About Lab 4

Lab 4 asks you to hand in 3 programs. One uses the drawing features of the picture module to make a picture.

The picture is fun and not hard. Unfortunately it is also mesmerizing. There are always more tweaks you can add to improve it. I suggest that you just make a quick start on it during the lab, and return to it after you have finished the last part of the lab, which is a game.

The picture is fun and not hard. Unfortunately it is also mesmerizing. There are always more tweaks you can add to improve it. I suggest that you just make a quick start on it during the lab, and return to it after you have finished the last part of the lab, which is a game.

The second part of the lab gives you a main( ) function and asks you to write 3 functions that it calls

- square(x) returns x*x
- checkEvenOrOdd(x) prints a statement about whether x is even or odd
- reverse(x) starts with integer x and returns the number with the digits of x in reverse order: reverse(235) is 532

Here is an algorithm for reverse(). Suppose the argument is n and you start variable total at 0.
At each step d = n%10, n is n//10, and
total =10*total+d
This continues until n is 0

For example, start with 235

Step 0: n is 235, total = 0

Step 1: d is 5, n is 23, total is 5

Step 2: d is 3, n is 2, total is 53

Step 3: d is 2, n is 0, total is 532

The primary thing that will take up your time in this week's lab is implementing the game Mastermind. In your game the computer will randomly select a "code" consisting of 4 letters from the string "RGBOYP" (which stand for "Red", "Green", "Blue", "Orange", "Yellow", and "Purple"). The user's job is to guess the code. The user gets 10 guesses.

The response the computer makes to a user's guess consists of two kinds of "pegs". A black peg indicates a choice of the correct color in the correct position. A white peg indicates a correct color in the wrong position.

For example, suppose the code and guess are

code: RRYG
guess: RYGB

There is one slot where the guess has the correct color and two slots in the guess where the colors are correct but in the wrong location. So the correct response is

1 black peg 2 white pegs

How should we respond to this?

code	RRYB
guess	BYRB

A. 1 black, 3 white
B. 1 black, 2 white
C. 1 black, 1 white
D. 2 black, 2 white

To this?

code	RRYB
guess	RRRR

A. 2 black, 3 white
B. 2 black, 2 white
C. 2 black, 1 white
D. 2 black, 0 white

To this?

code	RRYB
guess	RBBR

A. 1 black, 3 white
B. 1 black, 2 white
C. 1 black, 1 white
D. 2 black, 2 white

Here is pseudo code for the main( ) function:

```
def main( ):
        < print welcome to game>
        code = generateCode( )
        done = False
        while not done:
                guess = input( <prompt for a guess> )
                numGuesses = numGuesses + 1
                black, white = evaluateGuess(guess, code)
                < respond to guess>
                if black == 4:
                        done = True
```

This pseudocode has the game continuing until the user guesses the code. The lab actually asks you to also keep a count of the number of guesses and to stop the game when that gets up to a constant NUM_GUESSES, which you should set to 10.

This organizational structure has you writing two primary functions:

- generateCode( ) builds and then returns a random string of length 4 made from the letters RGBOYP
- evaluateGuess(guess, code) returns the number of black pegs and the number of white pegs for the guess

Function generateCode( ) should be easy. Let variable *colors* be the string

colors = "RGBYOP"

If you let i be a random index between 0 and 5, then

colors[i]

is the next letter to add to your code. Do this 4 times (gee, how can we make something happen 4 times???) and you have your code.

The evaluateGuess(guess, code) function is harder. You want to do this in 2 stages -- first count the number of black pegs, then count the number of white ones. Since black pegs correspond to colors in the right location, you need to be able to compare corresponding entries of the two strings. Have a loop on variable *i* and increment the number of black pegs when guess[i] == code[i].

To ensure that you don't use an entry for both a black peg and a white one, when you find a match replace the code entry by 'x' and the guess entry by 'y'. For example, with

code RRYB
guess RRRR

after the loop counting black pegs the strings should be

code xxYB
guess yyRR

The loop for counting white pegs has one loop going through every index i of the guess and every index j of the code. If you find that guess[i]==code[j], then increment the white counter and replace the letters by 'x' and 'y' to avoid reusing the pegs.

For the example,

code RRYB

guess BYRB

after the black peg loop we have

code RRYx

guess BYRy

and after the white peg loop it is

code xRxx

guess Byyy

The lab makes a suggestion for how to replace one letter of a string. If you want to make the ith letter of guess be 'x' you could say

```
guess = guess[0:i]+'x'+guess[i+1:]
```

Or you could make a function

```
def replace(s, i, newLetter):
    answer = s[0:i]+newLetter+s[i+1:]
    return answer
```

and call

```
guess = replace(guess, i, 'x')
```