

Assignment 2

Complementary materials

Dr. Weitao Xu

weitaoxu@cityu.edu.hk

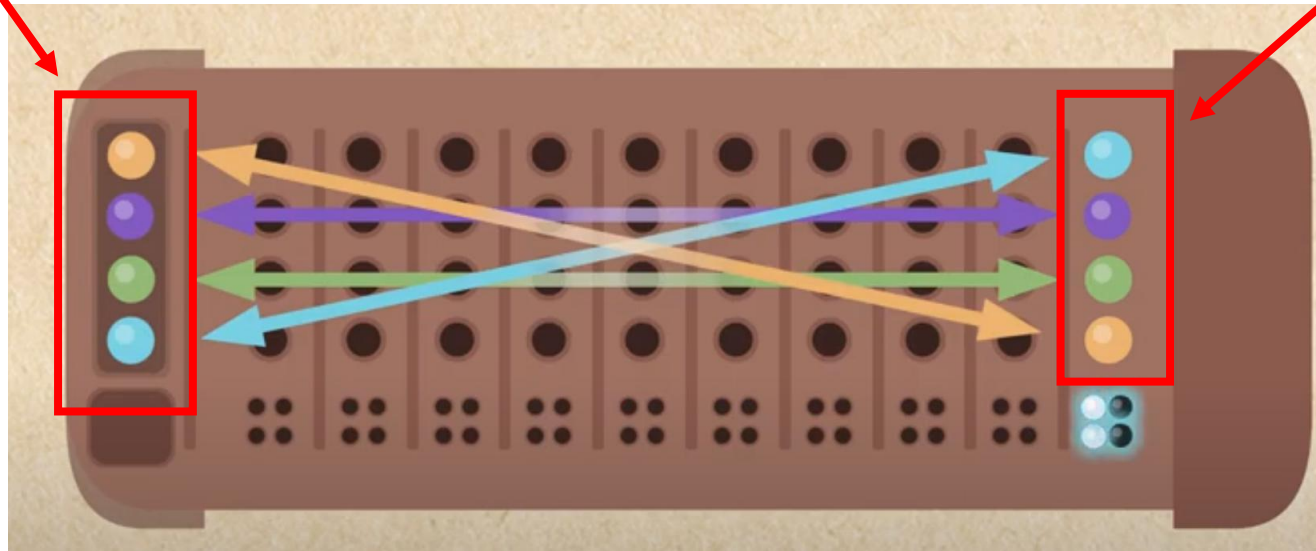
This assignment is challenging, pls try your best

code: it's a string, it represents the correct answer, in this example it's "YPGB" (yellow-purple-green-blue)

code_length: the length of this string,

guess: it's user's input, it's also a string.

In this example, it's "BPGY" (blue-purple-green-yellow)



In this example, the feedback is bbww, meaning 2 black_key+2 white_key

Purple and green are correct color in correct positions->2 black

Yellow and blue are correct color in incorrect positions->2 white

feedback: every time, the system generates a feedback for user's guess.

black_key_pegs_count is the number of code pegs that are the correct colors in the correct positions

white_key_pegs_count is the number of code pegs that are the correct colors but in incorrect positions

Random Code Generation

- import random
- random.random() → # random floating point numbers in [0,1)
- random.randint(3,10) → # random integer in range [3,10]
- random.choice('RBG') → # random element in the sequence 'RBG'

```
In [5]: for i in range(10):  
        print(random.random()) # random floating point numbers in [0,1)  
  
0.22658523115537088  
0.7588514427678638  
0.8849896000051662  
0.08168255563570737  
0.9865611576826422  
0.678993312640145  
0.12242438912135811  
0.7797939786512964  
0.35017164899600406  
0.716457101584178  
  
In [6]: for i in range(10):  
        print(random.randint(3, 10), end=" ") # random integer in range [3,10]  
  
6 6 3 10 5 4 6 6 5 8  
  
In [7]: for i in range(10):  
        print(random.choice("RBG"), end="") # random element in the sequence 'RBG'  
  
RRRGRGGBRR
```

Exercise 1 (2 marks)

Define a function that generates a random `code`. The function takes in


- a string `colors` whose characters represent distinct colors to choose from, and
- a positive integer `code_length` representing the length of the code.

For instance, `get_code('ROYGBP',4)` returns a code of `4` code pegs randomly with colors chosen from

- `'R'` ed,
- `'O'` range,
- `'Y'` ellow,
- `'G'` reen,
- `'B'` lue, and
- `'P'` urple.

One possible outcome is `'ROYY'`.

```
] : def get_code(colors, code_length):  
    code = ''  
    # YOUR CODE HERE  
    pass  
  
    return code
```



Solution 1: use `random.randint()`

- 1) use a for loop to generate some random numbers
- 2) use the random numbers as index to generate random colors
- 3) concatenate these colors together

Solution 2: use `random.choice()`

- 1) use a for loop to generate some random characters from `colors`
- 2) concatenate these colors together

Guess Validation: used to validate if user's input is valid or not. (A solution template is provided to assist you)

Exercise 2 (3 marks)

Define a function `valid_code` that

- takes `colors`, `code_length`, and `guess` as the first, second, and third arguments respectively, and
- returns `True` if `guess` is a valid code, i.e., a string of length `code_length` with characters from those of `colors`, and
- `False` otherwise.

```
] def valid_code(colors, code_length, guess):  
    # YOUR CODE HERE  
    pass
```



Hint

Solution template (This is just a suggestion, you may also use your own code):

```
def valid_code(colors, code_length, guess):  
    if len(guess) == code_length:  
        is_valid = True  
    else:  
        for peg in guess:  
            for color in colors:  
                if peg == color:  
                    is_valid = True  
                    break  
            else:  
                is_valid = False  
                break  
        else:  
            is_valid = True  
    return is_valid
```

If you don't understand while/else and for/else, pls go to Chapter 5.6 of our reference book.

This function is used to check whether user's input is correct or not. There're two types of invalid input, first the user inputs the wrong number of colors, for example, user should enter 3 colors, but he enters 2. This possibility is checked by the first if statement.

```
if len(guess) __ code_length:  
    is_valid = ____
```

If the length of user's input is equal to the correct code_length, another possible type of invalid input is he enters some colors we don't have, for example, we only have three possible colors, say "RGB", but users may input "RGY". So the code below checks whether user's input color is what we expect. To do so, we need to check each letter in user's input "guess", compare it with each letter in our defined "colors", to see if it belong to our defined colors or not. If one letter doesn't match, we can terminate immediately because we know it's invalid. Let's use the above example again, we define colors be "RGB", and user enters "RGY",

- 1) we check the first letter in guess "R", to see if any letters in colors is equal to it or not, we find that there's a "R" in colors,
- 2) so we move to the next letter "G", we compare each letter in "RGB" to see if G exist or not, and we find there's a "G" in "RGB",
- 3) so we move to the last letter "Y", we iterate all the letters in "RGB", and we find that there's no letter equal to "Y", so we know it's invalid. That's the logic to solve this problem, you don't need to strictly follow the hints below, cause it's just one way to do so. If you understand this logic, you can write your own code, no need to follow this template strictly.

```
for peg in guess:  
    for color in colors:  
        if peg == color: ____  
    else:  
        is_valid = ____  
        ____  
else:  
    is_valid = ____  
return is_valid
```

Hints:

- 1) as long as we find user's input is invalid, we can terminate immediately, how?
- 2) the value of is_valid can only be True or False

Feedback Generation

"Hint: Use help to learn how to use the imported functions.", that means you can type

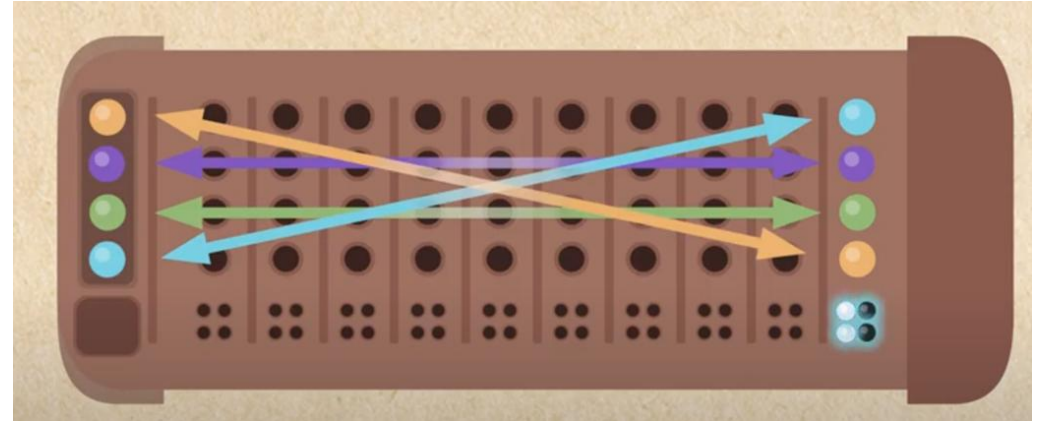
- `help(mark_as_counted)`
- `help(check_if_counted)`
- `help(reset_all_to_not_counted)`

to get information of these three functions.

- As I said in the first question, for each of user's input, we need to check whether it's correct or not.
- Suppose the correct color is "RGB", and user enters "BGR", we need to check whether the first letter is correct or not, the second letter "G" is correct or not, and the third letter "R" is correct or not.
- After we check the first letter, we mark the first letter is counted, we use `mark_as_counted(1)` to represent this operation.
- If we want to check if the second letter has been counted or not, we use `check_if_counted(2)`, if it's true, it means we already checked it's correctness. Otherwise, if it's false, meaning we haven't checked it's correctness.
- `reset_all_to_not_counted()` means we reset the boolean value of each letter to false, meaning we reset to the default condition: all the letters have not been verified.

Why we need these three functions?

-To avoid double count



black_key_pegs_count is the number of code pegs that are the **correct colors** in the **correct** positions

white_key_pegs_count is the number of code pegs that are the **correct colors** but in **incorrect** positions

Give a user's input color, it has three possibilities:

- 1) If it is wrong color → we do nothing
- 2) If it is **correct color** and in the correct position → we add black_key_pegs_count by 1
- 3) If it is **correct color** but in the wrong position → we add white_key_pegs_count by 1

It is possible that you count a correct color in both white_key_pegs_count and black_key_pegs_count

So every time after you determine whether a color is case 2) black_key or case or 3) white_key, we need to mark this position as counted. Otherwise, we may count it again → to mark the i-th color, we use mark_as_counted(i)

To check whether i-th color is counted or not, we use check_if_counted(i), and its return value is True or False

True means it's counted and False means it's not counted.

This is the most difficult question

Exercise 3 (5 marks)

Define a function `get_feedback` that

- takes `code` and `guess` as the first and second arguments respectively, and
- returns a feedback string that starts with the appropriate number of characters `'b'` (for black key pegs) followed by the appropriate number of characters `'w'` (for white key pegs).


```
] def get_feedback(code, guess):  
    #your code here  
    pass
```



Hint

Solution template (This is just a suggestion, you may also use your own code):

```
def get_feedback(code, guess):  
    black_key_pegs_count = white_key_pegs_count = 0  
    reset_all_to_not_counted()  
    for i in ____:  
        if ____:  
            black_key_pegs_count += 1  
            mark_as_counted(i)  
    for i in range(len(guess)):  
        for j in range(len(code)):  
            if ____:  
                white_key_pegs_count += 1  
                mark_as_counted(j)  
                break  
    key = 'b' * black_key_pegs_count + 'w' * white_key_pegs_count  
    return key
```

This function will generate the number of `black_key_pegs` and the number of `white_key_pegs`. 

The idea is : iterate each letter in “guess”, and compare It’s color and location with colors in “code” to determine The number of `black_key_pegs` and `white_key_pegs`.

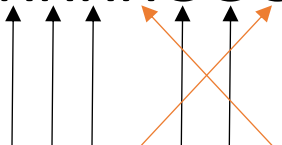
`Black_key_pegs_count`: the total number of `black_key_pegs`
`White_key_pegs_count`: the total number of `white_key_pegs`

Test example for get_feedback()

- `get_feedback("RRRRGGG","RRRGGGR")` will return $5*'b'+ 'w'*2$, why?

bbbwbbw $\rightarrow 5*'b'+ 'w'*2$

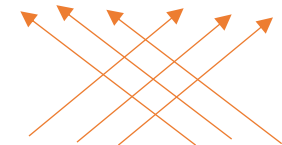
Code: RRRRGGG
Guess: RRRGGGR



- `get_feedback("RRRRGGG","GGGGRRR")` will return $0*'b'+ 'w'*6$, why?

www www $\rightarrow 0*'b'+ 'w'*6$

Code: RRRRGGG
Guess: GGGRRR



Note: the length of code is 7 but the length of the feedback is only 6!
Do not double count!