Ethan Ward
graded by Christine

~~/100~~ ~~65~~

67/100

```
import random

LENGTH_OF_CODE = 4

ALLOWED_CHARACTERS = ['1', '2', '3', '4']
```

(+10) ~~+10~~

```
def generate_code():
    """
    Returns secret code, which is a randomly generated four-element list of
    numbers between 1-4.

    >>> generate_code()
    ['1', '4', '3', '2']
    >>> generate_code()
    ['4', '4', '1', '3']
    >>> generate_code()
    ['2', '3', '4', '3']
    """
    nums = []
  for i in range(LENGTH_OF_CODE):
      p = int(random.choice(ALLOWED_CHARACTERS)[:LENGTH_OF_CODE])
      nums.append(str(p))
  print(str(nums))
```

code works, difficult to read.

The allowed characters are already in string format.

maybe try list comprehension

~~[for char~~

[random.choice(ALLOWED_ CHARACTERS) for _ in range(0, LENGTH

(+8)

```
def wrong_code_length(code_breaker_attempt):
    """
    Returns True if the length of the code_breaker_attempt is not the allowed
length set
    LENGTH_OF_CODE.

    >>> wrong_code_length(['1', '2', '4', '3'])
    False
    >>> wrong_code_length(['1', '2', '4', '5', '4'])
    True
    >>> wrong_code_length(['1', '2'])
    True
    """

    if len(code_breaker_attempt) == LENGTH_OF_CODE:
      return True
  else:
      return False
```

syntax error

this function was supposed to return the opposite → return True if its wrong code length. Right logic though!

(+5)

```
def wrong_characters(code_breaker_attempt):
    """
    Returns True if code_breaker_attempt contains characters that are not
allowed,
    which is set by ALLOWED_CHARACTERS.

    >>> wrong_code_length(['1', '2', '4', '3'])
    False
    >>> wrong_code_length(['b', 'e', 'e', 'p'])
    True
    >>> wrong_code_length(['m', 'a', 'r', 's'])
    True
    """

    if code_breaker_attempt == ALLOWED_CHARACTERS:
      return False
```

This is a comparison of two lists to see if they are the same lists. What you want is to check if each character in codebreaker_attempt is in allowed characters. Do this by iteration.

```
    else:
        return True
```

(+6)

```
def get_code_breaker_attempt():
      """
      The code breaker attempts to guess the secret code. This functions returns
that
      attempt as a four-element list. This function also checks to make sure if the
      attempt is valid (the right length and using valid letters). If not, then it
      continues to ask for a valid answer until one is given.
      """
      while True:
    if wrong_code_length(generate_code()) is  True:
        print(generate_code())
    if wrong_characters(generate_code()) is True:
        print(generate_code())
    else:
        print(False)
```

~~needs~~ missing line that gets code from player

this generate_code() is the secret code. It's assumed this is always the right length and characters.

should ask for code_breaker_attempt that is valid

this code should be indented under the while loop

print is different from a return statement

also the function overall should return a list, not a boolean

(+6)

```
def check_numbers(code, code_breaker_attempt):
      """
      Checks if colors guessed by the code breaker exist in the secret code.
Returns
      the number of correct colors.

      >>> check_numbers(['1', '1', '2', '4'], ['1', '1', '2', '4'])
      4
      >>> check_numbers(['1', '1', '2', '4'], ['2', '1', '3', '4'])
      3
      >>> check_numbers(['2', '1', '2', '1'], ['3', '1', '4', '1'])
      2
      """

      result = False
    for i in code:
        for j in code_breaker_attempt:
            if i == j:
                result = True
                return result
        print(result.count(True))
```

The logic of this function works more for check order, because you are comparing whether elements are in the right spot.

nice use of nested for loop!

function returns number of correct ~~colors~~ not boolean.

oh snap! I just realized I said colors not numbers.

not sure what this line is for

(+7)

```
def check_order(code, code_breaker_attempt):
      """
      Checks if numbers are in the same position as the corresponding number in
      the code. Returns the number of colors in the correct position.

      >>> check_order(['1', '1', '2', '4'], ['1', '1', '2', '4']))
      4
      >>> check_order(['1', '1', '2', '4'], ['2', '1', '3', '4'])
      2
      >>> check_order(['2', '1', '2', '1'], ['3', '1', '4', '3'])
      1
      """
```

" while list has same contents as other list, return the length of the list"

was this for diagnosing if the list was changing?

```
      while code == code_breaker_attempt:
          return len(code)
```

nice use of constant value! YASSS

oh wow! This is a really cool way to see if they are the same.

```
    for i in range(LENGTH_OF_CODE):
        match = abs(code[-1]-code_breaker_attempt[-1])
        if match > 0:
            return False
        else:
           return True
        return len(True)
```

you are doing math with two strings here

not supported by Python.

you can add strings or multiply, but nothing else.

? return length of a boolean?

(+10)

```
def get_key_pegs(numbers_check, order_check):
      """
      Key pegs returns feedback to code breaker about how much of their code is
correct.

      Red: Number is in the right position
      White: Number is in the wrong position
      Empty: Number (or duplicate of number) does not exist in the secret code.
      """

      if check_order(code, code_breaker_attempt) > 0:
        return 'Red'
    elif:
        return 'White'

    if check_numbers(code, code_breaker_attempt) == 0:
        return 'Empty'
```

indentation

good

gives feedback w/ print statements, but the feedback is not the feedback supposed to be given

(+~~0~~)
3

```
def give_player_feedback(key_pegs):

      if get_key_pegs(numbers_check, order_check) == 'Red':
    print('Red')
    if get_key_pegs(numbers_check, order_check) == 'White':
    print('White')
    if get_key_pegs(numbers_check, order_check) == 'Empty':
    print('Empty')


MAX_ATTEMPTS = 10

def continue_game_condition(key_pegs, attempts):

      "*** (OPTIONAL) YOUR CODE HERE ***"
```

(+10) ← I could not get the code to run :(

```
def mastermind():

      games = 1
    while games:
        games+=1
    if generate_code() < 10:
        return False
    else:
        return True
    if 'Red' = 4:
        return 'win'
    elif:
        return code

mastermind()
```