InterBit
Consulting & Training Ltd.

Python Programming
Exercise Book

Chapter 3: Types and Operators
==========

Ex 1
Read from standard input a line containing digits.
If the line contains non-digits, print an error message.
If it's valid, count the number of times each digit (0-9) appears in the line and print a report.

Ex 2
Define a dictionary with unique values. Now “switch sides” in the dictionary: make a new dictionary where the keys are the values and vice versa. Print the new dictionary.

Ex 3
Read a file. Count the number of times each word appears in the file. Produce a report (write it to a new file).

Chapter 5: Functions
==========

Ex 4
Write a function that accepts a series of numbers, and returns the biggest number, the smallest number and the average.
Hint: Most of the work is already done by built-in functions.

Ex 5
Repeat Ex 2, but now as a function:
Write a function that receives a dictionary and returns a new dictionary with “switched sides”.
Write a documentation string, including a doctest.

Chapter 6: Data Processing

==========

Ex 6

Write a function that receives a list of unary functions and a single argument, and calls each function with the argument. It should return a list of the results.
Note that this is the opposite of map() which applies one function to different arguments.

[Bonus Ex: take any number of arguments, including named arguments, and pass them to each function.]

Ex 7

Repeat Ex 6 but now write it as a generator. It should call the functions only as you request their results. Verify this by calling it with functions that print something.

Ex 8

Write a generator that combines lines of text into paragraphs, separated by blank lines.
It will receive an iterables of lines (e.g. a file) as an argument, and should yield a whole paragraph (as strings with newlines).
[Bonus Ex: use textwrap.wrap() to reformat paragraphs produced by this generator.]

Chapter 7: Modules

==========

Ex 9

Write a module that implements a “hangman” game.
It starts with a secret word (e.g. “banana”), and the module will have one public function: Guess (letter) that will either say there is no such letter, or reveal it in the word:

```
>>> import hangman
??????
>>> guess('a')
?a?a?a
>>> guess('s')
No such letter
?a?a?a
>>> guess('n')
?anana
```

The original game limits the number of mistakes you can make – ignore it for now.

For simplicity, the module should implement just a single game, initialized with a constant word. It should use global variables to store state between guess() calls.
Yes, this is bad design – we'll fix all this when we learn classes.

Hint: for simpler design, have a separate function that prints the partially-revealed "?a?a?a" string based on the current state.

Chapter 8: Classes

Ex 10
Rewrite your code from exercise 9 as a Hangman class.
It should take the secret word as a constructor argument.
It should store all state on the instance, so that you may play several totally separate games with several instances.

Ex 11
Subclass the class from Ex 10 and add a counter of the number of mistakes the user made, telling the user when he has lost.
[Bonus: add various user-friendly checks: tell the user when he is trying a letter he tried before, don't let him try guessing after he won/lost, etc…]

Chapter 9: System Programming

Ex 11
Write a script that imports the module with the runs a hangman game.
It should receive the secret word as a command-line argument.
It should ask the user for guesses, exiting when he types "exit".