Python program can be divided into 3 kinds of structure, namely the sequence structure, selection (branch) structure and loop structure:

- A sequential structure is one in which a program executes each piece of code in a sequence from beginning to end, without repeating or skipping over any code.
- Selection structures, also known as branching structures, allow programs to execute code selectively. In other words, you can skip over useless code and execute only useful code.
- Loop structure is to let the program repeatedly execute the same piece of code.

The sequence structure is well understood and needs no introduction. We will focus on the selection structure and the loop structure.

comparison operator

A comparison operator, also known as a relational operator, is used to size a constant, a variable, or the result of an expression. Returns True if the comparison is valid and False if it is not.

True and False are bool types that are used to indicate whether something is True or False, or whether an expression is True or False.

Comparison operator	Description
>	Greater-than, returns True if the previous value of `>` is greater than the subsequent value, otherwise False.
<	Smaller-than, returns True if the previous value of `<` is smaller than the subsequent value, otherwise False.
==	Equal-to, if the values are equal on both sides of the `==`, it returns True, otherwise it returns False.
>=	Greater than or equal to (equivalent to ≥ in math) , returns True if the previous value of `>=` is greater than or equal to the subsequent value, otherwise False.

Comparison operator	Description
<=	Less than or equal to (equivalent to ≤ in math) , returns True if `<=` the preceding value is less than or equal to the following value, otherwise False.
!=	Not equal to (equal to ≠ in mathematics) , returns True if the values on both sides of the `!=` are not equal, or False otherwise.
is	Determines whether the objects referenced by two variables are the same, returns True if they are the **same**, or False otherwise.
is not	Determines whether the objects referenced by the two variables are **different**, returns True if they are different, or False otherwise.

Differences between `==` and `is`

`==` is used to compare if two variables's values are same, and `is` is used to compare if two variables are the same object.

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

Selection structrues

You can use the `if else` statement to judge conditions and then execute different code based on different results.

if

```
if condition expression:
    do something
```

```
TypeError: value is not an object
```

For example:

```
a = 10
if a>5:
    a = a*2
print(a)
```

if else

```
if conditional expression:
    do something
else:
    do some other things
```

```
TypeError: value is not an object
```

For example:

```
a = 10
if a>5:
    a = a*2
else:
    a = a-1
print(a)
```

if elif else

```
if statement1:
    do task1
elif statement2:
    do task2
elif statement3:
    do task3
...
else:
    do other things
```

```
TypeError: value is not an object
```

For example:

```
a = 7
if a>10:
    print("a>10")
elif a>5:
    print("a>5")
elif a>0:
    print("a>0")
else:
    print("a<=0")
print("a =",a)
```

Indentation

Python recognizes code blocks by indentation, and several lines of code with the same indentation belong to the same code block, so you can't indent randomly, which can easily lead to syntax errors.

Loop structrues

for

The for loop, which is often used to traverse sequence types such as strings, lists and so on, getting each element of the sequence one by one.

```
for iteration_variable in sequence:
    task
```

```
TypeError: value is not an object
```

```
result = 0
for i in range(101):
    print(i)
    result = result + i
print(result)
```

```
my_list = [6,7,8,9]
for ele in my_list:
    print('ele =', ele)
```

while

As long as the condition is true, while will execute that block of code repeatedly.
When the value of the conditional expression is judged to be True, the statement in the code block is executed, and when it is finished, the statement then looks back to determine if the value of the conditional expression is True, or if it is True, continue to re-execute the code block... This loop does not end until the conditional expression has a value of False.

```
while conditional expression:
    code block
```

```
TypeError: value is not an object
```

```
num = 1
while num <= 100 :
    print("num=", num)
    num += 1
print("loop over")
```

and/or/not