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')
```

Out[5]:

```
True
```

In [6]:

```
# Check
animal_crackers('Crazy Kangaroo')
```

Out[6]:

```
False
```

MAKES TWENTY: Given two integers, return True if the sum of the integers is 20 *or* if one of the integers is 20. If not, return False

```
makes_twenty(20,10) --> True
makes_twenty(12,8) --> True
makes_twenty(2,3) --> False
```

In [7]:

```
def makes_twenty(n1,n2):
    return (n1+n2)==20 or n1==20 or n2==20
```

In [8]:

```
# Check
makes_twenty(20,10)
```

Out[8]:

```
True
```

In [9]:

```
# Check
makes_twenty(12,8)
```

Out[9]:

```
True
```

In [10]:

```
#Check
makes_twenty(2,3)
```

Out[10]:

```
False
```

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 [11]:

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

In [12]:

```
# Check
old_macdonald('macdonald')
```

Out[12]:

```
'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 [13]:

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

In [14]:

```
# Check
master_yoda('I am home')
```

Out[14]:

```
'home am I'
```

In [15]:

```
# Check
master_yoda('We are ready')
```

Out[15]:

```
'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 [16]:

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

In [17]:

```
# Check
almost_there(90)
```

Out[17]:

```
True
```

In [18]:

```
# Check
almost_there(104)
```

Out[18]:

```
True
```

In [19]:

```
# Check
almost_there(150)
```

Out[19]:

```
False
```

In [20]:

```
# Check
almost_there(209)
```

Out[20]:

```
True
```

LEVEL 2 PROBLEMS

FIND 33:

Given a list of ints, return True if the array contains a 3 next to a 3 somewhere.

```
has_33([1, 3, 3]) → True
has_33([1, 3, 1, 3]) → False
has_33([3, 1, 3]) → False
```

In [21]:

```
def has_33(nums):
    for i in range(0, len(nums)-1):

        # nicer looking alternative in commented code
        #if nums[i] == 3 and nums[i+1] == 3:

        if nums[i:i+2] == [3,3]:
            return True

    return False
```

In [22]:

```
# Check
has_33([1, 3, 3])
```

Out[22]:

```
True
```

In [23]:

```
# Check
has_33([1, 3, 1, 3])
```

Out[23]:

```
False
```

In [24]:

```
# Check
has_33([3, 1, 3])
```

Out[24]:

```
False
```

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') --> 'MMMiiissssssiiippppppiii'
```

In [25]:

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

In [26]:

```
# Check
paper_doll('Hello')
```

Out[26]:

```
'HHHeeellllllooo'
```

In [27]:

```
# Check
paper_doll('Mississippi')
```

Out[27]:

```
'MMMiiissssssiiissssssiiippppppiii'
```

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 [28]:

```
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 [29]:

```
# Check
blackjack(5,6,7)
```

Out[29]:

```
18
```

In [30]:

```
# Check
blackjack(9,9,9)
```

Out[30]:

```
'BUST'
```

In [31]:

```
# Check
blackjack(9,9,11)
```

Out[31]:

```
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 [32]:

```
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 [33]:

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

Out[33]:

```
9
```

In [34]:

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

Out[34]:

```
9
```

In [35]:

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

Out[35]:

```
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 [36]:

```
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 [37]:

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

Out[37]:

```
True
```

In [38]:

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

Out[38]:

```
True
```

In [39]:

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

Out[39]:

```
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 [40]:

```
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 [41]:

```
# 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[41]:

```
25
```

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

In [42]:

```
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 [43]:

```
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[43]:

```
25
```

Just for fun, not a real problem :)

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 alphabet to specific 5-line combinations of patterns.
For purposes of this exercise, it's ok if your dictionary stops at "E".

In [44]:

```
def print_big(letter):
    patterns = {1:'  *  ',2:' * * ',3:'*   *',4:'*****',5:'**** ',6:'   * ',7:'
 *   ',8:'*   * ',9:'*    '}
    alphabet = {'A':[1,2,4,3,3],'B':[5,3,5,3,5],'C':[4,9,9,9,4],'D':[5,3,3,3,5],
'E':[4,9,4,9,4]}
    for pattern in alphabet[letter.upper()]:
        print(patterns[pattern])
```

In [45]:

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

```
  *
 * *
*****
*   *
*   *
```

Great Job!