Procedural Programming Example in Python

Dominic Carr

1 What's this Procedural thing again?

In the procedural programming paradigm the overall program objective is decomposed into a series of functions which accomplish some smaller piece of the overall objective. This makes code easier to understand, to maintain, and to reuse in future.

Consider the snippet in Listing 1 therein the programmer is importing two functions from the math library and using one of them. This is much easier than "reinventing the wheel"[1], and we can add this thinking to our own programs.

Python has a vast amount of reusable functions available in it's standard library. This is in contrast to more low-level languages like C where the standard library is much smaller.

```
from math import exp, expm1
exp(1e-5) - 1  # gives result accurate to 11 places
```

Listing 1: Using external functions

2 Imperative Program

Listing 2 is a small Python program which opens a CSV file and performs from rudimentary extraction of statistical information. The program is not the the most readable as there are no comments and the code has not been decomposed into a series of functions. It is essentially[2] an **imperative program**. Imagine if this program were 30 times as long with no functions and no comments? It would be unreadable and not maintainable.

```
import csv

with open('example.csv', mode ='r') as file:
    csvFile = csv.DictReader(file)
    scores = []

    for lines in csvFile:
        score = int(lines["Score"])
        scores.append(score)

    total = 0
    for score in scores:
        total += score

    average = total / len(scores)

    minimum = scores[0]
    for score in scores:
        if minimum < score:
            minimum = score

    maximum = scores[0]
    for score in scores:
        if maximum > score:
            maximum = score
```

[1] That said, who in their right mind would hire an engineer who couldn't make a wheel?
[2] What are the exceptions?

```
26
27      list1 = scores.copy()
28      for i in range(0,len(list1)-1):
29          for j in range(len(list1)-1):
30              if(list1[j]>list1[j+1]):
31                  temp = list1[j]
32                  list1[j] = list1[j+1]
33                  list1[j+1] = temp
34
35      median = list1[int(len(list1)/2)]
36
37      highest_freq = 0
38      mode = scores[0]
39      for score in scores:
40          frequency = 0
41          for score2 in scores:
42              if score == score2:
43                  frequency += 1
44          if frequency > highest_freq:
45              mode = score
46              highest_freq = frequency
47
48      print(f'Average: {average} Median: {median} Smallest: {minimum} Largest: {maximum} mode: {mode}')
```

Listing 2: Small Imperative program

3 Procedural Program

Listing 3 is a modified version of the Python program which conforms to the **procedural programming paradigm**. The modified program produces the same results as the original but the code is more readable, it can be more easily understood by a programmer. The computer doesn't care about paradigms, every paradigm, every abstraction is for the benefit of the programmer. Most functions represent the extraction of one piece of information from the imported data. The function names are descriptive, making it easy to understand what they are intended to accomplish.

The function **get_median_value** uses another function **bubble_sort** to sort the data. This is good example of functions calling each other.

The function **get_average** shows an example of a docstring which is a way to document what a method does, what inputs it has and what the return value is. It may seem irrelevant here as the code is fairly simple but such documentation is very useful in real-world coding scenarios and this is just a simple example. Do you recognise the code beginning on lines 27-33?

```
1  import csv
2
3  def get_maximum_value(list):
4      maximum = list[0]
5      for l in list:
6          if maximum > l:
7              maximum = l
8      return maximum
9
10 def get_minimum_value(list):
11     minimum = list[0]
12     for l in list:
13         if minimum < l:
14             minimum = l
15
16 def get_average(list):
17     """
18         Given a list of numbers as input this function will return the numerical average.
19
20         :param list: the list of numbers given as input
21         :return: the numerical average of the list
22     """
```

```python
    total = 0
    for l in list:
        total += l

    average = total / len(list)
    return average

def get_median_value(list):
    list1 = list.copy()
    bubble_sort(list1)
    median = list1[int(len(list1)/2)]
    return median

def bubble_sort(list1):
    for i in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp

def get_mode(list):
    highest_freq = 0
    mode = scores[0]
    for score in scores:
        frequency = 0
        for score2 in scores:
            if score == score2:
                frequency += 1
        if frequency > highest_freq:
            mode = score
            highest_freq = frequency
    return mode

def read_scores_from_csv(filename):
    scores = []
    with open(filename, mode ='r') as file:
        csvFile = csv.DictReader(file)

        for lines in csvFile:
            score = int(lines["Score"])
            scores.append(score)
    return scores

if __name__ == "__main__":

    scores = read_scores_from_csv('example.csv')

    average = get_average(scores)
    minimum = get_minimum_value(scores)
    maximum = get_maximum_value(scores)
    median = get_median_value(scores)
    mode = get_mode(scores)

    print(f'Average: {average} Median: {median} Smallest: {minimum} Largest: {maximum} mode: {mode}')
```

Listing 3: Procedural version

4 Tasks

Before you begin review all of the code discussed above and make sure you understand it all.

1. Add a doc string to each function in the procedural program, make sure to document each parameter and the return value. You can see this has done for **get_average** in Listing 3.

2. Move the functions around, you should find that this has no effect on the program execution as they are just blocks of code which are called by the "main method" at the bottom. This won't be true in older languages like C, so be aware of that.

3. Move bubblesort into another .py file and use the import keyword to import it into your file for use in the median calculation. This will look something like:

```
    from sorting_algorithms import bubble_sort

```

   This can be very useful when programs grow large.

4. Add in two new functions which extract some statistics from the data. No imports allowed here.

5. Write a new function to bring in data from a separate column in the CSV file, can you do this by modifying the existing method to be more generalised? Consider default parameters (this is a language feature of Python).

6. Replace 4 functions in the procedural program with imports or the use of the standard Python library. For example you could replace **get_minimum_value** with the **min()** function for lists which is built into Python.