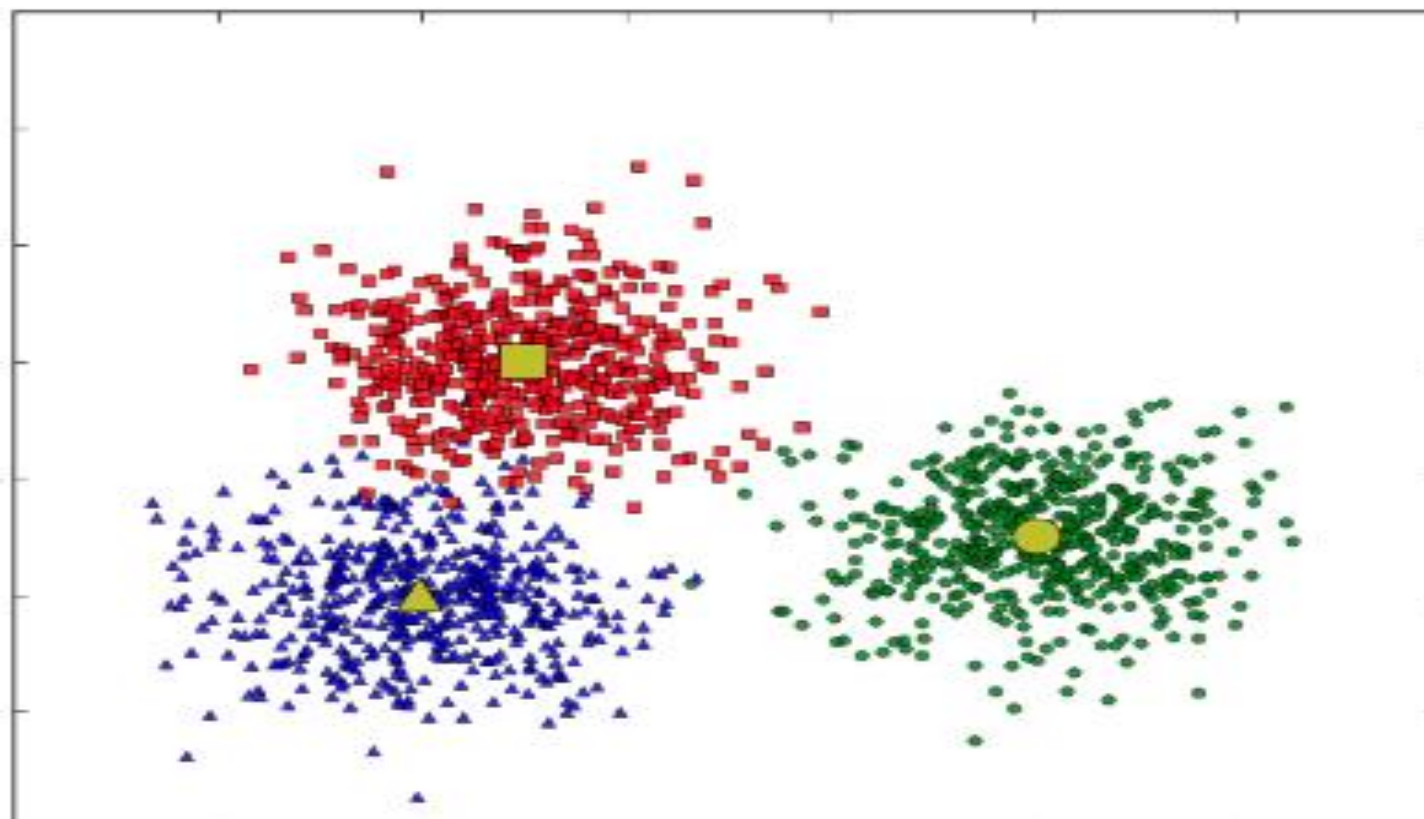
- Quay lại bước 2



# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

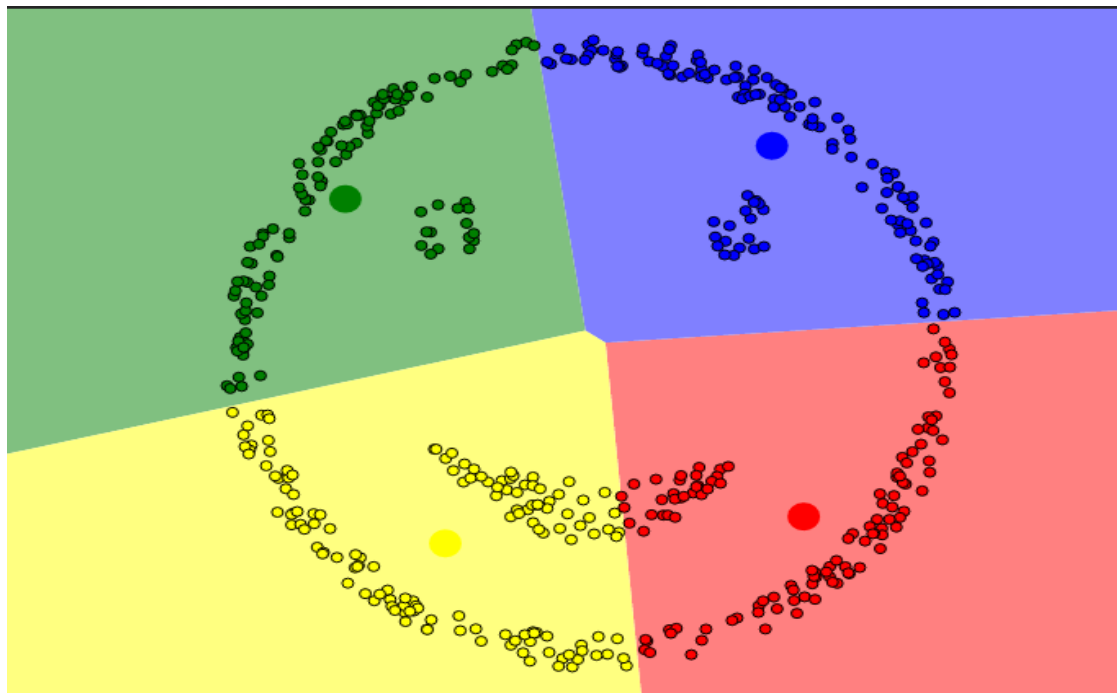


# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

-> Trường hợp nhiều: Khi một cluster nằm trong 1 cluster khác, thuật toán này không thể phân cụm được



# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.1 Thuật toán phân cụm (clustering):

-> Câu hỏi đặt ra: Chọn  $K$  = bao nhiêu. (Tức là muốn phân thành bao nhiêu cụm ) ?

-> Trả lời: Tùy vào bài toán, cũng như kinh nghiệm của người phát triển. Data đầu vào càng rõ ràng, chúng ta càng dễ chọn. Ngoài ra việc chọn  $K$  đôi khi không chính xác, dẫn đến đầu ra sai. Lúc này nên chạy nhiều lần thuật toán với các giá trị  $K$  khác nhau để tìm  $K$  phù hợp dựa vào hàm **mất mát**. (Hàm mất mát là gì ? Cả một câu chuyện dài :D )

# Ngôn ngữ Python

## **3. Python với machine learning:**

### **3.1 Thuật toán phân cụm (clustering):**

**-> Demo**

# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.2 Thuật toán phân loại (classify):

- Bài toán phân loại cũng là một bài toán quan trọng trong data mining. Giả sử, website của bạn có rất nhiều lượt comment, ngoài những comment tích cực mang tính xây dựng, có rất nhiều comment tiêu cực (nội dung xấu, sử dụng các từ tục tĩu...). Bài toán đặt ra đó là bằng cách nào đó cần lọc tự động các comment tiêu cực đó.

-> Trong trường hợp này, các thuật toán phân loại cần được nghĩ tới. Một trong những thuật toán đơn giản, nhưng cực kỳ hiệu quả đó là thuật toán Naive Bayes. Công thức toán học của thuật toán đó chính là xác suất Bayes. Công thức toán học:

*(Naive tức là ngây thơ, còn Bayes là tên nhà toán học đã tìm ra định lý)*

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{\text{likelihood} * \text{prior}}{\text{normalizing\_constant}}$$

# Ngôn ngữ Python

## 3. Python với machine learning:

### 3.2 Thuật toán phân loại (classify):

- Xác suất Bayes nói rằng: Nếu ta có 2 sự kiện A, B thỏa mãn các điều kiện sau:
  - >  $P(A)$  xảy ra ngẫu nhiên không liên quan tới  $P(B)$ .
  - >  $P(B)$  xảy ra ngẫu nhiên không liên quan tới  $P(A)$ .
  - >  $P(B/A)$  là xác suất xảy ra B khi biết A. Từ đó xác suất  $P(A/B)$  có thể tính được như công thức slide trước.
- Trên thực tế tìm được các điều kiện như trên là gần như không thể bởi vì trên thực tế chúng ta rất ít khi tìm được một tập dữ liệu mà các thành phần của nó không liên quan gì đến nhau. Và ý tưởng này khá ngây thơ (Naive): ***chúng ta có thể tính toán một xác suất chưa biết dựa vào các xác suất có điều kiện khác***
- Thế nhưng khi áp dụng vào ML thì thuật toán này chạy tốt bất ngờ.

# Ngôn ngữ Python

## **3. Python với machine learning:**

### **3.2 Thuật toán phân loại (classify):**

**-> Demo**

# Ngôn ngữ Python

## 4. Python với web:

-Như trong ví dụ demo với machine learning, framework Flask được sử dụng để viết API. Ngoài ra nếu cần support nhiều hơn để phát triển 1 website thì Django là một lựa chọn tuyệt vời. Chi tiết vui lòng xem tại trang chủ: <https://www.djangoproject.com/start/>