

MCS 260 – Introduction to Computer Science – Fall 2020 – Emily Dumas

Project 1 – Due Fri Sep 18 at 6:00pm Central time

OVERVIEW

This document describes a computer program you must write in Python and submit to Gradescope.

In Lecture 1 we discussed a method to convert a number from decimal to binary by repeatedly dividing by 2 and using the remainders as binary digits. Here is the example given in that lecture:

To convert a number to binary, just keep track of the remainders when you repeatedly integer-divide by 2.

x	$x//2$	$x\%2$
312	156	0
156	78	0
78	39	0
39	19	1
19	9	1
9	4	1
4	2	0
2	1	0
1	0	1

So $312 = 0b100111000$, i.e.

$$312 = 256 + 32 + 16 + 8.$$

In this project you will write a program that reads a positive integer from the keyboard and prints a table similar to the one above followed by a statement of the binary representation of the integer.

OPERATIONAL SPECIFICATION

This section describes how your program must operate.

Your program must perform the following tasks in order:

- (1) Read a single positive integer from the keyboard using `input()`. No prompt should be displayed. From now on we refer to the integer entered in this step by n .
- (2) Print the string " $x \ x//2 \ x\%2$ " on a line by itself.
- (3) Repeatedly print lines consisting of three integers separated single space characters. These lines should reproduce the kind of table in the lecture slide above, but for the conversion of n to binary. The three integers on each line should be, from left to right:
 - A “current integer” x
 - The result of integer division of x by 2
 - The remainder of $x \bmod 2$

The first line should begin with the integer specified by the user, n . Each subsequent line should begin with the middle integer from the previous line. The last line should be the one where integer division gives zero, i.e.

1 0 1

- (4) Print a statement about the binary expansion in the format

Therefore, $[N] = 0b[DIGITS]$

where [N] is replaced by the decimal representation of n and [DIGITS] is replaced by the binary digits of n .

SOURCE CODE SPECIFICATION

This section describes how your program must be written and submitted.

Your program must be contained in a single file named `binary.py`.

The single file containing your program must be the only file you submit for grading (unless you complete the Extra Challenge described in the final section of this document).

The program must be able to run with Python 3.6 (which is consistent with all programming practices taught in this course).

Use of the `import` statement is not permitted.

The first lines of your program must follow a precise format:

- Line 1: A string literal describing the function of your program in your own words. (That is, a *docstring*.)
- Line 2: The following comment line, verbatim: `# MCS 260 Fall 2020 Project 1`
- Line 3: A comment line consisting of your full name
- Line 4+: A comment line beginning with `# Declaration:`, and then containing a full sentence that says, in your own words, that you are the sole author of the program you are submitting and that you followed the rules from the course syllabus in preparing it. (This part can span multiple lines.)

Here is an example of an acceptable start of a source file for a project submitted by a student named Srinivasa Ramanujan:

```
"""Compute the binary digits of an integer entered by the user"""
# MCS 260 Fall 2020 Project 1
# Srinivasa Ramanujan
# Declaration: I, Srinivasa Ramanujan, am the sole author of this code, which
# was developed in accordance with the rules in the course syllabus.
```

ACCEPTABLE USE POLICY AND PLAGIARISM

The program you submit will be run by an autograder as part of its evaluation. Submitting a program that attempts to sabotage, circumvent, or examine the autograder, or which is designed to bypass course policies in any other way, is a violation of the ACCC acceptable use policy. Such violations will be reported for possible disciplinary action.

Giving or receiving aid to other students on projects in MCS 260 is prohibited. Submitting code of which you are not the sole author (for example, adapting work of another student or from a textbook or online resource) constitutes plagiarism and will be reported for possible disciplinary action.

SUGGESTIONS

The previous sections contain all of the rules that your program needs to follow. This section attempts to clarify those rules by describing some of their consequences, and by pointing out certain things that may make your development easier.

Remember that the autograder is unforgiving. A program might satisfy the *spirit* of the assignment and yet receive no points from the autograder because it does not pass the automated tests. These tests require the exact output format described above. Please understand that attempting to make the autograder more “forgiving” would almost certainly make it less fair. For example, we would probably not anticipate all variations on the output format that students would produce, and so some minor variations would be accepted and others would not be.

Output formatting hints. The operational specifications describe the required output from your program. No additional output is allowed, and the format must be followed carefully. In particular:

- No prompt is to be displayed while waiting for user input.
- The table heading must be the exact string specified in step (2) above.
- There should be no extra line skips between the end of the table and the statement *Therefore*, ... from step (4).
- You must print integers, not floats. Printing 37.0 instead of 37 is not acceptable.


Undefined behavior. Note that it was specified that the input to your program will be a positive integer. The way your program behaves if the input is not a positive integer has no bearing on its evaluation. In particular you need not handle any of these possibilities:

- The input is not a decimal integer
- The input is zero or negative
- The input is empty

SAMPLE INPUT AND OUTPUT

This section contains examples of input and output for a program meeting the specifications of this project.

Note that producing the same output as in these examples is a *necessary* characteristic of a successful project submission, but it is not *sufficient*. Submissions will be tested in many other cases beyond the ones shown here.

In what follows,  indicates that the return key is pressed.

Example 1. Keyboard input:

75 

Output (available as <https://dumas.io/teaching/2020/fall/mcs260/project1/out75.txt>):

```
x x//2 x%2
75 37 1
37 18 1
18 9 0
9 4 1
4 2 0
2 1 0
1 0 1
Therefore, 75 = 0b1001011
```

Example 2. Keyboard input:

1932

Output (available as <https://dumas.io/teaching/2020/fall/mcs260/project1/out1932.txt>):

```
x x//2 x%2
1932 966 0
966 483 0
483 241 1
241 120 1
120 60 0
60 30 0
30 15 0
15 7 1
7 3 1
3 1 1
1 0 1
```

Therefore, $1932 = 0b11110001100$

EXTRA CHALLENGE

Only read this part if you have completed the main project. It describes an extra challenge that **will not affect your grade**, but which you can submit along with project 1 to receive feedback.

Write a second program called `extra.py` that can handle conversion to any base between 2 and 16. Submit it along with your main project 1 program. (Thus the extra challenge also includes an exception to the rule that you must submit only one source file.) It should read two integers from the keyboard, the first being n and the second being a base b , and print a table similar to the following example that exhibits the division/remainder process for converting to base b .

Keyboard input:

1000000 16

Output:

```
x x//16 x%16
1000000 62500 0
62500 3906 4
3906 244 2
244 15 4
15 0 15
```

Therefore, 1000000 in base 16 is f4240

REVISION HISTORY

- 2020-09-04 Initial publication