**Practice! Practice Practice!**

The best way to get good at anything is a lot of practice. This lesson is full of practice problems for you to work on. Each exercise will contain minimal instructions to help you solve these problems. The goal is to help you take your programming skills and apply them to real life problems.

The more challenging programs will contain some helpful hints to nudge you in the right direction.

**is\_even**

All right! Let's get started.

Remember how an even number is a number that is divisible by 2?

**Instructions**

**1.**

Define a function is\_even that will take a number x as input.

If x is even, then return True.

Otherwise, return False.

Hint

The modulo % operation is useful for determining if one number is divisible by another.

Make sure to return True instead of printing it!

|  |
| --- |
| def is\_even(x):  if x%2 == 0:  return True  else:  return False |

**is\_int**

An integer is just a number without a decimal part (for instance, -17, 0, and 42are all integers, but 98.6 is not).

For the purpose of this lesson, we'll also say that **a number with a decimal part that is all 0s is also an integer**, such as 7.0.

This means that, for this lesson, you can't just test the input to see if it's of type int.

If the difference between a number and that same number rounded is greater than zero, what does that say about that particular number?

**1.**

Define a function is\_int that takes a number x as an input.

Have it return True if the number is an integer (as defined above) and False otherwise.

For example:

is\_int(7.0) # True is\_int(7.5) # False is\_int(-1) # True

|  |
| --- |
| def is\_int(x):  absolute = abs(x)  rounded = round(absolute)  return absolute - rounded == 0 |

**digit\_sum**

Awesome! Now let's try something a little trickier. Try summing the digits of a number.

**1.**

Write a function called digit\_sumthat takes a positive integer n as input and returns the sum of all that number's digits. For example: digit\_sum(1234) should return 10which is 1 + 2 + 3 + 4. (Assume that the number you are given will always be positive.)

Check the hint if you need help!

Hint

One way might be to convert the integer to a string with str(), iterate over it, and turn the substrings back into integers with int() to do the addition.

If you're looking for a challenge, try this: to get the rightmost digit of a number, you can modulo (%) the number by 10. To remove the rightmost digit you can floor divide (//) the number by 10. (Don't worry if you're not familiar with floor division—you can look up the documentation [here](http://docs.python.org/2/reference/expressions.html#binary-arithmetic-operations). Remember, this is a challenge!)

Try working this into a pattern to isolate all of the digits and add them to a total.

|  |
| --- |
| def digit\_sum(x):  sum = 0  for temp in str(x):  sum += int(temp)  return sum |

|  |
| --- |
| def digit\_sum(x):  total = 0  while x > 0:  total += x % 10  x = x // 10  print x  return total |

**factorial**

All right! Now we're cooking. Let's try a factorial problem.

To calculate the factorial of a non-negative integer x, just multiply all the integers from 1 through x. For example:

* factorial(4) would equal 4 \* 3 \* 2 \* 1, which is 24.
* factorial(1) would equal 1.
* factorial(3) would equal 3 \* 2 \* 1, which is 6.

**Instructions**

**1.**

Define a function factorial that takes an integer x as input.

Calculate and return the factorial of that number.

Hint

Consider having factorial() call itself. When the input is 1, your function could just return 1. Otherwise, it could return the number multiplied by factorial(n - 1).

Note that mathematically, factorial(0) is 1.

|  |
| --- |
| def factorial(x):  total = 1  while x>0:  total \*= x  x-=1  return total |

# is\_prime

A prime number is a positive integer greater than 1 that has no positive divisors other than 1 and itself. (That's a mouthful!)

In other words, if you want to test if a number in a variable x is prime, then no other number should go into x evenly besides 1 and x. So 2 and 5 and 11 are all prime, but 4 and 18 and 21 are not.

If there is a number between 1 and x that goes in evenly, then x is not prime.

**1.**

Define a function called is\_primethat takes a number x as input.

For each number n from 2 to x - 1, test if x is evenly divisible by n.

If it is, return False.

If none of them are, then return True.

Hint

Remember: all numbers less than 2 are not prime numbers!

|  |
| --- |
| def is\_prime(x):  if x < 2:  return False  else:  for n in range(2, x):  if x % n == 0:  return False  return True |

# reverse

Great work so far! Let's practice writing some functions that work with strings.

**Instructions**

**1.**

Define a function called reverse that takes a string textand returns that string in reverse. For example: reverse("abcd") should return "dcba".

You may not use reversed or [::-1]to help you with this.

You may get a string containing special characters (for example, !, @, or #).

Hint

Consider how you would loop through text starting from the last character through the first character.

|  |
| --- |
| def reverse(text):  word = ""  l = len(text) - 1  while l >= 0:  word = word + text[l]  l -= 1  return word |

|  |
| --- |
| def reverse(text):  word = ""  for temp in text:  print temp  word = temp + word  print word  return word |

# anti\_vowel

Nice work. Next up: vowels!

**1.**

Define a function called anti\_vowelthat takes one string, text, as input and returns the text with all of the vowels removed.

For example: anti\_vowel("Hey You!")should return "Hy Y!". Don't count Y as a vowel. Make sure to remove lowercase and uppercase vowels.

Hint

To check to see if c is a vowel, you can do: c in "aeiouAEIOU".

|  |
| --- |
| def anti\_vowel(text):  t=""  for c in text:  for i in "ieaouIEAOU":  if c==i:  c=""  else:  c=c  t=t+c  return t |

**scrabble\_score**

Scrabble is a game where players get points by spelling words. Words are scored by adding together the point values of each individual letter (we'll leave out the double and triple letter and word scores for now).

To the right is a dictionary containing all of the letters in the alphabet with their corresponding Scrabble point values.

For example: the word "Helix" would score 15 points due to the sum of the letters: 4 + 1 + 1 + 1 + 8.

**1.**

Define a function scrabble\_scorethat takes a string word as input and returns the equivalent scrabble score for that word.

* Assume your input is only one word containing no spaces or punctuation.
* As mentioned, no need to worry about score multipliers!
* Your function should work even if the letters you get are uppercase, lowercase, or a mix.
* Assume that you're only given non-empty strings.

Hint

Have your function loop through the word that you are given as input and look up the score for each letter in the score dictionary. Add the score for each letter into a total of some sort.

Remember you can use a string's .lower() method to make your string lower case!

|  |
| --- |
| score = {"a": 1, "c": 3, "b": 3, "e": 1, "d": 2, "g": 2,  "f": 4, "i": 1, "h": 4, "k": 5, "j": 8, "m": 3,  "l": 1, "o": 1, "n": 1, "q": 10, "p": 3, "s": 1,  "r": 1, "u": 1, "t": 1, "w": 4, "v": 4, "y": 4,  "x": 8, "z": 10}  def scrabble\_score(word):  total = 0  for temp in word:  temp = temp.lower()  total = total + score[temp]  return total |

**censor**

You're doing great with these string function challenges. Last one!

**Instructions**

**1.**

Write a function called censor that takes two strings, text and word, as input. It should return the text with the word you chose replaced with asterisks. For example:

censor("this hack is wack hack", "hack")

should return:

"this \*\*\*\* is wack \*\*\*\*"

* Assume your input strings won't contain punctuation or upper case letters.
* The number of asterisks you put should correspond to the number of letters in the censored word.

Hint

You can use

string.split() # and " ".join(list)

to help you here.

Remember: "\*" \* 4 equals "\*\*\*\*"

After splitting the string with string.split(), you can loop through the indices in the list and replace the words you are looking for with their asterisk equivalent. Join the list at the end to get your sentence!

|  |
| --- |
| def censor(text, word):  words = text.split()  result = ''  stars = '\*' \* len(word)  count = 0  for i in words:  if i == word:  words[count] = stars  count += 1  result =' '.join(words)  return result |

**count**

Great work so far. Let's finish up by practicing with a few functions that take lists as arguments.

**1.**

Define a function called count that has two arguments called sequenceand item.

Return the number of times the item occurs in the list.

For example: count([1, 2, 1, 1], 1)should return 3 (because 1 appears 3 times in the list).

* There is a list method in Python that you can use for this, but you should do it the long way for practice.
* Your function should return an integer.
* The item you input may be an integer, string, float, or even another list!
* Be careful not to use list as a variable name in your code—it's a reserved word in Python!

Hint

You can set a sum variable inside count. You can then iterate over sequence and increment sum every time you find an element in the sequence that matches item.

|  |
| --- |
| def count(sequence, item):  count = 0  for i in sequence:  if i == item:  count += 1  return count |

# purify

Awesome! Now let's practice filtering a list.

**Instructions**

**1.**

Define a function called purify that takes in a list of numbers, removes all odd numbers in the list, and returns the result. For example, purify([1,2,3]) should return [2].

Do not directly modify the list you are given as input; instead, return a new list with only the even numbers.

Hint

Your code should share something in common with the is\_evenfunction you defined earlier.

|  |
| --- |
| def purify(lst):  res = []  for ele in lst:  if ele % 2 == 0:  res.append(ele)  return res |

**product**

Great! Now let's try a little multiplication.

**1.**

Define a function called product that takes a list of integers as input and returns the product of all of the elements in the list. For example: product([4, 5, 5]) should return 100 (because 4 \* 5 \* 5 is 100).

* Don't worry about the list being empty.
* Your function should return an integer.

Hint

You can use a loop to go through the elements of the list.

It'll probably be useful to use the \*= operator.

Be careful not to start your total at 0, as this would make the overall result of the multiplication equal to 0! (Anything multiplied by zero equals zero.)

|  |
| --- |
| def product(x):  total = 1  for i in x:  total = total \* i  return total |

**remove\_duplicates**

Awesome! Now for something a bit trickier.

**Instructions**

**1.**

Write a function remove\_duplicatesthat takes in a list and removes elements of the list that are the same.

For example: remove\_duplicates([1, 1, 2, 2]) should return [1, 2].

* Don't remove every occurrence, since you need to keep a single occurrence of a number.
* The order in which you present your output does not matter. So returning [1, 2, 3] is the same as returning [3, 1, 2].
* **Do not** modify the list you take as input! Instead, return a new list.

Hint

The easiest way to approach this problem is to create a new list in your function, loop through your input list, and add items from your input list to your new list if the current item is not already contained in your new list. Using the a not in b syntax might help you here.

Also, note that destructively modifying a list while you are looping through it is bad practice and will likely lead to bugs somewhere down the line! That's why we always make a fresh copy to work on.

|  |
| --- |
| def remove\_duplicates(x):  is\_list = []  for i in x:  if i not in is\_list:  is\_list.append(i)  return is\_list |

**median**

Great work! You've covered a lot in these exercises. Last but not least, let's write a function to find the median of a list.

The *median* is the middle number in a sorted sequence of numbers.

Finding the median of [7, 12, 3, 1, 6]would first consist of sorting the sequence into [1, 3, 6, 7, 12] and then locating the middle value, which would be 6.

If you are given a sequence with an **even**number of elements, you must average the two elements surrounding the middle.

For example, the median of the sequence [7, 3, 1, 4] is 3.5, since the middle elements after sorting the list are 3 and 4and (3 + 4) / (2.0) is 3.5.

You can sort the sequence using the sorted() function, like so:

sorted([5, 2, 3, 1, 4]) [1, 2, 3, 4, 5]

**Instructions**

**1.**

Write a function called median that takes a list as an input and returns the median value of the list. For example: median([1, 1, 2]) should return 1.

* The list can be of any size and the numbers are not guaranteed to be in any particular order. Make sure to sort it!
* If the list contains an even number of elements, your function should return the average of the middle two.

Hint

In order to find the median of a list with an even number of elements, you're going to need to find the indices of the middle two elements.

You can find the middle two elements by halving the length of the array to find the index of the first element, and subtracting one from the first index to find the second index.

For example, with an array of length 6 like [0, 1, 2, 3, 4, 5], the two middle elements that need to be averaged in order find the median would be 2 and 3. You get 3 by cutting the length of the array in half and 2 by subtracting 1 from the previous index: 3. You can use a similar pattern to find the middle element of an odd-length list.

Last but not least, note that (2 + 3) / 2 is not the same as (2 + 3) / 2.0! The former is *integer division*, meaning Python will try to give you an integer back. You'll want a float, so something like (2 + 3) / 2.0 is the way to go.

|  |
| --- |
| def median(lst):  sorted\_list = sorted(lst)  if len(sorted\_list) % 2 != 0:  #odd number of elements  index = len(sorted\_list)//2  return sorted\_list[index]  elif len(sorted\_list) % 2 == 0:  #even no. of elements  index\_1 = len(sorted\_list)/2 - 1  index\_2 = len(sorted\_list)/2  mean = (sorted\_list[index\_1] + sorted\_list[index\_2])/2.0  return mean    print median([2, 4, 5, 9]) |

|  |
| --- |
| def median(x):  sorted\_list = sorted(x)  if len(x) % 2 == 0:  count = len(x)/2 - 1  total = (sorted\_list[count] + sorted\_list[count+1])/2.0  else:  count1 = len(x)/2  if len(x)/2 <= 0:  total = sorted\_list[0]  else:  total = sorted\_list[count1]    return total |