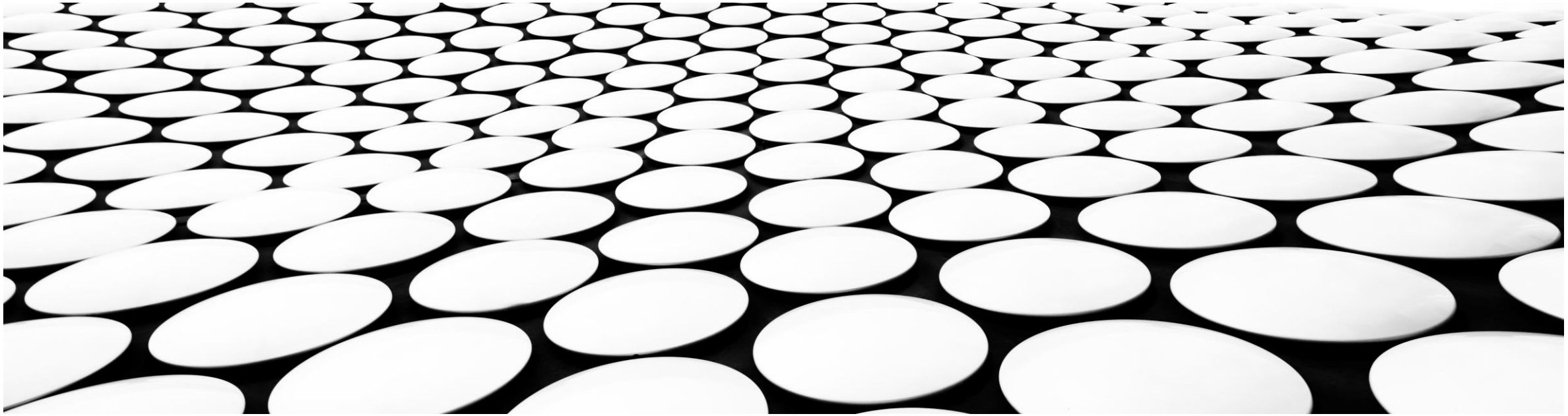
KĨ THUẬT LẬP TRÌNH PYTHON

NGUYỄN MẠNH HÙNG



CHƯƠNG 3: CÁC CẤU TRÚC DỮ LIỆU

1. CHUỖI VÀ PHƯƠNG THỨC TRÊN CHUỖI <*TIẾP THEO*>

a. Tìm kiếm trong chuỗi:

- Chương trình sau đây xác định xem một kí tự nào đó có xuất hiện trong một chuỗi hay không.

```
st=input("Nhập chuỗi: ")
letter=input("Nhập kí tự tìm kiếm: ")
index=0
while index<len(st):
    if st[index]==letter:
        print("Kí tự nằm trong chuỗi!")
        break
    index+=1
if index==len(st):
    print("Kí tự không nằm trong chuỗi")
```

```
Nhập chuỗi: Hà Nội mùa vắng những cơn mưa
Nhập kí tự tìm kiếm: b
Kí tự không nằm trong chuỗi
```

- Toán tử **in**: là toán tử logic kiểm tra xem một chuỗi có phải là chuỗi con của một chuỗi khác hay không.

```
st=input("Nhập chuỗi: ")
st1=input("Nhập chuỗi tìm kiếm: ")
if st1 in st:
    print("Là chuỗi con!")
else:
    print("Không là chuỗi con!")
```

```
Nhập chuỗi: Hà Nội mùa vắng những cơn mưa
Nhập chuỗi tìm kiếm: Hà Nội
Là chuỗi con!
```

Chuỗi là một **đối tượng** (*object*) trong Python. Một đối tượng sẽ chứa dữ liệu và các **phương thức** (*method*), là các hàm được xây dựng trên đối tượng và có thể sử dụng cho bất kì bản sao nào của đối tượng đó.

- Phương thức **find**:
  - **Cú pháp**: <biến chuỗi>**.find**(*st1*)
  - **Kết quả**: Phương thức trả về vị trí chỉ số đầu tiên mà chuỗi *st1* xuất hiện trong <biến chuỗi>. Nếu *st1* không có trong <biến chuỗi> thì trả về kết quả -1.
  - **Tham số** với phương thức **find**:

<biến chuỗi>**.find**(*st1,i*) : trả về vị trí chỉ số đầu tiên mà chuỗi *st1* xuất hiện trong <biến chuỗi> tính từ chỉ số *i*.

<biến chuỗi>**.find**(*st1,i,j*) : trả về vị trí chỉ số đầu tiên mà chuỗi *st1* xuất hiện trong <biến chuỗi> tính từ chỉ số *i* đến chỉ số *j-1*.

```python
st=input("Nhập chuỗi: ")
st1=input("Nhập chuỗi tìm kiếm: ")
index=st.find(st1)
if index!=-1:
    print("Chuỗi con bắt đầu từ chỉ số:",index)
else:
    print("Không là chuỗi con!")
```

```
Nhập chuỗi: Hà Nội Hải Phòng Đà Nẵng Sài Gòn
Nhập chuỗi tìm kiếm: Đà Nẵng
Chuỗi con bắt đầu từ chỉ số: 17
```

b. Tách và ghép chuỗi:

- Phương thức **split**:
  - **Cú pháp**: <biến chuỗi>**.split**()
  - **Kết quả**: chuỗi được tách thành danh sách các chuỗi con, kí tự phân tách là kí tự trắng.
  - **Tham số** với phương thức **split**:

  <biến chuỗi>**.split**(<kí tự phân tách>) : tách chuỗi theo kí tự phân tách.

  <biến chuỗi>**.split**(<kí tự phân tách>, <num>) : tách chuỗi theo kí tự phân tách với số chuỗi con = <num> + 1.

- Phương thức **join**:
  - **Cú pháp**: <biến chuỗi>**.join**(<danh sách>)
  - **Kết quả**: trả về chuỗi bằng cách ghép nối các chuỗi con trong một danh sách, các phần tử của danh sách được phân cách bởi <biến chuỗi>.

```python
st="Ngôn ngữ Python#C#Java rất phổ biến"
print("Tách chuỗi mặc định:")
print(st.split())
print("Nối chuỗi")
print(" ".join(st.split()))
print("Tách chuỗi bởi kí tự \"#\" : ")
print(st.split("#"))
print("Tách chuỗi với số chuỗi con = 2 : ")
print(st.split("#",1))
```

```
Tách chuỗi mặc định:
['Ngôn', 'ngữ', 'Python#C#Java', 'rất', 'phổ', 'biến']
Nối chuỗi
Ngôn ngữ Python#C#Java rất phổ biến
Tách chuỗi bởi kí tự "#" :
['Ngôn ngữ Python', 'C', 'Java rất phổ biến']
Tách chuỗi với số chuỗi con = 2 :
['Ngôn ngữ Python', 'C#Java rất phổ biến']
```

c. Chuyển đổi kí tự hoa, kí tự thường:

- Phương thức **upper**:
  - **Cú pháp**: <biến chuỗi>**.upper**()
  - **Kết quả**: chuyển đổi các kí tự thành kí tự hoa
- Phương thức **lower**:
  - **Cú pháp**: <biến chuỗi>**.lower**()
  - **Kết quả**: chuyển đổi các kí tự thành kí tự thường
- Phương thức **capitalize**:
  - **Cú pháp**: <biến chuỗi>**.capitalize**()
  - **Kết quả**: chuyển đổi kí tự đầu tiên thành kí tự hoa
- Phương thức **title**:
  - **Cú pháp**: <biến chuỗi>**.title**()
  - **Kết quả**: chuyển đổi kí tự đầu tiên của mỗi từ thành kí tự hoa
- Phương thức **swapcase**:
  - **Cú pháp**: <biến chuỗi>**.swapcase**()
  - **Kết quả**: chuyển đổi các kí tự thường thành kí tự hoa, kí tự hoa thành kí tự thường.

```python
st="HOc lập tRình pYthoN"
print(st)
print(st.swapcase())
print(st.upper()+" ; "+st.lower())
print(st.capitalize())
print(st.title())
```

```
HOc lập tRình pYthoN
hoC LẬP TrÌNH PyTHOn
HOC LẬP TRÌNH PYTHON ; học lập trình python
Hoc lập trình python
Hoc Lập Trình Python
```

d. Đếm số lần xuất hiện:

- Phương thức **count**:
  - **Cú pháp**: <biến chuỗi>**.count**(*st1*)
  - **Kết quả**: đếm số lần xuất hiện chuỗi *st1* trong <biến chuỗi>.
  - **Tham số** với phương thức **count**:

<biến chuỗi>**.count**(*st1,i*) : đếm số lần xuất hiện chuỗi *st1* trong <biến chuỗi> tính từ chỉ số *i*.

<biến chuỗi>**.count**(*st1,i,j*) : đếm số lần xuất hiện chuỗi *st1* trong <biến chuỗi> tính từ chỉ số *i* đến chỉ số *j-1*.

```python
st=input("Nhập chuỗi: ")
print("Số kí tự \"h\" trong chuỗi: ",st.count("h"))
print("Số kí tự \"h\" từ chỉ số 5: ",st.count("h",5))
print("Số kí tự \"h\" từ chỉ số 5 đến 15: ",st.count("h",5,16))
```

```
Nhập chuỗi: Tự học lập trình Python trong 21 ngày
Số kí tự "h" trong chuỗi:  3
Số kí tự "h" từ chỉ số 5:  2
Số kí tự "h" từ chỉ số 5 đến 15:  1
```

e. Thay thế:

- Phương thức **replace**:
  - **Cú pháp**: <biến chuỗi>**.replace**(*str_cũ,str_mới*)
  - **Kết quả**: thay thế tất cả chuỗi con *str_cũ* bởi *str_mới* trong <biến chuỗi>.
  - **Tham số** với phương thức **replace**:

<biến chuỗi>**.replace**(*str_cũ,str_mới,num*) : thay thế các chuỗi con *str_cũ* bởi *str_mới* trong <biến chuỗi> với số lần tối đa = *num*.

```
st="ĐHGTVT ở số 3 Cầu Giấy"
st.replace("ĐH","Đại học ")
```

```
'Đại học GTVT ở số 3 Cầu Giấy'
```

f. Xóa kí tự:

- Phương thức **lstrip**:
  - **Cú pháp**: <biến chuỗi>**.lstrip**()
  - **Kết quả**: xóa tất cả các kí tự trắng ở đầu/bên trái <biến chuỗi>.
- Phương thức **rstrip**:
  - **Cú pháp**: <biến chuỗi>**.rstrip**()
  - **Kết quả**: xóa tất cả các kí tự trắng ở cuối/bên phải <biến chuỗi>.
- Phương thức **strip**:
  - **Cú pháp**: <biến chuỗi>**.strip**()
  - **Kết quả**: xóa tất cả các kí tự trắng ở bên trái và bên phải <biến chuỗi>.
- **Tham số** với các phương thức xóa kí tự:
  - **Cú pháp**:

    <biến chuỗi>**.lstrip**([các kí tự])

    <biến chuỗi>**.rstrip**([các kí tự])

    <biến chuỗi>**.rstrip**([các kí tự])
  - **Kết quả**: xóa các kí tự thuộc tập [các kí tự] nằm tương ứng ở bên trái, bên phải, và hai bên của <biến chuỗi>.

```python
st="#đây#là#dòng#chú#thích#"
print(st)
print(st.lstrip("#"))
print(st.rstrip("#"))
print(st.strip("#"))
```

```
#đây#là#dòng#chú#thích#
đây#là#dòng#chú#thích#
#đây#là#dòng#chú#thích
đây#là#dòng#chú#thích
```

g. Căn chỉnh chuỗi:

- Phương thức **ljust**:
  - **Cú pháp**: <biến chuỗi>**.ljust**(width)
  - **Kết quả**: trả về chuỗi có độ dài width bằng cách thêm kí tự trắng vào bên phải <biến chuỗi>.
- Phương thức **rjust**:
  - **Cú pháp**: <biến chuỗi>**.rjust**(width)
  - **Kết quả**: trả về chuỗi có độ dài width bằng cách thêm kí tự trắng vào bên trái <biến chuỗi>.
- Phương thức **center**:
  - **Cú pháp**: <biến chuỗi>**.center**(width)
  - **Kết quả**: trả về chuỗi có độ dài width bằng cách thêm kí tự trắng vào 2 bên sao cho <biến chuỗi> được căn ở giữa.
- **Tham số** với các phương thức xóa kí tự:
  - **Cú pháp**:

    <biến chuỗi>**.ljust**(width,[kí tự thêm vào])

    <biến chuỗi>**.rjust**(width,[kí tự thêm vào])

    <biến chuỗi>**.center**(width,[kí tự thêm vào])
  - **Kết quả**: trả về chuỗi có độ dài width bằng cách chèn tương ứng [kí tự thêm vào] ở bên trái, bên phải, và hai bên của <biến chuỗi>.

```
st="hướng dẫn"
print(st)
print(st.ljust(15,"-"))
print(st.rjust(15,"+"))
print(st.center(15,"*"))
```

```
hướng dẫn
hướng dẫn------
++++++hướng dẫn
***hướng dẫn***
```

h. Một số phương thức is___

- Phương thức **isdigit**:
  - **Cú pháp**: <biến chuỗi>**.isdigit**()
  - **Kết quả**: trả về *True* nếu <biến chuỗi> chỉ chứa kí tự số và *False* nếu ngược lại.
- Phương thức **isspace**:
  - **Cú pháp**: <biến chuỗi>**.isspace**()
  - **Kết quả**: trả về *True* nếu <biến chuỗi> chỉ chứa kí trắng và *False* nếu ngược lại.
- Phương thức **islower**:
  - **Cú pháp**: <biến chuỗi>**.islower**()
  - **Kết quả**: trả về *True* nếu <biến chuỗi> chỉ chứa kí tự chữ thường và *False* nếu ngược lại.
- Phương thức **isupper**:
  - **Cú pháp**: <biến chuỗi>**.isupper**()
  - **Kết quả**: trả về *True* nếu <biến chuỗi> chỉ chứa kí tự chữ hoa và *False* nếu ngược lại.
- Phương thức **isalpha**:
  - **Cú pháp**: <biến chuỗi>**.isalpha**()
  - **Kết quả**: trả về *True* nếu <biến chuỗi> chỉ chứa kí tự ‘a’, ..., ‘z’, ‘A’, ..., ‘Z’ và *False* nếu ngược lại.
- Phương thức **istitle**:
  - **Cú pháp**: <biến chuỗi>**.istitle**()
  - **Kết quả**: trả về *True* nếu <biến chuỗi> có chữ cái đầu của các từ đều là chữ hoa và *False* nếu ngược lại.

k. Một số hàm thao tác chuỗi

- Hàm **max**(), **min**():
  - **Cú pháp**: **max**(<biến chuỗi>),
    min(<biến chuỗi>)
  - **Kết quả**: trả về kí tự lớn nhất và nhỏ nhất trong <biến chuỗi>.
- Hàm **eval**():
  - **Cú pháp**: **eval**(<biến chuỗi>)
  - **Kết quả**: trả về giá trị của biểu thức toán học đúng được viết dưới dạng chuỗi.
- Hàm **sorted**():
  - **Cú pháp**: **sorted**(<biến chuỗi>)
  - **Kết quả**: trả về danh sách các kí tự trong <biến chuỗi> đã được sắp thứ tự.

```
from math import *
st="1+sqrt(2)**4"
print("Giá trị của biểu thức: ",eval(st))
print("Kí tự lớn nhất là: ",max(st))
print("Kí tự nhỏ nhất là: ",min(st))
```

```
Giá trị của biểu thức:  5.000000000000001
Kí tự lớn nhất là:  t
Kí tự nhỏ nhất là:  (
```

```
st="akskkgki"
sorted(st)
```

```
['a', 'g', 'i', 'k', 'k', 'k', 'k', 's']
```

Thực hành:

- **Bài tập 1.1**: Viết chương trình nhập vào một chuỗi. Chương trình sẽ kiểm tra xem chuỗi có chứa kí tự số hay không, nếu có thì tính tổng các chữ số trong chuỗi.
- **Bài tập 1.2**: Hai chuỗi *st1* và *st2* cùng độ dài được gọi là hai chuỗi hoán vị của nhau nếu có thể đổi chỗ các kí tự trong chuỗi *st1* để trở thành chuỗi *st2*. Viết một chương trình nhập vào hai chuỗi và trả lời xem hai chuỗi có phải là hoán vị của nhau hay không.

- **Bài tập 1.3**: Cho một chuỗi st gồm các kí tự chữ cái La-tinh thường. Viết một chương trình thực hiện nén chuỗi theo quy tắc như sau: với một dãy các kí tự giống nhau liên tiếp, ta sẽ thay thế dãy này bằng: <kí tự><số lần xuất hiện>.

  Minh họa: st="aabcccddddda" được nén thành "a2bc3d4a"

- **Bài tập 1.4**: Viết một chương trình đọc chuỗi:

  st="'X-DSPAM-Confidence:0.8475"

Hãy tìm và trích xuất ra chuỗi con đứng sau dấu hai chấm (:) và chuyển đổi thành dữ liệu số.

- **Bài tập 1.5**: Viết chương trình nhập vào một chuỗi gồm các kí tự số. Hãy chén một kí tự “+” và một kí tự “=” (dấu + đứng trước dấu =) vào giữa chuỗi kí tự để được một biểu thức toán học đúng và đưa biểu thức ra màn hình. Nếu không có cách đặt các kí tự “+” và “=” thì đưa ra thông báo “Không có cách đặt phù hợp!”

  Minh họa: 123 ➔ 1+2=3 *144 ➔ “Không có cách đặt phù hợp!”*

- **Bài tập 1.6**: Mật mã Caesar là một dạng mật mã thay thế đơn giản. Mỗi kí tự trong văn bản sẽ được thay thế bằng một kí tự cách nó một khoảng không đổi **d** trong bảng chữ cái, và do đó văn bản gốc sẽ biến thành văn bản mã hóa.

  *Ví dụ: **d** = 2 thì a ➔ c, b ➔ d, ...*

Giá trị **d** được gọi là khóa. Khi biết khóa **d** thì từ văn bản mã hóa, ta có thể tìm được văn bản gốc ban đầu.

Hãy viết một chương trình nhập vào chuỗi st, nhập vào khóa d. Chương trình sẽ đưa ra chuỗi được mã hóa.

2. KIỂU DỮ LIỆU DANH SÁCH (LIST)

- **Khai báo kiểu dữ liệu danh sách**: Danh sách (list) là một tập hợp các phần tử. Để khai báo một danh sách, ta sử dụng cú pháp:

  <*biến danh sách*> = **[** phần tử 0**,** phần tử 1**, …,** phần tử n-1 **]**

- **Ví dụ**:

```
list1 = [ ]                                  # danh sách rỗng
list2 = [‘xanh’, ‘đỏ’, ‘tím’, ‘vàng’]        # danh sách gồm 4 chuỗi là tên các màu
list3 = [ 1, 2, 3, 4, 5, 6, 7, 8, 9]        # danh sách gồm 9 số nguyên
```

- **Lấy số phần tử của danh sách**: sử dụng hàm **len(**<biến danh sách>**)**

  Chẳng hạn: **len**(list1) = 0 **len**(list2) = 4 **len**(list3) = 9

- **Truy cập từng phần tử**:
  - **Cú pháp**: <*biến danh sách*>**[**chỉ số**]**
  - **Ví dụ**: *lst* = [‘xanh’, ‘đỏ’, ‘tím’, ‘vàng’] ➔ *lst*[0] = ‘xanh’, *lst*[1] = ‘đỏ’, …
  - Nếu chỉ số không tồn tại trong danh sách thì trình biên dịch sẽ báo lỗi **IndexError**.
  - Các phần tử trong danh sách có thể thay đổi: <*biến danh sách*>**[**chỉ số**]** = <*giá trị mới*>

- **Lấy danh sách con**:
  - **Cú pháp**:

    <*biến danh sách*>**[**a:b**]** : trả về danh sách các phần tử từ chỉ số a đến b-1.

    <*biến danh sách*>**[**:b**]** : trả về danh sách các phần tử từ chỉ số 0 đến b-1.

    <*biến danh sách*>**[**a:**]** : trả về danh sách các phần tử từ chỉ số a đến hết danh sách.

    <*biến danh sách*>**[**a:b:c**]** : trả về danh sách các phần tử có chỉ số a, a+c, ..., a+tc<b trong danh sách.

  - **Ví dụ**: *lst* = [‘xanh’, ‘đỏ’, ‘tím’, ‘vàng’]

    lst[1:3] = [‘đỏ’, ‘tím’] *lst*[2:] = [‘tím’, ‘vàng’]

    lst[:2] = [‘xanh’, ‘đỏ’] *lst*[::2] = [‘xanh’, ‘tím’]

- **Ghép nối** danh sách và **lặp** danh sách:
  - **Cú pháp**:

    <danh sách 1> + *<danh sách 2>* + ... + *<danh sách n>* : trả về một danh sách gồm các phần tử của các danh sách con liên tiếp nhau.

    *<num>***<danh sách>* hoặc *<danh sách>***<num>* : trả về danh sách bằng cách ghép *<num>* lần *<danh sách>*

  - **Ví dụ**: *lst1* = [‘Học’, ‘vui’] *lst2* = [‘Python’, 2020]

    lst1 + *lst2* = [‘Học’, ‘vui’, ‘Python’, 2020]

    2**lst2* = [‘Python’, 2020, ‘Python’, 2020]

- Phương thức **append**:
  - **Cú pháp**: <*danh sách*>**.append**(x)
  - **Kết quả**: Thêm phần tử x vào cuối <*danh sách*>

- Phương thức **extend**:
  - **Cú pháp**: <*ds1*>**.extend**(<*ds2*>)
  - **Kết quả**: Nối <*ds2*> vào cuối <*ds1*>

```
lst=['Văn Anh','Bảo','Khang']
print("Danh sách gốc: ")
print(lst)
lst_1=['Hạnh','Mai']
lst_1.append('Quyền')
lst.extend(lst_1)
print("Danh sách mới: ")
print(lst)
```

```
Danh sách gốc:
['Văn Anh', 'Bảo', 'Khang']
Danh sách mới:
['Văn Anh', 'Bảo', 'Khang', 'Hạnh', 'Mai', 'Quyền']
```

- Toán tử logic **in** và **not in**:
  - <*phần tử*> **in** <*danh sách*> : trả về giá trị **True** nếu <*phần tử*> thuộc <*danh sách*> và trả về giá trị **False** nếu ngược lại.
  - <*phần tử*> **not in** <*danh sách*> : trả về giá trị **True** nếu <*phần tử*> không thuộc <*danh sách*> và trả về giá trị **False** nếu ngược lại.

```
lst=[1,2,4,6]
sm=0
for i in lst:
    sm+=i
print(sm)
```

```
13
```

- Phương thức **index**:
  - **Cú pháp**: <*biến danh sách*>**.index**(x)
  - **Kết quả**: trả về chỉ số của phần tử x đầu tiên trong danh sách
  - **Tham số** với phương thức **index**: <*biến danh sách*>**.index**(x,i) - trả về chỉ số của phần tử x đầu tiên trong danh sách tính từ chỉ số i.

```
lst=['Văn Anh','Bảo','Khang','Hạnh','Mai','Quyền']
lst.index('Mai')
```

```
4
```

- Một số **hàm** thao tác trên danh sách:

Hàm **min**, **max**:

- **Cú pháp**: **min**(<*danh sách*>)

  max(<*danh sách*>)

- **Kết quả**: trả về giá trị nhỏ nhất và lớn nhất trong danh sách.

```
lst=['Văn Anh','Bảo','Khang','Hạnh','Mai','Quyền']
print(min(lst),max(lst),sep=' ; ')
```

```
Bảo ; Văn Anh
```

Hàm **sum**:

- **Cú pháp**: **sum**(*<danh sách>*)
- **Kết quả**: trả về tổng giá trị các phần tử nằm trong danh sách.
- **Chú ý**: *<danh sách>* chỉ bao gồm các số.

```
lst=[1,2,4,6]
sum(lst)
```

```
13
```

```
lst=['Văn Anh','Bảo','Khang','Hạnh','Mai','Quyền']
sum(lst)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-19-c6e7e8ae1418> in <module>
      1 lst=['Văn Anh','Bảo','Khang','Hạnh','Mai','Quyền']
----> 2 sum(lst)

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Thực hành:

- **Bài tập 2.1**: Viết chương trình nhập vào một danh sách gồm các số và một số nguyên k. Chương trình sẽ tìm và in ra dãy gồm k số liên tiếp trong danh sách sao cho tổng các số này là lớn nhất.

  Ví dụ: Dãy [1, 2, -19, 3, 4, -1] thì dãy con gồm k = 3 phần tử có tổng lớn nhất là [3, 4, -1].

- **Bài tập 2.2**: Viết một chương trình nhập vào một danh sách gồm các số. Chương trình sẽ tìm và in ra số dương nhỏ nhất trong danh sách cùng với chỉ số của nó. Nếu danh sách không chứa số dương nào thì in ra thông báo "Không có số dương nào trong danh sách!"

  Ví dụ: Dãy [0, -2, 1, 3, 2, -1] thì số dương nhỏ nhất = 1 và chỉ số = 2.

- **Bài tập 2.3**: Viết một chương trình tạo một danh sách gồm các số thực được nhập lần lượt từ bàn phím cho đến khi không có kí tự nào được nhập thì dừng. Nếu nhập vào một số chưa có trong danh sách thì bổ sung vào danh sách, ngược lại thì không bổ sung vào danh sách. Chương trình sẽ in ra danh sách các số và tính giá trị trung bình.

Minh họa: Nhập số: 2.1

Nhập số: 1.3

Nhập số: 2.1

Nhập số: 3.4

Nhập số: # bỏ trống để thoát

Danh sách được tạo: [2.1, 1.3, 3.4]

Giá trị trung bình: 2.26666666

3. MỘT SỐ VẤN ĐỀ NÂNG CAO VỀ DANH SÁCH (LIST)

a. Khởi tạo danh sách

- Sử dụng vòng lặp **for**:

[<*biểu thức*> **for** <*phần tử*> **in** <*tập giá trị*> **]** : tạo một danh sách gồm các phần tử là giá trị của <*biểu thức*> khi <*phần tử*> lần lượt nhận giá trị trong <*tập giá trị*>

[<*biểu thức*> **for** <*phần tử*> **in** <*tập giá trị*> **if** <*điều kiện*> **]** : tạo một danh sách gồm các phần tử là giá trị của <*biểu thức*> khi <*phần tử*> lần lượt nhận giá trị trong <*tập giá trị*> và thỏa mãn <*điều kiện*>

```
lst1=[i for i in range(10) if i%2!=0]
lst2=[2*i+1 for i in (5,6,7,8,9)]
lst3=[(i,j) for i in range(3) for j in range(3) if i+j<=2]
print(lst1)
print(lst2)
print(lst3)
```

```
[1, 3, 5, 7, 9]
[11, 13, 15, 17, 19]
[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (2, 0)]
```

- Sử dụng hàm tạo **list**:
  - **Cú pháp**: **list**(<*dãy*>)
  - **Kết quả**: Tạo danh sách từ một <*dãy*> nào đó. Hàm list sẽ biến một bộ (tuple) hay một chuỗi (str) thành một danh sách.
- Phương thức **count**:
  - **Cú pháp**: <*biến danh sách*>**.count**(x)
  - **Kết quả**: trả về số lần xuất hiện của phần tử x có trong danh sách

```
st="Python learning"
lst=list(st)
print(lst)
x='n'
print("Số kí tự %s trong chuỗi là: %s" % (x,lst.count(x)))
x='k'
print("Số kí tự %s trong chuỗi là: %s" % (x,lst.count(x)))
```

```
['P', 'y', 't', 'h', 'o', 'n', ' ', 'l', 'e', 'a', 'r', 'n', 'i', 'n', 'g']
Số kí tự n trong chuỗi là: 3
Số kí tự k trong chuỗi là: 0
```

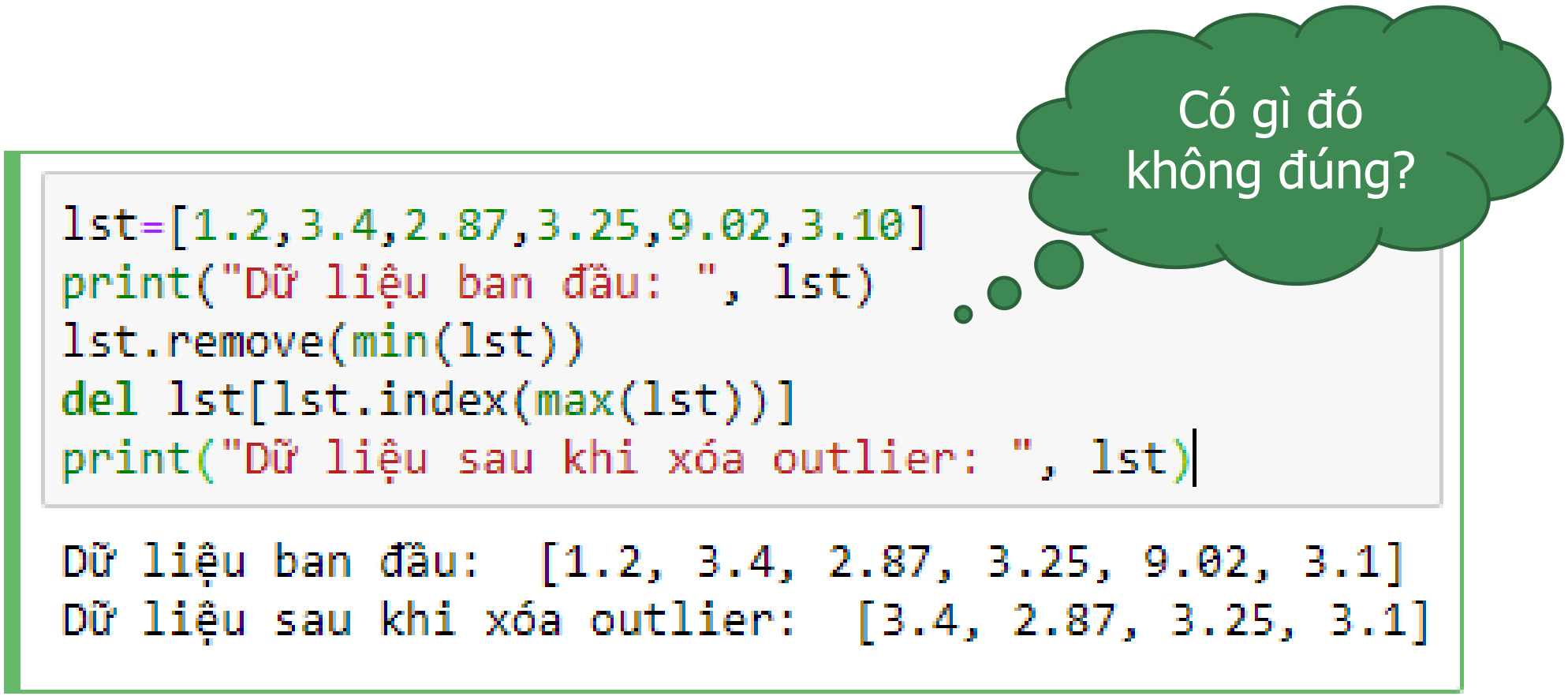
b. Xóa phần tử khỏi danh sách

- Phương thức **remove**:
  - **Cú pháp**: <*danh sách*>**.remove**(x)
  - **Kết quả**: Xóa phần tử x xuất hiện đầu tiên trong danh sách
- Phương thức **pop**:
  - **Cú pháp**: <*danh sách*>**.pop**()
  - **Kết quả**: Xóa phần tử cuối cùng của danh sách
- Phương thức **clear**:
  - **Cú pháp**: <*danh sách*>**.clear**()
  - **Kết quả**: Xóa tất cả các phần tử trong danh sách
- Hàm **del**:
  - **Cú pháp**: **del** <*danh sách*>[*index*]
  - **Kết quả**: Xóa phần tử có chỉ số *index* trong danh sách
  - **Tham số** của hàm **del**:

  del <*danh sách*>[a:b] : xóa các phần tử từ chỉ số a đến b-1 trong danh sách

  del <*danh sách*>[a:b:c] : xóa các phần tử có chỉ số a, a+c, ..., a+tc<b trong danh sách

Làm sạch dữ liệu bằng cách xóa giá trị lớn nhất, nhỏ nhất



c. Chèn phần tử vào danh sách

- Phương thức **insert**:
  - **Cú pháp**: <*danh sách*>**.insert**(i,x)
  - **Kết quả**: Chèn phần tử x vào vị trí có chỉ số i trong danh sách

d. Sắp xếp các phần tử trong danh sách

- Phương thức **sort**:
  - **Cú pháp**: <*danh sách*>**.sort**()
  - **Kết quả**: Sắp xếp các phần tử trong danh sách theo thứ tự tăng dần
  - **Chú ý**: để sắp xếp các phần tử trong danh sách theo thứ tự giảm dần, ta sử dụng:

    <*danh sách*>**.sort**(reverse=True)

- Hàm **sorted**:
  - **sorted**(<*danh sách*>): trả về danh sách gồm các phần tử được sắp xếp tăng dần.
  - **sorted**(<*danh sách*>, reverse=True): trả về danh sách được sắp xếp giảm dần.
- Phương thức <danh sách>**.reverse**() : đảo ngược thứ tự danh sách

```
lst=[1.2,3.4,2.87,3.25,9.02]
lst.insert(2,3.98)
print("Chèn vào danh sách: ", lst)
lst.sort()
print("Sắp xếp tăng dần: ", lst)
lst.reverse()
print("Sắp xếp giảm dần: ", lst)
```

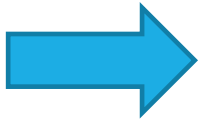
```
Chèn vào danh sách:  [1.2, 3.4, 3.98, 2.87, 3.25, 9.02]
Sắp xếp tăng dần:  [1.2, 2.87, 3.25, 3.4, 3.98, 9.02]
Sắp xếp giảm dần:  [9.02, 3.98, 3.4, 3.25, 2.87, 1.2]
```

e. Ma trận - danh sách 2 chiều

- Trong Python, các phần tử trong danh sách có thể là một danh sách. Có thể biểu diễn ma trận bằng một danh sách mà mỗi phần tử của nó là một danh sách gồm các phần tử trên một dòng của ma trận.

Ví dụ:

1	4	-5
2	1	2
-3	1	4



list = [[1,4,-5], [2,1,2], [-3,1,4]]

- Để truy cập đến một phần tử nào đó trong bảng, ta sử dụng cú pháp:

<danh sách>[chỉ số dòng][chỉ số cột]

- **Ví dụ**: Viết chương trình nhập vào một ma trận và in ra ma trận đó.

```
Nhập số hàng: 2
Nhập số cột: 3
Nhập các phần tử hàng  1
1
-2
4
Nhập các phần tử hàng  2
3
2
-5
Ma trận A là:
1 -2 4
3 2 -5
```

```python
# Nhập kích thước ma trận A
m=int(input("Nhập số hàng: "))
n=int(input("Nhập số cột: "))

# Nhập các phần tử của ma trận
A=[]
for i in range(m):
    print("Nhập các phần tử hàng ",i+1)
    row=[]
    for j in range(n):
        x=int(input())
        row.append(x)
    A.append(row)

# In ma trận A
print("Ma trận A là:")
for i in range(m):
    for j in range(n):
        print(A[i][j],end=' ')
    print()
```

- Có thể khởi tạo ma trận bằng vòng lặp **for** theo cú pháp:

```
[ [int(input()) for j in range(n)] for i in range(m) ]
```

```python
# Nhập kích thước ma trận A
m=int(input("Nhập số hàng: "))
n=int(input("Nhập số cột: "))

# Khởi tạo ma trận
A=[[int(input()) for j in range(n)] for i in range(m)]

# In ma trận A
print("Ma trận được nhập là:")
for x in A:
    print(x)
```

```
Nhập số hàng: 2
Nhập số cột: 3
1
-4
1
2
3
-5
Ma trận được nhập là:
[1, -4, 1]
[2, 3, -5]
```

g. Danh sách của các danh sách

- Phần tử của một danh sách cũng có thể là một danh sách:

  Ví dụ: lst = [“Python”, [3,0], [-3,1,4]]

- Ma trận hai chiều là một danh sách của các danh sách, ở đó mỗi phần tử là một hàng, và các hàng có số phần tử bằng nhau.

h. So sánh hai danh sách

- Python cho phép so sánh hai danh sách bằng cách lần lượt so sánh các phần tử từ chỉ số 0 đến hết danh sách. Khi gặp hai phần tử cùng chỉ số nhưng khác nhau thì danh sách nào chứa phần tử lớn hơn thì danh sách đó lớn hơn.

Input: lst1 = [“Python”, 3.0]

lst2 = [“Python”, 2.7]

lst1 > lst2

Output: True

- **Ví dụ**: Cho một danh sách, phần tử của danh sách cũng là danh sách. Viết một chương trình chuyển các phần tử của các danh sách thành phần thành phần tử thực sự của danh sách.

  Minh họa:

  Input: [[2], [‘a’, ‘b’, ‘c’], [-3, 1]]

  Output: [2, ‘a’, ‘b’, ‘c’, -3, 1]

```
lst=[[2],['a','b','c'],[-3,1]]
lst1=[]
for x in lst:
    lst1.extend(x)
print(lst1)
```

```
[2, 'a', 'b', 'c', -3, 1]
```

Thực hành:

- **Bài tập 3.1**: Khi xử lý dữ liệu, người ta thường loại bỏ các giá trị "cực trị" (extreme values), sau đó mới thực hiện tính toán. Hãy viết một chương trình nhập vào một danh sách các số thực, và một số nguyên dương ***n***. Chương trình sẽ tạo ra một danh sách mới, sau khi đã xóa ***n*** giá trị lớn nhất và ***n*** giá trị nhỏ nhất. Chương trình sẽ đưa ra thông báo lỗi nếu số phần tử bị xóa lớn hơn số phần tử của danh sách. Xét 2 trường hợp sau đây:
  - Thứ tự các phần tử trong danh sách mới giống như trong danh sách ban đầu.

    Chẳng hạn: [2, -1, 3, 7, -2, 1, 4], ***n*** *= 2* ➔ *[2, 3, 1]*
  - Thứ tự các phần tử trong danh sách mới không nhất thiết giống với danh sách ban đầu

    Chẳng hạn: [2, -1, 3, 7, -2, 1, 4], ***n*** *= 2* ➔ *[1, 2, 3]*

- **Bài tập 3.2**: Viết một chương trình tạo ra một danh sách gồm n số nguyên, bằng cách sinh ngẫu nhiên các số nằm trong đoạn [0,10]. Chương trình sẽ thống kê và in ra các phần tử phân biệt trong danh sách cùng với số lần xuất hiện.

Minh họa: *[1, 7, 4, 2, 2, 4, 1, 5 , 2]*

Số 1 xuất hiện 2 lần

Số 2 xuất hiện 3 lần

Số 4 xuất hiện 2 lần

Số 5 xuất hiện 1 lần

Số 7 xuất hiện 1 lần

- **Bài tập 3.3**: Viết một chương trình nhập vào các cặp số liệu thực (x,y), cho đến khi không có dữ liệu được nhập (bỏ trống). Chương trình sẽ in ra danh sách 2-chiều mô tả dữ liệu. Sau đó, tính toán và đưa ra công thức đường hồi quy tuyến tính phù hợp nhất với dữ liệu.

Minh họa: *Nhập x: 1*

Nhập y: 1

Nhập x: 2

Nhập y: 2.1

Nhập x: 3

Nhập y: 2.9

Nhập x: *(bỏ trống+gõ Enter)*

Dữ liệu: [[1,2,3], [1,2.1,2.9]]

Đường hồi quy tuyến tính: y = 0.95 x + 0.1

4. KIỂU DỮ LIỆU BỘ (TUPLE)

- Một **tuple** là một dãy các giá trị **khá giống** với **danh sách**.
- Các giá trị lưu trong tuple có kiểu tùy ý, được đánh chỉ số là các số nguyên.
- **Khai báo** kiểu tuple:

    <biến tuple> = **(***<phần tử 0>***,** *<phần tử 1>***,** …**,** *<phần tử n-1>***)**

  Nếu bộ chỉ có 1 phần tử:

    <biến tuple> = **(***<phần tử 0>***,)**

- **Ví dụ**:

```
t=('a','b','c',4,5)
t1=('a',)
# Thiếu dấu phẩy, t2 không phải 'tuple'
t2=('a')
print(t,t1,t2,sep="\n")
```

```
('a', 'b', 'c', 4, 5)
('a',)
a
```

Truy cập phần tử của tuple:

- **len**(<*biến tuple*>): trả về số phần tử của <*biến tuple*>
- <*biến tuple*>[*a*] : trả về phần tử có chỉ số *a* trong <*biến tuple*>
- <*biến tuple*>[*a*:*b*] : trả về một tuple gồm các phần tử có chỉ số *a, a+1, ..., b-1*
- <*biến tuple*>[:*b*] : trả về một tuple gồm *b* phần tử đầu tiên
- <*biến tuple*>[*a*:] : trả về một tuple gồm các phần tử có chỉ số *a, a+1, ...*

```
t=('a','b','c',4,5)
print("Độ dài tuple:",len(t))
print("P.tử đầu tiên:",t[0])
print("P.tử từ 1 đến 3:",t[1:4])
print("P.tử từ 3:",t[3:])
```

```
Độ dài tuple: 5
P.tử đầu tiên: a
P.tử từ 1 đến 3: ('b', 'c', 4)
P.tử từ 3: (4, 5)
```

- **Nối và lặp tuple**:

  <biến tuple 1> + ... + <biến tuple n>

  *n * <biến tuple>* **hoặc** *<biến tuple> * n*

- **tuple**() : trả về một tuple từ một danh sách hay một chuỗi.

```
st1='Learn'
st2='Pyth'
t1=tuple(st1)
t2=tuple(st2)
print(t1,t2,t1+t2,2*t2,sep="\n")
```

```
('L', 'e', 'a', 'r', 'n')
('P', 'y', 't', 'h')
('L', 'e', 'a', 'r', 'n', 'P', 'y', 't', 'h')
('P', 'y', 't', 'h', 'P', 'y', 't', 'h')
```

- **Packing** và **Unpacking**:

```
# Packing
t = 1,2,3
t1 = 'a',
# Unpacking
x,y,z = t
# Display
print(t,x,y,z,sep="\n")
print(t1)
```

```
(1, 2, 3)
1
2
3
('a',)
```

- <*biến tuple*>[**: :** -1] : đảo ngược thứ tự tuple.
- **max**(*tuple*), **min**(*tuple*): trả về phần tử có giá trị lớn nhất, nhỏ nhất trong tuple.
- Toán tử **in**: <*phần tử*> **in** <*tuple*>:

  trả về **True** nếu <*phần tử*> nằm trong <*tuple*>, và **False** nếu ngược lại.
- <*biến tuple*>**.count**(x): trả về số lần xuất hiện x trong <*biến tuple*>.
- <*biến tuple*>**.index**(x): trả về chỉ số đầu tiên của x trong <*biến tuple*>, nếu x không nằm trong <*biến tuple*> thì báo lỗi.
- **So sánh** các tuple: so sánh từng phần tử cùng chỉ số.

- **Gán** cùng lúc **nhiều giá trị** với **tuple**:

```
t=((1,2),(2,-4),(3,3))
for i,x in t:
    print(x**i)
```

```
2
16
27
```

```
t=((1,2),(2,-4),(3,3))
for x in t:
    print(x[1]**x[0])
```

```
2
16
27
```

Thực hành:

- **Bài tập 4.1**: Viết một chương trình thực hiện các chức năng sau:

  + Nhập vào dữ liệu mua các mã chứng khoán theo định dạng tuple:

    (Ngày mua, Giá mua, Số lượng cổ phiếu, Mã CK, Giá hiện tại)

  + In ra danh mục đầu tư (portfolio), ví dụ:

    portfolio= [("25-Jan-2001", 43.50, 25, 'CAT', 92.45),

    ("25-Jan-2001", 42.80, 50, 'DD', 51.19),

    ("25-Jan-2001", 42.10, 75, 'EK', 34.87),

    ("25-Jan-2001", 37.58, 100, 'GM', 37.58)]

  + Chương trình sẽ tính và đưa ra: tổng số tiền đầu tư, các mã CK có lãi, các mã CK bị lỗ, và tổng số tiều lãi/lỗ của danh mục đầu tư đó.

- **Bài tập 4.2**: Viết một chương trình:

  + Sinh ra *n* (*n* < *1e+3*) số nguyên ngẫu nhiên nằm trong khoảng [-10,10].

  + In ra danh sách gồm các tuple dạng (*giá trị, tần số*) theo thứ tự tăng dần giá trị, ví dụ:

    data = [(-6,3), (-3,2), (0,4), (1,3), (2,1)]

  + Mô tả thống kê dữ liệu bao gồm: giá trị trung bình, phương sai, độ lệch tiêu chuẩn, trung vị, mode, độ rộng.

- **Bài tập 4.3**: Viết một chương trình:

  + Nhập vào *n* cặp số thực (x_i, y_i), bỏ trống khi muốn dừng nhập dữ liệu.

  + In ra danh sách dữ liệu, ví dụ:

    data = [(1,1), (2,2.1), (3,2.9), (4,4)]

  + Tính và đưa ra đường hồi quy tuyến tính phù hợp nhất với dữ liệu.

5. KIỂU DỮ LIỆU TẬP HỢP (SET)

- **Khai báo** tập hợp:

  <biến tập hợp> = **{** *<phần tử 0>*, *<phần tử 1>*, …,

  <phần tử n-1> **}**

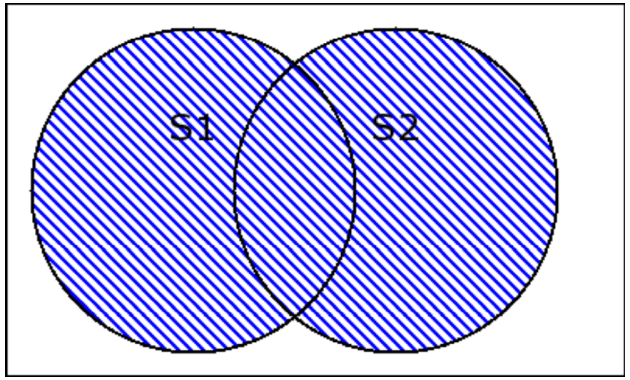
- Hàm **set**(): biến một dãy phần tử thành một tập hợp.
- Chú ý: các phần tử của một tập hợp **không thể trùng lặp**.

```
tupl=(1,'a',4,1,2)
st=set(tupl)
print(tupl)
print(st)
```

```
(1, 'a', 4, 1, 2)
{1, 2, 4, 'a'}
```

Các phép toán tập hợp:

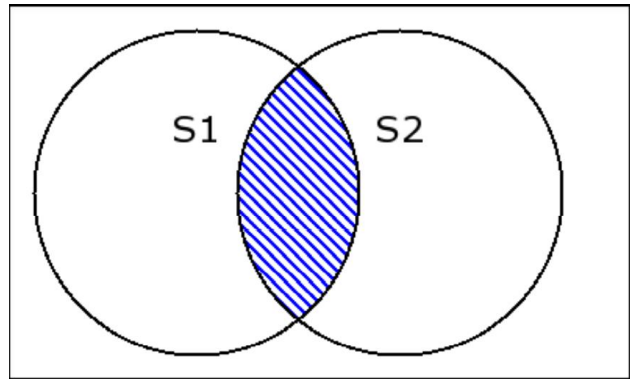
- **Hợp** của hai tập hợp S_1 và S_2
  - Phép toán: S_1 **|** S_2
  - Phương thức: S_1**.union**(S_2)




```
set1={1,2,3,4}
set2={1,3,5,7}
print(set1 | set2)
print(set1.union(set2))
```

```
{1, 2, 3, 4, 5, 7}
{1, 2, 3, 4, 5, 7}
```

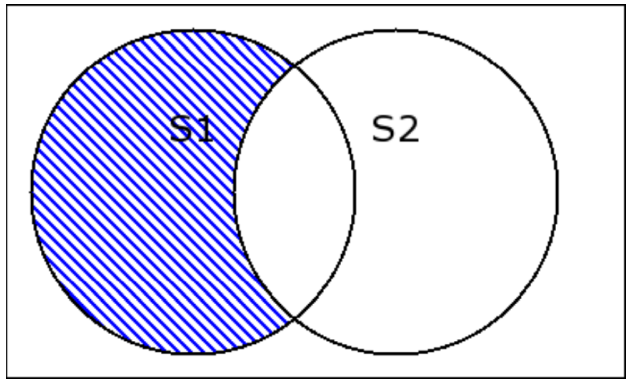
- **Giao** của hai tập hợp S_1 và S_2
  - Phép toán: S_1 **&** S_2
  - Phương thức: S_1**.intersection**(S_2)




```
print(set1 & set2)
print(set1.intersection(set2))
```

```
{1, 3}
{1, 3}
```

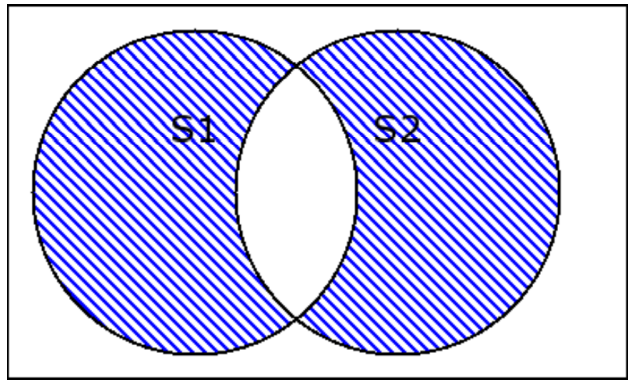
- **Hiệu** của hai tập hợp S_1 và S_2
  - Phép toán: S_1 **-** S_2
  - Phương thức: S_1**.difference**(S_2)




```
print(set1 - set2)
print(set1.difference(set2))
```

```
{2, 4}
{2, 4}
```

- **Hiệu đối xứng** của hai tập hợp S_1 và S_2
  - Phép toán: S_1 **^** S_2
  - Phương thức: S_1**.symmetric_difference**(S_2)




```
print(set1 ^ set2)
print(set1.symmetric_difference(set2))
```

```
{2, 4, 5, 7}
{2, 4, 5, 7}
```

Một số phương thức trên tập hợp:

- **clear**(): xóa tất cả các phần tử của *set.*
- **add**(*new*): thêm phần tử *new* vào *set.*
- **remove**(*old*), **discard**(*old*): xóa phần tử *old* khỏi *set*.
- **issubset**(*other*): trả về **True** nếu *set* là một tập con của *other*, ngược lại trả về **False**.
- **issuperset**(*other*): trả về **True** nếu *set* chứa *other*, ngược lại trả về **False**.

Thực hành:

- **Bài tập 5.1**: (Sàng Eratosthenes) Sàng Eratosthenes là một kỹ thuật để tìm các số nguyên tố từ 2 đến giới hạn nào đó. Mô tả thuật toán như sau:
  - Viết ra tập gồm các số từ 0 đến giới hạn

    Gạch bỏ 0 và 1 vì chúng không phải là số nguyên tố

    Đặt p bằng 2
  - Trong khi p nhỏ hơn giới hạn, lặp lại công việc sau:

    Gạch bỏ tất cả các bội số của p (nhưng không phải của chính p)

    Đặt p bằng số tiếp theo trong danh sách không bị gạch bỏ
  - In ra tập các số không bị gạch bỏ là số nguyên tố

  Hãy viết một chương trình thực hiện thuật toán sàng Eratosthenes với giới hạn được nhập từ bàn phím.

6. KIỂU DỮ LIỆU TỪ ĐIỂN (DICTIONARY)

- *Dictionary* là kiểu dữ liệu tương tự danh sách nhưng tổng quát hơn. Trong danh sách, chỉ số là số nguyên, nhưng trong *dictionary*, chỉ số có kiểu tùy ý.
- *Từ điển* có thể coi như một ánh xạ giữa tập chỉ số (gọi là ***key***) và một tập giá trị (gọi là ***value***)
- **Khai báo**:

  <biến từ điển> = **{***<key_0>***:***<value_0>***, …,** *<key_n>***:***<value_n>***}**

Key	Value
1	2
2	‘Toán’
3	‘Tin’

- Hàm **dict**(): tạo ra một từ điển rỗng

```
tu_dien=dict()
tu_dien
```

```
{}
```

- **Truy cập** *key* và *value* của từ điển:

```
for x in tu_dien:
    print(x, end =", ")
```

```
1, 2, 3,
```

```
for x in tu_dien:
    print(tu_dien[x], end =", ")
```

```
2, Toán, Tin,
```

- **Thêm, sửa** một giá trị cho từ điển:

```
tu_dien[1]=2
tu_dien[2]='Toán'
tu_dien[3]='Tin'
print(tu_dien)
tu_dien[1]='K59'
tu_dien
```

```
{1: 2, 2: 'Toán', 3: 'Tin'}

{1: 'K59', 2: 'Toán', 3: 'Tin'}
```

- Phương thức **values**():

```
Dict={1:'Toán', 2:'Ứng dụng', 3:'K59'}
for x in Dict.values():
    print(x, end=', ')
```

```
Toán, Ứng dụng, K59,
```

- Phương thức **get**(<*key*>):

```
print(Dict.get(1))
print(Dict.get(4))
```

```
Toán
None
```

```
print(Dict.get(5,'Không tồn tại'))
Dict.get(1,'Không tồn tại')
```

```
Không tồn tại
'Toán'
```

- Phương thức **items**():

```
for x in Dict.items():
    print(x)
```

```
(1, 'Toán')
(2, 'Ứng dụng')
(3, 'K59')
```

```
for x,y in Dict.items():
    print(x,':',y)
```

```
1 : Toán
2 : Ứng dụng
3 : K59
```

- Phương thức **pop**(): xóa phần tử theo khóa khỏi từ điển,

  <*biến từ điển*>**.pop**(<*key*>)

- Phương thức **popitem**(): xóa phần tử được đưa vào sau cùng của từ điển

  <*biến từ điển*>**.popitem**()

- Hàm **del**:

  del <*biến từ điển*>[<*key*>]

```
Dict={1:'Toán', 2:'Ứng dụng', 3:'K59'}
Dict.pop(2)
print(Dict)
Dict.popitem()
print(Dict)
```

```
{1: 'Toán', 3: 'K59'}
{1: 'Toán'}
```

```
Dict={1:'Toán', 2:'Ứng dụng', 3:'K59'}
del Dict[1]
Dict
```

```
{2: 'Ứng dụng', 3: 'K59'}
```

Thực hành:

- **Bài tập 6.1**: Viết chương trình mô phỏng quá trình gieo ngẫu nhiên 2 con xúc xắc 1000 lần. Chương trình sẽ thống kê và đưa ra dữ liệu về tổng số chấm và tần suất xuất hiện khi gieo 2 con xúc xắc như bảng minh họa sau:

Total	Simulated Percent
2	2.90
3	6.90
4	9.40
5	11.90
6	14.20
7	14.20
8	15.00
9	10.50
10	7.90
11	4.50
12	2.60

- **Bài tập 6.2**: Viết chương trình nhập vào **Tên**, **Điểm** thi môn kĩ thuật lập trình Python của sinh viên. Việc nhập liệu sẽ dừng nếu không có **Tên** nào được nhập. Điểm thi là số thực nằm từ 0 đến 10. Chương trình sẽ thống kê các sinh viên đạt điểm chữ A, B, C, D, F theo quy đổi sau:

Điểm số	<4	>=4	>=5.5	>=7	>=8.5
Điểm chữ	F	D	C	B	A

Thống kê minh họa:

Điểm	***Sinh viên***
A	*Hùng, Mai, Quỳnh*
B	*Nam*
C	*Kiên, Khanh*

- **Bài tập 6.3**: Cho đoạn văn bản dưới đây:

> “ But soft what light through yonder window breaks
>
> It is the east and Juliet is the sun
>
> Arise fair sun and kill the envious moon
>
> Who is already sick and pale with grief ”

Hãy viết một chương trình thống kê số lần xuất hiện các kí tự chữ (không phân biệt in hoa hay thường) trong đoạn văn bản trên.

Minh họa:

Ký tự chữ	***Số lần xuất hiện***
a	*11*
b	*2*
c	*1*