Python course materials

# 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

def lesser\_of\_two\_evens(a,b):  
 if a%2 == 0 and b%2 == 0:  
 return min(a,b)  
 else:  
 return max(a,b)

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

2

# Check  
lesser\_of\_two\_evens(2,5)

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):  
 wordlist = text.split()  
 return wordlist[0][0] == wordlist[1][0]

# Check  
animal\_crackers('Levelheaded Llama')

True

# Check  
animal\_crackers('Crazy Kangaroo')

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

def makes\_twenty(n1,n2):  
 return (n1+n2)==20 or n1==20 or n2==20

# Check  
makes\_twenty(20,10)

True

# Check  
makes\_twenty(12,8)

True

#Check  
makes\_twenty(2,3)

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'

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

# Check  
old\_macdonald('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'

def master\_yoda(text):  
 return ' '.join(text.split()[::-1])

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

'home am I'

# Check  
master\_yoda('We are ready')

'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

def almost\_there(n):  
 return ((abs(100 - n) <= 10) or (abs(200 - n) <= 10))

# Check  
almost\_there(90)

True

# Check  
almost\_there(104)

True

# Check  
almost\_there(150)

False

# Check  
almost\_there(209)

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

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

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

True

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

False

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

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'

def paper\_doll(text):  
 result = ''  
 for char in text:  
 result += char \* 3  
 return result

# Check  
paper\_doll('Hello')

'HHHeeellllllooo'

# Check  
paper\_doll('Mississippi')

'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

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'

# Check  
blackjack(5,6,7)

18

# Check  
blackjack(9,9,9)

'BUST'

# Check  
blackjack(9,9,11)

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

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

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

9

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

9

# Check  
summer\_69([2, 1, 6, 9, 11])

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

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

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

True

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

True

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

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.

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)

# 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]  
  
  
  
  
  
25

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

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)

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]  
  
  
  
  
  
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”.

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])

print\_big('a')

\*   
 \* \*   
\*\*\*\*\*  
\* \*  
\* \*

## Great Job!