Chapter 3: Compound statements

Compound statements contain one or groups of other statements; they affect or control the execution of those other statements in some way.

In general, they span multiple lines, but can be also be listed in a single line.

The `if` , `while` and `for` statements implement **traditional control flow constructs**, whereas `try` specifies **exception handlers and/or cleanup code** for a group of statements, while the `with` statement allows the execution of initialization and finalization code around a block of code. `Function` and `class` definitions are also **syntactically compound statements**.

They consists of one or more *‘clauses’*. A *clause* consists of a 'header' and a ‘suite’.

The 'clause' headers of a particular compound statement are all at the same indentation level. They should begins with an uniquely identifying keyword and should ends with a colon.

'suite' is a group of statements controlled by a clause. It can be of one or more semicolon-separated simple statements on the same line as the header, following the header’s colon (one liner), or it can be one or more indented statements on subsequent lines. Only the latter form of a suite can contain nested compound statements; the following is illegal, mostly because it wouldn’t be clear to which if clause a following else clause would belong:

Traditional Control Flow Constructs

`if` Statement

The if statement is used for conditional execution similar to that in most common languages. If statement can be constructed in three format depending on developers need.

- if
- if .. else
- if .. elif ..else

`if`

This format is used when specific operation needs to be performed if a specific condition is met. Syntax:

```
if <condition>:
       <code block>
```

Where:

- `<condition>` : sentence that can be evaluated as true or false.
- `<code block>` : sequence of command lines.
- The clauses `elif` and `else` are optional and several `elifs` for the `if` may be used but only one `else` at the end.
- Parentheses are only required to avoid ambiguity. Example:

```
x = 10
if(x > 9):
    print("Hello")
    if x > 9:
        print("Hello again")
```

```
Hello
Hello again
```

```
x = 10
y = None

if x:
    print("Hello in x")

if y:
    print("Hello in Y")
```

```
Hello in x
```

`if` ... `else` statement

```
if <condition>:
    <code block>
```

```
else:
    <code block>
```

Where:

- `<condition>` : sentence that can be evaluated as tru### `if` statement### `if` statemente or false.
- `<code block>` : sequence of command lines.
- The clauses `elif` and `else` are optional and several `elifs` for the `if` may be used but only one `else` at the end.
- Parentheses are only required to avoid ambiguity.

```
x = "mayank"
if(x == "mayank"):
    print("Name is mayank")
else:
    print("Name is not mayank")
```

```
Name is mayank
```

`if` ... `elif` ... `else` statement

```
if <condition>:
    <code block>
elif <condition>:
    <code block>
elif <condition>:
    <code block>
else:
    <code block>
```

Where:

- `<condition>` : sentence that can be evaluated as true or false.
- `<code block>` : sequence of command lines.
- The clauses `elif` and `else` are optional and several `elifs` for the `if` may be used but only one `else` at the end.
- Parentheses are only required to avoid ambiguity.

```
temp = 25 # temperature value used to test
```

```
if temp < 0:
    print ('Freezing...')
elif 0 <= temp <= 20:
    print ('Cold')
elif 21 <= temp <= 25:
    print ('Normal')
elif 26 <= temp <= 35:
    print ('Hot')
else:
    print ('Very Hot!')
```

```
Normal
```

Imagine that in the above program, `23` is the temperature which was read by some sensor or manually entered by the user and `Normal` is the response of the program.

If the code block is composed of only one line, it can be written after the colon:

```
if temp < 0: print 'Freezing...'
```

Since version 2.5, Python supports the expression:

```
<variable> = <value 1> if <condition> else <value 2>
```

Where `<variable>` receives `<value 1>` if `<condition>` is true and `<value 2>` otherwise.

> ... continue in next chapter

```
x = 20

if x > 10: print ("Hello ")

print("-"*30)

val = 1 if x < 10 else 24
print(val)
```

```
Hello
------------------------------
24
```