Open in Colab

SWEN90006 Tutorial 1

What is testing really?

> *It works! Trust me, I've tested the code thoroughly!*

With those bold words many software projects have been made bankrupt.

The aim of this tutorial is to start to get an understanding of our major themes; that is, of the factors that make up quality and how testing effects them.

While we have not yet looked at testing formally yet you will be asked in this tutorial to test a small program fragment with the purpose of starting to see where the difficulties of testing lay in practice.

Important terminology

Before beginning the exercises, consider the following three definitions, the first three of which are defined in the chapter Introduction to software testing of the subject notes:

- *Fault*: An incorrect step, process, or data definition in a computer program. Faults are the source of failures -- fault in the program triggers a failure under the right circumstances.
- *Failure*: A deviation between the observed behaviour of a program, or a system, from its specification. Faults cause failures; and in fact, one fault can cause many failures.
- *Error*: An incorrect *internal* state that is the result of some fault. An error may not result in a failure -- it is possible that an internal state is incorrect but that it does not affect the output.

Faults cause failures and errors. A fault in a program can trigger a failure and/or an error under the right circumstances. In fact, a single fault could cause multiple failures and multiple errors.

In normal language, software faults are usually referred to as "bugs", but the term "bug" is ambiguous and can mean to faults, failures, or errors; as such, as will avoid this term.

Your tasks

The small programs below are taken from the exercises in Chapter 1 of *Introduction to Software Testing* by Offutt and Ammann. For each program, the test case below the program results in a failure that is caused by a fault in the program.

For each of the programs below, perform the following tasks:

1. Specify the input domain and valid input domain of the function.
2. Identify the fault.
3. If possible, identify a test case that does not execute the faulty statement.
4. If possible, identify a test case that executes the faulty statement, but does not result in an failure.
5. If possible, identify a test case that forces the program into an error (that is, the first step that a program deviations from its intended behaviour, but does not produce a failure).
6. Fix the fault and verify that the given test now produces the expected output.

The programs

In [1]:
```
'''
  x is a list of integers
  y is an integer
  Return the index of the last element in x that equals y.
  If no such element exists, return -1
'''
def find_last (x, y):
  i = len(x) - 1
  # while i > 0
  while i >= 0:
    if x[i] == y:
      return i
    i -= 1
  return -1

# The expected output is 0 (the 0th element)
x = [2, 3, 5]
y = 2
assert find_last(x, y) == 0, "Test failed: find_last(%s, %d) == %d" % (x, y, find_last(x, y))

# 1. x is a list of elements in Z
#    y is in Z
# 2. while i > 0:
# 3. x = [], y = 1
# 4. x = [0, 1], y = 1
# 5. -
```

In [2]:
```
'''
  x is a list of integers
  Return the index of the LAST 0 in x.
  Return -1 if 0 does not occur in x.
'''
def last_zero(x):
  # i = 0
  i = len(x) - 1
  #while i < len(x):
  while i >= 0:
    if x[i] == 0:
      return i
    #i += 1
    i -= 1
  return -1

# The expected output is 2
x = [0, 1, 0]
assert last_zero(x) == 2, "Test failed: last_zero(%s) == %d" % (x, last_zero(x))

# 1. x is a list of elements in Z
# 2. Returns the first zero, not the last
# 3. x = []
# 4. x = [0, 1, 2, 3, 4]
# 5. -
```

In [3]:
```
'''
  x is a list of integers
  Return the number of _strictly_ positive elements in x.
'''
def count_positive(x):
  count = 0
  i = 0
  while i < len(x):
    #if x[i] >= 0:
    if x[i] > 0:
      count += 1
    i += 1
  return count

# The expected output is 2
x = [-4, 2, 0, 2]
assert count_positive(x) == 2, "Test failed: count_positive(%s) == %d" % (x, count_positive(x))

# 1. x is a list of elements in Z
# 2. if x[i] >= 0
# 3. x = []
# 4. x = [1, 2, 3]
# 5. -
```