MSc in High Performance Computing with Data Science
Programming Skills Coursework 1 - Code Review

Introduction

Software, like people, has a lifespan. Well written software will have a longer lifespan than poorly written software, but what is meant by "*well written*"? To begin to answer this another question must be asked: What is of paramount importance to a programmer? The answer is simple - their time. Time is saved when code is consistent and easy to understand, making the program easier to update, maintain, trust and reuse. "Through the simple act of writing intelligible source code, you can increase the efficiency of software development, enhance maintainability and improve overall productivity" (Zokaites, 2002). The aim of this project is to carry out a code review of a small program, written in python.

Code Review

At first glance the code is perplexing. The name of the program is ambiguous, nothing in the code is self documenting, and if not for a description in the handout, it would be very difficult to decipher what exactly is occurring. The first step to improving this code would be to change the name of the program. It is very difficult to find specific programs in a folder full of non-specific names. Renaming it to something relevant speeds up the process of finding each individual program, especially if has been a long time since it was edited rendering the *last edited* property useless.

Next, a comment giving a brief introduction as what the code is intending to do, with specific mention to key parts, would help a great deal. This allows the reader of the code to comprehend the objective of the code much quicker, and hastily spot any mistakes and amend them. Once the reader is in full knowledge of the codes objective, it becomes clear that the names of the function and array do not clarify what each part of the code does. The remedy for this is simple - to makes the names self-documenting, for example: renaming *ps* to *powers_array* would be constructive. Self-documenting names make the code more readable and understandable, and hence save time when maintaining and reusing the code.

Python has many preexisting mathematical functions installed such as: *factorial(), exp()* and *sum()*. Naming variable of a function after one of these functions is a blunder, along with the fact that the name is actually misleading. Variables with names such as these can get particularly confusing when the code gets reused in a program which utilises that particular function. These mistakes make the code more confusing and difficult to read, ultimately taking up more of the programmers time. Renaming *sum* to *input_value* is the elegant fix in this situation. On line 9 there is a mathematical statements within the for loop. This could be made much more understandable by changing the formula to *if (sum % 2==1):*. This explicitly tells us *if sum is odd then:.* This makes the code a whole lot more readable and understandable.

Comments can be a great benefit to programs which are difficult to understand, but on the other hand they are often inappropriately used where the following line is obvious. Unnecessary comments can reduce the clarity of code and are fairly unsightly. Moreover, some comments add extra confusion to the mix. If you write good code you shouldn't need a lot of comments (Rossum 2013). In *program.py,* line 11 is a useless and somewhat confusing. The comment on line 11 suggests a bug may be present, and states the what the code does even though it is inherently obvious. There is a comment on line 12 beside an unusual piece of code, but it gives no indication as to what a *shift* actually does. The comment should be edited to explain that this shifts the binary values to the right one place - essentially halving the decimal value and rounding down where possible.

Lastly, on lines 14-16 a for loop has been implemented in order to print out the values of the different powers of two. This is excessive. Writing extra for loops makes the code seem more complex and reduces readability. The comment on line 13 explains what is what the next section of code does, but to make the code more efficient the three lines could be changed to *print‘ ‘.join(ps),* or a similar function which prints the values of the array.

Conclusions

Code is read much more that it is written (Rossum 2013), and in order to slash times it must be quick to read and understand. The aim of writing programs is not to write functional code in as little lines of possible, but to write code for other people so that these times can be reduced. Readable and understandable is easier to validate and reuse, reducing the overall time a programmer must spend on programming tasks. The code presented in this task was for a program which was essentially pointless when you notice that numbers written in binary already do exactly what the program does. The location of the *1*s in binary form indicate that that particular power is present, however it does write it out nicely, i.e.

```
 1   1  0  0  1
16   8  4  2  1
```

The code in our particular example worked perfectly, but was a struggle to understand what each part of the code did and whether it was necessary. After implementing the changes suggested above the code will be much more readable and hence easier to reuse and validate. The two key points to be taken are that a persons time in much more important than a machines, and in order to save a person time we should make our program as easy to understand as possible.

Code and Bibliography

```
#!/usr/bin/python

import sys

def p2(sum):
    ps = []
    x = 1
    while (sum > 0):
        if (sum % 2):
            ps.insert(0, x)
        x = x * 2  # Multiply by 2. Is this a bug?
        sum = sum >> 1 # Do a shift.
    # Print the powers.
    for x in ps:
        print x
    return ps

if __name__ == '__main__':
    # TODO Convert sys.arvg[1] into an integer.
    value = int(sys.argv[1])
    p2(value)
```

Zokaites, David Michael. "Writing understandable code." SOFTWARE DEVELOPMENT-SAN FRANCISCO- 10.1 (2002): 48-50. (Zokaites, 2002)

Guido van Rossum, Barry Warsaw, Nick Coghlan. “Style Guide for Python Code.” PEP 0008, (2013) (Rossum 2013)