Python Exercises

In this assignment, you will implement simple functions that work with lists, strings, and text files. My solutions are between one and seven lines of code.

Sean Devlin:

Submission instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Please make sure to have entered your name above.
3. Select Edit -> Clear Output of All Cells. This will clear all the outputs from all cells (but will keep their content).
4. Select Run -> Run All Cells. This will run all the cells in order, and may take several seconds.
5. Once you've rerun everything, select File -> Save and Export Notebook As -> PDF and download a PDF version showing the code and the output of all cells, and save it in the same folder that contains the notebook file.
6. Look at the PDF file and make sure all your solutions and outputs are there, displayed correctly.
7. Submit **both** the PDF and your notebook file .ipynb on Canvas. If you encounter issues conveting to PDF (make sure 'nbconvert' is installed first), saving as HTML is acceptable too (PDF is preferable).
8. Make sure your your Canvas submission contains the correct files by downloading it after posting it on Canvas.

Virtual Environments and Packages

It is recommended that you install the packages required for this class in a *virtual environment*, for which I recommend the Python virtual environment tool `venv` (other package managers exist that you can use, such as 'conda'). To use `venv` (to manage the environment) and `pip` (to install packages) please see this quick guide. You can first create a folder named "itcs4101" for the course, inside which you will place all your homework assignment folders. A prototypical setup in a Unix-based system looks like this:

1. *mkdir itcs4101* # this creates the course folder.

2. *cd itcs4101* # this changes the current directory to that folder.

3. *python3 -m venv .venvnlp* # this creates the virtual environment named 'venlp' (you can

use any name you want).

4. *source .venvnlp/bin/activate* # this activates the virtual environment 'venvnlp'. After this, anything you isntall with pip will be installed in this environment.
5. *python3 -m pip install --upgrade pip* # this installs the 'pip' package manager.
6. *pip install jupyterlab* # this installs the 'jupyterlab' package.
7. *pip install nbconvert* # this isntalls the 'nbconvert' package, needed to convert notebooks to PDF.

We will use 'jupyterlab' to edit the notebooks for the homework assignments. You can similarly use the 'pip' command (inside the activated '.venvnlp' environment) to install other packages required for homework assignments.

Working with sequences

Simple sum (10p)

Write a function `mysum(l)` that takes as input a list `l` and outputs the sum of all the elements in the list. Do not use the predefined `sum()` function, implement it yourself.

In [1]:
```
def mysum(l):
    #YOUR CODE HERE


    result = 0
    for num in l:
        result += num


    return result

# This call should return 45
mysum(range(10))
```

Out[1]: 45

Complex sum (10p)

Write a function `sum_list(l)` that takes as input a list `l` and outputs another list `q` of the same length as `l` and that contains at position `j` the sum of all the elements from positions `0, 1, ...j`, meaning `q[j] = l[0] + l[1] + ... + l[j]`.

In [2]:
```
def sum_list(l):
    q = []

    # YOUR CODE HERE
```

```
    for i in range(len(l)):
        q.append(l[0])
        for j in range(1, i+1):
            q[i] += l[j]




    return q

# This call should return [1, 6, 4, 7]
sum_list([1, 5, -2, 3])
```

Out[2]: [1, 6, 4, 7]

Generic function (10p)

Does your function `sum_list(l)` work with strings too? If not, change it so that the function works with **both** numbers and strings.

In [3]:
```
# This call should return ['u', 'un', 'unc', 'uncc']
print(sum_list(['u', 'n', 'c', 'c']))

# This call should return [1, 6, 4, 7]
print(sum_list([1, 5, -2, 3]))
```

```
['u', 'un', 'unc', 'uncc']
[1, 6, 4, 7]
```

Working with strings

Lines and tokens (10p)

Write a function `stats(s)` that returns in a tuple the number of lines and the number of tokens in an input string `s`. For this exercise, we consider that a token is any maximal string that does not contain white spaces such as ' ', tabs, or newlines.

Hint: you can use the string methods `splitlines()` and `split()`.

In [4]:
```
def stats(s):
    clines, ctokens = 0, 0

    # YOUR CODE HERE
    clines = len(s.splitlines())
    ctokens = len(s.split())



    return clines, ctokens

# Note how strings can be written on multiple lines in Python code.
s = 'There\'s a passage in the Principia Discordia where Malaclypse complain
```

```
    'about the evils of human society. "Everyone is hurting each other, the
    'with injustices, whole societies plunder groups of their own people, mo
    'children perish while brothers war."\n' \
    '\n' \
    'The Goddess answers: "What is the matter with that, if it\'s what you w
    '\n' \
    'Malaclypse: "But nobody wants it! Everybody hates it!"\n' \
    '\n' \
    'Goddess: "Oh. Well, then stop."\n' \
    'https://slatestarcodex.com/2014/07/30/meditations-on-moloch/'

# This will display the string value.
print(s)

# This should display 8 lines and 76 tokens.
lines, tokens = stats(s)
print()
print('There are', lines, 'lines and', tokens, 'tokens.')
```

```
There's a passage in the Principia Discordia where Malaclypse complains to t
he Goddess about the evils of human society. "Everyone is hurting each other
, the planet is rampant with injustices, whole societies plunder groups of t
heir own people, mothers imprison sons, children perish while brothers war."

The Goddess answers: "What is the matter with that, if it's what you want to
do?"

Malaclypse: "But nobody wants it! Everybody hates it!"

Goddess: "Oh. Well, then stop."
https://slatestarcodex.com/2014/07/30/meditations-on-moloch/

There are 8 lines and 76 tokens.
```

Character substitutions (10p)

Write a function `letterize(s)` that takes as input a string `s` and returns another string that is a copy of `s` where all non-alphabet characters are replaced with the whitespace character `' '`.

Hint: you can use the string methods `isalpha()`.

In [5]:
```
def letterize(s):
    result = ''

    # YOUR CODE HERE
    for char in s:
        if char.isalpha():
            result += char
        else:
            result += ' '
```

```
    return result

r = letterize(s)
print(r)

# The length of the result should be 537.
print(len(r))
```

```
There s a passage in the Principia Discordia where Malaclypse complains to t
he Goddess about the evils of human society  Everyone is hurting each other
the planet is rampant with injustices  whole societies plunder groups of the
ir own people  mothers imprison sons  children perish while brothers war
The Goddess answers  What is the matter with that  if it s what you want to
do  Malaclypse  But nobody wants it  Everybody hates it  Goddess  Oh
Well  then stop  https  slatestarcodex com  meditations on moloc
h
537
```

Token substitutions (10p)

Write a function `substitute(text, source, target)` that replaces each occurrence of the `source` string in `text` with the `target` string and returns the result. You are not allowed to use the `replace()` function, implement this yourself, e.g. using `find()`.

In [6]:
```
def substitute(text: str, source: str, target: str):
    result = text

    # YOUR CODE HERE
    index = result.find(source)
    while index != -1:

        result = result[:index] + target + result[index+len(source):]

        index = result.find(source)


    return result

# Note how strings can be written on multiple lines in Python code.
text = 'There\'s a passage in the Principia Discordia where Malaclypse compl
        'about the evils of human society. "Everyone is hurting each other,
        'with injustices, whole societies plunder groups of their own people
        'children perish while brothers war."\n' \
        '\n' \
        'The Goddess answers: "What is the matter with that, if it\'s what y
        '\n' \
        'Malaclypse: "But nobody wants it! Everybody hates it!"\n' \
        '\n' \
        'Goddess: "Oh. Well, then stop."\n'
print(substitute(text, 'Malaclypse', 'Mazikeen'))
```

```
There's a passage in the Principia Discordia where Mazikeen complains to the
Goddess about the evils of human society. "Everyone is hurting each other, t
he planet is rampant with injustices, whole societies plunder groups of thei
r own people, mothers imprison sons, children perish while brothers war."

The Goddess answers: "What is the matter with that, if it's what you want to
do?"

Mazikeen: "But nobody wants it! Everybody hates it!"

Goddess: "Oh. Well, then stop."
```

Swapping tokens (10p)

Write a function `swap(text, source, target)` that replaces each occurrence of the `source` string in `text` with the `target` string and each `target` string with the `source` string.

In [7]:
```python
def swap(text, source, target):
    result = ''

    # YOUR CODE HERE
    i = 0
    src_i = text.find(source)
    tgt_i = text.find(target)
    while i < len(text):
        if i == src_i:
            result += target
            i += len(source)
            src_i = text.find(source, src_i+1)
        elif i == tgt_i:
            result += source
            i += len(target)
            tgt_i = text.find(target, tgt_i+1)
        else:
            result += text[i]
            i += 1



    return result

# Note how strings can be written on multiple lines in Python code.
text = 'There\'s a passage in the Principia Discordia where Malaclypse compl
        'about the evils of human society. "Everyone is hurting each other,
        'with injustices, whole societies plunder groups of their own people
        'children perish while brothers war."\n' \
        '\n' \
        'The Goddess answers: "What is the matter with that, if it\'s what y
        '\n' \
        'Malaclypse: "But nobody wants it! Everybody hates it!"\n' \
        '\n' \
        'Goddess: "Oh. Well, then stop."\n'
```

```
print(swap(text, 'Malaclypse', 'Goddess'))
```

```
There's a passage in the Principia Discordia where Goddess complains to the
Malaclypse about the evils of human society. "Everyone is hurting each other
, the planet is rampant with injustices, whole societies plunder groups of t
heir own people, mothers imprison sons, children perish while brothers war."

The Malaclypse answers: "What is the matter with that, if it's what you want
to do?"

Goddess: "But nobody wants it! Everybody hates it!"

Malaclypse: "Oh. Well, then stop."
```

Working with text files

Lines and paragraphs (20p)

Write a function `text_stats(fname)` that read a text file **line by line** and returns a tuple containing the following elements:

1. The total number of *lines* in the file.
2. The total number of *text lines* in the file.
   - A *text line* is defined as a line that contains at least one non white space character. For example, a line that contains two white spaces followed by a tab character is not considered a text line.
   - An *empty line* is a line that is not a text line.
3. The total number of *text paragraphs* in the file.
   - A *text paragraph* is a maximal sequence of text lines. Thus, a text paragraph (a) must be preceded by an empty line or the beginning of the file, (b) must be followed by an empty line or the end of the file, and (c) must not contain any empty lines.

In [8]:
```
def text_stats(fname):
    # YOUR CODE HERE
    with open(fname) as file:
        lines = file.readlines()

    text_lines = 0
    text_paras = 0
    previous_line_text = False
    for line in lines:
        if not line.isspace():
            text_lines += 1

            #each paragraph is represented by a transition from whitespace l
            if not previous_line_text:
```

```
                text_paras += 1

            previous_line_text = True
        else:
            previous_line_text = False


    return len(lines), text_lines, text_paras

# This call should return a tuple that specifies 6 paragraphs.
print(text_stats('../data/colorless.txt'))
```

```
(26, 16, 6)
```

Flatten lists (10p)

Write a function `flatten(l)` that takes as input a *deep* list that may contain other lists that may contain other lists ... and returns a *shallow* list that contains just the atmoic elements of the original list.

In [9]:
```
def flatten(l):
    result = []

    # YOUR CODE HERE
    for ele in l:
        if not isinstance(ele, list):
            result.append(ele)
        else:
            result.extend(flatten(ele))

    return result

l = [1, [2, 3], [4, [5, 6, [7, [8, 9]]], 10], 11, 12]

# This call should print [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
print(flatten(l))
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

Bonus points: One liner (5p)

Define a function `sum_elements(l1, l2)` that adds the two lists given as arguments, element-wise. Assume the two lists have the same length. You should do this with only one line of code.

Hint: use list comprehensions.

In [10]:
```
def sum_elements(l1, l2):
    # YOUR CODE HERE
    return [l1[i] + l2[i] for i in range(len(l1))]

# This call should return [3, 5, 7, 9].
sum_elements([1, 2, 3, 4], [2, 3, 4, 5])
```

Out[10]: [3, 5, 7, 9]

Bonus points: Brute force sorting (10p)

Define a function `force_sort(l)` that takes as input a list of integers `l` and outputs a list of the same integers in sorted order. The function should use a brute-force sorting algorithm, meaning that it generates permutations of the initial list until it finds a permutation that is sorted. You should write code to generate permutations yourself (do not use library functions for that). Write a separate function `check_sorted(l)` that returns `True` if and only if the input list is sorted.

In [11]:
```
def check_sorted(l):
    # YOUR CODE HERE
    for i in range(len(l) - 1):
        if l[i] > l[i+1]:
            return False

    return True


def force_sort(l):
    # YOUR CODE HERE
    perms = generate_permutations(l)
    for perm in perms:
        if check_sorted(perm):
            return perm




    return []

def generate_permutations(l):
    if len(l) == 0:
        return []
    if len(l) == 1:
        return [l]

    result = []
    for i in range(len(l)):
        elem = l[i]
        remaining = l[:i] + l[i+1:]
        for p in generate_permutations(remaining):
            result.append([elem] + p)

    return result
```

```
# This call should return [-1, 3, 5, 9, 13].
print(force_sort([9, 13, -1, 5, 3]))

# This call should return False.
print(check_sorted([-1, 3, 13, 5, 9]))
```

```
[-1, 3, 5, 9, 13]
False
```

Bonus points: Matrix CSP (15p)

Implement a function `matrix_csp(N)` that returns a square N by N matrix that contains all integers between 0 and $N \times N - 1$ arranged such that the sum of any 3 consecutive entries (horizontally or vertically) is not divisible by 3. The function should be:

- Correct for any value of N (8 points)
- What is its worst case time complexity? (2 points)
- As efficient as possible (5 points)

In [12]:
```
import numpy as np

def matrix_csp(N):
    numbers = np.arange(N * N)
    np.random.shuffle(numbers)
    M = numbers.reshape(N, N)

    # YOUR CODE HERE
    while check_csp(M):
        np.random.shuffle(numbers)
        M = numbers.reshape(N, N)

    return M

def check_csp(matrix: np.matrix):
    print(f"checking matrix {matrix}")

    N = matrix.shape[0]
    for i in range(N-2):
        vert_slice = matrix[:,i:i+3]
        horizontal_slice = matrix[i:i+3,:]

        slices = np.vstack((vert_slice, horizontal_slice.T))
        sums = np.sum(slices, axis=1) % 3
        print(slices)
        print(sums)
        if np.isin(sums, 0).any():
            return False
```

```
    return True

#I was unable to find a concise solution, but I did implement an algorithm t
```