香 港 中 文 大 學（深 圳）
The Chinese University of Hong Kong, Shenzhen

Introduction to Computer Science:
Programming Methodology

Lecture 3 Flow Control

Guiliang Liu
School of Data Science

Conditional flow

Program

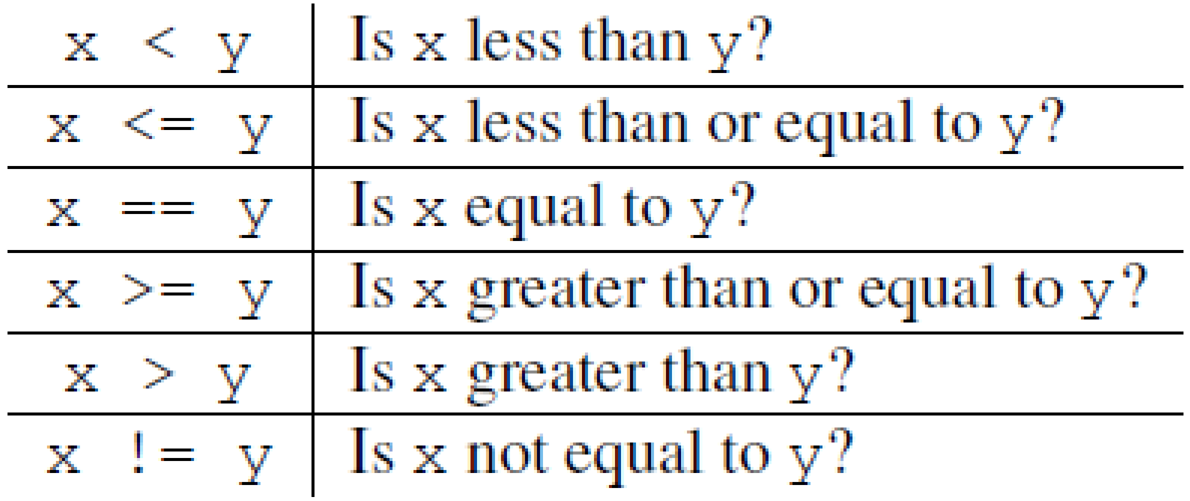
```
x=5
if x<10:
    print("smaller")
if x>20:
    print("bigger")
print("finished")
```

Outputs

```
smaller
finished
>>> |
```

Comparison operators

- Boolean expressions ask a question and produce a Yes/No result, which we use to control program flow
- Boolean expressions use comparison operators to evaluate Yes/No or True/False
- Comparison operators check variables but do not change the values of variables
- Careful!! “=“ is used for assignment



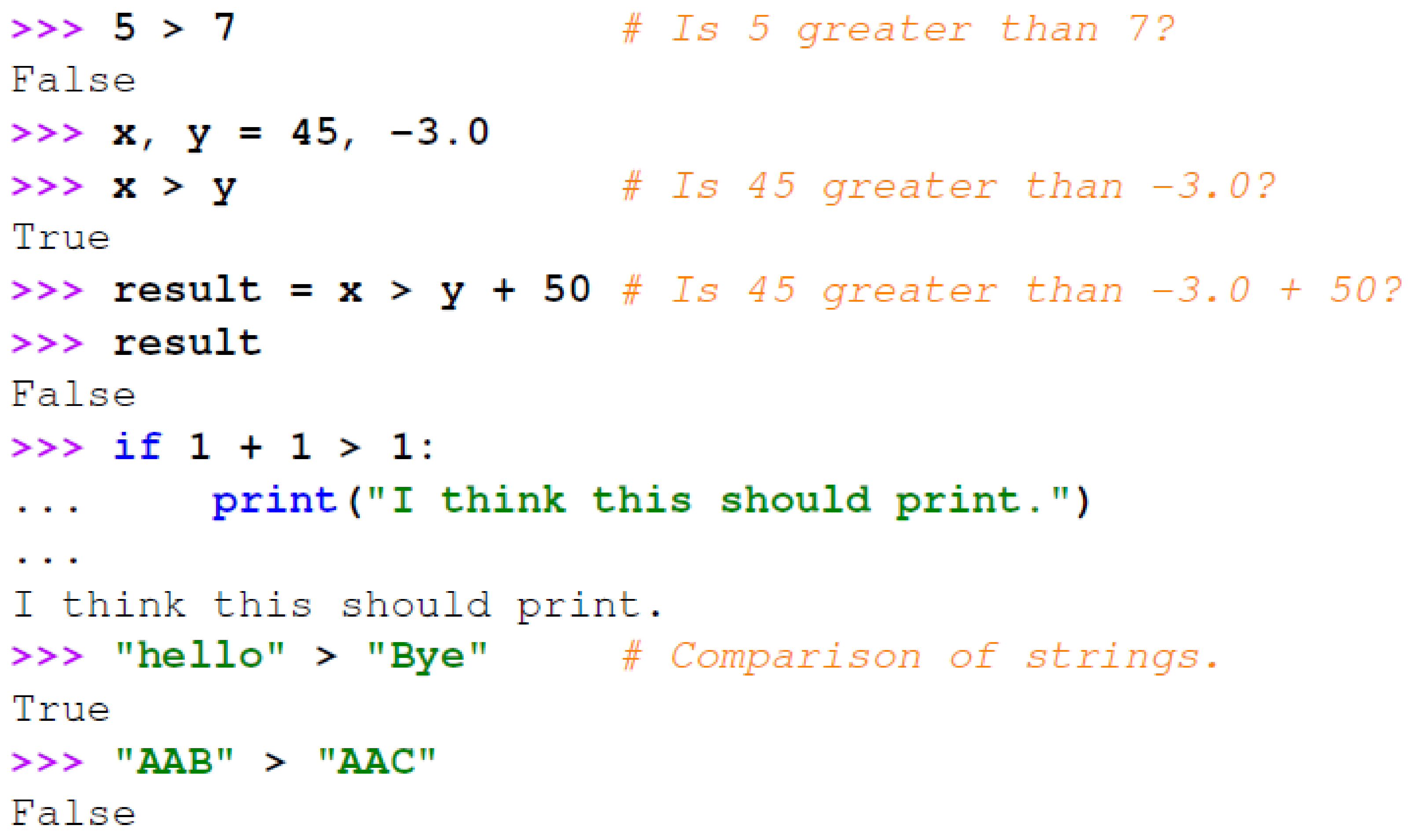
x < y	Is x less than y?
x <= y	Is x less than or equal to y?
x == y	Is x equal to y?
x >= y	Is x greater than or equal to y?
x > y	Is x greater than y?
x != y	Is x not equal to y?

Comparison operators

```
x=5
if x==5:
    print("Equals 5")

if x>4:
    print("Greater than 4")

if x>=5:
    print("Greater than or equal to 5")

if x<=5:
    print("Less than or equal 5")

if x!=6:
    print("Not equal 6")
```

```
Equals 5
Greater than 4
Greater than or equal to 5
Less than or equal 5
Not equal 6
```

Examples of comparison

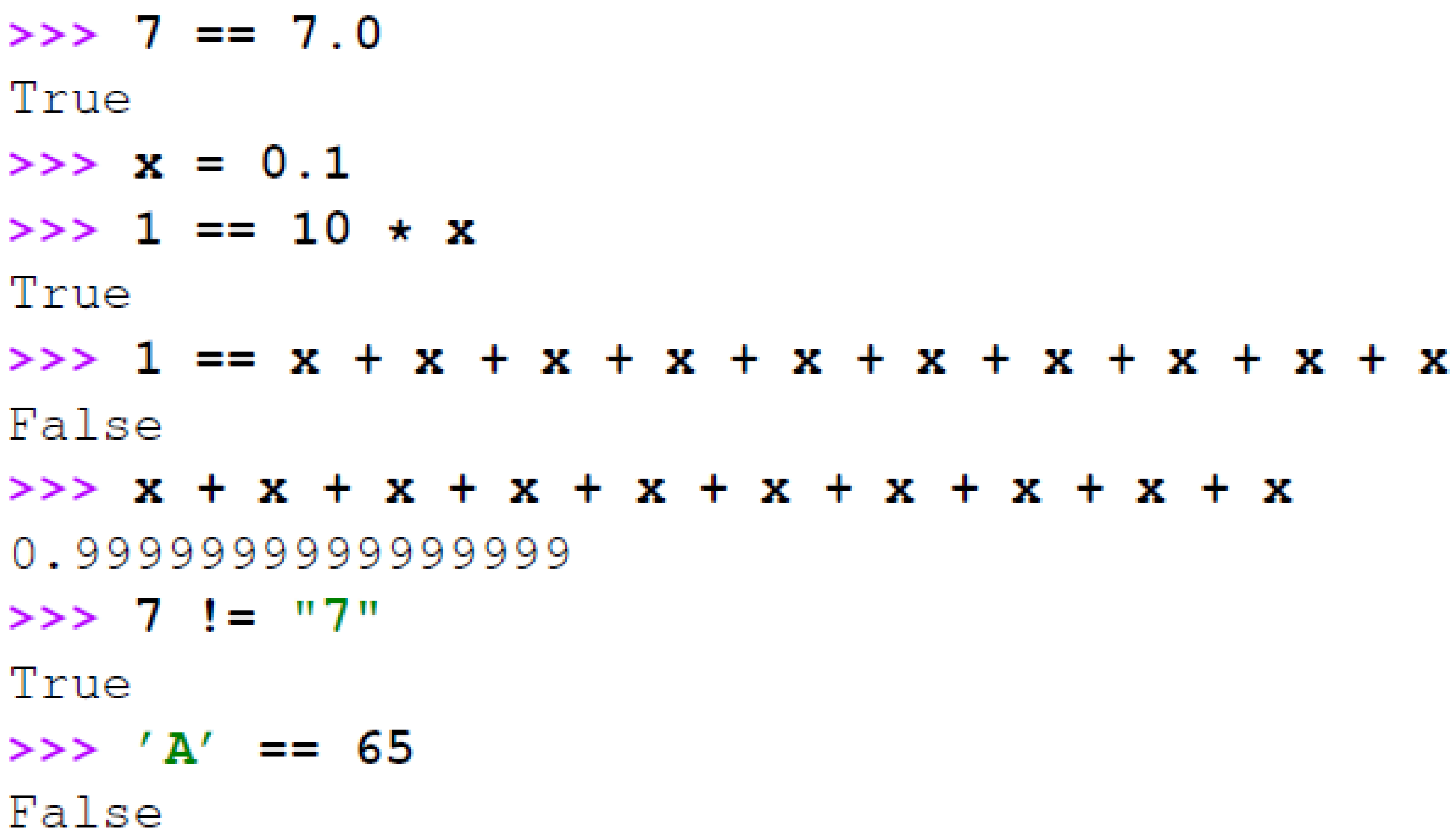


```
>>> 5 > 7                   # Is 5 greater than 7?
False
>>> x, y = 45, -3.0
>>> x > y                   # Is 45 greater than -3.0?
True
>>> result = x > y + 50 # Is 45 greater than -3.0 + 50?
>>> result
False
>>> if 1 + 1 > 1:
...     print("I think this should print.")
...
I think this should print.
>>> "hello" > "Bye"         # Comparison of strings.
True
>>> "AAB" > "AAC"
False
```

Python 3 uses the **lexicographic** (dictionary) order for strings

Capital letters are **always before** lower case letters

Examples of comparison

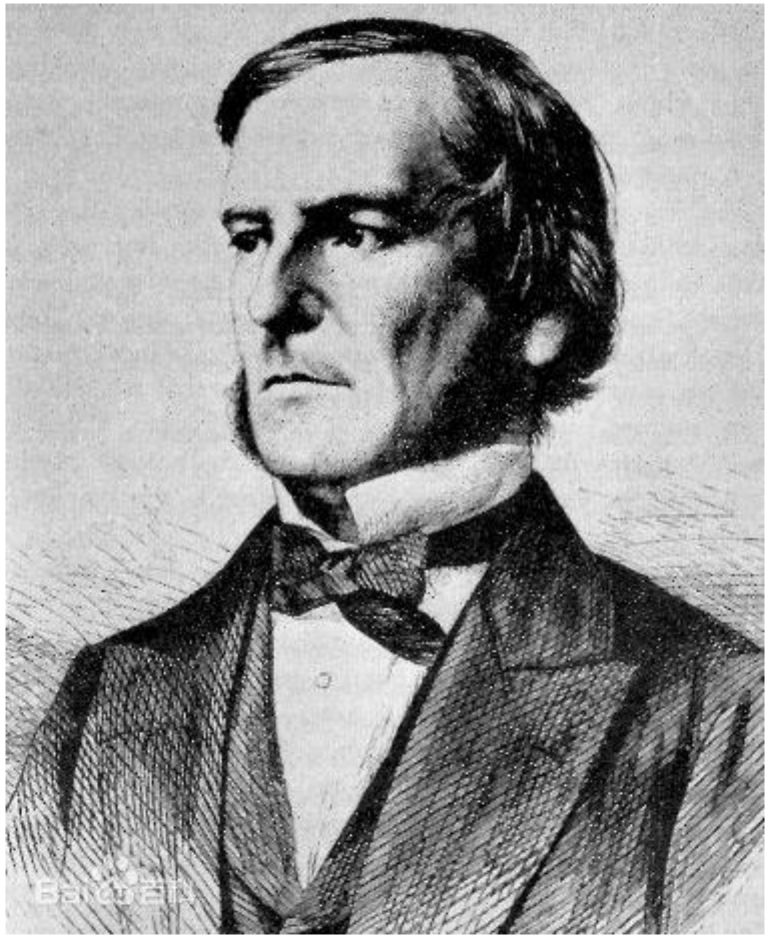


```
>>> 7 == 7.0
True
>>> x = 0.1
>>> 1 == 10 * x
True
>>> 1 == x + x + x + x + x + x + x + x + x + x
False
>>> x + x + x + x + x + x + x + x + x + x
0.9999999999999999
>>> 7 != "7"
True
>>> ’A’ == 65
False
```

Boolean type

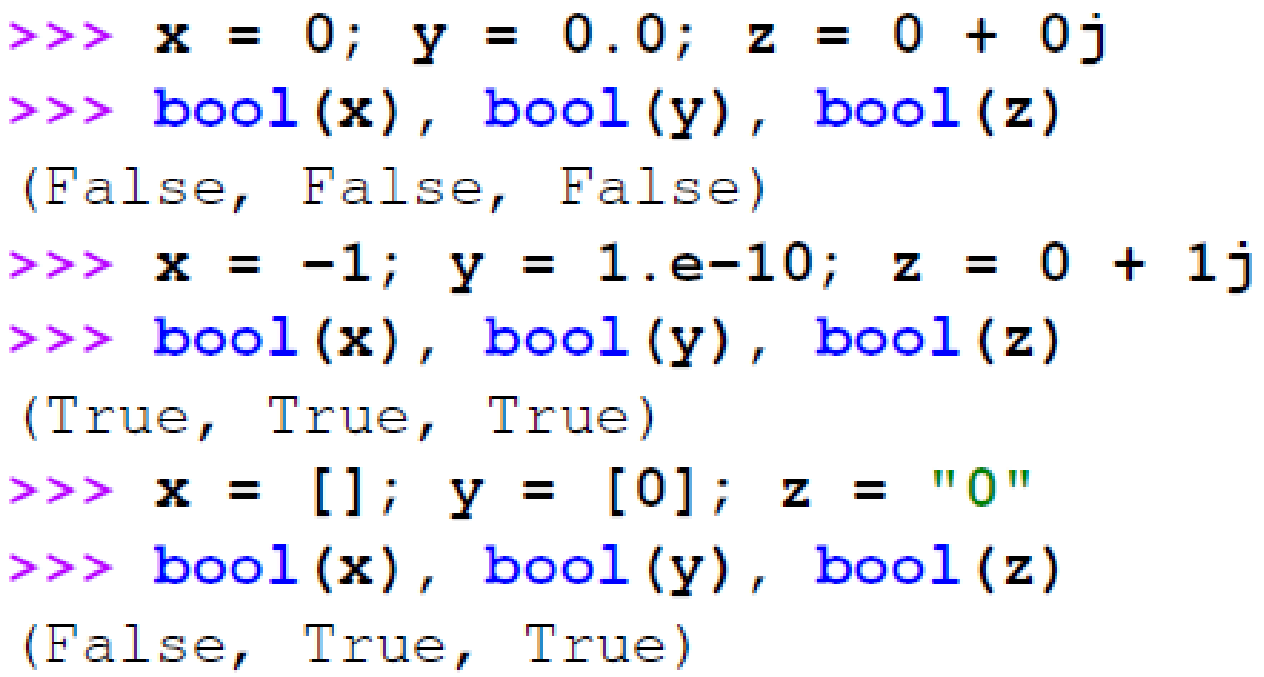
- Python contains a built-in Boolean type, which takes two values True/False

- Number 0 can also be used to represent False. All other numbers represent True



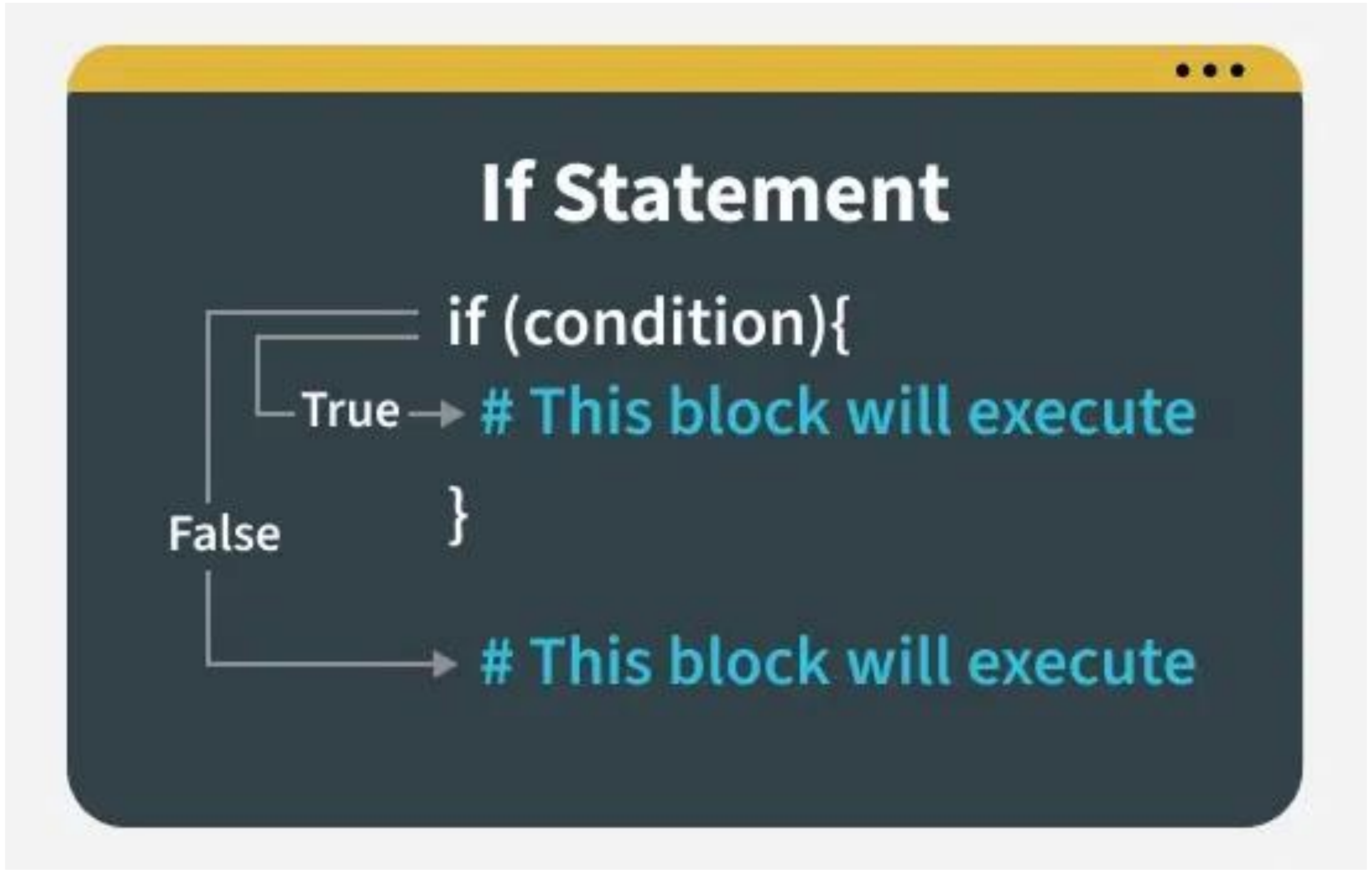
George Boole (1815 - 1864): Mathematician, inventor of mathematical logic, significant contributions to differential and difference equations

Bool()



```
>>> x = 0; y = 0.0; z = 0 + 0j
>>> bool(x), bool(y), bool(z)
(False, False, False)
>>> x = -1; y = 1.e-10; z = 0 + 1j
>>> bool(x), bool(y), bool(z)
(True, True, True)
>>> x = []; y = [0]; z = "0"
>>> bool(x), bool(y), bool(z)
(False, True, True)
```

if statement



Source: Geeksforgeeks

if statement

```
x=5
print('Before 5')
if x==5:
    print('Is 5')
    print('Is still 5')
    print('Third 5')

print('Afterwards 5')

print('Before 6')
if x==6:
    print('Is 6')
    print('Is still 6')
    print('Third 6')

print('Afterwards 6')
```

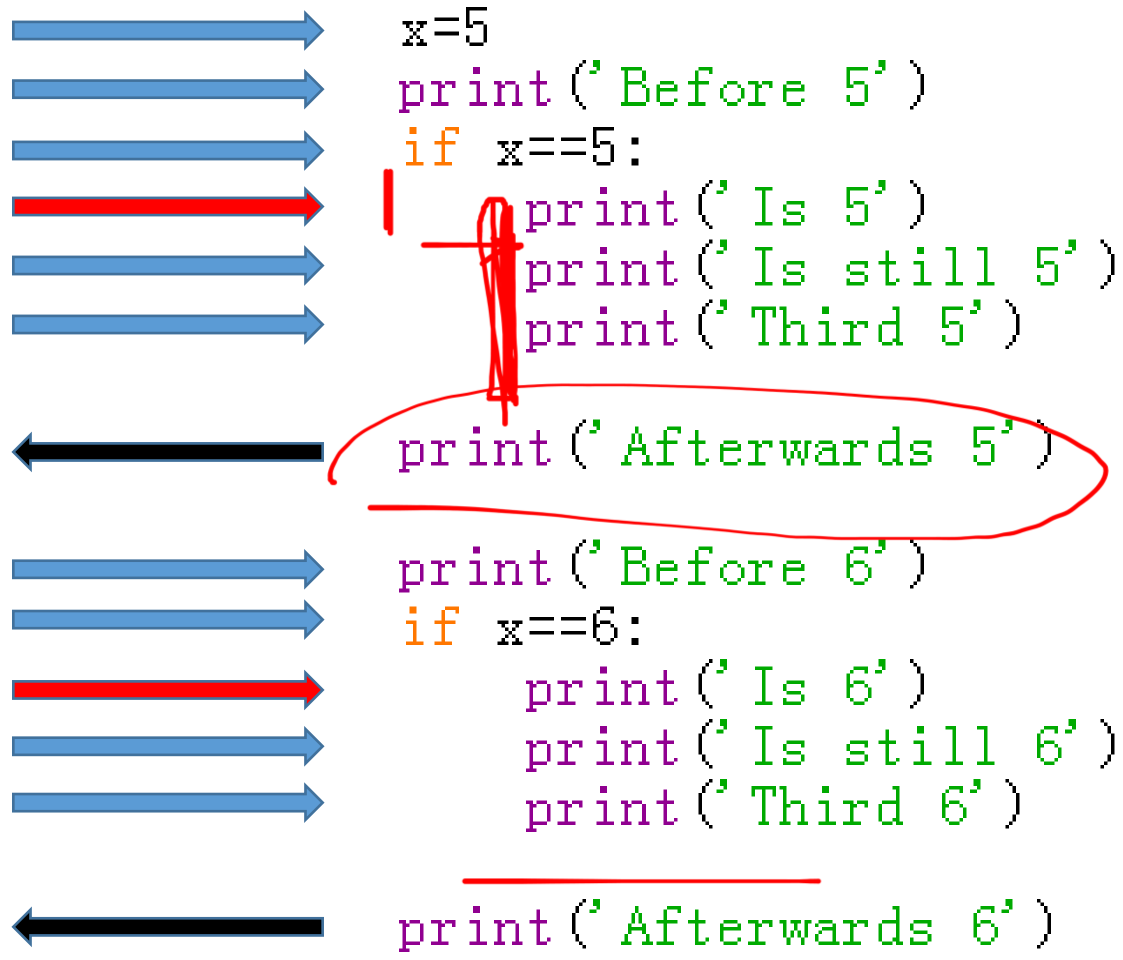
```
Before 5
Is 5
Is still 5
Third 5
Afterwards 5
Before 6
Afterwards 6
```

Indentation

- Increase indent: indent after an if or for statement (after :)
- Maintain indent: to indicate the scope of the block (which lines are affected by the if/for)
- Decrease indent: to back to the level of the if statement or for statement to indicate the end of the block
- Blank lines are ignored – they do not affect indentation
- Comments on a line by themselves are ignored w.r.t. indentation

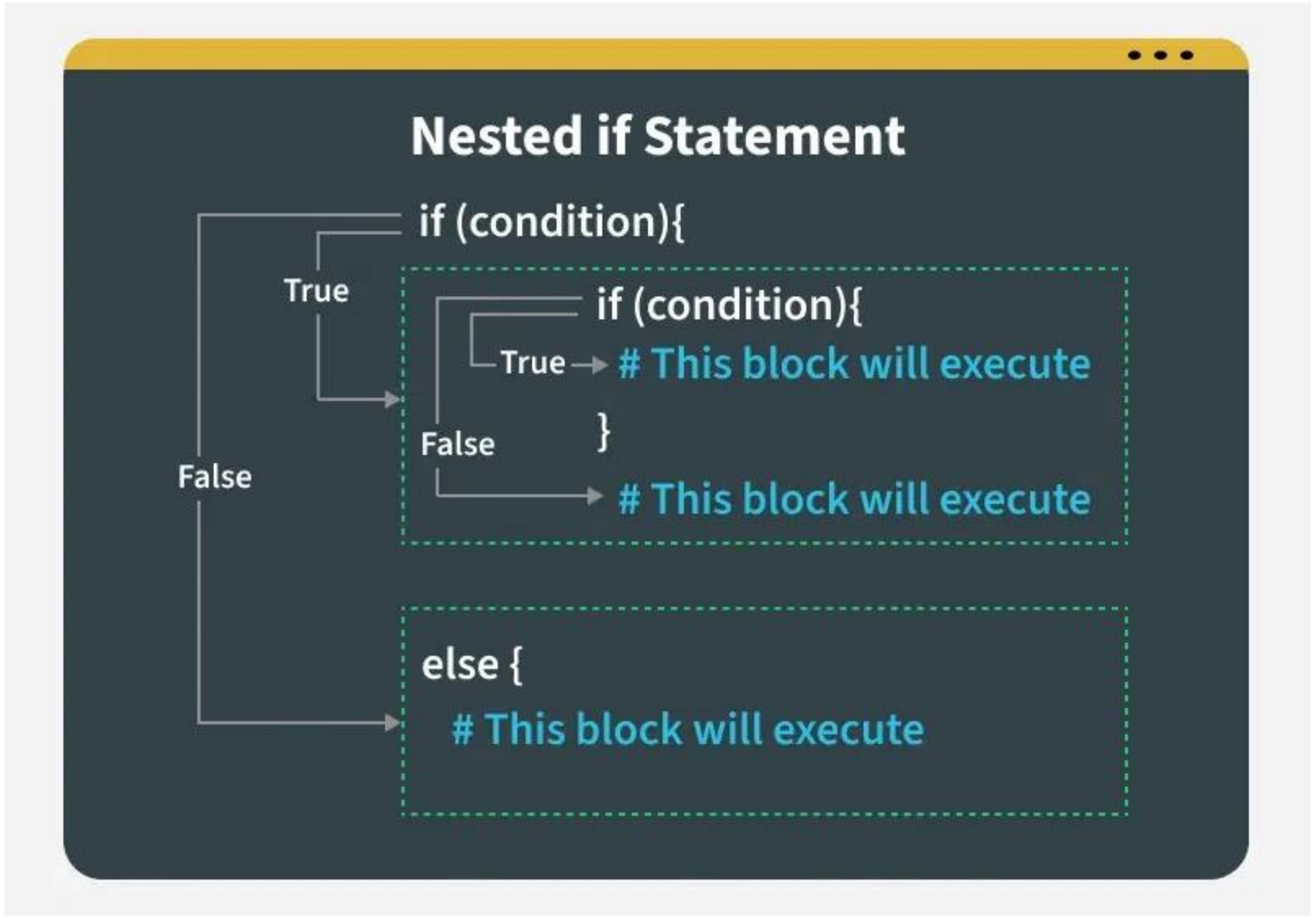
Increase/maintain/decrease

- Increase/maintain after if/for statements
- Decrease to indicate the end of a block
- Blank lines and comments are ignored



```
x=5
print(’Before 5’)
if x==5:
    print(’Is 5’)
    print(’Is still 5’)
    print(’Third 5’)

print(’Afterwards 5’)

print(’Before 6’)
if x==6:
    print(’Is 6’)
    print(’Is still 6’)
    print(’Third 6’)

print(’Afterwards 6’)
```

Nested if statement


Nested if Statement
if (condition){
True
if (condition){
True
# This block will execute
}
False
# This block will execute
False
else {
# This block will execute

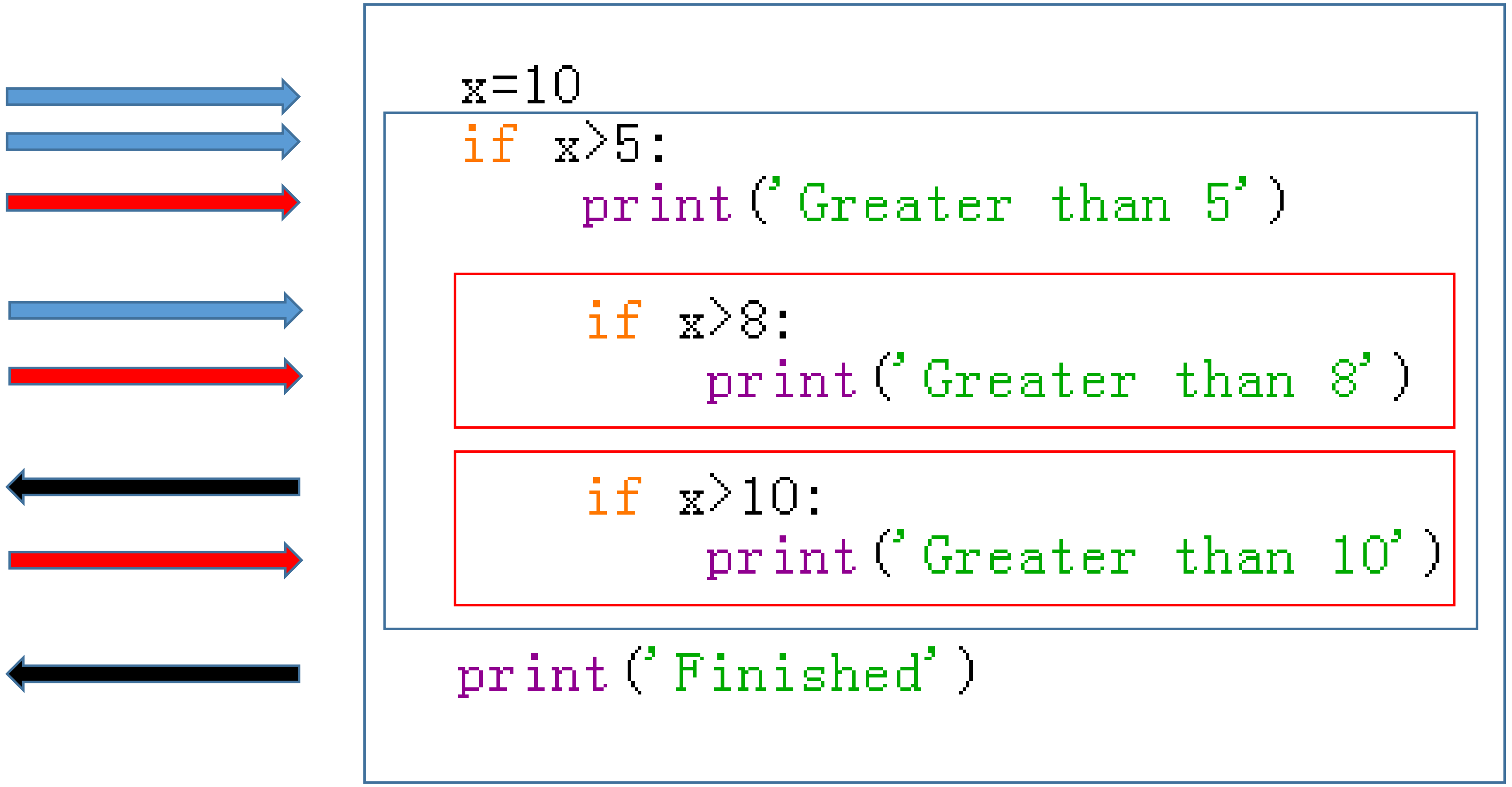
Source: Geeksforgeeks

Nested decisions

Example

```
x=42
if x>1:
    print('More than 1')

    if x<100:
        print('Less then 100')

print('Finished')
```

Mental begin/end



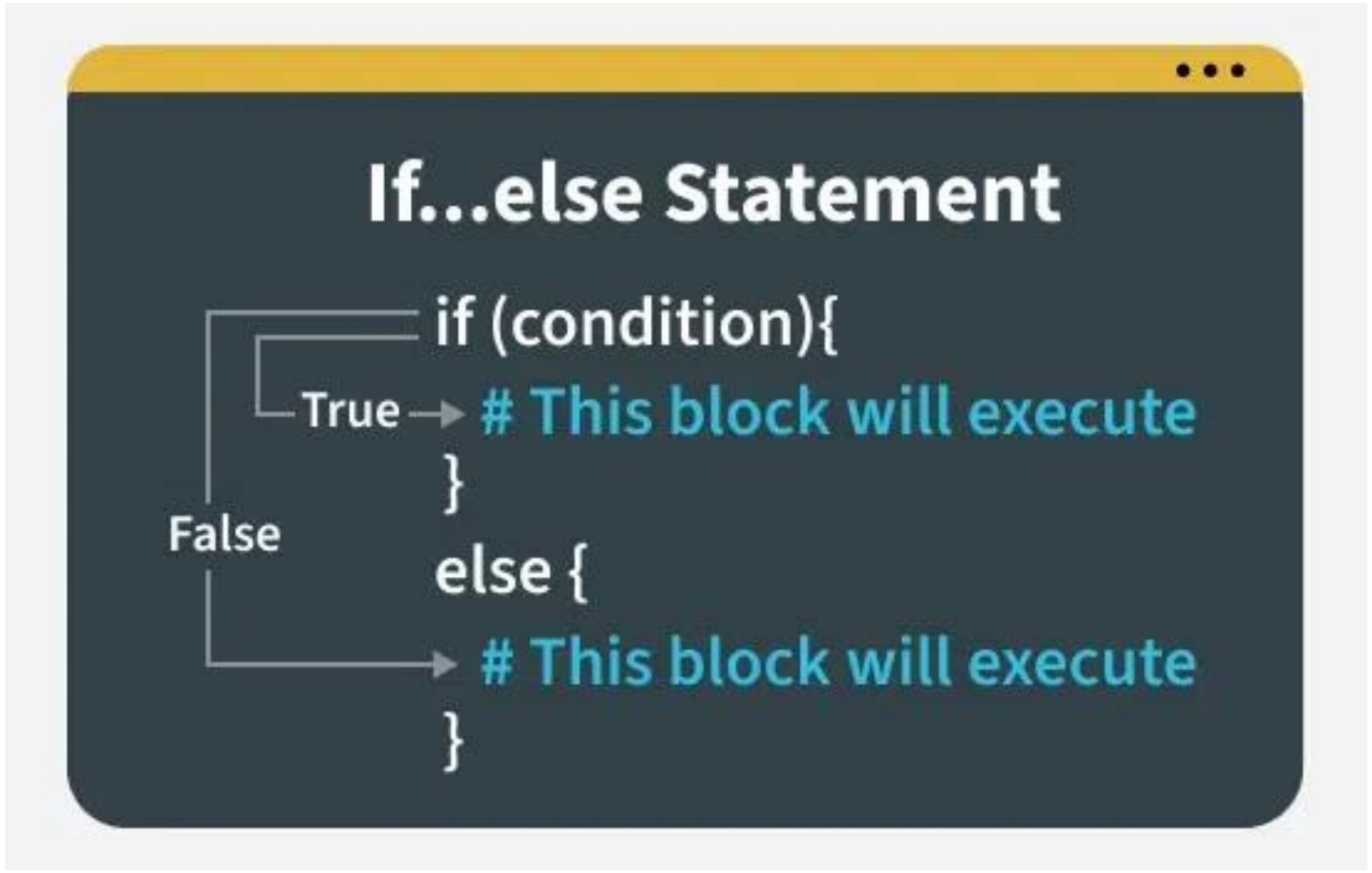
```
x=10
if x>5:
    print('Greater than 5')

    if x>8:
        print('Greater than 8')

    if x>10:
        print('Greater than 10')

print('Finished')
```

Exercise

Write a simple script for a teacher. The script will check a student's grade from a variable and print whether they passed or failed.

The passing grade is **60** or higher.

if....else statement

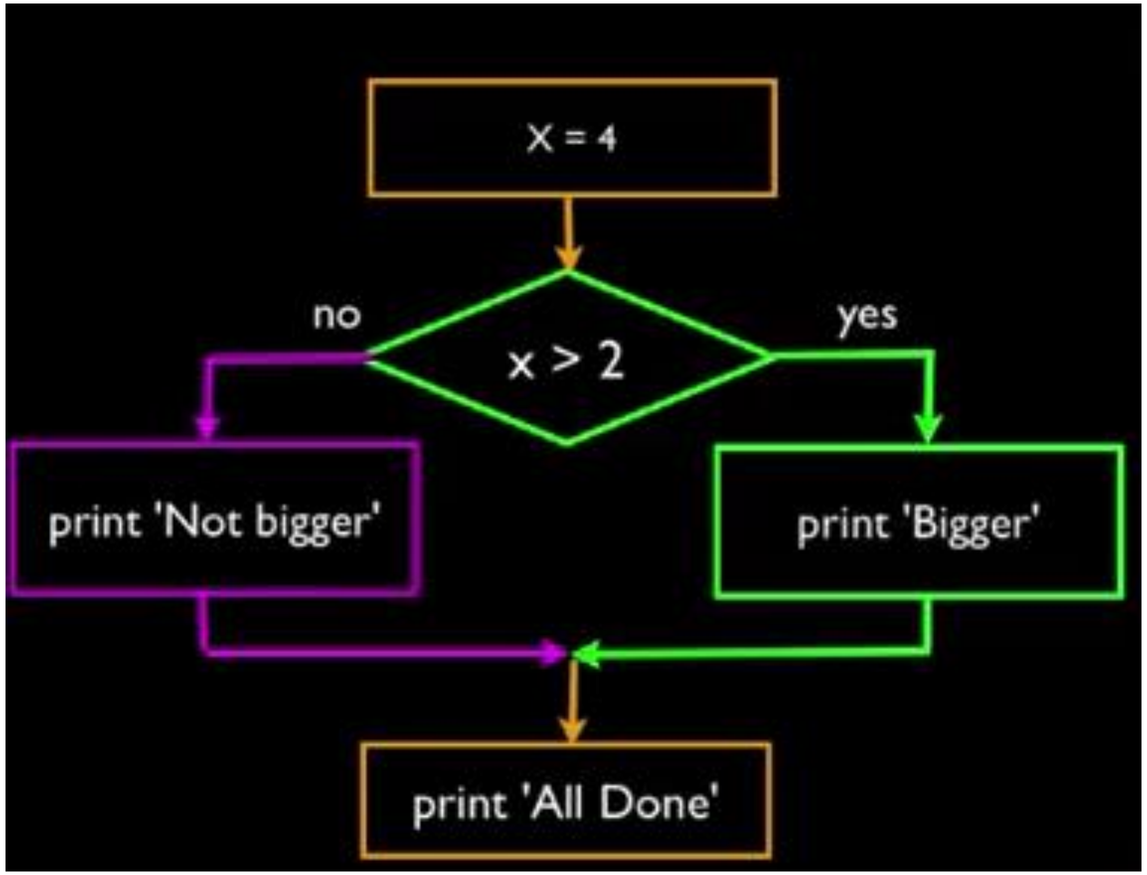


```
if (condition){
 # This block will execute
}
else {
 # This block will execute
}
```

Source: Geeksforgeeks

if....else statement

```
x=1

if x>2:
    print('Bigger')
else:
    print('Smaller')

print('Finished')
```



Tips on if - else

```
x=1

if x>2:
    print(‘Bigger’)
else:
    print(‘Smaller’)

print(‘Finished’)
```

✓

```
x=1

if x>2:
    print(‘Bigger’)
    else:
    print(‘Smaller’)

print(‘Finished’)
```

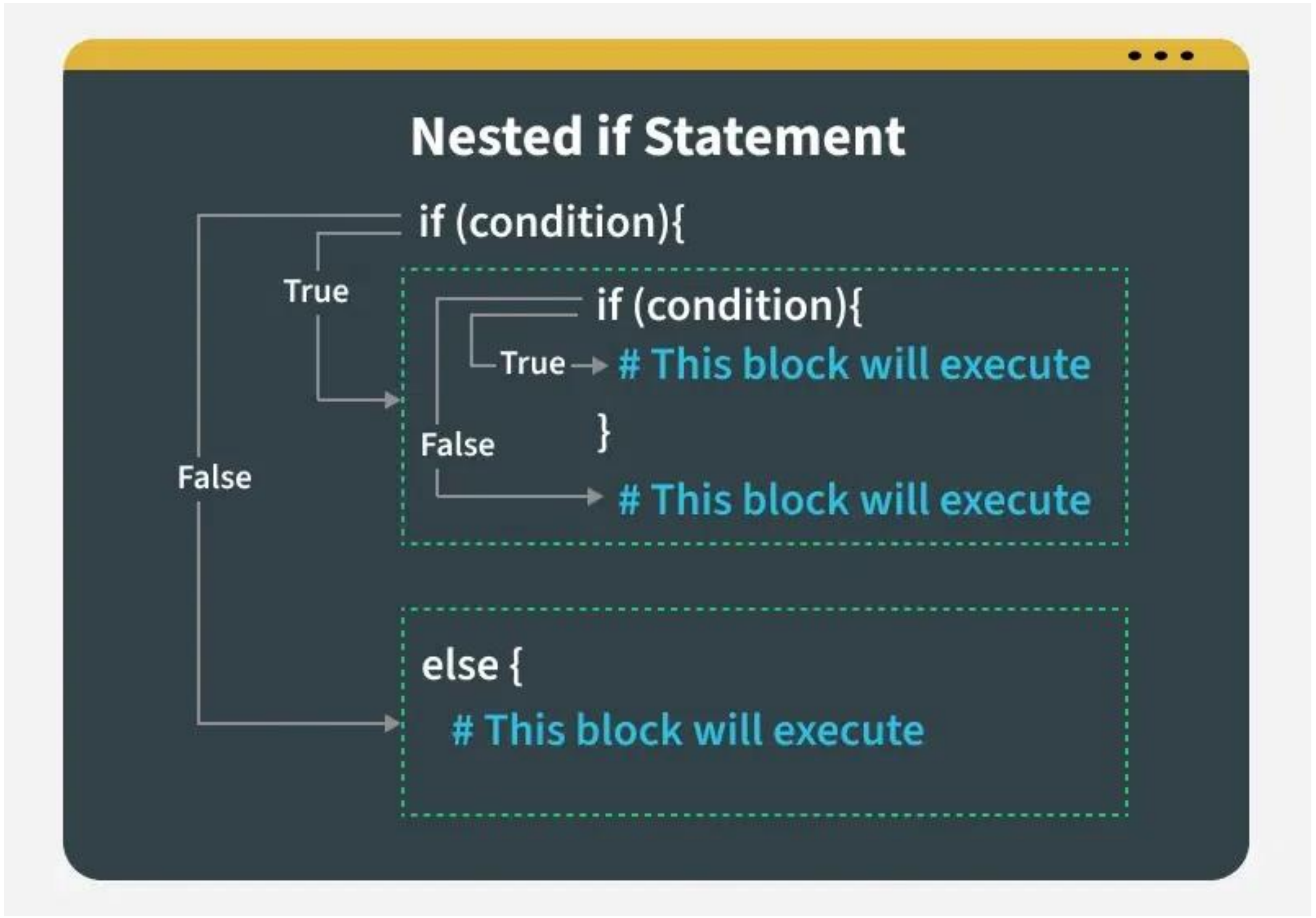
✗

- else must come after if
- Use indentation to match if and else

Example

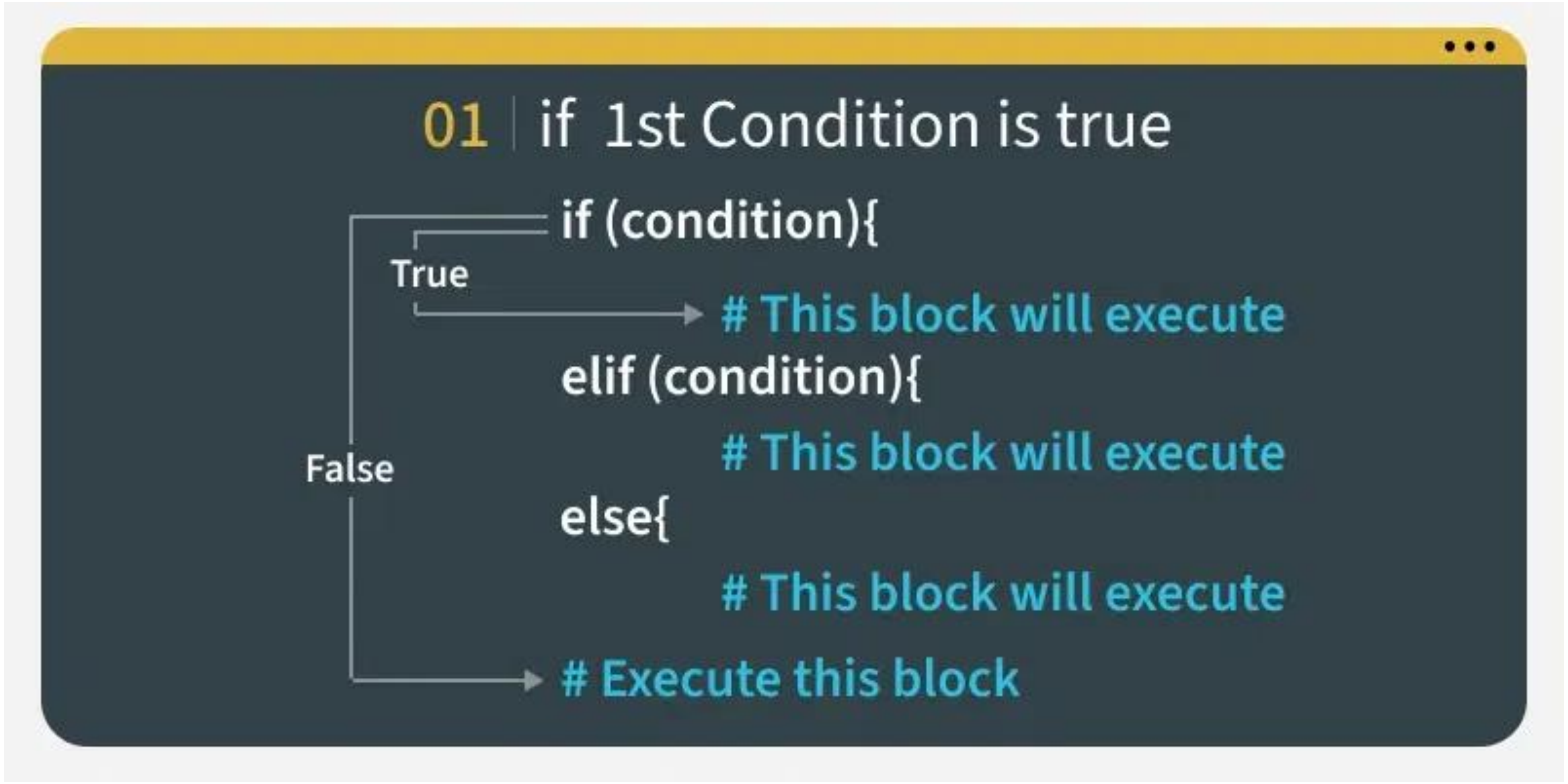
```
x=1

if x>2:

    if x>5:
        print('Bigger than 5')
    else:
        print('Smaller than 5')

print('Finished')
```

Nested if else statement


Nested if Statement
if (condition){
True
if (condition){
True
# This block will execute
}
False
# This block will execute
False
else {
# This block will execute

Source: Geeksforgeeks

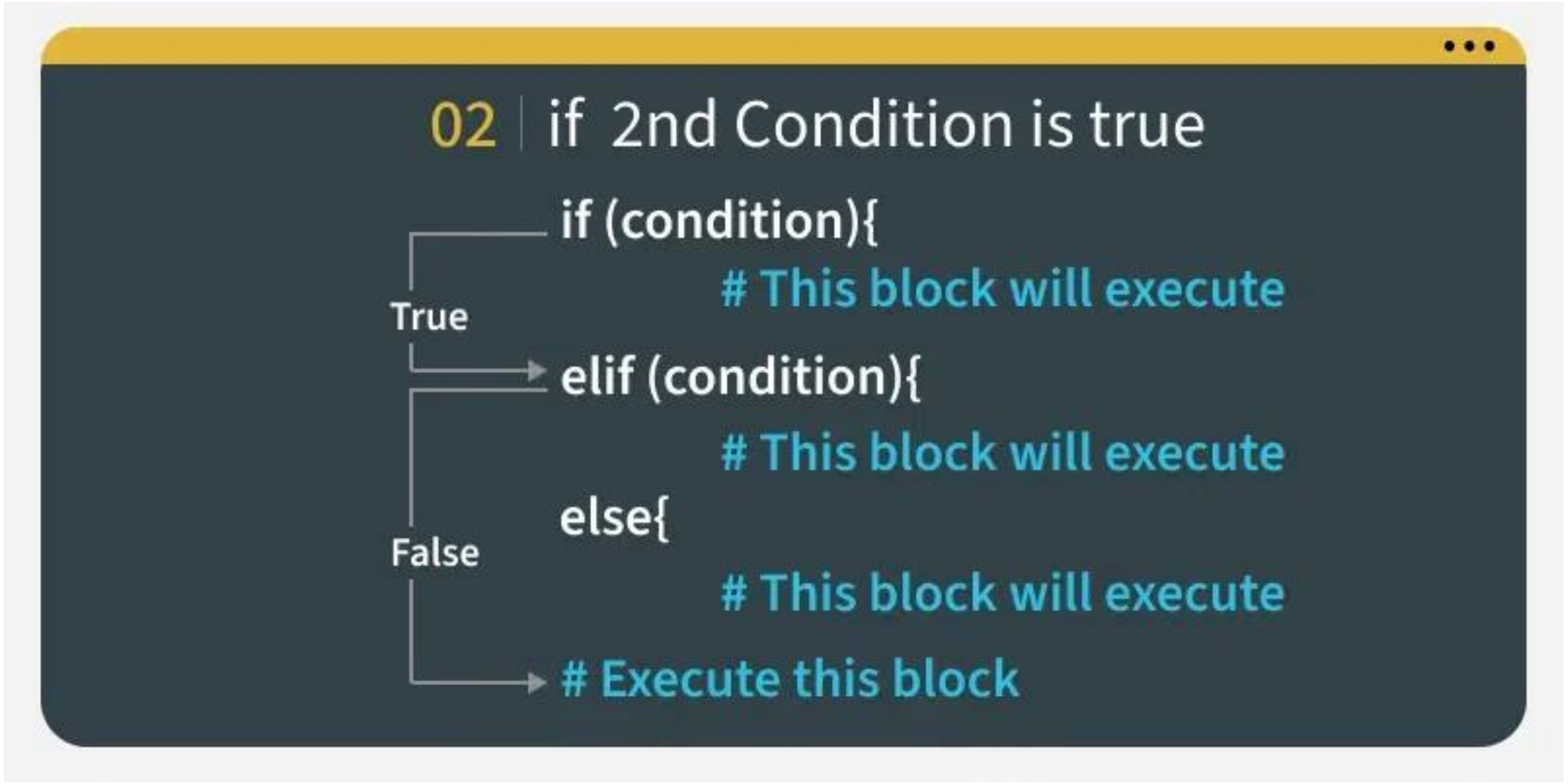
if...elif...else Statement


01 | if 1st Condition is true

```
if (condition){
        # This block will execute
elif (condition){
        # This block will execute
else{
        # This block will execute
# Execute this block
```

Source: Geeksforgeeks

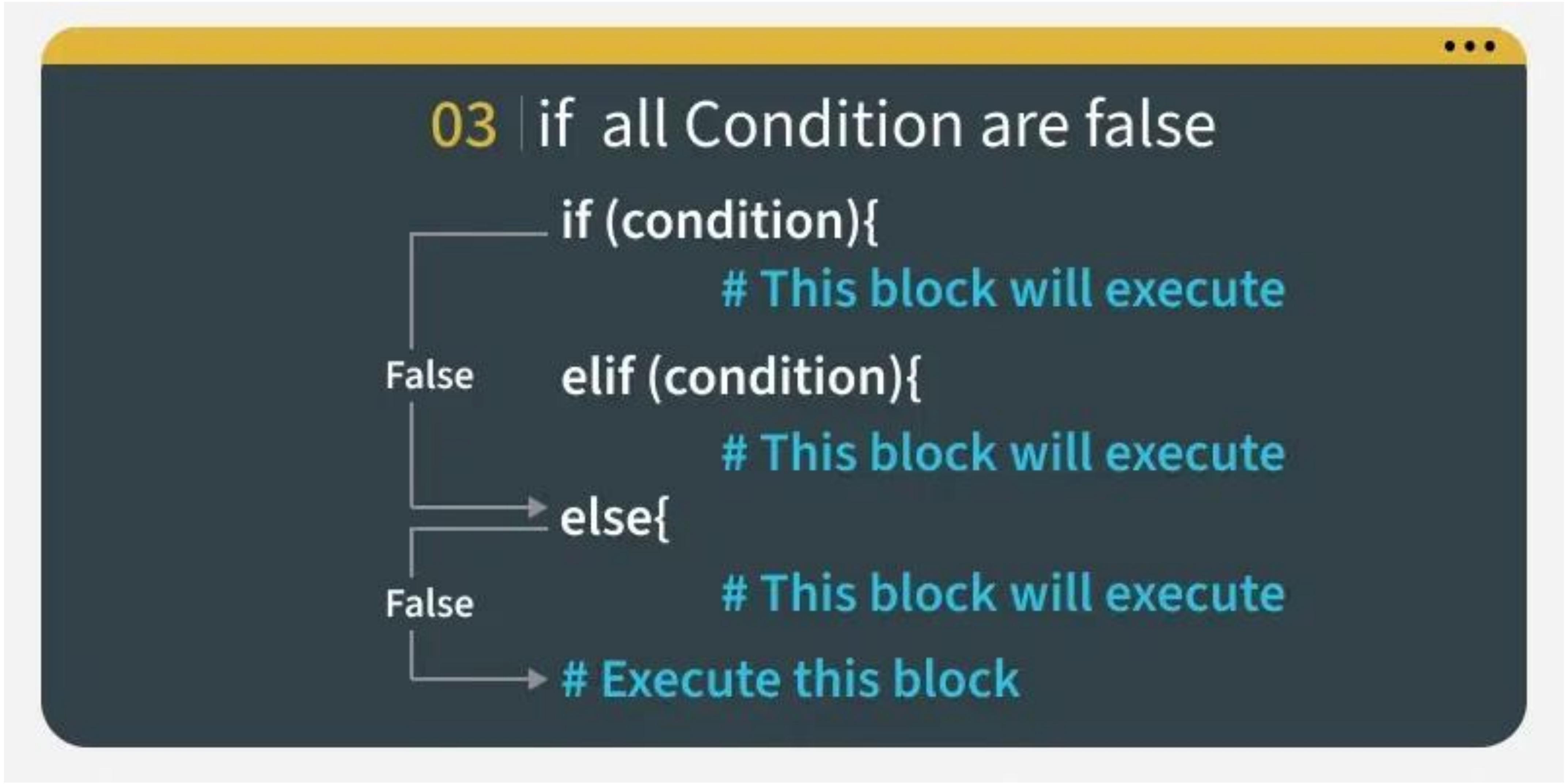
if...elif...else Statement



```
if (condition){
        # This block will execute
elif (condition){
        # This block will execute
else{
        # This block will execute
# Execute this block
```

Source: Geeksforgeeks

if...elif...else Statement

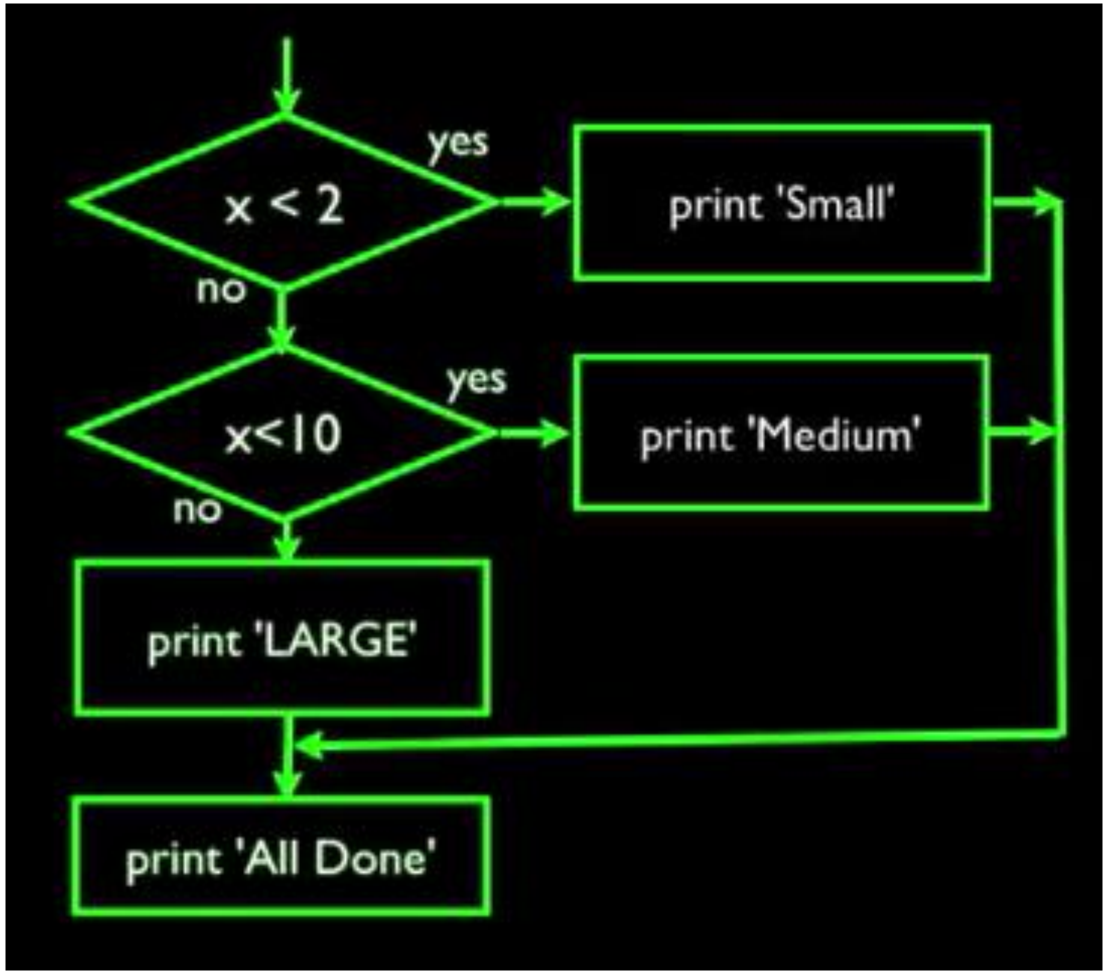


```
if (condition){
        # This block will execute
elif (condition){
        # This block will execute
else{
        # This block will execute
# Execute this block
```

Source: Geeksforgeeks

if...elif...else Statement

```
x=2
if x<2:
      print('Small')
elif x<10:
      print('Medium')
else:
      print('Large')

print('Finished')
```



Multi-way decision

```
#No else

x=2
if x<2:
    print('Small')
elif x<10:
    print('Medium')

print('Finished')
```

Multi-way decision

```
x=56
if x<2:
    print(’Small’)
elif x<10:
    print(’Medium’)
elif x<20:
    print(’Large’)
elif x<40:
    print(’Huge’)
else:
    print(’Ginormous’)

print(’Finished’)
```

Which will never print?

```
x=4

if x<=2:
    print('Below 2')
elif x>2:
    print('Above 2')
else:
    print('Something else')
print('Finished')
```

```
x=8

if x<2:
    print('Below 2')
elif x<20:
    print('Below 20')
elif x<10:
    print('Below 10')
else:
    print('Something else')
print('Finished')
```

Logical operators

- Logical operators can be used to combine several logical expressions into a single expression

- Python has three logical operators: not, and, or

Example

```
>>> not True
False
>>> False and True
False
>>> not False and True
True
>>> (not False) and True     # Same as previous statement.
True
>>> True or False
True
```

Example

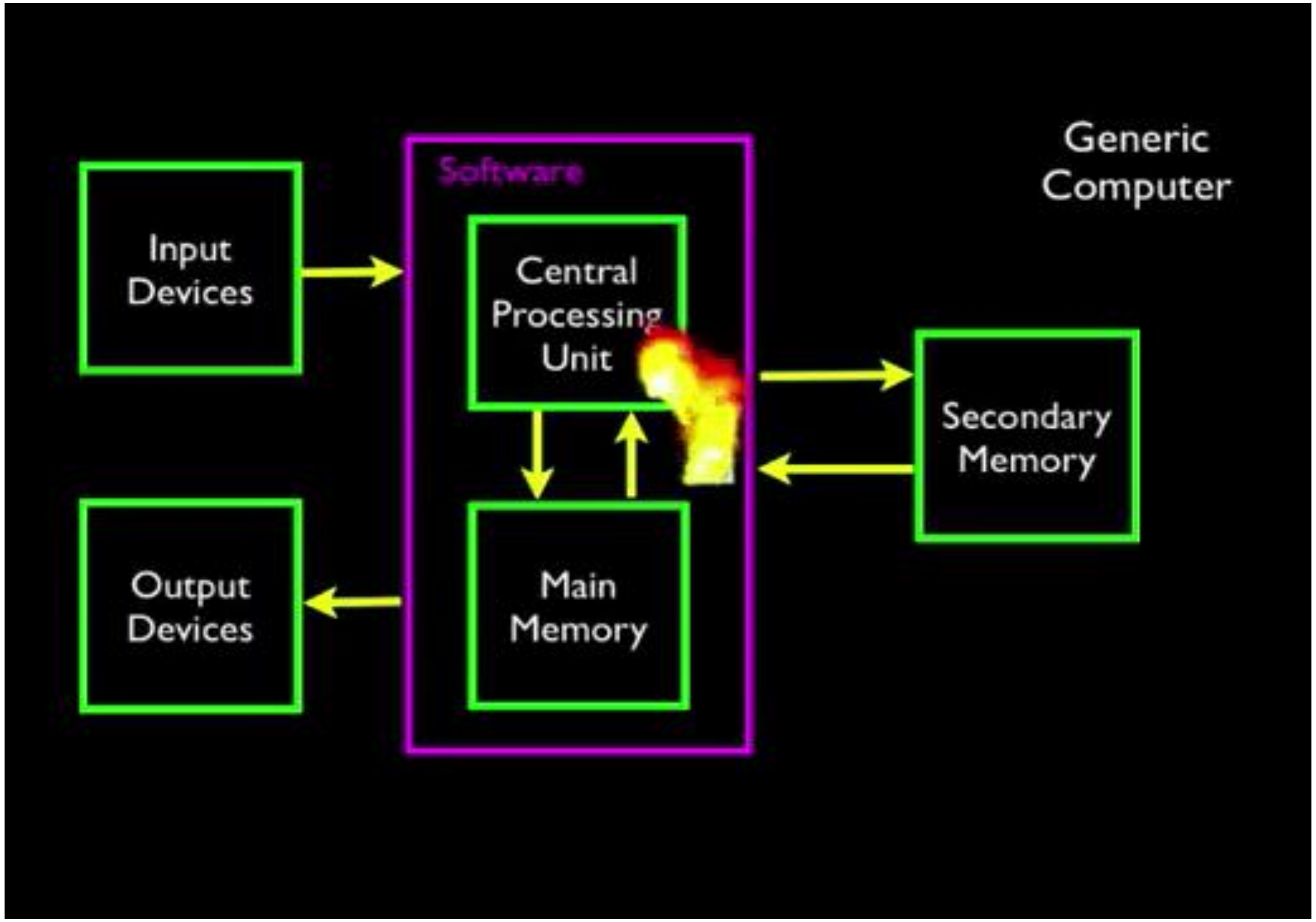
```
>>> not False or True          # Same as: (not False) or True.
True
>>> not (False or True)
False
>>> False and False or True    # Same as: (False and False) or True.
True
>>> False and (False or True)
False
```

Try/except structure

- You surround a dangerous part of code with try/except
- If the code in try block works, the except block is skipped
- If the code in try block fails, the except block will be executed

Example

```
astr = 'Hello bob'
istr = int(astr)
print('First',istr)

astr = '123'
istr = int(astr)
print('Second',istr)
```


Generic Computer
Software
Input Devices
Central Processing Unit
Secondary Memory
Output Devices
Main Memory

Use try/except to capture errors

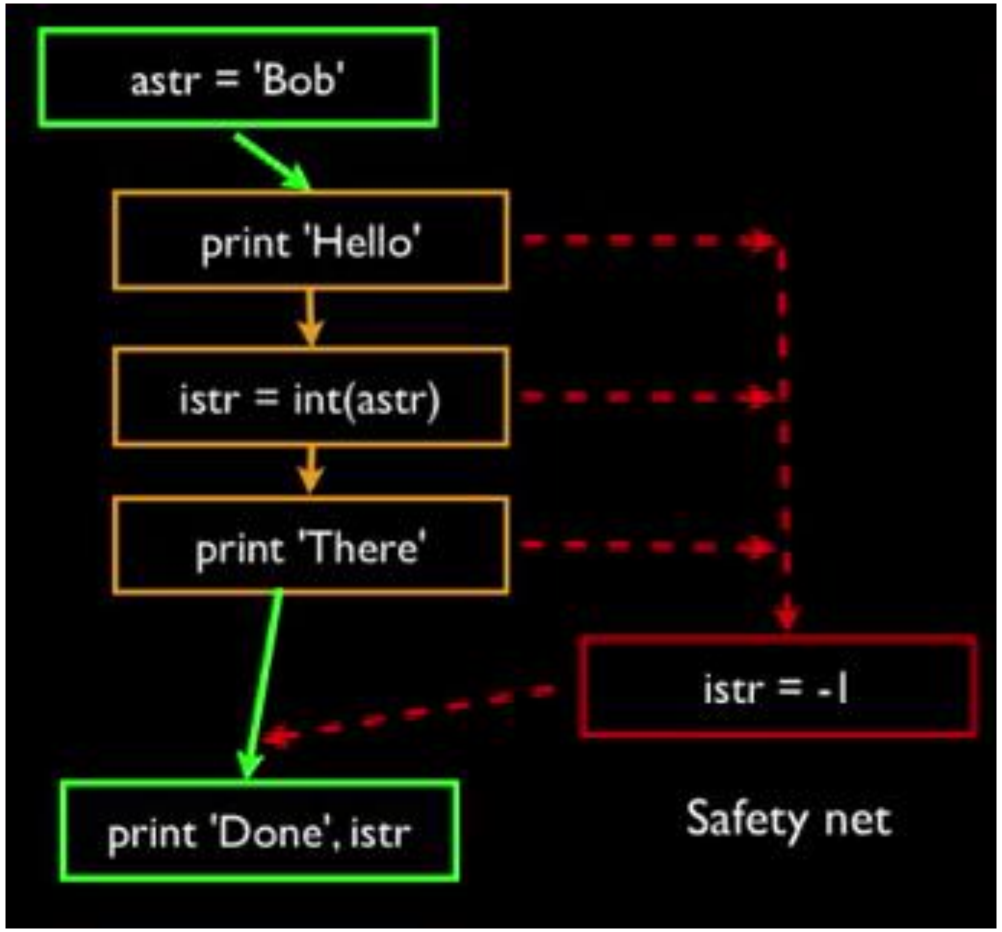
```
astr = 'Hello bob'
try:
    istr = int(astr)
except:
    istr = -1
print('First',istr)

astr = '123'
try:
    istr = int(astr)
except:
    istr = -1
print('Second',istr)
```

- When the first conversion fails, it just stops into the except block, and the program continues

- When the second conversion succeeds, it just skips the except block

Try/except

```
astr = 'Bob'
try:
    print('Hello')
    istr = int(astr)
    print('There')
except:
    istr = -1
print('Done',istr)
```



Example

```
rawstr = input('Enter a number:')

try:
    ival = int(rawstr)
except:
    ival = -1

if ival>0:
    print('Nice work')
else:
    print('Invalid number')
```

Practice

- Write a program to instruct the user to input the working hours and hourly rate, and then output the salary. If the working hours exceed 40 hours, then the extra hours received 1.5 times pay.

Practice

- Write a program to instruct a user to input a date (both month and day), and then output the new month and day when the inputted date is advanced by one day (leap years are ignored)

Answer

```
#Add a day to a given date

month = int(input('Enter a month (1-12):'))
day = int(input('Enter a day (1-31):'))

daysInMonth = (31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

if day<daysInMonth[month-1]:
    print(month, day+1)
else:
    month = month%12 + 1
    print(month, 1)
```

Repeated flow

Program

```
n=5
while n>0:
    print(n)
    n = n - 1
print("Finish")
```

Outputs

```
5
4
3
2
1
Finish
>>>
```

- Loops (repeated steps) have iterative variables that change each time through a loop
- Often these iterative variables go through a sequence of numbers

An infinite loop

```
n=5
while n>0:
    print('Lather')
    print('Rinse')
n=n-1
print('Dry off!')
```

- What is wrong with this program?

Another loop

```
n=0
while n>0:
    print('Lather')
    print('Rinse')
    n=n-1
print('Dry off!')
```

- What is wrong with this program?

Breaking out of a loop

- The break statement ends the current loop, and jumps to the statement which directly follows the loop

```
while (True):
    line = input('Enter a word:')
    if line == 'done':
        break
    print(line)
print('Finished')
```

Finishing an iteration with continue

```
while True:
    line = input('Input a word:')
    if line[0] == '#': continue
    if line == 'done':
        break
    print(line)
print('Done')
```

- The continue statement ends the current iteration, and start the next iteration immediately

Indefinite loop

- While loops are called “indefinite loops”, since they keep going until a logical condition becomes false
- Till now, the loops we have seen are relatively easy to check whether they will terminate
- Sometimes it can be hard to determine whether a loop will terminate

Definite loop

- Quite often we have a finite set of items
- We can use a loop, each iteration of which will be executed for each item in the set, using the for statement
- These loops are called “definite loops” because they execute an exact number of times
- It is said that “definite loops iterate through the members of a set”

A simple for loop

Example

```
for i in [5,4,3,2,1]:
    print(i)
print('Finished')
```

Output

```
5
4
3
2
1
Finished
```

Another example

Example

```
friends = ['Tom','Jerry','Bat']
for friend in friends:
    print('Happy new year',friend)
print('Done')
```

Output

```
Happy new year Tom
Happy new year Jerry
Happy new year Bat
Done
```

For loop

Example

```
for i in [5,4,3,2,1]:
    print(i)
print('Finished')
```

Output

```
5
4
3
2
1
Finished
```

- For loops (definite loops) have explicit iteration variables that change each time through a loop.
- These iteration variables move through a sequence or a set

In

- The iteration variable "iterates" through a sequence (ordered set)
- The block (body) of the code is executed once for each value in the sequence
- The iteration variable moves through all of the values in the sequence

Iteration variable

Sequence with five elements

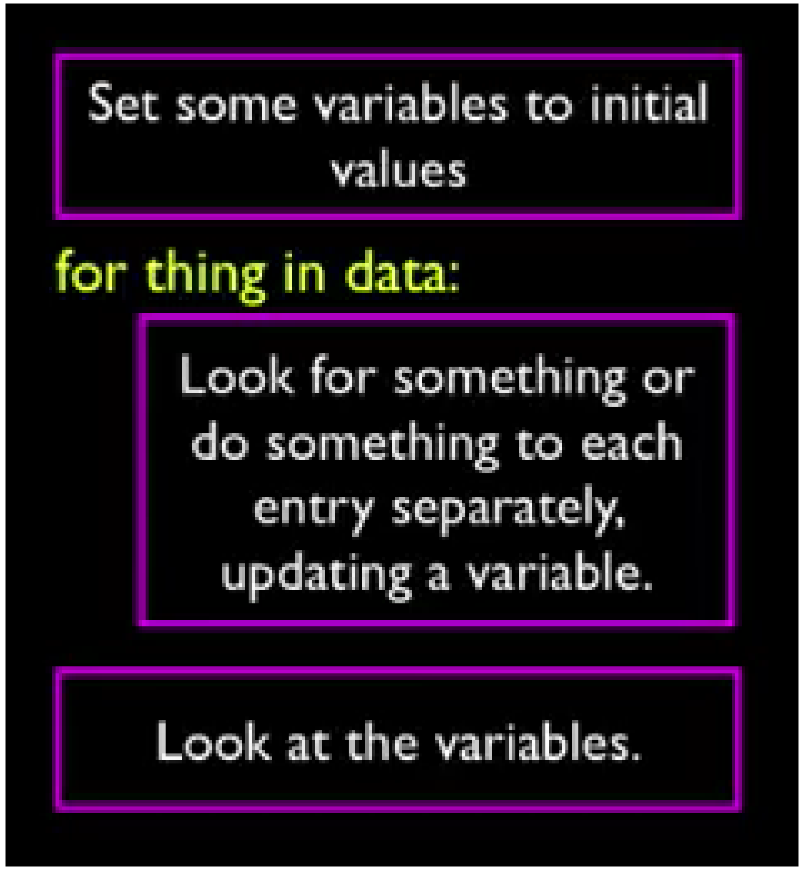
```
for i in [5, 4, 3, 2, 1]:
    print(i)
```

Loop patterns

- Note: though these examples are simple, the patterns apply to all kinds of loops

Making “smart” loops

- The trick is “knowing” something about the whole loop when you are stuck writing code that only sees one entry at a time


Set some variables to initial values
for thing in data:
Look for something or do something to each entry separately, updating a variable.
Look at the variables.

Looping through a set

Example

```
print('Before')
for thing in [3,5,100,34,6,87]:
    print(thing)
print('After')
```

Output

```
Before
3
5
100
34
6
87
After
```

Finding the largest number

Example

```
largest_so_far = -1
print('Before',largest_so_far)

for num in [9,39,21,98,4,5,100,65]:
    if num>largest_so_far:
        largest_so_far = num
    print(largest_so_far,num)

print('After',largest_so_far)
```

Output

```
Before -1
9 9
39 39
39 21
98 98
98 4
98 5
100 100
100 65
After 100
```

- Use a variable to store the largest number we have seen so far
- If the current number is larger, we assign it to the store variable

Counting in a loop

Example

```
count = 0
print('Before',count)
for thing in [3,4,98,38,9,10,199,78]:
    count = count + 1
    print(count,thing)
print('After',count)
```

Output

```
Before 0
1 3
2 4
3 98
4 38
5 9
6 10
7 199
8 78
After 8
```

- To count how many times we have executed a loop, we can introduce a counting variable, which increases itself in each iteration

Practice

- **Given a set of numbers, write a program to calculate their sum using for loop**

Answer

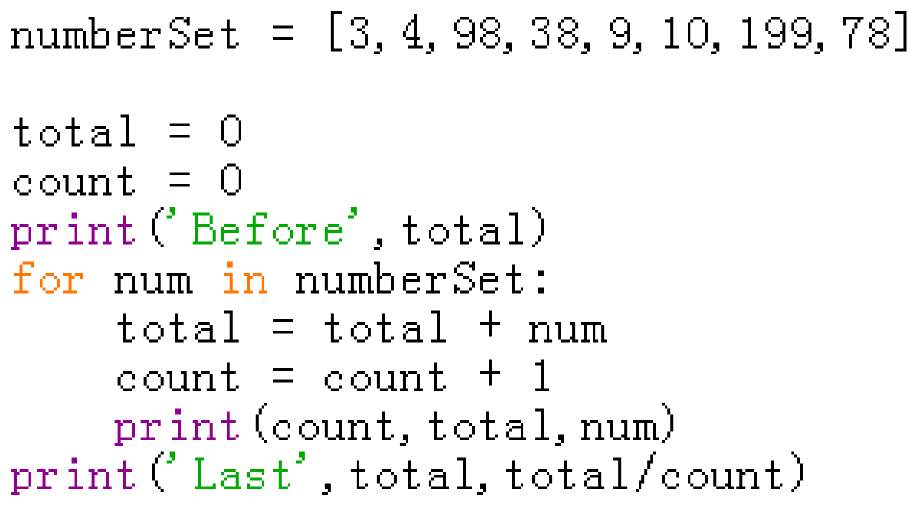
```
numberSet = [3, 4, 98, 38, 9, 10, 199, 78]

total = 0
print('Before',total)
for num in numberSet:
    total = total + num
    print(total,num)
print('Last',total)
```

```
Before 0
3 3
7 4
105 98
143 38
152 9
162 10
361 199
439 78
Last 439
```

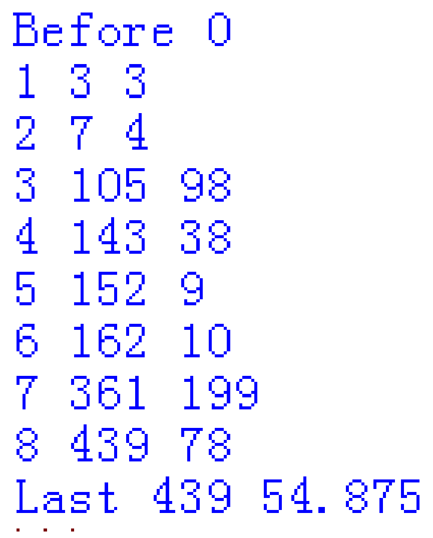
Practice

- **Given a set of numbers, write a program to calculate their average using for loop**

Answer



```python
numberSet = [3,4,98,38,9,10,199,78]

total = 0
count = 0
print('Before',total)
for num in numberSet:
    total = total + num
    count = count + 1
    print(count,total,num)
print('Last',total,total/count)
```



```
Before 0
1 3 3
2 7 4
3 105 98
4 143 38
5 152 9
6 162 10
7 361 199
8 439 78
Last 439 54.875
```

Filtering in a loop

Example

```
print('Before')

for value in [23,3,43,39,80,111,99,3,65]:
    if value>50:
        print('Large value:',value)

print('After')
```

Output

```
Before
Large value: 80
Large value: 111
Large value: 99
Large value: 65
After
```

- We can use an if statement in a loop to catch/filter the values we are interested at

Search using a Boolean variable

Example

```
found = False

print('Before',found)

for value in [9,41,12,3,74,15]:
    if value == 74:
        found = True
    print(found,value)
print('After',found)
```

Output

```
Before False
False 9
False 41
False 12
False 3
True 74
True 15
After True
```

- If we want to search in a set and double check whether a specific number is in that set
- We can use a Boolean variable, set it to False at the beginning, and assign True to it as long as the target number is found

Finding the largest number

Example

```
largest_so_far = -1
print('Before',largest_so_far)

for num in [9,39,21,98,4,5,100,65]:
    if num>largest_so_far:
        largest_so_far = num
    print(largest_so_far,num)

print('After',largest_so_far)
```

Output

```
Before -1
9 9
39 39
39 21
98 98
98 4
98 5
100 100
100 65
After 100
```

- Use a variable to store the largest number we have seen so far
- If the current number is larger, we assign it to the store variable

Finding the smallest number

```
smallest_so_far = -1
print('Before',smallest_so_far)

for num in [9,39,21,98,4,5,100,65]:
    if num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far,num)

print('After',smallest_so_far)
```

- Use a variable to store the smallest number we have seen so far
- If the current number is smaller, we assign it to the store variable
- What is the problem with this program?

Finding the smallest number

Example

```
smallest_so_far = None
print('Before',smallest_so_far)

for num in [9,39,21,98,4,5,100,65]:
    if smallest_so_far == None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far,num)

print('After',smallest_so_far)
```

Output

```
Before None
9 9
9 39
9 21
9 98
4 4
4 5
4 100
4 65
After 4
```

- We still use a variable to store the smallest value seen so far
- In the first iteration, the smallest value is none, so we need to use an if statement to check this

The is and is not operator

```
smallest_so_far = None
print('Before',smallest_so_far)

for num in [9,39,21,98,4,5,100,65]:
    if smallest_so_far is None:
        smallest_so_far = num
    elif num < smallest_so_far:
        smallest_so_far = num
    print(smallest_so_far,num)

print('After',smallest_so_far)
```

- Python has a "is" operator which can be used in logical expression
- Implies "is the same as"
- Similar to, but stronger than ==
- "is not" is also an operator

Is operator

Example

```
print(10 is 10)

a = 10
b = 10
print (a is b)

a = ’123’
b = ’123’
print (a is b)

a = [1,2,3]
b = [1,2,3]
print (a is b)
```

Output

```
True
True
True
False
```