Lesson 4 Designing test-cases

# Learning goal

Design test-cases for programs before coding and use them to validate programs after coding.

# Agenda

1. Recall the definition of algorithm correctness.
2. How NOT to test a program for correctness:
   1. Test every single input (impossible)
   2. Test only inputs of the same type (gives biased results)
   3. Test only a few inputs (incomplete)
3. The right way to test a program:
   1. Design test-cases that represent a broad class of inputs
   2. Design a SET of test-cases that would represent all allowable inputs
4. How to go about testing: Enclose your whole program in a while-loop! So you don’t have to keep re-starting the program to run every test-case.

## Definition of algorithm correctness

An algorithm (or program) is correct if it generates the desired output for all allowable inputs.

## How not to test a program for correctness

Example 1: The ordinal number program.

Should we test the program on every single value? That’s usually either impossible or impractical.

What if we just tested it on numbers 1-9 and declared ourselves done? That’s incomplete because those cases don’t represent all that the program is designed to test for.

**Goal**: Design a set of test-cases that represents all of the allowable inputs.

|  |  |
| --- | --- |
| Input | Desired output |
| 1 | 1st |
| 2 | 2nd |
| 3 | 3rd |
| 4 | 4th |
| 5 | 5th |
| 11 | 11th |
| 12 | 12th |
| 13 | 13th |
| 14 | 14th |
| 21 | 21st |
| 22 | 22nd |
| 23 | 23rd |
| 24 | 24th |

Programmer is thinking:

“If my program ran correctly on all these test-cases, I’d be convinced that my program is correct.”

Why? Why don’t we have to test numbers like 3467? Or 56?

We don’t have to test anything bigger than 100 because we know that only the last two digits have any effect on the ending to be used.

We don’t need to test 6, 7, 8, 9 or 10 because if 4 and 5 run correctly, we’re convinced that the program is correctly outputting “th” from now on.

**Example 2: Trinomial Formatting from Problem 1 of Assignment 2**

Let’s design a set of test-cases (i.e. combinations of a, b and c) that

1. represents all the allowable inputs
2. but is small enough to be manageable.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INPUTS** | | | **DESIRED OUTPUT** | Comments |
| ***a*** | ***b*** | ***c*** |
| 3 | 6 | 8 | 3x^2 + 6x + 8 | Do we need to test 3,6,9 also? Why not? |
| 3 | -6 | -7 | 3x^2 – 6x – 8 | Do we need to test any more negatives? Yes! -1! |
| -1 | -1 | -1 | -x^2 – x – 1 | Note we’ve combined several cases into one. This is more efficient than testing 3,-1,8 and 3,6,-1 and -1,6,8 separately. |
| 1 | 1 | 1 | x^2 + x + 1 | Again combining three cases into one to save time |
| 0 | 0 | 0 | no output | All three zeroes is a special case that must be tested |
| 3 | 0 | 0 | 3x^2 | Test all cases where there are two zeroes |
| 0 | 6 | 0 | 6x |
| 0 | 0 | 8 | 8 |
| 0 | 6 | 8 | 6x+8 | Test all cases where there is exactly one zero |
| 3 | 0 | 8 | 3x^2 + 8 |
| 3 | 6 | 0 | 3x^2 + 6 |

## How to test a program quickly

Instead of running the program, entering one test case, closing the window, then running the program again on the next test-case, we can do them all at once using a while-loop.

while True:

a = int(input(“Enter a: “”))

b = int(input(“Enter a: “”))

c = int(input(“Enter a: “”))

if a == 0:

etc.

#the rest of your code goes here.

This will run the program repeatedly without your having to close the window and re-start each time.