

# Homework 3 Instructions

The zipped file contains 2 files (besides this one), `HW3.py` and `HW3_cli.py`.

Please review the *About Homework* document for reminders and pointers. When you are finished with the homework, compress the two files **together** (not a folder, just the files) into a zip file named `HW3.zip` and upload it to Canvas.

**Read the instructions VERY carefully!** Read **everything** before you start coding. If you have any questions please email me!

**Part 1.** The file `HW3.py` is a module file with function skeletons in it, just like we've seen before. Each function has documentation (below) about what it is supposed to do and how you can run the functions in the REPL to do manual testing.

**Part 2.** The second file, `HW3_cli.py`, is a skeleton of a CLI program. The program will accept one or more input strings and up to 3 flags to determine what to do with the input strings. The details are described below.

## Part 1 – Details of the functions in 'HW3.py'

Nothing in `HW3.py` should print anything!

### words\_containing (10 points)

Given a sentence (ie, a string) and a lowercase letter, return a list of words from the sentence that contain the letter, ignoring the case of the words in the sentence. Look up methods on the `str` object to get a list of words from the sentence. Remember we know how to test if something (like a set or list) *contains* some element.

**Required:** Use a **list comprehension** to get the final list to return.

It should work like this in the REPL:

```
>>> from HW3 import words_containing
>>> sentence = "Anyone who has never made a mistake has never tried anything new"
>>> words_containing(sentence, 'a')
['Anyone', 'has', 'made', 'a', 'mistake', 'has', 'anything']
>>> words_containing(sentence, 'x')
[]
>>> words_containing('', 'x')
[]
>>> sentence = 'The cow jumped over the moon'
>>> words_containing(sentence, 't')
['The', 'the']
>>> words_containing(sentence, 'o')
['cow', 'over', 'moon']
```

### len\_safe (10 points)

**Return** (do not print) the length of the input object as given by the function `len()`, or return `-1` (the **number** `-1`, not a string) if the object has no length. Use exception handling for this one.

What does it mean for an object to have no length? It is an object for which the `len()` method does not apply. What happens when you try to get the `len` of something that has no `len`? Try doing `len(5)` in the REPL. What happens? We want our `len_safe` function to return `-1` for this situation, instead of getting an error. We need to capture the exception and return `-1` for objects that have no `len`.

It should work like this in the REPL:

```
>>> from HW3 import len_safe
>>> my_dict = {'a': 23, 'b': 8}
>>> len_safe(my_dict)
2
>>> len_safe([]) == 0
True
>>> len_safe(0.25) == -1
True
>>> len_safe(7)
-1
>>> len_safe(None)
-1
>>> len_safe('cat')
3
>>> len_safe('')
0
>>> animals = ['dog', 'cat', 'bird', 'cat', 'fish']
>>> len_safe(animals)
5
>>> import math
>>> len_safe(math.pi)
-1
```

## unique (20 points)

Create a *generator* function that loops over the input iterable sequence, yielding one element at a time, but skipping duplicates. Note the output should be in the *same order* as the input sequence. You may assume that all elements in the input iterable are atomic, namely they are all single objects and not lists or dictionaries, etc. **It must not modify the input sequence object.**

Note that it should be a *generator function*. This means that in the `unique` function, **do not make a new list containing all the unique elements** and *then* produce the output. **Do not use any methods of `dict` or import any libraries.** However, you may want to use a set or list as a helper. For best results, use `yield`.

It should work like this in the REPL:

```
>>> from HW3 import unique
>>> numbers = [4, 5, 2, 6, 2, 3, 5, 8]
>>> nums = unique(numbers)
>>> next(nums)
4
>>> next(nums)
5
>>> next(nums)
2
>>> next(nums)
6
>>> next(nums)
3
>>> next(nums)
8
>>> next(nums)
Traceback (most recent call last):
[...]
```

```

StopIteration
>>>
>>> things = unique(['dog', 'cat', 'bird', 'cat', 'fish'])
>>> next(things)
'dog'
>>> next(things)
'cat'
>>> next(things)
'bird'
>>> next(things)
'fish'
>>>
>>> next(things)
Traceback (most recent call last):
[...]
StopIteration

```

Note: I put the [...] to represent the traceback from Python. Your traceback might be slightly different from mine, so I use that as a placeholder.

You can also test whether you have the correct *values and order* by putting the generator returned from `unique` into a list using the list constructor function. This tells you if it returns the correct values, but it does not tell you if it is an iterator/generator. You need to make sure that you can call `next()` on the object you get back from `unique()` and that you get a `StopIteration` error when all the correct elements have been returned with `next()`.

## Part 2 Details of writing 'HW3\_cli.py'

Use `argparse` to implement the CLI portion of the program so it works as shown here. It is one of the few programs in this course that actually prints output.

Output from the program should look **exactly** like this when you use the `-h` help flag. **Hint:** You get the `-h` help flag for free with `argparse`. You do not have to implement the `-h` flag yourself.

```
$ python HW3_cli.py -h
usage: HW3_cli.py [-h] [-p] [-c] [-l] texts [texts ...]

positional arguments:
  texts                Input strings to operate on

optional arguments:
  -h, --help            show this help message and exit
  -p, --print            Just print the input strings
  -c, --combine          Print input strings combined in a continuous string
  -l, --len             Print the lengths of the input strings

Does print, combine, then len. If no flags given, does nothing.
```

See that description line at the end telling what it does? It is created with the `epilog`.

If no arguments are given at all, it should give an error that the `texts` arguments are required. **Hint:** You get this error for free if you program the `texts` argument correctly.

```
$ python HW3_cli.py
usage: HW3_cli.py [-h] [-p] [-c] [-l] texts [texts ...]
HW3_cli.py: error: the following arguments are required: texts
```

The three flags are explained below. If at least one input string is given, but no flags are given, the program should **do nothing**. Since there are no flags used in the lecture examples, you will need to check the `argparse` documentation (look at the [tutorial here](#)) to find how to implement flag arguments. They are called "Short options" in the documentation. Look for the example of implementing the `-v` "verbose" option.

### Arguments

The flag arguments control what the program does. The flags can be given in any order or combination on the command line, however, these are the rules for implementation: They should be implemented in the order shown below - in other words, the `print` flag (if given) is executed first, then the `combine` flag (if given), then the `len` flag (if given). **Please make all the help strings exactly as shown above in the output from using help, as shown above.** Feel free to copy/paste from this document.

The `-p` or `--print` flag will print out the input strings with spaces in between each string. We know how to make one string from a list of strings with one of the `str` methods.

The `-c` or `--combine` flag will print all the input strings concatenated together. Again, we know how to do that with a string method.

The `-l` or `--len` flag prints out the lengths of each of the input strings. No, you don't need to use `len_safe`, since they are always strings. This is a bit more challenging, but you can figure it out! Options for implementation of `-l` or `--len` include generator expressions, special arguments for the print function, and others.

The program collects the input strings from the command line as positional arguments into the argparse variable `texts`. At least one input string is required. You will need to adjust `nargs` for `texts` to allow one or more input strings - ie, one or more positional arguments are required. See the docs for `nargs` for details.

Examples of how it should work (I added a blank line between some commands for clarity):

```
$ python HW3_cli.py -c These Strings Get Concatenated
TheseStringsGetConcatenated

$ python HW3_cli.py -c -p These Strings Get Printed And Concatenated
These Strings Get Printed And Concatenated
TheseStringsGetPrintedAndConcatenated

$ python HW3_cli.py -l -c -p These Strings Get Printed And Concatenated
These Strings Get Printed And Concatenated
TheseStringsGetPrintedAndConcatenated
5 7 3 7 3 12

$ python HW3_cli.py -c -l --print These Strings Get Printed And Concatenated
These Strings Get Printed And Concatenated
TheseStringsGetPrintedAndConcatenated
5 7 3 7 3 12

$ python HW3_cli.py --len --combine --print These Strings Get Printed And Concatenated
These Strings Get Printed And Concatenated
TheseStringsGetPrintedAndConcatenated
5 7 3 7 3 12

$ python HW3_cli.py --len --combine a b c d e f g
abcdefg
1 1 1 1 1 1 1
$ python HW3_cli.py testing
$ python HW3_cli.py -l testing
7
$ python HW3_cli.py -p testing
testing
$ python HW3_cli.py -c testing
testing
```

**NOTE:** If the next command prompt is on the same line after using `-l` or `--len`, as in the following example, then you are not quite done:

```
(myvenv) $ python HW3_cli.py --len some text strings are here
4 4 7 3 4 (myvenv) $
```

The next command prompt *should* be on its own line, like this:

```
(myvenv) $ python HW3_cli.py --len some text strings are here
4 4 7 3 4
(myvenv) $
```

**If you are using an IDE (Sublime, PyCharm, VSCode, etc)** that has its own "command window", please do not use it, but use an actual command window to test this homework. Otherwise you might not get the `-l` or `--len` to work properly.

Also, if you get things showing up in **SHOUTY CAPITALS** in the help output, then you need to follow the link above to the argparse tutorial to find how to code a *short option*.

## Grading 100 points total

`HW3.py` totals 40 points. `HW3_cli.py` totals 60 points: Each argument setup is 8 points (32) points). Implementing each flag in order is 6 points (24 points). The Epilog is 4 points.

My email is [dianechen.ucsdext@gmail.com](mailto:dianechen.ucsdext@gmail.com). Please do not hesitate to email me if you have questions.