Pseudocode step by step

Introduction

A valuable tool in a programmer's toolbox is pseudocode. In this reading, you will learn about why you should use pseudocode, when it should be used, and how to write pseudocode.

Why should you use pseudocode?

Pseudocode is a legitimate first step when starting to devise a solution. Pseudocode is a high-level representation of ideas written in a way that looks like code. Fundamentally it can be an aid to highlight for yourself what elements a program should include.

Each line will give you a moment to pause and consider what is required to achieve a given outcome. When writing an application, a decision made in step 3 could impact how step 6 should be coded. Each stage has an element of constraint that informs how a subsequent step will be completed.

Consider that you are writing an application that must store data in step 3. You settle on an array before continuing. At step 6, you realize several lookups are required for an application. While writing pseudocode, it is easy to revisit step 3 and change the data structure to something more compatible with step 6, such as using a dictionary instead of an array. Slight changes could increase your overhead if an implementation has already begun because it alters how steps 4 and 5 operate.

When should you write pseudocode?

- As a beginner, when plotting out the planned progress of your approach.
- As an experienced programmer, when attempting to wrestle with a complex problem.
- If you are trying to convey a concept to an influential audience, such as a team or potential clients.
- If you are signposting your work for future coders who may be maintaining the code or application you wrote.
- In an interview, when demonstrating your ability to reason out a problem.

How to write pseudocode

There is no one set way to write pseudocode. Each organization may have its own standard. In general, you can say that pseudocode can be considered to be any textual representation that outlines a program's operation.

Consider how to represent the FizzBuzz challenge used to test candidates' ability to reason in code:

Write a program in a given language that iterates over numbers 1 to 40. Print out a number for every number except multiples of three, in which case output Fizz. For multiples of five, output Buzz, and for multiples of 3 and 5 output FizzBuzz.

You might start by representing each requirement as a line of pseudocode:

```
FOR number 1 to 40
    IF multiple of 3
```

```
        output(Fizz)
    IF multiple of 5
        output(Buzz)
    IF multiple of 3 and 5
        output(FizzBuzz)
     ELSE
        output(number)
```

The key to this assignment is knowing how to order the conditional statement. Line 1 indicates that there will be an iteration. In this instance, an indent demonstrates that the subsequent eight lines of code are part of this iteration. Alternatively, the following block could have been added:

```
START FOR LOOP

#code

END FOR LOOP
```

It is clear that there are three conditional statements and then a catchall **`else`** clause. It is possible to visualize the code's outcome by looking at the pseudocode. The **`else`** statement will work fine, it is only to print out non-multiples of 3 and 5, but there is an issue with how the code will handle the number 15, printing an instance of Fizz, Buzz and FizzBuzz.

```
FOR number 1 to 40
    IF multiple of 3 and 5
        output(FizzBuzz)
    ELSE IF multiple of 3
        output(Fizz)
    ELSE IF multiple of 5
        output(Buzz)
    ELSE
        output(number)
```

Examining the question's first instance as pseudocode enables you to spot where the issues might be. Visually, it is far easier to see how the conditional statements should be organized. First, the crux of the question is in what order the conditional statements should be placed and to use exclusionary conditional statements. In the above instance **`else if`** was used in place of just **`if`**. Second, the ordering of the statements has an important impact. You can try the output of above by running the code found below.

```
1  for number in range(40):
2      if number % 3 == 0 and number % 5 == 0:
3          print("FizzBuzz")
4      elif number % 3 == 0:
5          print("Fizz")
6      elif number % 5 == 0:
7          print("buzz")
8      else:
9          print(number)
```

Run

Reset

Conclusion

In this reading, you learned about why you should use pseudocode, when it should be used, and how to write pseudocode.

The scope for pseudocode goes beyond your remit to grasp a new programming language. It is a valuable practice that helps visualize the code's flow. It outlines what skills or libraries may be required to complete a task.

Software engineers use it at the start of their journey to gain insight into how a program might flow. Senior engineers use it to demonstrate ideas to a team. There is no one way of writing it, but the style you settle on will resemble the structure of the programming language you like best.