Stats 21 - HW 2 - Ethan Warren

The questions have been entered into this document. You will modify the document by entering your code.

Make sure you run the cell so the requested output is visible. Download the finished document as a PDF file. If you are unable to convert it to a PDF, you can download it as an HTML file and then print to PDF.

Homework is an opportunity to practice coding and to practice problem solving. Doing exercises is where you will do most of your learning.

Copying someone else's solutions takes away your learning opportunities. It is also academic dishonesty.

Reading

- Think Python: Chapters 6 through 10

Reading is important! Keep up with the reading. I recommend alternating between reading a chapter and then working on exercises.

Additional recommended reading:

- String methods documentation https://docs.python.org/3/library/stdtypes.html#string-methods

Textbook Chapter 5 Problems

Exercise 5.1

In [12]:

```
import time
```

In [ ]:

```
time.time()
```

Write a function `now()` that reads the current time and prints out the time of day in hours, minutes, and seconds, plus the number of days since the epoch. The function does not need to return a value, just print output to the screen.

The result should look like:

"Current time is: 15:25:47. It has been 18370 days since the epoch."

Use `int()` to drop decimal values. You do not need to try to find the date with years and months.

Tip: build your function incrementally. Start by finding how many days have passed since the epoch. (check your answer at the bottom of the page: https://www.epochconverter.com/seconds-days-since-y0) From there find how many hours, etc. Keep in mind the hours will be UTC time.

In [63]:

```
def now():
    secs = int(time.time())
    days = int(secs/60/60/24)
    r = secs % (60 * 60 * 24)
    hours = int(r/60/60)
    r = r % (60 * 60)
    minutes = int(r/60)
    seconds = r % (60)
    tout = str(hours) + ":" + str(minutes) + ":" + str(seconds)
    print("Current time is: " + tout + ". It has been " + str(days) + " days
```

In [65]:

```
now()
```

```
Current time is: 16:39:32. It has been 19018 days since the epoch.
```

Textbook Chapter 6 Problems

Exercise 6.2

The Ackermann function, A(m, n), is defined:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

See http://en.wikipedia.org/wiki/Ackermann_function . Write a function named `ack` that evaluates the Ackermann function. Use your function to evaluate a few test cases. Don't test with $m \geq 4$ as it grows very fast very quickly.

In [74]:

```
def ack(m,n):
    if m < 0 or n < 0:
        return -1
    if m == 0:
        result = n + 1
    elif n == 0:
        result = ack(m - 1, 1)
    else:
        result = ack(m - 1, ack(m, n - 1))
    return result
```

In [75]:

```
# test case, should be 61
ack(3, 3)
```

Out[75]:

```
61
```

In [73]:

```
# test case, should be 125
ack(3, 4)
```

Out[73]:

```
125
```

Exercise 6.4

A number, `a` , is a power of `b` if it is divisible by `b` and `a/b` is a power of `b` . Write a function called `is_power` that takes parameters `a` and `b` and returns `True` if a is a power of b. Note: you will have to think about the base case.

In [87]:

```
def is_power(a, b):
    if a % b == 0:
        if a / b == 1:
            return True
        else:
            return is_power(a/b, b)
    else:
        return False
```

In [88]:

```
is_power(1024, 2)
```

Out[88]:

```
True
```

In [89]:

```
is_power(6561, 3)
```

Out[89]:

```
True
```

In [90]:

```
is_power(4374, 3)
```

Out[90]:

```
False
```

In [91]:

```
is_power(768, 2)
```

Out[91]:

```
False
```

Exercise 6.5

The greatest common divisor (GCD) of a and b is the largest number that divides both of them with no remainder.

One way to find the GCD of two numbers is based on the observation that if `r` is the remainder when `a` is divided by `b`, then `gcd(a, b) = gcd(b, r)`.

As a base case, we can use `gcd(a, 0) = a`.

Write a function called `gcd` that takes parameters `a` and `b` and returns their greatest common divisor.

In [97]:

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
```

In [102…

```
gcd(21, 7)
```

Out[102…

```
7
```

In [99]:

```
gcd(42, 28)
```

Out[99]:

```
14
```

In [100…

```
gcd(105, 140)
```

Out[100…

```
35
```

Textbook Chapter 7 Problems

Exercise 7.1

Copy the loop from Section 7.5 on square roots and encapsulate it into a function called `mysqrt()` that takes `a` as a parameter. For a starting value `x` use `a/2`. It then iterates through the code to estimate the square root of a value.

Write another function called `test_square_root(start, end)` that will print out a table as shown in the textbook.

In [193…

```
# write your code here
def mysqrt(a):
    x = a/2
    while True:
        y = (x + a/x) / 2
        if x == y:
            break
        x = y
    return x

def test_square_root(start, end):
    print('a', 'mysqrt(a)', 'math.sqrt(a)', 'diff', sep = '\t')
    print('-', '---------', '------------', '----', sep = '\t')
    for a in range(int(start), int(end) + 1):
        mine = mysqrt(a)
        maths = math.sqrt(a)
        diff = abs(mine - maths)
        mine = round(mine, 11)
        maths = round(maths, 11)
        if len(str(mine)) < 9:
            mine = str(mine) + '\t'
        if len(str(maths)) < 9:
            maths = str(maths) + '\t'
        print(a, str(mine), str(maths), diff, sep = '\t')
```

In [194…

```
# test code, do not modify:
test_square_root(1.0, 9.0)
```

```
a       mysqrt(a)       math.sqrt(a)    diff
-       ---------       ------------    ----
1       1.0             1.0             0.0
2       1.41421356237   1.41421356237   2.220446049250313e-16
3       1.73205080757   1.73205080757   0.0
4       2.0             2.0             0.0
5       2.2360679775    2.2360679775    0.0
6       2.44948974278   2.44948974278   0.0
7       2.64575131106   2.64575131106   0.0
8       2.82842712475   2.82842712475   4.440892098500626e-16
9       3.0             3.0             0.0
```

In [191…

```
test_square_root(30, 35)
```

```
a       mysqrt(a)       math.sqrt(a)    diff
-       ---------       ------------    ----
30      5.47722557505   5.47722557505   0.0
31      5.56776436283   5.56776436283   8.881784197001252e-16
32      5.65685424949   5.65685424949   8.881784197001252e-16
33      5.74456264654   5.74456264654   0.0
34      5.83095189485   5.83095189485   0.0
35      5.9160797831    5.9160797831    0.0
```

Textbook Chapter 9 Problems

Exercise 9.1

Download this list of words: http://thinkpython2.com/code/words.txt

Write and run a script that reads `words.txt` and prints out only the words with more than 20 characters (after stripping whitespace).

In [195…

```
fin = open("words.txt")
for line in fin:
    if len(line.strip()) > 20:
        print(line)
```

```
counterdemonstrations

hyperaggressivenesses

microminiaturizations
```

Exercise 9.2

Write a function called `has_no_e` that returns True if the word doesn't have the letter e. You can use any of Pythons availble string methods.

In [196…

```
def has_no_e(text):
    return True if text.count('e') == 0 else False
```

In [197…

```
has_no_e("hello")
```

Out[197…

```
False
```

In [198…

```
has_no_e("quit")
```

Out[198…

```
True
```

With your function, write a script. The script should read the list of words (`words.txt`), print out the number of words that do not have the letter 'e' and the proportion of words that do not have the letter 'e'

In [199…

```
fin = open("words.txt")
total = 0
count = 0
for line in fin:
    if has_no_e(line):
        count += 1
    total += 1
print("There are ",count," words that do not contain an 'e'. This is ",round(
```

```
There are 37621 words that do not contain an 'e'. This is 33.06% of the words
```

Textbook Chapter 10 Problems

Exercise 10.1

Write a function called `nested_sum` that takes a list of lists of integers and adds up the elements from all of the nested lists. For example:

```
t = [[1, 2], [3], [4, 5, 6]]
nested_sum(t)
21
```

You may want to build the function recursively in case there are many levels of nested lists.

You can assume that all elements in any of the nested lists are numeric.

In [236...

```
def denest(l):
    new_list = list()
    for x in l:
        if type(x) == list:
            new_list += x
        else:
            new_list.append(x)
    return new_list

def nested_sum(t):
    for x in t:
        if type(x) == int:
            isallint = True
        elif type(x) == list:
            isallint = False
            break
    if isallint:
        return sum(t)
    else:
        return nested_sum(denest(t))
```

In [238...

```
t = [1, 2]
nested_sum(t)
```

Out[238...

```
3
```

In [239...

```
t = [[1, 2], [3], [4, 5, 6]]
nested_sum(t)
```

Out[239...

```
21
```

In [240...

```
x = [[1, 2, [3]], 4, 5, 6, [7], 8]
nested_sum(x)
```

Out[240...

```
36
```

In [241...

```
t = [[[1, 2, [3]], [4, [5, 6, [7]], 8]]]
nested_sum(t)
```

Out[241...

```
36
```

Exercise 10.2

Write a function called `cumsum` that takes a list of numbers and returns the cumulative sum; that is, a new list where the ith element is the sum of the first i + 1 elements from the original list.

For example:

```
t = [1, 2, 3]
cumsum(t)
[1, 3, 6]
```

You can assume that all elements in the lists are numeric and the list does not contain nested lists.

In [242...

```
def cumsum(t):
    cumlist = list()
    sum = 0
    for x in t:
        sum += x
        cumlist.append(sum)
    return cumlist
```

In [243...

```
cumsum([1, 2, 3, 4])
```

Out[243...

```
[1, 3, 6, 10]
```

In [244...

```
cumsum(range(12))
```

Out[244...

```
[0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66]
```

Exercise 10.6

Two words are anagrams if you can rearrange the letters from one to spell the other. Write a function called `is_anagram` that takes two strings and returns `True` if they are anagrams.

You can remove spaces and convert to lowercase using `string.replace(" ","").lower()`

In [258…

```
def is_anagram(word1, word2):
    # Strip whitespace from words and covert to lowercase
    w1 = word1.replace(" ", "").lower()
    w2 = word2.replace(" ", "").lower()

    # If length of words are not the same they are not anagrams
    if len(w1) != len(w2):
        return False

    # Loop over characters in word 1
    for i in range(0, len(w1)):
        char = w1[i]
        # If word 2 does not contain character words are not anagrams
        if char not in w2:
            return False

        # Otherwise remove the character from word 2 and continue with loop
        idx = w2.index(char)
        w2 = w2[:idx] + w2[1+idx:]
    return True
```

In [254…

```
is_anagram("hello", "o hell")
```

Out[254…

```
True
```

In [255…

```
is_anagram("dormitory" , "dirty room")
```

Out[255…

```
True
```

In [256…

```
is_anagram("dormitory" , "dirty rooms")
```

Out[256…

```
False
```

In [257…

```
is_anagram("astronomers" , "moon starers")
```

Out[257…

```
True
```

Exercise 10.7

Write a function called `has_duplicates` that takes a list and returns `True` if there is any element that appears more than once. It should not modify the original list.

You can assume that the list will not have nested lists.

In [260…

```
def has_duplicates(t):
    l = list(t)
    while len(l) != 0:
        if l.pop(0) in l:
            return True
    return False
```

In [261…

```
has_duplicates(['a','b','c'])
```

Out[261…

```
False
```

In [262…

```
has_duplicates(['a','b','b','c'])
```

Out[262…

```
True
```

In [263…

```
has_duplicates(['a','b','c','a'])
```

Out[263…

```
True
```

Exercise 10.10

To check whether a word is in the word list, you could use the in operator, but it would be slow because it searches through the words in order.

Because the words are in alphabetical order, we can speed things up with a bisection search (also known as binary search). You start in the middle and check to see whether the word you are looking for comes before the word in the middle of the list. If so, you search the first half of the list the same way (perform a bisection search on the first half). Otherwise you search the second half.

Either way, you cut the remaining search space in half. If the word list has 113,809 words, it will take about 17 steps to find the word or conclude that it's not there.

Write a function called `in_bisect` that takes a sorted list and a target word and will returns `True` if the word is in the list and `False` if it's not.

Hint: it's a recursive function.

In [1]:

```python
# Use this function. No need to rewrite it.
def make_word_list():
    """Reads lines from a file and builds a list."""
    t = []
    fin = open('words.txt')
    for line in fin:
        word = line.strip()
        t.append(word)
    return t

t = make_word_list()
```

In [16]:

```python
# define this function
def in_bisect(word_list, word):
    while len(word_list) > 0:
        i = len(word_list)//2
        if word_list[i] == word:
            return True
        if word_list[i] > word:
            word_list = word_list[:i]
        else:
            word_list = word_list[i+1:]
    return False
```

In [19]:

```python
in_bisect(t, "hello")
```

Out[19]:

```
True
```

In [20]:

```python
in_bisect(t, "xyz")
```

Out[20]:

```
False
```

Exercise 10.11

Two words are a "reverse pair" if each is the reverse of the other.

Now that you have the `in_bisect` search, write a script that finds all the reverse pairs in the word list that are 6 letters or longer. (It takes a little bit of time to run.)

In [26]:

```
foundlist = list()
for word in t:
    if len(word) < 6:
        continue
    if word in foundlist:
        continue
    if in_bisect(t, word[::-1]):
        foundlist.append(word[::-1])
        if len(word) > 7:
            print(word, word[::-1], sep = '\t')
        else:
            print(word, word[::-1], sep = '\t\t')
```

```
agenes          senega
animal          lamina
animes          semina
degami          imaged
deified         deified
deifier         reified
deliver         reviled
denier          reined
denies          seined
denned          denned
depots          stoped
derats          stared
dessert         tressed
desserts        stressed
dewans          snawed
dialer          relaid
diaper          repaid
dormin          nimrod
drawer          reward
elides          sedile
eviler          relive
gelder          redleg
halalah         halalah
hallah          hallah
levins          snivel
looter          retool
marram          marram
pupils          slipup
recaps          spacer
redder          redder
redips          spider
redraw          warder
redrawer        rewarder
reflet          telfer
reflow          wolfer
reifier         reifier
reknit          tinker
reknits         stinker
remeet          teemer
```

```
rennet         tenner
repaper        repaper
repins         sniper
reviver        reviver
rotator        rotator
sallets        stellas
scares         seracs
secret         terces
selahs         shales
selles         selles
sememes        sememes
skeets         steeks
sleeps         speels
sleets         steels
sloops         spools
snoops         spoons
spirts         strips
sports         strops
sprits         stirps
struts         sturts
terret         terret
```