Mod 3 Homework - Running time analysis

We can often calculate the running time of a function by adding up the running times of every line. For instance, the below function has a O(n) running time to add up all the items in a list:

```
def sum(L):
   total = 0          # 1

   for item in L:     # n
      total += item   #     2 (add, then assign)

   return total       # 1
                      #--------------
                      # 1 + 2n + 1 = O(n)
```

Part 1 - create functions

Write a file called `functions.py` with three functions with the following running times, respectively:

- O(1)
- O(n)
- O(n^2)

These functions should do something useful, not just arbitrarily run for a long time.

- Use a docstring to describe what each function does, including what `n` is in your problem (e.g number of items in a collection)
- Use comments to denote the cost of each individual line, as well as the total cost, as above.
- Do not use functions from the book, VQs, or this assignment.

Part 2 - time those functions

Write a file called `time_functions.py` that uses the `time` module to time the functions defined above.

- Create a function `time_f` that takes three parameters - a function, arguments for that function, and the number of trials you want to run
- By default, use 10 trials
- Return the minimum time taken to run that function on the arguments (**not** the average)
- Print a table that shows the running times of the functions you defined in Part 1 as n grows. Format the table nicely, for example:

```
================================================
n     t_const (ms)   t_lin (ms)     t_quad (ms)
------------------------------------------------
100   10             100            1
200   11             200            4
400   9              300            16
600   8              400            36
800   10             500            64
------------------------------------------------
```

(Note - times above are guesses. Yours will probably not be as clean. Pick a range of `n` values and a unit for your times that works for your functions).

Part 3 - Python's built-ins

Continue working in `time_functions.py`. Create a table as above, but for searching python's built-in collections

- list
- tuple
- string
- set

using the `in` function, e.g.:

```
>>> L = [1, 2, 3, 4, 5]
>>> S = {1, 2, 3, 4, 5}
>>> 1 in L
True
>>> -1 in S
False
```

Use the worst case scenario - searching for an item that is not in the collection. You will probably need to use *very* big collections to see a meaningful difference. Make sure you only time how long it takes to search the collection, not how long it takes to create it. Your final table might look like

```
================================================
`item in collection` (times in ms)
------------------------------------------------
n     t_list      t_tup      t_str      t_set
------------------------------------------------
...
------------------------------------------------
```

You may need to make a modified `time_f` for this part - that's fine.

Submitting

At a minimum, submit the following files:

- `functions.py`
  - contains your 3 functions with running times noted
- `time_functions.py`
  - prints out a table showing running times for those 3 functions
  - prints out a table showing running times for searching python's built-in collections