HW2

April 21, 2021

1 Stats 21 - HW 2

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.

1.1 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

1.2 Textbook Chapter 5 Problems

1.3 Exercise 5.1

[1]:
```
import time
```

[2]:
```
time.time()
```

[2]:
```
1619039222.3329349
```

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.

[3]:
```
def now():
    t = int(time.time())
    days = str(t // 86400)
    x = t % 86400
    h = str(x // 3600)
    m = str(x // 60 % 60)
    s = str(x % 60)
    hms = h + ":" + m + ":" + s
    print("Current time is: " + hms + ". It has been " + days + " days since␣
 ↪the epoch.")
```

[4]:
```
now()
```

```
Current time is: 21:7:2. It has been 18738 days since the epoch.
```

1.4 Textbook Chapter 6 Problems

1.5 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.

[5]:
```
def ack(m,n):
    if m == 0:
        return n + 1
    elif n == 0:
        return ack(m-1, 1)
    else:
        return ack(m-1, ack(m, n-1))
```

[6]:
```
# test case, should be 61
ack(3, 3)
```

[6]:
```
61
```

[7]:
```
# test case, should be 125
ack(3, 4)
```

[7]:
```
125
```

1.6 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.

[8]:
```
def is_power(a, b):
    if (a == b) or (a == 1):
        return True
    elif a < b:
        return False
    return is_power(a/b, b)
```

[9]:
```
is_power(1024, 2)
```

[9]:
```
True
```

[10]:
```
is_power(6561, 3)
```

[10]:
```
True
```

[11]:
```
is_power(4374, 3)
```

[11]:
```
False
```

[12]:
```
is_power(768, 2)
```

[12]:
```
False
```

1.7 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.

[13]:
```
def gcd(a, b):
    if b == 0:
```

```
        return a
    return gcd(b, a%b)
```

[14]:
```
gcd(21, 7)
```

[14]: 7

[15]:
```
gcd(42, 28)
```

[15]: 14

[16]:
```
gcd(105, 140)
```

[16]: 35

1.8 Textbook Chapter 7 Problems

1.9 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.

[17]:
```
import math
```

[18]:
```
# write your code here
def mysqrt(a):
    x = a/2
    y = (x + a/x) / 2
    while abs(y-x) > 1e-6:
        x = y
        y = (x + a/x) / 2
    return y

def test_square_root(start, end):
    for a in range(int(start), int(end+1)):
        mysq = str(mysqrt(a)).ljust(18)
        mathsq = str(math.sqrt(a)).ljust(18)
        print(a, mysq, mathsq, abs(mysqrt(a)-math.sqrt(a)))
```

[19]:
```
# test code, do not modify:
test_square_root(1.0, 9.0)
```

```
1 1.000000000000001  1.0                1.1102230246251565e-15
2 1.414213562373095  1.4142135623730951 2.220446049250313e-16
3 1.7320508075688772 1.7320508075688772 0.0
```

```
4 2.0                2.0                0.0
5 2.23606797749979   2.23606797749979   0.0
6 2.4494897427831788 2.449489742783178  8.881784197001252e-16
7 2.6457513110646933 2.6457513110645907 1.0258460747536446e-13
8 2.82842712474619   2.8284271247461903 4.440892098500626e-16
9 3.0                3.0                0.0
```

[20]:
```
test_square_root(30, 35)
```

```
30 5.477225575051661  5.477225575051661  0.0
31 5.567764362830022  5.5677643628300215 8.881784197001252e-16
32 5.65685424949238   5.656854249492381  8.881784197001252e-16
33 5.744562646538029  5.744562646538029  0.0
34 5.830951894845301  5.830951894845301  0.0
35 5.916079783099616  5.916079783099616  0.0
```

1.10 Textbook Chapter 9 Problems

1.11 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).

[21]:
```
fin = open("words.txt")
for line in fin:
    if len(line.strip()) > 20:
        print(line)
```

```
counterdemonstrations

hyperaggressivenesses

microminiaturizations
```

1.12 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.

[22]:
```
def has_no_e(text):
    return 'e' not in text.lower()
```

[23]:
```
has_no_e("hello")
```

[23]: False

[24]:
```
has_no_e("quit")
```

[24]:
```
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'

[25]:
```
fin = open("words.txt")
no_e = 0
for line in fin:
    if has_no_e(line):
        no_e += 1
print(no_e)
print(no_e / len(list(open("words.txt"))))
```

```
37641
0.3307383423103621
```

1.13 Textbook Chapter 10 Problems

1.14 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.

[26]:
```
type([1, 2]) == list
```

[26]:
```
True
```

[27]:
```
def nested_sum(t):
    total = 0
    for x in t:
        if type(x) == list:
            total += nested_sum(x)
        else:
            total += x
    return total
```

[28]:
```
t = [1, 2]
nested_sum(t)
```

[28]: 3

[29]:
```
t = [[1, 2], [3], [4, 5, 6]]
nested_sum(t)
```

[29]: 21

[30]:
```
x = [[1, 2, [3]], 4, 5, 6, [7], 8]
nested_sum(x)
```

[30]: 36

[31]:
```
t = [[[1, 2, [3]], [4, [5, 6, [7]], 8]]]
nested_sum(t)
```

[31]: 36

1.15 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.

[32]:
```
def cumsum(t):
    cumal = 0
    new_t = []
    for i, x in enumerate(t):
        cumal += x
        new_t.append(cumal)
    return new_t
```

[33]:
```
cumsum([1, 2, 3, 4])
```

[33]: [1, 3, 6, 10]

[34]:
```
cumsum(range(12))
```

[34]: [0, 1, 3, 6, 10, 15, 21, 28, 36, 45, 55, 66]

1.16 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()`

[35]:
```
test1 = {"a":4, "b":3}
test2 = {"b":3, "a":4}
test1 == test2
```

[35]: True

[36]:
```
def is_anagram(word1, word2):
    w1 = word1.replace(" ","").lower()
    w2 = word2.replace(" ","").lower()
    if len(w1) != len(w2):
        return False
    dic1 = {}
    dic2 = {}
    for l in w1:
        if l in dic1:
            dic1[l] += 1
        else:
            dic1[l] = 1
    for l in w2:
        if l in dic2:
            dic2[l] += 1
        else:
            dic2[l] = 1
    return dic1 == dic2
```

[37]:
```
is_anagram("hello", "o hell")
```

[37]: True

[38]:
```
is_anagram("dormitory" , "dirty room")
```

[38]: True

[39]:
```
is_anagram("dormitory" , "dirty rooms")
```

[39]: False

[40]:
```
is_anagram("astronomers" , "moon starers")
```

[40]: True

1.17 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.

[41]:
```
def has_duplicates(t):
    return len(set(t)) != len(t)
```

[42]:
```
has_duplicates(['a','b','c'])
```

[42]: False

[43]:
```
has_duplicates(['a','b','b','c'])
```

[43]: True

[44]:
```
has_duplicates(['a','b','c','a'])
```

[44]: True

1.18 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.

[45]:
```
# 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()
```

[46]:
```
1 // 2
```

[46]:
```
0
```

[47]:
```
# define this function
def in_bisect(word_list, word):
    m = len(word_list) // 2
    if len(word_list) == 0:
        return False
    elif word < word_list[m]:
        return in_bisect(word_list[:m], word)
    elif word > word_list[m]:
        return in_bisect(word_list[m+1:], word)
    elif word == word_list[m]:
        return True
    return False
```

[48]:
```
in_bisect(t, "hello")
```

[48]:
```
True
```

[49]:
```
in_bisect(t, "xyz")
```

[49]:
```
False
```

1.19 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.)

[50]:
```
rev_pair = []
for word in t:
    if (len(word) >= 6) and (in_bisect(t, word[::-1])):
        rev_pair.append(word)
```

[51]:
```
rev_pair
```

[51]:
```
['agenes',
 'animal',
 'animes',
 'degami',
 'deified',
 'deifier',
 'deliver',
 'denier',
 'denies',
```

```
'denned',
'depots',
'derats',
'dessert',
'desserts',
'dewans',
'dialer',
'diaper',
'dormin',
'drawer',
'elides',
'eviler',
'gelder',
'halalah',
'hallah',
'imaged',
'lamina',
'levins',
'looter',
'marram',
'nimrod',
'pupils',
'recaps',
'redder',
'redips',
'redleg',
'redraw',
'redrawer',
'reflet',
'reflow',
'reified',
'reifier',
'reined',
'reknit',
'reknits',
'relaid',
'relive',
'remeet',
'rennet',
'repaid',
'repaper',
'repins',
'retool',
'reviled',
'reviver',
'reward',
'rewarder',
```

'rotator',
'sallets',
'scares',
'secret',
'sedile',
'seined',
'selahs',
'selles',
'sememes',
'semina',
'senega',
'seracs',
'shales',
'skeets',
'sleeps',
'sleets',
'slipup',
'sloops',
'snawed',
'sniper',
'snivel',
'snoops',
'spacer',
'speels',
'spider',
'spirts',
'spools',
'spoons',
'sports',
'sprits',
'stared',
'steeks',
'steels',
'stellas',
'stinker',
'stirps',
'stoped',
'stressed',
'strips',
'strops',
'struts',
'sturts',
'teemer',
'telfer',
'tenner',
'terces',
'terret',

```
'tinker',
'tressed',
'warder',
'wolfer']
```