# Technical Challenge Senior Python Engineer

## Candidate: Andres Balverde

## Solution Introduction

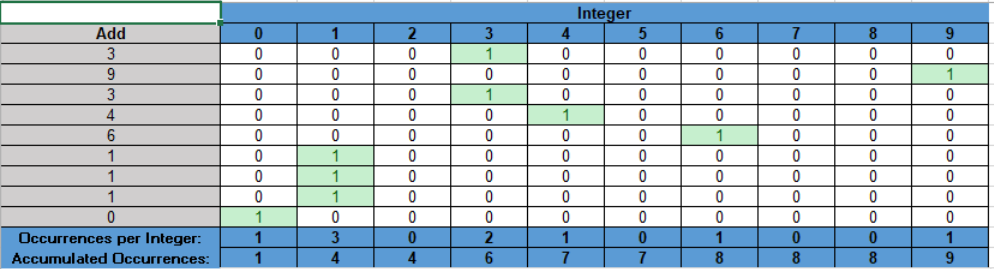
The input items (integers) are processed into a dictionary that orders the inputs in a sequence (keys) and provides the accumulated occurrences (values) for each input. This approach makes it easy for the stats functions to obtain the ‘less than’, ‘greater than’ or ‘between’ just as ‘delta accumulations’ in the specified range. Accomplishing this way the condition that states “methods add(), less(), greater(), and between() should have constant time O(1)”.

## Solution Details:

Given the condition that the stats methods should have a constant time O(1), the design is focused on building a dictionary (count\_dict) where:

* Keys: are the sequence of integers from 0 to number\_of\_items (the maximum integer added as input item).
* Values: are the accumulated occurrences per each of the integers.

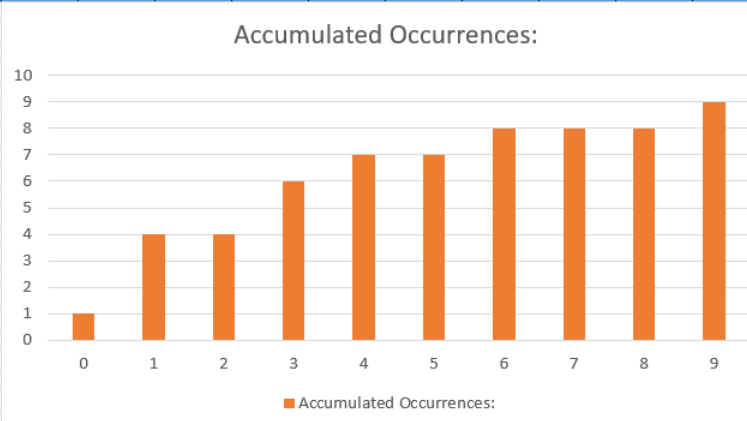
The following section clarifies the above definitions using an example.



Key arrays are:

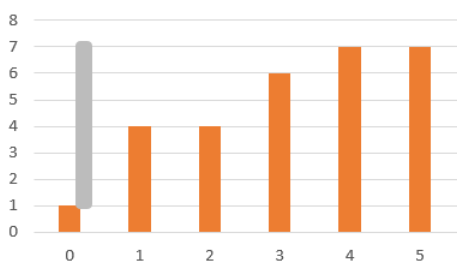
* **Add**: is the given list of input items, meaning the values *i* provided by *capture.add(i)* is given by the column Add: {3, 9, 3, 4, 6, 1, 1, 1, 0}
* **Integers**: is the sequence of integers from 0 to the maximum item provided, which in this case is 9: Integers: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
* **Occurrences per Integer**: counts the number of occurrences for each of the integers that we encounter in our input items: for example we have only one occurrence for “0”, whereas the input “1” occurred three times, and so on: {1, 3, 0, 2, 1, 0, 1, 0, 0, 1}
* **Accumulated Occurrences**: this is just the added occurrences for each integer in the sequence. For example we have one occurrence for “0” and three occurrences for “1”, so this sequence starts as {1, 4…}, the full sequence being: {1, 4, 4, 6, 7, 7, 8, 8, 8, 9}

A friendlier visualization of this Accumulated Occurrences is given by the following bar graphic.



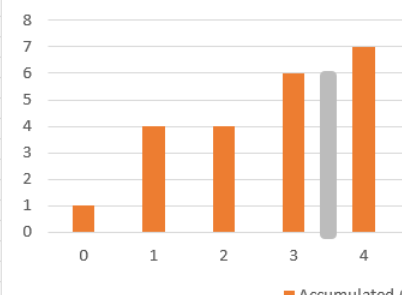
Example: between(1, 4):

Since the ‘between’ function includes the limits, is given by the gray bar below, with a length of 6, meaning there are 6 items within that range.



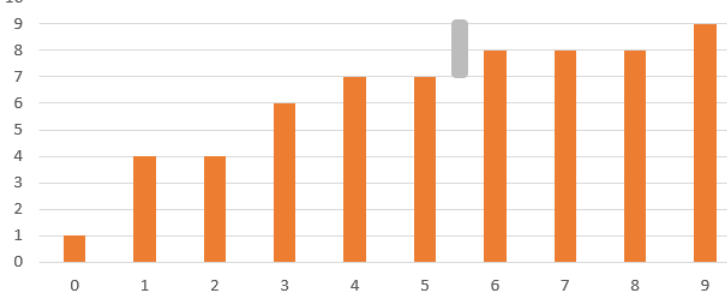
Example: less(4)

Since the ‘greater’ and ‘less’ stats functions do not include the specified limit, the number of items that are less than 4 is 6, given by the grey bar below:



Example: greater(5)

Since the ‘greater’ functions doesn’t include the specified limit, the number of items that are greater than 5 is 2, given by the grey bar below:



# Test Design, Execution and Results

A set of functional automatic tests has been put in place under /test folder.

Having at least two functional tests per each of the stats methods, using pytest assertions as detailed below.

## Project Execution and Functional Test Options

Use one of the following options to execute the project:

1. From root folder, run main.py which includes a default set of test cases to demonstrate all the features properly working
2. From /test folder, run test\_cases.py using pytest, to verify every function is properly working under at least two different test cases for each.
   1. Run all Test Functions using the command line:

*pytest test\_cases.py*

* 1. Run particular test cases to validate each of the functions: less, greater or between:

*pytest test\_cases.py::test\_greater\_small\_set*

*pytest test\_cases.py::test\_greater\_big\_set*

*pytest test\_cases.py::test\_less\_small\_set*

*pytest test\_cases.py::test\_less\_big\_set*

*pytest test\_cases.py::test\_between\_small\_set*

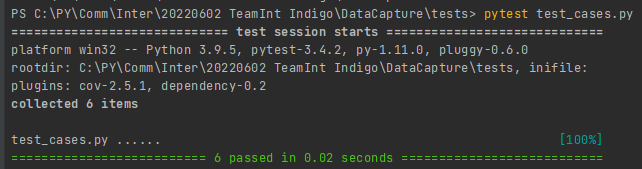
*pytest test\_cases.py::test\_between\_big\_set*

# Tests Executions and Results:



## Tests Results & Evidences:

All six tests can be executed at once with the following command line:



Tests Results Test ID 1 to 6:

