TRƯỜNG ĐẠI HỌC THƯƠNG MẠI
THUONGMAI UNIVERSITY

CHƯƠNG 4.
KIỂU DỮ LIỆU TỔNG HỢP

Số tiết: 6 tiết LT + 3 tiết TH

ĐẶT VẤN ĐỀ

Bài toán quản lý điểm thi học phần Tin lớp 2311info01:

- Yêu cầu nhập điểm thi của 15 sinh viên
- Xác định điểm thi cao nhất/thấp nhất trong lớp

→ **Thực hiện:**

Yêu cầu	Câu lệnh
Cần 15 biến lưu điểm thi của 15 sv	# 15 bien: a,b,c,d,e,f,g,h,i,k;l,m,n,o,p,q
Nhập giá trị cho 15 biến	a=eval(input("nhập diem thi cua sv thu nhat")) b=eval(input("nhập diem thi cua sv thu hai")); …
Xác định điểm thi cao nhất/thấp nhất trong lớp	diem_max = max(a,b,c,…) diem_min = min(a,b,c,…)

→ Cách giải quyết này chỉ phù hợp với dãy ít phần tử, nhiều phần tử thì cần thực hiện cách khác.

NỘI DUNG CHƯƠNG

1. Kiểu dữ liệu danh sách (List)
2. Kiểu dữ liệu bộ (Tuple)
3. Kiểu dữ liệu từ điển (Dictionary)
4. Kiểu tập hợp (Set)
5. Kiểu dữ liệu tự định nghĩa (self-defined data structure)

NỘI DUNG CHƯƠNG

4.1. Kiểu dữ liệu danh sách (List)

4.2. Kiểu dữ liệu bộ (Tuple)

4.3. Kiểu dữ liệu từ điển (Dictionary)

4.4. Kiểu tập hợp (Set)

4.5. Kiểu dữ liệu tự định nghĩa (self-defined data structure)

4.1 Kiểu dữ liệu danh sách (list)

- **Định nghĩa: Danh sách (list) là 1 thùng chứa các phần tử mà giá trị của các phần tử có thể thuộc các kiểu dữ liệu khác nhau.**
- **Cú pháp** khởi tạo list:
  - Được tạo ra bằng cách sử dụng cặp ngoặc vuông

    tên_ds = [giá_trị_1, giá_trị_2, …, giá_trị_n]
  - Hoặc dùng hàm thiết lập list()

    tên_ds = list(ds các giá trị)
- **Ví dụ 1:**

```
L1 = [1, 5, 8]                          # Tạo 1 danh sách 3 phần tử kiểu số
L2 = list( “lan”, “hong”, “hue”)        # Tạo 1 ds gồm 3 phần tử kiểu ký tự
L3 = []                                 # Tạo một ds rỗng
L4 = [1, “Lan”, 5, “Hong”, True] # Tạo 1 ds các phần tử thuộc nhiều kiểu dữ liệu khác nhau
L5 = [1, 2.71, ‘abc’, [5,6,7] ]     # Tạo danh sách có phần tử cuối là một danh sách
```

4.1 Kiểu dữ liệu danh sách (list)

- **Ví dụ 2: Nhập một danh sách bất kỳ từ bàn phím**

```
L = eval(input('Hay nhap vao 1 danh sach bat ky: '))
print('Danh sach vua nhap: ',L)
print('Phan tu dau tien cua trong danh sach L: ',L[0])
```

- **Ví dụ 3: Hãy tạo danh sách sau L3=[[1,2,3], [2,3,4], [3,4,5]]**

```
L5 = [[n,n+1,n+2] for n in range(1,4)]
print('L5= ',L5)
```

- **Ví dụ 4:** Tạo ds gồm các phần tử từ 0 đến 9 lớn hơn 2 nhỏ hơn 6 và khác 5

```
L6 = [x for x in range(0,10) if x>2 and x<6 and x !=5]
print("L6= ",L6)
```

- **Kết quả:**

```
Nhap vao 1 danh sach bat ky: [1,3,5,7]
Danh sach vua nhap:  [1, 3, 5, 7]
Phan tu dau tien cua trong danh sach L:  1
```

```
L5=  [[1, 2, 3], [2, 3, 4], [3, 4, 5]]
L6=  [3, 4]
```

Truy cập đến các phần tử trong danh sách (list)

- **Có 2 cách truy cập đến các phần tử trong list:**
  - **Cách 1**: sử dụng chỉ số của phần tử
    - Cú pháp: **ten_danh_sách[chi_so]**
    - Trong đó:
      - Tên_danh_sách: tên biến do người dùng đặt
      - Chỉ số: số nguyên nhận các giá trị từ -n đến n-1 (với n là số phần tử trong danh sách)
        - » Phần tử đầu tiên trong danh sách có chỉ số là 0, phần tử thứ 2 có chỉ số 1, …
        - » Chỉ số của các phần tử có thể âm, ám chỉ lấy từ cuối danh sách ngược về đầu.

Truy cập đến các phần tử trong danh sách (list)

- **Ví dụ 5**: Xác định chỉ số của phần tử trong danh sách sau

Chỉ số	0	1	2	3	4	5
L2 =	"toan",	"ly",	"hoa",	"van",	"su",	"dia"]
Chỉ số	-6	-5	-4	-3	-2	-1

Kết quả in ra màn hình của các lệnh print:

```
print(L2[1])    # in ra ly
print(L2[-1])   # in ra dia
print (L2[5])   # in ra dia
print(L2[3])    # in ra van
```

Truy cập đến các phần tử trong danh sách (list)

- Truy cập phần tử trong list:
  - **Cách 2**: Sử dụng khoảng chỉ số
    - Cú pháp:

Tên_danh_sách[chỉ_số_đầu : chỉ_số_cuối : bước_nhảy]

    - **Khoảng** được xác định thông qua chỉ số đầu: chỉ số cuối
      - **Chỉ số đầu**: là 1 số nguyên chỉ vị trí của phần tử đầu lấy trong danh sách, nếu thiếu chỉ số đầu này thì mặc định lấy phần tử đầu tiên trong danh sách.
      - **Chỉ số cuố**i: là 1 số nguyên chỉ vị trí của phần tử cuối cần lấy trong danh sách, nếu thiếu giá trị này thì mặc định lấy phần tử cuối cùng trong ds
      - **Bước nhảy**: giúp xác định chỉ số phần tử tiếp theo, nếu bước nhảy bằng 1 thì không cần chỉ định giá trị này.

Truy cập đến các phần tử trong danh sách (list)

- Ví dụ 6: Xác định kết quả hiển thị của các lệnh print trên L2

```
L2 = ["toan", "ly", "hoa", "van", "su", "dia"]
print(L2[2:5])          # hiển thị từ phần tử thứ 2 đến phân tử
                        # thứ 4 trong ds "hoa", "van", "su"
print(L2[-5:-2])        # "ly" "hoa" "van"
print(L2[1:-1] # Hien thị ds con ko chứa p.tử đầu, cuối của L2
print(L2[1:0])  # Hiển thị ds rỗng
print(L2[0:6]   # Hiển thị ds L2
print(L2[0:6:2] # Hiển thị ds con có chỉ số chẵn
Hoặc  print(L2[::2]
      print(L2[1:6:2] # Hiển thị ds con có chỉ số lẻ
Hoặc   print(L2[1::2]
```

Xóa phần tử trong danh sách

- Có 2 cách giúp thực hiện xóa 1 phần tử trong danh sách
  - C1: Xóa theo chỉ số của phần tử trong danh sách
  - C2: Xóa theo giá trị của phần tử trong danh sách

Xóa phần tử trong danh sách

- **Cách 1**: xóa theo chỉ số của phần tử trong danh sách
  - **Cú pháp**:

    Ten_danh_sach._delitem_(chỉ_số) hoặc del ten_danh_sach[k]

  - **Lưu ý**:
    - Chỉ số $\in$ [-n, n-1]
    - Chỉ số $\notin$ [-n, n-1] thì câu lệnh không làm gì và không báo lỗi
  - **Ví dụ 7**: Xóa phần tử có chỉ số bằng 1 trong danh sách L2

```
L2 = ["toan", "ly", "hoa", "van", "su", "dia"]
L2.__delitem__(1)
print(L2)

del L2[1]
print(L2)
```

Xóa phần tử trong danh sách

- **Cách 2**: Xóa theo giá trị của phần tử trong danh sách
  - **Cú pháp**:

```
Ten_danh_sach. remove (giá_trị_x)
```

  - **Ý nghĩa**: Xóa phần tử có giá trị x hoặc phần tử thứ k khỏi danh sách
  - **Lưu ý**:
    - Nếu trong danh sách có nhiều hơn 1 phần tử có giá trị bằng x thì phần tử x **đầu tiên** trong danh sách sẽ bị xóa.
    - Nếu trong danh sách không có phần tử x thì trình dịch Python sẽ phát sinh 1 ngoại lệ với thông báo lỗi: "ValueError: list.remove(x): x not in list" và chương trình sẽ bị dừng lại.
  - **Ví dụ 7**: Xóa phần tử có giá trị "hoa" trong danh sách L2 gốc

    → lệnh xóa: L2.remove("hoa")

    → Kết quả: L2 = `["toan", "ly", "van", "su", "dia"]`

Duyệt qua các phần tử trong danh sách

- Có 3 cách để duyệt các phần tử trong ds:
  - C1: Dùng vòng lặp for với toán tử in
  - C2: Dùng vòng lặp for và tham chiếu chỉ số của ds với hàm range và len()
  - C3: Sử dụng với vòng lặp while với hàm len()

Duyệt qua các phần tử trong danh sách

- **Toán tử in**:
  - Ý nghĩa: giúp kiểm tra 1 phần tử có trong danh sách hay không?
  - Ví dụ 8: Kiểm tra môn sinh có trong danh sách L2 không?

```
if "sinh" in l2:
    print("mon sinh co trong l2")
if "hoa" not in L2
    print("mon hoa khong co trong L2")
```

- **Hàm len():**
  - **Ý nghĩa:** trả về số phần tử trong danh sách đang xem xét
  - **Cú pháp:**

len(ten_danh_sach)

  - **Ví dụ 9:**

```
L1 = [1,5,9]
print('len(L1)= ',len(L1)) →len(L1)= 3
```

Duyệt qua các phần tử trong danh sách

- Hàm range()
  - **Ý nghĩa**: tạo ra 1 miền các số nguyên và thuộc kiểu dữ liệu tuần tự
  - **Cú pháp**:

range ([giá_trị_đầu_x1=0],<giá_trị_dừng_x2>,[khoảng_cách_n=1)]

  - **Trong đó**:
    - Hàm có 2 tham số tùy chọn là x1 và n; 1 tham số bắt buộc là x2
    - Hàm trả về 1 vùng số nguyên, xuất phát từ x1, sau đó tăng hoặc giảm theo giá trị n và dừng lại trước khi đạt x2
  - **Ví dụ 10**: Tạo ra 1 danh sách bất kỳ

List(range(0,10,2) → thu được danh sách [0,2,4,6,8]

List(range(10,0,-3)) → thu được danh sách [10,7,4,1]

Duyệt qua các phần tử trong danh sách

Cho danh sách: `L2=["toan", "ly","hoa", "van","su", "dia"]`

→ Yêu cầu: Duyệt qua các phần tử trong danh sách

- Cách 1: Dùng vòng lặp for với toán tử in

```
for x in L2:
    print(x)
```

- Cách 2: Dùng vòng lặp for và tham chiếu chỉ số của ds với hàm range và len

```
for i in range(len(L2)):
    print(L2[i])
```

- Cách 3: Sử dụng với vòng lặp while với hàm len()

```
i=0
while i<len(L2):
  print(L2[i])
  i=i+1
```

Các phép toán và phương thức của List

Tên phép toán	Cú pháp	Ý nghĩa	Ví dụ

Phép cộng +	danh_sach_1 + danh_sach_2	Ghép nối tự nhiên 2 danh sách	`A = [1,2]`
			`B = [3,4,5]`
			`C = A + B`
			→ `C = [1,2,3,4,5]`

Phép nhân *	danh_sách * n	Phép cộng n lần với danh sách xem xét	`L1 = [a,b,c]`
			`L2 = L1*2`
			→ `L2 = [a,b,c,a,b,c]`

Các phép toán và phương thức của List

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
append	a.append(<đối tượng>)	Bổ sung <đối tượng> vào cuối danh sách như một phần tử	lst = ['Hello', 'Python'] print(lst) lst.append('Tutorialspoint') print(lst) → ['Hello', 'Python'] ['Hello', 'Python','Tutorialspoint']
clear	a.clear(list)	Xóa dữ liệu, đưa danh sách a thành rỗng	lst = ['Hello','Python','Tutorialspoint'] print(lst) lst.clear() print(lst) → ['Hello', 'Python', 'Tutorialspoint'] []

Các phép toán và phương thức của List

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
copy	b=a.copy()	Trả về một danh sách khác có giá trị giống a (một bản sao của a)	lst = ['Hello', 'Python', 'Tutorialspoint'] lst1 = lst.copy() lst1.append("Java") print(lst) print(lst1) → ['Hello', 'Python', 'Tutorialspoint'] ['Hello', 'Python', 'Tutorialspoint', 'Java']
extend	a.extend(<đối tượng>)	Mở rộng, bổ sung <đối tượng> vào cuối danh sách a như một mở rộng tự nhiên	lst = ['Hello', 'Python'] lst.extend(['Java', 'CSharp']) print(lst) → ['Hello', 'Python', 'Java', 'CSharp']

Các phép toán và phương thức của List

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
index	k=a.index(<giá trị>[,start[,end]])	Trả về chỉ số index đầu tiên trong danh sách của giá trị <giá trị>	lst = ['Hello', 'Python', 'Tutorialspoint', 'Python'] print(lst.index('Python')) print(lst.index("Python", 2)) → **Kết quả:** 1 3
insert	a.insert(index,<đối tượng>)	Chèn <đối tượng> vào danh sách ở trước vị trí index	lst = ['Hello', 'Python', 'Tutorialspoint', 'Python'] lst.insert(0, "CPlusPlus") print(lst) lst.insert(3, "Java") print(lst) → ['CPlusPlus', 'Hello', 'Python', 'Tutorialspoint', 'Python'] ['CPlusPlus', 'Hello', 'Python', 'Java', 'Tutorialspoint', 'Python']

Các phép toán và phương thức của List

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
count	a.count(<“giá trị”>)	Đếm số lần xuất của “giá trị” trong a	lst = ['Hello', 'Python', 'Tutorialspoint', 'Python'] print(lst.count("Python")) print(lst.count("Tutorialspoint")) print(lst.count(" ")) → **Kết quả 2 1 0**

pop	x=a.pop(list)	Xóa và lấy ra phần tử cuối cùng của danh sách a. Đây chính là thao tác pop của ngăn xếp.	lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Python']
			lst.pop() #Without index
			print(lst)
			lst.pop(3) #With Index
			print(lst)
			→ ['CPlusPlus', 'Hello', 'Python', 'Java']
			['CPlusPlus', 'Hello', 'Python']

Các phép toán và phương thức của List

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
reverse	a.reverse(list)	Sắp xếp lại các phần tử của A theo thứ tự ngược lại so với danh sách ban đầu	lst = ['CPlusPlus', 'Hello', 'Python', 'Java', 'Python'] print(lst) lst.reverse() → **Kết quả:** ['CPlusPlus', 'Hello', 'Python', 'Java', 'Python'] ['Python', 'Java', 'Python', 'Hello', 'CPlusPlus']

sort	a.sort(list)	Sắp xếp lại danh sách a theo thứ tự tăng dần của giá trị của phần tử	lst = [2, 3, 7, 1, 13, 8, 49]
			print(lst)
			lst.sort() #default
			print(lst)
			→ **Kết quả** [2, 3, 7, 1, 13, 8, 49]
			[1, 2, 3, 7, 8, 13, 49]

Các phép toán và phương thức của List

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
sort	a.sort(list, reverse=true)	Trả về d.sách chứa các phần tử trong d.sách đã được sắp xếp theo thứ tự giảm dần về giá trị	lst = [2, 3, 7, 1, 13, 8, 49] lst.sort(reverse = True) print(lst) → **Kết quả:** [49, 13, 8, 7, 3, 2, 1]
max/min	max(a) min(a)	Trả về phần tử có giá trị lớn nhất/ nhỏ nhất trong danh sách a. (Python không hỗ trợ tìm max/min trên danh sách có các phần tử thuộc nhiều kiểu dữ liệu khác nhau, như integers và string)	list1 = [4, -4, 8, -9, 1] maxValue1 = max(list1) print(maxValue1) → **Kết quả : 8** list2 = ['a', '$', 'e', 'E'] maxValue2 = max(list2) print(maxValue2) → **Kết quả**: **e** vì với ký tự, Python trả về ký tự có mã lớn nhất trong AscII

Các phép toán và phương thức của List

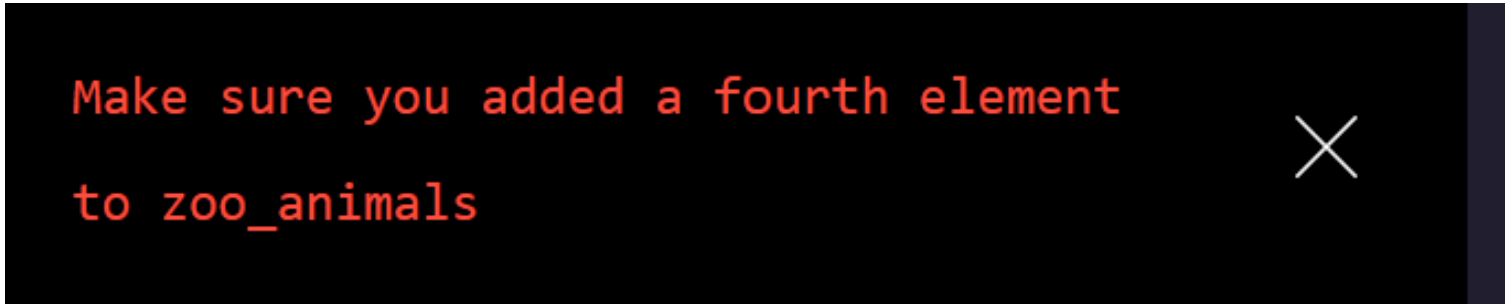
- **Yêu cầu**: Sử dụng vòng lặp **for**, **range** và phép cộng **+** tạo 1 danh sách 100 phần tử từ 1 đến 100
- **Thực hiện**:

```
a = []
for i in range(1,101):
    a = a + [i]
print(a)
```

Các phép toán và phương thức của List

- Ví dụ 11: Xác định kết quả đoạn chương trình sau

```
zoo_animals = ["pangolin", "cassowary", "sloth",];
#              One              animal              is              missing!
if len(zoo_animals) > 3:
  print "The first animal at the zoo is the " + zoo_animals[0]
  print "The second animal at the zoo is the " + zoo_animals[1]
  print "The third animal at the zoo is the " + zoo_animals[2]
  print "The fourth animal at the zoo is the " + zoo_animals[3]
```



CÂU HỎI TRẮC NGHIỆM

Câu 1. Lệnh `list(range(-3,3))` trả lại giá trị nào?

A. [-3, -2, -1, 0, 1, 2, 3]

B. [-3, -2, -1, 0, 1, 2]

C. [-2, -1, 0, 1, 2]

D. [-3, 3]

CÂU HỎI TRẮC NGHIỆM

Câu 2. Kết quả của các lệnh sau như thế nào?

```
A=[9, 1, 15, 0, "abc", 2, "de"]
A.sort
```

A. [“abc”, “de”, 0, 1, 2, 9, 15]

B. [0, 1, 2, 9, 15, “abc”, “de”]

C. Thông báo lỗi

D. [0, 1, 2, 9, 15]

CÂU HỎI TRẮC NGHIỆM

Câu 3. Kết quả đoạn chương trình sau là gì?

```
A = []
A.append([1, 2, 3])
print(A)
```

A. [1, 2, 3]

B. 1, 2, 3

C. [[1,2,3]]

D. ([1, 2, 3])

CÂU HỎI TRẮC NGHIỆM

Câu 4. Kết quả của đoạn chương trình sau là gì?

```
X=[]
X.append(1)
X.append(5)
print(X)
```

A. []

B. 1, 5

C. [1, 5]

D. 1 5

CÂU HỎI TRẮC NGHIỆM

Câu 5. Hàm số sau thực hiện công việc gì?

```
def multi(start, stop):
    p=1
    for i in range(start,stop):
        p=p*i
    return p
multi(1,6)
```

A. 24

B. 100

C. 120

D. 720

Bài tập về kiểu dữ liệu danh sách list

Bài 1. Viết chương trình thực hiện nhiệm vụ sau:

- Nhập từ bàn phím số tự nhiên n
- Nhập lần lượt n số tự nhiên tạo thành dãy a_1, a_2, …, a_n. Giả sử dãy này gọi là A.
- Đưa ra dãy B= b_1, b_2, …, b_k. B là dãy con của A bao gồm các phần tử là nguyên tố của dãy A.
- Tính tổng các phần tử của dãy B

Bài tập về kiểu dữ liệu danh sách list

Bài 2: Cho trước 2 dãy số A, B. Dãy C được tạo ra theo quy tắc sau:

Giả sử số phần tử của A là n và nhỏ hơn số phần tử của B. Trường hợp ngược lại làm tương tự.

- n phần tử đầu tiên của C là tổng các số hạng tương ứng của A, B
- Các phần tử còn lại lấy từ B

Viết thủ tục với 2 tham số đầu vào là A, B và in ra màn hình dãy C.

List của List (Ma trận trong Python)

- Cho ma trận A

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

- Ma trận này được mô tả bằng kiểu dữ liệu list của list như sau:

A= [<list1>, <list2>, …, <listN>]

Hoặc

A=[[a_{11}, a_{12}, …, a_{1n}], [a_{21}, a_{22}, …, a_{2n}], …, [a_{m1}, a_{m2}, …, a_{mn}]]

List của List (Ma trận trong Python)

- Ý tưởng:
  - Tạo ma trận
    - Danh sách A với m phần tử
    - Danh sách A_i có n phần tử
    - Lần lượt bổ sung danh sách A_i vào danh sách A
  - Duyệt ma trận: sử dụng 2 vòng lặp for kết hợp chỉ số i, j, cụ thể phần tử A[i][j] là phần tử thứ j của danh sách A[i]

List của List (Ma trận trong Python)

- Nhập ma trận từ bàn phím:

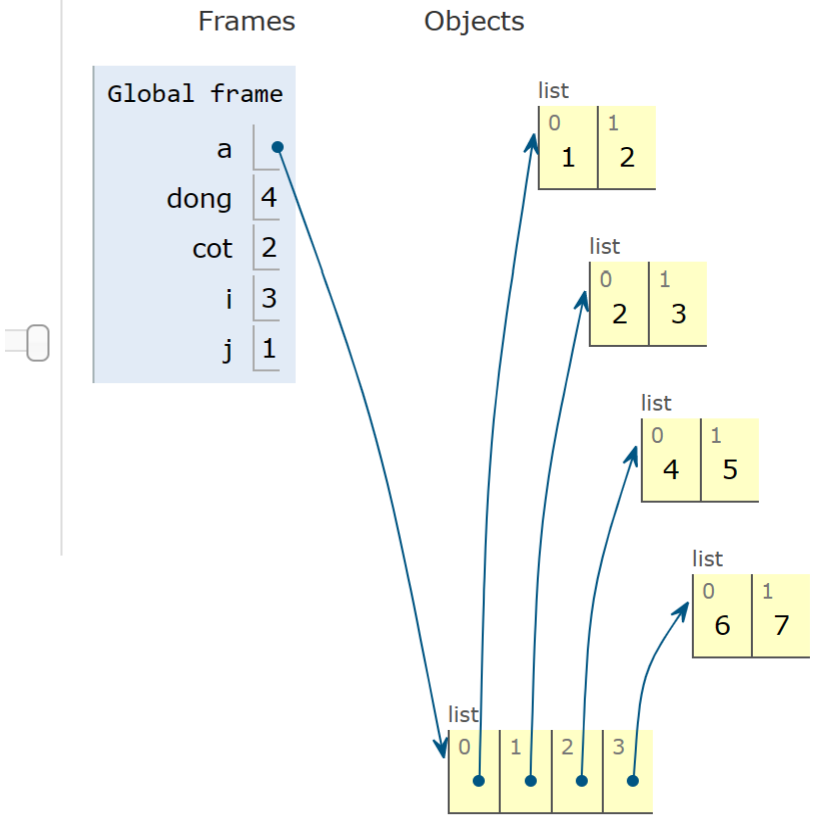
```
1   a =[]
2   m = int(input("Nhap so tu nhien m: ")) # m dong
3   n = int(input("Nhap so tu nhien n: ")) # n cot
4
5   for i in range(m):
6       print("Chuan bị nhap ma tran hang thu ",i+1,":")
7       b = []
8       for j in range(n):
9           x = int(input("Nhap phan tu thu "+str(j+1)+": "))
10          b = b + [x]
11      a.append(b)
12  print("Ma tran a da nhap xong.")
```

List của List (Ma trận trong Python)

```
1  a = [[1,2],[2,3],[4,5],[6,7]]
2  dong=len(a)
3  cot=len(a[0])
4  for i in range(0,dong):
5      for j in range(0,cot):
6          print(a[i][j],end=" ")
7      print(" ")
```

Print output (drag lower right corner to resize)

```
1 2
2 3
4 5
6 7
```



List của List (Ma trận trong Python)

- Hiển thị ma trận từ bàn phím: có 2 cách
  - Cách 1:
  - Ví dụ 12:

for <item> in <dữ liệu tuần tự>:

```
14 print("Hien thi ma tran vua nhap: ")
15 # A[m][n] m dòng và n cột
16 for i in range(m):
17         for j in range(n):
18                 print(a[i][j],end = " ")
19         print(" ") # xuong dong
```

List của List (Ma trận trong Python)

- Hiển thị ma trận từ bàn phím: có 2 cách
  - **Cách 2:**

for <item1>, <item2>, …, <itemN> in <list>:

  - **Ví dụ 13:** In ra danh sách học sinh A gồm các thông tin như sau: Tên học sinh, năm sinh, quê quán.

Biết A = [["Trần Tuấn Anh", 2000, "Hà Nội"],
["Đỗ Ngọc Linh",2000,"Thái Bình"],
["Hoàng Xuân Lan",2000,"Nam Định"]]

→ **code:**

```
for ten, namsinh, que in A:
    print("Tên HS: ",ten,"Năm sinh: ",namsinh,"Quê quán: ",que)
```

List của List (Ma trận trong Python)

Bài 3: Yêu cầu in bảng điểm của các học sinh trong lớp, biết mỗi học sinh sẽ gồm có 3 thông tin sau: Tên học sinh, các điểm hệ số 1, các điểm hệ số 2.

Thông tin học sinh được lưu trong danh sách A:

A=[["Tuấn ", [6,7,9], [7,9,10,5]], ["Linh", [9,8,7], [9,10,8,9]], ["Lan", [6,7,8], [8,9,10,9]]]

→ code:

```
for ten, diem1, diem2 in A:
    print (ten)
    print("          Hệ số 1: ",end=" ")
    for diem in diem1:
        print(diem, end= " ")
    print()
    print("          Hệ số 2: ", end="  ")
    for diem in diem2:
        print(diem, end=" ")
    print()
```

BÀI TẬP

Bài 1. Nhập vào hai ma trận A(m,n) và B(n,p) in ra tích của hai ma trận

Bài 2. Nhập vào một ma trận, kiểm tra xem ma trận đã cho có phải là ma trận đơn vị không.

Bài 3. Nhập vào một ma trận, kiểm tra xem ma trận đã cho có phải là ma trận đối xứng không.

Bài 4. Nhập vào một dãy số nguyên (list) và thực hiện:

a. In ra dãy đã nhập

b. Tính tổng các phần tử của dãy

c. Tính tổng các phần tử dương và chia hết cho 3 trong dãy

d. Đếm số phần tử là số nguyên tố của dãy.

e. Tìm vị trí xuất hiện đầu tiên của giá trị x trong dãy, với x là giá trị nhập vào từ bàn phím

BÀI TẬP (Bài 1)

```
def nhapMT(A,x,y):
    for i in range(0,x):
        print(" - Nhap dong thu",i+1, "cua
ma tran ")
        Adong=[]
        for j in range(0,y):
            z = int(input("     Phan tu thu
"+str(j+1)+"="))
            Adong=Adong+[z]
        A.append(Adong)
    print("Hien thi ma tran vua nhap: ")
    for i in range(0,x):
        for j in range(0,y):
            print(A[i][j],end=" ")
        print()
    print()
```

```
A = []
print("Kich thuoc ma tran A: ")
m= int(input("- So dong m= "))
n= int(input("- So cot  n= "))
B = []
print("Kich thuoc ma tran B:")
p=0
while(n!=p):
    p = int(input("- So dong p= "))
q = int(input("- So cot q= "))
```

BÀI TẬP (Bài 1)

```
print("Nhap gia tri cho ma tran A: ")
nhapMT(A,m,n)
print("Nhap gia trị cho ma tran B")
nhapMT(B,p,q)
C = []
print("Tinh tich hai ma tran A va B")
for i in range(0,m):
    tam = []
    for j in range(0,q):
        tong=0
        for k in range(0,p):
            tong=tong+A[i][k]*B[k][j]
        tam=tam+[tong]
    C.append(tam)
```

```
print("Ma tran C:")
for i in range(0,m):
    for j in range(0,q):
        print(C[i][j],end="  ")
    print()
```

NỘI DUNG CHƯƠNG

4.1. Kiểu dữ liệu danh sách (List)

4.2. Kiểu dữ liệu bộ (Tuple)

4.3. Kiểu dữ liệu từ điển (Dictionary)

4.4. Kiểu tập hợp (Set)

4.5. Kiểu dữ liệu tự định nghĩa (self-defined data structure)

4.2. Kiểu dữ liệu bộ (Tuple)

- Python hỗ trợ một cấu trúc dữ liệu tương tự với **List**, có tên là **Tuple**.
- Tuy nhiên khác với **List**, **Tuple** là một danh sách bất biến, nghĩa là ngay sau khi khởi tạo Tuple, chúng ta không thể thay đổi nó.
  - Ví dụ 14:
```
a = (1,2,3) # tuple a có 3 phần tử 1, 2, 3
a[0]=10     # phần tử thứ nhất của a gán b
```
    → báo lỗi: Type error: “tuple” object does not support item assignment
- Nhưng, phép + các dữ liệu kiểu tuple vẫn thực hiện được.
- Tuple không bị giới hạn về số lượng phần tử và phần tử có thể thuộc nhiều kiểu dữ liệu khác nhau như số nguyên, số thập phân, list, string, …

4.2. Kiểu dữ liệu bộ (Tuple)

- **Cú pháp khởi tạo:**
  - **Sử dụng dấu ngoặc đơn**

tên_tuple = (ds các phần tử)

✓ Các phần tử được đặt trong dấu ngoặc đơn (có thể bỏ dấu ngoặc đơn)
✓ Các phần tử phân tách nhau bởi dấu phẩy

  - **Hoặc hàm thiết lập tuple()**

ten_tuple = tuple((ds các phần tử)) # chú ý có hai dấu ngoặc đơn

- Ví dụ 15: khởi tạo tuple

```
T1=("An", "Cường", "Linh")

T2=tuple((2,5,8))
```

4.2. Kiểu dữ liệu bộ (Tuple)

- **Lưu ý:** nếu tuple chỉ có 1 phần tử thì phải có dấu phẩy sau phần tử để nhận dạng kiểu tuple, nếu ko có sẽ được hiểu kiểu của giá trị phần tử

```
T=("An",);
print(type(T));
print(type(T3))
```

```
T2=("An");
print
(type(T2));
```

```
T3=(4)
```

→ Kết quả:

```
"C:\Users\Public\BT Python\venv\Scripts\python.exe"
<class 'tuple'>
<class 'str'>
<class 'int'>

Process finished with exit code 0
```

4.2. Kiểu dữ liệu bộ (Tuple)

- **Đặc điểm (giống list):**
  - Các phần tử có thứ tự, cho phép giá trị trùng nhau và thuộc các kiểu dữ liệu khác nhau.
  - Để lấy số phần tử của tuple, dùng hàm len()
  - Có thể nối hai tuple với nhau bằng toán tử + (giống list)
- **Ví dụ 16:**

  a = (1, 2, 3)

  b = (“one”, “two”, “three”)

  c= a+b

  → ta có: c = (1, 2, 3, “one”, “two”, “three”)

4.2. Kiểu dữ liệu bộ (Tuple)

- **Đặc điểm (giống list):**
  - Các phần tử trong tuple được đánh chỉ số, phần tử đầu tiên có chỉ số là 0. Chỉ số của các phần tử có thể là số âm, ám chỉ lấy từ cuối tuple ngược về đầu.
  - Có thể lấy các phần tử trong tuple trong khoảng chỉ số. Nếu thiếu chỉ số đầu thì mặc định lấy từ phần tử thứ nhất, nếu thiếu chỉ số cuối thì mặc định lấy đến phần tử cuối cùng.
  - Để duyệt các phần tử trong tuple, có thể dùng vòng lặp for với toán tử in hoặc tham chiếu chỉ số của tuple với hàm range và len hoặc dùng với vòng lặp while

4.2. Kiểu dữ liệu bộ (Tuple)

- **Đặc điểm (khác list):**
  - Không thể thay đổi giá trị hay thêm phần tử vào tuple → giải pháp
    - Chuyển tuple thành list và dùng hàm append, insert, extend của list
    - Hoặc ghép tuple với tuple bằng toán tử +
  - Không thể xóa phần tử trong tuple → giải pháp
    - Chuyển tuple thành list và dùng hàm remove, pop của list (hàm del là xóa cả tuple)
  - Ví dụ 17:

```
T=("Apple", "Grape", "Mango")
L=list(T)                  #chuyển tuple thành ds
L[1]="Cherry"              #thay đổi giá trị của phần tử thứ 2
L.append("Lemon")          #thêm phần tử vào cuối ds
L.pop(2)                   #xóa phần tử ở vị trí thứ 3
T=tuple(L)                 #chuyển ngược lại
print(T)
```

4.2. Kiểu dữ liệu bộ (Tuple)

- Khi nào sử dụng **Tuple**:
  - Cần xử lý nhanh vì Tuple có tốc độ xử lý nhanh hơn List
  - Khi định nghĩa một tập hợp các giá trị là hằng số
  - Tuple giúp code an toàn hơn, bởi vì Tuple giúp cho dữ liệu không thể thay đổi.

4.2. Kiểu dữ liệu bộ (Tuple)

- **Mở gói/ giải nén tuple**
  - Khi tạo ra 1 tuple, thường gán các giá trị cho nó → gọi là đóng gói tuple.
  - Khi trích các giá trị của tuple vào các biến → gọi là giải nén/ mở gói tuple.
  - **Ví dụ 18:**

```
T=("Math", "Physics", "Chemistry")
(to,ly,ho)=T
print(to); print(ly); print(ho)
```

  - Có thể dùng dấu hoa thị * (asterisk) để thay thế cho một phần của tuple mà khớp với số biến tương ứng
  - **Ví dụ 19:**

```
T=("Math", "Physics", "Chemistry", "Biology", "History")
(to,*ph, su)=T
print(to); print(ph); print(su)
→ #Math    ['Physics', 'Chemistry', 'Biology']      History
```

4.2. Kiểu dữ liệu bộ (Tuple)

- Các phương thức làm việc với Tuple

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
index	k=a.index(<giá trị>[,start[,end]])	Trả về chỉ số index đầu tiên trong tuple của giá trị tìm <giá trị>	tup = ('Hello', 'Python', 'Tutorialspoint', 'Python‘) print(tup.index('Python')) print(tup.index("Python", 2)) → **Kết quả:** 1 3
count	a.count(<“giá trị”>)	Đếm số lần xuất của “giá trị” trong a	tup = ['Hello', 'Python', 'Tutorialspoint', 'Python'] print(tup.count("Python")) print(tup.count("Tutorialspoint")) print(tup.count(" ")) → **Kết quả 2 1 0**

NỘI DUNG CHƯƠNG

4.1. Kiểu dữ liệu danh sách (List)

4.2. Kiểu dữ liệu bộ (Tuple)

4.3. Kiểu dữ liệu từ điển (Dictionary)

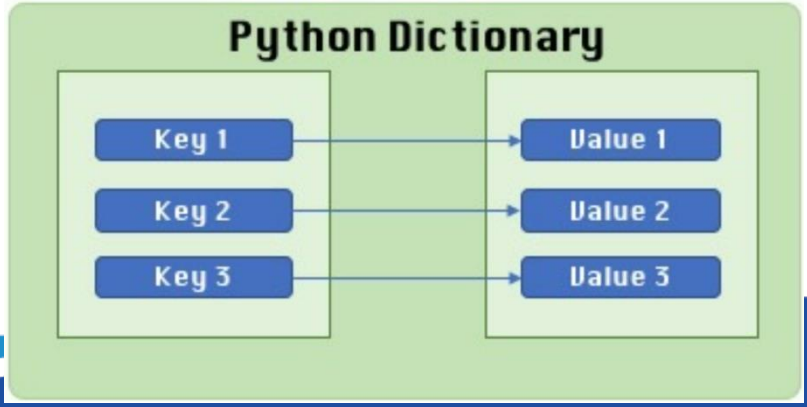
4.4. Kiểu tập hợp (Set)

4.5. Kiểu dữ liệu tự định nghĩa (self-defined data structure)

4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Từ điển**: Là kiểu dữ liệu được sử dụng để lưu trữ các giá trị dữ liệu theo khóa, dưới dạng từng cặp <khóa> <giá trị>
  - Được giới hạn bởi cặp ngoặc nhọn {}, tất cả những gì nằm trong đó là những phần tử của từ điển.
  - Các phần tử của từ điển được phân cách nhau bởi dấu phẩy dưới “,”
  - Mỗi phần tử của từ điển là một cặp <khóa>:<giá trị>
  - Ngăn cách giữa thành phần <khóa> và thành phần <giá trị> bởi dấu hai chấm “:”
  - Các <khóa> trong một từ điển buộc phải

  là một đối tượng (duy nhất)



4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Cú pháp tạo từ điển**
  - **Cách 1: sử dụng dấu ngoặc nhọn**

<biến_từ điển> = {<khóa_1: giá_trị_1>, <khóa_2: giá_trị_2>, …, <khóa_n>:<giá_trị_n>}

  - Trong đó:
    - <biến_từ_điển>: là tên của 1 biến do người dùng đặt tên theo quy tắc đặt tên
    - <khóa_1>, <khóa_2>, <khóa_n>: là danh mục các khóa được dùng như chỉ số của các phần tử trong từ điển và các khóa này không được trùng nhau
    - <giá_trị_1>, <giá_trị_2>, …, <giá_trị_n>: là dãy các giá trị được lưu trữ trong từ điển đi theo khóa và được quản lý bởi <biến_từ_điển>. Nếu ta không đưa cặp khóa giá trị nào vào danh sách thì Python sẽ khởi tạo một từ điển rỗng.

4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Cú pháp tạo từ điển**
  - **Cách 2: sử dụng hàm dict()**

<biến_từ điển> = dict(khóa_1 = giá_trị_1, khóa_2 = giá_trị_2, …)

  - Ví dụ:

```
numdict = dict(I='one', II='two', III='three')
print(numdict)
```

4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Chú ý:**
  - Phần khóa key của từ điển phải thuộc loại bất biến như kiểu dữ liệu số, xâu, tuple, không được phép thuộc list.
  - Các giá trị trong từ điển có thể thuộc bất kỳ loại nào, số, xâu, tuple, list thậm chí là từ điển hay tập hợp.
  - Các khóa từ điển có phân biệt chữ hoa chữ thường.

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Truy cập đến các phần tử trong từ điển
  - Cách 1: Truy xuất tự nhiên

<biến_từ điển> [<khóa>]

  - Cách 2: Sử dụng phương thức get()

<biến_từ điển> .get (<khóa>, default = None)

Trong đó,

- khóa: là tên khóa được tìm trong từ điển
- default: là giá trị được trả về của hàm get() trong trường hợp khóa key không tồn tại

4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Ví dụ 20: Cho từ điển person, hãy lấy thông tin về tên, tuổi :**

```
person = {
          'Ten' : 'Nguyễn Tuấn Anh',
          'dia chi' : "Hà  Nội",
          'tuoi' : 22,
          'gioi tinh' : 'Nữ'
          }
print ("Tên = ",person['Ten']          # hiển thị tên
print ("Tuổi =", person.get('tuoi')) # hiển thị tuổi
print("G.tính = ", person.get('gtinh', "sai khóa key")
```

→ Hiển thị: Tên = Lê Thị Mận
Tuổi = 22
Sai khóa key

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Truy cập dữ liệu từ điển
  - Lấy tất cả các khóa trong từ điển

<biến_từ điển>.keys()

  - Lấy tất cả các giá trị trong từ điển

<biến_từ điển>.values()

  - Lấy đồng thời cả khóa và giá trị trong từ điển

<biến_từ điển>.items()

4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Ví dụ 21: Lấy tất cả các giá trị, khóa, và khóa + giá trị trong từ điển person**

```
print("Tu dien person co: ")
print(" - Danh sach khoa: ",person.keys())
print(" - Danh sach gia tri: ",person.values())
print(" -> Khoa+gia tri: ",person.items())
```

→ **Kết quả hiển thị:**

```
Tu dien person co:
 - Danh sach khoa:  dict_keys(['Ten', 'dia chi', 'tuoi', 'gioi tinh'])
 - Danh sach gia tri:  dict_values(['Nguyễn Tuấn Anh', 'Hà  Nội', 22, 'Nữ'])
 -> Khoa+gia tri:  dict_items([('Ten', 'Nguyễn Tuấn Anh'), ('dia chi', 'Hà  Nội'), ('tuoi', 22), ('gioi tinh', 'Nữ')])
```

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Duyệt qua tất cả các phần tử trong từ điển: Sử dụng cấu trúc **for** hoặc **while**:
  - Duyệt khóa:

```
for biến_duyệt in biến_từ_điển.keys()
```

  - Duyệt giá trị:

```
for biến_duyệt in biến_từ_điển.values()
```

  - Duyệt cả khóa và giá trị:

```
for biến_duyệt_khóa, biến_duyệt_giá trị in biến_từ_điển.items()
```

4.3 Kiểu dữ liệu từ điển (Dictionary)

- **Ví dụ 22: Cho từ điển svien, hãy duyệt từ điển để hiển thị thông tin về môn thi, điểm thi**

```
svien = {'Toan': 7, 'Ly': 9, 'Hoa': 10}
```

```
i = 1 # thứ tự môn theo điểm giảm dần
j = count = 0 #biến đếm số môn thi
sum = 0 # biến lưu tổng điểm 3 môn thi
for monthi, diemthi in sorted(svien.items()):
    print("Danh sach mon thi la:")
    for monthi in sorted(svien.keys()):
        print("Hien thi diem thi theo thu tu giam dan:")
        print("Diem thi mon ", monthi, "cao thu ", j, ":
",diemthi)
for diemthi in sorted(svien.values(),reverse=True):
    print("- Diem cao thu ",i,": ",diemthi)
    count = count + 1
    j = j+1
    i = i+1
    print("-> Tong so mon thi =", count)
    sum=sum+diemthi
    print()
print("-> Tong diem thi= ",sum)
```

```
Danh sach mon thi la:
    - Toan
    - Ly
    - Hoa
Hien thi diem thi theo thu tu giam dan:
Diem mon  Hoa cao thu  1 :  10
Diem mon  Ly cao thu  2 :  9
Diem mon  Toan cao thu  3 :  7
```

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Thêm/sửa một phần tử trong từ điển:

<biến_từ điển>.[<khóa>] = <giá_trị>

Trong đó:

- Nếu khóa đã có giá trị thì giá trị cũ mất đi, giá trị mới thay vào;
- Nếu chưa có giá trị trong từ điển thì một giá trị mới của khóa được thêm vào.

- Xóa phần tử trong từ điển
  - Xóa 1 phần tử

<biến_từ điển>.__delitem__(<khóa>)

  - Xóa tất cả các phần tử trong từ điển

<biến_từ điển>.clear()

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Các phương thức làm việc với từ điển

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ

len()	biến_từ_điển.Len()	Trả về độ dài của từ điển	`svien = {'Toan': 7, 'Ly': 9, 'Hoa': 10}`
			`print(len(svien))`
			→ Kết quả: 3

copy()	Từ_điển_mới_b = Từ_điển_gốc_a.copy ()	Sao chép toàn bộ dữ liệu của từ điển a sang từ điển b	`svien1 = {'Toan': 7, 'Ly': 9, 'Hoa': 10}`
			`svien2 = svien1.copy()`
			`print(svien2)`
			→ Kết quả `{'Toan': 7, 'Ly': 9, 'Hoa': 10}`

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Các phương thức làm việc với từ điển

Xét từ điển: **`svien = {'Toan': 7, 'Ly': 9, 'Hoa': 10}`**

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
sorted()	sorted(từ_điển)	Sắp xếp các phần tử trong dictionary cũ và tạo ra một list mới.	**`svien1=sorted(svien); print(svien1)`** → **kết quả:** **`['hoa', 'ly', 'toan']`**
	sorted.(từ_điển.keys())	Sắp xếp theo keys	**svien2=sorted(svien.keys())** **print(svien2)** → **`['hoa', 'ly', 'toan']`**
	sorted.(từ_điển.values())	Sắp xếp theo values	**`svien3=sorted(svien.values())`** **`print(svien3)`** → **`[7,9,10]`**

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Các phương thức làm việc với từ điển

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
sorted()	sorted(từ_điển.items(), reverse=False))	Sắp xếp các cặp key và value (dựa trên khóa key): reverse = True: tăng dần reverse = False: giảm dần	**svien = {'Toan': 7, 'Ly': 9, 'Hoa': 10}** **print(sorted(svien.items())** **→ Kết quả:** **[('Hoa', 10), ('Ly', 9), ('Toan', 7)]**

	sorted.(từ_điển.items(), key=lambda x:x[1], reverse=False)	Sắp xếp các cặp key và value theo value	svien = {'Toan': 7, 'Ly': 9, 'Hoa': 10}
			svien1=sorted(svien.items(),key = lambda x:x[1], reverse=True)
			print(svien1) #danh sach sx theo value
			svien2=dict(svien1)
			print(svien2) #tu dien sx theo value

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Các phương thức làm việc với từ điển

Tên phương thức	Cú pháp	Ý nghĩa	Ví dụ
pop()	**pop**(*key*[, *default*])	Xóa phần tử với key được chỉ định. Nếu khóa key có không có trong từ điển thì trả về giá trị default, nếu không có default thì Python sẽ báo lỗi KeyError	**L2 = {1:"toan", 2:"ly", 3:"hoa", 4:"van"}** **print(L2.pop(0,"khong co key nay"))** **print(L2)** **L2.pop(5)**
popitem()	**popitem**()	Xóa phần tử cuối cùng trong từ điển theo nguyên tắc LIFO.	**L2 = {1:"toan", 2:"ly", 3:"hoa", 4:"van« }}** **L2.popitem()** **print(L2)**

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Bài tập:

Một thương hiệu thời trang X bao gồm một chuỗi 10 cửa hàng được đặt trên khắp cả nước. Hãy viết chương trình sử dụng từ điển giúp chủ cửa hàng giải quyết các công việc:

- Nhập doanh số bán hàng từ 10 cửa hàng gửi về
- Sắp xếp doanh số bán hàng từ bé đến lớn
- Tính doanh số trung bình trên 10 cửa hàng
- Hiển thị doanh số bán hàng lớn nhất.

Từ đó, chủ của chuỗi cửa hàng này lập kế hoạch kinh doanh và tặng thưởng cho cửa hàng có doanh số cao nhất.

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Có thể sao chép từ điển này sang từ điển khác bằng cách:
  - Dùng phương thức copy
  - Dùng hàm dict()

```
d={"subj":"toan","credit":3,"type":"Optional"}
d1=d.copy()
d2=dict(d1)
print(d1)
print(d2)
```

4.3 Kiểu dữ liệu từ điển (Dictionary)

- Cho phép từ điển lồng từ điển

```
myfamily = {
    "child1"    :  {
        "name"  : "Emil",
        "year"  : 2004
    },
    "child2"    :  {
        "name"  : "Tobias",
        "year"  : 2007
    },
    "child3"    : {
        "name"  : "Linus",
        "year"  : 2011
    }
}
```

- Hoặc

```
child1 = {
        "name"   : "Emil",
        "year"   : 2004
},
child2 = {
        "name"   : "Tobias",
         "year"  : 2007
 },
child3 = {
         "name"  : "Linus",
         "year"  : 2011
}
myfamily = {
    "child1" : child1,
    "child2" : child2,
    "child3" : child3
}
```

NỘI DUNG CHƯƠNG

4.1. Kiểu dữ liệu danh sách (List)

4.2. Kiểu dữ liệu bộ (Tuple)

4.3. Kiểu dữ liệu từ điển (Dictionary)

4.4. Kiểu tập hợp (Set)

4.5. Kiểu dữ liệu tự định nghĩa (self-defined data structure)

4.4. Kiểu dữ liệu tập hợp (Set)

- **Tập hợp set:** Là kiểu dữ liệu chứa nhiều phần tử trong 1 biến, tuy nhiên các phần tử không có thứ tự, không thể thay đổi, không được đánh chỉ số và không cho phép giá trị trùng nhau, nhưng có thể thuộc các kiểu dữ liệu khác nhau như số, logic, xâu ký tự, tuple (đây là các kiểu dữ liệu không thay đổi immutability).
- **Tập hợp set** không thể chứa phần tử là list, dict hay set (là các kiểu dữ liệu container nhưng có thể thay đổi - mutability)

4.4. Kiểu dữ liệu tập hợp (Set)

- **Cú pháp khởi tạo giá trị**:
  - Cách 1: Được khai báo/ thiết lập giá giá trị trong cặp ngoặc nhọn

```
tên_set={danh sách các giá trị}
```

  - Cách 2: Hoặc dùng hàm thiết lập set()

```
tên_set=set((danh sách các giá trị))
```

  - Lưu ý: Để tạo tập hợp rỗng ta dùng lệnh set() không dùng {}
- Ví dụ 23: Khởi tạo giá trị tập S1, S2, S3

```
S1 = {"Math","Physics","Chemistry"}
S2 = set(("Math",31,"Chemistry"))
S3 = set()
```

4.4. Kiểu dữ liệu tập hợp (Set)

- **Thêm/bớt phần tử của tập hợp**
  - ❑ **Thêm 1 phần tử**: bằng phương thức add()

    Tập hợp Set không cho phép thay đổi các phần tử trong tập hợp nhưng có thể thêm 1 phần tử vào tập hợp bằng phương thức add()
    - **Cú pháp:**

```
Biến_tập_hợp.add(phần tử thêm)
```

    - **Ví dụ 24:** Minh họa hàm add()

```
S1={"Math","Physics","Chemistry"}
S1.add("Biology")
S1.add([Science, History]) → # Returns: TypeError: unhashable type: 'list'
```

4.4. Kiểu dữ liệu tập hợp (Set)

- **Thêm/bớt phần tử của tập hợp**
  - ❑ **Thêm nhiều phần tử:** bằng phương thức update()

    Thực hiện thêm nhiều phần tử từ một tập hợp, list, tuple, dictionary sang một tập hợp khác bằng phương thức update()
    - **Cú pháp:**

      ```
      Biến_tập_hợp.update(đối_tượng_lặp)
      ```

    - **Ví dụ 25:** Minh họa hàm update()

      ```
      S={"Math","Physics","Chemistry"}
      L=["Biology","History"]
      S.update(L)
      print(S)
      ```

→ **Kết quả:** `{'Biology', 'Chemistry', 'Math', 'Physics', 'History'}`

4.4. Kiểu dữ liệu tập hợp (Set)

- **Thêm/bớt phần tử của tập hợp:**
  - ❑ **Nối 2 tập hợp**: bằng phương thức union() (sẽ loại bỏ các giá trị trùng nhau)
  - – **Cú pháp:**

```
Biến_tập_hợp1.union(biến_tập_hợp2)
```

  - – **Ví dụ 26:** Minh họa hàm union()

```
S={"Math","Physics","Chemistry"}
S1={"Physics", "Geography" }
L=["Biology","History"]
S.update(L)
S2=S.union(S1)
```

→ Kết quả: {'History', 'Chemistry', 'Biology', 'Physics', 'Math', 'Geography'}

4.4. Kiểu dữ liệu tập hợp (Set)

- **Thêm/bớt phần tử của tập hợp:**
  - ❑ **Xóa 1 phần tử: dùng hàm remove()**
  - – **Cú pháp**:

```
Biến_tập_hợp.remove(phần tử xóa)
```

  - – Lưu ý: Phần tử trong tham số của hàm remove() không có trong tập hợp thì chương trình báo lỗi.
  - – **Ví dụ 27:** Minh họa hàm remove()

```
S={"Math","Physics","Chemistry"}
S.remove("Math")
S.remove("History")          #error
```

4.4. Kiểu dữ liệu tập hợp (Set)

- **Thêm/bớt phần tử của tập hợp**
  - Xóa tất cả phần tử làm rỗng tập hợp: bằng hàm clear()
    - Cú pháp:

```
Biến_tập_hợp.clear()
```

    - Ví dụ 28: Minh họa hàm clear()

```
S={"Math","Physics","Chemistry"}
S.clear()
```

4.4. Kiểu dữ liệu tập hợp (Set)

- **Thêm/bớt phần tử của tập hợp:**
  - Xóa phần tử cuối cùng của tập hợp: bằng hàm pop():
    - Cú pháp:

```
Biến_tập_hợp.pop()
```

    - Lưu ý: Do set không có thứ tự nên hàm pop() sẽ xóa **ngẫu nhiên** một phần tử của set.
    - Ví dụ 29: Minh họa hàm pop()

```
S={"Math","Physics","Chemistry"}
S.pop()
→Chạy lần 1: {"Math","Physics"}
→Chạy lần 2: {"Physics","Chemistry"}
```

4.4. Kiểu dữ liệu tập hợp (Set)

- **Kiểm tra quan hệ nằm trong, tập con, tập chứa tập hợp**

STT	Toán tử/ hàm	Ý nghĩa	Ví dụ
1	in	Kiểm tra 1 phần tử có nằm trong 1 tập hợp hay không (True/False)	**A = {(1,2), 2, 'Two'}** **2 in A → trả về True** **1 in A → trả về False**
2	issubset()	Kiểm tra 1 tập hợp có nằm trong tập hợp khác hay không (True/False)	**A = {(1,2), 2, 'Two'};** B= {2,6} B.issubset(A) → Ktra B có là tập hợp con của A → False
3	issuperset()	Kiểm tra 1 tập hợp có chứa tập hợp khác nhau không (True/False)	A **= {(1,2), 2, 'Two'};** B= {2} A.issuperset(B) → Kiểm tra A có chứa tập hợp B → True

4.4. Kiểu dữ liệu tập hợp (Set)

- **Các phép toán trên tập hợp:**

STT	Tên quan hệ	Cách viết	Ý nghĩa	Ví dụ
1	Phép hợp (union)	A|B	Tạo ra 1 tập hợp mới (từ các phần tử nằm trong A hoặc B) mà không làm thay đổi tập hợp gốc	A={1,5} ; B= {2,6} A|B={1,2,5,6}
2	Phép giao (intersection)	A&B	Tạo ra 1 tập mới (từ các phần tử nằm đồng thời trong A và B) mà không làm thay đổi tập hợp gốc	A={1,2} ; B= {2,6} A&B={2}

4.4. Kiểu dữ liệu tập hợp (Set)

- **Các phép toán trên tập hợp:**

STT	Tên quan hệ	Cách viết	Ý nghĩa	Ví dụ
3	Phép trừ (difference)	A-B	Tạo ra 1 tập mới (là các phần tử nằm trong A mà không nằm trong B) và không làm thay đổi tập hợp gốc	A={1,5} ; B= {2,6,1} A-B={5}
4	Phép trừ đối xứng (symmetric difference)	A^B	Tạo ra 1 tập mới (là các phần tử hoặc nằm trong A nhưng không nằm trong B, hoặc nằm trong B nhưng không nằm trong A) mà không làm thay đổi tập hợp gốc	A={1,5} ; B= {2,6,5} A^B={1,2,6}

4.4. Kiểu dữ liệu tập hợp (Set)

Bài tập: Giả sử có 2 tập hợp A, B. A là tập hợp tên các bạn học sinh lớp 9A; B là họ tên các bạn học sinh lớp 9B. Cụ thể:

A = {"Nguyễn Hà", "Trần Hải", "Lưu Hà"}
B = {"Trần Tiến", "Nguyễn Hà", "Lưu Hà"}

Viết một chương trình tính:

- Số các học sinh có tên trùng nhau của 2 lớp 9A, 9B. Trong ví dụ trên số học sinh trùng tên là 2 của mỗi lớp.
- Số các học sinh có họ trùng nhau của 2 lớp 9A, 9B. Trong ví dụ trên số học sinh trùng họ của mỗi lớp là 3

4.4. Kiểu dữ liệu tập hợp (Set)

- Gợi ý:

```
A = {"Nguyễn Hà", "Trần Hải", "Lưu Hà"}
B = {"Trần Tiến", "Nguyễn Hà", "Lưu Hà"}
print("Danh sach ten trung nhau: ",A&B)
L=[]     # tao danh sach rong
S=set() # tao tap hop rong
for x in A:
    L1=x.split()     # tach tu
    L.append(L1[0]) # them 1 ptu vao danh sach
for x in B:
    L2 = x.split()
    L.append(L2[0])
S.update(L)
print("So hoc sinh co ho trung nhau la: ", len(S))
print("     ",S)
```

NỘI DUNG CHƯƠNG

4.1. Kiểu dữ liệu danh sách (List)

4.2. Kiểu dữ liệu bộ (Tuple)

4.3. Kiểu dữ liệu từ điển (Dictionary)

4.4. Kiểu tập hợp (Set)

4.5. Kiểu dữ liệu tự định nghĩa (self-defined data structure)

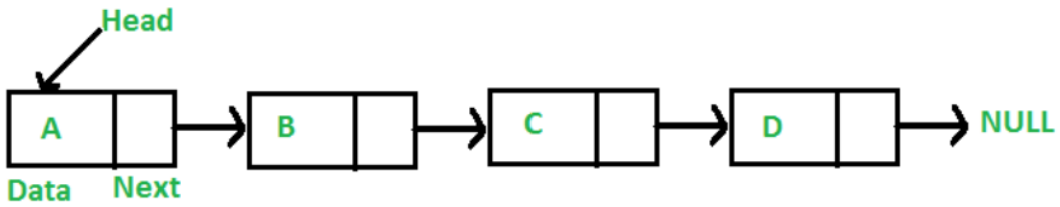
4.5. Kiểu dữ liệu tự định nghĩa

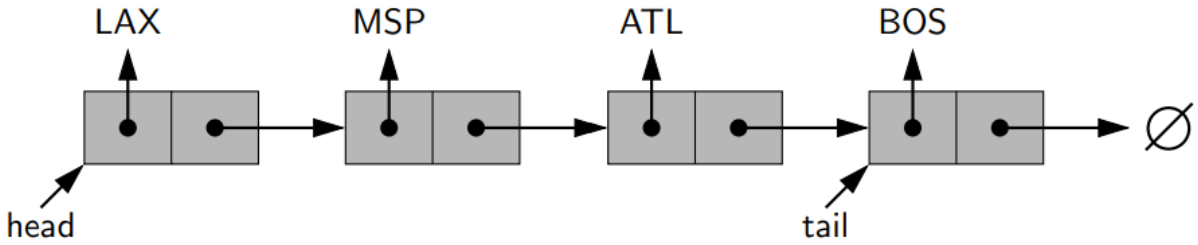
- Các kiểu dữ liệu tự định nghĩa gồm:
  - Linked list
  - Stack
  - Queue
  - Tree
  - Graph
  - Hashmap

4.5.1. Kiểu dữ liệu tự định nghĩa – Linked list

- **Linked list – danh sách liên kết:**

Danh sách liên kết là một cấu trúc dữ liệu tuyến tính, trong đó các phần tử không được lưu trữ trong các vị trí nhớ liên tục. Từ đó, các phần tử trong 1 danh sách liên kết được liên kết bằng cách sử dụng các con trỏ:





4.5.1. Kiểu dữ liệu tự định nghĩa – Linked list

- **Thực hiện Linked-list trên Python:**
  - Bước 1: Định nghĩa lớp tạo nút mới

    Đối tượng của lớp này là 1 nút thực sự sẽ được chèn vào linked list. Thông qua hàm khởi tạo, ta thiết lập giá trị và địa chỉ trỏ tới nút tiếp theo.
  - Bước 2: Định nghĩa lớp tạo danh sách liên kết linked list.

    Lớp này chứa các phương thức như chèn, xóa, duyệt, sắp xếp, … danh sách liên kết linked list.
  - Bước 3: Nạp các lớp đã định nghĩa sử dụng phương thức trong lớp để thực hiện các thao tác trên linked list.

4.5.1. Kiểu dữ liệu tự định nghĩa – Linked list

- **Tạo Linked list có 3 phần tử 10, 20, 30**

```
class Node:  # Bước 1
    def __init__(self, gia_tri):
        self.gia_tri = gia_tri
        self.nut_ke_tiep = None

class LinkedList: # Bước 2
    def __init__(self):
        self.nut_dau_tien = None

    def in_ds(self):
        if self.nut_dau_tien is None:
            print("Ds rong")
            return
        else:
            n = self.nut_dau_tien
            while n is not None:
                print(n.gia_tri, " ")
                n = n.nut_ke_tiep
```

```
# Chèn vào đầu danh sách liên kết Linked list
def chen_nut_dau_tien(self, gia_tri):
    nut_moi = Node(gia_tri)
    nut_moi.nut_ke_tiep = self.nut_dau_tien
    self.nut_dau_tien = nut_moi

def chen_nut_cuoi_ds(self, gia_tri):
    nut_moi = Node(gia_tri)
    if self.nut_dau_tien is None:
        self.nut_dau_tien = nut_moi
        return
    n = self.nut_dau_tien
    while n.nut_ke_tiep is not None:
        n = n.nut_ke_tiep
    n.nut_ke_tiep = nut_moi

def tim(self, gia_tri):
    pass

def xoa(self, gia_tri):
    pass

def cap_nhat(self, vi_tri, gia_tri):
    pass
```

4.5.1. Kiểu dữ liệu tự định nghĩa – Linked list

- Tạo Linked list có 3 phần tử 10, 20, 30 (tiếp theo)

 # Bước 3:

```
from dslk import *
def main():
    ds = LinkedList()
    ds.chen_nut_dau_tien(10)
    ds.chen_nut_cuoi_ds(20)
    ds.chen_nut_cuoi_ds(30)
    ds.in_ds()
```

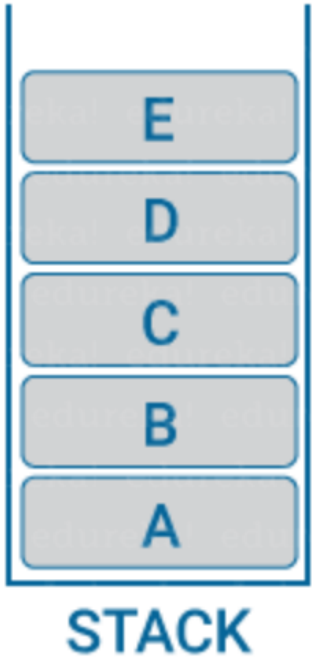
Kết quả

```
"C:\Users\Public\BT Python\venv\Scripts\python.exe"
10
20
30

Process finished with exit code 0
```

4.5.2. Kiểu dữ liệu tự định nghĩa - Stack

- **Stack-Ngăn xếp:**
  - Stacks là các cấu trúc dữ liệu tuyến tính dựa trên nguyên tắc *Last-In-First-Out (LIFO)* trong đó dữ liệu được nhập sau cùng sẽ là dữ liệu đầu tiên được truy cập.
  - Stacks được sử dụng nổi bật trong các ứng dụng như theo dõi URL, Recursive Programming, đảo ngược các từ, hoàn tác các cơ chế trong trình soạn thảo văn bản, v.v.



4.5.2. Kiểu dữ liệu tự định nghĩa - Stack

- **Stack – ngăn xếp:**
  - Trong python, chúng ta có thể triển khai ngăn xếp bằng cách sử dụng danh sách list.
  - Phương thức append () để thêm một mục vào stack hiện có
  - Phương thức pop () xóa phần tử ở vị trí đã chỉ định khỏi stack
- **Ví dụ 30:**

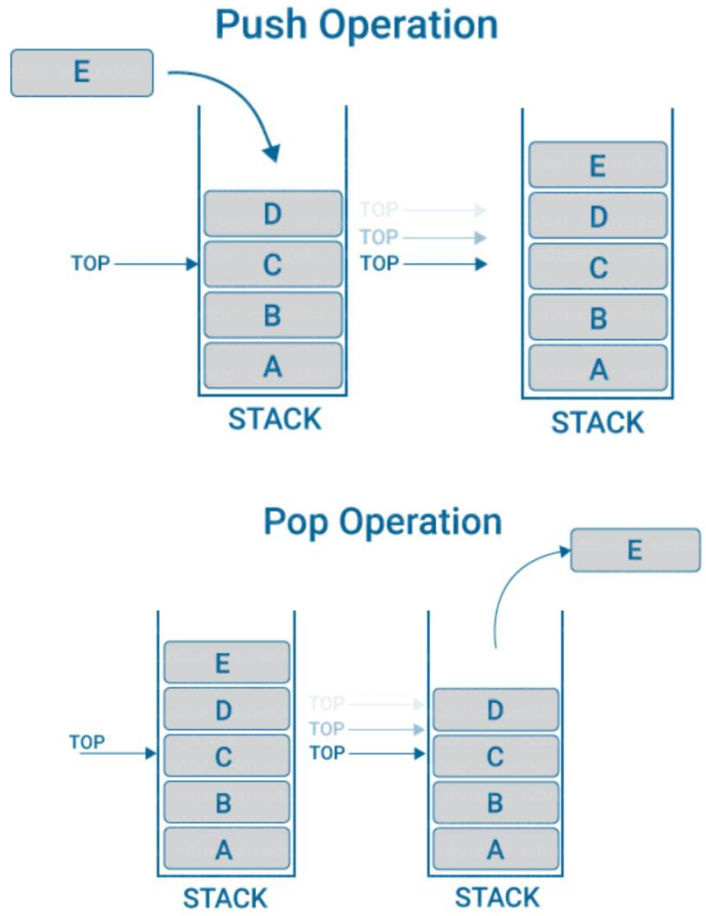
  Cho stack gồm các phần tử A, B, C, D. Hãy thêm phần tử E vào stack trên sau đó xóa phần tử vừa thêm khỏi stack.

4.5.2. Kiểu dữ liệu tự định nghĩa - Stack

- Ví dụ 30:
  - Đẩy thêm phần tử “E” vào stack
  - Xóa phần tử “E” khỏi stack

→ Code:

```
stack1=['A','B','C','D']
stack1.append('E')
print(stack1)
stack1.pop()
print(stack1)
```



4.5.2. Kiểu dữ liệu tự định nghĩa - Stack

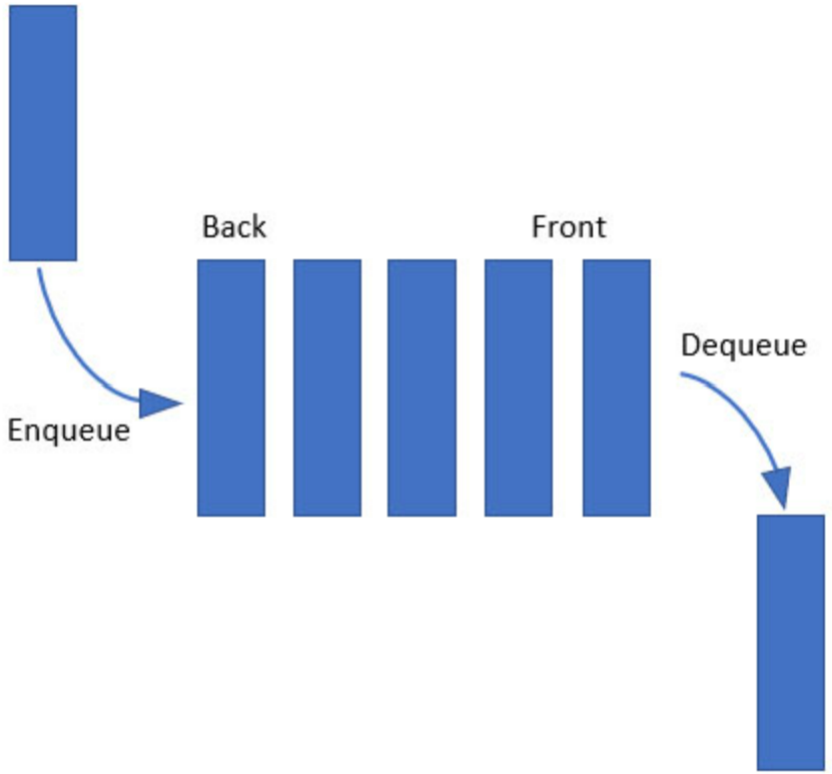
- Ngoài ra, ta còn có 1 số thao tác khác đối với Stack thông qua list.

Stack Method	*Realization with Python list*
S.push(e)	L.append(e)
S.pop( )	L.pop( )
S.top( )	L[−1]
S.is_empty( )	len(L) == 0
len(S)	len(L)

- Yêu cầu: Sử dụng hàm trong python viết chương trình thực hiện các công việc trên.

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- **Queue – hàng đợi:**
  - Queue cũng là một cấu trúc **dữ liệu tuyến tính** dựa trên nguyên tắc *Nhập trước xuất trước (FIFO)* trong đó dữ liệu được nhập trước sẽ được truy cập trước.
  - Enqueue: thao tác chèn vào hàng đợi
  - Dequeue: thao tác xóa khỏi hàng đợi
  - Queue được sử dụng làm *Bộ đệm mạng (Network Buffers)* để quản lý tắc nghẽn lưu lượng, được sử dụng trong Hệ điều hành để lên lịch công việc, …



4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- Trong Python, để triển khai hàng đợi ta sử dụng các cách sau:
  - Danh sách list
  - collections.deque
  - queue.Queue

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

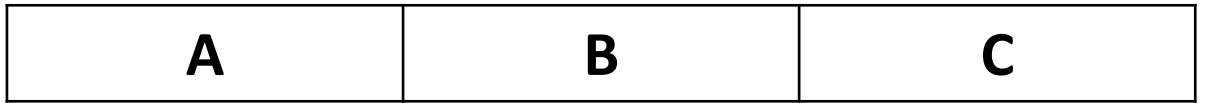
- **Triển khai Queue từ list:**
  - Phương thức append() để chèn các phần tử vào queue (dequeue)
  - Phương thức pop() để xóa các phần tử khỏi queue (enqueue)
- **Lưu ý**: Triển khai queue từ list tương đối chậm vì việc chèn thêm và xóa phần tử khỏi queue được thực hiện từng phần tử một nên việc này chiếm thời gian 0(n).
- **Ví dụ 31**: Khởi tạo 1 queue rỗng, sau đó lần lượt thêm các phần tử A, B, C vào queue. Sau đó lần lượt xóa phần tử A, B khỏi queue

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- **Ví dụ 31 (tiếp):**
  - Queue Q rỗng
  - Lần lượt thêm các phần tử A, B, C vào Q
  - Lần lượt xóa phần tử A, B khỏi Q

→ Code:

```
Q = []
Q.append('A')
Q.append('B')
Q.append('C')
print(Q)
print(Q.pop(0))
print(Q.pop(0))
print(Q)
```




```
['A', 'B', 'C']
A
B
C
[]
```

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- **Triển khai Queue từ collections.deque:**
  - Dequeue: Phương thức append() để chèn phần tử vào queue
  - Enqueue: Phương thức popleft() để xóa phần tử khỏi queue.
- **Lưu ý:** So với list thì collections.deque chạy nhanh hơn. Độ phức tạp về mặt thời gian là 0(1)
- **Ví dụ 32:** Sử dụng collections.deque để triển khai queue cho bài toán trong ví dụ 31.

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- **Ví dụ 32:**

A	B	C

```
from collections import deque

q = deque() # Khởi tạo queue

# Thêm các phần tử vào queue
q.append('A')
q.append('B')
q.append('C')

print("Hien thi queuen luc dau:")
print(q)

# Xóa các phần tử khỏi queue
print("\nCac phan tu bi enqueue khoi queue:")
print(q.popleft())
print(q.popleft())
```

```
Hien thi queue luc dau:
deque(['A', 'B', 'C'])

Cac phan tu bi enqueue khoi queue:
A
B

Hien thi queue sau xoa:
deque(['C'])
```

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- **Triển khai Queue từ queue.Queue:**

Hàm	Ý nghĩa	Ví dụ
queue.Queue(maxsize)	hàm khởi tạo queue với kích cỡ maxsize	#khai báo queue q chứa 3 phần tử import queue q = queue.Queue(maxsize = 3)
put(item)	hàm thêm item vào queue	q.put(1) # Thêm 1 vào q
get()	hàm xóa và trả về item của queue. Nếu queue rỗng thì đợi cho đến khi có item.	q.get() #Lấy 1 ra khỏi q

get_nowait()	Trả về 1 item nếu có sẵn ngược lại gọi ngoại lệ Queue Empty	try:
		  q.get_nowait()
		except queue.Empty:
		  print('queue Empty')

4.5.3. Kiểu dữ liệu tự định nghĩa - Queue

- **Triển khai Queue từ queue.Queue:**

Hàm	Ý nghĩa	Ví dụ
put_nowait(item)	Đưa 1 phần tử vào queue nếu queue chưa đầy, ngược lại nếu queue đầy thì sẽ gọi ngoại lệ QueueFull	try: q.put_nowait(2) except queue.Full: print('queue Full')
maxsize()	Số phần tử được phép chứa trong queue	q.maxsize() → 3
empty()	Trả về True nếu queue rỗng ngược lại là False	q.empty() → False
full()	Trả về True nếu queue có maxsize phần tử trong queue ngược lại là False	q.full() → False
qsize()	Trả về số phần tử đang có trong queue.	q.qsize() → 1

4.5. Kiểu dữ liệu tự định nghĩa - Queue

Ví dụ 33: Triển khai Queue từ queue.Queue

```
from queue import Queue

# Khởi tạo q kích cỡ 3 phần tử
q = Queue(maxsize = 3)

print(q.qsize()) # Hiển thị kích cỡ của q

# Thêm phần tử vào q
q.put('a')
q.put('b')
q.put('c')

#Kiểm tra tình trạng q đầy/không đầy

print("\nFull: ", q.full())

# Xóa phần tử khỏi q
print(q.get())
print(q.get())
print(q.get())

# Kiểm tra tình trạng q rỗng/ không rỗng
print("\nEmpty: ", q.empty())

q.put(1)

# Kiểm tra tình trạng q rỗng
print("\nEmpty: ", q.empty())

# Kiểm tra q đầy
print("Full: ", q.full())

# print(q.get()) # Xóa phần tử khỏi q
```

```
0

Full:  True
a
b
c

Empty:  True

Empty:  False
Full:  False
1
```

4.5.4. Kiểu dữ liệu tự định nghĩa - Tree

- **Tree - Cây:**
  - Tree là cấu trúc dữ liệu phi tuyến tính có gốc và nút.
  - **Gốc (Root)** là nút từ nơi dữ liệu bắt nguồn và các nút là các điểm dữ liệu khác có sẵn cho chúng ta. Nút có trước là cha và nút sau được gọi là con. Có những cấp độ mà một Tree phải thể hiện độ sâu của thông tin. Các nút cuối cùng được gọi là *leaf*.
  - Tree tạo ra một hệ thống phân cấp có thể được sử dụng trong rất nhiều ứng dụng trong thế giới thực như các trang HTML sử dụng Tree để phân biệt thẻ nào nằm dưới khối nào. Cấu trúc dữ liệu trong Python này cũng hiệu quả trong mục đích tìm kiếm và nhiều hơn nữa.

4.5.5. Kiểu dữ liệu tự định nghĩa - Graph

- **Graph – đồ thị:**
  - Đồ thị graph được sử dụng để lưu trữ dữ liệu thu thập các điểm được gọi là đỉnh (nút) và cạnh (cạnh). Đồ thị có thể được gọi là đại diện chính xác nhất của bản đồ thế giới thực. Chúng được sử dụng để tìm khoảng cách **cost-to-distance** khác nhau giữa các điểm dữ liệu khác nhau được gọi là các nút và do đó tìm được đường dẫn ít nhất.
  - Nhiều ứng dụng như *Google Maps*, *Uber* và nhiều ứng dụng khác sử dụng Biểu đồ để tìm khoảng cách ít nhất và tăng lợi nhuận theo những cách tốt nhất.

4.5.6. Kiểu dữ liệu tự định nghĩa - Hashmap

- Hashmap – bản đồ băm:
  - HashMaps giống như từ điển trong Python.
  - Chúng có thể được sử dụng để thực hiện các ứng dụng như danh bạ, điền dữ liệu theo danh sách và nhiều hơn nữa

4.5. Kiểu dữ liệu tự định nghĩa

- **Linked list**

Ví dụ:

```
llist = ['first', 'second', 'third']
print(llist)
print()
# adding elements
llist.append('fourth')
llist.append('fifth')
llist.insert(3, 'sixth')
print(llist)
print()

llist.remove('second')
print(llist)
print()
```

```
['first', 'second', 'third']

['first', 'second', 'third', 'sixth', 'fourth', 'fifth']

['first', 'third', 'sixth', 'fourth', 'fifth']
```

TỔNG KẾT CHƯƠNG 4

Tên cấu trúc	Đặc điểm thứ tự	Giá trị phần tử	Đặc điểm trùng nhau	Ghi chú
list	Có thứ tự	Có thể thay đổi	Các phần tử được phép trùng nhau	
tuple	Có thứ tự	Không thay đổi	Các phần tử được phép trùng nhau	
set	Không có thứ tự	Không thay đổi	Các phần tử không được phép trùng nhau	Cho phép xóa và thêm phần tử mới vào set
dictionary	Có thứ tự	Có thể thay đổi	Các khóa không được phép trùng nhau	Từ Python version 3.7, từ điển có thứ tự, trước đó thì không

TỔNG KẾT CHƯƠNG 4

- Nêu được định nghĩa, cú pháp khởi tạo dữ liệu và đưa ví dụ minh họa về cú pháp đó cho các kiểu dữ liệu sau:
  - Kiểu dữ liệu danh sách (list) trong Python
  - Kiểu dữ liệu bộ (Tuple) trong Python,
  - Kiểu dữ liệu từ điển (Dictionary) trong Python,
  - Kiểu dữ liệu tập hợp (Set) trong Python,
- Trình bày được định nghĩa, cú pháp khởi tạo của các kiểu dữ liệu tự định nghĩa (self-defined data structure) trong Python.

GIẢI THÍCH THUẬT NGỮ - GLOSSARY

Thuật ngữ	Ý nghĩa
list	**Kiểu dữ liệu danh sách**
tuple	**Kiểu dữ liệu bộ**
set	**Kiểu dữ liệu tập hợp**
dictionary	**Kiểu dữ liệu từ điển**
Linked list	Danh sách liên kết
Tree	Cây
Graph	Đồ thị
hashmap	Bản đồ băm
Stack	Ngăn xếp
Queue	Hàng đợi

CÂU HỎI ÔN TẬP

Câu 1. Trình bày khái niệm, cú pháp khởi tạo kiểu dữ liệu danh sách (list) trong ngông ngữ lập trình Python?

Câu 2. Trình bày khái niệm, cú pháp khởi tạo kiểu dữ liệu bộ (tuple) trong ngông ngữ lập trình Python

Câu 3. Trình bày khái niệm, cú pháp khởi tạo kiểu dữ liệu tập hợp (set) trong ngông ngữ lập trình Python

Câu 4. Trình bày khái niệm, cú pháp khởi tạo kiểu dữ liệu từ điển (Dictionary) trong ngông ngữ lập trình Python

Câu 5. Trình bày khái niệm, cú pháp khởi tạo kiểu dữ tự định nghĩa (self-defined data structure) trong Python.

HẾT CHƯƠNG 4
CHÚC CÁC EM HỌC TỐT!!!