Complete-Python-3-Bootcamp / 03-Methods and Functions / .ipynb_checkpoints / **09-Function Practice Exercises - Solutions-checkpoint.ipynb**

TiVentures improved function exercises

1 contributor

1013 lines (1013 sloc) | 20.3 KB

Function Practice Exercises - Solutions

Problems are arranged in increasing difficulty:

- Warmup - these can be solved using basic comparisons and methods
- Level 1 - these may involve if/then conditional statements and simple methods
- Level 2 - these may require iterating over sequences, usually with some kind of loop
- Challenging - these will take some creativity to solve

WARMUP SECTION:

LESSER OF TWO EVENS: Write a function that returns the lesser of two given numbers *if* both numbers are even, but returns the greater if one or both numbers are odd

```
lesser_of_two_evens(2,4) --> 2
lesser_of_two_evens(2,5) --> 5
```

In [1]:
```
def lesser_of_two_evens(a,b):
    if a%2 == 0 and b%2 == 0:
        return min(a,b)
    else:
        return max(a,b)
```

In [2]:
```
# Check
lesser_of_two_evens(2,4)
```

Out[2]: 2

In [3]:
```
# Check
lesser_of_two_evens(2,5)
```

Out[3]: 5

ANIMAL CRACKERS: Write a function takes a two-word string and returns True if both words begin with same letter

```
animal_crackers('Levelheaded Llama') --> True
animal_crackers('Crazy Kangaroo') --> False
```

In [4]:
```
def animal_crackers(text):
    wordlist = text.split()
    return wordlist[0][0] == wordlist[1][0]
```

In [5]:
```
# Check
animal_crackers('Levelheaded Llama')
```

```
animal_crackers('Levelheaded Llama')
```

Out[5]: True

In [6]:
```
# Check
animal_crackers('Crazy Kangaroo')
```

Out[6]: False

THE OTHER SIDE OF SEVEN: Given a value, return a value that is twice as far away on the other side of 7

```
other_side_of_seven(4) --> 13
other_side_of_seven(12) --> -3
```

In [7]:
```
def other_side_of_seven(num):
    return 7 - 2*(num-7)
```

In [8]:
```
# Check
other_side_of_seven(4)
```

Out[8]: 13

In [9]:
```
# Check
other_side_of_seven(12)
```

Out[9]: -3

LEVEL 1 PROBLEMS

OLD MACDONALD: Write a function that capitalizes the first and fourth letters of a name

```
old_macdonald('macdonald') --> MacDonald
```

Note: `'macdonald'.capitalize()` returns `'Macdonald'`

In [10]:
```
def old_macdonald(name):
    if len(name) > 3:
        return name[:3].capitalize() + name[3:].capitalize()
    else:
        return 'Name is too short!'
```

In [11]:
```
# Check
old_macdonald('macdonald')
```

Out[11]: 'MacDonald'

MASTER YODA: Given a sentence, return a sentence with the

words reversed

```
master_yoda('I am home') --> 'home am I'
master_yoda('We are ready') --> 'ready are We'
```

In [12]:
```
def master_yoda(text):
    return ' '.join(text.split()[::-1])
```

In [13]:
```
# Check
master_yoda('I am home')
```

Out[13]: 'home am I'

In [14]:
```
# Check
master_yoda('We are ready')
```

Out[14]: 'ready are We'

ALMOST THERE: Given an integer n, return True if n is within 10 of either 100 or 200

```
almost_there(90) --> True
almost_there(104) --> True
almost_there(150) --> False
almost_there(209) --> True
```

NOTE: `abs(num)` returns the absolute value of a number

In [15]:
```
def almost_there(n):
    return ((abs(100 - n) <= 10) or (abs(200 - n) <= 10))
```

In [16]:
```
# Check
almost_there(104)
```

Out[16]: True

In [17]:
```
# Check
almost_there(150)
```

Out[17]: False

In [18]:
```
# Check
almost_there(209)
```

Out[18]: True

LEVEL 2 PROBLEMS

LAUGHTER: Write a function that counts the number of times a given pattern appears in a string, *including overlap*

```
laughter('hah','hahahah') --> 3
```

Note that `'hahahah'.count('hah')` only returns 2.

In [19]:
```
def laughter(pattern,text):
    out = 0
    for x in range(len(text)-2):
        if text[x:x+len(pattern)] == pattern:
            out += 1
    return out
```

In [20]:
```
# Check
laughter('hah','hahahah')
```

Out[20]: 3

PAPER DOLL: Given a string, return a string where for every character in the original there are three characters

```
paper_doll('Hello') --> 'HHHeeellllllooo'
paper_doll('Mississippi') --> 'MMMiiissssssiiipppppppiii'
```

In [21]:
```
def paper_doll(text):
    result = ''
    for char in text:
        result += char * 3
    return result
```

In [22]:
```
# Check
paper_doll('Hello')
```

Out[22]: 'HHHeeellllllooo'

In [23]:
```
# Check
paper_doll('Mississippi')
```

Out[23]: 'MMMiiissssssiiissssssiiipppppppiii'

BLACKJACK: Given three integers between 1 and 11, if their sum is less than or equal to 21, return their sum. If their sum exceeds 21 *and* there's an eleven, reduce the total sum by 10. Finally, if the sum (even after adjustment) exceeds 21, return 'BUST'

```
blackjack(5,6,7) --> 18
blackjack(9,9,9) --> 'BUST'
blackjack(9,9,11) --> 19
```

In [24]:
```
def blackjack(a,b,c):
```

```
    if sum((a,b,c)) <= 21:
        return sum((a,b,c))
    elif sum((a,b,c)) <=31 and 11 in (a,b,c):
        return sum((a,b,c)) - 10
    else:
        return 'BUST'
```

In [25]:
```
# Check
blackjack(5,6,7)
```

Out[25]: 18

In [26]:
```
# Check
blackjack(9,9,9)
```

Out[26]: 'BUST'

In [27]:
```
# Check
blackjack(9,9,11)
```

Out[27]: 19

SUMMER OF '69: Return the sum of the numbers in the array, except ignore sections of numbers starting with a 6 and extending to the next 9 (every 6 will be followed by at least one 9). Return 0 for no numbers.

```
summer_69([1, 3, 5]) --> 9
summer_69([4, 5, 6, 7, 8, 9]) --> 9
summer_69([2, 1, 6, 9, 11]) --> 14
```

In [28]:
```
def summer_69(arr):
    total = 0
    add = True
    for num in arr:
        while add:
            if num != 6:
                total += num
                break
            else:
                add = False
        while not add:
            if num != 9:
                break
            else:
                add = True
                break
    return total
```

In [29]:
```
# Check
summer_69([1, 3, 5])
```

Out[29]: 9

In [30]:
```
# Check
summer_69([4, 5, 6, 7, 8, 9])
```

Out[30]: 9

In [31]:
```
# Check
summer_69([2, 1, 6, 9, 11])
```

Out[31]: 14

CHALLENGING PROBLEMS

SPY GAME: Write a function that takes in a list of integers and returns True if it contains 007 in order

```
spy_game([1,2,4,0,0,7,5]) --> True
spy_game([1,0,2,4,0,5,7]) --> True
spy_game([1,7,2,0,4,5,0]) --> False
```

In [32]:
```
def spy_game(nums):

    code = [0,0,7,'x']

    for num in nums:
        if num == code[0]:
            code.pop(0)   # code.remove(num) also works

    return len(code) == 1
```

In [33]:
```
# Check
spy_game([1,2,4,0,0,7,5])
```

Out[33]: True

In [34]:
```
# Check
spy_game([1,0,2,4,0,5,7])
```

Out[34]: True

In [35]:
```
# Check
spy_game([1,7,2,0,4,5,0])
```

Out[35]: False

COUNT PRIMES: Write a function that returns the *number* of prime numbers that exist up to and including a given number

```
count_primes(100) --> 25
```

By convention, 0 and 1 are not prime.

In [36]:
```
def count_primes(num):
    primes = [2]
    x = 3
    if num < 2:  # for the case of num = 0 or 1
        return 0
    while x <= num:
        for y in range(3,x,2):  # test all odd factors up to x-1
            if x%y == 0:
                x += 2
                break
        else:
            primes.append(x)
            x += 2
    print(primes)
    return len(primes)
```

In [37]:
```
# Check
count_primes(100)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]
```

Out[37]: 25

BONUS: Here's a faster version that makes use of the prime numbers we're collecting as we go!

In [38]:
```
def count_primes2(num):
    primes = [2]
    x = 3
    if num < 2:
        return 0
    while x <= num:
        for y in primes:  # use the primes list!
            if x%y == 0:
                x += 2
                break
        else:
            primes.append(x)
            x += 2
    print(primes)
    return len(primes)
```

In [39]:
```
count_primes2(100)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67,
71, 73, 79, 83, 89, 97]
```

Out[39]: 25

Just for fun:

PRINT BIG: Write a function that takes in a single letter, and returns a 5x5 representation of that letter

```
print_big('a')
```

```
out:   *
        * *
       *****
       *   *
       *   *
```

HINT: Consider making a dictionary of possible patterns, and mapping the