Problem 2: Function Fun!

This problem introduces you to some helpful built-in Python functions. Then, you'll write some of your own Python functions.

When you're finished with this assignment, submit your code at the bottom of this page.

Using Built-In Functions

First, you'll try out some of Python's built-in functions. These will *not* go into your homework solution.

Try out each of these functions in the trinket below.

range

`range` returns a list of integers.

Input: `range(100)`
Output: `[0,1,2,...,99]`

Note that when you use `range`, as with almost everything in Python, the **right endpoint is omitted!**

sum

`sum` sums a list of numbers.

Input: `sum(range(0,101))`
Output: `5050`

Input: `sum([40,2])`
Output: `42`
This is a roundabout way of adding `40+2`!

Importing other code (or "modules")

To access functions that are not built-in by default, you need to load them from their modules. Try out these examples to get familiar with how to access Python's many libraries:

You can import a module, i.e., a library, and then access its functions using that module name:
`import`
Input: `math`
Output: no response from Python

Input: `math.sqrt(9)`
Output: `3.0`

Note that `sqrt` returns a `float` even if its input is an `int`.

Input: `math.cos(3.14159)`
Output: `-0.999...`
Note that `cos` and the other trigonometric functions take radians as input. Also, 3.14159 is less than `math.pi` (in other words, using `math.pi` is more exact than typing the first few digits of π).

Tired of typing `math.` in front of things? You can avoid this with `from math import *`.
The asterisk `*` means "everything" here. This will bring all of the functions and constants from the math module into your current python environment, and you can use them without prefacing them by `math.`

Input: `cos(pi)`
Output: `-1.0`
This would have had to be `math.cos(math.pi)` before the new `from math import *` import statement.

Creating Your Own Functions

For this assignment, you'll create a few functions in a new trinket.

Note: CS005x's lectures will cover functions in detail in the next lecture. Here, in anticipation, you'll create a few functions just to get started.

Functions are the fundamental building blocks of computation. What distinguishes Python from other computing environments is the ease and power of creating your own functions.

To get started, make a copy of the trinket below.

This trinket has some comments and a definition of a function named `dbl`. `dbl` outputs twice what it gets as input.

You can run this function by adding a call like `print dbl(21)` at the bottom of the trinket. You should get `42` as the output.

Docstrings

The string inside triple quotes `"""` is called the **docstring**, short for "documentation string." We will ask you to include a docstring in all of your functions (even simple ones such as these, in order to feed the habit).

A docstring should describe what the function outputs and what the function inputs. As you see above, it may include other important information, too.

Warning: The first set of triple quotes of a docstring needs to be indented underneath the function definition `def` line, at the same level of indentation as the rest of block of code that defines the function.

Functions to Write

Let's go!

For each of these functions, be sure to include a docstring that describes **what your function does** and **what its inputs are** for each function. See the `tpl` example, below, for a reasonable starting point and guide.

Math functions

0. **Example problem**: Write the function `tpl(x)`, which takes in a numeric input and outputs three times that input.

   Answer to example problem: Copy the following solution (after a few blank lines to leave space and help readability) into your Homework 2, Problem 2 trinket:

```
def tpl(x):
    """ output: tpl returns thrice its
input
          input x: a number (int or float)
    """
    return 3*x
```

1. Write `sq(x)`, which takes in a number named `x` as input. Then, `sq` should output the square of its input.

2. `interp(low,hi,fraction)` takes in three numbers, `low`, `hi`, and `fraction`, and should return the floating-point value that is `fraction` of the way between `low` and `hi`.

   What!?

   That is to say, if `fraction` is zero, `low` will be returned. If `fraction` is one, `hi` will be returned, and values of `fraction` between `0` and `1` will lead to results between `low` and `hi`. (In fact, values of `fraction` can go below `0`, yielding outputs less than `low`, and they can go above `1`, producing outputs greater than `high`. Purists would call this extrapolation, rather than interpolation, however.)

   From the above description, it might be tempting to divide this function into several cases and use `if`, `elif`, and the like. Yet, this function can be written using ***no*** conditional (`if/elif/else`) constructions at all! Try it *without* using `if`!

   As noted, your function should also work if `fraction` is less than zero or greater than one. In this case, it will be linearly extrapolating, rather than interpolating. We'll stick with the name `interp` anyway. Here are examples that will help clarify how `interp` works:

```
>>> interp(1.0, 9.0, 0.25)      # a quarter (.25) of the way from 1.0 to
9.0
3.0

>>> interp(1.0, 3.0, 0.25)      # a quarter of the way from 1.0 to 3.0
1.5

>>> interp(2, 12, 0.22)         # 22% of the way from 2 to 12
4.2
```

   Hint: If you're unsure of where to begin on this problem, look at the first example above. In it:
   `low` is `1.0`
   `hi` is `9.0`
   `fraction` is `0.25`

   See if you can determine how to combine those three values to yield the correct output of `3.0`. (Consider starting with `low`) `(hi -`)

   Here are two more examples to try:

```
>>> interp(24, 42, 0)           # 0% of the way from 24 to 42
24.0

>>> interp(102, 117, -4.0)        # -400% of the way from 102 to 117
(whoa!)
42.0
```

String functions

The next several functions involve strings of characters. Write each one in your Homework 2, Problem 2 trinket. After you write each function, be sure to test it! Also, be sure to include a docstring for each function that tells (very briefly) what it does.

3. Write a function `checkends(s)`, which takes in a string `s` and returns `True` if the first character in `s` is the same as the last character in `s`. It returns `False` otherwise. The `checkends` function does not have to work on the empty string (the string `''`).

   There is a hint below, but read through the examples first.

   These examples will help explain `checkends`—read them over now and be sure to try them once you have a first draft of your function. Notice that the final, fourth example below is the *string of one space character*, which is different from the empty string, which contains no characters.

   ```
   >>> checkends('no match')
   False

   >>> checkends('hah! a
   match')
   True

   >>> checkends('q')
   True

   >>> checkends(' ')
   True
   ```

   Make sure to check that this last example (the string of a single space) works for your `checkends` function. The empty string does not need to work.

   Hint: For this function you could use an `if` and `else` construction. Here is a start:

   ```
   if s[0] == ______
   :
       return True
   else:
       return False
   ```

   You might find a solution that doesn't use the `if` and `else` at all—this is fine, too. Notice that the *last* character is missing above—you'll need to fill that in!

Warning! Your function **should not return strings**! Rather, it should return a *boolean value*, either `True` or `False`, without any quotes around them. These `True` and `False` are keywords recognized by Python as representing one bit of information.

You'll see these booleans turn a different color, pink in trinket, indicating that Python does recognize them as `bool` values. (If you'd accidentally made them strings, they'd be quoted, and they'd be red in trinket.) In sum, booleans and strings are different. For `True` and `False` you would almost always want the unquoted boolean values.

4. Write a function `flipside(s)`, which takes in a string `s` and returns a string whose first half is `s`'s second half and whose second half is `s`'s first half. If `len(s)` (the length of `s`) is odd, the first half of the input string should have one fewer character than the second half. (Accordingly, the second half of the output string will be one shorter than the first half in these cases.) There's also a hint after the examples below.

   Here you may want to use the built-in function `len(s)`, which returns the length of the input string, `s`.

   Examples:

   ```
   >>>
   flipside('homework')
   workhome

   >>> flipside('carpets')
   petscar
   ```

   Hint: This function is simpler if you create a variable equal to `len(s)/2` on the first line, e.g.,

   ```
   def flipside( s ):
       """ put your docstring
   here
       """
       x = len(s)/2
       return   _____________
   ```

   where the return statement has been left up to you... it will use the variable `x` *twice* , which is why it's nice to give it a name, rather than type and re-type it!

String and arithmetic processing functions

These last functions combine string and arithmetic processing.

5. Write `convertFromSeconds(s)`, which takes in a nonnegative **integer** number of seconds `s` and returns a list (we'll call it `L`) of four nonnegative integers that represents that number of seconds in more conventional units of time, such that:

   - the initial element represents a number of days
   - the next element represents a number of hours
   - the next element represents a number of minutes
   - the final element represents a number of seconds

You should be sure that

- 0 ≤ seconds < 60
- 0 ≤ minutes < 60
- 0 ≤ hours < 24

There are no limits on the number of days.

For instance,

```
>>> convertFromSeconds(610)
[0, 0, 10, 10]
>>>
convertFromSeconds(100000)
[1, 3, 46, 40]
```

How to do this? Feel free to copy-and-paste this starter code that uses four variables:

```
def convertFromSeconds( s ):
    days = s / (24*60*60)  # # of days
    s = s % (24*60*60)     # the leftover
    hours =
    minutes =
    seconds =
    return [days, hours, minutes,
seconds]
```

The idea here is that, when those four variables are all correctly set, you can return them all in a list, which is the final line:

```
    return [days, hours, minutes,
seconds]
```

For instance, the line that sets `days` could be

```
    days = s /
(24*60*60)
```

What would be other lines be?

It's possible to do this without changing the variable `s` at all. However, as the above starting code suggests, it's also possible to *alter* `s` as you go. Try this latter approach, just to get the hang of this powerful strategy.

6. The sixth (and final) function to write is `front3` from CodingBat, an excellent Python-practice website. The easiest way to work on this problem is to head over to CodingBat, which will give you feedback on your function from within the browser! After you've finished the function, copy it into your Homework 2, Problem 2 trinket.

Submit Homework 2, Problem 2

25.0/25.0 points (graded)

To submit your Homework 2, Problem 2 code, you'll need to copy it from the trinket above and paste it into the box below. After you've pasted your code below, click the "Check" button.

IMPORTANT: Make sure that there aren't spaces at the beginning of your code, and that you copied all of the characters. If there are extra spaces or you are missing spaces, our server won't be able to run your code and we won't be able to give you any of the points you deserve for your hard work.

```
1


2


3


4


5


6


7


8


9
def dbl(x):
10
    """  output: dbl returns twice its
input
```

```
11
        input x: a number (int or
float)
12
        Spam is great, and dbl("spam") is
better!
13
"""
14
    return
2*x
15
16
def tpl(x):
17
    """ output: tpl returns thrice its
input
18
        input x: a number (int or
float)
19
"""
```

```
20

    return
3*x

21

22

23

def sq(x):

24

    """ output: sq returns the square its
input

25

          input x: a number (int or
float)
```

Press ESC then TAB or click outside of the code editor to exit
correct

correct

Test results
CORRECT See full outputSee full output

You have used 1 of 3 attempts Some problems have options such as save, reset, hints, or show answer. These options follow the Submit button.