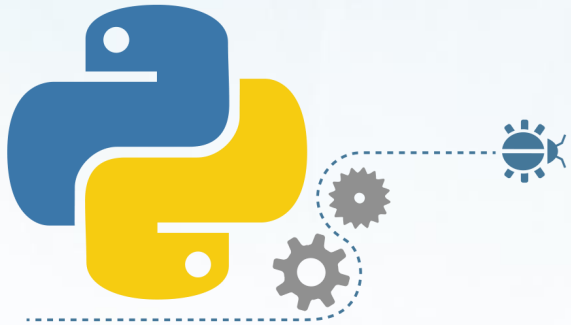
TRƯỜNG ĐẠI HỌC
SƯ PHẠM KỸ THUẬT TP. HỒ CHÍ MINH
HCMC University of Technology and Education

KHOA CÔNG NGHỆ THÔNG TIN
BỘ MÔN HỆ THỐNG THÔNG TIN

NHẬP MÔN LẬP TRÌNH PYTHON (IPPA233277)

CẤU TRÚC RẼ NHÁNH VÀ ĐIỀU KIỆN



GV. Trần Quang Khải

1. Hiểu và vận hành được các cấu trúc điều kiện: boolean, if, else, elif
2. Nắm được biểu thức pass
3. So sánh được số thực trong Python

1. Biểu thức Boolean
2. Biểu thức if
3. Biểu thức if-else
4. Biểu thức if-else lồng nhau
5. Biểu thức pass
6. So sánh số thực trong python
7. Sử dụng if-else như phép gán

- Boolean expression còn được gọi là predicate
- Có giá trị True/False

Ví dụ:

```
a = True
b = False
print('a =', a, ' b =', b)
# gán lại kết qua cho a
a = False
print('a =', a, ' b =', b)
```

➔ a = True b = False

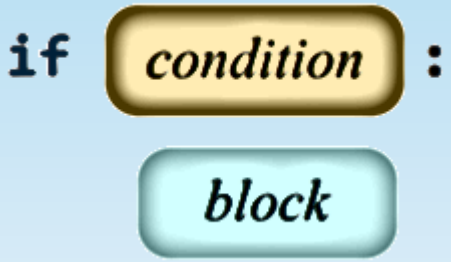
➔ a = False b = False

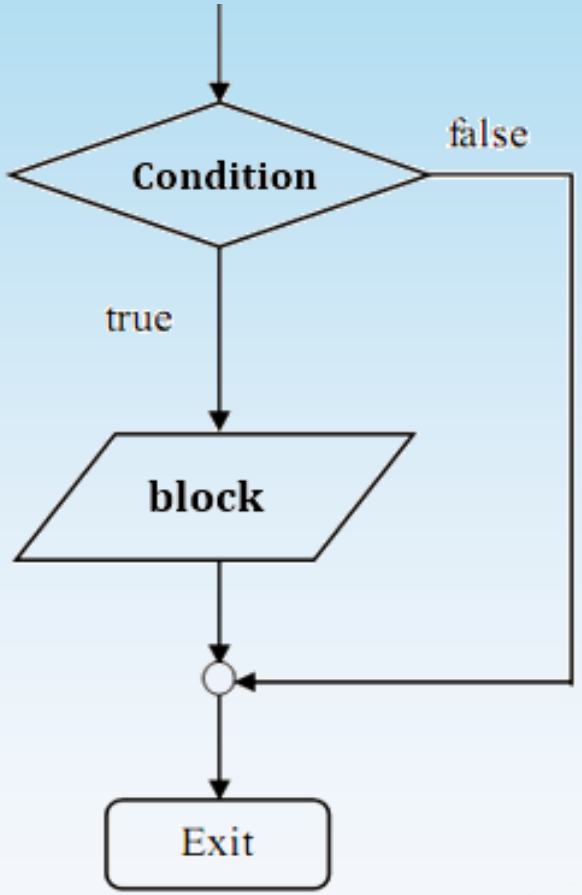
Biểu thức	Ý nghĩa
x == y	True nếu x = y , False nếu x khác y
x < y	True nếu x < y , False nếu x >= y
x <= y	True nếu x <= y , False nếu x > y
x > y	True nếu x > y , False nếu x <= y
x >= y	True nếu x >= y , False nếu x < y
x != y	True nếu x khác y , False nếu x = y

- Biểu thức if là một biểu thức điều kiện rất quan trọng và phổ biến trong python
- Biểu thức if đứng một mình chỉ quan tâm tới điều kiện đúng (True). Khi điều kiện đúng thì khối lệnh bên trong if sẽ được thực thi

Cú pháp:

```
if expression:
    # if-block
```





Ví dụ:

```
diem_trung_binh = float(input("Nhập điểm trung bình: "))
if diem_trung_binh >= 5:
    print("Đạt")
    print("Chúc mừng bạn!")
```

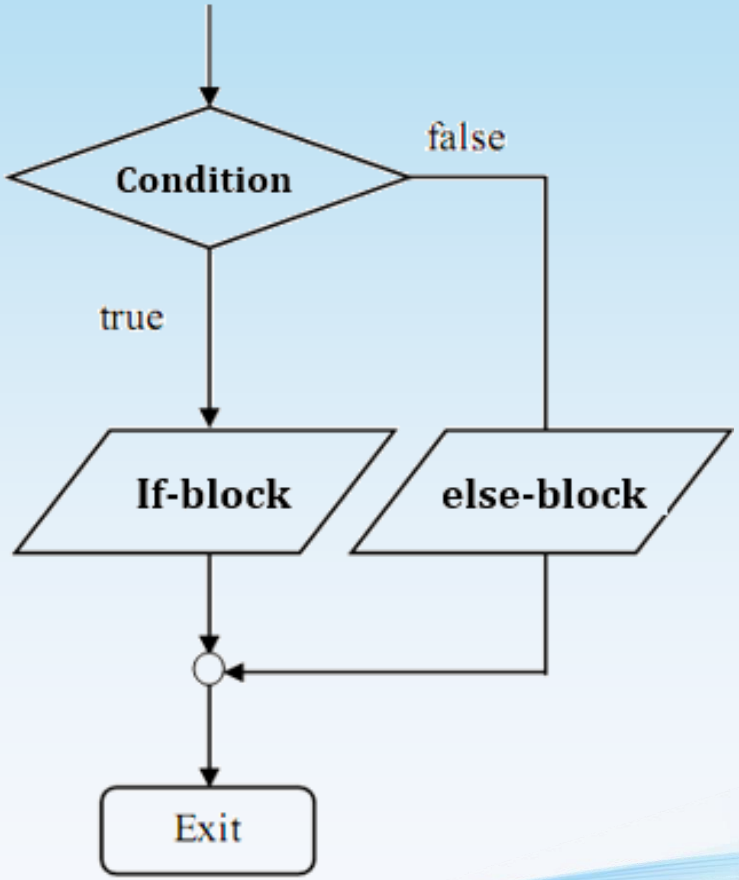
Lưu ý: Python không dùng ngoặc ngọn để bao bọc khối lệnh mà xác định bằng tab hoặc khoảng trắng thụt đầu dòng

- Biểu thức if-else là một biểu thức điều kiện rất quan trọng và phổ biến trong python
- Biểu thức if-else quan tâm tới điều kiện đúng (True) và sai (False). Khi điều kiện đúng thì khối lệnh bên trong if sẽ được thực thi, ngược lại nếu điều kiện sai thì khối lệnh trong else sẽ được thực thi

Cú pháp:

```
if expression:
    # if-block
else:
    # else-block
```

```
if condition :
    if-block
else:
    else-block
```



Ví dụ:

```
diem_trung_binh = float(input("Nhập điểm trung bình: "))
if diem_trung_binh >= 5:
    print("Đạt")
    print("Chúc mừng bạn!")
else:
    print("Chưa đạt")
    print("Chúc bạn cố gắng kỳ sau")
```


- Với các điều kiện phức tạp, python cũng hỗ trợ kiểm tra điều kiện if elif lồng nhau

Cú pháp:

```
if expression:
    # If-block
elif 2-expression:
    # 2-if-block
elif 3-expression:
    # 3-if-block
…
elif n-expression:
    # n-if-block
```

Ví dụ:

```python
diem_trung_binh = float(input("Nhập điểm trung bình: "))
if diem_trung_binh >= 9:
    print("Bạn xếp loại giỏi")
elif diem_trung_binh >= 7:
    print("Bạn xếp loại khá")
elif diem_trung_binh >= 5:
    print("Bạn xếp loại trung bình")
else:
    print("Chưa đạt")
```


- Biểu thức pass khá hữu dụng, dùng để dành chỗ lập trình.
- Khi phải viết rất nhiều dòng code, nhưng tại thời điểm đó chưa kịp làm, ta dùng pass để đánh dấu vị trí

```
a = float(input("Nhập hệ số a: "))
b = float(input("Nhập hệ số b: "))

if a == 0:
    # lỗi cú pháp nếu bỏ trống
else:
    x = b/a
    print("{0}x + {1} = 0".format(a, b))
    print("có nghiệm x = ", x)
```

```
a = float(input("Nhập hệ số a: "))
b = float(input("Nhập hệ số b: "))

if a == 0:
    pass # hợp lệ
else:
    x = b/a
    print("{0}x + {1} = 0".format(a, b))
    print("có nghiệm x = ", x)
```


- Khi thao tác với số thực cần để ý về việc sai số hoặc độ chính xác của kết quả.
- Mặc định, số thực (float) trong python có độ chính xác tối đa 15 chữ số phần thập phân

Ví dụ:

```
a = 1.1 + 2.2
print(a)
print("a == 3.3?", (a == 3.3))
```

➔ 3.3000000000000003

➔ False

Rõ ràng, kết quả không phải là **3.3** mà là **3.3000000000000003**. Điều này là do cơ chế lưu trữ dấu chấm động của phần thập phân trong máy tính bằng các số nhị phân 0 và 1 (do máy tính chỉ hiểu bit 0 và 1).

Ví dụ, số thập phân **0.1** được biểu diễn ở hệ nhị phân sẽ là một dãy nhị phân dài vô hạn là **0.0001100110011001100110011….** Tuy nhiên, máy tính của chúng ta chỉ lưu trữ một số hữu hạn các số nhị phân để biểu diễn số thập phân **0.1**. Do đó, bất cứ số thập phân nào cũng mang **tính chất chính xác tương đối**.

- Sử dụng module decimal được cài đặt sẵn trong python để xác định độ chính xác là bao nhiêu số thập phân

Ví dụ:

```
import decimal
print(decimal.Decimal(0.05))    # 0.05000000000000000277555756156289135105907917022705078125
print(decimal.Decimal('0.05')) # 0.05
```

hoặc

```
import decimal
a = 1.1 + 2.2
print(a) # 3.3000000000000003
b = decimal.Decimal('1.1') + decimal.Decimal('2.2')
print(b) # 3.3
```


- Để kiểm soát độ chính xác phần thập phân có bao nhiêu chữ số, sử dụng hàm getcontext() với thuộc tính prec

Ví dụ:

```
import decimal
print(decimal.Decimal(1) / decimal.Decimal(13)) # 0.076923076923076923076923076923
decimal.getcontext().prec = 10
print(decimal.Decimal(1) / decimal.Decimal(13)) # 0.07692307692
```

- Việc thực hiện if-else quá nhiều trong chương trình sẽ làm phức tạp hóa không cần thiết, do đó, ta cần chuyển đổi để đơn giản hóa dòng lệnh.

Cú pháp: *expression-1* **if** *condition* **else** *expression-2*

Ví dụ:

```
a = 5
b = 7
if a != b:
    c = 3
else:
    c = 2
print(c)
```

```
c = 3 if a != b else 2
```


- Python sử dụng định dạng code để suy ra các khối lập trình (block) trong chương trình
- Câu lệnh mở block kết thúc bằng dấu hai chấm (:) , câu lệnh phía sau buộc phải xuống dòng và lùi lề vào trong và có tối thiểu một câu lệnh để không bỏ trống block
- Những dòng lệnh cùng lề thì cùng một block
- Khi canh lền không nên sử dụng cả tab lẫn space
- Nên sử dụng 4 spaces để căn lề một block

Lưu ý: Sau khi sử dụng câu lệnh có dấu hai chấm (:) buộc phải xuống dòng và lùi lề vào trong. Tuy nhiên, vẫn có thể đi ngược lại điều này trong một vài trường hợp, tuy nhiên không được khuyến khích:

```
a = 3
if a - 1 > 0: print('a lớn hơn 1'); print('có thể a lớn hơn 2’)
```

- Python sử dụng khối lệnh try và except để xử lý các exception cho những trường hợp khi exception xảy ra, chương trình sẽ bị dừng đột ngột và thông báo lỗi

Ví dụ:

```
try:
    numb = 1/0
except:
    print("An exception occurred!")
```

➔ An exception occurred!

- Ngoài ra, cũng có thể bắt nhiều ngoại lệ khác nhau

Ví dụ:

```
try:
    numb = 1/0
except TypeError:
    print("Type Error")
except ZeroDivisionError:
    print("Zero Division Error")
except:
    print("An exception occurred!")
```

➔ Zero Division Error

- Có thể sử dụng từ khóa else để định nghĩa khối lệnh thực thi nếu không có exception nào xảy ra, hoặc không được thực thi nếu có bất kỳ exception nào

```
try:
    x = 1/1
    print(x)
except ZeroDivisionError:
    print("Cannot divide by 0!")
else:
    print("Nothing wrong.")
```

➔ 1.0

➔ Nothing wrong.

- Khối lệnh try với khối lệnh finally trong đó khối lệnh finally luôn được thực thi bất kể khối lệnh try có xảy ra exception hay không, thường dùng đóng các stream đọc/ghi tập tin hoặc kết nối đến database

Ví dụ:

```
try:
    x = 1/0
    print(x)
except ZeroDivisionError:
    print("Cannot divide by 0!")
finally:
    print("The 'try except' is finished!")
```

➔ Cannot divide by 0!

➔ The 'try except' is finished!

```
try:
    file = open("laptrinhPython.txt")
    try:
        file.write("Welcome to laptrinhPython.com!")
    except:
        print("Something went wrong when writing!")
    finally: # always close file
        file.close()
except:
    print("Something went wrong when opening!")
```

➔ Something went wrong when opening!

- Từ khóa raise trong python cho phép “ném” ra một exception nếu thỏa một điều kiện nào đó.

Ví dụ:

```
a = 1
b = 0
if b == 0:
    raise Exception("Sorry, cannot divide by 0!")
else:
    print("x = ", a/b)
```

➔ Traceback (most recent call last):

File "/Users/hodientuananh/Documents/workspace/python-hello-world/first-python.py", line 4, in <module>

raise Exception("Sorry, cannot divide by 0!")

Exception: Sorry, cannot divide by 0!

- ✓ Họ tên : Trần Quang Khải
- ✓ Email : khaitq@hcmute.edu.vn
- ✓ Zalo (mã Qr)