ME 5194: HW 3

Adrian Bakhtar

CHAPTER 7

Exercise 7.1

Copy the loop from Section 7.5 and encapsulate it in a function called mysqrt that takes a as a parameter, chooses a reasonable value of x, and returns an estimate of the square root of a.

In [83]:
```
import math
from tabulate import tabulate

def mysqrt(a,x,epsilon):
    '''
    Estimates the square root of 'a' using an estimate 'x' and threshold epsilon

    Parameters
    ----------
    a : int
        Number whose square root is to be estimated
    x : int/float
        Intitial estimate
    epsilon : int/float
        Threshold that determines how close is close enough

    Returns
    -------
    y : int/float
        Estimated square root of 'a' based of 'x' and epsilon

    '''
    while True:
        y = (x + a / x) / 2
        if abs(y - x) < epsilon:
            return y
        x = y

def sqrt_compare(a,x, **kwargs):
    '''
    Tabulates the number 'a', the square root of 'a' based of mysqrt(), the
    square root of 'a' using the math module, and the difference between the
    two square roots.

    Parameters
    ----------
    a : int
        Number whose square root is to be estimated
    x : int/float
        Intitial estimate
    **kwargs :
        epsilon : int/float
```

```
            Threshold that determines how close is close enough


    Returns
    -------
    None.

    '''
    default_kwargs = {'epsilon': 0.0000001}
    kwargs = { **default_kwargs, **kwargs }

    data = []
    headers = ["a", "mysqrt(a)", "math.sqrt(a)", "diff"]
    for val in a:
        mysqrt_vals = mysqrt(val,x, epsilon = kwargs['epsilon'])
        math_sqrt_vals = math.sqrt(val)
        diff = abs(mysqrt_vals - math_sqrt_vals)
        data.append([val, mysqrt_vals, math_sqrt_vals, diff])
    print(tabulate(data, headers))

data = range(1,10)
sqrt_compare(data,100, epsilon = 0.1)
print('\n\n')
sqrt_compare(data,100, epsilon = 0.0000001)
```

```
  a    mysqrt(a)    math.sqrt(a)         diff
---  -----------  --------------  -----------
  1      1.00007             1   7.14039e-05
  2      1.41624             1.41421  0.00202777
  3      1.73254             1.73205  0.000487416
  4      2.00014             2        0.000142662
  5      2.23612             2.23607  4.75989e-05
  6      2.44951             2.44949  1.74728e-05
  7      2.64576             2.64575  6.90472e-06
  8      2.82843             2.82843  2.89495e-06
  9      3                   3        1.27463e-06



  a    mysqrt(a)    math.sqrt(a)         diff
---  -----------  --------------  -----------
  1      1                   1   0
  2      1.41421             1.41421  2.22045e-16
  3      1.73205             1.73205  1.33227e-15
  4      2                   2        0
  5      2.23607             2.23607  0
  6      2.44949             2.44949  0
  7      2.64575             2.64575  0
  8      2.82843             2.82843  4.44089e-16
  9      3                   3        0
```

Exercise 7.2

The built-in function *eval* takes a string and evaluates it using the Python interpreter.

Write a function called *eval_loop* that iteratively prompts the user, takes the resulting input and evaluates it using eval, and prints the result.

It should continue until the user enters 'done', and then return the value of the last expression it evaluated.

In [87]:
```
def eval_loop():
    '''
    Iteratively prompts the user for an input, evaluates the input using the eval()
    function, then prints out the result until the user inputs 'done'

    Returns
    -------
    None.

    '''
    current_input = input("Enter a statement: ")
    last_eval = ''

    while current_input != 'done':
        current_eval = eval(current_input); last_eval = current_eval
        print(current_eval)
        current_input = input("Enter a statement: ")

    print(last_eval)

eval_loop()
```

```
Enter a statement: 1+23
24
Enter a statement: 43*123
5289
Enter a statement: 5/123
0.04065040650406504
Enter a statement: (1,2,3,4,5,6)
(1, 2, 3, 4, 5, 6)
Enter a statement: done
(1, 2, 3, 4, 5, 6)
```

CHAPTER 8

Exercise 8.3

A string slice can take a third index that specifies the “step size”; that is, the number of spaces between successive characters. A step size of 2 means every other character; 3 means every third, etc.

A step size of -1 goes through the word backwards, so the slice [::-1] generates a reversed string.

Use this idiom to write a one-line version of is_palindrome from Exercise 6.3.

In [88]:
```
def one_line_is_palindrome(word):
    '''
    Checks to see if the user's word is a palindrome or not

    Parameters
    ----------
    word : str
        Word that the user wants to check

    Returns
    -------
```

```
    bool
        Returns True if word is palindrome, False if not

    '''
    return word == word[::-1]

one_line_is_palindrome('book')
```

Out[88]: False

In [89]:
```
one_line_is_palindrome('mom')
```

Out[89]: True

Exercise 8.5

A Caesar cypher is a weak form of encryption that involves “rotating” each letter by a fixed number of places. To rotate a letter means to shift it through the alphabet, wrapping around to the beginning if necessary, so ‘A’ rotated by 3 is ‘D’ and ‘Z’ rotated by 1 is ‘A’.

To rotate a word, rotate each letter by the same amount. For example, “cheer” rotated by 7 is “jolly” and “melon” rotated by -10 is “cubed”. In the movie 2001: A Space Odyssey, the ship computer is called HAL, which is IBM rotated by -1.

Write a function called rotate_word that takes a string and an integer as parameters, and returns a new string that contains the letters from the original string rotated by the given amount. You might want to use the built-in function ord, which converts a character to a numeric code, and chr, which converts numeric codes to characters.

In [94]:
```
def letter_check(letter, rot):
    '''
    Takes a single letter and rotates it by a user-defined amount. Ensures that
    the rotated letter stays in between the ord() bounds for A-Z or a-z depending
    on if the letter was uppercase or lowercase

    Parameters
    ----------
    letter : str
        Single letter, can be upper or lowercase
    rot : int
        Desired rotation amount, can be positive or negative

    Returns
    -------
    new_ord : int
        Unicode character of rotated letter

    '''
    new_ord = ord(letter) + rot

    if letter.isupper():
        if new_ord > ord('Z'):
            new_ord = ord('A') + (new_ord - ord('Z') - 1)
        elif new_ord < ord('A'):
            new_ord = ord('Z') - (ord('A') - new_ord - 1)
```

```
    elif letter.islower():
        if new_ord > ord('z'):
            new_ord = ord('a') + (new_ord - ord('z') - 1)
        elif new_ord < ord('a'):
            new_ord = ord('z') - (ord('a') - new_ord - 1)

    return new_ord



def rotate_word(word, rot):
    '''
    Performs a Caesar cypher on the word argument based off the rotation
    argument.


    Parameters
    ----------
    word : string
        Word that the user wants to be encrypted.
    rot : int
        Desired Caesar Cypher rotation, can be positive or negative

    Returns
    -------
    new_word : str
        New, Caesar Cypher encrypted word

    '''
    characters = []
    for i in word:
        characters.append(letter_check(i, rot))


    new_char = [chr(j) for j in characters]
    new_word = "".join(new_char)

    return new_word
```

In [96]:
```
word = 'test'
rotate_word(word, -10)
```

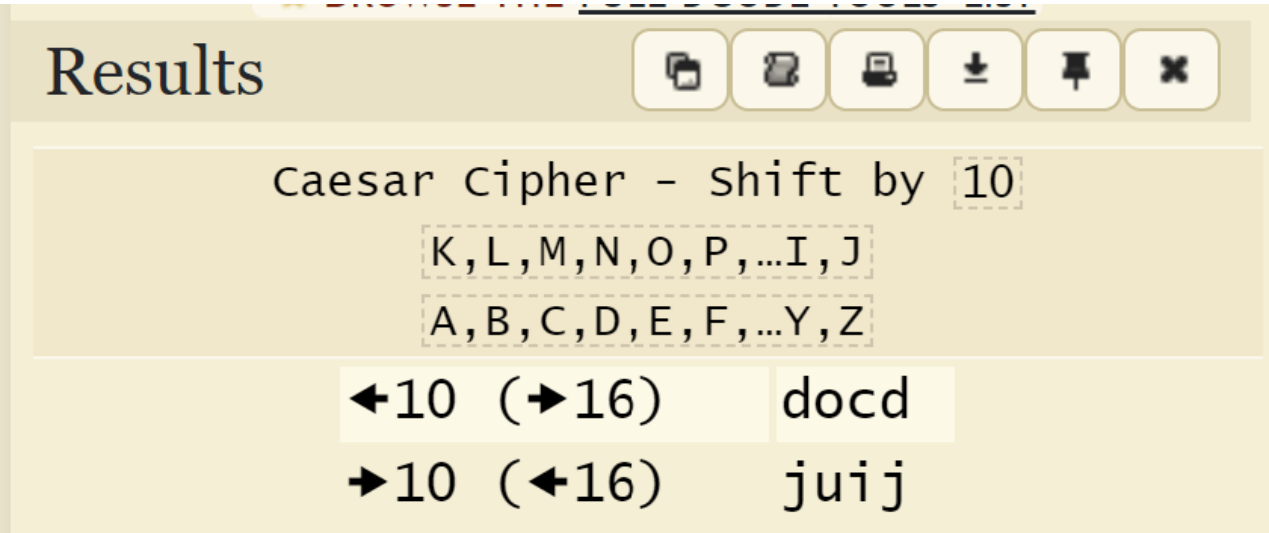
Out[96]: 'juij'

In [100…
```
rotate_word(word, 10)
```

Out[100… 'docd'

In [101…
```
from IPython.display import Image
Image("C:/Users/Adrian Bakhtar/Documents/Smart Products/caesar cipher verification.png"
```

Out[101…



CHAPTER 9

Exercise 9.3 - Finished besides smallest combo of letters

Write a function named avoids that takes a word and a string of forbidden letters, and that returns True if the word doesn't use any of the forbidden letters.

In [103…

```
def avoids(word, forbidden_l):
    '''
    Checks to see if any of the user's word avoided all of the user's forbidden
    letters

    Parameters
    ----------
    word : str
        The word that the user wants to check.
    forbidden_l : str
        The forbidden letters, not separated by any spaces

    Returns
    -------
    bool
        Returns True if the word avoided the forbidden letters, False if not

    '''
    for letter in forbidden_l:
        if letter in word:
            return False
    return True
```

In [104…

```
avoids('super duper', 'azjkb')
```

Out[104… True

In [105…

```
avoids('super duper', 'uper')
```

Out[105… False

Write a program that prompts the user to enter a string of forbidden letters and then prints the number of words that don't contain any of them. Can you find a combination of 5 forbidden letters that excludes the smallest number of words?

In [76]:
```
fin = open("C:/Users/Adrian Bakhtar/Documents/Smart Products/words.txt")
forbidden_l = input('Enter a string of 5 forbidden letters: ')

if len(forbidden_l)!= 5:
    print("Please enter exactly 5 letters without spaces.\n\n")
else:
    for words in fin:
        if avoids(words, forbidden_l):
            print(words)

fin.close()
```

```
Enter a string of 5 forbidden letters: aeiou
by

byrl

byrls

bys

crwth

crwths

cry

crypt

crypts

cwm

cwms

cyst

cysts

dry

dryly

drys

fly

flyby

flybys

flysch

fry

ghyll

ghylls
```

```
glycyl
glycyls
glyph
glyphs
gym
gyms
gyp
gyps
gypsy
hymn
hymns
hyp
hyps
lymph
lymphs
lynch
lynx
my
myrrh
myrrhs
myth
myths
nth
nymph
nymphs
phpht
pht
ply
pry
psst
psych
psychs
```

```
pygmy
pyx
rhythm
rhythms
rynd
rynds
sh
shh
shy
shyly
sky
sly
slyly
spry
spryly
spy
sty
stymy
sylph
sylphs
sylphy
syn
sync
synch
synchs
syncs
syzygy
thy
thymy
try
tryst
trysts
tsk
```

```
tsks

tsktsk

tsktsks

typp

typps

typy

why

whys

wry

wryly

wych

wynd

wynds

wynn

wynns

xylyl

xylyls

xyst

xysts
```

Exercise 9.6

Write a function called is_abecedarian that returns True if the letters in a word appear in alphabetical order (double letters are ok). How many abecedarian words are there?

In [106…

```
def  is_abecedarian(word):
    '''
    Determines whether the letters in a word appear in alphabetical order

    Parameters
    ----------
    word : str
        Word that the user wants to check

    Returns
    -------
    bool
        True if the letters of the word are in alphabetical order, False if not

    '''
```

```
    alphabetized = "".join(sorted(word))
```

In [122…
```
fin = open("C:/Users/Adrian Bakhtar/Documents/Smart Products/words.txt")
word_list = ['book', 'best', 'bed', 'abcdefg', 'aslfkj', 'klmnop']
counter = 0
total_words = 0

for word in fin:
    total_words +=1
    word = word.strip()
    if is_abecedarian(word):
        counter+=1

print('There are {} abecedarian words out of the {} total words in the words.txt file'.
fin.close()
```

```
There are 596 abecedarian words out of the 113783 total words in the words.txt file
```

In [108…
```
word_list = ['book', 'best', 'bed', 'abcdefg', 'aslfkj', 'klmnop', 'aabbcc']
counter = 0

for words in word_list:
    if is_abecedarian(words):
        counter+=1
        print(words)
print('There are {} abecedarian words in the inputted list.'.format(counter))
```

```
best
abcdefg
klmnop
aabbcc
There are 4 abecedarian words in the inputted list.
```

Exercise 9.9

Here's another Car Talk Puzzler you can solve with a search (http://www.cartalk.com/content/puzzlers):

"Recently I had a visit with my mom and we realized that the two digits that make up my age when reversed resulted in her age. For example, if she's 73, I'm 37. We wondered how often this has happened over the years but we got sidetracked with other topics and we never came up with an answer.

"When I got home I figured out that the digits of our ages have been reversible six times so far. I also figured out that if we're lucky it would happen again in a few years, and if we're really lucky it would happen one more time after that. In other words, it would have happened 8 times over all. So the question is, how old am I now?"

Write a Python program that searches for solutions to this Puzzler. Hint: you might find the string method zfill useful.

In [114…
```
def int_to_str_fill(int1, int2):
    '''
    Takes two integer arguments and returns them as a string with
    a leading zero if the int argument was a single digit
```

```
    Parameters
    ----------
    int1 : int
        First integer value.
    int2 : int
        Second integer value.

    Returns
    -------
    str1 : str
        Filled, converted string of int1
    str2 : str
        Filled, converted string of int2

    '''

    str1 = str(int1).zfill(2)
    str2 = str(int2).zfill(2)

    return str1, str2

def reset_age(age1, age2):
    '''
    Takes two ages and decrements each one until either age is equal to 1.

    Parameters
    ----------
    age1 : int
        Age of first person
    age2 : int
        Age of second person.

    Returns
    -------
    age1 : str
        Decremented age of first person.
    age2 : str
        Decremented age of second person.

    '''
    while (age1!=1) and (age2!=1):
        age1 -=1
        age2 -=1

    age1, age2 = int_to_str_fill(age1, age2)

    return age1, age2

def reverse_age_counter(age1, age2):
    '''
    Takes the decremented ages as arguments, increments both ages until one
    reaches age 100, and counts every time that the two ages are reverses of each
    other.

    Parameters
    ----------
    age1 : str
        Decremented age of first person.
    age2 : str
        Decremented age of second person.
```

```
    Returns
    -------
    age_counter : int
        The number of times the two ages were reverses of each other

    '''
    age_counter = 0
    (re_age1, re_age2) = reset_age(age1,age2)
    print('Deaged1: {}    Deaged2: {}\n\n'.format(re_age1, re_age2))
    print('\t\t\t\tReversible ages: \n')
    while (int(re_age1)<=100) and (int(re_age2) <=100):

        if re_age1 == re_age2[::-1]:
            print('First Age: {}\t\tSecond Age: {}\t\tReverse of Second Age: {}'.format
            age_counter +=1

        re_age1, re_age2 = int_to_str_fill((int(re_age1)+1), (int(re_age2)+1))

    print('\n\nThe ages have been reversilble {} times.\n\n\n\n'.format(age_counter))

    return age_counter
```

In [115…

```
age_1 = 73
age_2 = 37
num_times = reverse_age_counter(age_1, age_2)

num_times = reverse_age_counter(37, 74)
```

```
Deaged1: 37    Deaged2: 01


                                Reversible ages:

First Age: 40           Second Age: 04          Reverse of Second Age: 40
First Age: 51           Second Age: 15          Reverse of Second Age: 51
First Age: 62           Second Age: 26          Reverse of Second Age: 62
First Age: 73           Second Age: 37          Reverse of Second Age: 73
First Age: 84           Second Age: 48          Reverse of Second Age: 84
First Age: 95           Second Age: 59          Reverse of Second Age: 95


The ages have been reversilble 6 times.



Deaged1: 01    Deaged2: 38


                                Reversible ages:



The ages have been reversilble 0 times.
```