Chapter - 5

Flow of Control

* Types of statements in Python

statements are the instructions given to the computer to perform any kind of action, be it data movements, and be it making decisions or be it repeating actions.

1. **Empty statement (Null statement)**

the simplest statement is the empty statement i.e., a statement which does nothing. In Python an empty statement is pass statement. It takes the following form:

```
pass
```

Wherever Python encounters a pass statement, Python does nothing and moves to the next statement in the flow of control.

2. **Simple statement**

Any single executable statement is a simple statement in Python. For example, following is a simple statement in Python:

```
name = input("Your name")
```

Another example:—

```
print(name)
```

3. Compound statement

A compound statement represents a group of statement executed as a unit. The compound statements of Python are written in a specific pattern as shown below:

```
<compound statement header> :
    <indented body containing multiple simple and/or
        compound statements>
```

That is, a compound statement has:

→ A header line which begins with a keyword and ends with a colon.

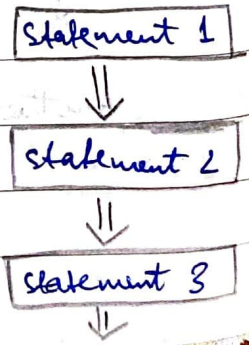
→ A body consisting of one or more Python statements, each indented inside the header line. All statements in the body are at the same level of indentation.

* Statement Flow control

In a program, statements may be executed sequentially, selectively or iteratively. Every programming language provides constructs to support sequence, selection or iteration.

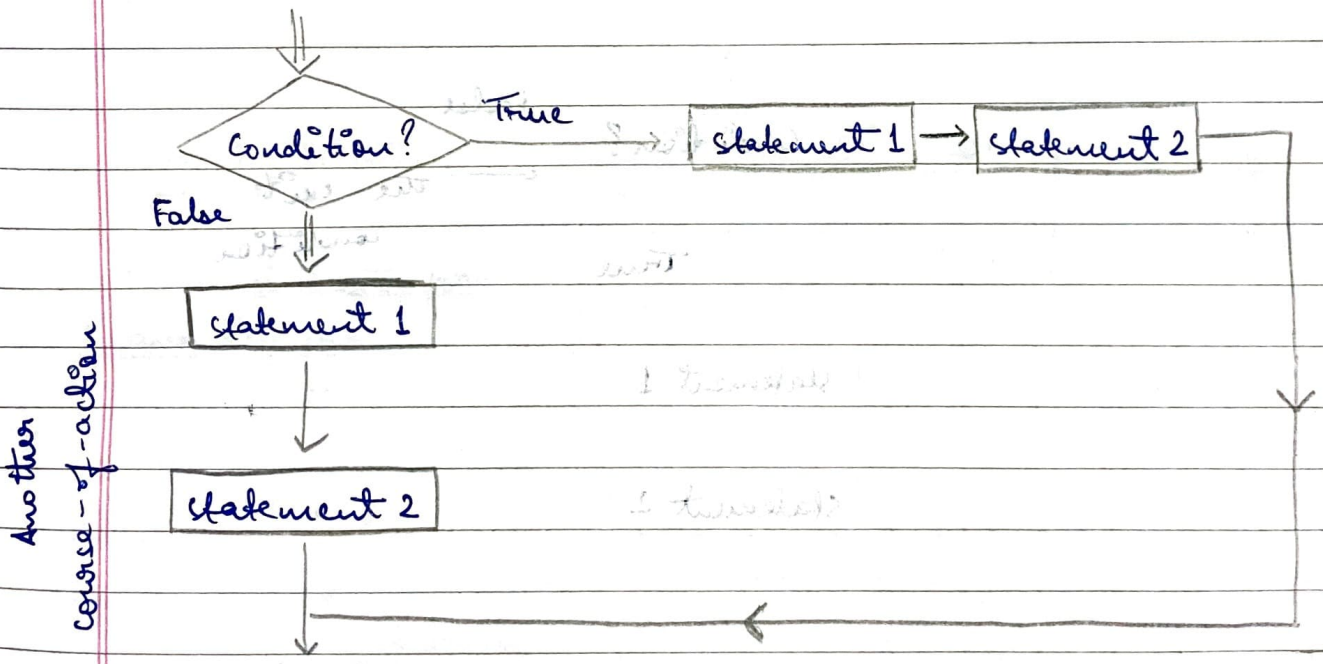
1. Sequence

The sequence construct means the statements are being executed sequentially. This represents the default flow of statement.



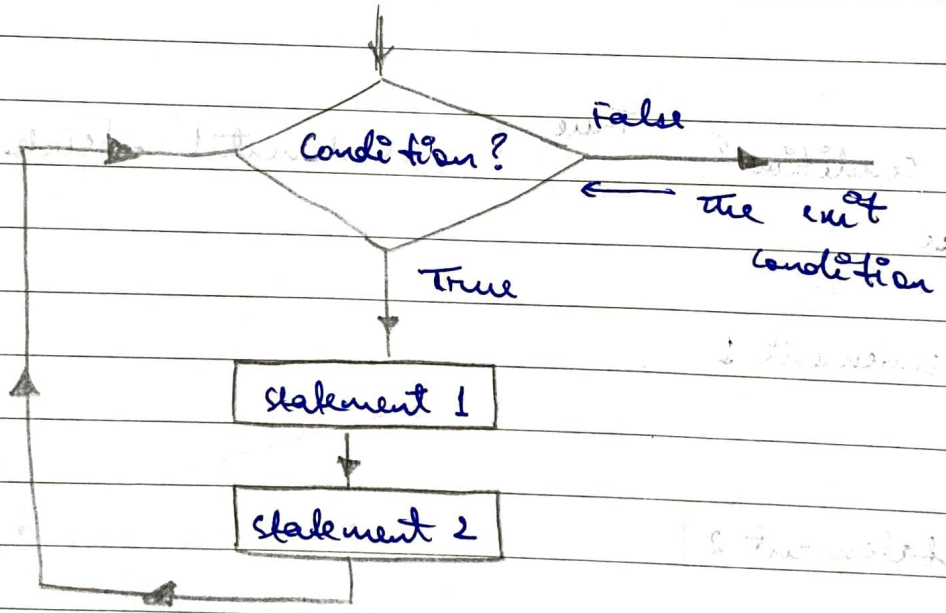
2. **Selection**

The selection construct means the execution of statement(s) depending upon a condition-test. If a condition evaluates to True, a course-of-action (a set of statements) is followed otherwise another course-of-action is followed. This construct is also called decision construct because it helps in making decision about which set-of-statements is to be executed.



3. **Repetition Iteration (Looping)**

The iteration constructs mean repetition of a set-of-statements depending upon a condition-test. Till the time a condition is True (or False depending upon the loop), a set of statements are repeated again and again. As soon as the condition becomes False (or True), the repetition stops. The iteration construct is also called looping construct.



classmate
Date
Page

* The if statements of Python

The if statements are the conditional statements in ~~Python~~ Python and these implement selection constructs (decision constructs).

An if statement tests a particular condition; if the condition evaluates to true, a course-of-action is followed i.e., a statement or set-of-statements is executed. Otherwise (if the condition evaluates to false), the course-of-action is ignored.

1. The if statement

The simplest form of if statement tests a condition and if the condition evaluates to true, it carries out some instructions and does nothing in case condition evaluates to false.

```
if <conditional expression>:
    statement
    [statements]
```

← The statements inside an if are indented at same level.

For e.g.

```
ch = input("Enter a single character: ")
if ch >= '0' and ch <= '9':
    print("You entered a digit.")
```

2. The if-else statement

This form of if statement tests a condition and if the condition evaluates to true, it carries out statements indented below if and in case condition evaluate to false, it carries out statements indented below else. The syntax (general form) of the if-else statement is as shown below :—

```
if <conditional expression>:
    statement
    [statements]
else:
    statement
    [statements]
```

For e.g.

```
a = int ( input ("Enter number:") )
if a>=0:
    print (a, "is zero or a positive number")
else:
    print (a, "is a negative number")
```

3. The if-elif statement

Syntax of if-elif and if-elif-else statements :

```
if <conditional expression>:
    statement
    [statement]
elif <conditional expression>:
    statement
    [statement]

if <conditional expression>:
    statement
    [statement]
elif <conditional expression>:
    statement
    [statement]
else:
    statement
    [statement]
```

4. The nested if statement

A nested if is an if that has another if in its if's body or in elif's body or in its ~~else~~ else's body.

Form 1 (if inside if's body)

```
if <conditional expression>:
    if <conditional expression>:
        statement(s)
    else:
        statement(s)
elif <conditional expression>:
    statement
    [statements]
else:
    statement
    [statement]
```

Form 2 (if inside elif's body)

```
if <conditional expression>:
    statement
    [statement]
elif <conditional expression>:
    if <conditional expression>:
        statement(s)
    else:
        statement(s)
else:
    statement
    [statement]
```

Form 3 (if inside else's body)

```
if <conditional expression>:
    statement
    [statements]
elif <conditional expression>:
    statement
    [statements]
else:
    if <conditional expression>:
        statement(s)
    else:
        statement(s)
```

Form 4 (if inside if's as well as else's or elif's body, i.e, multiple ifs inside)

```
if <conditional expression>:
    if <conditional expression>:
        statement(s)
    else:
        statement(s)
elif <conditional expression>:
    if <conditional expression>:
        statement(s)
```

```
else :
    statement(s)
else :
    if <conditional expression> :
        statement(s)
    else :
        statement(s)
```

* The range( ) Function

1. The range( ) function of Python generates a list which is a special sequence type.
2. A sequence in Python is a succession of values bound together by a single name.
3. Some Python sequence types are :— strings, list, tuples etc.

range(<lower limit>, <upper limit>) # both limits should be integers.

range(l, u):
will produce a list having values
l, l+1, l+2 . . . . . (u-1)
The default step value is +1.

range(<lower limit>, <upper limit>, <step value>):
All values should be integers.

→ **Operators in and not in**

1. The in operator tests if a given value is contained in a sequence or not and returns True or False accordingly.

2. Operators in and not in are also called membership operators.

* **Iteration / Looping statements**

The iteration statements or repetition statements allow a set of instructions to be performed repeatedly until a certain condition is fulfilled.

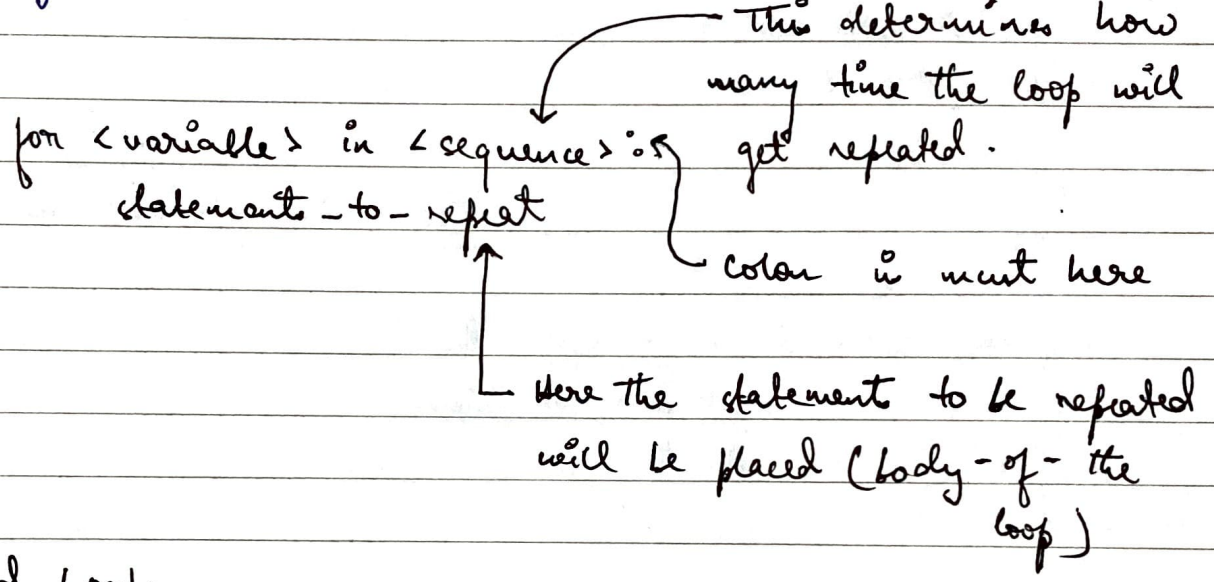
Python provides two kinds of loops: for loop and while loop to represent two categories of loops, which are:

Counting loops :— The loop that repeat a certain number of times; Python's for loop is a counting loop.

Conditional loops :— The loops that repeat until a certain thing happens i.e. they keep repeating as long as some condition is true; Python's while loop is conditional loop.

1. The for Loop

The for loop of Python is designed to process the items of any sequence, such as a list or a string, one by one.



2. Nested Loops

A loop may contain another loop in its body. This form of a loop is called nested loop.

Program to illustrate a nested for loop:

```
for i in range(1, 6):
    for j in range(1, i):
        print("*", end='')
    print()
```

Output:

```
*
**
***
****
```

3. while loop. (ENTRY CONTROL LOOP)

SYNTAX:-

```
while <condition>:
    <statements of loop>
```

EXAMPLE :-

```
name = "PYTHON"
a = 1
while a<5:
    print(name)
    a = a+1
```