Problems of the Day

Note: all problems should be coded in **the same** (single) Corona/Solar2D project, unless otherwise noted.

1. Scope of variables - local vs. global
    a. It is strongly recommended that you try to answer the following questions on paper first (i.e., run this "in your head"), and only then type in the following into a Lua interpreter (e.g., http://www.lua.org/demo.html) and verify/validate your answers/understanding.
    b. Then write down the results:
        i.
```
local a = 5
print(a)
```
        ii.
```
local a = 5
a = 6
print(a)
```
        iii.
```
local a = 5
print(a)
do
  local a = 6
  print(a)
end
print(a)
```
        iv.
```
function bar()
  print(x)
  local x = 6
  print(x)
end

function foo()
  local x = 5
  print(x)
  bar()
  print(x)
end

foo()
```
        v.
```
function bar()
  print(x)
  local x = 6
  print(x)
end
```

```
function foo()
  x = 5
  print(x)
  bar()
  print(x)
end

foo()
```

2. At first, skip this: In a new Corona/Solar2D project: Abbreviations (with Corona UI, see User Interface (UI) below) - Create a program which has 3 (or more, if you'd like) buttons with an abbreviation displayed/printed on each one (e.g. "CU", "BRB", "LOL").
   Each abbreviation will be associated with a full text sentence (e.g. "See You Later", "Be Right Back", Laughing Out Loud", respectively).
   When the user clicks on a button it will print/display the full text sentence in an area of text below the buttons.

   Extra Experience

   Make the layout of the screen such that you leave room for more than 3 buttons, and implement the area of text (below the abbreviation buttons) as an area where you can type in additional text through the keyboard and associate this text with the additional buttons.

3. Exploring prime numbers ("Primeness" Project)
   a. Write a function isPrime(n) to determine if a number n is a prime. Make sure it is as optimized/**fast** as you can make it since you will be using it later in large number calculations/loops.
      i. Check if 1234511 is prime.
      ii. What about 355555553?

   b. Use your FAST isPrime() function above, to find "**twin primes**".
      Twin primes are pairs of primes adjacent (next) to each other. For example: (3,5), (5, 7), or (11,13).
      i. Print the first twins up to 100.
      ii. (For the following part, don't print the actual primes! It's going to take too much time/space): **How many twin primes** (pairs) are there between 3 and 10,000? (you should get 205)

   c. Use your FAST isPrime() function above, to find "**triplet primes**".
      Triplet primes are three consecutive primes such that the first and the last differ by six, and there is only one prime in between. E.g., (11, 13, 17), (103, 107, 109), (641, 643, 647).
      i. Print the first triplet primes up to 100.

ii. (Don't print the actual primes! It's going to take too much time/space): **How many triplets** (groups of 3 primes) are there between 3 and 10,000? (you should get 112)

d. Use your FAST isPrime() function above, to find "**pal primes**".
Pal(indromic) primes are primes which can be read the same way from left-to-right, or right-to-left, e.g., 93239, 133020331.
   i. How many pal primes are there between 3 and **100,000**? (you should get 112)

e. Similarly, find how many of the following primes there are between 3 and 10,000:
   i. **plateau** primes: prime numbers which start and end with the same digit, and have a "flat/plateau" middle, consisting of the same digit. **Note**: by this definition, some single digit and double digit primes are plateau primes!
   e.g., 3, 11, 1777771, 355555553. (you should get 19)
   ii. **circular** primes: remain prime when we "rotate" their digits (e.g., 11, 13, 17, 37, 79, 113, 197, 199, 337, 1193, 3779, 11939, 19937, 193939, 199933).
   iii. **Extra Extra Experience - absolute** primes: remain prime for all permutations (e.g., 199, 919, 991).

4. Robber's Language app - **without** a GUI (Graphical User Interface):
In a new Corona/Solar2D project: Write a function translate(s) that will translate a sentence, s, into "rövarspråket" (Swedish for "robber's language"). That is, double every consonant and place an occurrence of "o" in between.
   a. For example, translate("this is fun") should return the string "tothohisos isos fofunon"
   b. **Extra Experience**: add **reverse translation** from "rövarspråket" to plain text.
      i. For example reverse("tothohisos isos fofunon") will return "this is fun".

5. Robber's Language app - **with** a GUI (Graphical User Interface): Enhance your Corona/Solar2D project above:
   a. **Create a UI** with one text field for entering the "plain sentence", and one display field for showing the "rövarspråket sentence". **Clicking a button** should start the translation, or if you'd like, **hitting 'enter'** on the keyboard should start the translation.
   b. **Extra Experience**: add a reverse translation User Interface element (button) from "rövarspråket" to plain text. Clicking on this reversing button will take the text in the "rövarspråket sentence" text field and display the plain text in the "plain text" field.

6. Same colors - What's the chance of randomly picking the same colors from a Lua table (AKA list in Python)? Create a table of 16 non-repeating colors. For example: "red", "orange", "yellow", "green", "blue", "indigo", "violet", "black", "gray", "magenta", "LightBlue", "gold", "LightGreen", "SkyBlue", "aqua", "lime"

   Write a program which randomly picks a number between 1 and 16, and then selects that many colors from the predefined table (AKA list in Python) of colors (see above). The program then prints the table of the randomly selected colors, and counts how many times each color shows up in this random list.

   For example, in the randomly selected table of 6 colors 'green', 'green', 'LightGreen', 'green', 'gold', 'black', your program should produce the following:

   ```
   I selected 6 colors:
   green, green, LightGreen, green, gold, black
   They show up:
   green,      3 times
   LightGreen, 1 time
   gold,       1 time
   black,      1 time
   ```

   Hint: use tables as dictionaries and counters (see the Lua Data Structures doc)

7. Same birthdays (builds on similar "logic" to the "Same colors" exercise above): In a room of 23 people, what are the chances that at least two of them will have the same birthday (same month and day, not necessarily year)?

   Write a program to run an experiment with R rooms and P people in each room, to see how often it happens.

   For each room, each person in the room is randomly assigned a valid birthday, namely a month and day (you don't have to deal with leap years/months, so there will not be a Feb. 29 b-day). If a person has the same birthday as another person in the room (i.e., same month and same day of the month), it increments a counter for that room.

   You can hardcode the user input or: Ask the user how many times they want to run the experiment (i.e., the number of rooms; say, anywhere in the range 20-100) and how many people they want in each experiment (i.e., room) (say, 20-100). At the end print the number of rooms you found at least 1 pair of people with the same birthday.

   The result should be like the following:

```
How many rooms to run? 4
How many people in each room? 23

Room: 0
no duplicate b-days

Room: 1
guests 10 & 16   have b-day=5/23

Room: 2
guests 10 & 12   have b-day=3/6
guests 17 & 19   have b-day=8/24

Room: 3
no duplicate b-days

2 rooms out of 4 ( 50 %) rooms had at least
2 people with the same b-day
```

The program should calculate the percentage of times it found at least 2 identical birthdays.

8. **Fairness of random()** - write a program which asks the user how many times they want to roll a 6-sided die (singular of dice :) or hardcode the value (say, a number between 100 and 1000), and then "rolls the die" (i.e. creates random numbers) that many times, and counts how many times each side came up.
   Is the random() function "fair"?

9. **Pangram -** write a program which takes a sentence and checks whether it's a pangram (a sentence containing at least one instance of each letter of the alphabet).

For example: "**Barely a few quips galvanized the jury box in the court of my king**" is a pangram.

Hint: you can break up the problem and write and use 2 functions in your solution:

found(c, sentence) which returns True if the character/letter 'c' is found in 'sentence'', and false otherwise.

missingLetters(sentence) which will use found() above to check if each letter of the alphabet is in 'sentence' and either return an empty table if 'sentence' is a pangram, or a table of all the missing alphabet letters in 'sentence', if 'sentence' is not a pangram.

Test your program with:

- The quick brown fox jumps over the lazy dog
- The five boxing wizards jump quickly

Use your program to identify the missing letters and then manually fix the following (non-pangrams) and make them pangrams:

- Few quips galvanize the King's court jury
- The five boxing champs jumped quickly

Come up with a few new pangrams of your own.

10. Printing M x N Matrix in "Spiral Order"

    For Example, given the 4 x 4 Matrix:

    11 12 13 14

    15 16 17 18

    19 20 21 22

    23 24 25 26

    Your program should print in Spiral Order (going clockwise) like so:

    11 12 13 14 18 22 26 25 24 23 19 15 16 17 21 20

    Extra Experience: you can solve it by recursion. Print the "outer layer" of the matrix (in the example above, the numbers from 11 all around the matrix up to 15), then print the remaining sub-matrix (e.g. the layer starting at 16 and all around to 20) recursively. And if the smaller number between M and N is odd, then we have to handle the base case (m == 1 or n == 1).

11. Optional: Write a module (see tutorial at http://www.tutorialspoint.com/lua/lua_modules.htm) called **binarymath.lua**, which supports the following operations:
    a. dAdd(5, 6) -- decimal addition; returns 11
    b. dSub(101, 10) -- decimal subtraction; returns 91
    c. bAdd(101, 11) -- binary addition; returns 1000
    d. bSub(1001, 10) -- binary subtraction; returns 111

    Extra Experience: each function/operation should handle invalid arguments (e.g. non-binary numbers for binary operations)

12. Look-and-Say: Watch the clip Conway's Look-and-Say Sequence explained.
    As shown in the clip you have just watched (you have watched it, haven't you?): The look-and-say sequence is such a sequence that for creating each new generation of this sequence you have to say (out-loud) the numbers in the current generation, and then write what you have said numerically. You can take any generation as a starting point (the "seed"), and then follow this rule to produce the next generation.

    For example, if you start with generation 3 as your seed (see row 3 below), and want to produce generation 4, you read (for gen. 3): I have one 1, one 2, and two 1s, and this produces gen. 4: 11 12 21

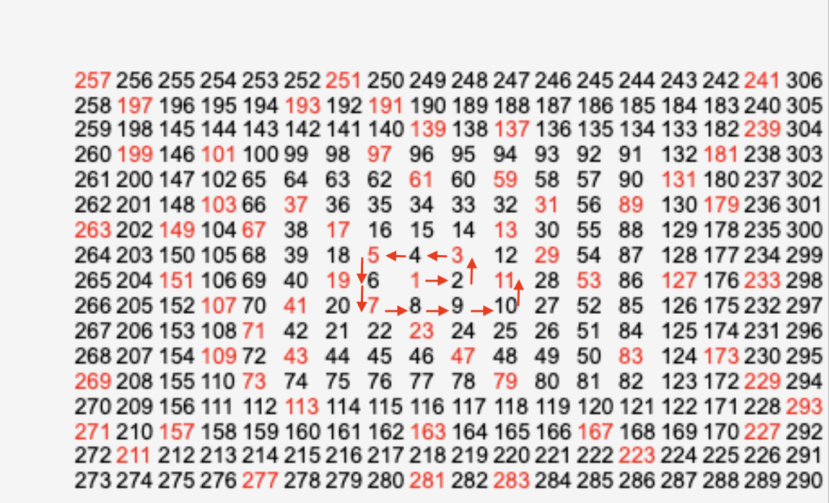
The output below shows the "Conway Look-and-Say Sequences" for 11 generations (not counting the "baseline" on line 0).

0	1
1	11
2	21
3	1211
4	111221
5	312211
6	13112221
7	1113213211
8	31131211131221
9	13211311123113112211
10	11131221133112132113212221
11	3113112221232112111312211312113211

Write a function lookAndSay(seed, gen), which prints gen generations starting with seed. You can print the output to the console (no need to code an app displaying this on a device screen (unless you like to :)).

13. Using your knowledge and code for Primeness (see your solution to a PotD above), create an app which displays Ulam's Spiral.
Watch the video at https://www.youtube.com/watch?v=3K-12i0jclM before you start.

Here is an example of the display of Ulam's Spiral starting from 1 in the middle (you should not draw it with arrows!):



If you start Ulam's Spiral from 41 in the middle: notice the straight line on the bottom-left to top-right diagonal):

```
185 184 183 182 181 180 179 178 177 176 175 174 173
186 141 140 139 138 137 136 135 134 133 132 131 172
187 142 105 104 103 102 101 100 99  98  97  130 171
188 143 106 77  76  75  74  73  72  71  96  129 170
189 144 107 78  57  56  55  54  53  70  95  128 169
190 145 108 79  58  45  44  43  52  69  94  127 168
191 146 109 80  59  46  41  42  51  68  93  126 167
192 147 110 81  60  47  48  49  50  67  92  125 166
193 148 111 82  61  62  63  64  65  66  91  124 165
194 149 112 83  84  85  86  87  88  89  90  123 164
195 150 113 114 115 116 117 118 119 120 121 122 163
196 151 152 153 154 155 156 157 158 159 160 161 162
197
```

Write an app that displays Ulam spiral on the device screen, using a function drawUlamSpiral(start, finish), where start is the beginning number (in the middle of the spiral), and finish is the largest/final number to display in the spiral.

14. Happy Numbers - Read the description and examples for happy and sad numbers, and write a program that finds all Happy and Sad numbers in a given range (e.g., 1-1000). Print your results to the console.

15. Next ...

Answers to Question 3 (Primes):

There are 205 twin primes between 3 and 10000

There are 112 triplets between 3 and 10000