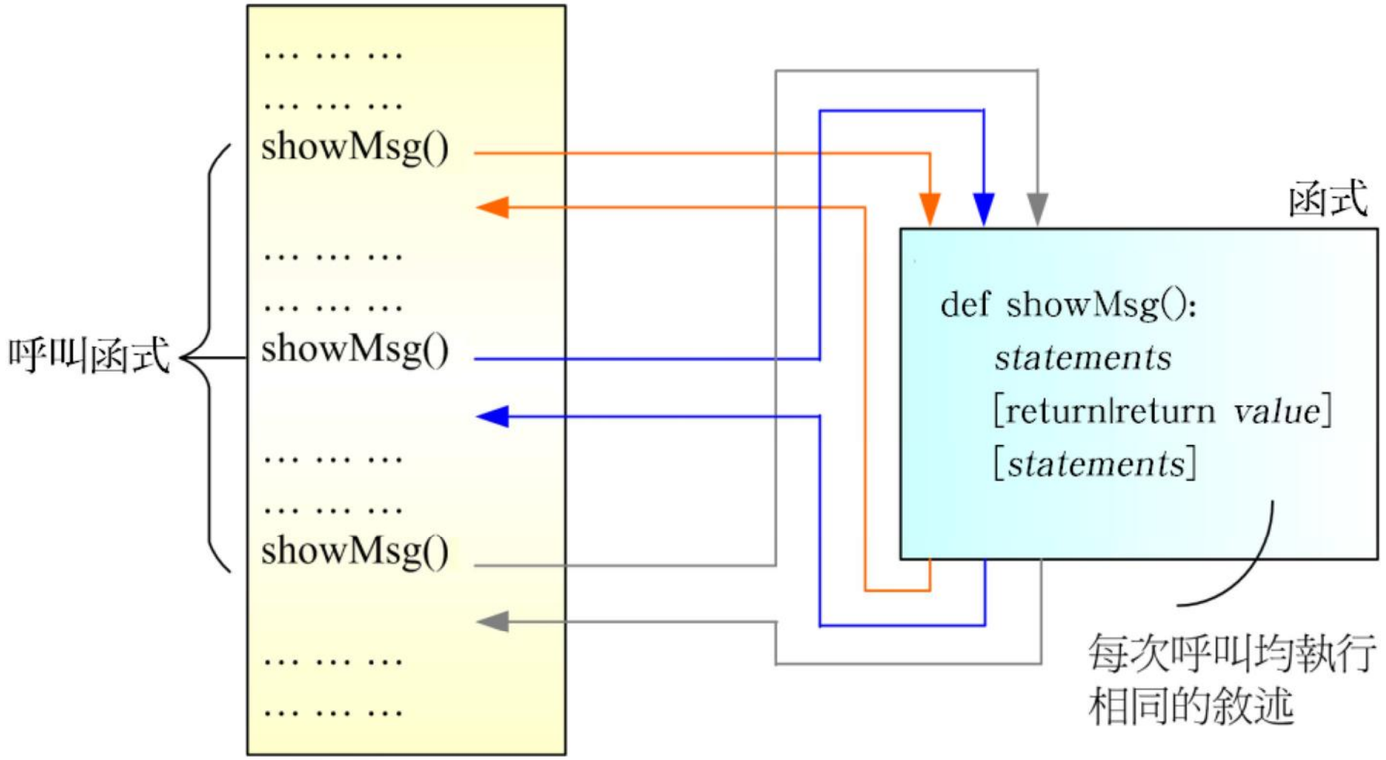
Function

Python Fundamental

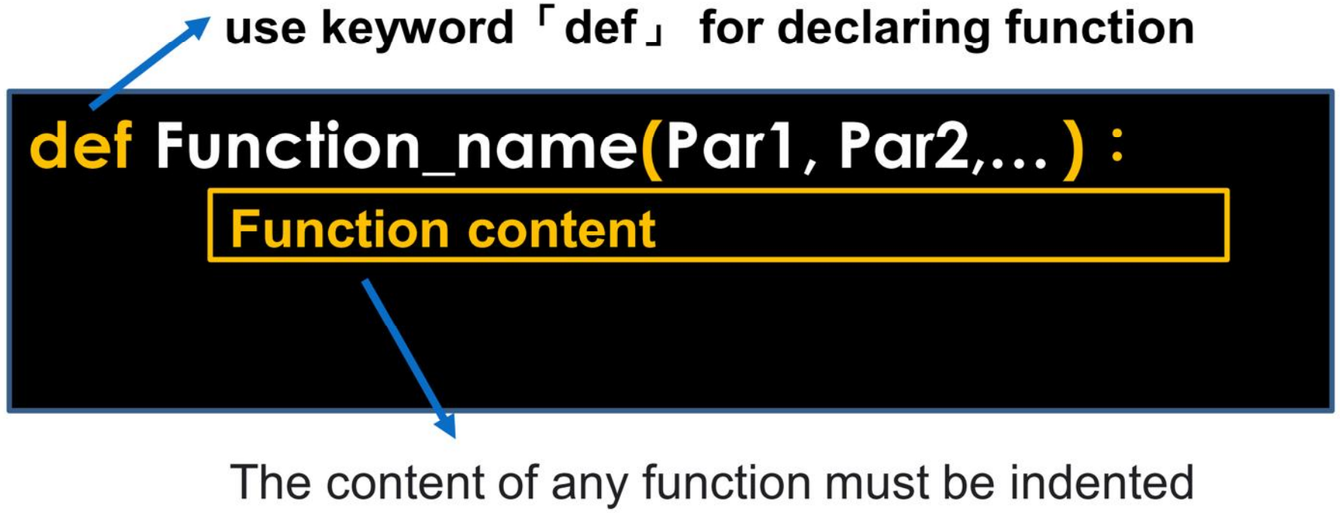
Recognition function

- Function, function, is mainly to write a section of a narrative with a certain function or repeated use into an independent program block, and then give it a name for subsequent calls, which can simplify the program and improve readability.
- Some programming languages call functions as functions, methods, procedures, or subroutines, and the concepts are similar.



Declare function

- If it is not a built-in function, we can declare and define the function by ourselves.
- The function must be declared before it can be called.



Declare function

- We can use the def keyword to declare functions, the syntax is as follows:

```
def functionName([parameters]) :
    statements
    [return|return value]
    [statements]
```

```
def degree(degreeC):
    degreeF = degreeC * 1.8 + 32
    print('Celsius', degreeC, 'can be converted to Fahrenheit ', degreeF, 'degree')
```

Call function

- After the function is declared and defined, it still needs to be called before it can be executed.
- Call function method: the function name is the same, and the number in the bracket is the same, then the function can be called.

```
Function_name ( Arg1, Arg2,… )
```

Call function

```
def degree(degreeC):
    degreeF = degreeC * 1.8 + 32
    print('Celsius', degreeC, 'can be converted to Fahrenheit ', degreeF, 'degree')


Celsius=input('Please enter Celsius degree:')
Celsius=int(Celsius)
degree(Celsius)
```

```
Please enter Celsius degree:50
Celsius 50 can be converted to Fahrenheit  122.0 degree
```

Default parameter value

- We can set the default argument value when defining the function.

```
def teaTime(dessert, drink="green tea"):
    print('My dessert:', dessert,',drink:', drink)


teaTime('cake')
teaTime('Macaron','coffee')
```

```
My dessert: cake ,drink: green tea
My dessert: Macaron ,drink: coffee
```

String function

- Python has many built-in functions for processing string, and the commonly used functions are explained below.
  - String conversion function
  - String Delete the specified character or blank function

String function

- **String conversion function**
  - str.upper(s) # s are converted to uppercase
  - str.lower(s) # s are converted to lowercase
  - str.swapcase(s) # returns the string s with the case of the parameter s interchanged
  - str.replace(old, new) # returns the string s that replaces the string parameter old with the string parameter new

String function

- **String conversion function**

```
In [1]: X='Good Afternoon! How are you?'

In [2]: print(X.upper())
GOOD AFTERNOON! HOW ARE YOU?

In [3]: print(X.lower())
good afternoon! how are you?

In [4]: print(X.swapcase())
gOOD aFTERNOON! hOW ARE YOU?

In [7]: print(X.replace('oo','xx'))
Gxxd Afternxxn! How are you?
```

String substring function

- String cutting function and delete specified character or blank function
  - str.split([chars])
  - str.strip([chars])

String function

String cutting function

- str.split([chars]) :
  - Cut from the entire string according to the parameter settings (chars).
  - The parameter chars can be omitted, which means character string is cut with a blank space.

```
In [1]: X='A B C D E'

In [2]: Y=X.split()

In [3]: print(Y)
['A', 'B', 'C', 'D', 'E']
```

```
In [1]: X='1,3,5,7,9'

In [2]: Y=X.split(',')

In [3]: print(Y)
['1', '3', '5', '7', '9']
```

String function

- **delete specified character or blank function**

  - str.strip([chars]) :

    - Delete the characters specified by the parameter chars from both sides of the string, stop deleting once it encounters characters that are not specified, and then return the remaining string.
    - The chars parameter can be omitted, which means that the specified character is blank, that is, the blank on both sides of the string is deleted.

```
In [1]: print('  abcde   '.strip())
abcde
```

Delete the white space on both sides of the string

```
In [2]: print('abcxyzd ef'.strip('abcdef'))
xyzd
```

Delete the abcdef characters on both sides of the string

Numerical function

- abs(x) : returns the absolute value of the numeric parameter x.

```
In [1]: print(abs(50))
50

In [2]: print(abs(-50))
50
```

- pow(x,y) : returns the value of the numeric parameter x to the power of y.

```
In [1]: print(pow(2,5))
32

In [2]: print(pow(3,4))
81
```

- divmod(x,y) : returns the quotient and remainder of X divided by Y.

```
In [1]: divmod(100,8)
Out[1]: (12, 4)

In [2]: divmod(125,9.5)
Out[2]: (13.0, 1.5)
```

Numerical function

- max/min(x1, x2, x3…) : returns the maximum/minimum value in the parameter.

```
In [1]: print(max(10,20,30))
30

In [2]: print(max(-10,-20,-30))
-10

In [3]: print(max(True,-20,-30))
True
```

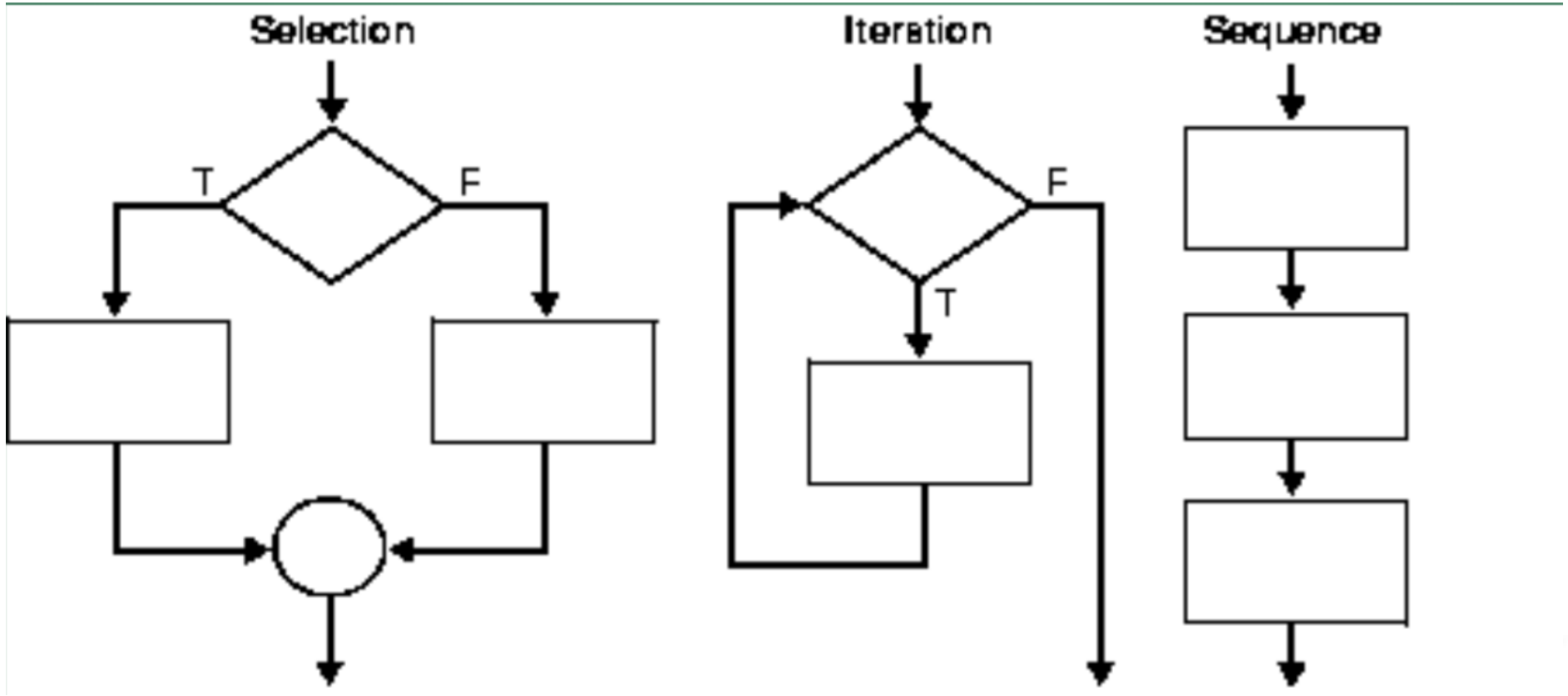
Q & A

Control Flow

Python Fundamental

Introduction to Control Flow

- The process of simple program starts from the first line and executes line by line in order, without skipping or turning.
- However, most program execute depending on different situations to complete more complex tasks. Therefore, flow control is needed to assist the program.



Type of Control Flow

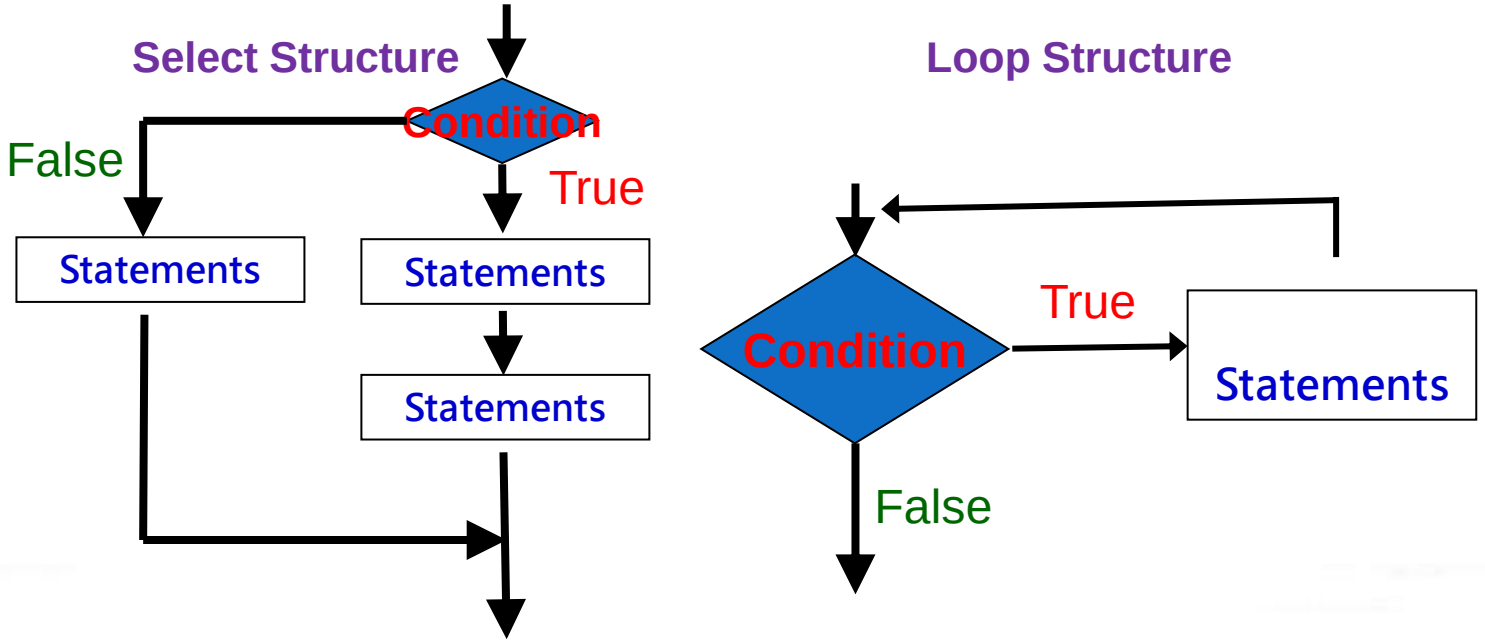
- Python's flow control is divided into the following two types:

  - **Select structure：**

    Execute different program descriptions based on whether the result of the condition is True or False。

  - **Loop structure：**

    Depending on whether the result of the condition is True or False, some narratives are repeated until a certain condition is met.

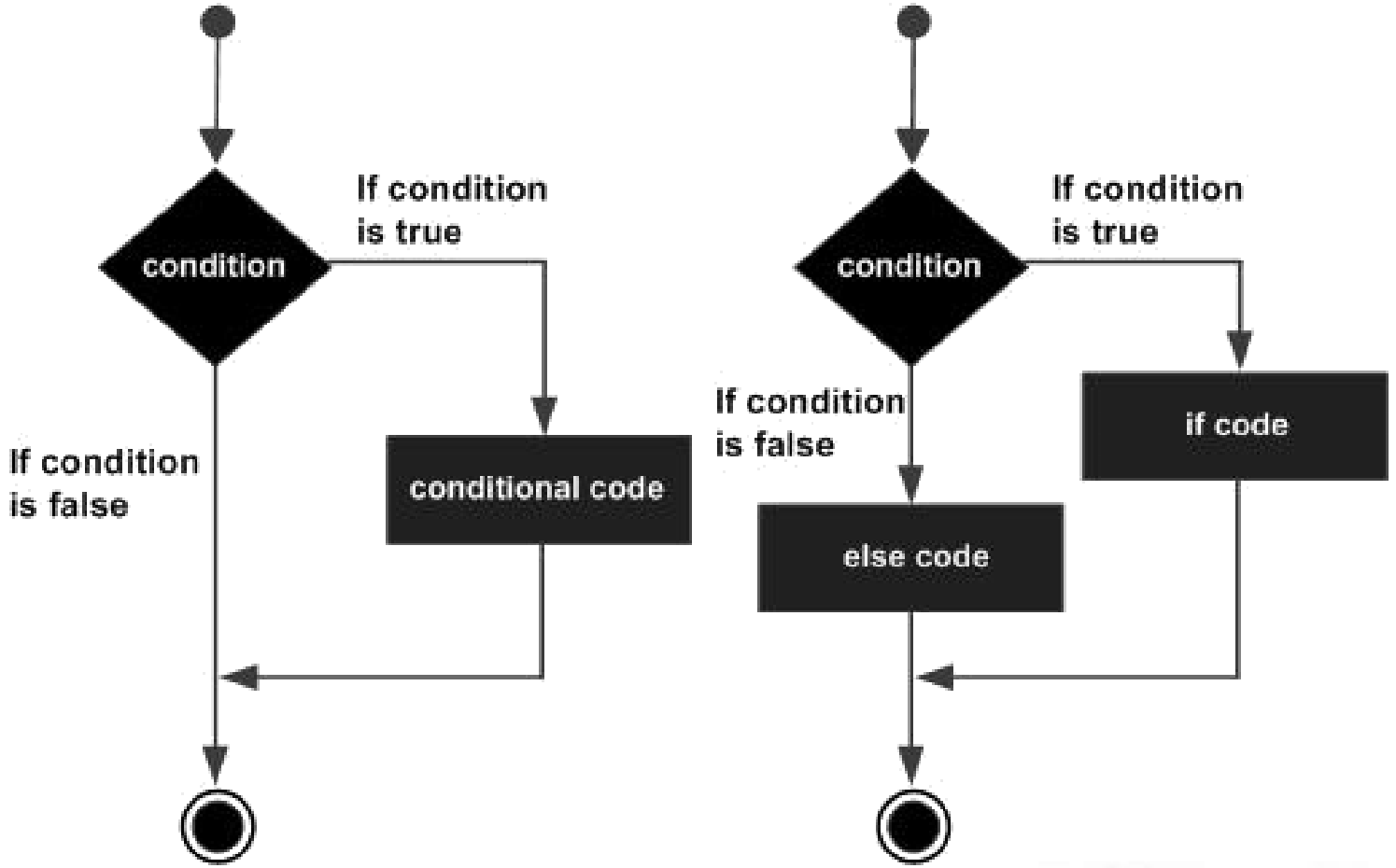


Select structure

Select structure type

- The selection structure can be divided into the following types

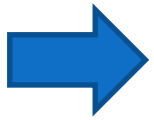
  - "One-way → if
  - "Two-way → if… else"
  - "Multidirectional → if...elif...else"



Select structure

- The selection structure can be used to check conditional expressions, and execute different statements based on whether the result of the condition is True or False.
- The program block starts with a colon ":", and then the same block range must have the same indentation.
- Do not mix different amounts of blanks, nor mix blanks and tabs

```
a=100
b=200
if a<b:
    print(a)
```



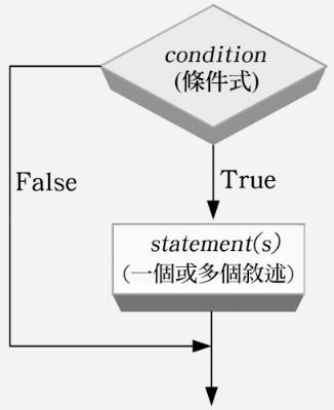
100

if-one-way if (if it is true... just do...)

- "If..." is the syntax of one-way selection, which means "if it is true (True), just do...".
- If the conditional expression is true (True), the statement in the indentation is executed; if the conditional expression is not established (False), the statement in the indentation is not executed.
- Statements(s) must be indented to the right by at least one blank (the default is four), and the indentation must be aligned to indicate that these statements are in a block.
- Since Python uses indentation to divide the execution block of the program, the program cannot be indented at will.

```
if condition :
    statement(s)
```

Indent →



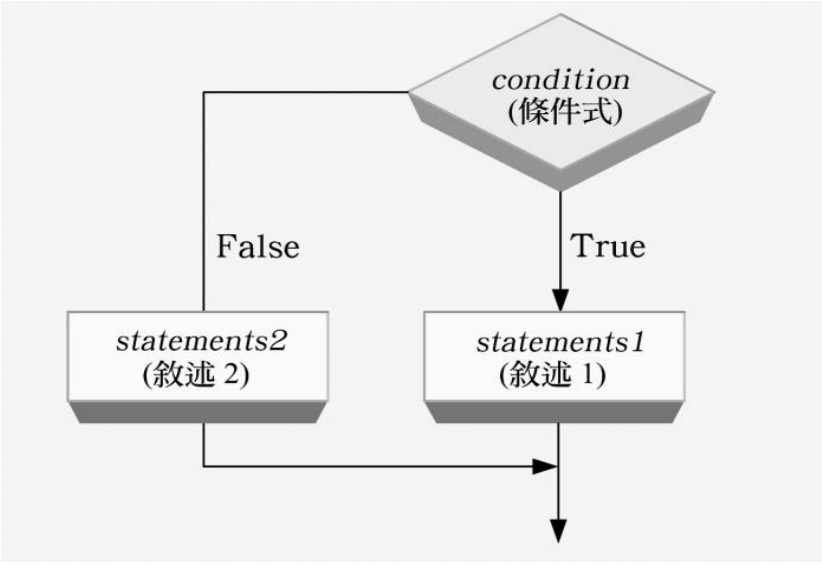

```
x = 15

y = 10

if x > y:

    z = x - y

    print(x,"is", z,"bigger than", y)
```

```
15 is 5 bigger than 10
```

if...else-bidirectional if (2 choose 1)

- "If...else" is a two-way selection syntax, which means "if it is true (True), ... execute the content of the if block, if it is not true (False), execute the content of the else block.
  - If the conditional expression is true, statements1 are executed, and statements2 are not executed.
  - If the conditional expression is not established, statements2 will be executed instead of statements1.

```
if condition:
    statements1
else:
    statements2
```



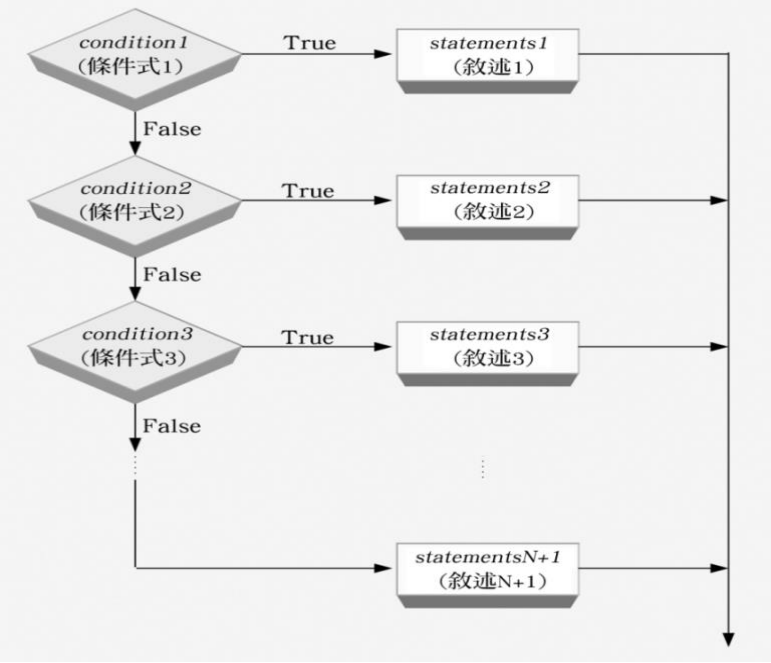

```
score = eval(input("Please enter a score(0 ~ 100) : "))
if score >= 60:
    print("Pass ! ")
else:
    print("Fail ! ")
```

```
Please enter a score(0 ~ 100) : 80
Pass !

Please enter a score(0 ~ 100) : 50
Fail !
```

if...elif...else-multi-directional if (multiple choice 1)

- "If...elif...else" is a syntax for multi-directional selection, meaning "if it is true...just...if not, then if...is true...just..."
- The elif keyword is short for else if, which can be none, one or more.
- Compared with elif, there can be no or only one else, and it must be placed last.
- Only one set of statements1~~~statmentsN+1 will be executed.

```
if condition1:
    statements1
elif condition2:
    statements2
elif condition3:
    statements3
…
else:
    statementsN+1
```



if...elif...else-multi-directional if example

```
score = eval(input("Please enter score (0 ~ 100) : "))
if score >= 90:
    print("Grade A")
elif score >= 80:
    print("Grade B")
elif score >= 70:
    print("Grade C")
elif score >= 60:
    print("Grade D")
else:
    print("Fail")
```

```
Please enter score (0 ~ 100) : 90
Grade A

Please enter score (0 ~ 100) : 85
Grade B

Please enter score (0 ~ 100) : 76
Grade C

Please enter score (0 ~ 100) : 68
Grade D

Please enter score (0 ~ 100) : 50
Fail
```

if-nested if

- Nested if refers to the if statement contains other if statements, and there is no depth limit.
- Because there is no depth limit, the indentation level must be correct so that no error will occur.
- If you want to avoid too deep and difficult to read, it is recommended to adopt multi-directional if...elif...else.

```
score = eval(input("Please eneter score(0 ~ 100) : "))
if score >= 90:
    print("Grade A")
else:
    if score >= 80:
        print("Grade B")
    else:
        if score >= 70:
            print("Grade C")
        else:
            if score >= 60:
                print("Grade D")
            else:
                print("Fail")
```

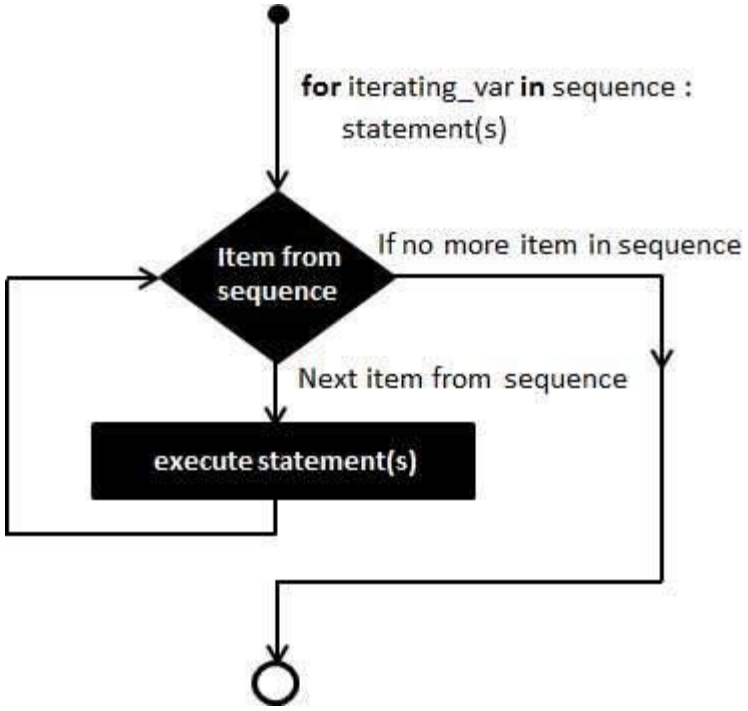
```
Please enter score (0 ~ 100) : 90
Grade A

Please enter score (0 ~ 100) : 85
Grade B

Please enter score (0 ~ 100) : 76
Grade C

Please enter score (0 ~ 100) : 68
Grade D

Please enter score (0 ~ 100) : 50
Fail
```

Looping Structure

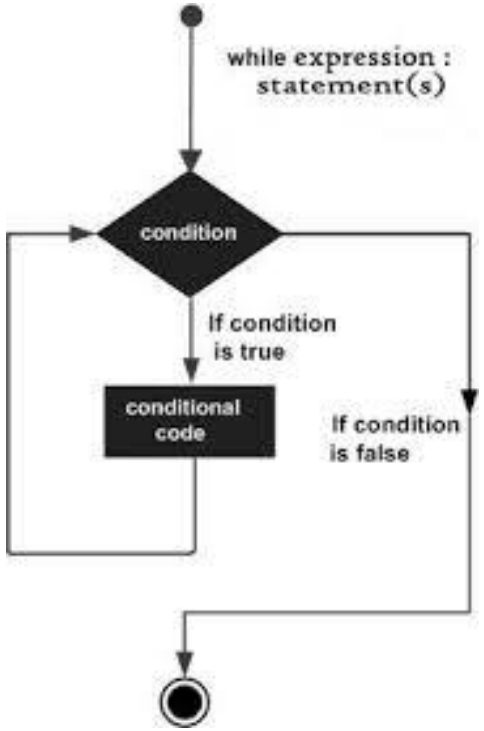
Loop structure

- Each repetition of the process is called "iteration (repetition operation)", and the result of each iteration will be used as the initial value of the next iteration.
- Iteration is the activity of repeating the process, the purpose is to reach the desired goal or result, and the loop is the way to solve the repeated execution.

for Looping

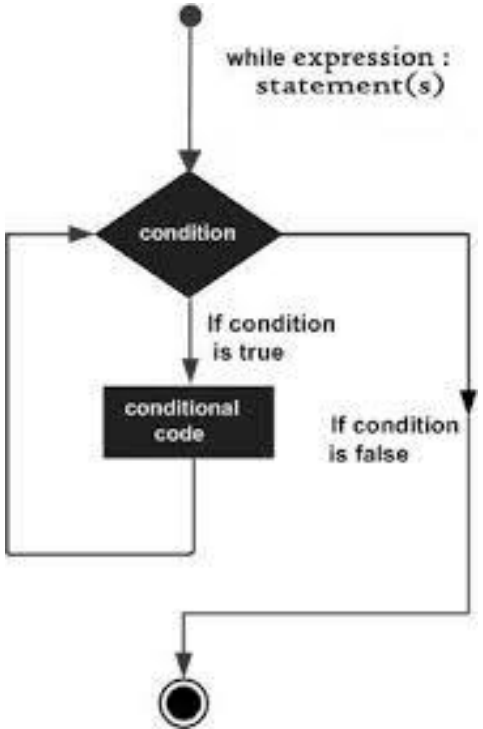


while Looping



Loop structure

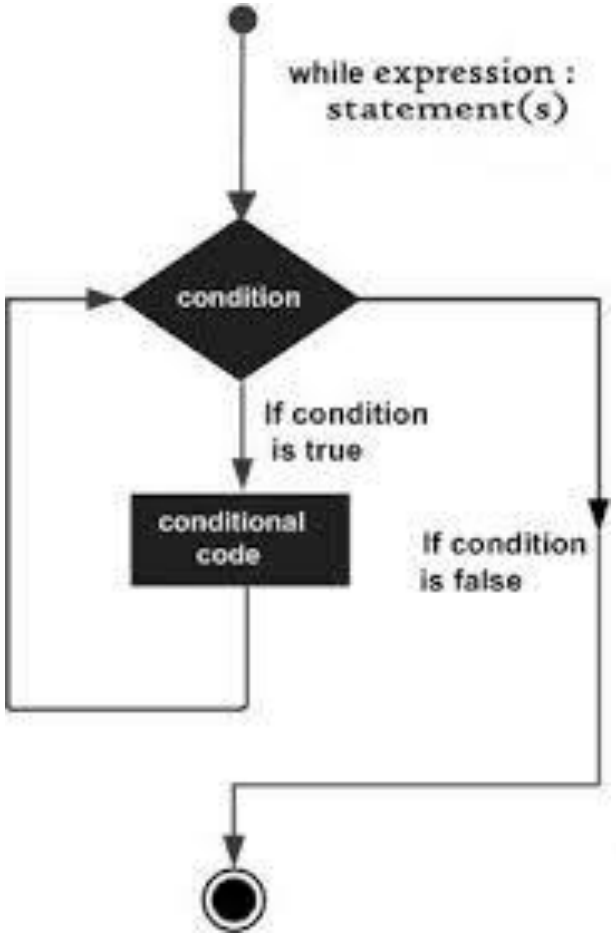
- When the condition is true (True), the block (suite) calculation is performed.
- After the block is executed, check the conditions again, if it is still true, execute the suite, otherwise start to execute the description after the block.
- This repeated structure is called a loop.
- The action of not continuing to execute the block is called jumping out of the loop or leaving the loop.




```
while condition :
    suite
```

while loop

- Use the while loop when the number of repetitions is difficult to calculate clearly.
- The syntax of while is as follows:




```
while condition:
    statements1
[else:
    statements2]
```

```
i = 0
while i < 5:
    print(i)
    i = i + 1
```

```
0
1
2
3
4
```

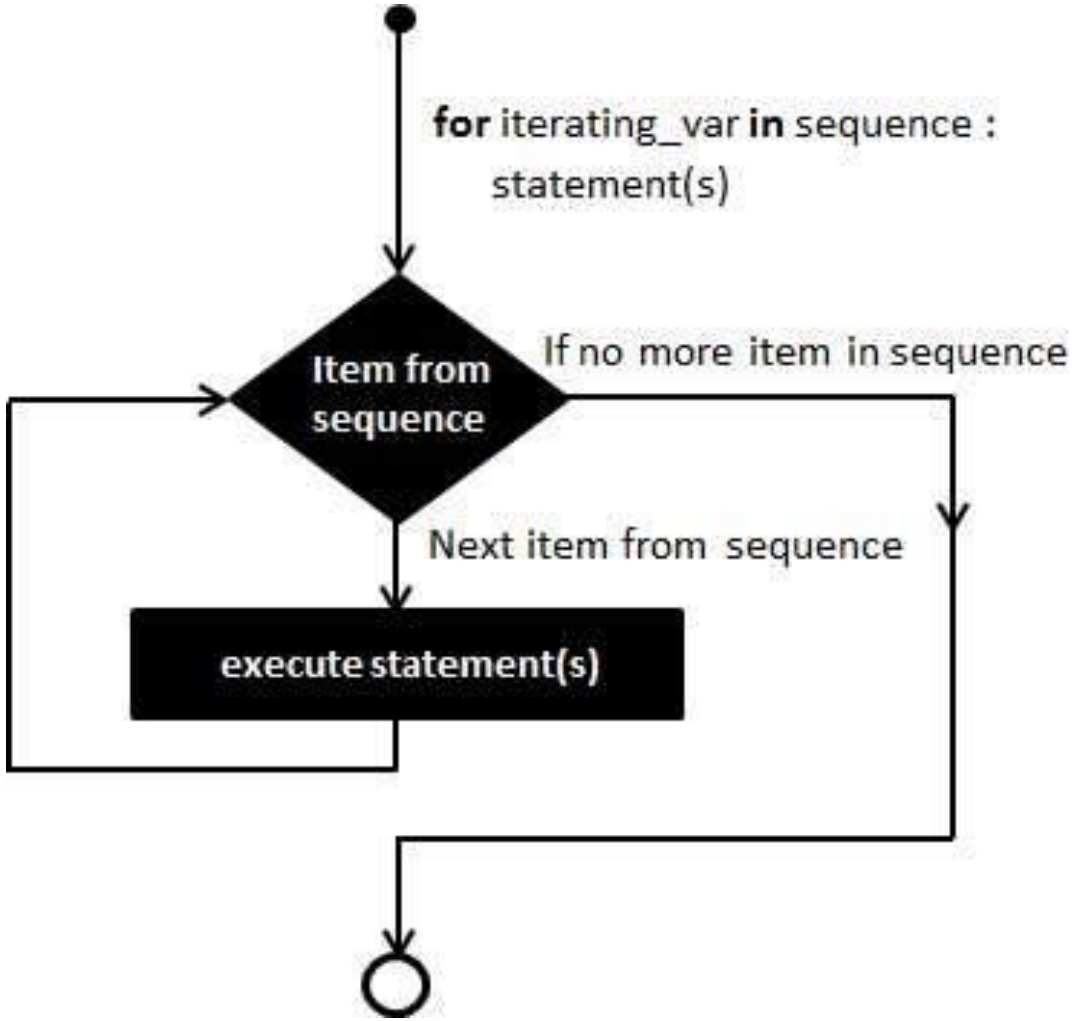
while loop

```
answer = input("Please enter Number 「1」:")
while answer!= "1":
    answer = input("It's wrong, please enter again:")
else:
    print("Got it!")
```

```
Please enter Number 「1」:5

It's wrong, please enter again:3

It's wrong, please enter again:8

It's wrong, please enter again:2

It's wrong, please enter again:1
Got it!
```

for loop

- The for loop is used when the number of repetitions can be clearly calculated, so the for loop is also called "counting loop".



for loop

- The syntax of the for loop is as follows. Iterator is an object with order and repetitive operations, such as the range() function or an ordered sequence of strings, lists, etc.

```
for var in iterator:
    statements1
```

```
range(stop)
range(start, stop [, step])
```

- The loop body statements1 must be indented to the right based on the for keyword, which means that it is in the for block. The else clause is optional and can be set or omitted.

- The syntax of the for loop is as follows. Iterator is an object with order and repetitive operations, such as the range() function or an ordered sequence of strings, lists, etc.

- The loop body statements1 must be indented to the right based on the for keyword, which means that it is in the for block. The else clause is optional and can be set or omitted.

for loop-range() function application

- We can use Python's built-in range() function to generate range objects:

1. range(stop)
2. range(start, stop)
3. range(start, stop, step)

```
# The start value is 0, the end value is 5 (exclusive), and the interval value is a sequence of integers.
list(range(5))
[0, 1, 2, 3, 4]
# The start value is 1, the end value is 10 (exclusive), the interval value is a sequence of integers
list(range(1, 10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
# The start value is 10, the end value is -10 (excluding), the interval value is a sequence of integers of -2
list(range(10, -10, -2))
[10, 8, 6, 4, 2, 0, -2, -4, -6, -8]
```

for loop-range() function application

```
# When i is not equal to the end value of 5, print i; when i is equal to the end value of 5, jump out of the loop.
for i in range(5):
    print(i)
```

```
0
1
2
3
4
```

```
name='Bob'
for i in range(len(name)):
    print(i, name[i])
```

```
0 B
1 o
2 b
```

```
name='Bob'
for i in name:
    print(i)
```

```
B
o
b
```

break & continue

- The break statement can be forced to leave the loop, usually used in conjunction with condition judgment.

```
answer = input("please enter number, -1 for quit:")

while answer!= "36":
    if answer.upper() == "-1":
        print("Bye! See you Next Time")
        break

    answer = input("Wrong, please enter number again:")
else:
    print("you are right!")
```

```
please enter number, -1 for quit:15

Wrong, please enter number again:35

Wrong, please enter number again:-1
Bye! See you Next Time
```

```
please enter number, -1 for quit:12

Wrong, please enter number again:25

Wrong, please enter number again:84

Wrong, please enter number again:69

Wrong, please enter number again:36
you are right!
```

break & continue

- The continue statement is used to skip the following statement in the loop and return directly to the beginning of the loop.

```
i = 0
while i < 10:
    i = i + 1
    if i % 3 != 0:
        continue
        i = i + 2
    print(i)
```

```
3
6
9
```

Q & A