03-Python Basic

Flow Control(loop, if-else)

Posted on September 7, 2021
Last updated on January 31, 2023

3 Flow Control

Flow control describes the order in which statements will be executed at runtime.

Python has 3 types of control-flow

- `Conditional` Statements or `Decision Making` statements

```
if
if-else
if-elif-else
```

- `Iterative` Statements

```
for
while
```

- `Transfer` Statements

```
break
continue
pass
```

3.1 Conditional (Decision Making) statements

It is kind of `making decision` during occurred situation of program execution and action can be taken according to specified conditions.

Structure of decision making evaluate several expressions that provide `True or False as a result.`

It is up to you to decide which type of action want to take and execute the statements based upon True and False.

```
# if Statements
age=18
if age>=18:
    print("You are eligible to vote")
```

`if-else statement` have a block of code, where if is executed when it is 1 or True and else when if is 0 or False

```
# if-else satetements
n1, n2 = 2, 4
if n1>n2:
    print("if")
else:
    print("else")
```

The `elif statement` is used to check for the multiple True expressions and so on if the condition is True it execute a block of code.

```
# if-elif-else statement
n = 2
if n==0:
    print("if")
elif n==1:
    print("elif")
else:
    print('else')
```

`Nested if-else` statement is used to check condition inside another check condition

```
# Nested if-else statement
price = 33
if price > 30:
    if price > 100:
        print("if-if")
    else:
       print("if-else")
elif price > 20:
    if price > 15:
        print("elif-if")
    else:
       print("elif-else")
else:
    print("else")
```

3.2 Iterative Statements

In loop, the statements are `sequentially executed` , execution of first function statement is done first, then second and so on.

In situation where you need to `perform a block of code many times` then loop statement will come in to picture.

It allows to `execute/run` group of statements or a statement `multiple time` .

for loop

If we want to execute some action `for every element present in some sequence` (it may be string or collection)then we should go for for loop.

```
nlist = ['Amrit', 1, 2, 3, 4, 'apple', 'banana']
for x in nlist:
    print(x, end=', ')
# Amrit, 1, 2, 3, 4, apple, banana,

for i in range(1, 5):
    print(i, end=' ')
# 1 2 3 4

for i in range(0, 7, 2):
    print(i, end=' ')
# 0 2 4 6
```

```
# Nested Loops
for i in range(4):
    for j in range(4):
        print((i, j), end=', ')
    print()
# (0, 0), (0, 1), (0, 2), (0, 3),
# (1, 0), (1, 1), (1, 2), (1, 3),
# (2, 0), (2, 1), (2, 2), (2, 3),
# (3, 0), (3, 1), (3, 2), (3, 3)
```

while loop

If we want to `execute` a group of statements iteratively `until some condition false` , then we should go for while loop.

```
total_amount=200
while total_amount > 0:
    print(total_amount)
    total_amount=total_amount-100
else:
    print("Put more money bank people")

# 200
# 100
# Put more money bank people
```

```
# Infinite Loops
i=0;
while True:
    i=i+1;
    print("Hello", i)
```

3.3 Transfer Statements

break

It is used for a `premature terminates` of current loop immediately and executes the further statement, just like break statement in C language.

```
for i in range(10):
    if i==7:
        print("processing is enough..plz break")
        break

# processing is enough..plz break
```

continue

We can use continue statement to `skip current iteration` ithout performing remaining statements in present iteration and continue next iteration.

```
for i in range(10):
    if i%2==0:
        continue
    print(i, end=' ')
# 1 3 5 7 9
```

pass

Used to `define an empty block` with pass keyword.

The pass statement is considered as `a null statement` which `won't do anything` . It is useful when a code (i.e., code required in future) has not written till now, but your code will go eventually.

```
def m1():
    pass


for i in range(20):
    if i%9==0:
        print(i, end=' ')
    else:
        pass
# 0 9 18
```

.

.

.

.

.

.