STEP IT ACADEMY

Programming with python

Lesson # 4

if Statement and Arithmetic Operations

CONTENTS

Conditional Operator

In your daily life and in games you face conditions. An example is the situation in a game, when the player has to make a choice: if more than 500 mysterious stones are collected, you can open a bonus level.

Another example: your character in the game is a warrior. A warrior's weapon is a sword. Such conditions are very easy to verify by an ordinary question, the answer to which can be yes or no, true or false (Figure 1). ***Are you a warrior? – Yes. – Get a sword***.

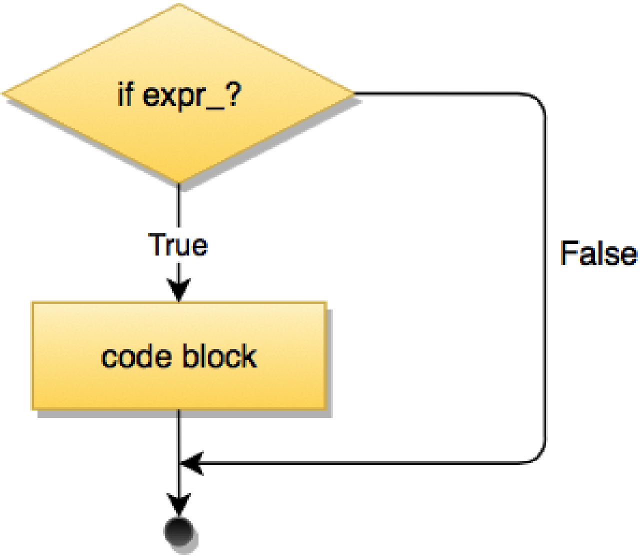

Figure 1

The condition is formulated in this way: first `if` is written, then a statement in parentheses, which the program verifies. If the statement in parentheses is `true`, then the code block after the colon is executed, if `false`, this section of code is skipped.

Comparison Operators

To work with conditions, **comparison operators are required: operators that answer either yes or no**, for example: *Do you have classes today at the academy?* – *Yes* – you go to a class, – *No* – you do other things.

Basic logic operators are given below on the simplest examples with numbers. Accordingly, `true` means that the result of the comparison is correct, and `false` means incorrect.

Operation	Name	Example	Result
>	Greater than	`print(5>2)`	`true`
<	Less than	`print(5<2)`	`false`
>=	Greater than/ equal to	`print(5>=2)`	`true`
<=	Less than/ equal to	`print(5<=2)`	`false`
==	Equal to	`print(5==2)`	`false`
		`print(5==5)`	`true`
!=	Not equal to	`print(5!=2)`	`true`

Among all the operators listed, you may have never encountered `!=` before. This is **"not equal"** and it asks the computer: ***Is it true that five is not equal to two?*** The answer is ***Yes, this is not true***, so the result of such an operation is `true`.

if–else Statement

The first logical statement is `if` (*if*) and it is fairly easy to use. Simply put, it results in `true` or `false`. If true, the code is executed in the body of the loop, i. e. after the colon. And so it repeats until the expression in brackets after the `if` statement is true.

The syntax of the conditional `if` statement is as follows:

```
if logical-statement:
   statement 1
   statement 2
   ...
```

A single `if` statement is not always enough. For such cases, there is an `else` block, which allows you to create another branch and execute another code if the expression after the `if` statement is not executed.

The `if-else` **syntax** is as follows:

```
if logical-statement:
  statement 1
  statement 2
  ...
else:
  statement 3
  ...
```

And now let's write a code to implement these statements and print the result.

```
# condition if-else
a = 5
b = 6
if (a < b):
 print("a severely b")
else:
 print("a strictly b")
```

This program uses the conditional statement `if-else`. After the word `if`, a logical expression (`a < b`) is specified. After this, there is a **block (sequence)** of instructions that will be executed if the condition is true.

Then goes `else` (*otherwise*) and the instruction block that will be executed if the condition is not true.

We will write a small program so that the user can enter a number from 0 to 24, and the program outputs whether it is day or night.

We have earlier discussed the principle of working with input/output functions `print()` and `input()`. And now let's merge the `if-else` conditional statements and the input/output functions in our code.

```
# time
time = input("Enter the time: ")
if time < 12:
   print("The first half of the day!")
else:
   print("Afternoon!")
```

Great! Now you know how to use conditional statements.

Lesson # 4

Adding math and random Modules

To write the following programs, you'll need additional `math` and `random` modules. As you already know from the previous lessons, a **module is a collection of functions or variables gathered together**.

In the previous lessons, we used the `turtle` module for drawing. As you remember, first we imported the module and after this we wrote the code.

As you can see, Python has a huge number of useful modules for solving a variety of tasks.

The math Module

We will always refer to the official documentation. **The** [`math module`] **allows you to implement many mathematical calculations**, such as rounding, raising to the power, taking the root of a number:

```
import math
print(math.ceil(2.5)) # ceil(2.5) ≈ 3
print(math.floor(2.5)) # floor(2.5) ≈ 2
print(math.pow(2, 4)) # pow(2, 4) =
                                  2*2*2*2 = 16.0
print(math.sqrt(16)) # sqrt(16) = 4*4 = 4.0
```

In the `math` module, the `math.ceil(x)`and `math.floor(x)` functions round, the first one rounds up, and the second rounds down. The `math.pow(x, y)` function raises to the power $x^y = x_1 * x_2 * ... * x_y$, and `math.sqrt(x)` takes the root $\sqrt{x}$, i. e. which number is to be raised to the square to get `x`.

The random Module

The `random module` **allows you to generate random numbers**. This is like if you ask a friend to think of any number, for example, from 0 to 10 or from 100 to 200.

Let's look at an example of two functions that allow you to get `int` and `float` numbers:

```
import random

print(random.random()) # 0.4956579385740163
print(random.randint(0, 1)) # 0
print(random.randint(1, 100)) # 66
print(random.randint(10, 20)) # 16
```

In the `random` module, the same-name `random()` function generates floating-point numbers (`float`) within the interval [0.0, 1.0). To use it, specify the module name and the function name `random.random()`. In turn, `randint()` generates integers (`int`) within `a <= N <= b`, where `N` is a random number.

As you can see, this is a very useful library. We will use it every time we need to simulate rolling a dice or tossing a coin because our task is to make the result completely random.

Mathematical Operations

Do not forget that there is a number of mathematical operations, which we will also need to solve a number of problems. Such operations include summation, subtraction, multiplication, division, and raising to the power.

> *Note that in the case of summation and subtraction, you can write them in different ways but they will have the same meaning.*

Summation:

```
a = 2
a += 1 # a = a+1
print(a)
```

Subtraction:

```
a = 2
a -= 1 # a = a-1
print(a)
```

In the examples considered, the combination of plus and equal sign `a += 1` is similar to the addition operation which is written in the comments `a = a+1`. The same goes for subtraction. Instead of +=, try execute these operations:

- multiplication `a *= 2`;
- division `a /= 2`, `a //= 2`, `a %= 2`;
- raising to the power `a **= 2`.

Raising to the power, as you already know, can also be implemented in several ways:

```
import math
a = 2
a = math.pow(a, 2) # a = a**2
print(a)
```

Here you can use the `**` operator or the `math.pow(x, y)` function from the `math` library.

Solving Math Problems with Python

It's cool when you can write a program on your own that would solve math problems.

Convert US Customary Units to Metric Ones

American books and films use the concept of ***"mile"*** when it comes to distance. Mile is one of the units of length. In addition, when it comes to the car speed (Figure 2), you can often hear the expression ***"miles per hour"***. One mile is equal to 1.609 km.

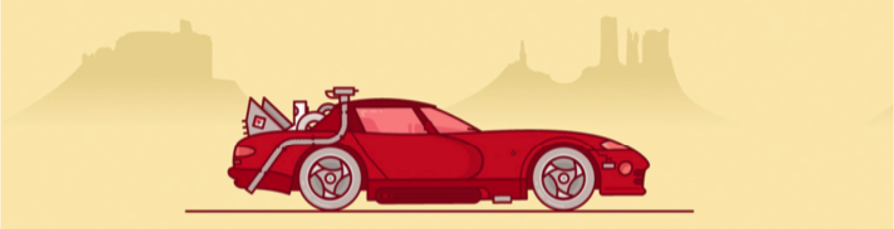

Figure 2

Let’s write a program that converts miles to kilometers:

```
# miles to km
n = input("Enter miles: ")
mile = 1.609
km = float(n) * mile
print(f"{n} mile = {km} km")
if km > 80:
 print("Too fast!")
```

Let’s solve a problem.

The car speedometer shows the speed in miles per hour. How many kilometers per hour does the speedometer display if the car is going 43 miles per hour?

To do this, we need to write 43 in the code and get the result.

Answer: 69.187 km.

Purchase Calculator

Imagine that you came to a store that sells magic lollipops for 270 coins per 1 kg. The store has a special offer: if you buy more than 500 g of lollipops, their cost is equal to 200 coins per 1 kg (Figure 3).

Let’s write a program that can count the purchase amount and convert grams to kilograms; 1 kilogram is equal to 1000 grams.

Figure 3

This program will solve this problem:

```
# calculate
import math

price_1 = 270
price_2 = 200
gramm = int(input("Enter weight (gramm): "))
kg = gramm / 1000
print("Your weight: {} kg".format(kg))
if (kg<0.5):
 total = kg * price_1
else:
 total = kg * price_2
total = math.ceil(total)
print("Your total: {} coin".format(total))
```

In this program, we used the `math` library, which allows you to round the final amount here.

Imagine that you entered the value:

`Enter weight (gramm): 10001`,

then the result of the code execution will be:

`Your total: 2000.1999999999998 coin`.

You can check for yourself, to do this, comment out the line `# total = math.ceil(total)` and run the code.

Let's return to our program. To prevent this, we added the `math` library and the `math.ceil(total)` function.

Generate Random Numbers

Sometimes programs require random numbers, for example, if we simulate rolling a dice or tossing a coin. In such cases, one should randomly generate a number from 1 to 6 (*roll the dice*) or 0 and 1 (*heads and tails, toss thecoins*) (Figure 4).

For this we will use the already familiar `random` module.

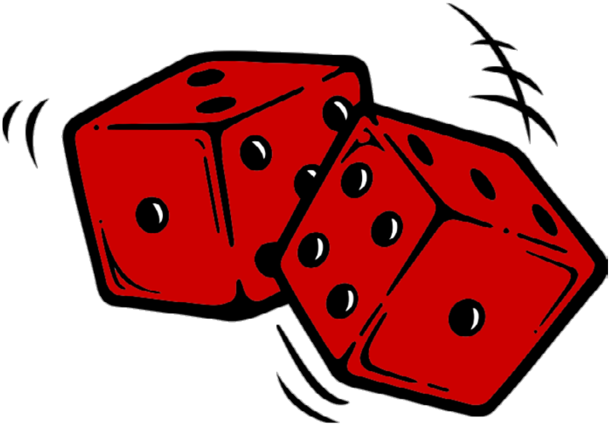

Figure 4

Let’s write a program where the computer will generate a random number and display it on the screen. Let the numbers displayed by the computer be in the range from 1 to 6 (*as on a dice*).

Consider this situation: number 6 on the dice is win. How many times will you see 6 if you roll the dice 20 times?

This problem will be solved by the following program:

```
# random numbers
import random
number = 0
for i in range(20):
 # random number from 1 to 6
 magic_number = random.randint(1, 6)
 #print("{}. The magic number is:
             {}" .format(i, magic_number))
 if magic_number == 6:
  number = number + 1
print("You got number 6: {} times"
                  .format(number))
```

We imported the `random` module to our program. We created the number variable and made it equal to 0 in the second line. Then we used the `for` loop, which we worked with in the previous lesson. The loop allows you to perform the same action a certain number of times. The `range(20)` function sets the range of numbers from 0 to 19, i. e. the code in the body of the loop (after `for`) will be repeated 20 times.

> *Note, if we did not specify the first number of the range, the program will count not from 1, as we used to, but from 0.*

Then a `magic_number` variable was created that contains a random number. Random number is generated by the `random.randint(1, 6)` function, it can be equal to 1,2, ..., 5,6.

You can remove comments from this line *`#print("{}. The magic number is: {}" .format(count, magic_number))`* by removing the #sign to display the generated number in the console.

Add 1 to the variable `number = number + 1` every time when we see 6 on the dice. Thus, we'll calculate how many times we get six.

Lesson # 4

if Statement and Arithmetic Operations