



Queen Mary
University of London

QHP4701

Introduction to Data Science Programming

Control Flow Tools: Program Development

Lecturer: Nikesh Bajaj, PhD

School of Physical and Chemical Sciences

<http://nikeshbajaj.in>

So far we have covered

- Familiar with Data Science Tasks.
- Python, Anaconda, Jupyter-notebook
- Data Types (int, float, str, None, bool)
- Collection types (list, dict, set, tuple)
- For-loop, linear algebra operations
- Numpy Arrays
- Reading Wave file, Image File, CSV file

Lecture Outline

Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loop

Comparison operators: revisit

- In Python, there are operators to compare two variables, e.g, x and y.

Algebraic operator	Python operator	Sample condition	Meaning
>	>	$x > y$	x is greater than y
<	<	$x < y$	x is less than y
\geq	>=	$x \geq y$	x is greater than or equal to y
\leq	<=	$x \leq y$	x is less than or equal to y
=	==	$x == y$	x is equal to y
\neq	!=	$x != y$	x is not equal to y

- Comparing two variables using these operators return **True** if condition passed else it return **False**
- $5 > 4 \rightarrow \text{True}$
- $5 == 4 \rightarrow \text{False}$

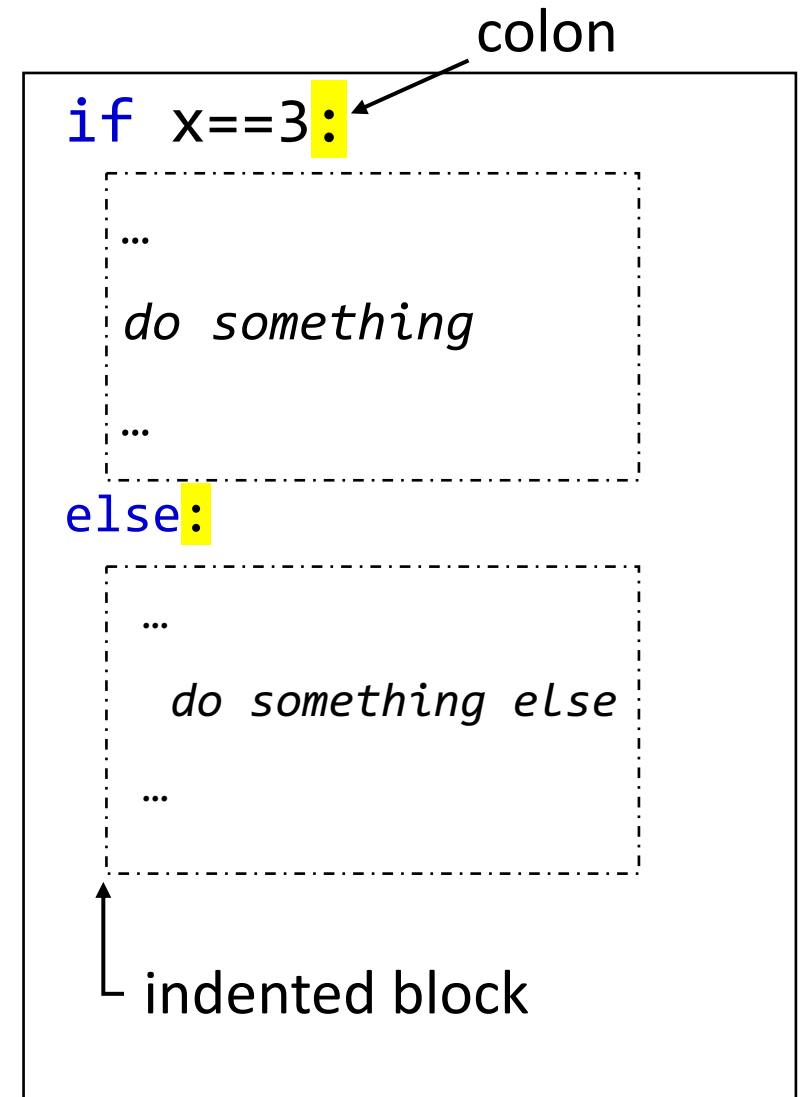
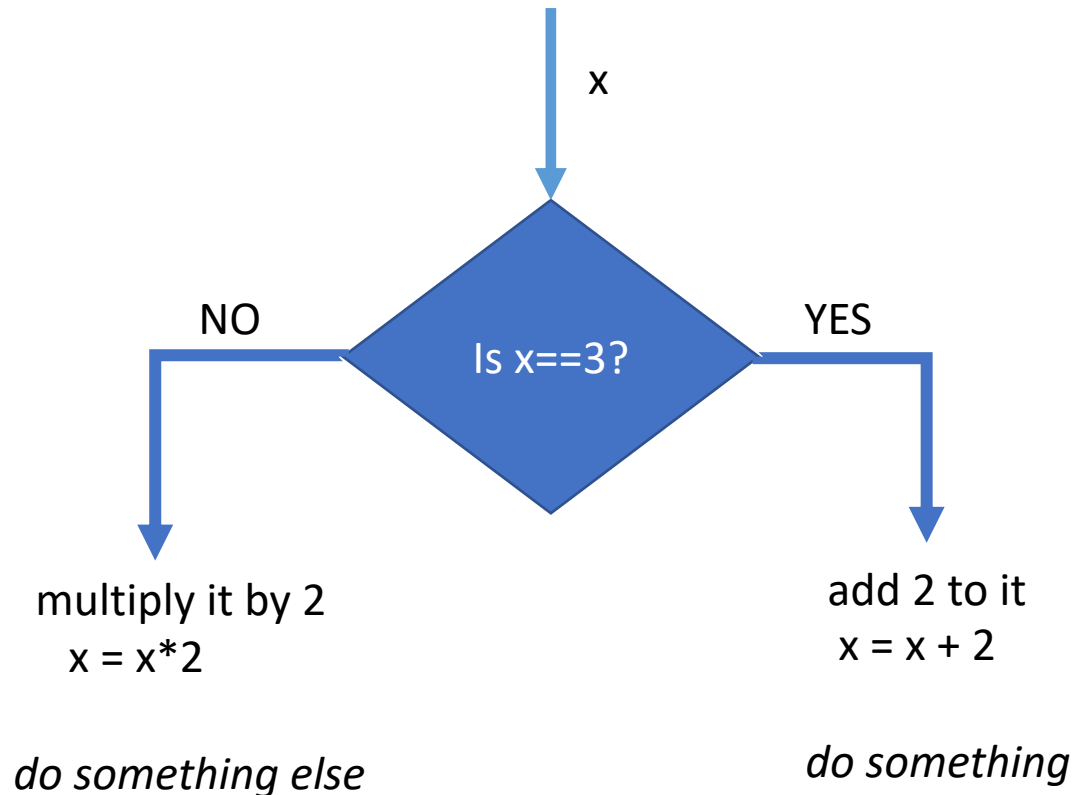
Lecture Outline

Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loop

Condition flow: if-else

- **If-else:** In programming languages, performing a test before doing something can be done by using (if-else) conditions.



Condition flow: if-else

If-else

examples

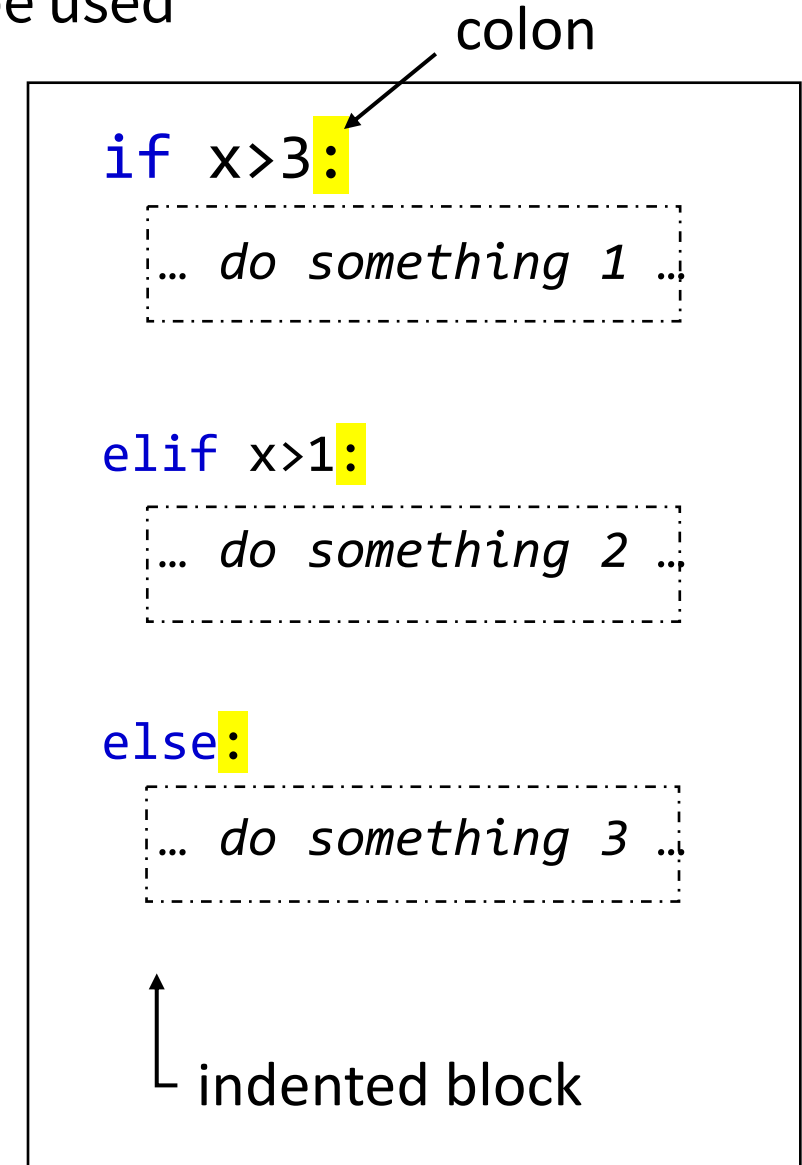
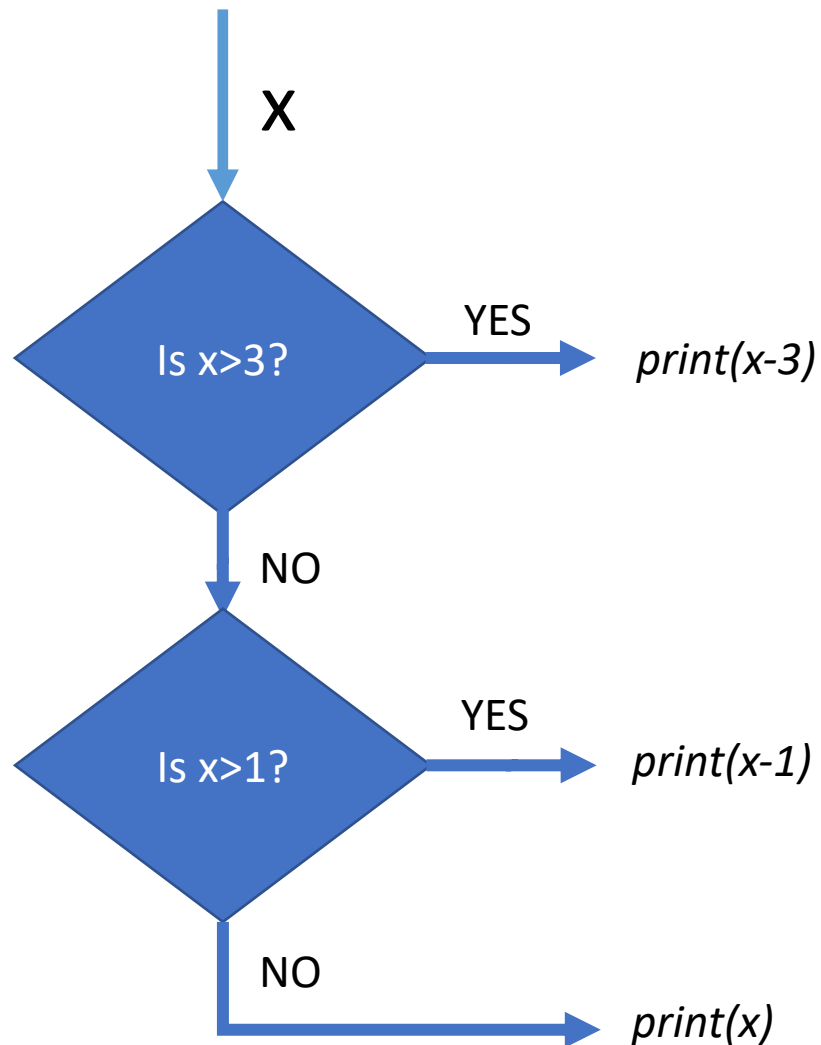
```
for x in range(10):  
    if x==3:  
        x = x+2  
    else:  
        x = x*2  
    print(x)
```

```
for x in range(10):  
    if x==3:  
        x = x+2  
  
    print(x)
```

else section is optional

Condition flow: if-elif-else

- **If-elif-else:** To use multiple conditions, elif (else-if) can be used



Condition flow: if-elif-else

- **If-elif-else:** examples

```
for x in range(10):  
    if x>3:  
        x = x-3  
    elif x>1:  
        x = x-1  
    else:  
        x = x+1  
    print(x)
```

```
for x in range(10):  
    if x==3:  
        x = x-3  
    elif x==4:  
        x = x-4  
    else:  
        x = x+1  
    print(x)
```

Important

- Conditions are tested sequentially, as a flow chart.
- If *'if'* condition satisfied, *'elif'* and *else* are not tested

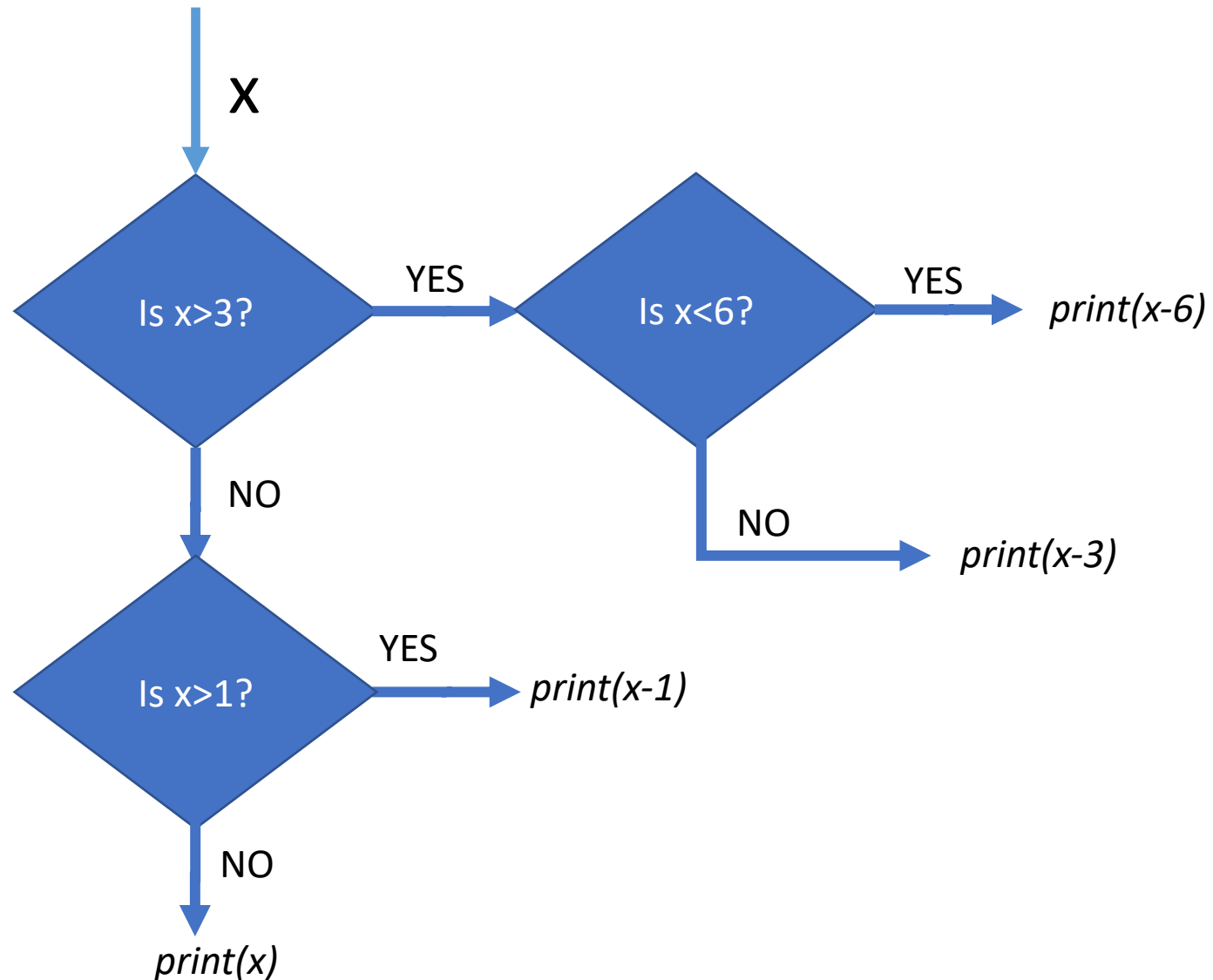
Condition flow: if-elif-else

- **If-elif-else:** Question - make a flow-chart to print a grade of a students as per table, with python expressions.

Marks	Grade
Above 90	A
Between 80 to 90	B
Between 70 to 80	C
Between 50 to 70	D
Below 50	F

Condition flow: Nested if-else

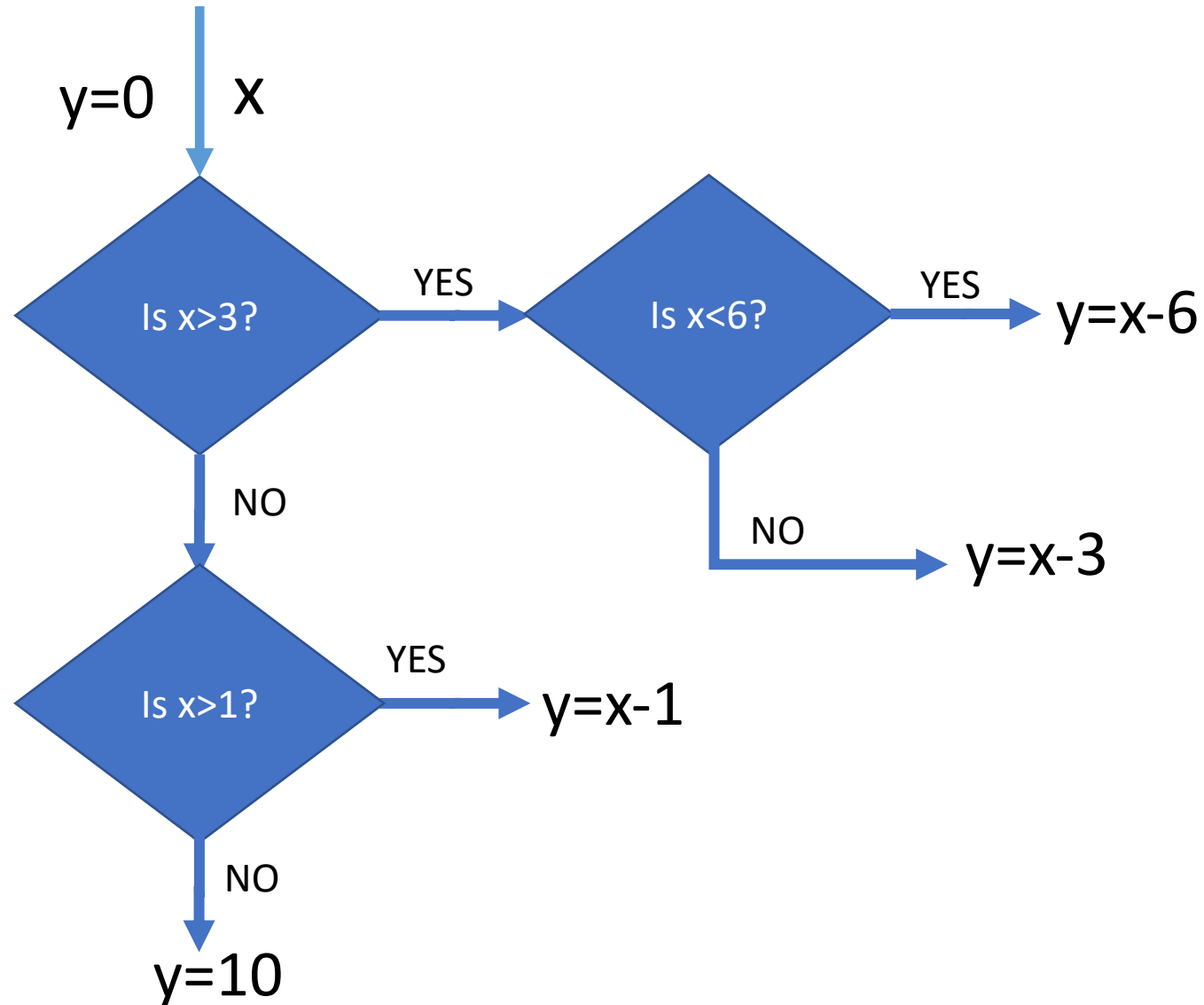
- **Nested If-else:** condition inside a condition



```
if x > 3:  
    if x < 6:  
        ... do 1...  
    else:  
        ... do 2...  
elif x > 1:  
    ... do 3...  
else:  
    ... do 4 ...
```

Condition flow: Nested if-else: Question

- What is value of y , if $x = 10$?



Condition flow: Nested if-else: Question

- What is the output of program for given input i, j, k as follow;

- A. i=3, j=5, k =7
- B. i=-2, j=-5, k=9
- C. i=8, j=15, k=12
- D. i = 25, j=15, k=17

```
if i < j:  
    if j < k:  
        i = j  
    else:  
        j = k  
else:  
    if j > k:  
        j = i  
    else:  
        i = k  
print(i,j,k)
```

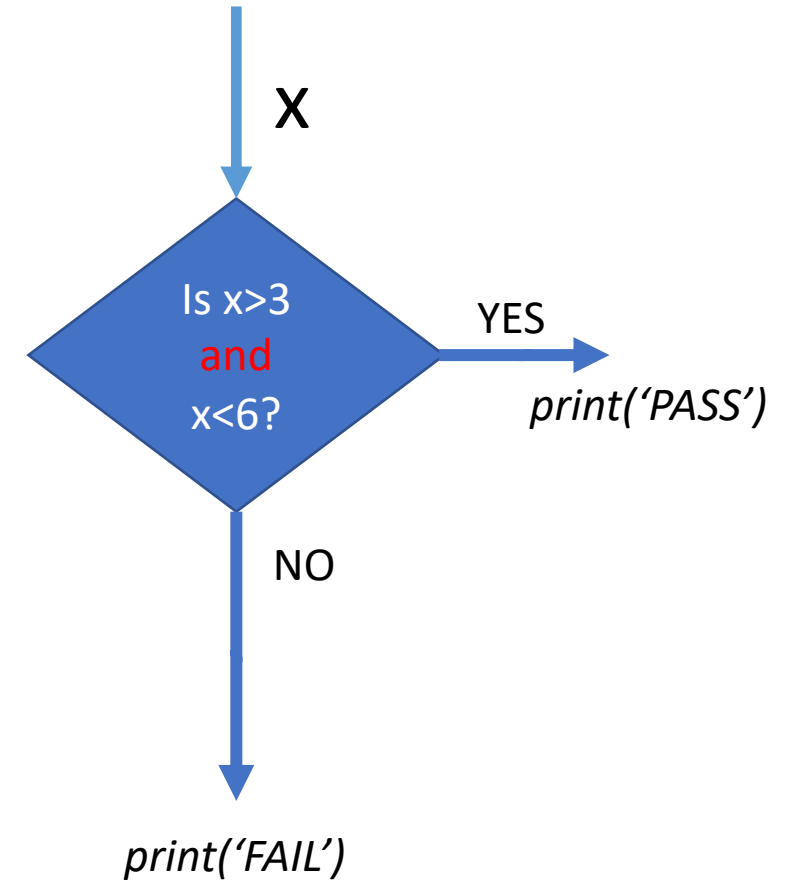
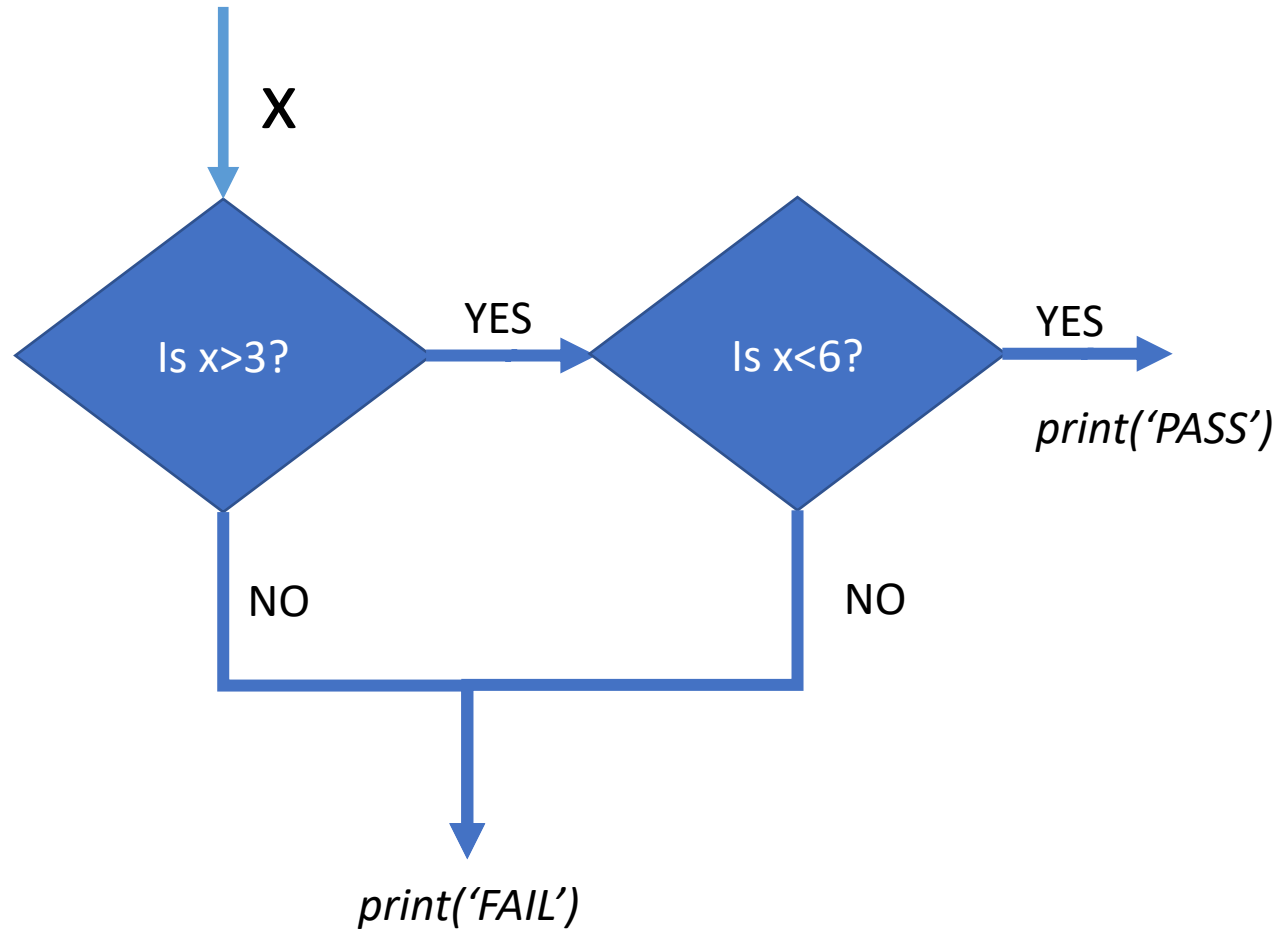
Lecture Outline

Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loop

Combing the conditions

- Test if x is between 3 and 6



Complex conditions with Boolean operators

- A complex condition can be expressed by combining simple conditions
 - $x > 3$ **and** $x < 6$
 - $x == 3$ **or** $x == 6$
 - x **not** equal to 3 **or** 6
- Python allows to combine conditions using keywords such as **and**, **or**, and **not** (also called as Boolean Operators)
 - **and** needs both conditions to be True
 - **or** needs at least one of both to be True
 - **not** reverses the True to False and False to True

Complex conditions with Boolean operators

- Truth tables for **and**, **or** and **not** operators are as follow;

and

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

or

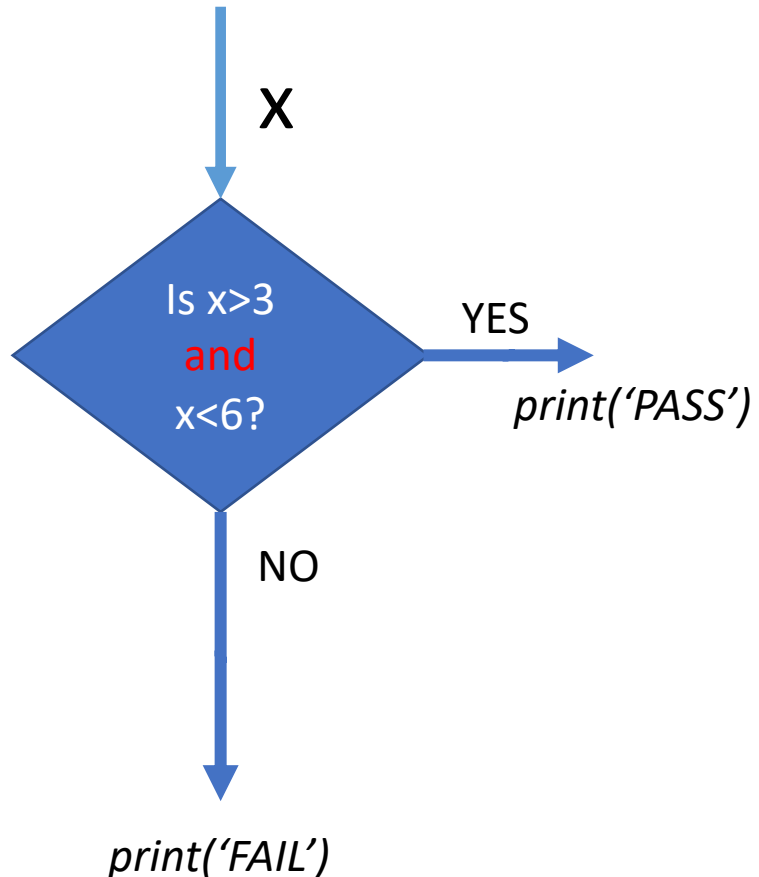
x	y	x or y
False	False	False
False	True	True
True	False	True
True	True	True

not

x	not x
False	True
True	False

Complex conditions

- Test if x is between 3 and 6

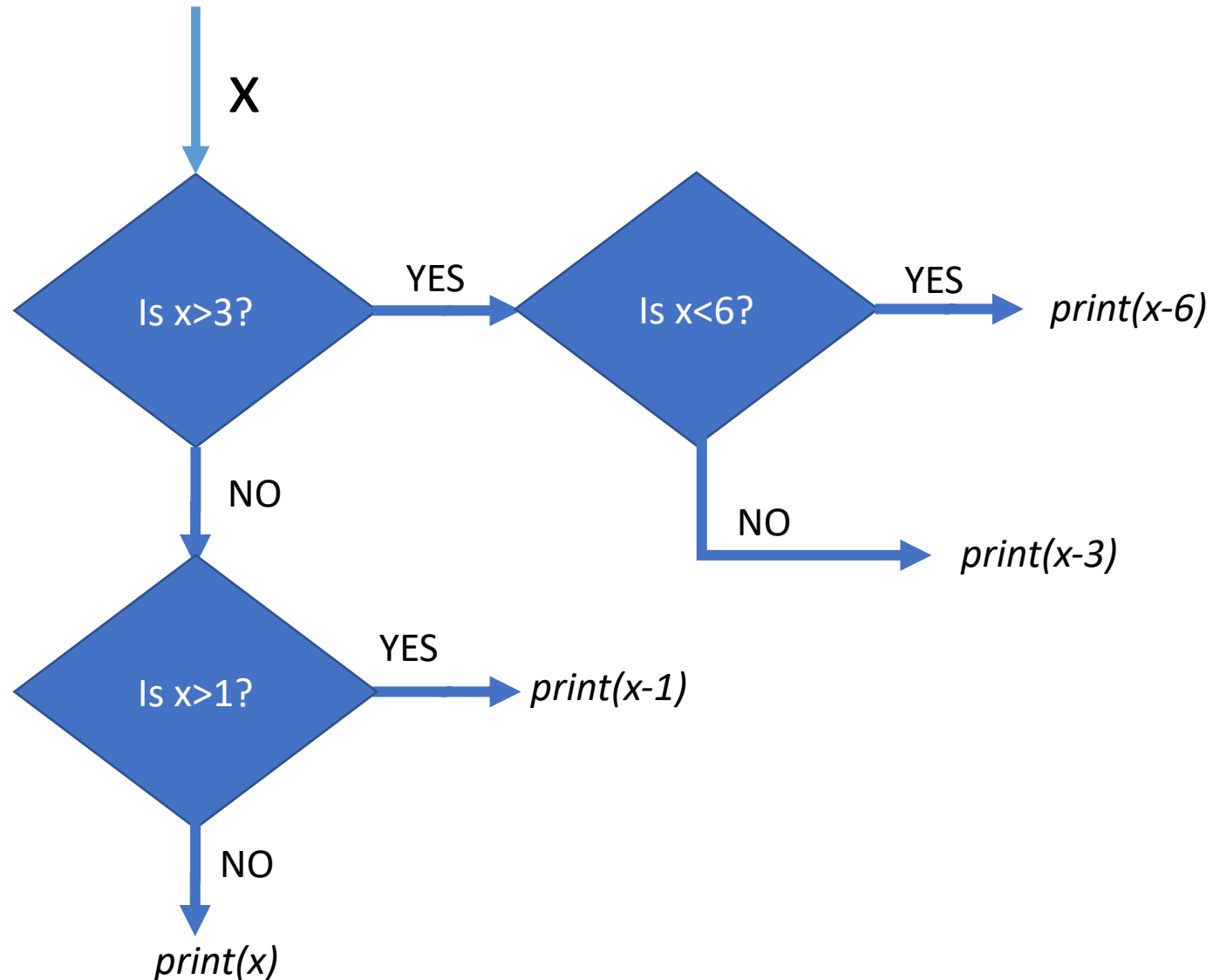


```
if (x>3) and (x<6):  
    print('PASS')  
else:  
    print('FAIL')
```

It is always good practice to use parenthesis () with operators

Complex conditions

- Convert following with complex conditions



Complex conditions

- Example

```
if (x>3) and (x<6):
```

```
    ... do 1...
```

```
elif (x>3) and not (x<6):
```

```
    ... do 2...
```

```
elif x>1:
```

```
    ... do 3...
```

```
else:
```

```
    ... do 4 ...
```

Complex conditions

What is the output of following

- $(i > j)$ and $(i - j > -1)$ or $(i == 0)$

A. $i=0, j=3$

B. $i=1, j=10$

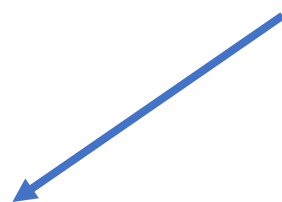
- $((i == j)$ or $(i - j > 1)$ or $(j - i > 1))$ and $(j - 5 < i - 3)$ and not $(i == 0)$

A. For $i=0, j=3$

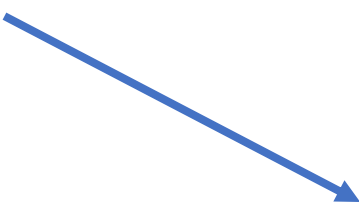
B. $i=10, j=10$

Complex conditions

- Boolean operators can also be represented as `&` and `|`
- `and` `&`
- `or` `|`



```
if (x>3) & (x<6):  
    print('PASS')  
else:  
    print('FAIL')
```



```
if (x>3) | (x<6):  
    print('PASS')  
else:  
    print('FAIL')
```

NOTE: These symbolic operators have other functionalities.

Lecture Outline

Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loop

Condition with None

- Variables can be tested if they are `None` or not as follow;

```
if x==None:  
    print('PASS')  
else:  
    print('FAIL')
```

```
if x is None:  
    print('PASS')  
else:  
    print('FAIL')
```


Lecture Outline

Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loop

Loops: for-loop

we have seen it in previous lectures

- **For-loop:** In programming languages, for repeating operation(s), a loop is used, which iterate over a sequence (goes over each element of a sequence)

```
fruits = ["apple", "banana", "cherry"]
```

```
for fruit in fruits:
```

```
    print(fruit)
```

```
numbers = [1,2,3,4,5]
```

```
squares = []
```

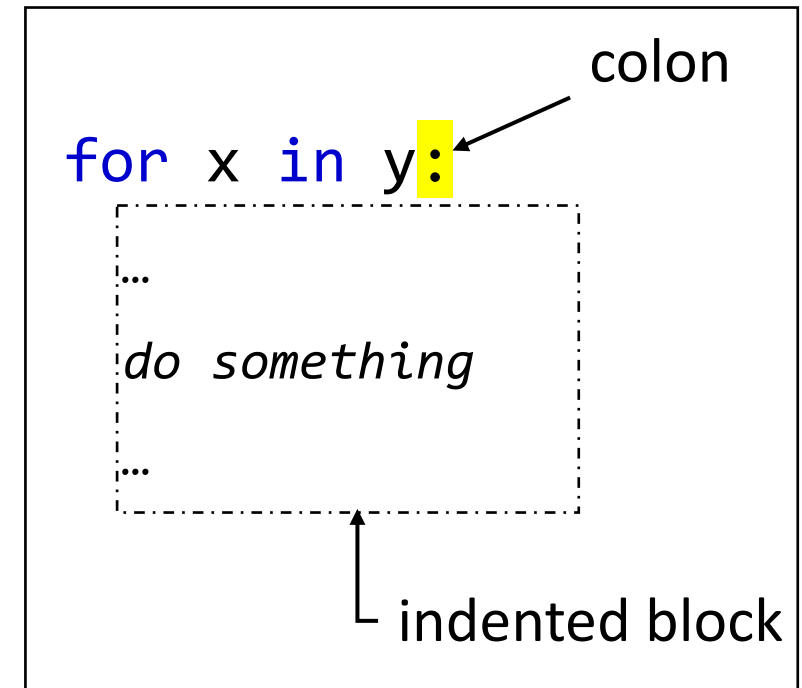
```
for num in numbers:
```

```
    squares.append(num ** 2)
```

```
for i in range(6):
```

```
    print(i)
```

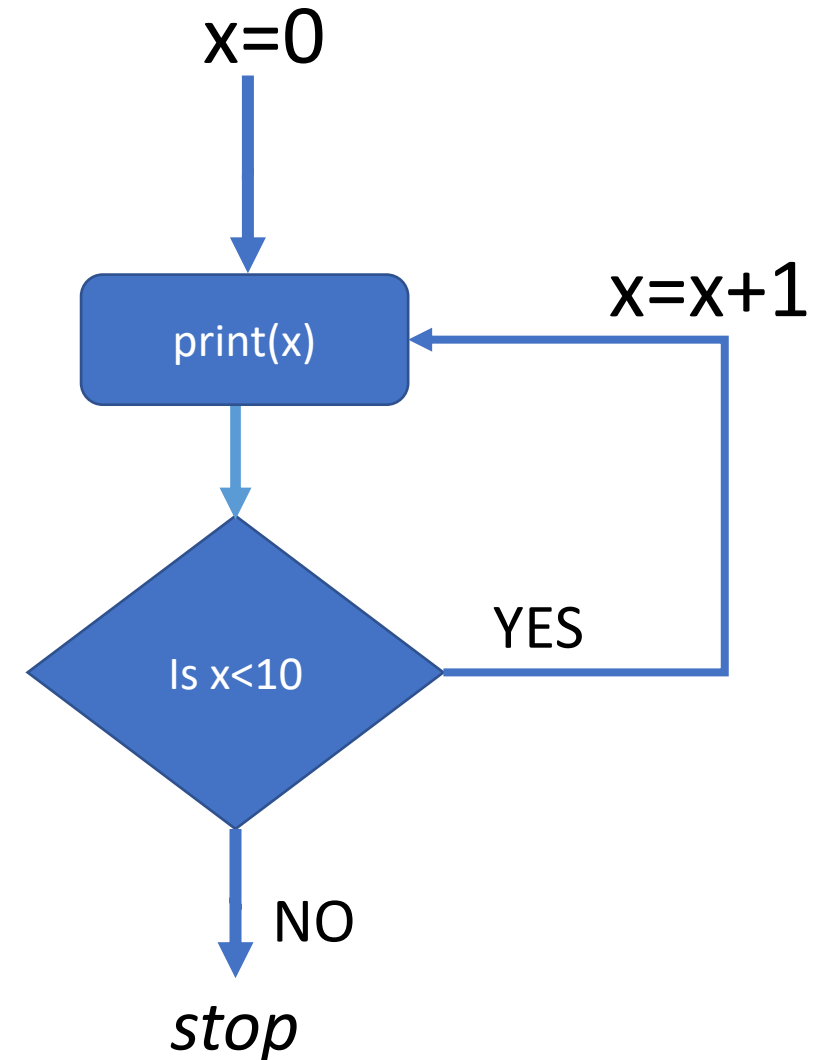
```
print(squares)
```



Loops: for-loop

- **For-loop:** It can also be seen as a flow-diagram containing a **loop**

```
for x in range(10):  
    print(x)
```



Loops: while-loop

- **while-loop:** In programming languages, for repeating operation(s), *as long as a condition is satisfied*, can be done by using while-loop.

```
x = 0
```

```
while x < 10:
```

```
    print(x)
```

```
    x = x + 1
```

```
Cond = TRUE
```

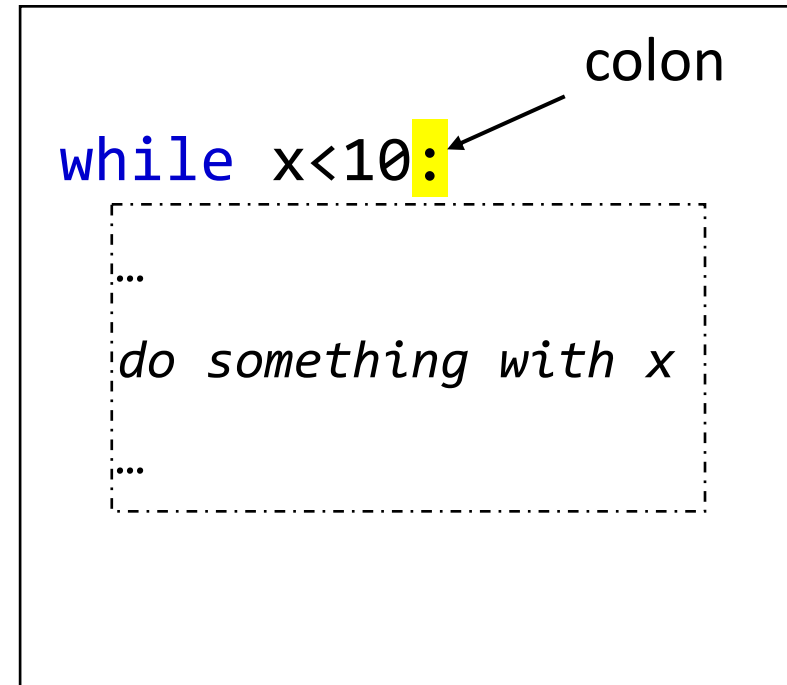
```
while Cond:
```

```
    X = np.random.rand()
```

```
    if X > 0.5:
```

```
        Cond = FALSE
```

```
    print(X)
```



Loops: while-loop

- **while-loop:** while-loop is used carefully as it can go on forever (*infinite-loop*) if condition (statement) is always **True**.

```
x = 1
while x>0:
    print(x)
    x = x+1
```

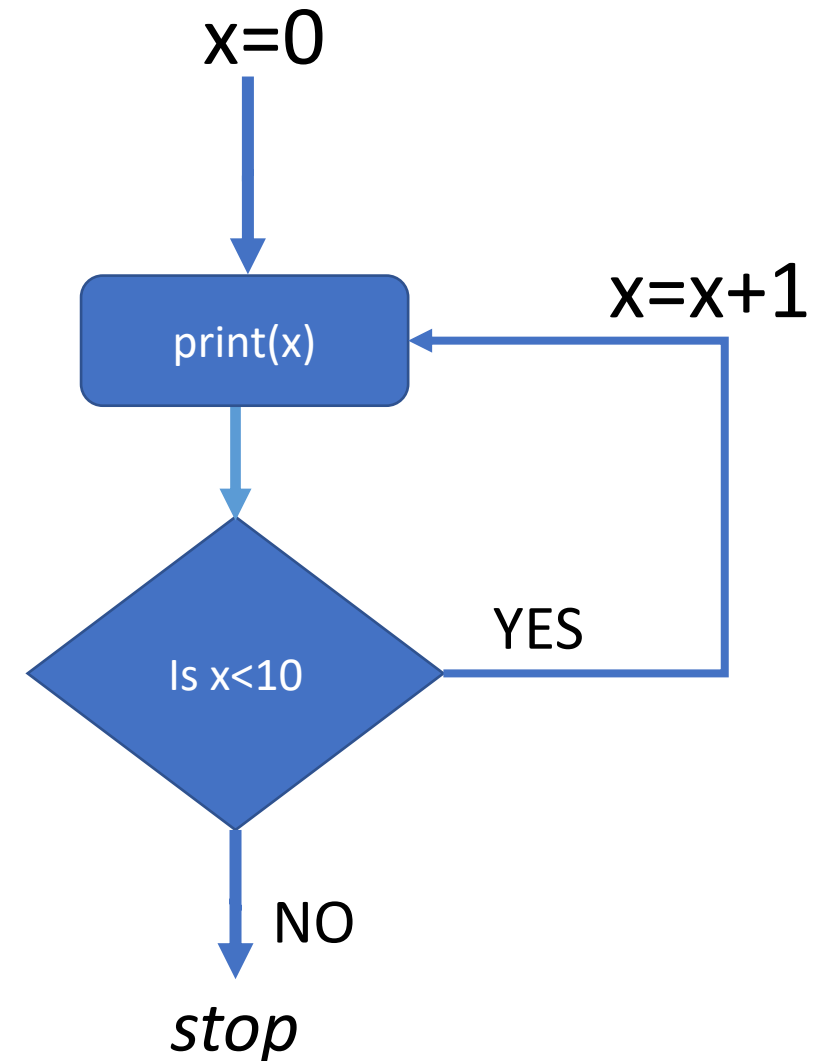
```
while True:
    print('Hello')
```

```
Cond = TRUE
while Cond:
    X = np.random.rand()
    if X>2:
        Cond = FALSE
    print(X)
```

Loops: while-loop

- **while-loop:** It can also be seen as a flow-diagram containing a **loop**

```
x = 0  
while x < 10:  
    print(x)  
    x = x + 1
```



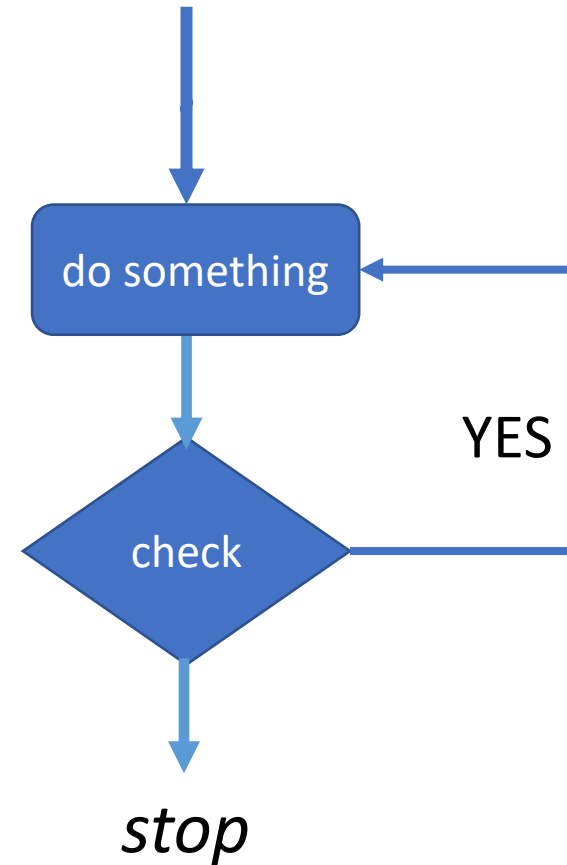
Lecture Outline

Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loop

Loops: Interruptions

- **for-loop and while-loop** can be interrupted to change the behaviour by using following keywords.
 - `break`
 - `continue`
 - `pass`*



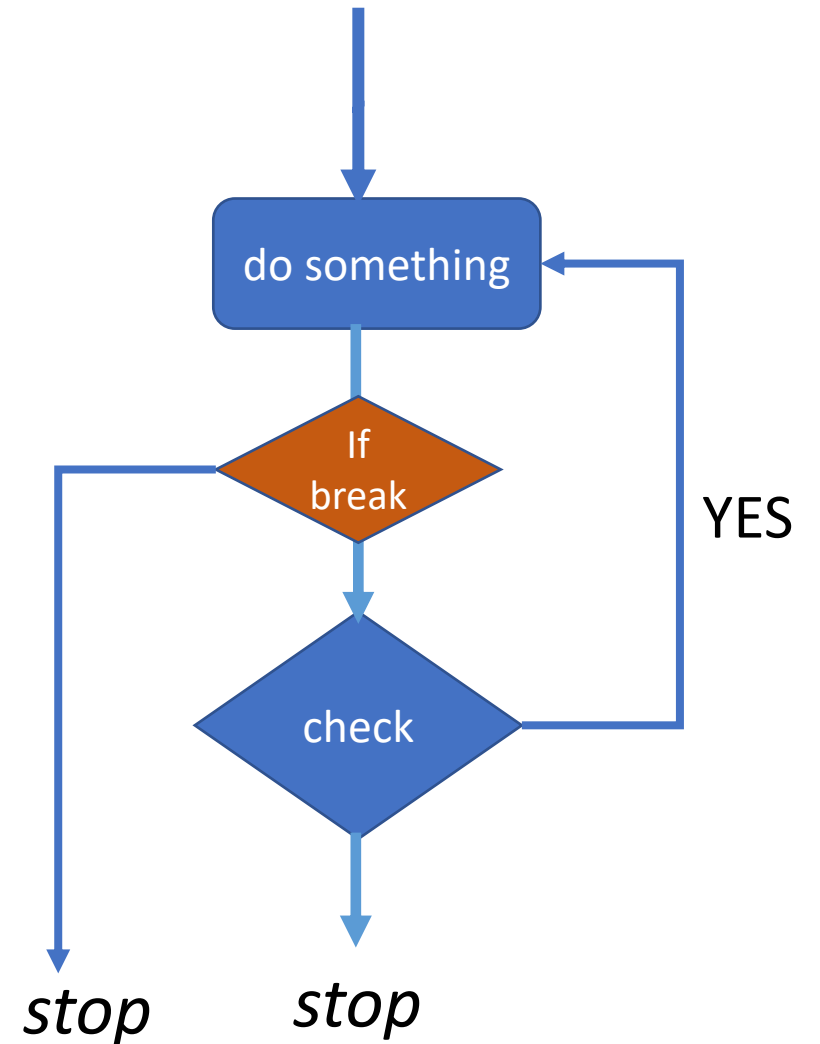
*`pass` does not do anything

Loops: Interruptions

break

- If during following the loop, program sees 'break' it completely stops the loop
- To stop the loop, *break out* of the loop, *exit* the loop

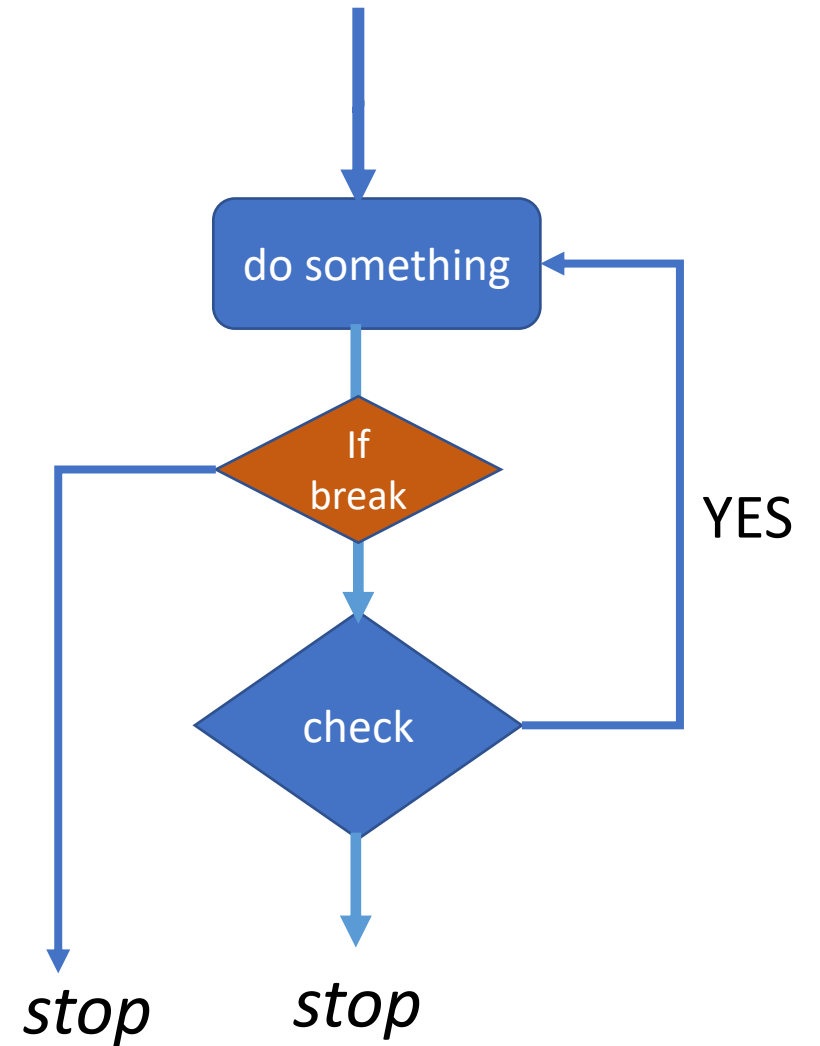
```
for x in range(100):  
    print('Hello')  
    if x>10:  
        break
```



Loops: Interruptions

break

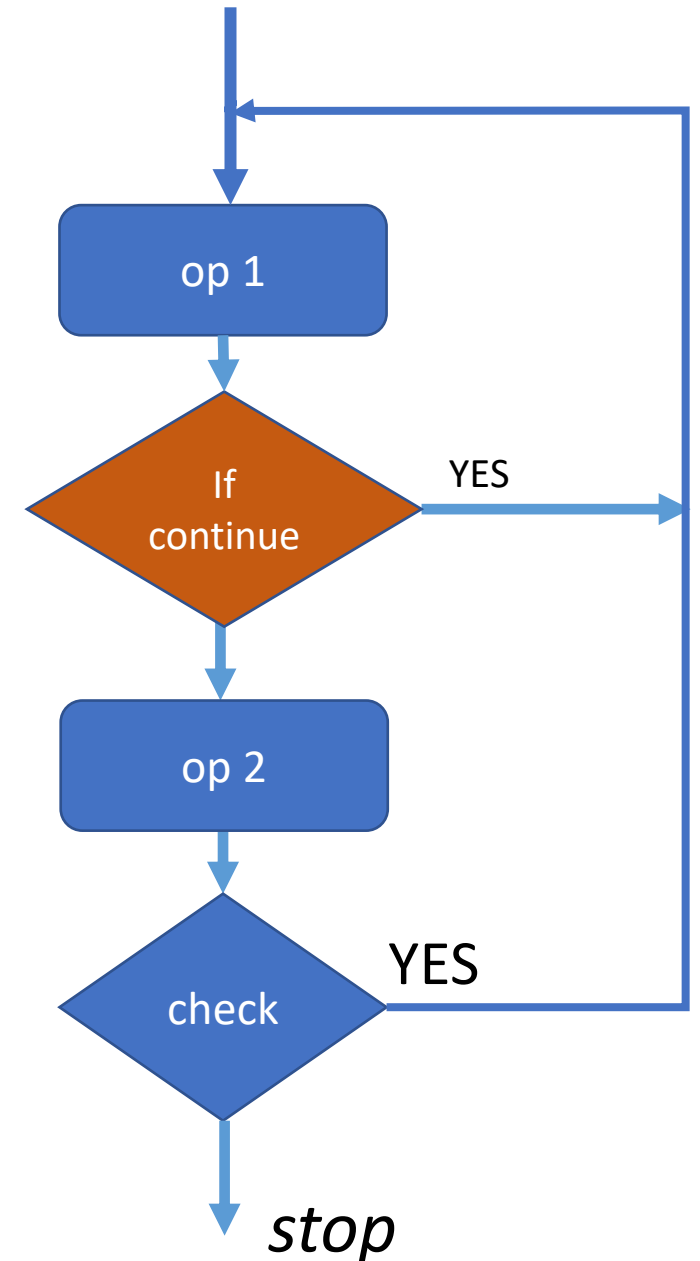
```
Alpha = ['D', 'B', 'A', 'C']  
i = 0  
while True:  
    c = Alpha[i]  
    print(c)  
    i = i+1  
    if c=='A':  
        break
```



Loops: Interruptions

continue

- If during following the loop, program sees 'continue' it skip everything after that to go to next iteration
- To skip the part of iteration
- In figure, op 2 will be skipped if program sees `continue`



Loops: Interruptions

continue

- example

```
x = [3, 10, None, 10, 50]
```

```
c = 0
```

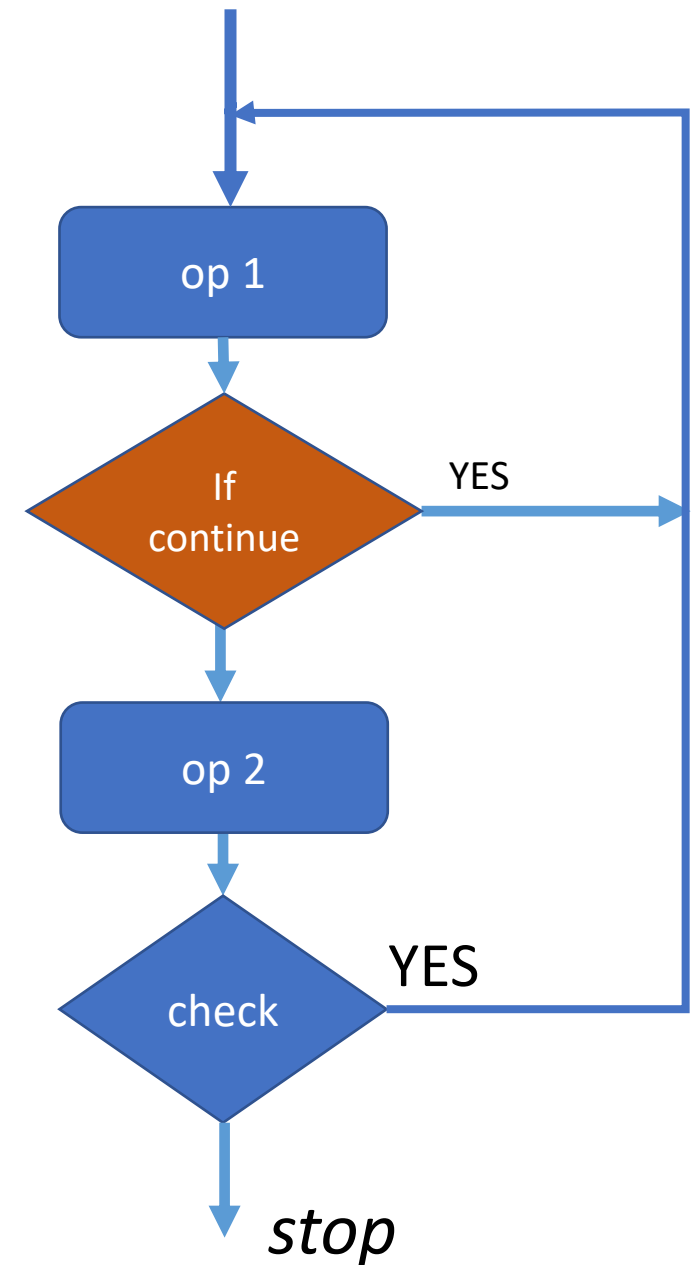
```
for a in x:
```

```
    print(a)
```

```
    if a is None:
```

```
        continue
```

```
    c = c + a
```



Loops: Interruptions

pass

- pass does nothing. It is used as a dummy operation

```
for x in range(100):  
    print('Hello')  
    if x>10:  
        pass
```

Lecture Outline

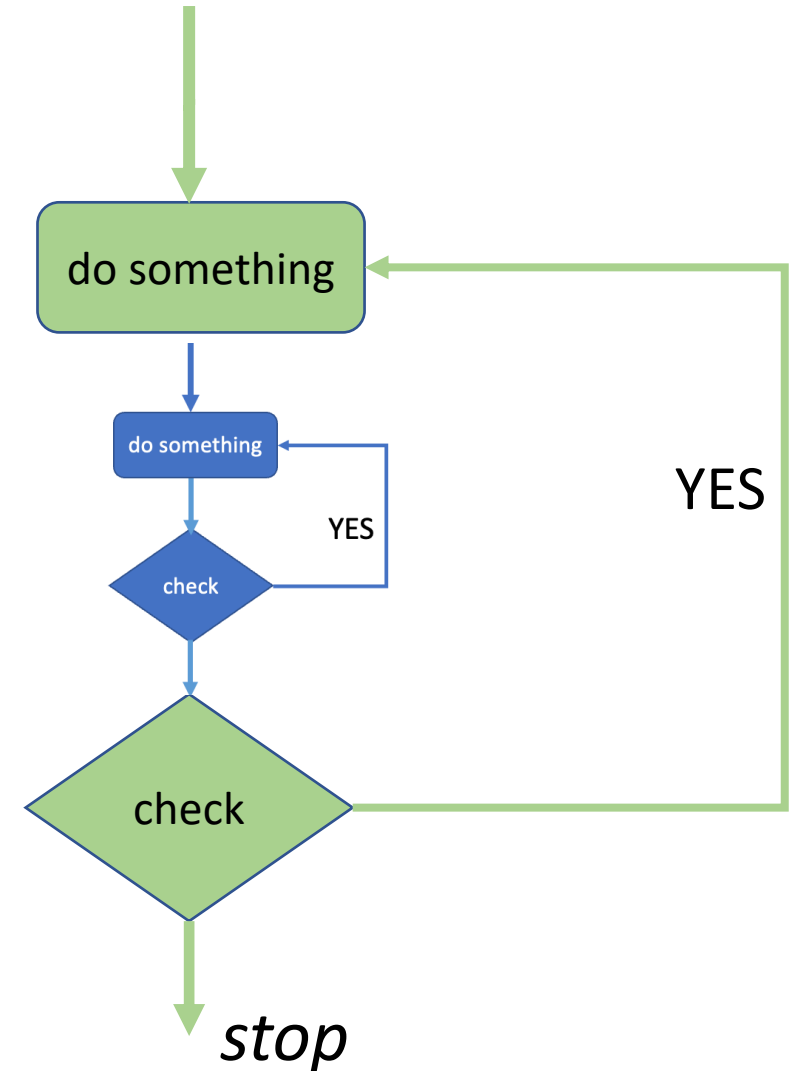
Control flow tools

- Condition operators
- Condition flow (if-else)
- Complex conditions with Boolean operators
- Condition with None
- Loops (for-loop, while-loop)
- Interruptions to loops
- Nested loops

Loops: nested loops

- **for-loop and while-loop** can have nested loops
- A loop inside a loop

```
XY = []  
for x in range(10):  
    for y in range(10):  
        z = x**2 + y**2  
        XY.append(z)
```



- Next !!!
 - 3.2: More on Control flow
Defining Functions



Queen Mary

University of London