Python course materials

# 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

def lesser\_of\_two\_evens(a,b):  
 pass

# Check  
lesser\_of\_two\_evens(2,4)

# 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

def animal\_crackers(text):  
 pass

# Check  
animal\_crackers('Levelheaded Llama')

# 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

def makes\_twenty(n1,n2):  
 pass

# Check  
makes\_twenty(20,10)

# 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'

def old\_macdonald(name):  
 pass

# 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"

def master\_yoda(text):  
 pass

# Check  
master\_yoda('I am home')

# 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

def almost\_there(n):  
 pass

# Check  
almost\_there(104)

# Check  
almost\_there(150)

# 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

def has\_33(nums):  
 pass

# Check  
has\_33([1, 3, 3])

# Check  
has\_33([1, 3, 1, 3])

# 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') --> 'MMMiiissssssiiippppppiii'

def paper\_doll(text):  
 pass

# Check  
paper\_doll('Hello')

# 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

def blackjack(a,b,c):  
 pass

# Check  
blackjack(5,6,7)

# Check  
blackjack(9,9,9)

# 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

def summer\_69(arr):  
 pass

# Check  
summer\_69([1, 3, 5])

# Check  
summer\_69([4, 5, 6, 7, 8, 9])

# 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

def spy\_game(nums):  
 pass

# Check  
spy\_game([1,2,4,0,0,7,5])

# Check  
spy\_game([1,0,2,4,0,5,7])

# 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.

def count\_primes(num):  
 pass

# 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”.

def print\_big(letter):  
 pass

print\_big('a')

## Great Job!