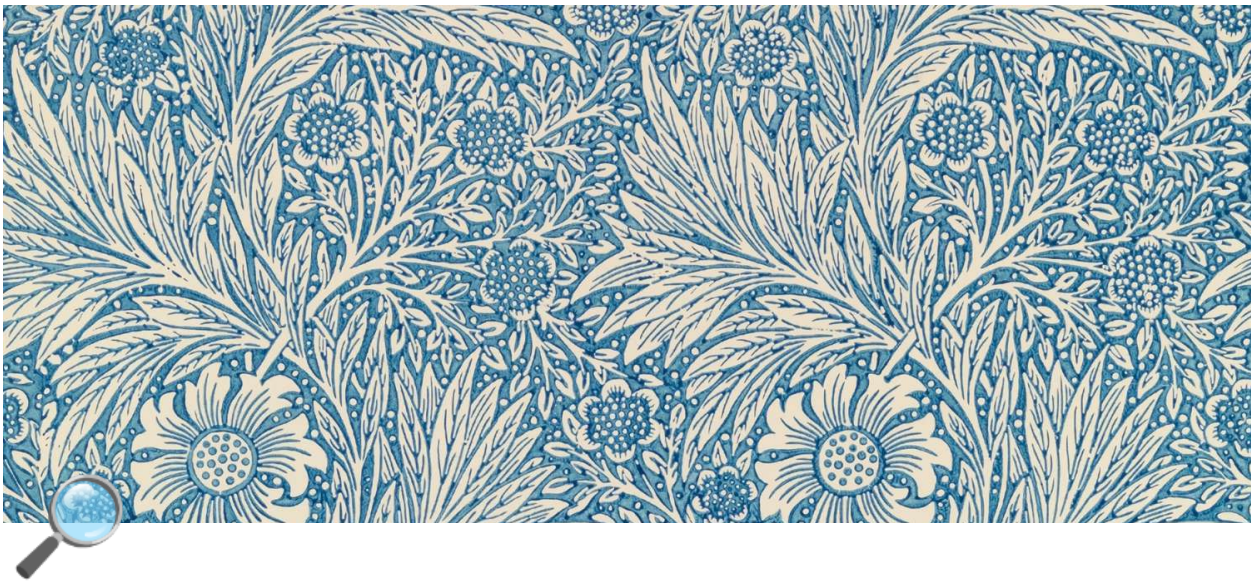

Algorithm mosaic patterns out of text (PA 3)

Made by Faycal Kilali for Programming Assignment 3, November 6, 2022 for Comp-1631.

Description / Utilization instructions

For Python: open the file in a python IDE, then run the code. Alternatively, invoke python onto the .py file and watch it work! You can invoke it in windows (if your pathing is correct for your Python installation environment) by ‘python A3-Kilali-Faycal.py’ without the ‘’ marks. For the additional tests, you’ll need to edit the scripts as appropriate to the test suite provided.

NOTE TO READER: The script comes with an input parameter, when you run the program it asks you for input. If you uncomment any of the test-cases they’ll override

the input anyway even if you put something inside it. I left the input on for you, as I figured you'd probably like it.

Intended output in general:

For each vowel in a word, the mosaic tile's red hex code will increment by 10, and for each other character in the word that isn't a vowel it'll increment the green hex code by 10, additionally the blue hex code will increment by 5 for each letter in the word.

For each word, a square will be drawn with the RGB coloring defined above, we'll draw to the right and then we'll move up based on the rectangular shape of the number of words (I considered the number of words in the text in the form of a rectangle, where height * width would give us the number of words in total in the text). Once all the squares in the width are drawn, we move up from the starting position an incremental height of side_length (constitutes a rise of 1 from the width to the height mathematically). Then we begin drawing eastwards again and so on repeating the same process until we reach our height.

Test Suite

Test 1:
This test was chosen in order to demonstrate the functionality of the mosaic pattern program. It uses the pre-requested test case.

text = 'O how I wish that ship the Argo had never sailed off’

Test 2:
Second mandatory test required by the assignment

text = 'Until the day when God will deign to reveal the futureto man, all human wisdom is contained in these two words Hope and Wait’

Test 3

text = "Onwards, choose a path for thyself, for that is the true freedom we have. Do not be average, seek your ambitions, seek to make the most out of every second of your life.”

Test 4:

A very interesting, and true case, in mathematics. Described in mosaic tiles. Absolutely stunning, especially if you actually bother reading it (you should!)

text = "Did you know, that there are more non-prime numbers than there are prime numbers? Think about it, every even number is divisible by 2, this suggests that at least half the set of integers is divisible by 2, but stop for a second! What about the integer 0? Well, its divisible by 2, but its also divisible by every other prime number, but by definition if a number is divisible by 2, then it is necessarily an even number. Now, this quantifies that for any consecutive integers subset of the integers set with the condition there is an even number of consecutive integers in the subset, suppose the size of the subset is K, then there would necessarily be >= K/2 even integers. Now, suppose our subset includes all the integers, then observe that, necessarily, half the integers will be divisible by 2, so at least half of the numbers will be composite numbers (non-prime), but then realize that a lot of those odd integers in the subset will be divisible by another odd integer, unless they are prime, so this tells us that in the subset of integers, there will be K/2 + c non-primes, where c is the number of integers not divisible by 2, but are composite as they are made of more than one prime factor, hence k/2 + c > 1/2n, so there are n - k/2 - c prime numbers in our subset of integer numbers.”

side_length = 10

Test 5:

Creative test, intended to describe the rectangular limitation of our implementation and how we row, and columnize the mosaic tiles.

text = "There's nothing to fear but fear iself.”

Self reflection (ideas on improving the project or one’s approach):

- I could adjust the size of the squares automatically corresponding to the number of words in a text, that way we can make sure no word will ever go off screen. However, this isn’t a requirement and I have other things to take care of currently.
- There may be a way to make multiple turtles draw instead of a single one, speeding up the drawing process. (Would appreciate input on this)
- The function limitation for the colours is a bit unfortunate, for I think we can do all 3 colors in one go (optimization wise) rather than having to loop through each word, separately, in each of the 3 coloring functions (this is bad optimization (because we could’ve only looped once through the word and assigned the hex codes! But you’ve essentially limited me to do it this way)
- Regarding the coloring, I figured this would create some interesting, unique, combinations for each word in terms of coloring. We can see the differences in the way we pronounce words, and the way they appear through their coloring in the mosaic tiles. Although, for the blue color I’m thinking of other ways to make it even more unique.
- I’ve chosen the square shape because it is the most consistent with tiling, and is relatively easy to work with. I have some glimpse of ideas on implementing additional shapes (perhaps based on the number of words on a phrase), that may make the tiles even more interesting. However, for now, the squares will do given the time constraints.

Comments:

Pretty neat project. However, I think we should not spend two programming assignments on Turtle. We should be doing something far more mathematical, something far more practical. We are still in the beginning stages here, and introducing Turtle, with its own set of documentation, syntax, and so on will not be as fruitful towards improving ones' understanding as say, pure mathematical algorithms in the default python environments (without any modules). We need to build off from there before we expand into other modules and their documentation in my opinion.

Algorithm:

(This algorithm uses indentation to better convey the order of execution of the process).

Acquire a set of words (phrase) from the user in the form of a file

    If there's no file, then ask the user for a set of words (phrase).

Split the phrase into a list of words

define starting x coordinate as starting_x_coordinate

define x, y coordinates to a feasible point for drawing the mosaic pattern we want

Define side_length, as the length of the side of the square, we can use any positive integer here.

Set our starting coordinate to (x, y) —, and we use that as our starting point.
Set our starting direction to the positive x axis (degree 0 from the x axis).

Define number of words in our list of words as n

As n is a rectangular number, then find the rectangular width by looping through the numbers up to n, and having a nested loop inside by looping up to n too, we check if the product of the two, at any point, is equal to n, then we define rectangular_width = width, for the rectangular height, define rectangular_length = height.

Set num_columns to 1

Loop through our list of words

    For the current word, we will loop through it so that each vowel (*by definition of being the letter a,e,i,o,u and or y as the last letter if a,e,i,o,u do not exist in the word — other forms of y as a vowel are not part of this algorithm*) increments the value of the

red color by 10, each other character that isn’t a vowel (except blank spaces — those are taken cacre of by the processing section) increment the green color by 10, and the number of letters in the word increment the blue color by 5 each.

Draw a filled square with the color (R,G,B) above in hex form, of side length = side_length, for each side, then move in the positive x direction by side_length.

If num_columns > rectangular_width,

    Set num_columns to 1

    reset our drawing position’s current x coordinate to starting_x_coordinate
    increment our drawing position’s y coordinate by side_length

Increment num_columns by 1.