OpenLearn

The Open University

Simple coding



This item contains selected online content. It is for use alongside, not as a replacement for the module website, which is the primary study format and contains activities and resources that cannot be replicated in the printed versions.

Unless otherwise stated, copyright © 2015, 2015 The Open University, all rights reserved.

Contents

1 Introduction 3
2 Sequence 4
2.1 The art of failure 4
3 Selection 6
3.1 And now for something not completely different 7
4 Iteration (part 1) 9
5 Functions 10
5.1 The input function 10
5.2 A conversion function 11
6 Iteration (part 2) 12
6.1 The oldest algorithm 13
Summary 14

1 Introduction

Software makes the world go round. Cars and TVs have software that controls how they work, and global commerce and finance are impossible without software that control the stocks, carry out payments, find the best transport route, etc.

Coding (or programming) is the construction of software. Coding involves writing a 'recipe', which in computing is called an **algorithm**, in a so called **programming language** that a computer can understand. When the computer **runs** the code we wrote, it follows the 'recipe', step by step.

We will use Python, a popular programming language for teaching and for professional software development. You will see that Python code reads almost like plain English. Writing simple programs in Python is not very difficult, once you have come up with the 'recipe', the algorithm.

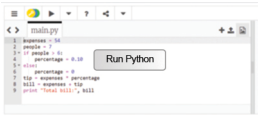
This tutorial shows how to code in Python the basic building blocks of all algorithms, and how to ask the user for input and produce some output on the screen. The skills, concepts and jargon (in boldface) you will learn are the foundations for coding all sorts of apps in all sorts of languages, not just Python.

You won't have to install any software, the code examples are embedded in this course and you can execute and change them right here.

> Wherever 'Run Python' appears we recommend that you open the link in a new tab and work from there.

Go ahead, click 'Run' (the ► icon) to execute the following program (I'll explain it later). You should see a message showing the output of running the code.

Interactive content is not available in this format.



You can also change the code. Click with the mouse next to the 7 on the first line, and use the keyboard to replace it by 2 (or another number of your choice). Also replace the 7 in the second line by the same number. Click again on 'Run'. The message should now read "The sum of 3 and 2 is 5", or similar.

In the rest of this tutorial I won't repeat that after you change a program you need to run it to see what it produces.

I will show you various small programs with exercises for you to practice your coding skills, by modifying or building upon my programs. You can share your code with friends and family by clicking on the down arrow next to 'Share'.

If, whilst practicing, you do any mistakes, you can always undo all your changes. Click on the 3 lines icon in the top left corner. A menu appears. Click on 'Reset' and confirm you want to put back the code as it was, before you changed the numbers.

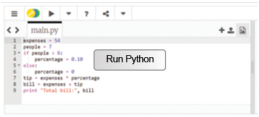
All done? Good. In the next section we will start learning to code.

2 Sequence

The simplest Python program is a **sequence** of **instructions**, written one per line, and executed one by one from top to bottom. Our first program only has two instructions. Here is the program again, already run for you. (You may just see the result of running the code. To see the code itself, click on the pencil icon.)

Remember to open the link to Python in a new tab.

Interactive content is not available in this format.



The first instruction is an **assignment**: the computer evaluates the **expression** to the right of the assignment (=) and stores the result in the **variable** on the left of the assignment. Each piece of data we need has to be stored in a variable. Translating Python to English, the assignment states ‘let result be the sum of 3 and 7’.

The second instruction, print, prints some text on the screen, followed by the computed result. Note the following:

- the comma separates the two things to be printed, in the same way we use commas in English to enumerate two, three, or more things;
- text is written between double-quotes, which are not printed themselves. In Python, a sequence of characters surrounded by double-quotes is called a **string**.

2.1 The art of failure

The following details are important once you start writing code. Computers are not as smart and accommodating as human readers: at the slightest spelling mistake (like pint instead of print) or missing punctuation (forgetting the comma or double-quotes, for example), the computer will give up on understanding and running your code and will display an error message.

On top of that, most error messages are rather cryptic. Hence it’s best to get used to them, by doing errors on purpose, so that you have a clue what might be wrong when you are writing your own code.

Activity 1

Put double-quotes around the 7 on the first line and run the program. You will get an error message saying that there is a **type error**. This basically means the code is mixing data of different types, mixing apples and oranges if you like. In this case, we are trying to add a number (3) to a string (`"7"`) which of course doesn’t make sense. Click on the X at the right end of the message to make it go away and then reset the program to undo the change.

Sometimes, code has more subtle errors, that lead to unexpected results. This is akin to miscommunication, when other people understand something different from what you intended to say. Here is an example.

Activity 2

Put double-quotes around the variable name in the second line. Run the program. Try to explain what happened before reading further.

When putting double-quotes around the variable name, it becomes a string. Hence, the computer will print the name literally, instead of printing the value stored in the variable.

To sum up, `"result"`, `"7"`, `result` and `7` are different things: the first two are strings (literal text), the third is a variable, and the last is a number.

Learning from mistakes is always helpful, so for this and the remaining programs in this tutorial, I encourage you to introduce errors and observe what message or output you get. Don't forget to reset the program after each error.

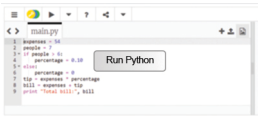
In a sequence, *all* instructions are executed. Next you will learn how to selectively execute some instructions but not others.

3 Selection

Further investigate the Python programming language by learning about conditions.

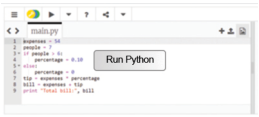
Let's imagine we are developing software for a restaurant, where a tip of 10% is added to the bill. Here's the code, a simple sequence of assignments and output. For example, the assignment on line 3 states 'let the tip be the expenses multiplied by the percentage'. Note that in Python the **multiplication operator** is the asterisk.

Interactive content is not available in this format.



Let's further assume the tip is only automatically added for groups above 6 people. We need one more variable, to store the number of people in the group, and one new instruction to handle both cases: *if* the number of people is more than 6, the percentage is 10%, *otherwise* it's 0%.

Interactive content is not available in this format.



Before I explain the code, let's **test** it, to make sure it works as intended. Large software companies employ many testers to check their code. Good testing includes choosing enough inputs (preferably borderline cases) to exercise all possible conditions. In this case, we should at least test for 6 and 7 people, the borderlines where the tip percentage changes.

Activity 3

Change the number of people to 6 and confirm the bill is just the expenses in that case.

Notice there is a colon (`:`) at the end of the `if` and `else` lines. Forgetting the colons and forgetting to indent the instructions will lead to error messages.

The **selection** instruction '`if` condition: block `else`: block' chooses which block to execute as follows. The computer checks the condition after the if. If it is true, the computer then executes the **block** of code (the indented instructions) belonging to the if part. If the condition is false, the computer executes *instead* the `else` block. The indentation is needed to know which instructions belong to which part. Afterwards the computer continues executing the non-indented instructions.

3.1 And now for something not completely different

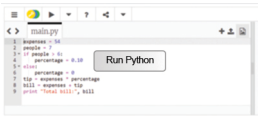
Usually there are many ways to solve the same problem. For the problem of adding the 10% tip only to groups over 6 people, we have set the tip rate to 0% or 10% or the tip itself but we can instead change how the tip is calculated.

Activity 4

Keep the rate at 10% but change the way the tip is computed: if there are more than 6 people, let the tip be the expenses times the rate, otherwise let the tip be zero. Change the code accordingly and check your solution against mine .

Finally, let's assume the restaurant decides to impose a higher tip of 15% for groups of at least 15 people. In Python we can chain various conditions as follows:

Interactive content is not available in this format.



In plain English: if the number of people is larger or equal (>=) to 15, let the percentage be 15%, otherwise if (`elif`, *not* `else if`) it is larger than 6 let the percentage be 10%, otherwise let the percentage be 0%.

Activity 5

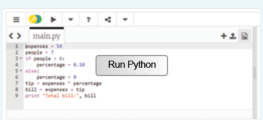
Test the code with the four borderline cases: 6, 7, 14 and 15 people.

Note that the order in which we write the conditions is important, because the computer checks them from top to bottom and executes only *one* block, for the *first* condition that is true. The `else` block has no condition, so it's a 'catch all' in case no condition is true.

Activity 6

Explain why the following code is wrong. Hint: repeat the tests and see what happens. That's what tests are for.

Interactive content is not available in this format.



4 Iteration (part 1)

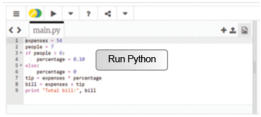
In this section you will learn about repetition in the Python programming language.

A restaurant bill is made from the various food and drink items ordered. Our program should sum those prices to obtain the expenses, on which the tip will then be added. The algorithm (which is independent of any programming language) is as follows.

1. Let items be a list of the prices of the food and drinks ordered.
2. Let the expenses be zero.
3. For each item in the items list:
    - add it to the expenses.
4. print the expenses

Step 3 is called an **iteration**: it goes through the list to process each item. This can be directly translated to Python.

Interactive content is not available in this format.



In Python, any line starting with # is a **comment**. The computer ignores comments, they are just notes by the code's author to clarify any finer points. Such notes come in very handy if the author (or someone else) needs to change the code later.

In Python, **lists** are simply comma-separated values, within square brackets.

The 'for variable in list: block' instruction goes through the given list and successively stores each value of the list in the variable and then executes the block, which will usually (but not always, see the exercise below) refer to the variable to get its value. In this case, I add the value of the item to the current expenses and store the result again in expenses, so that it is always up to date. In English: let the new expenses be the old expenses plus the item ordered.

Activity 7

Calculate and print how many items were ordered, i.e. 5 in the example above. First change the algorithm, then translate it to code. Hint: you need to count the items in the list, their prices are irrelevant. If you get stuck, you can see the algorithm .

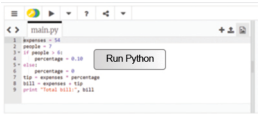
5 Functions

In this section you will learn about functions in the Python programming language.

Totting up the bill is basically calculating the sum of a list of numbers. This is such a common need that Python already provides a **function** for that, appropriately called `sum`. A function takes some data (the function's input) and returns some other data (the function's output). In this case, the `sum` function takes a list of numbers as input and returns a single number as output.

To apply a function to some data (whether it's a variable, a number or a string), just write the name of the function followed by the data in parenthesis. The computer will calculate the function's output, which can be used in further calculations or assigned to a variable.

Interactive content is not available in this format.



As you can see, using the `sum` function shortens our code and makes it easier to understand, because the function's name explicitly states what the code is doing. Good programmers don't just write code, they write *readable* code. They know that code always changes to accommodate further customer requests, and trying to modify cryptic code you wrote some weeks or months ago is no fun. Using comments and descriptive names for variables and functions helps making code readable.

The length function

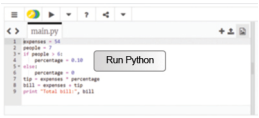
Counting the number of items in a list is also so common, that Python has a function for that too, called `len`, which returns the length of the list.

Change the code above to show the number of ordered items, using the `len` function.

5.1 The input function

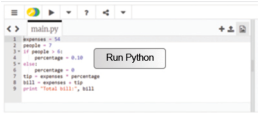
Another very useful function is `input`: it takes a string that it prints on the screen (the user's prompt), and returns a string with what the user typed on the keyboard until they pressed the RETURN or ENTER key. Run the code below and type in your name, followed by RETURN or ENTER.

Interactive content is not available in this format.



Note that `input` always returns a string, even if the user typed in a number. To see the implications of that, run the code below and type in your birth year.

Interactive content is not available in this format.

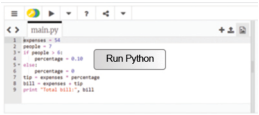


The computer reports a type error, because the code is mixing two different types of data. It is trying to subtract the string returned by `input` from the number 2015.

5.2 A conversion function

The solution is to use the built-in function `int` that takes a string and converts it to an **integer**, which is a number like –2, –1, 0, 1, 2, etc. If the string represents a decimal number, `int` will give an error.

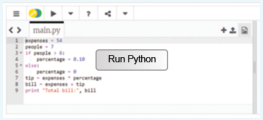
Interactive content is not available in this format.



Activity 8

Change the restaurant program (repeated below) so that it asks the user for the number of people.

Interactive content is not available in this format.



Next, in Section 6, you'll learn a different form of iteration.

6 Iteration (part 2)

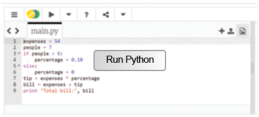
In this section you will learn more about iteration and repetition in the Python programming language.

Let's use input to ask for the price of orders instead of fixing them in a list. This requires a new iteration: an infinite loop that keeps asking until the user types 'stop', for example. We also need to convert the string returned by input into a number we can add to the expenses. The algorithm is:

1. let the expenses be zero
2. forever repeat the following:
    1. let answer be the reply to the question "Price of ordered item?"
    2. if the answer is equal to "stop":
       exit the loop
    3. otherwise:
    i. let the price be the answer converted to a decimal number
    ii. add the price to the expenses

Test the code below: provide the same values as before (4.35, 2, 19.95, 22.70 and 5), followed by stop, and check the sum is at the end 54.

Interactive content is not available in this format.



In Python, the 'while condition: block' instruction keeps executing the block while the condition is true. In this case the condition is always True (note the initial uppercase), which leads to an infinite loop. Doing forever the same thing can get a bit tedious, so at some point I have to break free with the appropriately named break instruction. At that point the computer starts executing whatever code follows the loop.

Note that food and drink prices may be in pence or cents, so I need to convert the user's input string to a decimal number (called a **floating point** number in Python), using the function `float` instead of `int`.

Note also that to check if two values or variables are equal, we write two equal signs, because a single equal sign is the assignment instruction.

Activity 9

Put a complete program together, that starts as above, then asks the user for the number of people in the group, computes the tip and the VAT (20% on the expenses) and finally prints the total bill.

6.1 The oldest algorithm

The while loop condition doesn't have to be always true, of course. As an example, here is one of the oldest algorithms known, Euclid's algorithm, used to find the greatest common divisor of two integers greater than zero, i.e. the greatest number that divides both without any remainder. For example, the greatest common divisor of 3 and 7 is 1, and of 21 and 49 is 7. This is needed to simplify fractions, for example. Explaining the maths that make the algorithm work is beyond the scope of this tutorial.

Activity 10

Translate the algorithm below to Python. One instruction was already translated for you. Don't forget to test your code.

Interactive content is not available in this format.



Note that for the loop to end, the condition must become false, i.e. both variables n and m have the same value. Therefore it doesn't matter which one you print.

Summary

In this brief introduction to programming you have seen the fundamental building blocks that almost every programming language provides:

- variables and assignment (=) to store and update data
- arithmetic operators (+, -, *)
- simple data types (strings and numbers, both integers and floating point)
- data structures (lists)
- sequence (one instruction per line)
- iteration (for and while loops)
- selection (if-else and if-elif-elif-...-else),
- comparisons (>, <, ==, !=, >=, <=)
- functions (len and sum for lists, int and float to convert strings to numbers)
- input from the keyboard (input function)
- output to the screen (print instruction)

Programming languages have to be automatically understood by a machine, so the syntax and grammar are much more constrained than English and other natural languages. Any spelling mistake, like writing flat instead of float or forgetting punctuation like commas and colons, or using the wrong data type, leads to an error.

You have also seen that programming involves writing clear and understandable code (e.g. by using comments and plain English names for variables and functions) to make it easier to change later, and testing it thoroughly.

Learning to program forces us to think clearly and rigorously when solving a problem, because the solution has to be described in very small and precise steps that even a machine can understand. Python makes it easy to write the code once we come up with a sufficiently detailed algorithm, but the thinking (still) has to be done by us.

Like learning any other skill, only practice makes perfect. We hope this course inspired you to learn more Python and to share your creations with friends and family.