

Bài 02: Lập trình Python cơ bản (t)

AI Academy Vietnam

1. Các toán tử trong Python

Toán tử số học | Toán tử gán | Toán tử so sánh | Toán tử logic | Toán tử membership

2. Cấu trúc điều khiển

Dạng 1, dạng 2, dạng 3, dạng 4

3. Cấu trúc vòng lặp

Vòng lặp While | Vòng lặp for | Break, continue

4. Cấu trúc dữ liệu cơ sở

Tuple | set | Dictionary

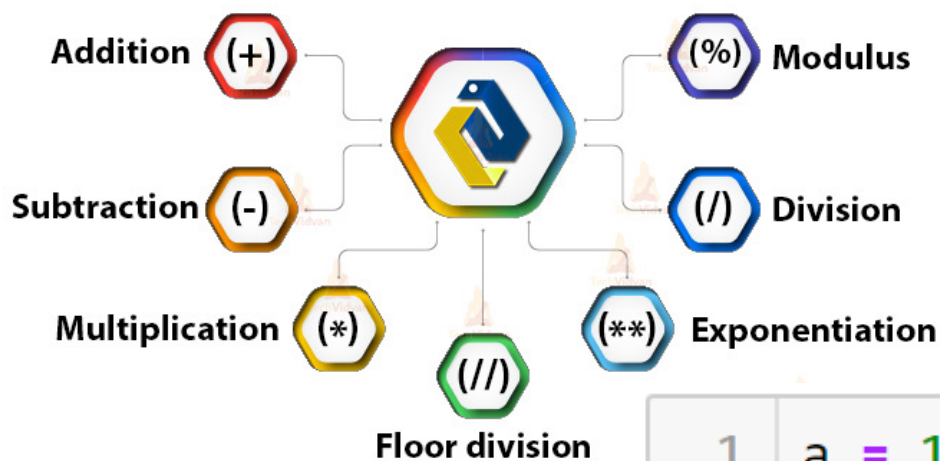
5. Ngoại lệ và xử lý ngoại lệ



1. Các toán tử trong Python

Các toán tử số học

Python Arithmetic Operators



```
1  a = 10
2  b = 8
3  #-----
4  tong = a + b           # Tổng của hai số (+)
5  hieu = a - b           # Hiệu của hai số (-)
6  tich = a*b             # Tích của hai số (*)
7  thuong = a/b           # Thương của hai số (/)
8  thuong_nguyen = a//b   # Phép chia lấy phần nguyên (//)
9  thuong_du = a % b      # Phép chia lấy phần dư (%)
10 mu = a**b              # Tính giá trị a lũy thừa b (**)
```

Các toán tử gán

Toán tử	Mô tả	Ví dụ
=	Toán tử này dùng để gán giá trị của một đối tượng cho một giá trị	$c = a$ (lúc này c sẽ có giá trị $= a$)
+=	Toán tử này cộng rồi gán giá trị cho đối tượng	$c += a$ (tương đương với $c = c + a$)
-=	Toán tử này trừ rồi gán giá trị cho đối tượng	$c -= a$ (tương đương với $c = c - a$)
*=	Toán tử này nhân rồi gán giá trị cho đối tượng	$c *= a$ (tương đương với $c = c * a$)
/=	Toán tử này chia rồi gán giá trị cho đối tượng	$c /= a$ (tương đương với $c = c / a$)
%	Toán tử này chia hết rồi gán giá trị cho đối tượng	$c \% = a$ (tương đương với $c = c \% a$)
**=	Toán tử này lũy thừa rồi gán giá trị cho đối tượng	$c ** = a$ (tương đương với $c = c ** a$)
//=	Toán tử này chia làm tròn rồi GÁN giá trị cho đối tượng	$c //= a$ (tương đương với $c = c // a$)

Các toán tử so sánh

Toán tử	Mô tả	Ví Dụ (a =8, b=10)
==	So sánh giá trị của các đối số xem có bằng nhau hay không. Nếu bằng nhau thì kết quả trả về sẽ là True và ngược lại sẽ là False.	a == b // False
!=	So sánh giá trị của các đối số xem có khác nhau hay không. Nếu khác nhau thì kết quả trả về sẽ là True và ngược lại sẽ là False.	a != b //True
<	Dấu < đại diện cho phép toán nhỏ hơn, nếu đối số 1 nhỏ hơn đối số 2 thì kết quả sẽ trả về là True và ngược lại sẽ là False.	a < b //True
>	Dấu > đại diện cho phép toán lớn hơn, nếu đối số 1 lớn hơn đối số 2 thì kết quả sẽ trả về là True và ngược lại sẽ là False.	a > b //False
<=	Dấu > đại diện cho phép toán nhỏ hơn hoặc bằng, nếu đối số 1 nhỏ hơn hoặc bằng đối số 2 thì kết quả sẽ trả về là True và ngược lại sẽ là False.	a <= b //True
>=	Dấu > đại diện cho phép toán lớn hơn hoặc bằng, nếu đối số 1 lớn hơn hoặc bằng đối số 2 thì kết quả sẽ trả về là True và ngược lại sẽ là False.	a>= b //False

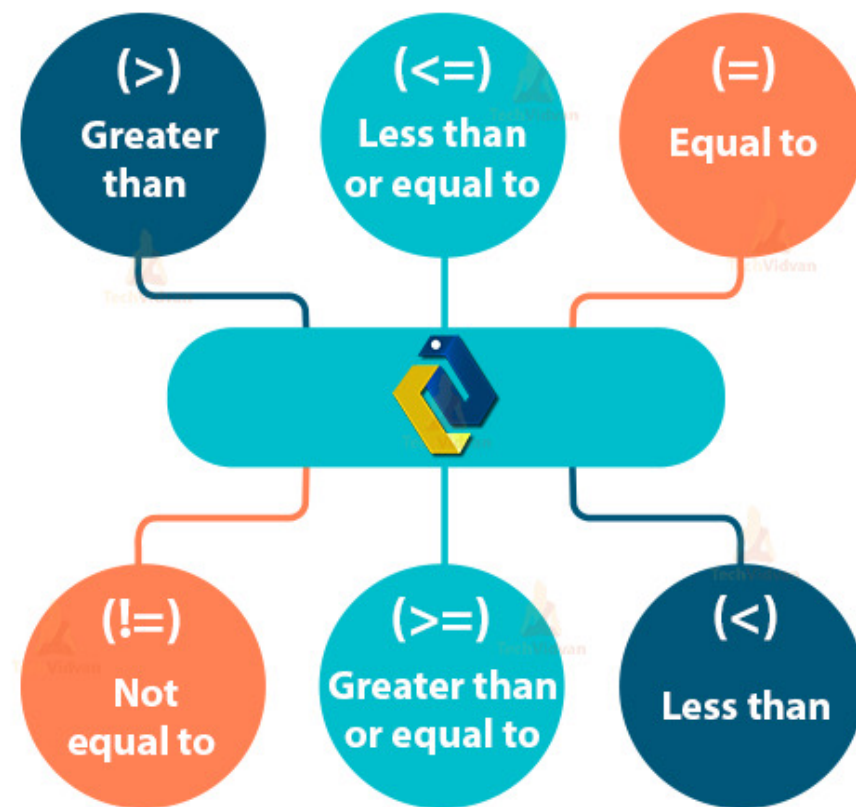
Các toán tử so sánh (t)

```
1 #Kết quả của các phép so sánh có kiểu dữ liệu Boolean
2 a = 8
3 b = 10
4 kt = a>b
5 print(type(kt))
6 #-----
7 print('1) SS lớn hơn (a>b):', a>b)
8 print('2) SS nhỏ hơn (a<b):', a<b)
9 print('3) SS bằng (a=b):', a==b)
10 print('4) SS lớn hơn hoặc bằng (a>=b):', a>=b)
11 print('5) SS nhỏ hơn hoặc bằng (a<=b):', a<=b)
12 print('6) SS khác (a!=b):', a!=b)
```

```
<class 'bool'>
```

```
1) SS lớn hơn (a>b): False
2) SS nhỏ hơn (a<b): True
3) SS bằng (a=b): False
4) SS lớn hơn hoặc bằng (a>=b): False
5) SS nhỏ hơn hoặc bằng (a<=b): True
6) SS khác (a!=b): True
```

PYTHON RELATIONAL OPERATORS



Học viên nhập code và đọc kết quả của các phép so sánh ở trên!

Các toán tử logic

Toán tử	Mô tả	Ví dụ
and	Nếu 2 vế của toán tử này đều là True thì kết quả sẽ là True và ngược lại nếu 1 trong 2 vế là False thì kết quả trả về sẽ là False.	$x < 5$ and $x < 10$
or	Nếu 1 trong 2 vế là True thì kết quả trả về sẽ là True và ngược lại nếu cả 2 vế là False thì kết quả trả về sẽ là False.	$x < 5$ or $x < 4$
not	Đây là dạng phủ định, nếu biểu thức là True thì nó sẽ trả về là False và ngược lại.	$\text{not}(x < 5 \text{ and } x < 10)$

Các toán tử xác thực

Toán tử	Mô tả	Ví dụ: $a=4$, $b=5$
is	Toán tử này sẽ trả về True nếu $a == b$ và ngược lại	$a \text{ is } b$ //False
is not	Toán tử này sẽ trả về True nếu $a != b$ và ngược lại	$a \text{ is not } b$ //True

Toán tử membership

Toán tử	Mô tả	Ví dụ
in	Nếu 1 đối số thuộc một tập đối số nó sẽ trả về True và ngược lại.	a in b
not in	Nếu 1 đối số không thuộc một tập đối số nó sẽ trả về True và ngược lại.	a not in b

```
1  #D/ Các toán tử membership
2  a = [4, 6, 9 , 0]
3  kt1 = 4 in a          #Kiểm tra 1 phần tử có trong danh sách không?
4  kt2 = 0 not in a      #Kiểm tra 1 phần tử không có trong danh sách không?
5
6  print('Kết quả 1: ', kt1)
7  print('Kết quả 2:', kt2)
```

Kết quả 1: True

Kết quả 2: False

Python Operator Precedence

Precedence	Operator Sign	Operator Name
Highest	**	Exponentiation
	+X, -X, ~X	Unary positive, unary negative, bitwise negation
	*, /, //, %	Multiplication, division, floor, division, modulus
	+, -	Addition, subtraction
	<<, >>	Left-shift, right-shift
	&	Bitwise AND
	^	Bitwise XOR
		Bitwise OR
	==, !=, <, <=, >, >=, is, is not	Comparison, Identity
	not	Boolean NOT
	and	Boolean AND
Lowest	or	Boolean OR

2. Cấu trúc điều khiển (if)

- Câu lệnh điều kiện là một trong những câu lệnh cơ bản của bất cứ ngôn ngữ lập trình nào.

Ngôn ngữ tự nhiên	Ngôn ngữ lập trình
Nếu Bạn đủ 18 tuổi thì <i>“Bạn được kết hôn”,</i> Ngược lại thì <i>“Bạn chưa được kết hôn”</i>	if (age > = 18): <i>print(‘Bạn được kết hôn!’)</i> else: <i>print(‘Bạn chưa được kết hôn!’)</i>

```
1  #Câu Lệnh điều kiện
2  so_tien = input ('Nhập vào số tiền bạn có: ')
3  so_tien = int(so_tien)
4  if (so_tien >= 1000000000):
5      print('Bạn đã là một tỷ phú!')
6  else:
7      print('Bạn còn phải kiếm nhiều tiền hơn!')
```

Các dạng câu lệnh điều kiện

- Dạng 1:

if (điều kiện1):

Nhóm lệnh 1

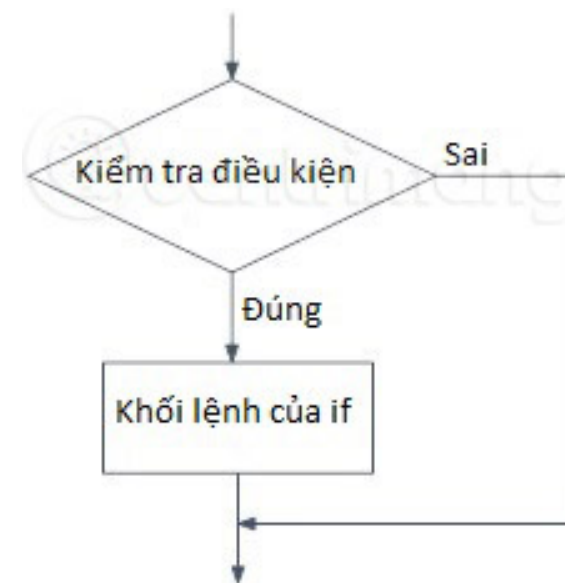
(Nếu điều kiện1 đúng thì thực hiện nhóm lệnh 1)

```
In [2]: num = 3
        if num > 0:
            print(num, "là số dương.")
        print("Thông điệp này luôn được in.")
```

```
3 là số dương.
Thông điệp này luôn được in.
```

```
In [3]: num = -1
        if num > 0:
            print(num, "là số dương.")
        print("Thông điệp này luôn được in.")
```

```
Thông điệp này luôn được in.
```



Ví dụ: Kiểm tra số chẵn

Nhập vào 1 số nguyên N, kiểm tra nếu N là số chẵn hiển thị thông báo.

- **Yêu cầu:**
Nhập vào một số: 12
- **Kiểm tra và hiển thị thông báo:**
Số 12 là số chẵn!

12

Odd

13

Even

Các dạng câu lệnh điều kiện (t)

- Dạng 2:

if (điều kiện1):

Nhóm lệnh 1

else:

Nhóm lệnh 2

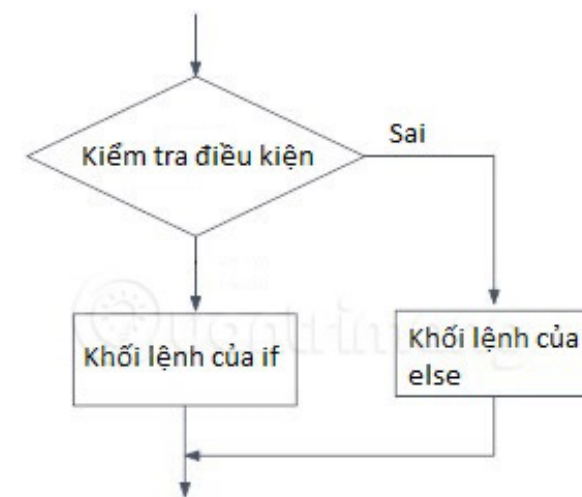
(Nếu điều kiện1 đúng thì thực hiện nhóm lệnh 1, nếu sai thực hiện nhóm lệnh 2)

```
In [5]: num = 3
        if num >= 0:
            print("Số dương")
        else:
            print("Số âm")
```

Số dương

```
In [6]: num = -1
        if num >= 0:
            print("Số dương")
        else:
            print("Số âm")
```

Số âm



Ví dụ: Kiểm tra số chẵn – lẻ

Nhập vào 1 số nguyên N, kiểm tra nếu N là số chẵn hiển thị thông báo “ Đây là số chẵn!”, ngược lại thông báo “Đây là số lẻ!”

12

Odd

Nhập vào một số:12
Đây là số chẵn!

13

Even

Nhập vào một số:13
Đây là số lẻ!

Các dạng câu lệnh điều kiện (t)

- Dạng 3 (if lồng nhau):

if (điều kiện1):

Nhóm lệnh 1

elif (điều kiện 2):

Nhóm lệnh 2

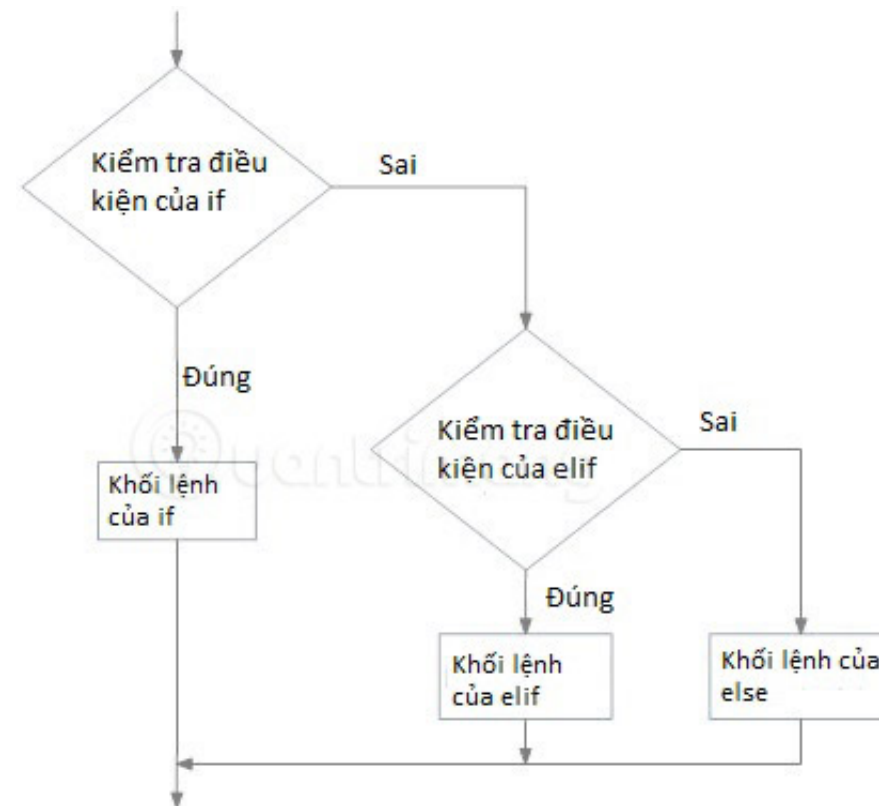
...

elif (điều kiện n):

Nhóm lệnh n

else:

Nhóm lệnh x

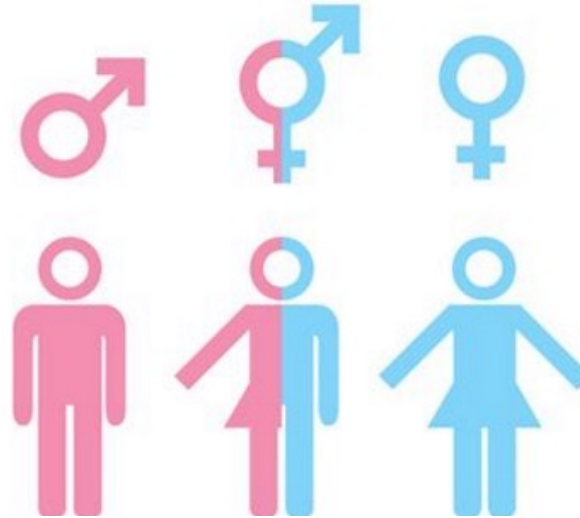


(Kiểm tra điều kiện1 đúng thì thực hiện nhóm lệnh 1, nếu sai kiểm tra điều kiện 2 đúng thực hiện nhóm lệnh 2 ...nếu tất cả các điều kiện sai thực hiện nhóm lệnh x)

Ví dụ: Chào theo giới tính

Nhập vào giới tính (0=Nam | 1=Nữ)

- Nếu nhập vào 0 → Chào anh đẹp trai!
- Nếu nhập vào 1 → Chào chị xinh gái!
- Nếu nhập khác 0 hoặc 1 → Cảnh báo: Giới tính không xác định!



Nhập giới tính (0:Nam - 1:Nữ):0
Chào anh đẹp trai!

Nhập giới tính (0:Nam - 1:Nữ):1
Chào chị xinh gái!

Nhập giới tính (0:Nam - 1:Nữ):2
Cảnh báo: Giới tính không xác định!

Các dạng câu lệnh điều kiện (t)



VINBIGDATA



Academy
Vietnam

- Dạng 4 (if lồng nhau – Nested if):

if (điều kiện1):

 if (điều kiện 1.1):

Nhóm lệnh 1.1

 elif (điều kiện 1.2):

Nhóm lệnh 1.2

 else:

Nhóm lệnh 1.x

else:

nhóm lệnh 2

```
num = float(input("Nhập một số: "))
if num >= 0:
    if num == 0:
        print("Số Không")
    else:
        print("Số dương")
else:
    print("Số âm")
```

Nhập một số: 0
Số Không

Thực hành

3. Cấu trúc vòng lặp trong Python

- Cũng như câu lệnh điều kiện, Câu lệnh vòng lặp là một trong những câu lệnh cơ bản của bất cứ ngôn ngữ lập trình nào.
- Để giải quyết bài toán, chúng ta cần thực hiện một công việc nào đó lặp đi lặp lại rất nhiều lần. Số lần lặp đó có thể biết trước hoặc không biết trước.

Ngôn ngữ tự nhiên	Ngôn ngữ lập trình
Tính tổng các số từ 1 đến 10: <i>1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10</i>	tong = 0 for i in range(1,11): <i>tong = tong + i</i> <i>print('Tổng từ 1 đến 10 là:', tong)</i>

Có 2 kiểu vòng lặp trong python:

- * Vòng lặp while
- * Vòng lặp for

Vòng lặp While

- Vòng lặp while sử dụng khi **không biết trước số lần lặp**.
- Cú pháp:

While <điều kiện>:
Nhóm lệnh 1

```
1 n = int(input('Em sinh tháng mấy?'))
2 i=1
3 while(i<=n):
4     print(i, ') I Love You!')
5     i=i+1
6
7 #Câu Lệnh Lặp ngoài vòng Lặp while
8 print('-----AIACADEMY-----')
```

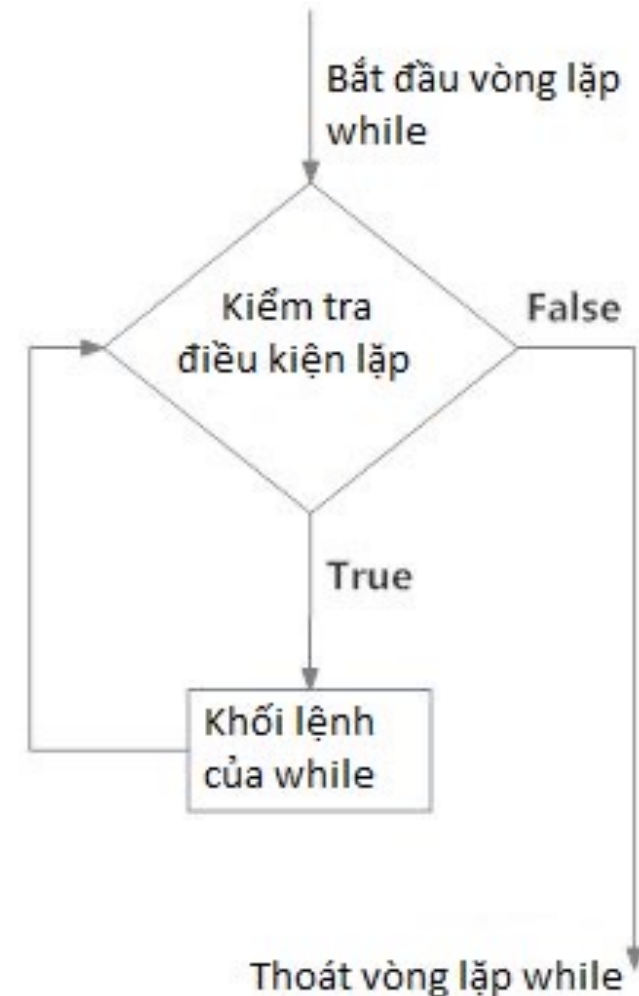
Em sinh tháng mấy?3

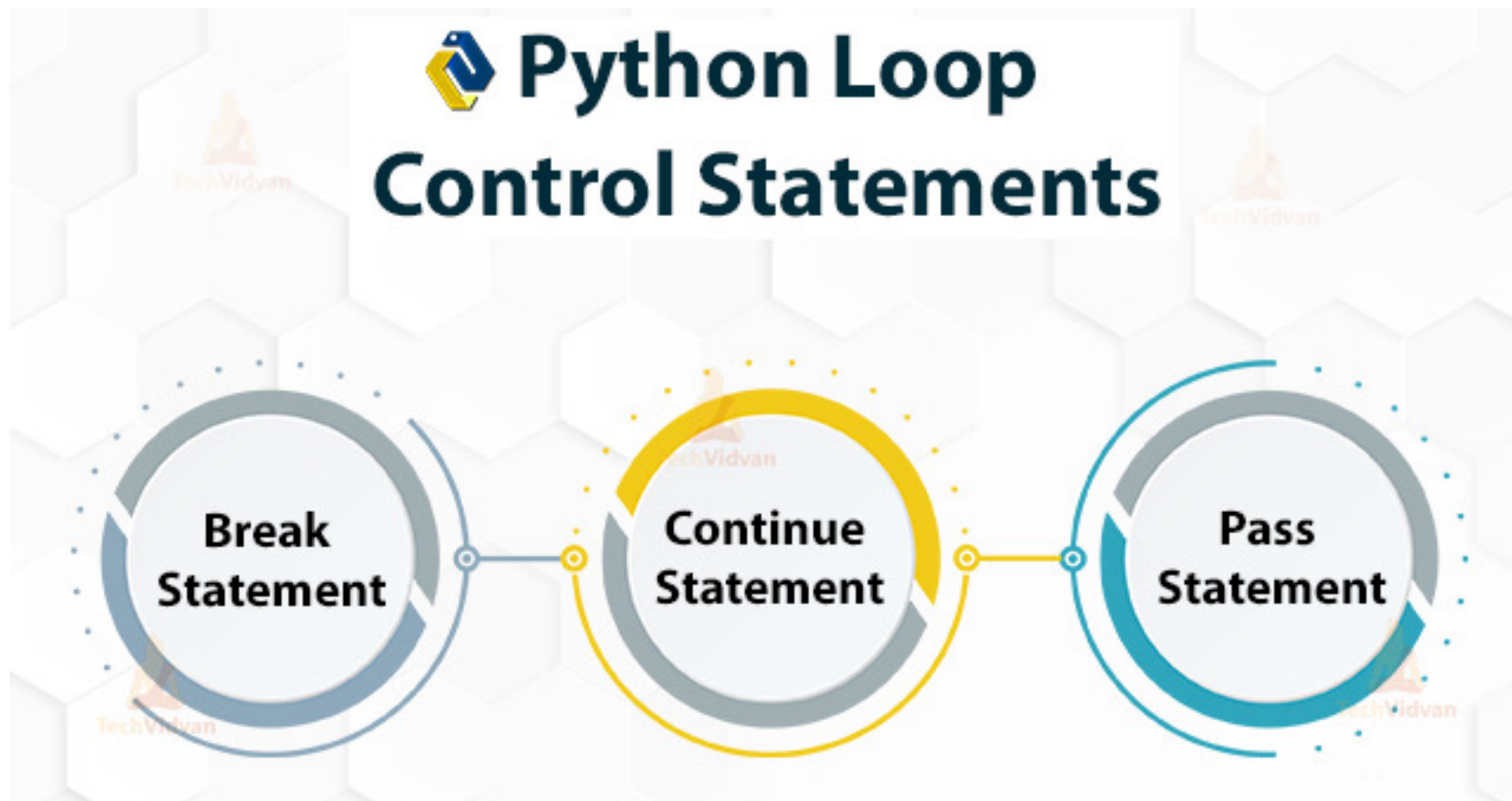
1) I Love You!

2) I Love You!

3) I Love You!

-----AIACADEMY-----





Lệnh break

- Lệnh **break kết thúc vòng lặp** chứa nó và truyền điều khiển đến lệnh tiếp theo sau khối lệnh của vòng lặp đó.

```
1 n = int(input('Em sinh tháng mấy? '))
2 i=1
3 while(i<=n):
4     print(i, ') I Love You!')
5
6     #Chỉ hiển thị tối đa 3 Lần
7     if (i==3):
8         break #Thoát ra khỏi vòng lặp while
9
10    i=i+1
11
12 #Câu Lệnh Lặp ngoài vòng Lặp while
13 print('-----AIACADEMY-----')
```

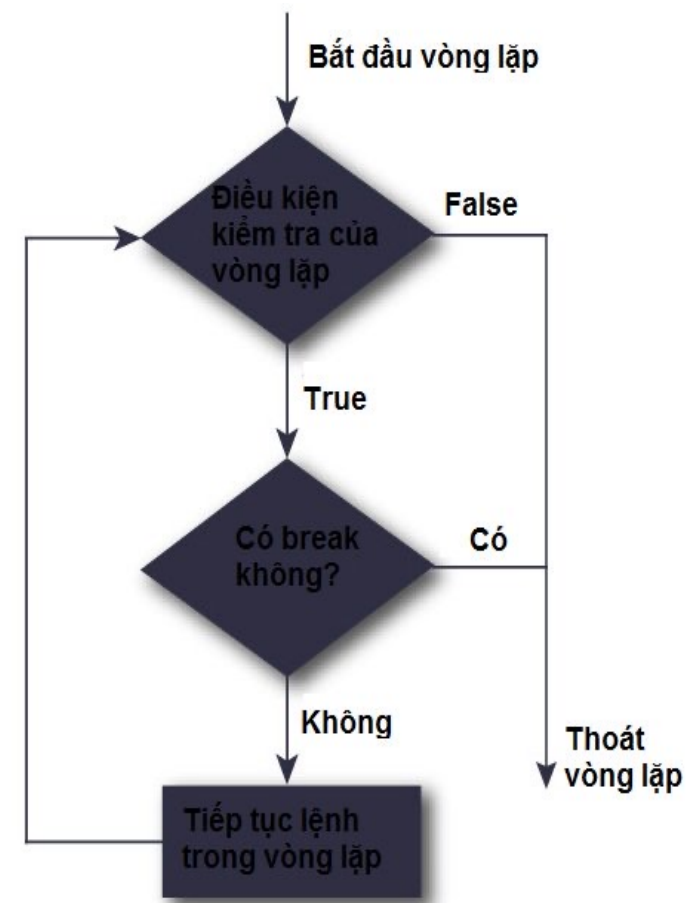
Em sinh tháng mấy? 8

1) I Love You!

2) I Love You!

3) I Love You!

-----AIACADEMY-----

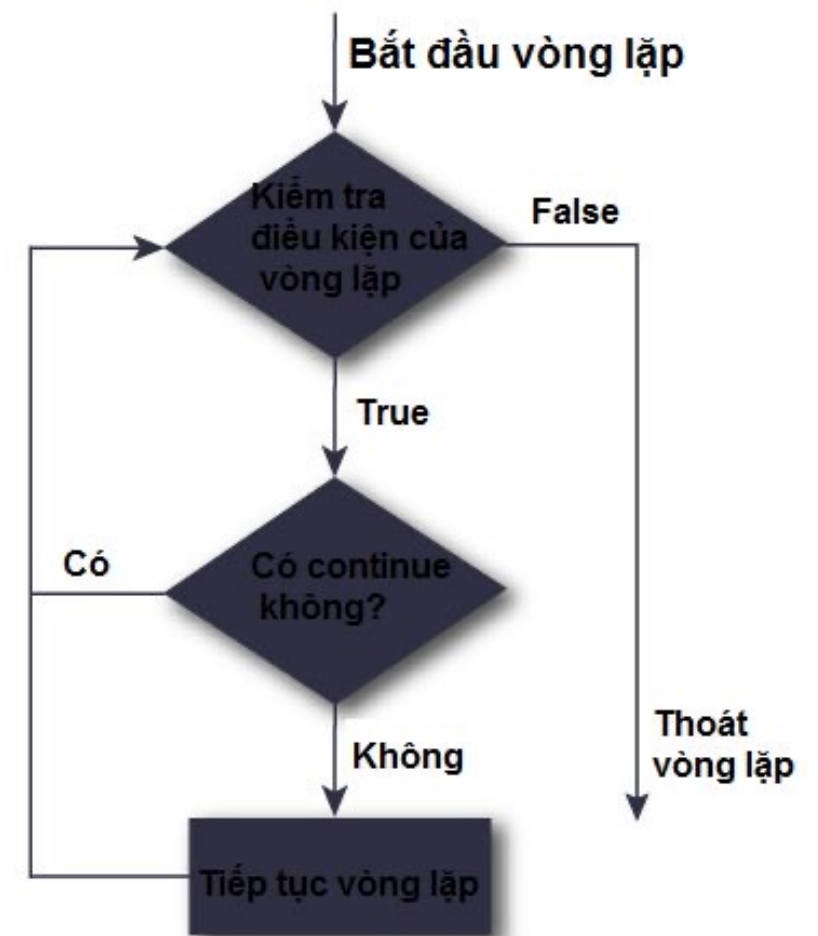


Lệnh continue

- Lệnh **continue** bỏ qua phần còn lại của khối lệnh bên trong vòng lặp, áp dụng cho **vòng lặp tiếp theo**. Nghĩa là vòng lặp không chấm dứt nó sẽ tiếp tục với số lần lặp kế tiếp

```
1 n = 20
2 i = 1
3 while (i<=n):
4     i = i+1
5     if (i%3!=0):
6         continue
7         #Bỏ qua các câu lệnh phía sau nếu ko chia hết cho 3
8     print(i)
9
10 #Câu Lệnh Lặp ngoài vòng Lặp while
11 print('-----AIACADEMY-----')
```

```
3
6
9
12
15
18
21
-----AIACADEMY-----
```



Lệnh while ...else

- Cấu trúc:

```
while <điều kiện>:  
    <Nhóm lệnh A>  
else:  
    <Nhóm lệnh B>
```

- Thực hiện nhóm lệnh A cho đến khi điều kiện còn đúng.
- Nếu điều kiện sai thực hiện nhóm lệnh B.
(ít nhất thực hiện nhóm lệnh B 1 lần)

```
1 counter = 4  
2 while counter < 3:  
3     print("Inside loop")  
4     counter = counter + 1  
5 else:  
6     print("Inside else")  
7 #-----  
8 print('----Bên ngoài vòng lặp while----')
```

Inside else

----Bên ngoài vòng lặp while----

Lệnh while True

- Cấu trúc:

while True:

<Nhóm lệnh thực hiện khi điều kiện đúng>

if <điều kiện dừng>:

break

Với cấu trúc này các **câu lệnh sẽ thực hiện lặp đi lặp lại**, cho đến khi biểu thức <điều kiện dừng> thỏa mãn. Lúc đó câu lệnh **if** sẽ giúp cho lệnh **break** được thực thi và dừng vòng lặp.

```
1  #chỉ cho phép nhập tháng sinh 1 - 12
2  while True:
3      n = int(input('Em sinh tháng mấy? '))
4      if (1<= n <= 12):
5          #'Tháng sinh nhập vào hợp lệ!'
6          break;
7      print('Tháng không đúng, vui lòng nhập lại')
8  #Câu lệnh ngoài vòng lặp while
9  print('Chào em cô gái tháng ', n)
```

```
Em sinh tháng mấy? 15
Tháng không đúng, vui lòng nhập lại
Em sinh tháng mấy? 10
Chào em cô gái tháng  10
```

- Vòng lặp for sử dụng khi **biết trước số lần lặp**.
- Cú pháp:

for <biến chạy> in <dãy>:

Nhóm lệnh 1

<Biến chạy> sẽ lần lượt nhận các giá trị của các thành phần có trong <dãy>. Dãy có thể là một danh sách (list), chuỗi ký tự (str), dãy số....

```
1  #Tính 10! = 1*2*3*4*5*6*7*8*9*10
2  #Tổng 10 = 1+2+3+4+5+6+7+8+9+10
3  n = 10
4  tích = 1
5  tong = 0
6  for i in range(1, n+1):
7      #Mỗi lần lặp biến i tăng lên 1
8      tích = tích*i
9      tong = tong+i
10
11  print('10! = ', tích)
12  print('10+ = ', tong)
```

10! = 3628800

10+ = 55

Vòng lặp for (2)

- **Vòng lặp for với chuỗi ký tự:** Biến chạy sẽ lần lượt nhận các giá trị là các ký tự trong chuỗi ký tự.

```
1 #Vòng lặp for với chuỗi ký tự:  
2 st = 'HUMG IN MY MIND'  
3 for i in st:  
4     print('ký tự: ', i)
```

1

ký tự: H
ký tự: U
ký tự: M
ký tự: G
ký tự: I
ký tự: N
ký tự: M
ký tự: Y
ký tự: M
ký tự: I
ký tự: N
ký tự: D

```
1 #Đếm số ký tự M trong chuỗi  
2 st = 'HUMG IN MY MIND'  
3 dem = 0  
4 for i in st:  
5     if (i=='M'): dem=dem+1  
6 print('Số ký tự M có trong chuỗi là: ', dem)
```

2

Số ký tự M có trong chuỗi là: 3

- **Vòng lặp for với danh sách:** Biến chạy sẽ lần lượt nhận các giá trị là các phần tử trong danh sách.

```
1  #Vòng lặp for với danh sách
2  hoc_sinh = ['Lê Thùy Dung', 'Trần Đức Hùng',
3              'Nguyễn Lan Anh', 'Mai Phương Thúy',
4              'Trần Thanh Thủy', 'Kiều Thành Công']
5
6  print('Danh sách học sinh bao gồm:')
7  tt = 1
8  for i in hoc_sinh:
9      print( tt, ') ', i)
10     tt = tt+1
```

Danh sách học sinh bao gồm:

- 1) Lê Thùy Dung
- 2) Trần Đức Hùng
- 3) Nguyễn Lan Anh
- 4) Mai Phương Thúy
- 5) Trần Thanh Thủy
- 6) Kiều Thành Công

- **Vòng lặp for với lệnh range():** Lệnh range() trong Python kết hợp với vòng lặp for sẽ trở nên rất hữu hiệu trong việc kiểm soát giá trị bắt đầu, kết thúc và bước nhảy của biến chạy.

- Cú pháp:

```
for <biếnchạy> in range(<bắtđầu>,<kếtthúc>,<bướcnhảy>) :  
    nhóm lệnh thực hiện
```

- **<bắt đầu>** là giá trị khởi gán ban đầu cho biến chạy (mặc định = 0)
- **<kết thúc>** là giá trị kết thúc cho biến chạy, nhưng không bao gồm chính nó (< kết thúc)
- **<bước nhảy>** là giá trị mà biến nhảy tăng thêm sau mỗi lần lặp (mặc định = 1)

Vòng lặp for (4)

- Vòng lặp for với lệnh range():

```
1 #Lệnh for với range()  
2 for i in range(5):  
3     #Giá trị khởi đầu mặc định = 0  
4     #Bước nhảy mặc định = 1  
5     print('i = ',i)
```

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4
```

```
1 #Lệnh for với range(m,n)  
2 for i in range(5,10):  
3     #Giá trị khởi đầu m = 5  
4     #Giá trị kết thúc n = 10  
5     #Bước nhảy mặc định = 1  
6     print('i = ',i)
```

```
i = 5  
i = 6  
i = 7  
i = 8  
i = 9
```

```
1 #Lệnh for với range(m,n,d)  
2 for i in range(2,11,2):  
3     #Giá trị khởi đầu m = 2  
4     #Giá trị kết thúc n = 11  
5     #Bước nhảy d = 2  
6     print('i = ',i)
```

```
i = 2  
i = 4  
i = 6  
i = 8  
i = 10
```

Vòng lặp for lồng nhau

- Trong một số bài toán chúng ta cần kết hợp và sử dụng nhiều câu lệnh lặp đặt lồng nhau để giải quyết.


```
1 #Hiển thị bảng cửu chương từ 2 -> 9
2 for i in range(2,10):
3     print('Bảng cửu chương ', i)
4     for j in range(1,11):
5         print (i , ' x ', j, ' = ', i*j)
6     print('-----')
```

Bảng cửu chương 2


```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
2 x 10 = 20
```

Break, continue cho cả while và for


```
for var in sequence:
    # codes inside for loop
    if condition:
        break
    # codes inside for loop
# codes outside for loop
```




```
while test expression:
    # codes inside while loop
    if condition:
        break
    # codes inside while loop
# codes outside while loop
```



```
for var in sequence:
    # codes inside for loop
    if condition:
        continue
    # codes inside for loop
# codes outside for loop
```

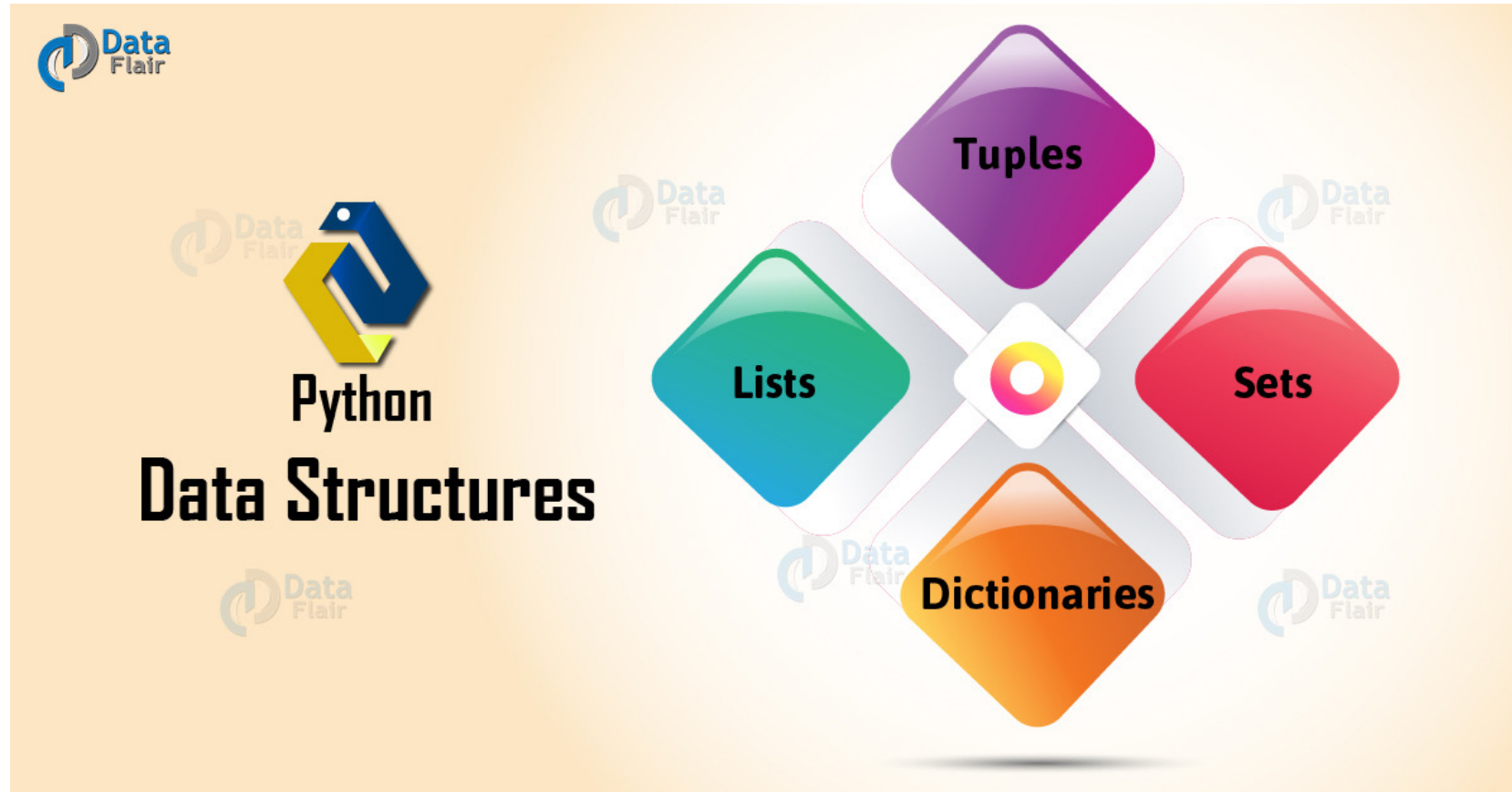


```
while test expression:
    # codes inside while loop
    if condition:
        continue
    # codes inside while loop
# codes outside while loop
```



Thực hành

4. Các cấu trúc dữ liệu cơ sở



List comprehension

- List comprehension là một cách dễ dàng, ngắn gọn và nhanh chóng để xây dựng một danh sách theo một điều kiện nào đó.
- List comprehension sử dụng một biểu thức đi kèm với lệnh for hoặc lệnh if được đặt trong cặp dấu ngoặc vuông []

Syntax

Example

[Output for **i** in list if condition]

[i**3 for i in [1,2,3,4] if i>2]

Variable

Filter Condition
(Optional)

```
1 #Tạo danh sách gồm 10 phần tử có giá trị là 3**x (x = [0,9])
2 list_cub3 = [3**x for x in range(10)]
3 list_cub3
```

[1, 3, 9, 27, 81, 243, 729, 2187, 6561, 19683, 59049]

```
1 #Tạo danh sách gồm các phần tử chia hết cho 4 nhỏ hơn 50
2 list_4 = [x for x in range(50) if x%4==0]
3 list_4
```

[0, 4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48]

Thực hành

4.2 Tuple

- Tuple giống như các list với ngoại lệ là dữ liệu một khi được nhập vào bộ dữ liệu không thể thay đổi bất kể điều gì.

LIST	TUPLE
Được sử dụng cho các loại dữ liệu đồng nhất	Thường được sử dụng cho các loại dữ liệu không đồng nhất
Có thể thay đổi trong môi trường	Bất biến trong môi trường giúp lặp lại nhanh hơn
Không có yếu tố bất biến	Các yếu tố bất biến có thể được sử dụng làm key cho từ điển
Không đảm bảo rằng dữ liệu được bảo vệ chống ghi	Việc thực hiện một bộ dữ liệu không thay đổi đảm bảo rằng nó được bảo vệ chống ghi

- Khai báo tuple:

<tên_danh_sách> = (<Phần tử 1>,<Phần tử 2>,....,<Phần tử n>)

```
1 #Khai báo biến Tuple
2 Tuple1 = (1,2,5,6)
3 Tuple2 = ('a', 'b', 'c', 'd')
4 Tuple3 = () #empty tuple
5 Tuple4 = 5,3,1
6 Tuple5 = ("London", "Tokyo", "Korea", 1986,1640, 1948)
7 print(Tuple1)
8 print(Tuple2)
9 print(Tuple3)
10 print(Tuple4)
11 print(Tuple5)
```

```
(1, 2, 5, 6)
('a', 'b', 'c', 'd')
()
(5, 3, 1)
('London', 'Tokyo', 'Korea', 1986, 1640, 1948)
```

```
1 #Lưu ý: nếu chỉ có một phần tử
2 Tup1 = (5)
3 Tup2 = (5,) #phải có thêm dấu ,
4 print(type(Tup1))
5 print(type(Tup2))
```

```
<class 'int'>
<class 'tuple'>
```

- Tuple là kiểu dữ liệu không thay đổi được nên nó chỉ có phương thức **count()** và **index()**

```
1 #Tuple ko cho phép thay đổi dữ liệu:
2 num = (1,2,3)
3 num[1] = 20 #Thay đổi phần tử tại index=1
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-53-e200330d0003> in <module>
      1 #Tuple ko cho phép thay đổi dữ liệu:
      2 num = (1,2,3)
----> 3 num[1] = 20 #Thay đổi phần tử tại index=1
```

TypeError: 'tuple' object does not support item assignment

```
1 #Tuple ko cho phép xóa dữ liệu:
2 tup = (1,2,3,4,5)
3 del tup[2] #Xóa phần tử tại index 2
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-54-2a4ac9218560> in <module>
      1 #Tuple ko cho phép xóa dữ liệu:
      2 tup = (1,2,3,4,5)
----> 3 del tup[2] #Xóa phần tử tại index 2
```

TypeError: 'tuple' object doesn't support item deletion

```
1 #count(n): đến số phần tử trong danh sách có giá trị n
2 (1,2,2,2,1,4,2).count(2)
```

4

```
1 #index(n): chỉ số phần tử đầu tiên trong tuple có giá trị n
2 (1, 3, 5, 7, 9, 7).index(7)
```

3

Tuple

```
1 #Truy cập tới các phần tử trong Tuple:
2 #Tương tự như với list
3 print(Tuple5)
4 print(Tuple5[0])
5 print(Tuple5[-1])
6 print(Tuple5[2:4])
7 print(Tuple5[3:])
```

('London', 'Tokyo', 'Korea', 1986, 1640, 1948)

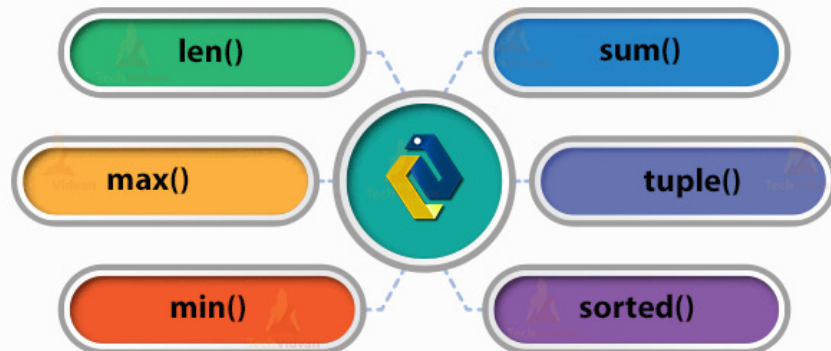
London

1948

('Korea', 1986)

(1986, 1640, 1948)

Python Tuple Functions



```
1 #Khai báo biến kiểu tuple:
2 tup = (4,8,9,0,5)
3 type(tup)
```

tuple

```
1 #1.Len: Số phần tử của tuple
2 print('1. len:',len(tup))
3 #2.max: phần tử lớn nhất của tuple
4 print('2. max:',max(tup))
5 #3.min: phần tử nhỏ nhất của tuple
6 print('3. min:',min(tup))
7 #4.sum: tổng các phần tử của tuple
8 print('4. sum:',sum(tup))
9 #5.sorted: sắp xếp các phần tử của tuple
10 print('5. sort:',sorted(tup))
```

```
1. len: 5
2. max: 9
3. min: 0
4. sum: 26
5. sort: [0, 4, 5, 8, 9]
```

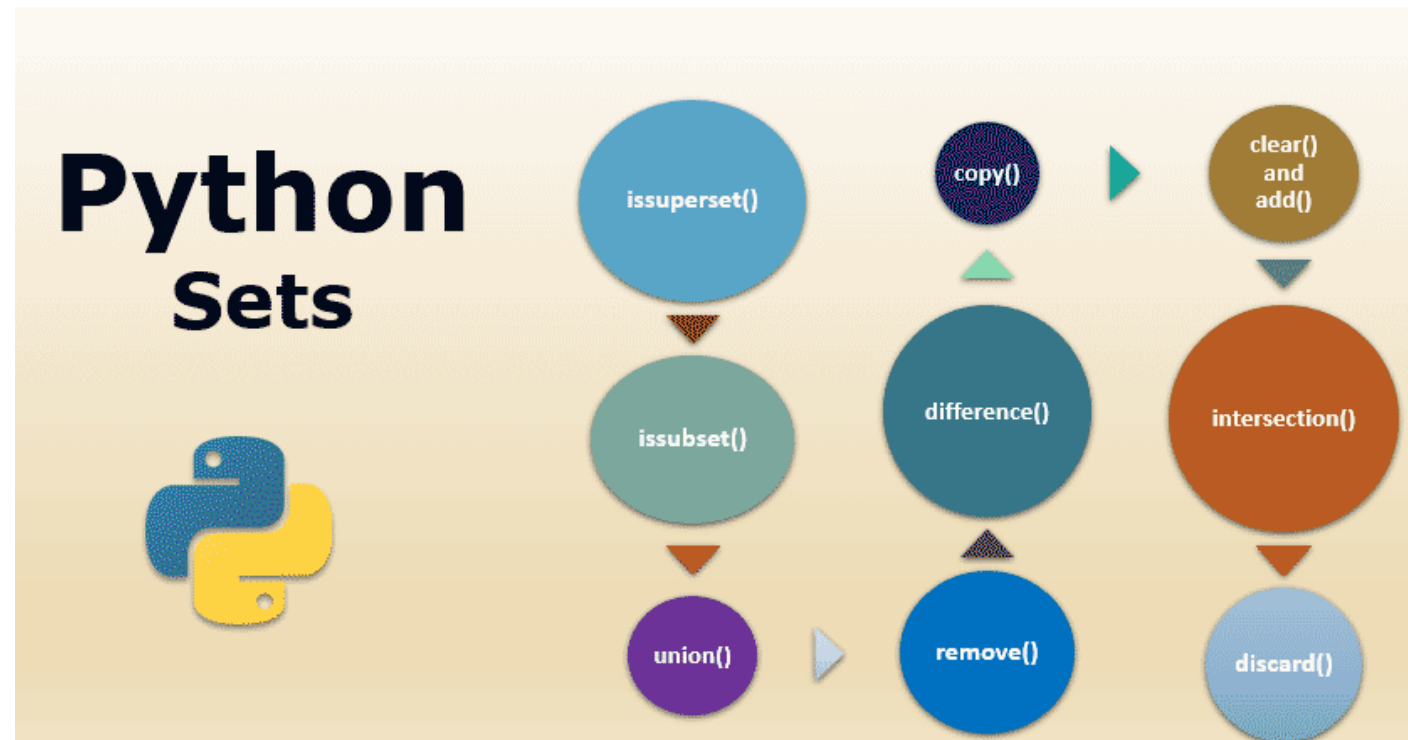
Thực hành

4.3 Set

- Set là bao gồm các phần tử theo thứ tự, không trùng nhau được bọc bởi hai dấu ngoặc nhọn {} hoặc khởi tạo bằng set().
- Các phần tử trong set không được đánh chỉ số

```
1 #Khai báo biến kiểu set
2 a = {1, 2, 7, 5, 5, 1, 3}
3 b = {99, 100, 200, 'yes', 'no'}
4 empty_set = set() #Khao báo set rỗng
5 print(a)
6 print(b)
7 print(empty_set)
8 print(type(empty_set))
```

```
{1, 2, 3, 5, 7}
{99, 100, 200, 'yes', 'no'}
set()
<class 'set'>
```



Phương thức của set

add()	Thêm một phần tử vào set.
clear()	Xóa tất cả phần tử của set.
copy()	Trả về bản sao chép của set.
difference()	Trả về set mới chứa những phần tử khác nhau của 2 hay nhiều set.
difference_update()	Xóa tất cả các phần tử của set khác từ set này.
discard()	Xóa phần tử nếu nó có mặt trong set.
intersection()	Trả về set mới chứa phần tử chung của 2 set.
intersection_update()	Cập nhật set với phần tử chung của chính nó và set khác.
isdisjoint()	Trả về True nếu 2 set không có phần tử chung.
issubset()	Trả về True nếu set khác chứa set này.
issuperset()	Trả về True nếu set này chứa set khác.
pop()	Xóa và trả về phần tử ngẫu nhiên, báo lỗi KeyError nếu set rỗng.
remove()	Xóa phần tử từ set. Nếu phần tử đó không có trong set sẽ báo lỗi KeyError.
symmetric_difference()	Trả về set mới chứa những phần tử không phải là phần tử chung của 2 set.
symmetric_difference_update()	Cập nhật set với những phần tử khác nhau của chính nó và set khác.
union()	Trả về set mới là hợp của 2 set.
update()	Cập nhật set với hợp của chính nó và set khác.

- Ví dụ

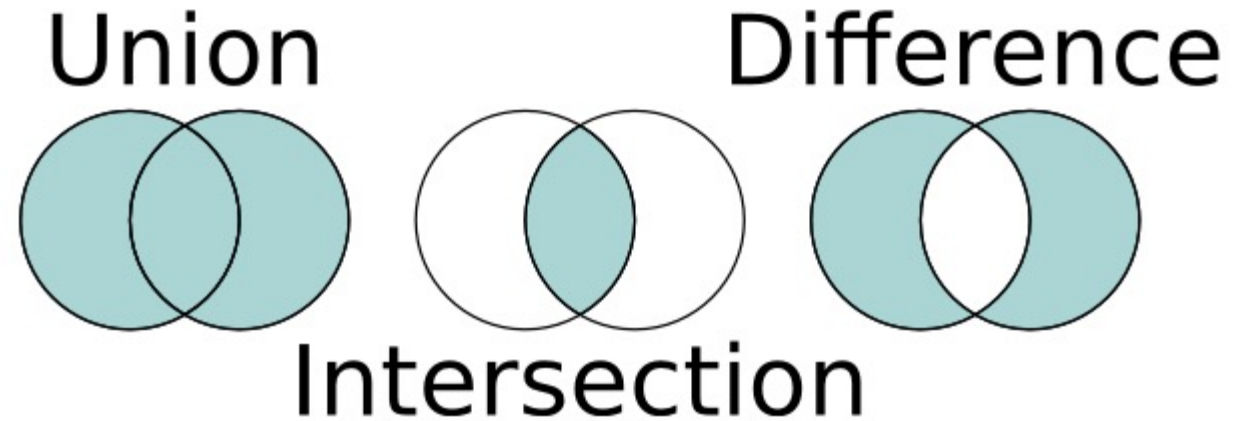
```
# initialize A and B
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

# Intersection of sets
# use & operator
print(A & B)

# use intersection function on A
print(A.intersection(B))

# Union of sets
# use | operator
print(A | B)

# use union function
print(A.union(B))
```

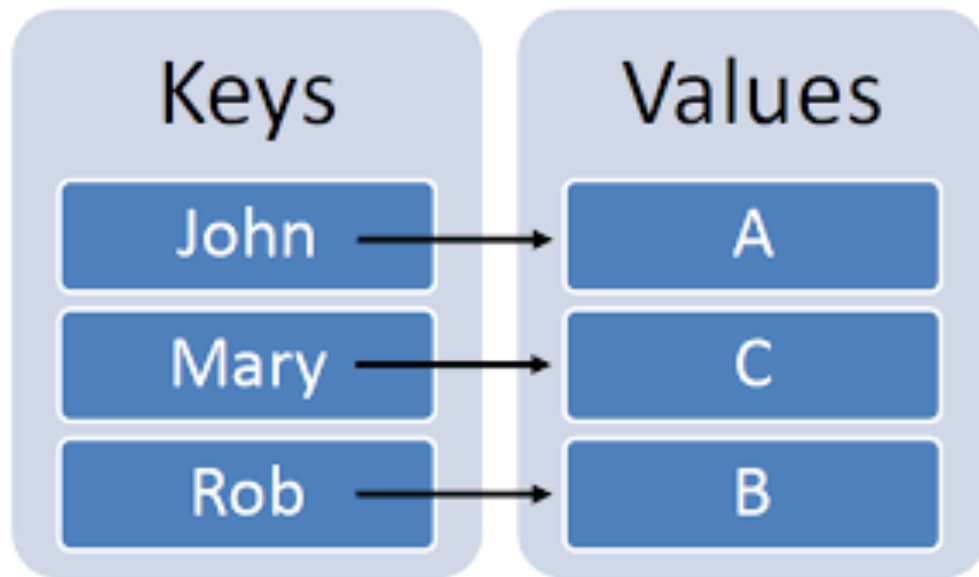


```
{4, 5}
{4, 5}
{1, 2, 3, 4, 5, 6, 7, 8}
{1, 2, 3, 4, 5, 6, 7, 8}
```

Thực hành

4.4 Dictionary

- Dictionary là tập hợp các cặp khóa - giá trị không có thứ tự.
- Nó thường được sử dụng khi chúng ta có một số lượng lớn dữ liệu.
- Các dictionary được tối ưu hóa để trích xuất dữ liệu với điều kiện bạn phải biết được khóa để lấy giá trị.



List

Index

0	Element 1
1	Element 2
2	Element 3
3	Element 4
.....

Element

Dictionary

Key: is a index by label

Key 1	Value 1
Key 1	Value 2
Key 2	Value 3
Key 3	Value 4
.....

Element/Values

- Ví dụ:

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict( {'name': 'John', 1: [2, 4, 3]})

# from sequence having each item as a pair
my_dict = dict([(1, 'apple'), (2, 'ball')])
```

Truy cập phần tử của dictionary:

- Các kiểu dữ liệu lưu trữ khác sử dụng index để truy cập vào các giá trị thì dictionary sử dụng các key.
- Key có thể được sử dụng trong cặp dấu ngoặc vuông hoặc sử dụng get().

```
my_dict = {'name': 'Jack', 'age': 26}
```

```
# Output: Jack
```

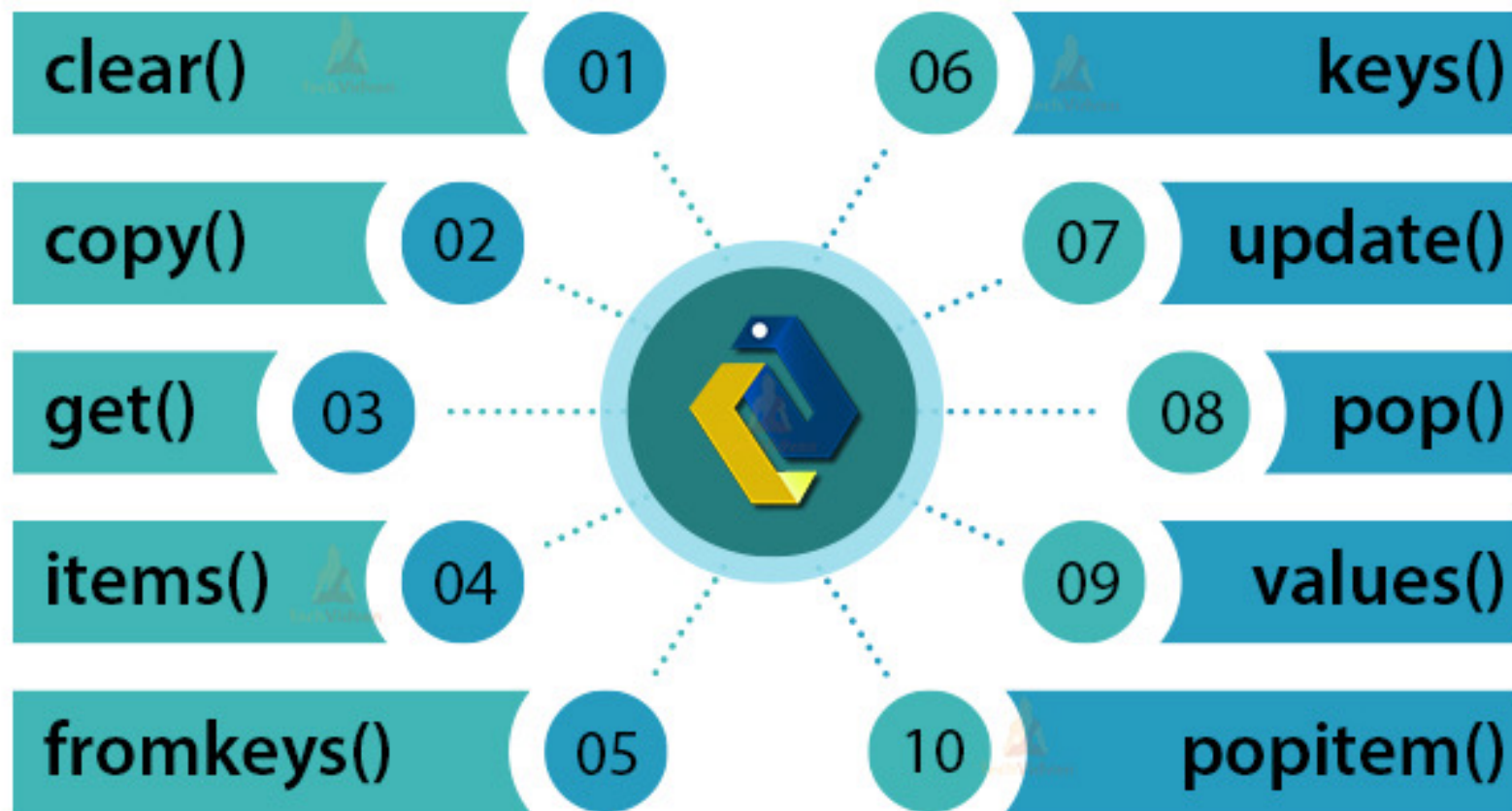
```
print(my_dict['name'])
```

```
# Output: 26
```

```
print(my_dict.get('age'))
```

- Dictionary có thể thay đổi, nên có thể thêm phần tử mới hoặc thay đổi giá trị của các phần tử hiện có bằng cách sử dụng toán tử gán.
- Nếu key đã có, giá trị sẽ được cập nhật, nếu là một cặp key: value mới thì sẽ được thêm thành phần tử mới.

Python Dictionary Methods



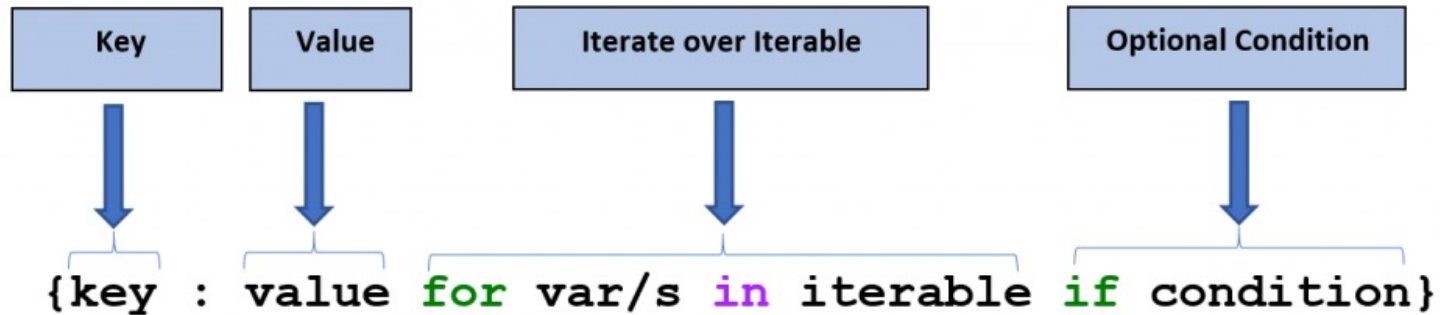
Phương thức của dictionary

clear()	Xóa tất cả phần tử của dictionary.
copy()	Trả về một bản sao shallow copy của dictionary.
fromkeys(seq[,v])	Trả về dictionary mới với key từ seq và value bằng v (default là None).
get(key[,d])	Trả về giá trị của key, nếu key không tồn tại, trả về d. (default là None).
items()	Trả lại kiểu xem mới của các phần tử trong dictionary (key, value).
keys()	Trả về kiểu xem mới của các key trong dictionary.
pop(key[,d])	Xóa phần tử bằng key và trả về giá trị hoặc d nếu key không tìm thấy. Nếu d không được cấp, key không tồn tại thì sẽ tạo lỗi KeyError.
popitem()	Xóa và trả về phần tử bất kỳ ở dạng (key, value). Tạo lỗi KeyError nếu dictionary rỗng.
setdefault(key[,d])	Nếu key tồn tại trả về value của nó, nếu không thêm key với value là d và trả về d (default là None).
update([other])	Cập nhật dictionary với cặp key/value từ <i>other</i> , ghi đè lên các key đã có.
values()	Trả về kiểu view mới của value trong dictionary.

Dictionary comprehension



VINBIGDATA



```
1 #Tự động tạo một biến dictionary theo quy định:  
2 dict_lapphuong = {x:x**3 for x in range(6)}  
3 print('output:', dict_lapphuong)
```

output: {0: 0, 1: 1, 2: 8, 3: 27, 4: 64, 5: 125}

```
1 dict_lapphuong_chan = {x:x**3 for x in range(10) if x%2==0}  
2 print('output:',dict_lapphuong_chan)
```

output: {0: 0, 2: 8, 4: 64, 6: 216, 8: 512}

Thực hành

5. Ngoại lệ (Exception)

- Ngoại lệ là một sự kiện, xảy ra trong quá trình thực thi chương trình làm gián đoạn luồng hướng dẫn bình thường của chương trình.
- Khi một tập lệnh trong python tạo ra một ngoại lệ, nó phải xử lý ngoại lệ đó ngay lập tức nếu không nó sẽ kết thúc và thoát.
- Một số ngoại lệ phổ biến trong Python:
 - ZeroDivisionError:
 - NameError:
 - IndentationError:
 - IOError:
 - EOFError:
 - ...

```
-----  
ZeroDivisionError                                Traceback (most recent call last)  
<ipython-input-4-7fec9efcd1af> in <module>  
      1 a = 10  
      2 b = 0  
----> 3 c = a/b  
      4 print('a/b = ',c)  
  
ZeroDivisionError: division by zero
```

```
-----  
NameError                                         Traceback (most recent call last)  
<ipython-input-2-780a7d5891ae> in <module>  
----> 1 print (A)  
  
NameError: name 'A' is not defined
```

```
File "<ipython-input-3-6f06c5334b49>", line 3  
    print(---)  
      ^  
IndentationError: unexpected indent
```

- Xử lý ngoại lệ:

Python try-except

Error & Exceptions Handling

try :

{ Execute/Run this code

except :

{ Execute this block when
exception occurred

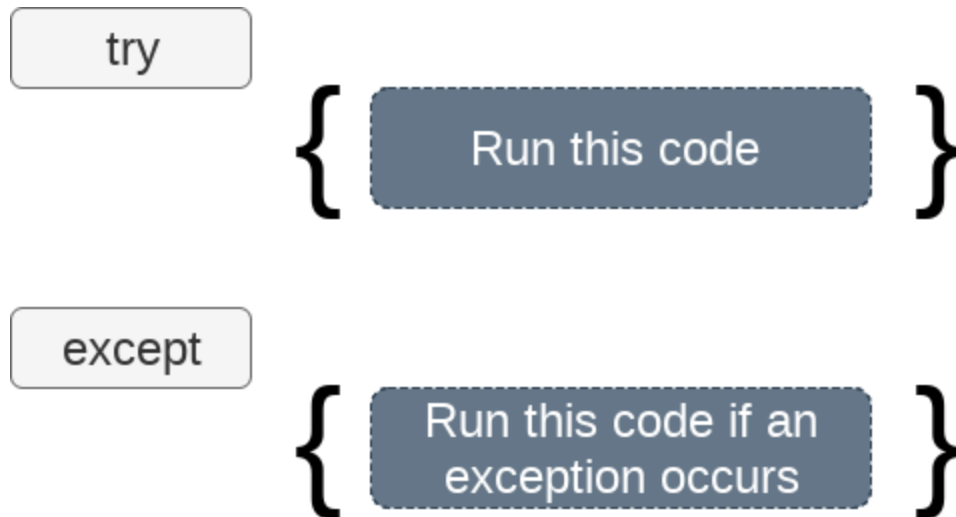
else :

{ If no exception run this
code

finally :

{ Always run this block of
code

- Xử lý ngoại lệ:



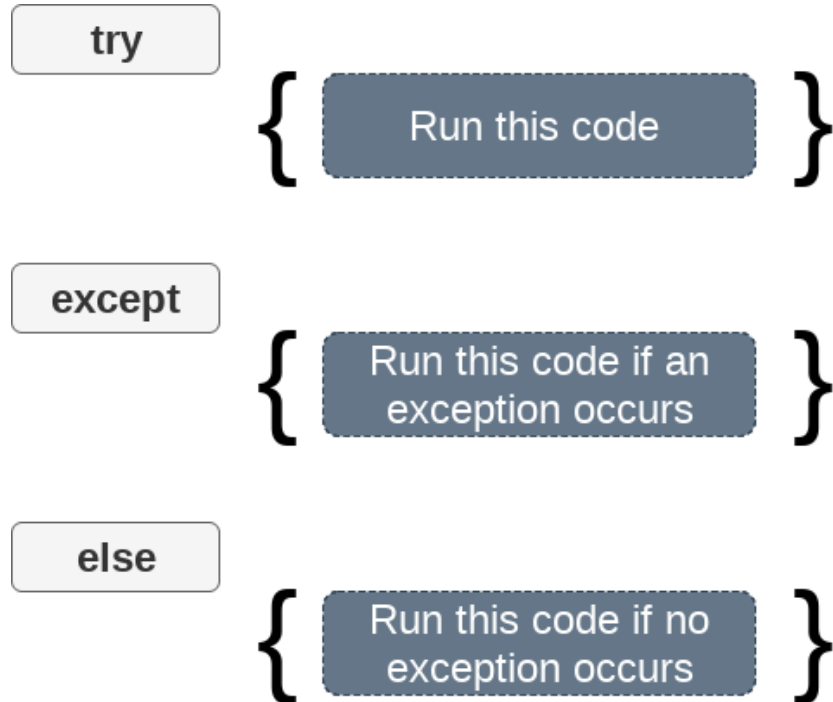
```
1 #try....except
2 try:
3     a = int(input("Enter a:"))
4     b = int(input("Enter b:"))
5     c = a/b
6 except:
7     print("Can't divide with zero")
```

```
Enter a:10
Enter b:0
Can't divide with zero
```

```
1 #Hiển thị ngoại lệ:
2 try:
3     fh = open("testfile", "r")
4     fh.write("This is my test file for exception handling!!")
5 except Exception as e:
6     print(e)
```

```
[Errno 2] No such file or directory: 'testfile'
```

- Xử lý ngoại lệ:

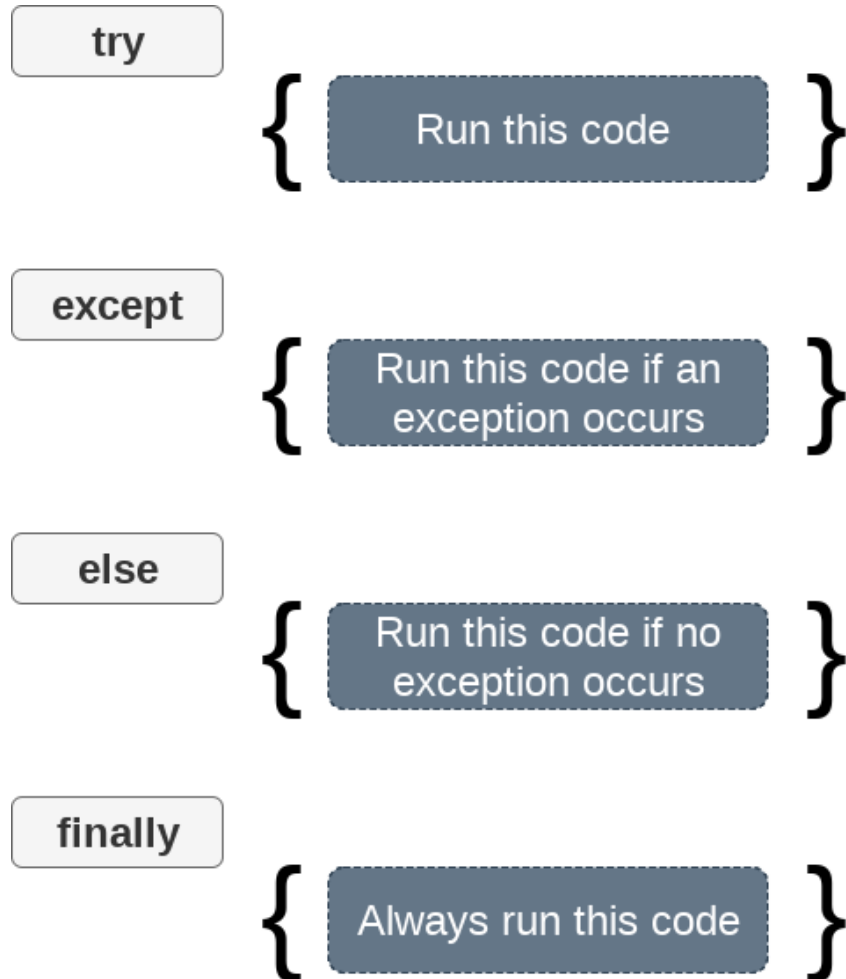


```
1 #Try....except...else:
2 try:
3     fh = open("testfile.txt", "r")
4     fh.write("This is my test file for exception handling!!")
5 except Exception as e:
6     print("Error: can't find file or read data")
7     print(e)
8 else:
9     print("Written content in the file successfully")
```

Error: can't find file or read data
not writable

- Một số lưu ý:
 - Một câu lệnh try duy nhất có thể có nhiều câu lệnh except.
 - Mã trong khối else thực thi nếu mã trong khối try không phát sinh ngoại lệ.
 - Khối lệnh else là không lệnh không cần sự bảo vệ của khối try

- Xử lý ngoại lệ:



- Khối finally là nơi để đặt khối lệnh cần phải được thực thi, cho dù khối try có đưa ra ngoại lệ hay không.

```
1  ##Try....except...else...finally
2  try:
3      fh = open("testfile.txt", "r")
4      fh.write("This is my test file for exception handling!!")
5  except IOError as e:
6      print("Error: can't find file or read data")
7      print(e)
8  else:
9      print("Written content in the file successfully")
10 finally:
11     fh.close()
12     print("End process....")
```

```
Error: can't find file or read data
not writable
End process....
```

Q & A
Thank you!