Python → Simple programs → Program with numbers

Theory: Program with numbers

31 minutes 6 / 15 problems solved

Start practicing

19171 users solved this topic. Latest completion was **25 minutes ago**.

Programs in which there's nothing to calculate are quite rare. Therefore, learning to program with numbers is never a bad idea. An even more valuable skill we are about to learn is the processing of user data. With its help, you can create interactive and by far more flexible applications. So let's get started!

§1. Reading numbers from user input

Since you have become familiar with the `input()` function in Python, it's hardly new to you that any data passed to this function is treated as a **string**. But how should we deal with numerical values? As a general rule, they are explicitly converted to corresponding numerical types:

```
integer = int(input())
floating_point = float(input())
```

Pay attention to current best practices: it's crucial not to name your variables as built-in types (say, **float** or **int**). Yet another caveat is related to user mistakes. If a user writes an inaccurate input, `ValueError` will occur. At the moment, we'll limit ourselves to this. But not to worry, more information about errors is available in a dedicated topic. Now, consider a more detailed and pragmatic example of handling numerical inputs.

§2. Free air miles

Imagine you have a credit card with a free air miles bonus program (or maybe you already have one). As a user, you are expected to input the amount of money you spend on average from this card per month. Let's assume that the bonus program gives you 2 free air miles for every dollar you spend. Here's a simple program to figure out when you can travel somewhere for free:

```
# the average amount of money per month
money = int(input("How much money do you spend per month: "))

# the number of miles per unit of money
n_miles = 2

# earned miles
miles_per_month = money * n_miles

# the distance between London and Paris
distance = 215

# how many months do you need to get
# a free trip from London to Paris and back
print(distance * 2 / miles_per_month)
```

This program will calculate how many months you need to travel the selected distance and back.

Although it is recommended to write messages for users in the `input()` function, avoid them in our educational programming challenges, otherwise your code may not pass our tests.

§3. Advanced forms of assignment

Current topic:

✓ Program with numbers Stage 1 17★ ···

Topic depends on:

✓ Integer arithmetic Stage 1 17★ ···

✓ Naming variables Stage 1 17★ ···

✓ Taking input Stage 1 17★ ···

Topic is required for:

✓ Comparisons Stage 1 16★ ···

Math functions ···

✓ Class vs instance 3★ ···

Table of contents:

Whenever you use an equal sign `=`, you actually assign some value to a name. For that reason, `=` is typically referred to as an **assignment operator**. Meanwhile, there are other assignment operators you can use in Python. They are also called **compound assignment operators**, for they carry out an arithmetic operation and assignment in one step. Have a look at the code snippet below:

```
# simple assignment
number = 10
number = number + 1  # 11
```

This code is equivalent to the following one:

```
# compound assignment
number = 10
number += 1  # 11
```

One can clearly see from the example that the second piece of code is more concise (for it doesn't repeat the variable's name).

Naturally, similar assignment forms exist for the rest of arithmetic operations: `-=`, `*=`, `/=`, `//=`, `%=`, `**=`. Given the opportunity, use them to save time and effort.

One possible application of compound assignment comes next.

§4. Counter variable

In programming, loops are used alongside special variables called **counters**. A **counter** counts how many times a particular code is run. It also follows that counters should be integers. Now we are getting to the point: you can use the operators `+=` and `-=` to increase or decrease the counter respectively.

Consider this example where a user determines the value by which the counter is increased:

```
counter = 1
step = int(input())  # let it be 3
counter += step
print(counter)  # it should be 4, then
```

In case you need only non-negative integers from the user (we are increasing the counter after all!), you can prevent incorrect inputs by using the `abs()` function. It is a Python built-in function that returns the absolute value of a number (that is, value regardless of its sign). Let's readjust our last program a bit:

```
counter = 1
step = abs(int(input()))  # user types -3
counter += step
print(counter)  # it's still 4
```

As you can see, thanks to the `abs()` function we got a positive number.

For now, it's all right that you do not know much about the mentioned details of **errors**, **loops** and **built-in functions** in Python. We will catch up and make sure that you know these topics comprehensively. Keep learning!

Thus, we have shed some light on new details about integer arithmetic and the processing of numerical inputs in Python. Feel free to use them in your future projects.

Report a typo

1503 users liked this theory. **32** didn't like it. **What about you?**

😍 🙂 😐 🙁 😡

Start practicing

Comments (5) Hints (0) Useful links (0) Show discussion