

Binary/decimal conversion

CONTENTS

[Summary](#)[Learning objectives](#)[Before you start](#)[Exercise 1: binary string I](#)[Exercise 2: binary string II](#)[Exercise 3: binary string III](#)[Exercise 4: convert a decimal number to a binary number](#)[Exercise 5: Comment all your code](#)[Exercise 6: Share your code](#)[Exercise 7: Think about other bases](#)

Summary

This challenge focuses on teaching binary string conversion to and from decimal values. It consists of a series of incremental exercises designed to help students understand the use of if statements and loops. The exercises move from constrained binary-to-decimal conversions (8 bits or fewer) to more generalised solutions. Finally, the ‘reverse’ decimal-to-binary conversion is also offered as a challenge.

LEARNING OBJECTIVES

- Understand and Use If Statements in Python:
 - Learn how to implement conditional logic to handle binary string conversion with `if` statements.
- Differentiate Between Specific and General Solutions:
 - Appreciate the differences between specific (constrained) solutions and more flexible (general) solutions when converting binary strings to decimal integers.
- Gain an Initial Understanding of Loops (for and while):
 - Use `for` loops to traverse binary strings for conversion.

- Implement `while` loops for decimal-to-binary conversion using repeated division.
- **Comment and Document Code:**
 - Develop the habit of writing clear and descriptive comments in code to improve readability and maintainability.

BEFORE YOU START

It is a very good idea to do some research into binary number systems, how they work generally... before you start hacking code together. Also, you should compare how binary number systems compare to decimal number systems. Spatial positioning, the index of a given position, the 'base', how the base gets raised to the power of an index, the 'alphabet' (base 2 and base 10) of number systems.

Worked example, using the binary string 00001010:

- Each position in a binary string has a value that is a power of 2. For 8 bits, reading from left to right, the values are 128, 64, 32, 16, 8, 4, 2 and 1 (that is, 2 to the power 7 down to 2 to the power 0).
- The string 00001010 has a 1 in the positions worth 8 and 2, and a 0 everywhere else.
- Adding the values of the positions that hold a 1 gives $8 + 2 = 10$, so the binary string 00001010 is the decimal number 10.

Exercise 1: binary string I (8 bits or less)

Create a Python script named `bin_to_dec_if.py` that converts a binary string into its corresponding decimal/integer value using conditional logic (specifically, using **if/else** statements).

Details:

- handle binary strings with a maximum length of 8 bits; if the string is longer, tell the user and do not convert it.
- pad the string with leading zeros if its length is less than 8 (for example, 1010 becomes 00001010).

- for binary strings, the leftmost bit represents the most significant bit (MSB), and the rightmost bit represents the least significant bit (LSB).
- use if statements to check each bit's position and calculate its contribution based on the associated exponent.
- ensure you sum up the contributions to obtain the final integer value.

This script will allow you to convert binary strings into integers, considering the significance of each bit position, as described in the task.

Exercise 2: binary string II (8 bits or less)

Create a Python script named `bin_to_dec_loop.py` that converts a binary string of 8 bits or fewer into its corresponding integer value. The functionality of this script will be the same as `bin_to_dec_if.py`. However, the code will look very different: rather than using if logic for each bit, please use a for loop to convert the string. You might find this script more difficult to implement.

Exercise 3: binary string III (arbitrary length string)

Create a Python script named `bin_to_dec_any_length.py` that converts an n-bit binary string into its corresponding integer value – i.e., the received string can be of an arbitrary length. Again, the binary string will represent a binary number, where each bit's position determines its significance in the number, similar to Exercise 2. However, this time, first check that the input string contains only the characters '0' and '1', and print a helpful message if it does not, thereby handling inappropriate input. Then implement a for loop on the string and print the value.

CHECK YOUR ANSWERS (EXERCISES 1 TO 3)

- 1010 should give 10
- 11111111 should give 255
- 00000000 should give 0
- 100000000 (9 bits) should be rejected in Exercises 1 and 2, and should give 256 in Exercise 3
- 10201 should be rejected in Exercise 3, because it is not a binary string

Exercise 4: convert a decimal number to a binary number

Inside a script called `dec_to_bin.py`, convert a decimal integer to a binary number. Receive input from the user, ensure that the input is a whole number that is not negative, and then convert it to its binary representation. If the input is negative, or is not a whole number, display an error message.

- Start with the decimal number.
- Initialize an empty binary string.
- Perform Repeated Division by 2:
 - Divide the decimal number by 2.
 - Record the remainder.
 - Insert the remainder at the beginning of the binary string.
 - Repeat until the decimal number becomes 0.
- Special case: if the decimal number is 0 to begin with, its binary representation is simply "0".

This process converts a positive decimal to binary by repeatedly dividing by 2 and recording remainders until the decimal number becomes 0.

Example, using the decimal number 10:

1. Start with the decimal number 10.
2. Initialize an empty binary string: ""
3. Perform Repeated Division by 2:
 - a. 10 divided by 2 = 5, remainder 0. Insert "0" at the beginning of the string.

- i. Resulting binary string: "0"
 - b. 5 divided by 2 = 2, remainder 1. Insert "1" at the beginning of the string.
 - i. Resulting binary string: "10"
 - c. 2 divided by 2 = 1, remainder 0. Insert "0" at the beginning of the string.
 - i. Resulting binary string: "010"
 - d. 1 divided by 2 = 0, remainder 1. Insert "1" at the beginning of the string.
 - i. Resulting binary string: "1010"
4. The binary representation is complete: the binary representation of 10 is "1010"

CHECK YOUR ANSWERS

10 should give 1010, 255 should give 11111111, 1 should give 1 and 0 should give 0.

Exercise 5: Comment all your code

It is important to comment code! You should ensure that you 1. go over each script, 2. clean up any code and 3. add comments.

Exercise 6: Share your code

Share your code with other students; you can learn how they approached the problem.

Exercise 7: Think about other bases

Have a think about base 2 and base 10 number systems. The methods you have used here are not special to binary: given any alphabet of length n , they can produce a base- n number system. Hexadecimal, for example, is base 16 and uses the alphabet 0–9 followed by A–F. Look back at your own scripts and have a go at

answering these questions:

- In `bin_to_dec_any_length.py`, which parts of your code assume base 2? What would you change so that it could convert a hexadecimal string such as FF?
- In `dec_to_bin.py`, which parts of your code assume base 2? What would you change so that it produced a hexadecimal string instead?
- In binary, the value of a character is easy to find, because '0' is worth 0 and '1' is worth 1. How would you find the value of a character such as 'F' in a longer alphabet?
- What is the smallest base that makes sense? What would happen to your repeated division if the base was 1? And what limits how large the base can be?

CHECK YOUR THINKING, WITH PEN AND PAPER

- The decimal number 255 should give FF in hexadecimal (255 divided by 16 = 15, remainder 15, and 15 is written as F).
- The hexadecimal string 1F should give the decimal number 31 ($1 \times 16 + 15 \times 1$).
- The decimal number 10 should give 101 in base 3, using the alphabet 012.

If you want to go further (optional):

Create a Python script named `base_n.py` that puts your answers into practice. Write one function that converts a decimal integer that is not negative into a string in any base, and a second function that converts such a string back into a decimal integer. Both functions should receive the alphabet as a string, for example "01" for binary or "0123456789ABCDEF" for hexadecimal; the base is simply the length of the alphabet. Ask the user for a decimal number and an alphabet, print the converted string, then convert it back again to check that you get the number you started with. Display an error message if the alphabet has fewer than 2 characters, or if a character appears in it more than once.

CHECK YOUR ANSWERS

- 10 with the alphabet 01 should give 1010
- 255 with the alphabet 0123456789ABCDEF should give FF
- 10 with the alphabet 012 should give 101
- 0 should give the first character of the alphabet, whichever alphabet you use
- FF with the alphabet 0123456789ABCDEF should convert back to 255
- The alphabet 0 should be rejected, because it has fewer than 2 characters, and so should the alphabet 011, because 1 appears twice