CT5132/CT5148 Lab Week 02

James McDermott

1. **Dictionaries**. Use a `dict` comprehension to invert a dictionary. That is, if in the original `dict` we have a key-value pair `k: v`, we should now have `v: k`. See `invert_dict.py` for doctests.
2. Is it possible to have multiple entries in a `dict` with the same key `k`? What is the effect of your invert code if there are multiple entries in the original `dict` with the same **value** `v`? Think about it, then try it. See `invert_dict_sol.py` for a solution.
3. **Higher-order functions**: we want to create a list containing $e^x \quad \forall x \in [0.0, 0.1, ..., 1.0]$. Use `range`, `lambda`, `map` and of course `math.exp` to do this. (Just a one-liner, no function, no doctests.) See `exp_map.py` for a solution.
4. **Exceptions**. In the following code, check that the user does not request too large a value of n. If they do, `raise ValueError` with an informative message such as `ValueError: Can't return 7 elements from abcde of length 5`. Hint: you could use an f-string to create that string. See `get_last_n_elements.py` for a version with doctests, and `get_last_n_elements_sol.py` for the solution.

```
def get_last_n_elements(s, n):
    return s[-n:]
```

5. **Itertools**: a **magic square** is an $n \times n$ grid containing the numbers $1, 2, ...n^2$ (used exactly once each) such that each row and column sums to the same value. Here is a 3×3 magic square:

```
(9, 5, 1)
(4, 3, 8)
(2, 7, 6)
```

   We will generate all magic squares for $n = 3$. Look up `itertools.permutations` and use it to generate all permutations of the numbers $1, 2, ...9$. Next, for each permutation `p`, think of it as a grid, like this:

```
(p[0], p[1], p[2])
(p[3], p[4], p[5])
(p[6], p[7], p[8])
```

   Check whether the rows and columns sum to the right value, and if so, print it out. You should find 72 of them, including the one mentioned above. See `magic_squares_output.txt` for the output, and see `magic_squares.py` for a solution.

   Hint: in Python, you can chain multiple comparisons together, e.g. `x == y == z`.

6. **Generators**: create a generator (a "function" that uses `yield`) that yields the squares of all the non-negative integers, starting at 0.
7. Test it by running `for s in sq(): print(s)`. Of course it creates an infinite loop. To exit the loop, we have to interrupt Python:
   - In Spyder, type Ctrl-C in the console
   - In Terminal or IPython, type Ctrl-C

- In Jupyter Notebook, find the "stop" button (a square icon), or go to the `Kernel` menu and select `interrupt`.

8. Without altering your generator, use it to create a `for`-loop that prints out all the even squares ≤ 100. This time, your `for`-loop could use `break` to avoid the infinite loop. See `square_generator.py` for a solution.