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

Project 1 Discussion

Solution

A solution has been posted at

https://dumas.io/teaching/2020/fall/mcs260/project1/project1soln.py

The code is:

```
"""Derive the binary expansion of a number provided by the user"""
# MCS 260 Fall 2020 Project 1 solution
# Emily Dumas

n = int(input())
print("x x//2 x%2") # header
binstr = "" # will store bits as a string
x = n # we will modify x but not n
while x > 0:
    print(x,x//2,x%2)
    if x % 2 == 0:
        binstr = "0" + binstr
    else:
        binstr = "1" + binstr
    x = x//2

# we use + in the next line to avoid extra space
print("Therefore,",n,"= 0b" + binstr)
```

Options

There are many ways to solve this problem. The method in the posted solution was chosen because it only uses concepts covered in Lectures 1–7 (that is, either before or just after posting of the project description). In general, the projects will not require class material taught near the due date.

Students did not need to solve the problem using a limited set of tools. In general, any method not explicitly forbidden in the project description is allowed. In this case, only the `import` statement was forbidden.

Some students used the built-in `bin()` function to get the binary digits, avoiding any need to save them while generating the table. This was allowed, but it *is* important to read and understand a solution that uses a minimal set of tools.

The most common difference in student solutions was the method used to store computed bits. Most students used either a string or a list.

- A string is easier to print later, but requires the use of `str(bit)` or a conditional (as in the posted solution).
- A list is simpler to work with, but `list.append()` will give a list of bits in the wrong order (see next section), and printing a list results in brackets and commas, making it more difficult to get the desired final output.

PITFALLS

While constructing the table, the bits are computed in the reverse order of the desired final printing, i.e. the first bit computed is the last one to be printed. Dealing with this order reversal was one of the main challenges of the assignment, and the source of most student difficulty.

The most straightforward way to handle this using Lecture 1–7 material is to put the newly computed bit at the *start* of a list or string. That way, the last bit computed will be the first element of the list/string. This is the function of lines 12 and 14 of the solution posted above. It was surprising how few student submissions did this, and how many opted to use methods to reverse a sequence that we hadn't yet discussed.

No direct method for reversing a list or string was taught until the day the project way due. One way to do this without resorting to very new material or concepts not yet introduced is to iterate over indices and use arithmetic to get the opposite order, e.g.:

```
1 s = "deliver"
2 for i in range(len(s)):
3     print(s[len(s)-i-1],end="")
4 print()
```

which has output

```
reviled
```

However, as discussed in Lecture 7, it is usually best to avoid `range(len())`. We will discuss other ways to reverse a list in the future.

A second issue in some solutions was the use of float values, e.g. substituting `int(x/2)` for the correct `x//2`. Floats have limited precision, and so any solution which uses floats will give incorrect results for very large integers.

EFFICIENCY

The posted solution to the project is not the most efficient one.

As mentioned in lecture, adding an element to the beginning of a list in Python is not an efficient operation for long lists. Similarly, adding a character to the start of a string is not efficient, as a new string is created each time.

With the benefit of additional knowledge of Python, we can write a more efficient solution in various ways. One way is to append each computed digits to a list, and then reverse and join the list at the end. For example, if `L` is a list of int digits such as $[1,0,0,1,1]$ then

```
binstr = "".join( [str(x) for x in L[::-1]] )
```

will reverse the digit order, convert to strings, and assemble into a single string (e.g. `"11001"`).

For 10,000 digits, this approach is about 50 times as fast as the more elementary solution that adds a character to the front of a string each time.

REVISION HISTORY

- 2020-09-22 Initial publication