QRGB+: Advanced QR Code Generator with RGB Color Method in Python to Expand Data Capacity

Ibar Federico Anderson*

Secretariat of Science and Technology, Department of Industrial Design, National University of La Plata, Argentina

Corresponding Author
Ibar Federico Anderson, Secretariat of Science and Technology, Department of Industrial Design, National University of La Plata, Argentina.

Citation: Anderson, I. F. (2024). QRGB+: Advanced QR Code Generator with RGB Color Method in Python to Expand Data Capacity. *J Sen Net Data Comm, 4*(2), 01-20.

Abstract
The present work entitled QRGB, consists of the development of an application in Python for the generation of QR codes using the additive color generation (RGB) method. This innovative method allows increasing the density of information stored in QR codes by using three color layers (red, green and blue), each representing a different set of data. QRGB offers an efficient and secure solution for storing and transmitting large amounts of information in limited spaces, significantly improving the capabilities of traditional black and white QR codes.

By using three layers of colors, QRGB codes can store up to three times more information in the same space. This technique not only increases storage capacity but also improves information security, making it difficult to forge or manipulate the code. The overlay of multiple data layers allows redundancy to be implemented, increasing the robustness of the code against damage or reading errors.

QRGBs are especially useful in applications that require the transmission of large amounts of data in limited spaces, such as in the packaging industry, digital business cards, and interactive advertising. Additionally, they have great potential in areas such as document and banknote security, where the authenticity and integrity of information are crucial.

These points provide a solid foundation for understanding the innovation and advantages of colored QR codes (QRGB) compared to traditional QR codes, highlighting their applicability and potential in various sectors.

This article presents a novel method for encoding and decoding information using a QRGB code, which involves the generation of three independent QR codes and their superimposition according to the additive color system (RGB). The research highlights the challenges encountered during the encoding and decoding processes due to the lack of specific libraries in Python, which required the creation of a custom solution using open source tools.

The implementation takes advantage of Python and its libraries: qrcode[pil] to generate QR codes with the Pillow dependency for image manipulation, Pillow to open, manipulate and save different image formats, and opencv-python to perform tasks such as image processing and object detection. Despite facing issues with color mixing and accurate information retrieval, the proposed method demonstrates a significant increase in data density within a single QR code. Future work will focus on optimizing the algorithm and exploring potential applications in data security and high-density information storage.

This Python script is designed to generate and decode QR codes with a logo overlay using a graphical user interface (GUI) built with Tkinter. The script combines several functionalities, such as creating QR codes, overlaying a logo, combining QR images of different colors, and manually decoding combined QR codes.

1. Introduction: What is a Traditional QR Code

A QR (Quick Response) code is a type of two-dimensional barcode that can store information efficiently and quickly. It was created in 1994 by the Japanese company Denso Wave to be used in the automotive industry, although its use has spread to many other areas due to its versatility and storage capacity.

The structure of QR codes is made up of a matrix of black and white modules (dots) that represent the encoded data. The modules are organized in a square and can contain a large amount of information compared to traditional one-dimensional barcodes.

In terms of their storage capacity, QR codes can store various types of data, including numbers, letters, special characters, and even binary data. The amount of information they can contain varies depending on the size and version of the QR code, but they can store up to 7,089 numerical characters or 4,296 alphanumeric characters.

Regarding ease of scanning, one of the most significant advantages of QR codes is that they can be quickly scanned from multiple angles, even if partially damaged, thanks to their error detection and correction patterns. Most smartphones and mobile devices with cameras can scan QR codes using specific apps or the device's camera.

The most common uses of QR codes in marketing and advertising: They are used to provide quick access to websites, promotions and discounts. In mobile payments, QR codes facilitate transactions by scanning a code containing payment information. In inventory management, they help track products and manage inventories in various industries. In information and education, they provide access to additional information about products, services or educational materials.

2. Components of the QR Code

• Pattern Finder: Three large squares in the corners that allow the scanner to identify and orient the QR code.
• Alignment Patterns: Small squares that help align the code if it is tilted or distorted.
• Timing Patterns: Zigzag lines that help determine the width of the modules.
• Data Area: The area containing the encoded data.
• Error Correction: Sections containing additional information to recover data if the code is corrupted.

In summary, QR codes are a powerful and versatile tool for the rapid encoding and transfer of information, and their use has spread to multiple sectors due to their ability to store a large amount of data and their ease of scanning.
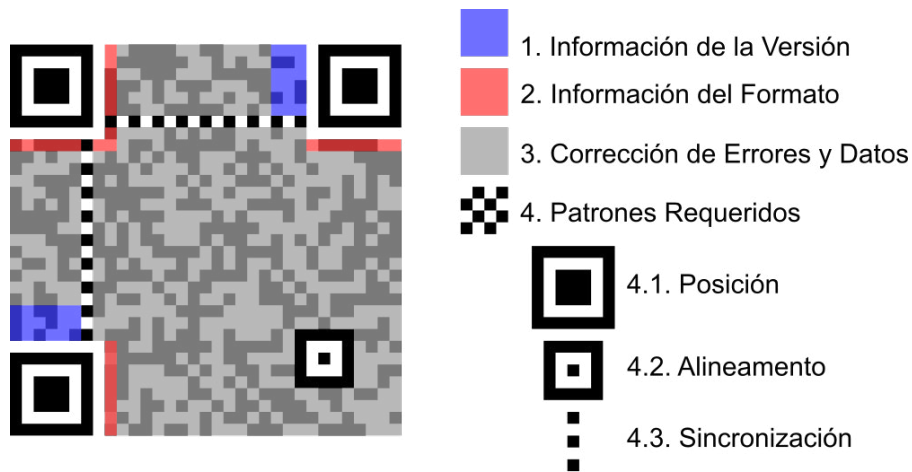
Figure 1: QR Code as it is conventionally known

Fountain: https://es.wikipedia.org/wiki/C%C3%B3digo_QR

3. QR Codes for Personalized Marketing

Using QR codes with images is a creative practice that has evolved over time. Although there is no single pioneer, several companies and developers have contributed to popularizing this technique. Some QR code generators with images include:

• Me-QR: This generator allows you to convert images into QR codes, which provides branding opportunities and greater user engagement.

• QRGateway: Offers advanced functions to create QR codes with images, such as access to promotions, itineraries or product information.

• Canva: Although it does not specialize in QR codes, Canva provides a free generator to create custom QR codes, including images.

• My QR Code: Provides a QR code generator with options to add logos, colors and styles to your QR codes.

In short, the combination of images and QR codes has been adopted by various tools and platforms, offering creative opportunities in marketing, art, and more.

The first QR code with an embedded image to gain public notoriety was not necessarily that of the BBC in London. In fact, the concept of inserting images or logos within QR codes was popularized on various QR code generation platforms such as QRhacker and Pageloot, which allowed users to personalize their codes with photos and logos to improve aesthetics and brand recognition.

These tools not only allowed inserting images, but also modifying the colors and arrangement of the pixels of the code, which led to the creation of more attractive and functional QR codes. Although the creation of the first image QR code cannot be attributed to a single entity, the technology and services to do so were developed around 2012.

However, some of the early notable examples and tools that popularized this technique are:

• Amit Agarwal of Digital Inspiration: Amit Agarwal is known for his technology innovations and tutorials, including customizing QR codes with images and logos. His 2012 article on how to embed images in QR codes helped spread this practice.

• Platforms like QRhacker and Pageloot: These tools have allowed users to create custom QR codes with images since the early 2010s. QRhacker, for example, offered advanced QR code editing options, including the ability to embed photos.

4. Theoretical Framework on the Creation of Color QR Codes (QRGB)

The use of colors to increase the capacity of QR codes has been the subject of several academic studies. Kato and Tan (2007) [1] in their study "Pervasive 2D Barcodes Using Color Information" explore how QR codes can use colors to increase information density, discussing the possibility of using multiple colors to represent more data compared to traditional QR codes. in black and white [1]. Liu and Qiao (2011) in "Enhancing QR Code Capacity with Color" discuss methods for increasing the data capacity of QR codes using color, presenting an approach that encodes additional information in different color channels and evaluating the effectiveness and limitations of this technique [2]. Choi and Woo (2012) in "Data Encoding Technique Using Color QR Code" propose a method for encoding additional data into QR codes using colors, including a comparative analysis with traditional QR codes and demonstrating how the use of color can significantly improve the ability of data [3]. Fang (2011) discusses offline QR code authorization based on visual cryptography, suggesting the use of color techniques to improve security and capacity [4]. Fu, Cheng, Liu, and Yu (2019) present a two-level information protection scheme using visual cryptography and QR codes with multiple decryptions, highlighting the usefulness of colors in data encoding for information protection [5]. Lin (2016) develops a distributed secret sharing approach with

cheater prevention based on QR codes, exploring the use of colors to increase security and informativeness [6]. Liu, Yan, and Pan (2019) investigate color visual secret sharing for QR codes with perfect module reconstruction, demonstrating how colors can improve the density and security of QR codes [7]. Tan, Liu, Yan, Wan, and Chen (2018) propose a visual secret sharing scheme for color QR codes, evaluating the effectiveness of this technique in improving information capacity [8]. Mishra (2016) in his thesis "Region Identification and Decoding of Security Markers Using Image Processing Tools" also addresses the use of image and color processing techniques in the identification and decoding of security markers, providing additional context for the use of colors in QR codes [9]. These studies show that the use of colors in QR codes can significantly increase information density. However, the practical implementation of these techniques has not been widely adopted in the commercial field, and more research is still required to overcome the technical and scannability challenges associated with color QR codes.

On the other hand, the idea of overlaying three QR codes using RGB colors, as described here, could be an innovative extension of these concepts, offering greater storage capacity in a single QR code.

In today's digital age, the need to store and transmit large amounts of information efficiently has led to the development of advanced technologies such as colored QR codes (QRGB). This section presents a detailed rationale for why QRGBs represent a significant improvement over traditional QR codes.

• Definition and Limitations of Traditional QR Codes: A QR (Quick Response) code is a type of two-dimensional barcode that can store information efficiently and quickly. It was created in 1994 by the Japanese company Denso Wave for the automotive industry, although its use has spread to many other areas due to its versatility and storage capacity. Traditional QR codes are made up of a matrix of black and white modules (dots) that represent the encoded data (QRGB).

• Justification of Colored QR Codes (QRGB): Traditional black and white QR codes are limited by their data storage capacity. By using three layers of colors (red, green and blue), QRGBs can store up to three times more information in the same space. This is because each color can represent a different set of data, allowing information to be overlaid without increasing the physical size of the code. QRGBs not only increase storage capacity, but also improve information security. Overlaying multiple layers of data can make it difficult to forge or manipulate code. In addition, by having multiple channels of information, redundancy can be implemented, which increases the robustness of the code against damage or reading errors (QRGB).

• Applications and Potential of QRGBs: QRGBs are especially useful in applications where the transmission of large amounts of data in limited spaces is required, such as in the packaging industry,

digital business cards and interactive advertising. They also have potential in areas such as document and banknote security, where the authenticity and integrity of information are crucial. Although QRGBs are based on the RGB color model, compatibility with printers using the CMYK model has been considered, ensuring that the codes maintain their integrity and are readable even when printed (QRGB).

• Development of Decoding Algorithms: The development of advanced decoding algorithms that can identify and separate the different color layers is an essential component of QRGBs. These algorithms allow current scanning devices, with minor software modifications, to accurately read and decode the information stored in QRGBs. The introduction of QRGB represents a significant advance in QR code technology, offering substantial improvements in storage capacity, security and applicability (QRGB).

5. Theory of the Additive RGB Color System and its Implementation in Color QR Codes (QRGB)
The additive RGB color system is based on the combination of red (Red), green (Green) and blue (Blue) light to create a wide range of colors. Combining these three colors in different intensities can produce any color in the visible spectrum.

The principle behind the RGB system is based on the way the human eye perceives color. Our eyes have three types of receptor cells, known as cones, that are sensitive to red, green and blue wavelengths. When light enters the eye, these cells activate to different degrees depending on the wavelength of the light, and the brain interprets the signals from these cells as color.

In the additive system, colors are created by adding light of different colors.

The primary colors of the RGB system (red, green and blue) are mixed to produce other colors by adding their intensities: Red + Green = Yellow, Red + Blue = Magenta, Green + Blue = Cyan, Red + Green + Blue = White. Each of these secondary colors is the result of the superimposition of two of the primary colors. When the three primary colors are mixed at their maximum intensity, they produce white light.

The RGB system is used in various technologies and applications, mainly in devices that emit light. Some examples include: electronic displays, computer monitors, televisions and mobile phone screens use the RGB system to produce color images. In image projection, video projectors use RGB lamps and filters to project color images onto a screen. LED lighting allows the creation of a wide range of colors by adjusting the intensity of the red, green and blue light-emitting diodes.

The use of the RGB additive system is theoretically justified by the nature of light and the way it interacts with the receptors in the human eye. Visible light is a small part of the electromagnetic spectrum and is made up of waves of different lengths. The cones in

our eyes are sensitive to these different wavelengths and allow us to see colors. Red Sensitive Cones (L): Sensitive to long wavelengths (~564–580 nm). Green Sensitive Cones (M): Sensitive to medium wavelengths (~534–545 nm). Blue Sensitive Cones (S): Sensitive to short wavelengths (~420–440 nm). The combination of light from these three primary colors in different proportions allows our brain to perceive a wide range of colors.

The additive RGB color system (Red, Green, Blue) is a model used to create colors in electronic devices by combining light at different intensities, represented by values from 0 to 255. The primary colors are: Red (255, 0, 0), Green (0, 255, 0) and Blue (0, 0, 255). By mixing these colors you get: Red + Green = Yellow (255, 255, 0), Red + Blue = Magenta (255, 0, 255), and Green + Blue = Cyan (0, 255, 255). The combination of all colors produces White (255, 255, 255), while the absence of light generates Black (0, 0, 0). Other colors can also be obtained such as Orange (255, 165, 0) by mixing red and green, Light Green (144, 238, 144) and Light Blue (173, 216, 230) with specific proportions. The RGB system allows you to create a wide range of colors, essential for digital visualization and graphic design. But we will only focus on this one:
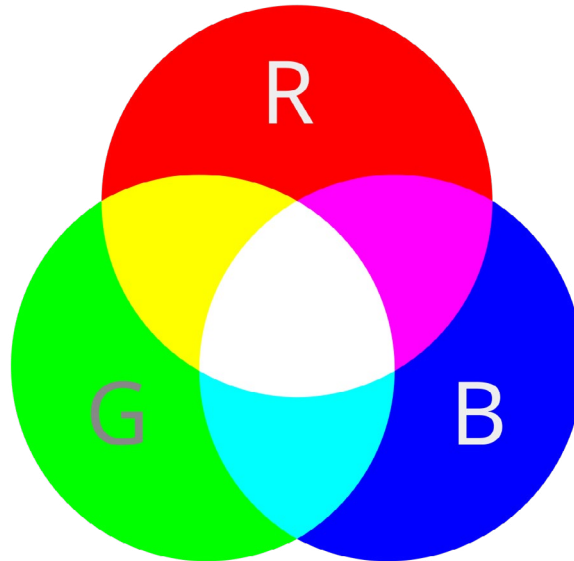
Figure 2: Additive RGB Color System

Fountain: https://en.wikipedia.org/wiki/RGB_color_model#/media/File:Venn_diagram_rgb.svg
For more information visit: https://en.wikipedia.org/wiki/RGB_color_model

6. Methodology to be Implemented in the Creation and Reading of QRGB Codes

Three individual QR codes are created in the colors red, green and blue, each encoding different parts of the information. These QR codes are generated using Python tools and libraries like PyQRCode and OpenCV. QR codes in red, green and blue overlap to form a single colored QR code. This overlay process is done in the RGB color space, combining the three layers into a single image. For encoding, you work in the CMYK color space to ensure that colors are represented correctly when printing the QR code. Decoding is performed in the RGB color space, using advanced algorithms to separate the different color layers and extract the encoded information. Image processing techniques such as segmentation and thresholding are used to identify and process the color modules in the QR code. This involves analyzing entire modules rather than individual pixels, ensuring better decoding accuracy. Specific algorithms are developed for the decoding of QRGB codes, which can identify and separate the color layers. These algorithms must be able to handle color variations caused by printing and other environmental factors. It ensures that QRGB codes are compatible with current scanning devices, allowing them to read and decode the data stored in the color codes. QRGB codes are tested in various applications, such as document security, digital business cards, and interactive advertising, to validate their effectiveness and security. Continuous evaluation is carried out to improve the methodology and ensure that the codes are robust and reliable in different scenarios.

7. Development in Repli.it Python Programming Language

Replit.it is an online platform that allows users to write, run, and collaborate on code in various programming languages, including Python. It is especially useful for learning to program, making rapid prototypes, and collaborating on projects easily.

Replit.it Python features are:
• IDE in the Cloud: You don't need to install anything on your computer. You can code from anywhere with Internet access.
• Collaboration: Allows multiple users to work on the same project in real time.
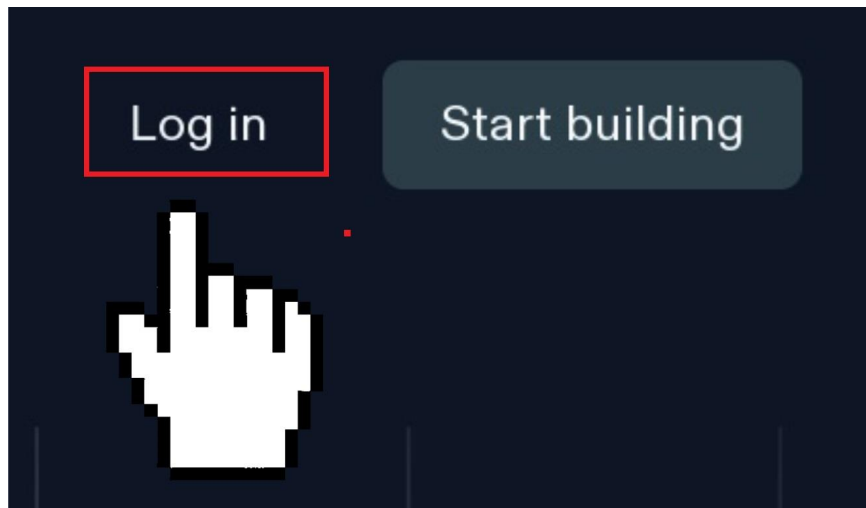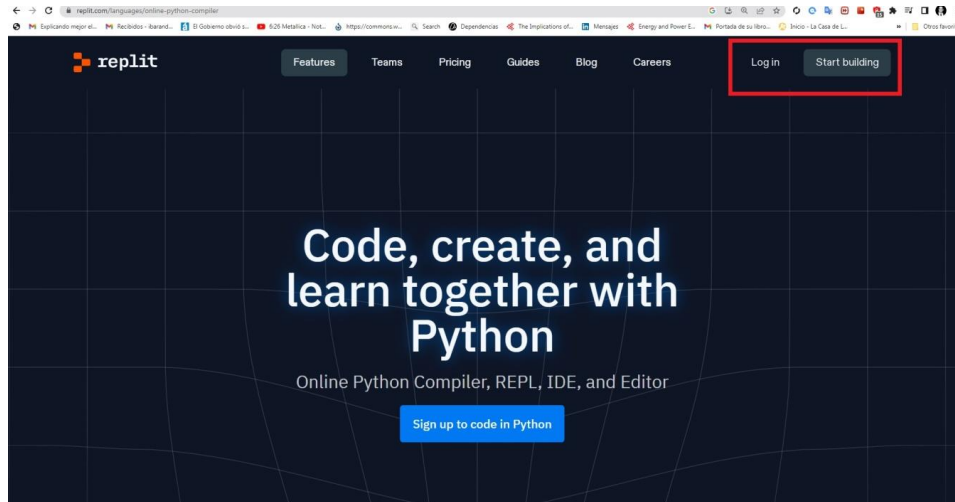• Support for Multiple Languages: In addition to Python, it

supports many other languages such as JavaScript, Ruby, HTML/CSS, among others.

• Packages and Libraries: You can easily install libraries using the terminal, such as pip for Python.

• Simple Deployment: You can create web applications and easily share them with others.

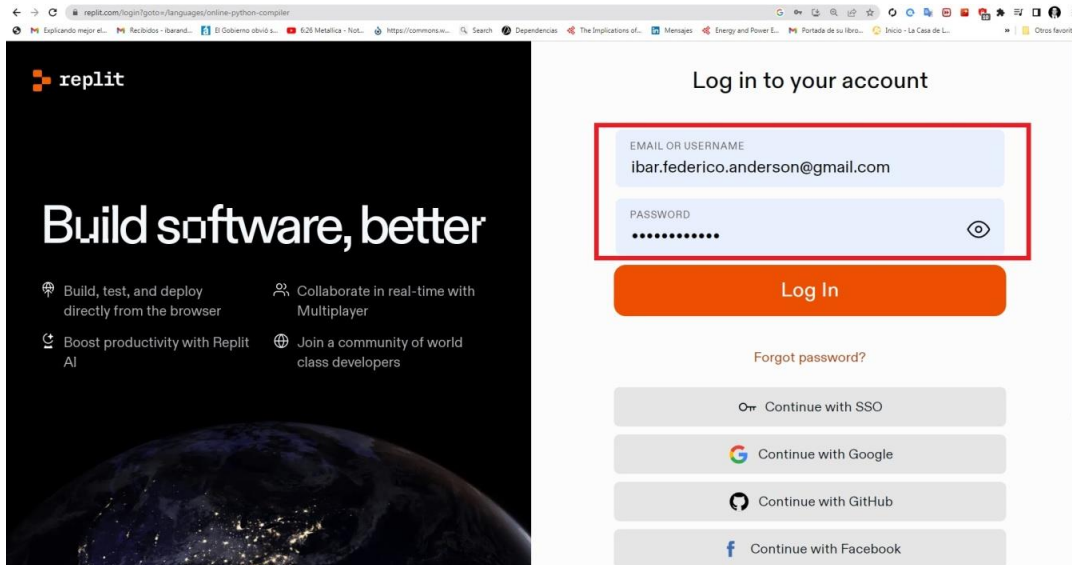You can access Replit and start using Python through the following link:

https://replit.com/languages/online-python-compiler

Enter Replit through the mentioned link, there you must log in with your Gmail account. Once inside, you will be able to select from several simulations, programs or online programming environments (select Python).

Figures 3 and 4: Repli.it Python

Fountain: https://replit.com/languages/online-python-compiler

Figures 5 and 6: Repli.it Python

Fountain: https://replit.com/languages/online-python-compiler

Figure 7: Repli.it Python

Fountain: https://replit.com/languages/online-python-compiler
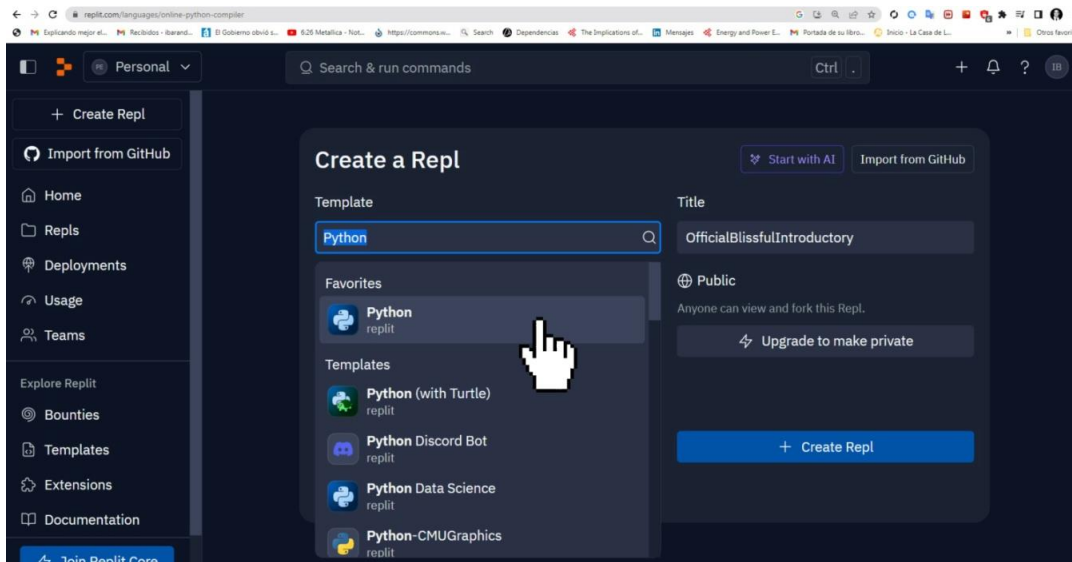
8. Shell Libraries that you will need to Install in Repli.it Python

You will first need to install the Python libraries. To use these libraries, open your Replit.it Python project (main.py) and go to the tab (in the left panel), find and click the "Shell" tab (to the right of "Console").
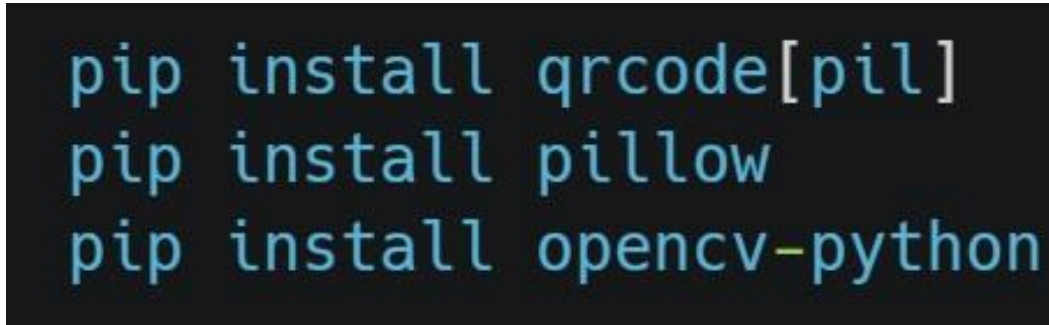
Figure 8: Python open source libraries: pip install qrcode [pil], pip install pillow, pip install opencv-python. Repli.it Python

Fountain:https://replit.com/languages/online-python-compiler
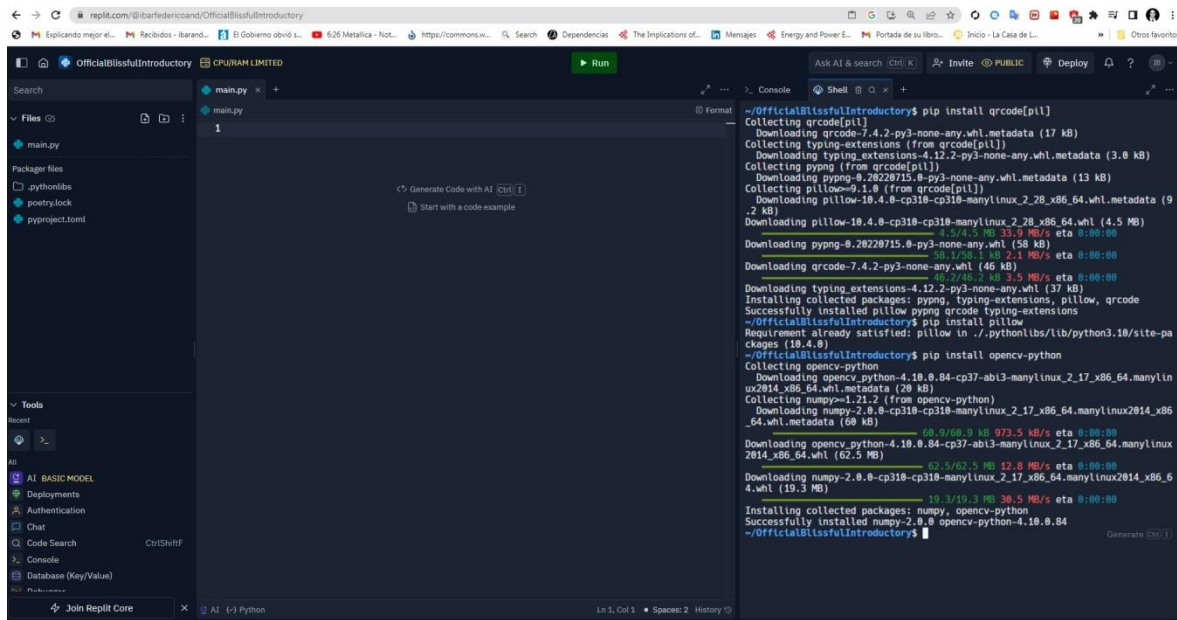
Figure 9: Installing the open source libraries in the Repli.it Python "Shell": Pip install qrcode [pil], pip install pillow, pip install opencv-python

Fountain:https://replit.com/languages/online-python-compiler

• pip install qrcode[pil]: This command installs the qrcode library, which is used to generate QR codes. The [pil] option indicates that the Pillow dependency must also be installed, which is an image manipulation library required to work with images generated by qrcode.

• pip install pillow: This command installs Pillow, a Python library for opening, manipulating, and saving different image formats. It is very useful for working with images in projects that involve graphics or visualization.

• pip install opencv-python: This command installs OpenCV, a powerful library for computer vision. It is used to perform tasks such as image processing and object detection. It is very versatile and widely used in image analysis projects.

9. Program to Create (encode) and Read (decode) QRGB Codes (overlapping) in Repli.it Python

Below I show you a single program that encodes and decodes:

```
import tkinter as tk
from tkinter import simpledialog, messagebox, filedialog
from PIL import Image, ImageTk
import qrcode
import you
import cv2

# Function to create a QR code with a superimposed logo
def create_qr_with_logo(data, color, logo_path, qr_version=10, box_size=10):
qr = qrcode.QRCode(
```

```python
    version=qr_version,
    error_correction=qrcode.constants.ERROR_CORRECT_H,
    box_size=box_size,
    border=4
)
qr.add_data(data)
qr.make(fit=True)

img = qr.make_image(fill_color=color, back_color="white").convert('RGBA')

if not os.path.exists(logo_path):
    raise FileNotFoundError(f"Logo file not found: {logo_path}")

logo = Image.open(logo_path).convert("RGBA")
basewidth = img.size[0] // 4
wpercent = (basewidth / float(logo.size[0]))
hsize = int((float(logo.size[1]) * float(wpercent)))
logo = logo.resize((basewidth, hsize), Image.LANCZOS)

pos = ((img.size[0] - logo.size[0]) // 2, (img.size[1] - logo.size[1]) // 2)
img.paste(logo, pos, logo)

return img

# Combine QR images ensuring they all have the same size
def combine_qr_images(img1, img2, img3, logo_path):
    size = img1.size
    if img2.size != size or img3.size != size:
        raise ValueError("All QR images must be the same size")

final_image = Image.new("RGBA", size, "black")

data_red = img1.getdata()
data_green = img2.getdata()
data_blue = img3.getdata()

new_data = []
for i in range(len(data_red)):
    r1, g1, b1, a1 = data_red[i]
    red_pixel = (r1, g1, b1) != (255, 255, 255)
    r2, g2, b2, a2 = data_green[i]
    green_pixel = (r2, g2, b2) != (255, 255, 255)
    r3, g3, b3, a3 = data_blue[i]
    blue_pixel = (r3, g3, b3) != (255, 255, 255)

    if red_pixel and green_pixel and blue_pixel:
        new_data.append((255, 255, 255, 255))
    elif red_pixel and green_pixel:
        new_data.append((255, 255, 0, 255))
    elif red_pixel and blue_pixel:
        new_data.append((255, 0, 255, 255))
    elif green_pixel and blue_pixel:
        new_data.append((0, 255, 255, 255))
    elif red_pixel:
        new_data.append((255, 0, 0, 255))
    elif green_pixel:
        new_data.append((0, 255, 0, 255))
    elif blue_pixel:
        new_data.append((0, 0, 255, 255))
    else:
        new_data.append((0, 0, 0, 255))

final_image.putdata(new_data)

logo = Image.open(logo_path).convert("RGBA")
basewidth = final_image.size[0] // 4
wpercent = (basewidth / float(logo.size[0]))
hsize = int((float(logo.size[1]) * float(wpercent)))
logo = logo.resize((basewidth, hsize), Image.LANCZOS)

pos = ((final_image.size[0] - logo.size[0]) // 2, (final_image.size[1] - logo.size[1]) // 2)
final_image.paste(logo, pos, logo)

return final_image

def generate_qrgb(red_data, green_data, blue_data, logo_path, mode):
    qr_version = 10 if mode == 'link' else 3 # Version for link or manual
    box_size = 10 if mode == 'link' else 20 # Box size for link or manual

    img_red = create_qr_with_logo(red_data, "red", logo_path, qr_version, box_size)
    img_green = create_qr_with_logo(green_data, "green", logo_path, qr_version, box_size)
    img_blue = create_qr_with_logo(blue_data, "blue", logo_path, qr_version, box_size)

    img_red.save("qr_red.png")
    img_green.save("qr_green.png")
    img_blue.save("qr_blue.png")

    combined_img = combine_qr_images(img_red, img_green, img_blue, logo_path)
    combined_img.save("superposed_qr.png")

    return combined_img

def show_qrgb_image(img):
    top = tk.Toplevel()
    top.title("QRGB Code")
    img = img.resize((300, 300), Image.LANCZOS)
    img_tk = ImageTk.PhotoImage(img)
    lbl = tk.Label(top, image=img_tk)
    lbl.image = img_tk
    lbl.pack(pady=20)
```

```
top.mainloop()

def get_data(mode):
root.withdraw()

if mode == 'manual':
red_data = simpledialog.askstring("Red Layer", "Enter the text for
the red layer:")
green_data = simpledialog.askstring("Green Layer", "Enter the
text for the green layer:")
blue_data = simpledialog.askstring("Blue Layer", "Enter the text
for the blue layer:")
elif mode == 'link':
red_data = input("Enter the link for the red layer: ")
green_data = input("Enter the link for the green layer: ")
blue_data = input("Enter the link for the blue layer: ")
else:
messagebox.showwarning("Invalid Mode", "Mode not recog-
nized. Please select 'Manual' or 'Link'.")
root.deiconify()
return

if red_data and green_data and blue_data:
logo_filename = "Logo.png"
logo_path = os.path.abspath(logo_filename)
try:
qrgb_image = generate_qrgb(red_data, green_data, blue_data,
logo_path, mode)
show_qrgb_image(qrgb_image)
except FileNotFoundError as e:
messagebox.showerror("Error", str(e))
except ValueError as e:
messagebox.showerror("Error", str(e))
else:
messagebox.showwarning("Incomplete Data", "All fields must be
filled.")
root.deiconify()

def manual_mode():
get_data('manual')

def link_mode():
get_data('link')

# Function to read a QR code from an image
def read_qr(filename):
img = cv2.imread(filename)
detector = cv2.QRCodeDetector()
data, vertices_array, _ = detector.detectAndDecode(img)
if vertices_array is not None:
return data
else:
return None

# Function to manually decode the superimposed QR
```

```
def manual_decode_superposed_qr(filename):
superposed_img = Image.open(filename)
superposed_data = superposed_img.getdata()

size = superposed_img.size
red_data = [(255, 255, 255, 255)] * len(superposed_data)
green_data = [(255, 255, 255, 255)] * len(superposed_data)
blue_data = [(255, 255, 255, 255)] * len(superposed_data)

for i in range(len(superposed_data)):
r, g, b, a = superposed_data[i]
if r != 0: # Network
red_data[i] = (0, 0, 0, 255)
if g != 0: # Green
green_data[i] = (0, 0, 0, 255)
if b != 0: # Blue
blue_data[i] = (0, 0, 0, 255)

red_img = Image.new("RGBA", size)
green_img = Image.new("RGBA", size)
blue_img = Image.new("RGBA", size)

red_img.putdata(red_data)
green_img.putdata(green_data)
blue_img.putdata(blue_data)

red_img.save("decoded_red.png")
green_img.save("decoded_green.png")
blue_img.save("decoded_blue.png")

data_red = read_qr("decoded_red.png")
data_green = read_qr("decoded_green.png")
data_blue = read_qr("decoded_blue.png")

return data_red, data_green, data_blue

def decode_qr():
root.withdraw()

qr_filename = filedialog.askopenfilename(
title="Select the superimposed QRGB code",
filetypes=[("PNG files", "*.png"), ("All files", "*.*")]
)

if qr_filename:
try:
data_red, data_green, data_blue = manual_decode_superposed_
qr(qr_filename)
# Show the decoded data in the popup window
response = messagebox.askokcancel("Decoding successful",
f"Red layer data: {data_red}\nGreen layer data: {data_green}\
nBlue layer data: {data_blue}\n\nDo you want to print this data
to the console? ")
if response:
# Show the decoded data in the console
```

```
print(f"Red layer data: {data_red}")
print(f"Green layer data: {data_green}")
print(f"Blue layer data: {data_blue}")
except Exception as e:
messagebox.showerror("Error", str(e))
else:
messagebox.showwarning("File not selected", "Please select a QR
code file.")

#Clear the current window
root.deiconify()

# Function to show the main menu
def show_main_menu():
root.withdraw()
top = tk.Toplevel()
top.title("Select option")

label = tk.Label(top, text="Select an option:", font=("Arial", 14))
label.pack(pady=10)

btn_encode = tk.Button(top, text="Encode QRGB", com-
mand=open_encode_menu, font=("Arial", 12))
btn_encode.pack(pady=5)

btn_decode = tk.Button(top, text="Decode QRGB", com-
mand=open_decode_menu, font=("Arial", 12))
btn_decode.pack(pady=5)

top.mainloop()

# Function to open the QRGB encoding window
```

```
def open_encode_menu():
#Clear the current window
root.withdraw()
top = tk.Toplevel()
top.title("Create QRGB Code")

label = tk.Label(top, text="Create QRGB Code:", font=("Arial",
14))
label.pack(pady=10)

btn_manual = tk.Button(top, text="Manual", command=manual_
mode, font=("Arial", 12))
btn_manual.pack(pady=5)

btn_link = tk.Button(top, text="Link", command=link_mode,
font=("Arial", 12))
btn_link.pack(pady=5)

top.mainloop()

# Function to open the QRGB decoding window
def open_decode_menu():
decode_qr()

# GUI configuration
root = tk.Tk()
root.title("QRGB Generator and Decoder")

show_main_menu() # Show the main menu at startup

root.mainloop()
```


Figure 10: We paste (Copy Paste) the code (script) in main.py. Source: Own development of the QRGB encoding code in Python
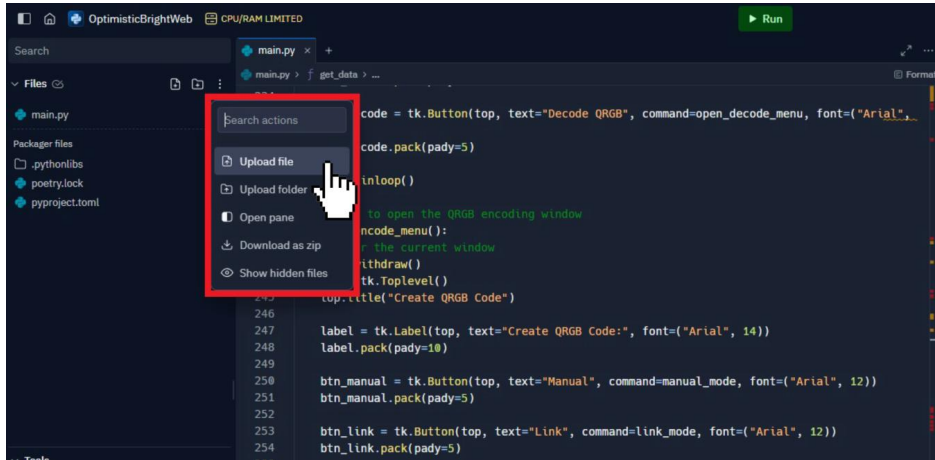
Figure 11: Before running the script (Run) pasted in main.py, the "Logo" must be uploaded in .png format. Source: self-made
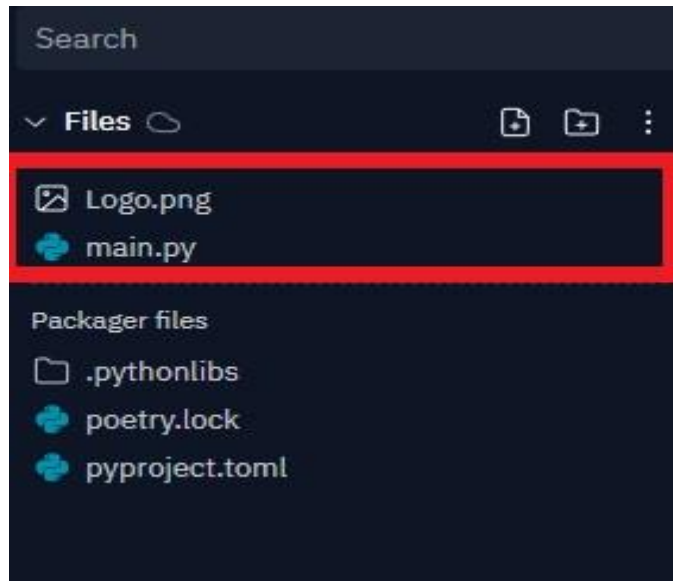

Figure 12: The image of the "Logo" in .png format should be seen next to main.py as seen in this image. Source: self-made
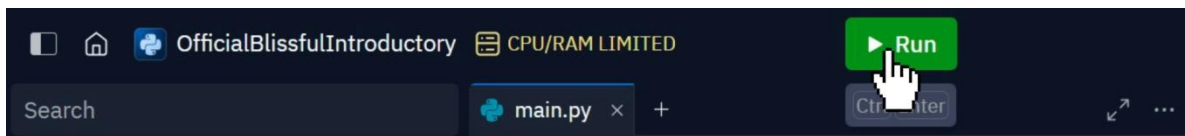

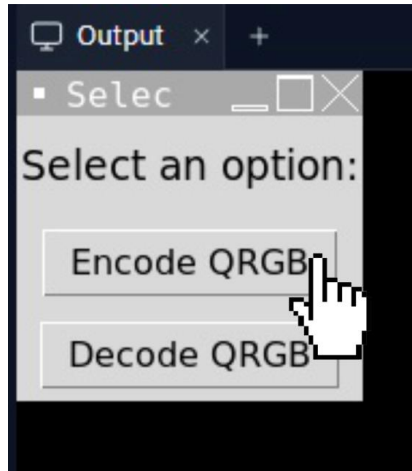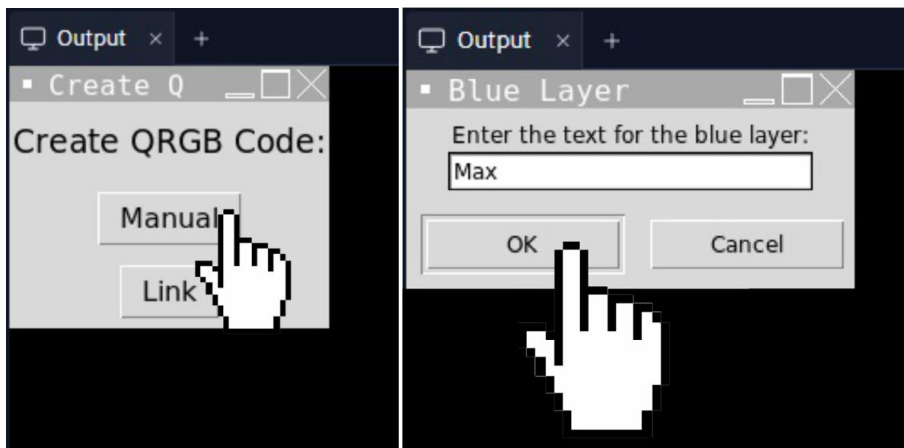**Figure 13: Own development of the QRGB encoding code in Python and its execution (Run)**

Figure 14: If the "Encode QRGB" option is selected, it is directed to the QRGB encoding, where two (2) options will appear: "Manual" and "Link". Source: self-made

Figures 15 and 16: If the "Manual" option is selected, it is directed to the QRGB encoding manually where it will request that the text be entered for the red, green and blue layers independently (having to click OK after each data entry), example (names were entered of individuals: Jack, Tom, Max). Source: self-made
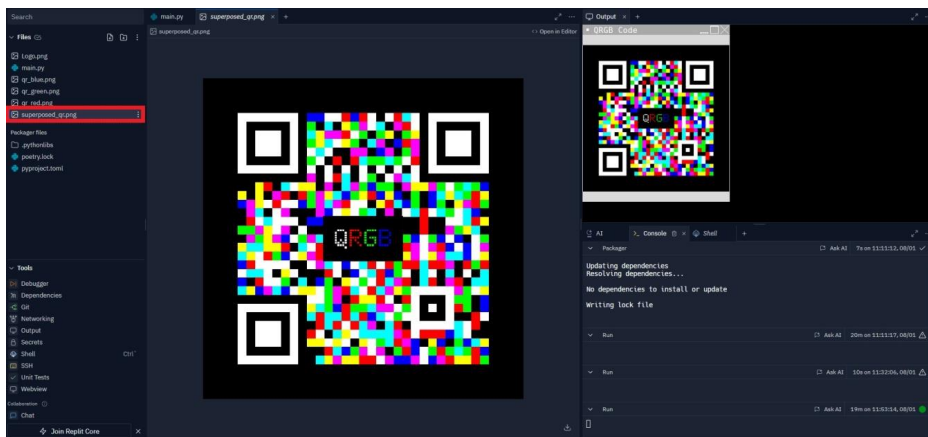

Figure 17: The QRGB code was generated in the Output with the image Logo (.png) in the center and on the left you can see the creation of the file "superposed_qr.png" (downloadable of the created or generated QRGB code). Source: self-made

Indeed, after running (Run) in the Console, the questions (imput) will appear to enter the information of the first, second and third QR code, which in these cases are my Google Scholar, Researchgate and Academia.edu profiles respectively.

Enter the information for the first QR code (Red):
https://scholar.google.com/citations?user=WfLtjeoAAAAJ&hl=en

Enter the information for the second QR code (Green):
https://www.researchgate.net/profile/Ibar-Federico-Anderson

Enter the information for the third QR code (Blue):
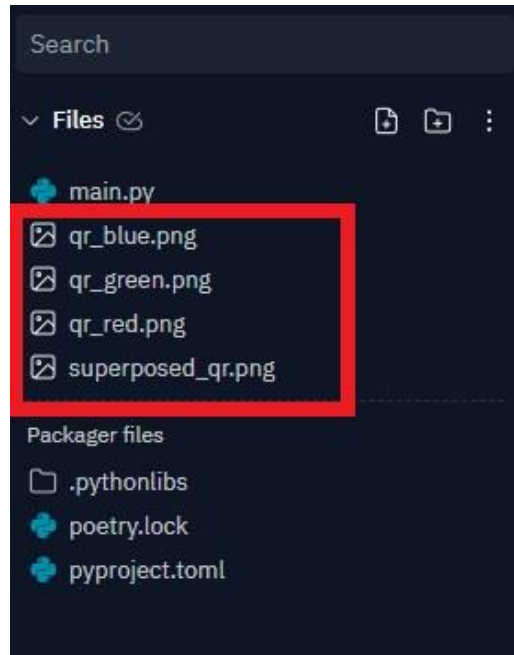https://unlp.academia.edu/IbarFedericoAnderson

Figure 18: The Python code generates the intermediate step of three (3) QR Codes in RGB colors (Red, Green and Blue) in image format (.png), with the data entered in "Console". Source: Own development of the QRGB encoding code in Python
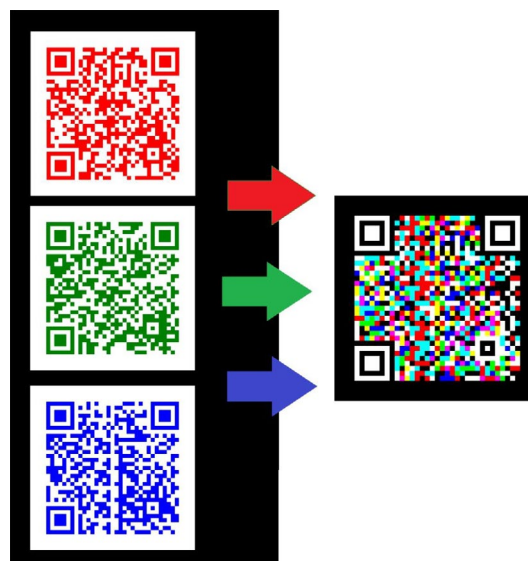

Figure 19: With the intermediate step of three (3) QR Codes in RGB colors (Red, Green and Blue) in image format (.png), the pixels to generate the modules are processed (the pixels to generate the modules) of the final QRGB code. Source: self-made

Figure 20: QRGB code generated in image format (.png) with the image Logo in the center (any image can be uploaded in .png format). Source: self-made

On the contrary, if instead of "Manual" "Link" is selected when creating the QRGB Code.

Figure 20: QRGB code generated in image format (.png) with the image Logo in the center (any image can be uploaded in .png format). Source: self-made
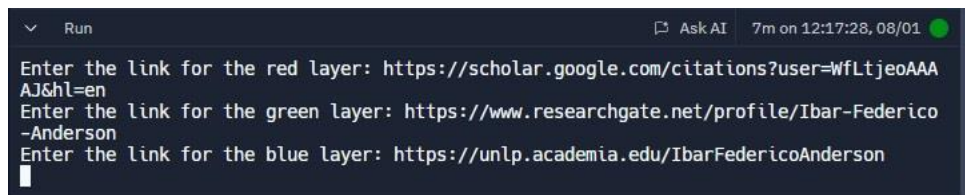

Figure 22: Entry through "Console" of links to web pages,which in these cases are my Google Scholar, Researchgate and Academia.edu profiles respectively

Enter the link for the network layer: https://scholar.google.com/citations?user=WfLtjeoAAAAJ&hl=en
Enter the link for the green layer: https://www.researchgate.net/profile/Ibar-Federico-Anderson
Enter the link for the blue layer: https://unlp.academia.edu/IbarFedericoAnderson
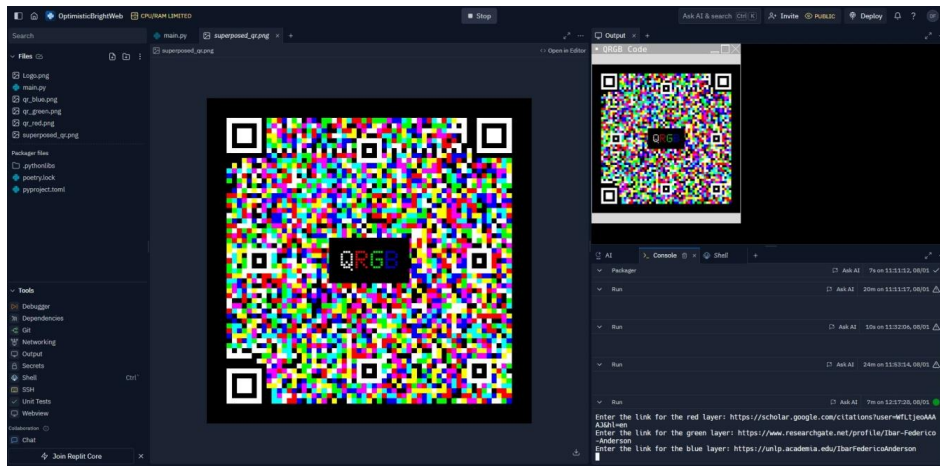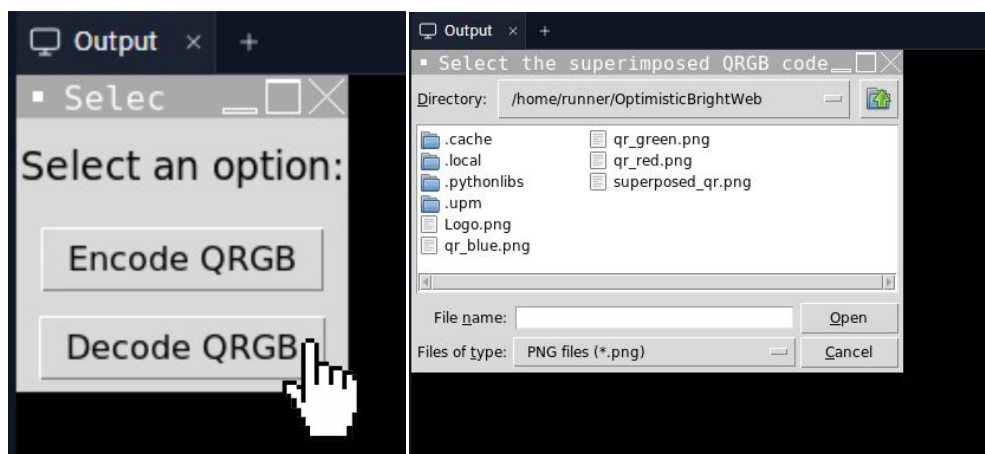
Figure 23: This is how the QRGB Code looks with links inside. Source: self-made

Figure 24: This is how the QRGB Code looks with links inside. Source: self-made

Now if what you want is to decode the QRGB Code, "Stop" the program and "Run" again and the following image will appear. "Decode QRGB" must be selected.

Figures 25 and 26: After selecting "Decode QRGB", the options will appear in the Output (you must select "superposed_qr.png") and click "Open". Source: self-made
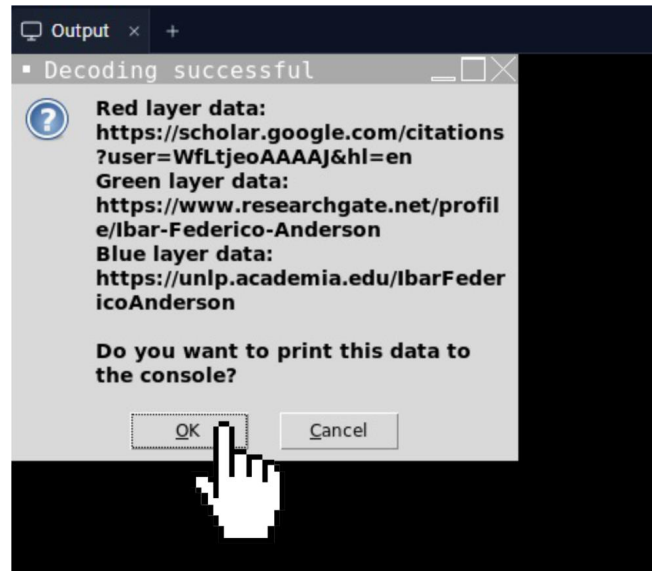
Figure 27: This is what the Output shows when it decodes the code and when "OK" is given it will show the information in the "Console". Source: self-made
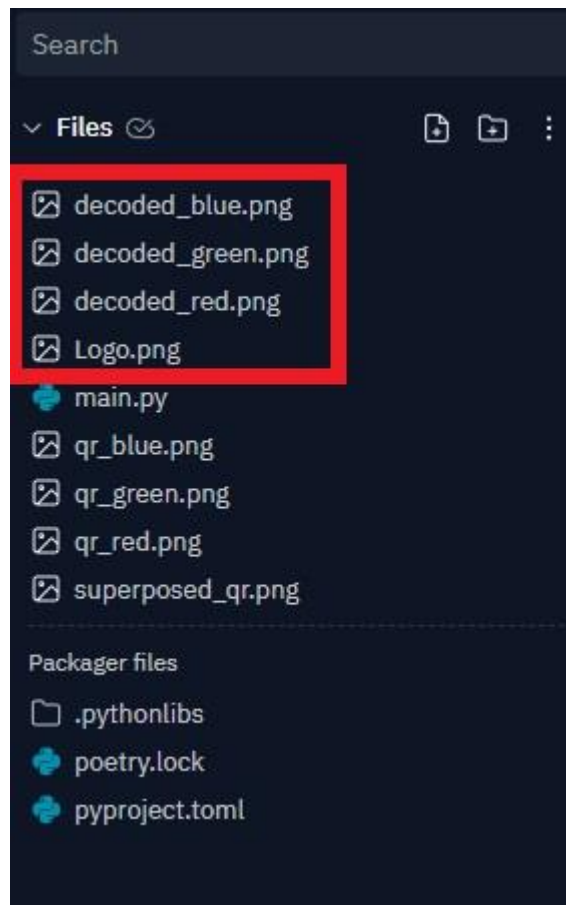

Figure 28: The Python code generates the intermediate step of decoding the three (3) QR Codes in RGB colors (Red, Green and Blue) in image format (.png), with the data entered in "Console". Source: Own development of the QRGB encoding code in Python

Figure 29: The script code after "OK" in Output shows decoded in "Console" the same information entered by "Console" and encoded. Source: self made

10. Conclusion

The concept described in the QRGB.docx file, which combines three QR codes into one using RGB color coding to increase information density, is an innovative idea and is not widely known in commercial applications or standard systems. However, there are some developments and concepts in similar areas that deserve mention.

High Capacity Colored Two-Dimensional Codes (HCC2D) is a system developed to increase the capacity of QR codes using colors. Each point in the code can represent more information by having a specific color. It uses colors to increase the information capacity in a two-dimensional code. However, generally, it does not focus on overlaying three different QR codes into one, but on encoding more information at each point of a single QR code.

Microsoft Tag is a 2D code system that uses colors to encode information, developed by Microsoft. Use colors to encode additional information. However, it is a different system from the standard QR and does not involve the superimposition of multiple QR codes into a single one.

There are academic studies that have explored the use of colors to increase the capacity of QR codes, but the practical implementation of these studies has not been widely adopted or commercialized.

The idea of superimposing three QR codes using RGB colors to create a single QR code with greater information density is quite novel and does not seem to have an exact implementation in currently known commercial systems. Although there is research and proposals on the use of colors to increase the capacity of QR codes, the specificity of combining three QR codes into one by overlaying RGB colors seems to be unique.

This approach may offer a new way to increase data density in a single QR code, which could be very useful in applications that require storing large amounts of information in small spaces.

Colored QR codes (QRGB) present an innovative solution to the growing demands for data storage and transmission in various sectors. By significantly improving storage capacity and security, QRGBs represent a significant technological advance over traditional QR codes (QRGB).

In today's digital age, the need to store and transmit large amounts of information efficiently has led to the development of advanced technologies such as colored QR codes. Below is a detailed rationale for why QRGBs represent a significant improvement over traditional QR codes.

Traditional black and white QR codes are limited by their data storage capacity. By using three layers of colors (red, green and blue), QRGBs can store up to three times more information in the same space. This is because each color can represent a different set of data, allowing information to be overlaid without increasing the physical size of the code.

QRGBs not only increase storage capacity, but also improve information security. Overlaying multiple layers of data can make it difficult to forge or manipulate code. Additionally, by having multiple channels of information, redundancy can be implemented, which increases the robustness of the code against damage or read errors.

QRGBs are especially useful in applications where the transmission of large amounts of data in limited spaces is required, such as in the packaging industry, digital business cards, and interactive advertising. They also have potential in areas such as document and banknote security, where the authenticity and integrity of information are crucial.

Although QRGBs are based on the RGB color model, compatibility with printers that use the CMYK model has been considered. This ensures that the codes maintain their integrity and are legible even when printed, overcoming one of the main challenges of implementing colored QR codes in the physical world.

The development of advanced decoding algorithms that can identify and separate the different color layers is an essential component of QRGBs. These algorithms allow current scanning devices, with

minor software modifications, to accurately read and decode the information stored in QRGBs.

The introduction of QRGBs represents a significant advance in QR code technology, offering substantial improvements in storage capacity, security and applicability. These codes are an innovative solution to the increasing demands for data storage and transmission in various sectors.

Observed that it is an innovative proposal that seeks to increase the density of information stored in QR codes using an additive generation method of RGB colors. Several important points about the content, its strengths, areas for improvement and potential future applications are presented here.

The QRGB proposal is very innovative (it has not been possible to find open source developments in software that allows other people to create, encode and decode it in the way in which it is freely presented here to the experience of other users, but with intellectual property). Allowing – as has already been said – to store up to three times more information in the same physical space by superimposing layers of colors (red, green and blue). And also as already said, this technique not only increases storage capacity, but also improves security, making it difficult to forge or manipulate QR codes.

Emphasizing, as already mentioned above, that QRGBs have potential applications in various sectors such as the packaging industry, digital business cards, interactive advertising and document security. The possibility of applying this technology in areas where the authenticity and integrity of information are crucial, such as banknotes and official documents, highlights its practical value.

The implementation using Python libraries such as qrcode[pil], Pillow and opencv-python is a strength, since it takes advantage of open source tools, facilitating their access and customization. Creating a custom solution due to the lack of specific libraries demonstrates a level of adaptation and technical creativity intermediate between the creation (encoding) and decoding process with red, green and blue QR files (which make an intermediate encoding and decoding process).

The overlay of multiple data layers allows redundancy to be implemented, increasing the robustness of the code against damage or reading errors. In addition, a correct color mix and adequate digital decoding were achieved. Only those who have the app to generate, transmit and decode it can use it. Those who do not have the app (or the Python code) will not be able to use it, since it is not widely implemented; It is an ongoing development.

Despite numerous strengths, the paper mentions technical challenges related to color mixing and accurate information retrieval in what I have previously defined as intermediate encoding and decoding. A detailed critique and details on how these problems can be solved and what steps are being taken to improve decoding accuracy will be included in a future paper.

On the other hand, ensuring compatibility with current scanners is vital (this work has not been done). The document could benefit from a more detailed section on how existing devices can be adapted to read these new QR codes. The scannability of QR codes in different lighting conditions and on different types of surfaces should be evaluated and improved if necessary.

Although some relevant studies are cited, the document could be enriched with more references to academic works and case studies that have explored the use of colors in QR codes. Including practical examples and real use cases where this technology has been successfully implemented would help strengthen the argument. Creating an intuitive user interface for QRGB generation and reading is essential for its mass adoption. Considering ease of use for both technical and non-technical users is crucial.

Using colors to improve security and storage capacity has great potential in visual cryptography and data protection. Exploring how this technology can be integrated with authentication and verification systems could open new opportunities.

The ability to create visually appealing QR codes with custom graphics and logos offers great potential in marketing and advertising. The ability to include more information in the same QR code can improve user interaction and engagement. In the educational field, QRGBs can be used to provide access to large amounts of information in compact, easy-to-scan formats. This could be especially useful in educational materials, allowing students to access additional resources with a simple scan.

The QRGB document presents an innovative proposal with great potential to revolutionize the use of QR codes in various industries. However, it faces technical challenges that must be addressed to ensure its viability and mass adoption. With improvements in decoding accuracy, compatibility with existing devices and a friendly user interface, this technology has the potential to offer efficient and secure solutions for storing and transmitting information in the digital age. This analysis highlights both the project's strengths and areas where it can be improved, providing a solid foundation for its future development and practical implementation (already in progress in this document) [10,11].

We will continue…

Bibliography

1. Kato, H., & Tan, K. (2007). Pervasive 2D Barcodes Using Color Information. Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops, 2007, 24-29.https://doi.org/10.1109/PERCOMW.2007.84
2. Liu, W., & Qiao, H. (2011). Enhancing QR Code Capacity with Color. *Journal of Information Science and Engineering, 27*(5), 1509-1520.
3. Choi, Y. S., Woo, W. T. (2012). Data Encoding Technique Using Color QR Code. *International Journal of Advanced Computer*

Science and Applications, 3(9), 45-49.

4. Fang, W. P. (2011, October). Offline QR code authorization based on visual cryptography. In *2011 Seventh International Conference on Intelligent Information Hiding and Multimedia Signal Processing* (pp. 89-92). IEEE.

5. Fu, Z., Cheng, Y., Liu, S., & Yu, B. (2019). A new two-level information protection scheme based on visual cryptography and QR code with multiple decryptions. *Measurement, 141*, 267-276.

6. Lin, P. Y. (2016). Distributed secret sharing approach with cheater prevention based on QR code. *IEEE Transactions on Industrial Informatics, 12*(1), 384-392.

7. Liu, T., Yan, B., & Pan, J. S. (2019). Color visual secret sharing for QR code with perfect module reconstruction. *Applied Sciences, 9*(21), 4670.

8. Tan, L., Liu, K., Yan, X., Wan, S., Chen, J., & Chang, C. (2018, June). Visual secret sharing scheme for color qr code. In *2018 IEEE 3rd International Conference on Image, Vision and Computing (ICIVC)* (pp. 961-965). IEEE.

9. Mishra, P. Region Identification and Decoding Of Security Markers Using Image Processing Tools. *Doctoral dissertation.*

10. Anderson, IF (2013). Design of QR Codes for Marketing.

11. Anderson, I. F. (2024). QRGB: App for QR Code Generation (3-in-1 Method), Additive Color Generation Method (RGB), Using Python Programming Code, to Increase Accumulated Information Density.