How can Programs Be Used to Solve Problems?

Stages in Developing a Program

1. Gather Requirements
2. Plan Solutions
3. Write Code
4. Test and Refine Code
5. Deploy Code

Gather Requirements

To determine the nature of the problem or their expectations. Some of the tasks that can be done during this stage:

1. Interviewing the intended audience of the program to understand the nature of the problem or their expectations
2. Specifying the complete set of outputs that is necessary for the problem and how the inputs can be supplied to the program being developed.
3. Specifying the complete set of inputs that is necessary for the problem and the format for the output

Plan Solutions

The goal of this stage is to consider the options available before any code is written, and to choose an algorithm based on the resources available (such as manpower and time). Some of the tasks that can be done at this stage:

1. Manually solving different simplified examples of the problem and generalising the steps needed to produce the required output
2. Trying different ways to break down the problem into smaller parts such that the intended output of each part gets closer and closer to what is needed to solve the problem
3. Comparing the problem (or its smaller parts) to other problems that have been solved before and identifying which algorithms can be used
4. Estimating the amount of effort needed to write the code or the time needed to complete the algorithm before making a definite choice
5. Writing possible algorithms using either flowcharts or pseudo-code

Write Code

The goal of this stage is to write code that performs the algorithm as planned in the previous stage as efficiently as possible.

Test and Refine Code

After the initial code is written, the resulting program is likely to require further refinement. Some possible reasons for this:

- The programmer may have made mistakes in translating the planned algorithm into code or may have forgotten to consider exceptional cases where the input would need to be treated specially. For example, the programmer may have made a syntax error or forgotten to check for invalid input. These are relatively minor errors that usually would not require a major rewriting of code as simply correcting the syntax error or adding an if statement would usually be sufficient to correct the program.
- The solution-planning stage may not have been performed properly, resulting in an unsuitable or incomplete algorithm. Depending on how serious the mismatch is, it may

be possible to keep most of the written code and simply make refinements. Otherwise, it may be necessary to discard the written code and redo the evaluation of algorithms.
- The requirement-gathering stage may have been incomplete, resulting in code that does not actually solve the problem. Depending on how serious the mismatch is, it may or may not be possible to reuse most of the written code.

Test Case

A test case usually consists of a set of inputs and the corresponding set of expected outputs.

Deploy Code

With the code tested and refined, this is the stage where the program is actually "rolled out" and used by its intended audience. Some of the tasks that can be performed under this stage:

- Training users to use the program
- Transitioning from an old program or system to a new program
- Evaluating the effectiveness of the program in solving the problem and considering any changes that might increase its usability or effectiveness

Finding Check Digits

A **check digit** is usually an additional digit or letter added in the end of a sequence of digits that is intended to be read by or entered into a computer manually. The check digit is mathematically related to the original sequence of the digits so that simple input errors would break this relationship and hence be detected. If the check digit is a letter, it would usually need to be converted to a number so that it can be used in the algorithm to check the sequence.

Check digits are generally designed to capture human transcription errors.

Examples of Check Digits

- **UPC, EAN, GLN, GTIN, numbers administered by GS1.**
  - The final digit of a Universal Product Code, International Article Number, Global Location Number or Global Trade Item Number is a check digit.
- **ISBN 10/ ISBN 13**
  - International Standard Book Number

Notable Algorithms

- Luhn Algorithm (1954) by IBM Scientists Hans Peter Luhn
  - Credit Card number
  - Canadian, Greek, Israeli, South African, Swedish ID.
- Verhoeff Algorithm (1969) by Dutch Mathematician Jacobus Verhoeff
  - The first decimal check digit algorithm which detects all single-digit errors, and all transposition errors involving two adjacent digits.
  - Uses three tables: a multiplication table *d*, an inverse table *inv*, and a permutation table *p*.
- Damm Algorithm (2004) by H. Michael Damm (PhD Dissertation)
  - Similar to the Verhoeff algorithm
  - Does not have the dedicated constructed permutations and its position-specific powers of the Verhoeff scheme.

```
# UPC-A Practice

def check_digit(code: str):
    result = 0
    sum_odd = sum([int(k) for k in code if code.index(k) % 2 == 0])
    result = sum_odd * 3
    sum_even = sum([int(k) for k in code if code.index(k) % 2 != 0])
    result += sum_even
    result = str(result)
    if result[-1] == '0':
        return 0
    else:
        return 10 - int(result[-1])

def validate_code(code: str):
    if int(code[-1]) != check_digit(code):
        return False
    else:
        return True
```