MCS 260 – Introduction to Computer Science – Fall 2020 – Emily Dumas

Week 5 Worksheet

Instructions. You are strongly encouraged to have a Python interpreter and programming editor open at all times while working on this, and to be in a group in Tue/Thu discussion where at least one person is sharing their screen.

Language features. In your solutions you can use (and may need) the following built-in functions/methods, even if they haven't been discussed in lecture:

- The syntax for taking slices of a sequence has an additional feature; instead of always moving to the next element, you can specify that steps of any fixed size should be taken. The syntax is

  ```
  seq[start:stop:stride]
  ```

  where `stride` indicates the number of items per step. For example,

  ```
  >>> seq = [ 1, 2, 3, 4, 5, 6, 7 ]
  >>> seq[1:6:2]  # steps of size two
  [2, 4, 6]
  ```

  or

  ```
  >>> s = "melodiousness"
  >>> s[::3]
  'moons'
  ```

  You can also use negative strides, which means to move backwards, in which case the default start position becomes the end of the string:

  ```
  >>> s = "deliver"
  >>> s[::-1]
  'reviled'
  ```

Problems.

(1) Consider the code below.

```
s = 1
def f(s,t,u=-5):
    """Work with local and global vars to test
    whether students understand the difference.
    """
    t = 2
    v = 3
    for i in range(5):
        s = s*t
    del t
    return(s+u+v)

t = 3
s = f(1,t)
print(s,t)
```

(a) The output is `30 3` as you can check. Explain why this is so. Specifically, explain what assignments take place, in what order, and whether the variable name assigned is local or global.

(b) There are four variable names in this code. Classify each one according to its role: Either "global" (for variable names that are exclusively global), "local" (for variable names that are exclusively local), or "both" (for variable names that sometimes refer to local variables, and sometimes global variables).

Name	Role
s	
t	
u	
v	

(2) Write a function `only_palindromes(L)` that accepts a list `L` of words (strings) and returns a list containing only the palindromes that appear in `L`. A palindrome is a word that is unchanged by reversing the order of its letters.

So, for example

```
L = [ "drab", "racecar", "tremble", "reviver", "sagas", "dressed" ]
print(onlypalindromes(L))
```

should print

```
['racecar','reviver','sagas']
```

Hint: The first step might be to write a function that reverses a string. Use the extended slice syntax with negative stride for this!

(3) Write a function `get_ints(s)` that extracts a list of integers from a string like:

```
" 57,23, 8, and 5    "
```

Specifically, the string will contain digits, commas, whitespace, and possibly some number of instances of the word "and". Your function should take the string as a parameter, ignore the commas, whitespace, and the word "and", treating all of these as separators. The numbers that appear between these separators should be converted to ints and returned in a list. Test your function on at least these cases:

```
get_ints("8,6,7,5,3") = [8,6,7,5,3]
get_ints(" 57,23, 8, and 5    ") == [57,23,8,5]
get_ints("1and2and    3") == [1,2,3]
get_ints("55,65  5") == [55,65,5]
```

(4) Write a function `roll(sides,times)` that simulates rolling a die with `sides` sides (numbered 1 to `sides`) a total of `times` times. (Use the random module to make the random choices you need for this.) It should return the sum of all the rolls (an int). Thus for example `roll(6,2)` simulates rolling two standard cube dice and taking the sum.

(5) Building on the previous exercise, write a function `wait_time(n)` that will repeatedly roll two standard cube dice until the total is equal to `n`. This function should return the total number of rolls that were necessary. The parameter `n` should have default value 7, so that calling `wait_time()` will return the number of rolls necessary to get 7.

(6) Write a script `argdata.py` that expects at least one command line argument, and which builds a dict that has the command line arguments as keys. The value for a key `a` should be a tuple, where the first element indicates the position of the argument (0 for the first argument after the script name, 1 for the next, ...) and the second element indicates the length of that argument. Finally, your script should print the dict it has created.

For example, if the script is called from powershell or a terminal as

```
C:\Users\ddumas\> python argdata.py melted into thin air
```

then the output should be

```
{'melted': (0, 6), 'into': (1, 4), 'thin': (2, 4), 'air': (3, 3)}
```

(or something equivalent with the keys listed in a different order).

Revision history:

- 2020-09-20 Initial publication