6. Conditional statements

Overview:

- Concept of **`if` control structures** i.e. compound statements that alter program control flow;
- How to group individual statements together into a **block**;
- One-line **`if`/`else`/`elif` statements**;
- The **`if`/`else` conditional expression** (ternary operator), which makes it possible to conditionally execute a statement based on evaluation of program data;
- Ternary **`if`/`else` expressions can be chained** with parentheses as alternative to **`if`/`elif`** construction;
- **`pass`** placeholder.

1 Introduction to the if-statement

If construction in its simplest form:

```
if <expr>:
   <statement>
```

<expr> is an expression evaluated in Boolean context. <statement> is a valid Python statement, which must be indented.
If <expr> is `True`, then <statement> is executed. If <expr> is `False`, then <statement> is skipped over. Note that the **colon :** following <expr> **is required**.

Example:

```
if 'aul' in 'grault': # True
   print('yes')
```

2 Grouping statements: identations and blocks

Say we need to evaluate a condition and then do more than one thing if it is true. Python follows a convention known as the **off-side rule**. I.e. in a Python program, contiguous **statements that are indented to the same level are considered to be part of the same block**.

Compound if-statement in Python looks like this:

```
if <expr>:
    <statement>
    <statement>
     ...
    <statement>

<following_statement>
```

3 The else and elif clauses

`else`: sometimes, we want to evaluate a condition and take one path if it is true but specify an **alternative path** if it is not:

```
if <expr>:
    <statement(s)>
else:
    <statement(s)>
```

There is also syntax for branching execution based on several alternatives. For this, use one or more **elif** <expr>: (short for "else if") clauses:

```
if <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
elif <expr>:
    <statement(s)>
    ...
else:
    <statement(s)>
```

Note: Using a lengthy **if/elif/else** series can be a little inelegant, especially when the actions are simple statements like **print()**. There may be a more Pythonic way to do (by **dict.get()** method of a dictionary):

```
names = {'Fred': 'Hello Fred',
         'Xander': 'Hello Xander',
         'Joe': 'Hello Joe',
         'Arnold': 'Hello Arnold'}

print(names.get('Joe', "I don't know who you are!"))
>> Hello Joe
```

4 One-line if, elif, else statements

it is permissible to write an **entire if statement on one line** although it is **discouraged by PEP8 except if the expression is short and simple enough**:

```
if <expr>: <statement>
```

Or even:

```
if <expr>: <statement_1>; <statement_2>; ...; <statement_n>
```

In this case: if <expr> is True, execute all of <statement_1> ... <statement_n>. Otherwise, don't execute any of them. Multiple statements may be specified on the same line as an **elif** or else clause as well:

```
x = 2
if x == 1:   print('foo'); print('bar'); print('baz')
elif x == 2: print('qux'); print('quux')
else: print('corge'); print('grault')

>> qux
>> quux
```

5 Conditional expression (ternary operator)

Python supports one additional decision-making entity called a **conditional expression**. In its simplest form, the syntax of the conditional expression is as follows:

```
<expr1> if <conditional_expr> else <expr2>
```

This is different from the if statement forms listed above because **it is not a control structure that directs the flow of program execution.**

It acts as an operator that defines an expression.

In the above example, <conditional_expr> is evaluated first. If it is True, the expression evaluates to <expr1>. If it is False, the expression evaluates to <expr2>.

Example:

```
age = 12
s = 'minor' if age < 21 else 'adult'
>> s = 'minor'
```

Another example:

```
raining = True
print("Let's go to the", 'beach' if not raining else 'library')
>> Let's go to the library
```

It can be used as part of a longer expression. The conditional expression has lower precedence than virtually all the other operators, so **parentheses are needed to group it by itself**. If you want the conditional expression to be evaluated first, you need to surround it with grouping parentheses:

```
x = y = 40
z = 1 + (x if x > y else y) + 2
>> z = 43
```

Another example, contrasting the previous one:

```
z = (1 + x) if x > y else (y + 2)
>> z = 42
```

If you are using a conditional expression as part of a larger expression, it probably is a **good idea to use grouping parentheses for clarification** even if they are not needed.

Conditional expressions can also be chained together, as a sort of alternative **if/elif/else** structure, as shown here:

```
s = ('foo' if (x == 1) else
     'bar' if (x == 2) else
     'baz' if (x == 3) else
     'qux' if (x == 4) else
     'quux')
```

The pass statement

Occasionally, you want to write what is called a code stub: a **placeholder where you will put a block of code that you haven't implemented yet**. The Python pass statement solves this problem. It doesn't change program behavior at all.

```
if True:
    pass # I don't know what to do here yet

print('foo')
> foo
```