(https://www.udemy.com/user/joseportilla/)

Function Practice Exercises

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 [ ]:
```
def lesser_of_two_evens(a,b):
    pass
```

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

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

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 [ ]:
```
def animal_crackers(text):
    pass
```

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

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

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 [ ]:
```
def makes_twenty(n1,n2):
    pass
```

In [ ]:
```
# Check
makes_twenty(20,10)
```

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

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 [ ]:
```
def old_macdonald(name):
    pass
```

In [ ]:
```
# Check
old_macdonald('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'
```

Note: The .join() method may be useful here. The .join() method allows you to join together strings in a list with some connector string. For example, some uses of the .join() method:

```
>>> "--".join(['a','b','c'])
>>> 'a--b--c'
```

This means if you had a list of words you wanted to turn back into a sentence, you could just join them with a single space string:

```
>>> " ".join(['Hello','world'])
>>> "Hello world"
```

In [ ]:
```
def master_yoda(text):
    pass
```

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

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

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 [ ]:
```
def almost_there(n):
    pass
```

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

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

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

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 [ ]:
```
def has_33(nums):
    pass
```

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

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

In [ ]:
```
# Check
has_33([3, 1, 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 [ ]:
```
def paper_doll(text):
    pass
```

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

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

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

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

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

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

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 [ ]:
```
def summer_69(arr):
    pass
```

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

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

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

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 [ ]:
```
def spy_game(nums):
    pass
```

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

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

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

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 [ ]:
```
def count_primes(num):
    pass
```

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

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

In [ ]:
```
def print_big(letter):
    pass
```

In [ ]:
```
print_big('a')
```

Great Job!