RAPPORT PROJET BPI

JUSTESSE DU CODE :

- The draw.py executable takes as argument the size of the image, the number of points and the number of digits after the comma and returns 10 images in ppm format as well as the corresponding gif.
- If the user enters an incorrect parameter, the program returns an error as well as the call stack, for example $. /draw.py -800 100000 4
- The names of the images are according to the instructions, the name of the gif is chosen without importance (legif.gif).
- The subprocess module was imported to use the convert program:

$ subprocess.call("convert -delay 50 " + les_images + " legif.gif", shell=True)

RESPECT DES CONSIGNES :

The exceptions are well launched for the various arguments imposed on draw.py. The use of f-string, specifically type f'bloblo{blabla:. {toto}f}tutu' has been taken into account, to create the name of each image.

The images have a name corresponding to the format img{i}_[0-9]-[0-9][0-9][0-9]. ppm, and are well within the expectation (verified with the ImageMagick tool).

Run times are good, passing all performance tests.

The pi approximation is displayed in accordance with the number of digits after the comma chosen by the user and without an external module. A 7-segment display was implemented, centered in the center of the image and adapted to the size of the image. Colors are editable but limited (the file is in shades of grey).

QUALITÉ DU CODE :

After analysis of pylint, both programs obtain scores above 9/10. The use of global variables could have been considered to improve the readability of the code and to reduce the number of parameters of certain functions.

ETUDE DE LA PERFORMANCE :

OPTIMISATION :

- Since the draw program will call the approximation_pi modules several times, it is necessary to optimize them as much as possible. So for point generation, the random.uniform function was first used, generating a floating in [-1,1].

  However, we can notice that this function itself uses random.random. So it was more interesting to directly the latter and make some adjustments to have the generation of a point in the right interval.

- At each loop iteration, the liste_image and the dictionnaire_point are not reset. Thus, the points already drawn are kept on the new image, the approximation_list function having to generate only one tenth of the points in each step.
- The use of a dictionary was used for the write function. This dictionary has for key i and cross reference the function ciph_i. The use of this dictionary replaces the 10 tests initially planned and costly in complexity.

- The use of append has been limited to the maximum. We preferred to create tables of known size and replace the values as we went along. This optimization allowed a 20% gain in execution time.

COMPLEXITÉ EN MÉMOIRE :

The use of yield could have been considered to greatly reduce memory complexity. However, this option required almost complete rewriting of the program architecture, and was therefore not implemented.

POIDS DES IMAGES :

The ppm file was finally written in P2 gray instead of the original RGB choice. Indeed, the chosen color is then represented by a single character instead of 11, a reduction of the total number of characters per file of about 90%.

PERFORMANCE DES FONCTIONS :

Approximate_pi.py :

- The time complexity as a function of the number of points drawn is linear since the only for loop has this parameter size.
- The complexity in memory is constant, because there is no data storage other than the list created, the iterator and only one point created and then saved at a time. The memory cost is simply the allocated RAM for proper program delivery

Draw.py :

- The draw function has a linear complexity in number of points and number of digits after the comma. For the size parameter, since we are browsing the pixels of the image for writing the file, a total number size*pixel size, the complexity is quadratic. Indeed, several other loops are present in the program but none is nested in the other.
- The memory complexity of the function is constant with the number of points in the image but increases significantly with the size.

CONCLUSION :

The program delivered is in line with expectations and allows to effectively approximate the number pi, with a visual input in the form of gif. Although the program passes the required tests, both temporal and memory optimizations could have been considered, especially for large image generation.