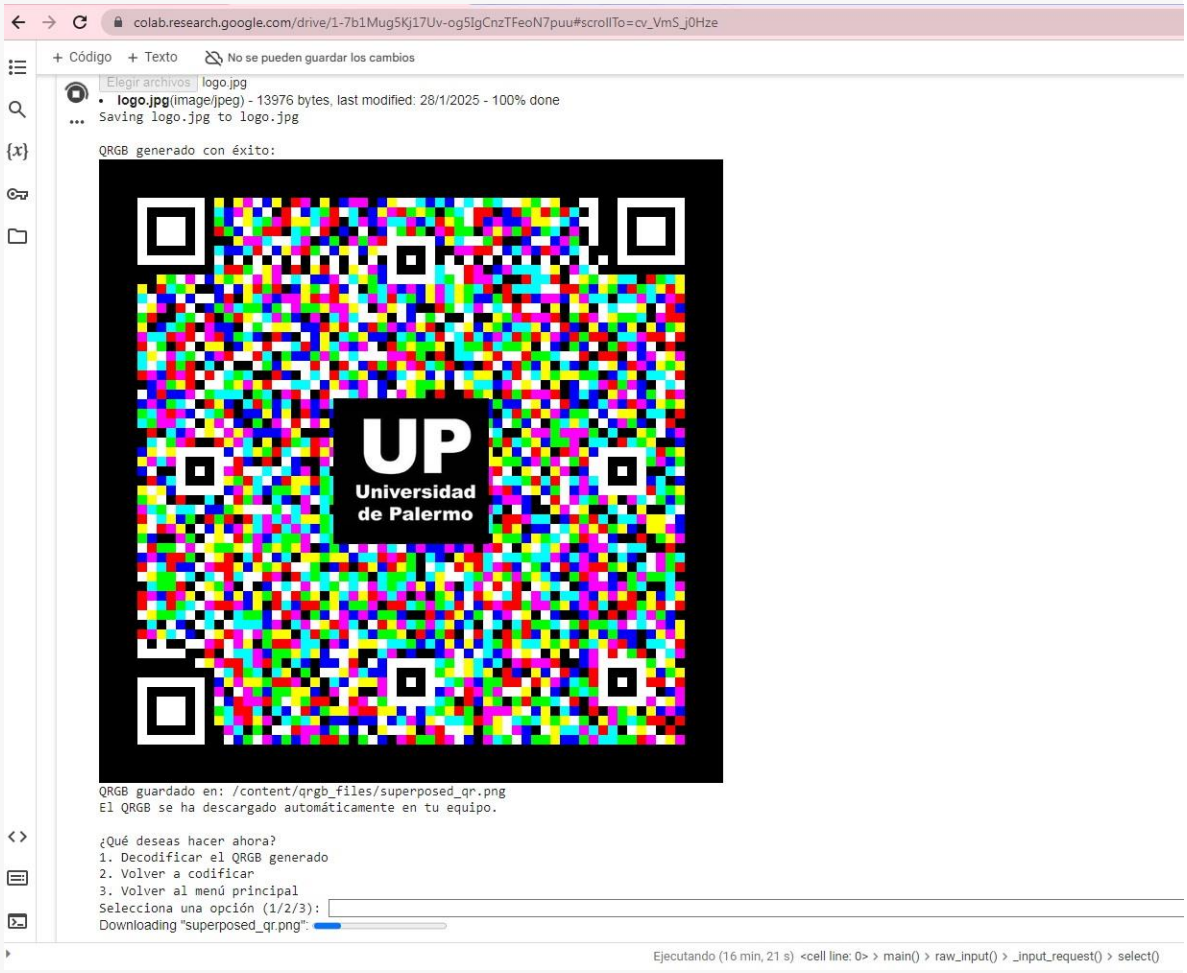
New QR Code App in the additive RGB color system for higher accumulated information density. Free implementation in Google Colab for designers, freelancers, makers and Open Source coders.

Ph.D., Master, Industrial Design: Ibar Anderson.
Email: ianderson@empleados.fba.unlp.edu.ar



1. Abstract.

The decision to use Google Colab for this project is based on the need to make the technology accessible to a wider audience (especially traditional graphic designers – old-school pencil and paper artists –, freelancers and diverse makers and/or budding AI coders), including all those with no prior programming experience. In this work, a detailed tutorial is presented on how to use Google Colab to run Python scripts without the need for local installations, with a focus on generating and decoding composite QR codes (QRGB) from information encoded in three color layers (red, green and blue). The process of adapting a script originally designed for Visual Studio Code is explained, highlighting the advantages of cloud computing for democratising access to advanced tools. Through this approach, we seek to foster digital inclusion and allow more people, regardless of their technical level, to

explore new possibilities in the use of open source for creativity and innovation and to offer.
This application is offered as an open source project and is available for free so that designers, freelancers, makers and any user interested in marketing or advertising can use it and share it with their potential clients, thus expanding its reach and applicability in different creative and commercial fields.
For more information visit:https://federicoandersonar.wixsite.com/qrgb

Keywords:QR Codes, RGB Colors, Python, Google Colab, Open Source.

2.Introduction.

In today's digital age, programming has become a fundamental tool for solving complex problems, automating tasks, and developing innovative applications. However, not all users have the technical knowledge necessary to install and configure programming environments, which can limit their ability to take advantage of these tools. It is in this context that platforms such as Google Colab and programming languages such as Python become especially relevant, as they allow users without prior programming experience to run scripts and perform advanced tasks in a simple and accessible way.
Python, created by Guido van Rossum in the late 1980s, is a high-level programming language that has gained popularity for its clear syntax and versatility. It is widely used in fields such as data science, artificial intelligence, web development, and automation, thanks to its extensive collection of open-source libraries (such as Kivy, Pillow, OpenCV, and qrcode, among others) that make it easy to implement complex functionality with relatively few lines of code. However, running scripts in Python traditionally requires installing the language in a local environment, such as Visual Studio Code (VSCode), along with the necessary libraries, which can be intimidating for non-technical users.
Using code editors like Visual Studio Code, which allow you to install extensions (plugins) for languages like Python, enables advanced features, such as development with frameworks like Kivy, that are not natively possible on cloud platforms or environments like Google Colab, where applications run primarily in a browser without direct access to local graphical interfaces.
This is where Google Colab comes in, a cloud platform developed by Google that allows you to run Python code directly from the browser, without the need for prior installations. Launched in 2017, Colab is designed to facilitate collaboration and access to advanced computing resources, such as GPUs and TPUs, making it an ideal tool for researchers, educators and developers alike. In addition, its intuitive interface and integration with Google Drive make it especially attractive for users who are not familiar with programming, but who need to run scripts or perform data analysis.
In this research paper, we present the adaptation of a Python script that had originally been designed to be run in Visual Studio Code, which uses open source libraries such as Kivy (for the creation of advanced graphical interfaces), Pillow (for image manipulation) and OpenCV (for image processing). This script aims to generate and decode composite QR codes, known as QRGB, which combine three layers of information (red, green and blue) in a single image. However, recognizing that not all users have access to a local programming

environment, we have adapted this script so that it can be run in the cloud on Google Colab, thus eliminating the need to install Python or any other dependencies.
The decision to use Google Colab for this project is based on the need to make the technology accessible to a broader audience (especially traditional graphic designers – old-school pen and paper artists –, freelancers and diverse makers and/or budding AI coders), including all those with no prior programming experience. With Colab, users can simply open a link, upload the script and run it in a matter of seconds, without worrying about complex technical configurations. This not only democratises access to advanced programming tools, but also encourages collaboration and learning in communities – such as those of designers who are not professional computer software programmers – that might otherwise be excluded from these technological advances.
In this report, we will explore in detail how the script works in Google Colab, from QR code generation to decoding, and how users can leverage this platform to perform tasks that previously required advanced technical knowledge. In addition, we will provide a direct link to the script in Colab so that readers can try it out and experiment with it in real-time.[1]This paper also shows a tutorial for its use.
In short, this work not only demonstrates the power of Python and Google Colab to solve complex problems, but also underscores the importance of making the technology accessible to all designers and other marketing professionals (whether or not they are professional programmers), regardless of their level of programming experience. Through this research, we hope to inspire more people to explore the world of programming and take advantage of the tools available in the cloud to boost their creativity and productivity.

3.Python code development for Google Colab.

```
# Import libraries
importyou
importcv2
importqrcode
fromPILimportImage
fromgoogle.colabimportfiles
fromIPython.displayimportdisplay, HTML
fromtypingimportOptional, Tuple

# Base route configuration
BASE_PATH ="/content/qrgb_files"
os.makedirs(BASE_PATH, exist_ok=True)

# Function to create a QR with a logo in the center
def create_qr_with_logo(data:str,color:str,logo_path:Optional[str]
=None,qr_version:int=10,box_size:int=10) -> Image.Image:
    """Create a QR code with a logo in the center."""
    qr = qrcode.QRCode(
```

[1] We invite you to use this link to test the script in Google Colab:https://colab.research.google.com/drive/1-7b1Mug5Kj17Uv-og5IgCnzTFeoN7puu

```
        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')

    # The following block had incorrect indentation, which caused the
error.
    # Fixed by increasing the indentation of the block so that it is part
of the function body.
    iflogo_path:
        if notos.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)

    returnimg

# Function to combine three QR images into one
def
combine_qr_images(img1:Image.Image,img2:Image.Image,img3:Image.Image,logo
_path:Optional[str] =None) -> Image.Image:
    """Combine three QR images into one with RGB channels."""
    size = img1.size
    ifimg2.size != sizeorimg3.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 = []
    forYoin range(lion(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)

        ifred_pixelandgreen_pixelandblue_pixel:
            new_data.append((255,255,255,255))
        elifred_pixelandgreen_pixel:
            new_data.append((255,255,0,255))
        elifred_pixelandblue_pixel:
            new_data.append((255,0,255,255))
        elifgreen_pixelandblue_pixel:
            new_data.append((0,255,255,255))
        elifred_pixel:
            new_data.append((255,0,0,255))
        elifgreen_pixel:
            new_data.append((0,255,0,255))
        elifblue_pixel:
            new_data.append((0,0,255,255))
        else:
            new_data.append((0,0,0,255))

    final_image.putdata(new_data)

    iflogo_path:
        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)

    returnfinal_image

# Function to generate the QRGB
def
generate_qrgb(red_data:str,green_data:str,blue_data:str,logo_path:Optiona
l[str] =None,mode:str='text') -> Image.Image:
    """Generate a combined QRGB code from three data strings."""
    # Force a fixed QR version and module size
    qr_version =10  # Fixed version for all QR
    box_size =10    # Fixed module size for all QR

    # Generate individual QR codes
    img_red = create_qr_with_logo(red_data,"grid", 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)

    # Resize images to ensure they are the same size
    size = img_red.size
    img_green = img_green.resize(size, Image.LANCZOS)
    img_blue = img_blue.resize(size, Image.LANCZOS)

    # Save individual images
    img_red.save(os.path.join(BASE_PATH,"qr_red.png"))
    img_green.save(os.path.join(BASE_PATH,"qr_green.png"))
    img_blue.save(os.path.join(BASE_PATH,"qr_blue.png"))

    # Combine QR images
    combined_img = combine_qr_images(img_red, img_green, img_blue,
logo_path)
    combined_img.save(os.path.join(BASE_PATH,"superposed_qr.png"))

    returncombined_img

# Function to read a QR code
def read_qr(filename:str) -> Optional[str]:
    """Read QR code from an image file."""
    img = cv2.imread(filename)
    detector = cv2.QRCodeDetector()
    data, vertices_array, _ = detector.detectAndDecode(img)
    returndataifvertices_arrayes not None else None

# Function to manually decode a QRGB
def manual_decode_superposed_qr(filename:str) -> Tuple[Optional[str],
Optional[str], Optional[str]]:
    """Manually decode a superposed QRGB code into its components."""
    superposed_img = Image.open(filename)
    superposed_data = superposed_img.getdata()
    size = superposed_img.size

    red_data = [(255,255,255,255)] *lion(superposed_data)
    green_data = [(255,255,255,255)] *lion(superposed_data)
    blue_data = [(255,255,255,255)] *lion(superposed_data)

    forYoin range(lion(superposed_data)):
        r, g, b, a = superposed_data[i]
        ifr !=0:# Grid
            red_data[i] = (0,0,0,255)
        ifg !=0:# Green
            green_data[i] = (0,0,0,255)
        ifb !=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(os.path.join(BASE_PATH,"decoded_red.png"))
    green_img.save(os.path.join(BASE_PATH,"decoded_green.png"))
    blue_img.save(os.path.join(BASE_PATH,"decoded_blue.png"))

    data_red = read_qr(os.path.join(BASE_PATH,"decoded_red.png"))
    data_green = read_qr(os.path.join(BASE_PATH,"decoded_green.png"))
    data_blue = read_qr(os.path.join(BASE_PATH,"decoded_blue.png"))

    returndata_red, data_green, data_blue

# Function to decode a QRGB
def decode_qrgb():
    """Decode a generated or uploaded QRGB."""
    print("\nQRGB Decoding")

    # Upload QRGB file
    print("Upload QRGB (PNG) file:")
    uploaded = files.upload()
    if notuploaded:
        print("No files uploaded. Returning to main menu.")
        return

    qrgb_path =list(uploaded.keys())[0]

    # Show a small version of the decoded QRGB
    print("\nPreview of decoded QRGB (small):")
    qrgb_img = Image.open(qrgb_path)
    small_qrgb = qrgb_img.resize((150,150), Image.LANCZOS)  # Resize to
150x150
    display(small_qrgb)

    # Decode the QRGB
    print("\nDecoding QRGB...")
    data_red, data_green, data_blue =
manual_decode_superposed_qr(qrgb_path)

    print("\nDecoding results:")
    print(F"Red Cape:{data_red}")
    print(F"Green Cape:{data_green}")
    print(F"Blue Cape:{data_blue}")
```

```
# Main function to generate QRGB in Colab
def main():
    while True:
        print("\n--- QRGB Generator in Google Colab ---")
        print("1. QRGB Encode")
        print("2. Decode QRGB")
        print("3. Exit")

        option =input("Select an option (1/2/3):")

        ifoption =="1":
            while True:
                print("\nQRGB Encoding")

                # Request input data
                red_data =input("Enter text or link for the red layer: ")
                green_data =input("Enter text or link for the green
layer: ")
                blue_data =input("Enter text or link for the blue layer:
")

                # Ask if you want to add a logo
                use_logo =input("Do you want to add a logo? (y/n):
").lower()
                logo_path =None
                ifuse_logo =='s':
                    print("Upload logo file (PNG or JPG):")
                    uploaded = files.upload()
                    if notuploaded:
                        print("No file uploaded. Continuing without
logo.")
                    else:
                        logo_path =list(uploaded.keys())[0]

                # Generate QRGB
                mode ='link' if any('http'
intext.lower()fortextin[red_data, green_data, blue_data])else 'text'
                combined_img = generate_qrgb(red_data, green_data,
blue_data, logo_path, mode)

                # Show the generated image
                print("\nQRGB generated successfully:")
                display(combined_img)

                # Save the image

combined_img.save(os.path.join(BASE_PATH,"superposed_qr.png"))
                print(F"QRGB saved
in:{os.path.join(BASE_PATH,'superposed_qr.png')}")
```

```
                # Download the image automatically

files.download(os.path.join(BASE_PATH,"superposed_qr.png"))
                print("The QRGB has been automatically downloaded to your
device.")

                # Ask if you want to decode or re-encode
                while True:
                    print("\nWhat do you want to do now?")
                    print("1. Decode the generated QRGB")
                    print("2. Re-encode")
                    print("3. Return to main menu")

                    sub_option =input("Select an option (1/2/3):")

                    ifsub_option =="1":
                        decode_qrgb()
                    elifsub_option =="2":
                        break  # Re-encode
                    elifsub_option =="3":
                        break  # Back to main menu
                    else:
                        print("Invalid option. Please try again.")

                ifsub_option =="3":
                    break  # Exit the coding loop

        elifoption =="2":
            decode_qrgb()

        elifoption =="3":
            print("Leaving the program...")
            break

        else:
            print("Invalid option. Please try again.")

# Run the main function
if __yam__=='__main__':
    main()
```

4.Discussion of Python script for Google Colab.

This Python script is designed to run on Google Colab and aims to generate and decode composite QR codes, called QRGB, that combine three layers of information (red, green, and blue) into a single image. Below, we will analyze the script part by part, explaining its functionality and structure.

4.1. Importing libraries

*you:*It is used to interact with the operating system, such as creating directories and handling file paths.
*cv2 (OpenCV):*It is used for detecting and decoding QR codes.
*qrcode:*Library for generating QR codes.
*PIL (Pillow):*For image manipulation, such as opening, resizing, and combining images.
*google.colab.files:*Allows uploading and downloading files in Google Colab.
IPython.display: To display images and HTML in the Colab environment.
*typing:*To define data types in functions, which improves code clarity and maintainability.

4.2. Setting the base route
*BASE_PATH:*Defines the path where the generated files will be saved.
*os.makedirs:*Create the directory if it does not exist, with exist_ok=True to avoid errors if the directory already exists.

4.3. create_qr_with_logo function
*Purpose:*Generate a QR code with a logo in the center.
Parameters:
-*data:*Information to be encoded in the QR.
-*color:*QR color.
-*logo_path:*Logo path to be superimposed in the center of the QR.
-*qr_version:*QR version (control size).
-*box_size:*Size of each QR module.
Process:
-Creates a QRCode object with the specified parameters.
-Add the data to the QR and generate it.
-If a logo is provided, it resizes and overlays it in the center of the QR.

4.4. combine_qr_images function
*Purpose:*Combine three QR images into one, assigning each image to a color channel (red, green, blue).
Parameters:
-*img1, img2, img3:*QR images to be combined.
-*logo_path:*Logo path to be overlaid in the center of the combined image.
Process:
-Check that all images are the same size.
-Create a new image by combining the pixels of the three images according to their color.
-If a logo is provided, it overlays it in the center.

4.5. generate_qrgb function
*Purpose:*Generates a QRGB from three data strings (red, green, blue).
Parameters:
-*red_data, green_data, blue_data:*Data to be encoded in each color layer.
-*logo_path:*Logo path to be overlaid in the center.
-*mode:*Encoding mode ('text' or 'link').
Process:
-Generates three QR codes (one for each color layer).
-Resize images to ensure they are the same size.

- Combine the images using combine_qr_images.
-Save and display the resulting image.

4.6. read_qr function
*Purpose:*Read and decode a QR code from an image file.
Parameters:
*-filename:*Path of the image file containing the QR.
Process:
-Use OpenCV to detect and decode the QR.
-Returns the decoded data.

4.7. manual_decode_superposed_qr function
*Purpose:*Manually decode a QRGB into its three components (red, green, blue).
Parameters:
-filename: Path of the image file containing the QRGB.
Process:
-Separates the image pixels into three color layers.
-Save each layer as a separate image.
-Use read_qr to decode each layer.

4.8. decode_qrgb function
*Purpose:*Allows the user to upload a QRGB and decode it.
Process:
-Requests the user to upload a QRGB file.
-Shows a preview of the image.
-Decode the QRGB using manual_decode_superposed_qr and display the results.

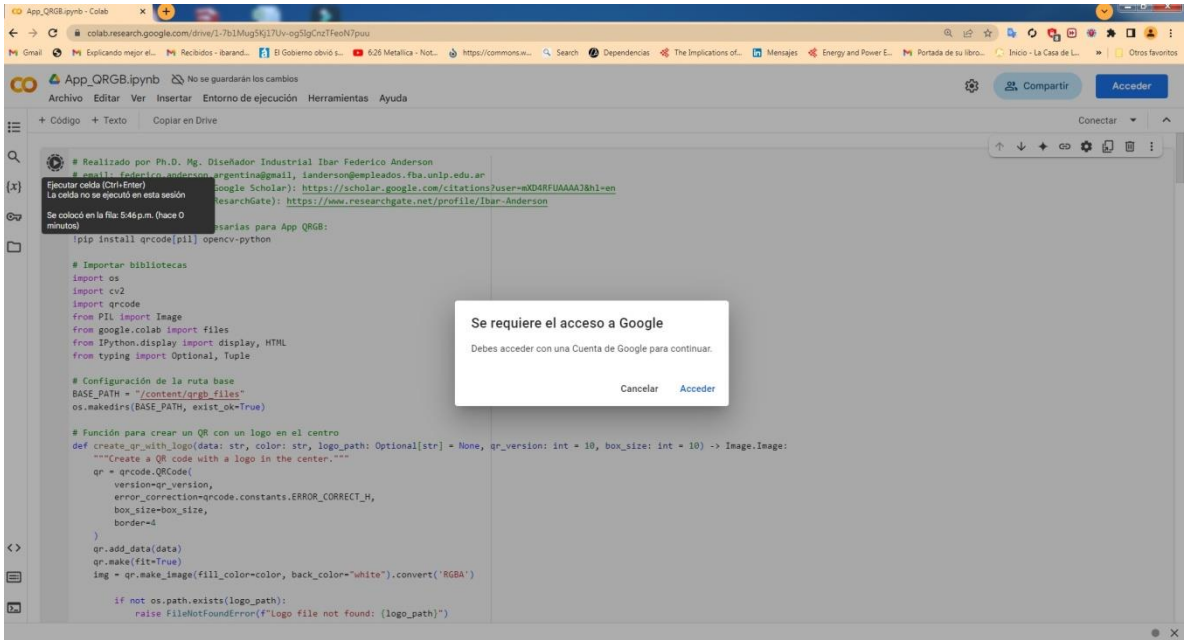
4.9. Main function
*Purpose:*Main function that handles the flow of the program.
Process:
-Displays a menu for the user to choose between encoding or decoding a QRGB.
-Depending on the selected option, calls the corresponding functions (generate_qrgb or decode_qrgb).
-Allows the user to re-encode, decode or exit the program.

4.10. Program execution
*Purpose:*Executes the main function when the script is run directly.
Additional considerations:
*Error handling:*The script could be improved with more robust error handling, especially on file uploads and QR decoding.
*Optimization:*Some operations, such as image resizing, could be optimized to improve performance.
*User Interface:*The script is interactive and designed to be used in Google Colab, making it easy to use without any additional configuration.
In summary, this script is a powerful tool for generating and decoding composite QR codes, with a user-friendly interface and advanced features such as overlaying logos and combining multiple layers of information. It is a clear example of how Python can be used

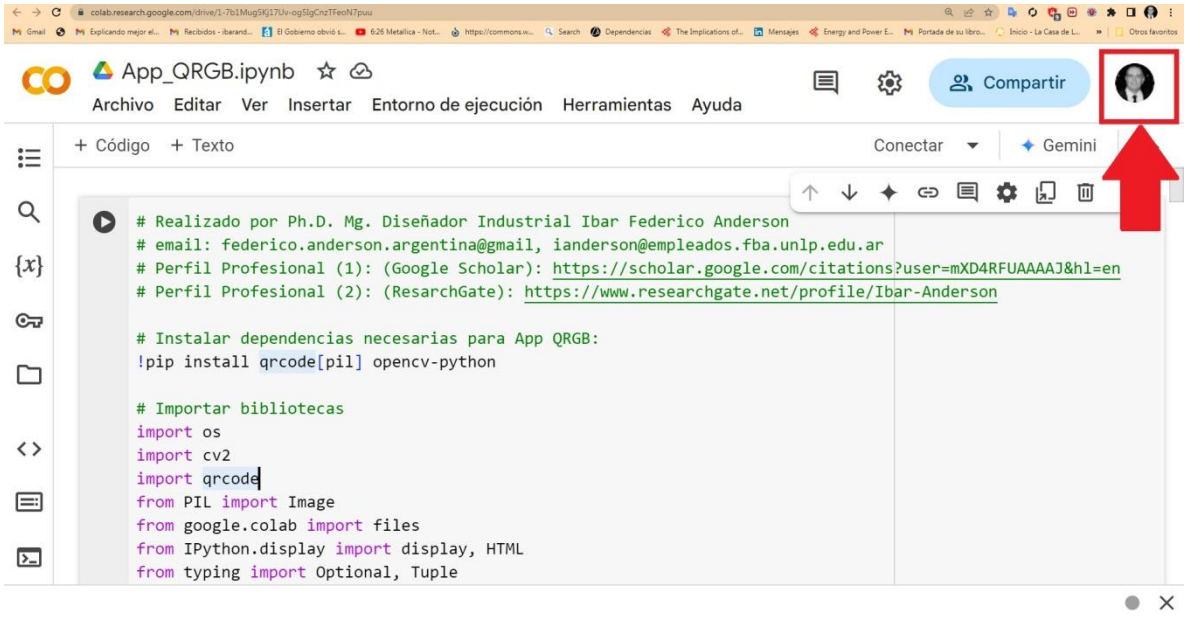
for image processing and automation tasks in a collaborative environment such as Google Colab.

5.Practical development of the application, illustrated with an example.

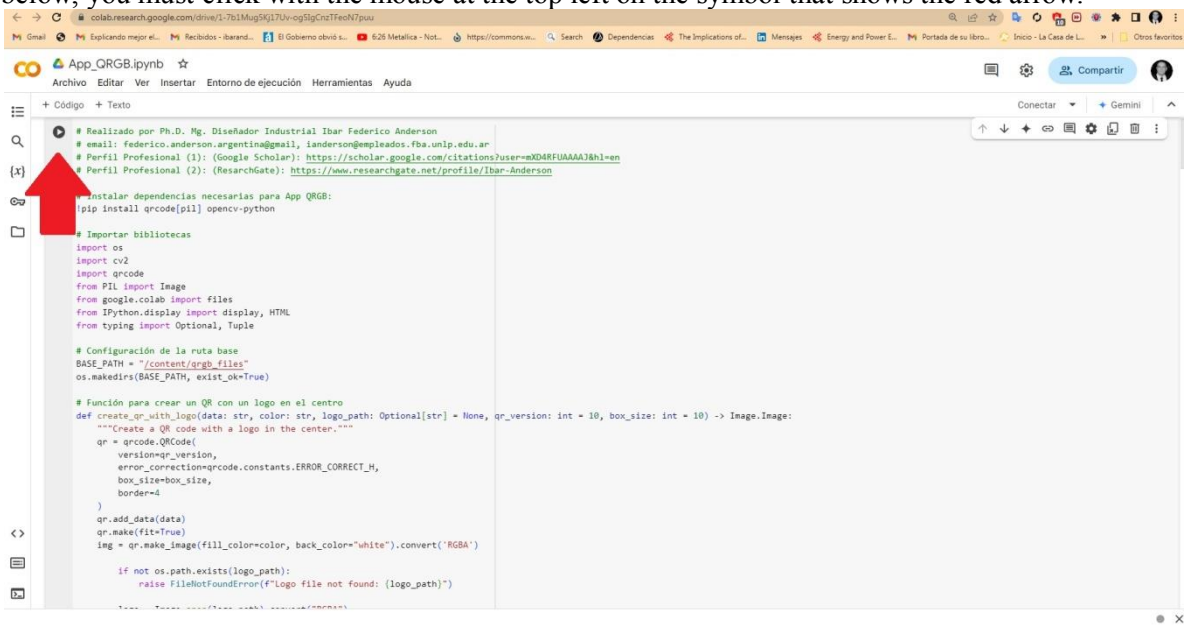
First, you must keep your email open, because if you don't, the following message will appear: "Google access required" (for which you must log in with your Gmail account, at the top right where it says "Login").

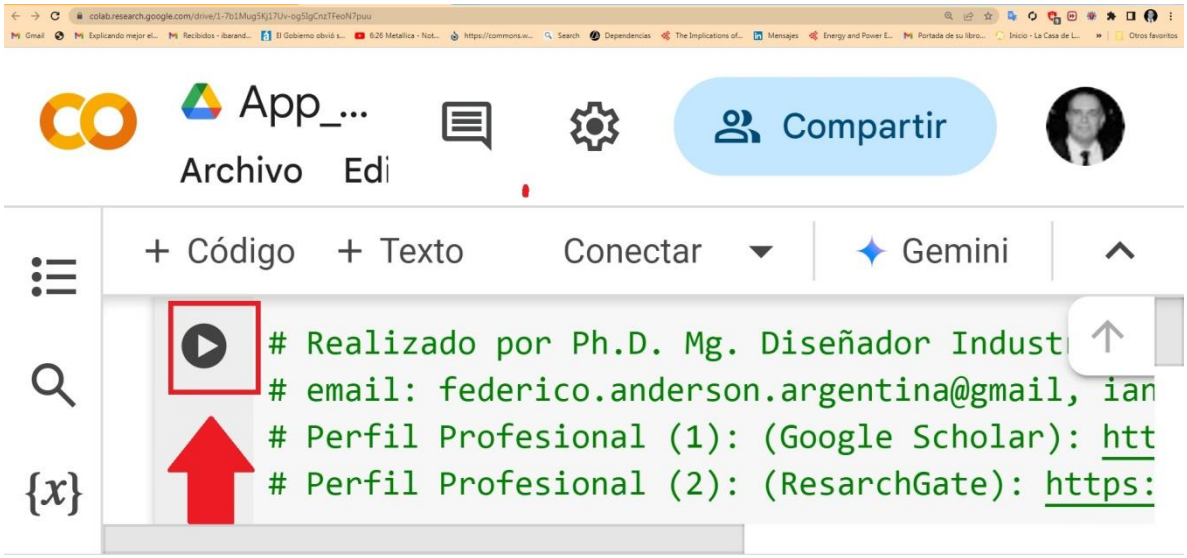


After you log into your Gmail account and re-enter the Google Colab link, you can see that you have logged in correctly because the logo or image that identifies your email account will appear at the top right of your monitor (in my case, the face of my Gmail account appears as shown below).

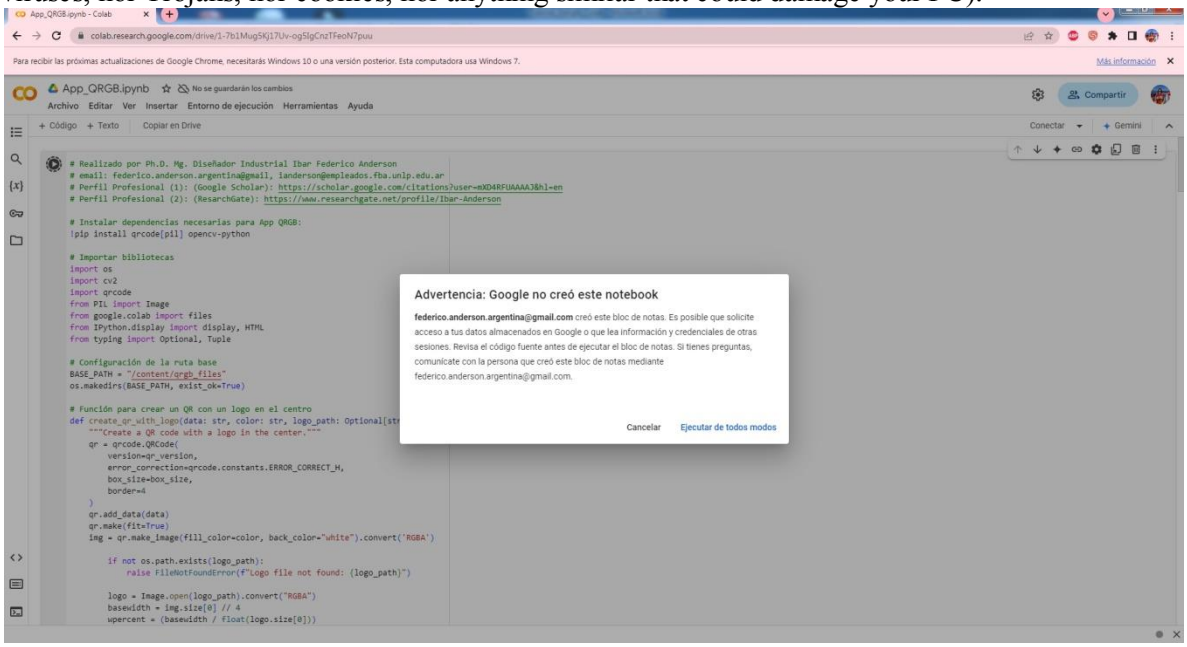


Once inside Google Colab and logged in, on your monitor screen, like the one shown in the image below, you must click with the mouse at the top left on the symbol that shows the red arrow.



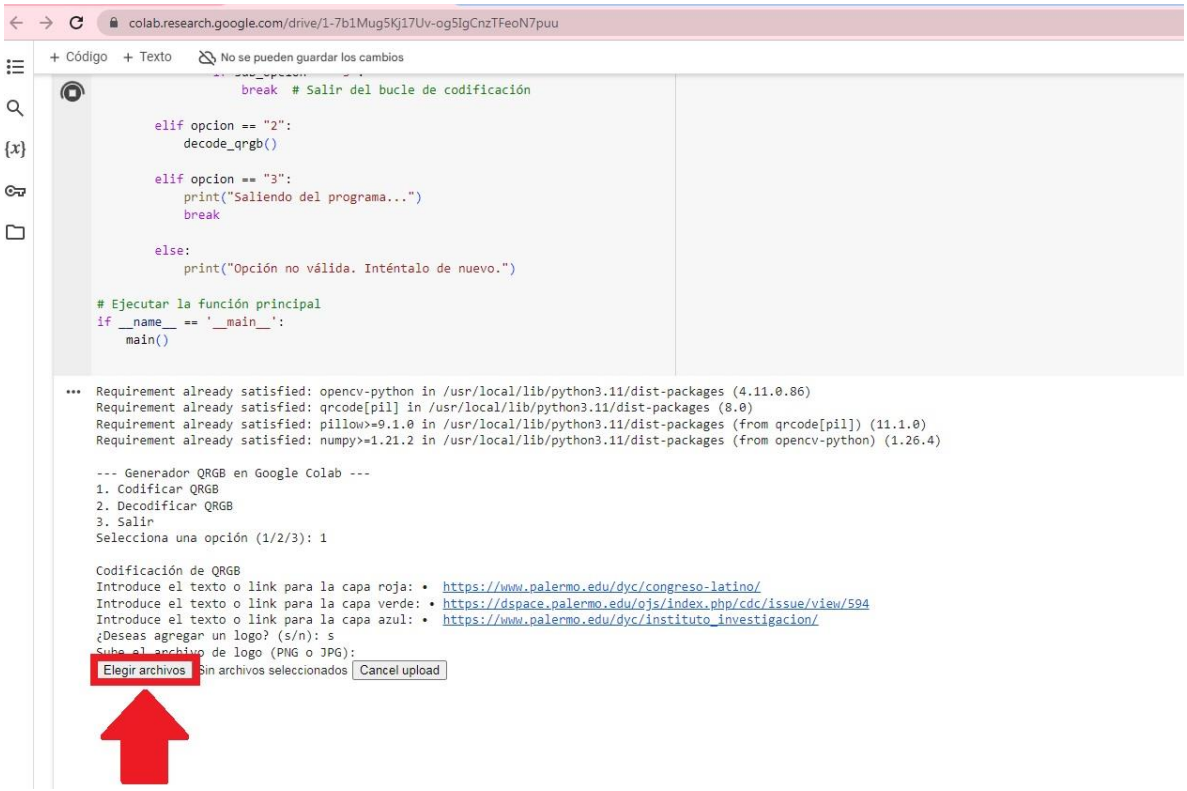


A message will appear like the one shown below in the following image (and you must click with the mouse where it says “Run anyway”, do it without fear, the script is safe, it does not have viruses, nor Trojans, nor cookies, nor anything similar that could damage your PC).

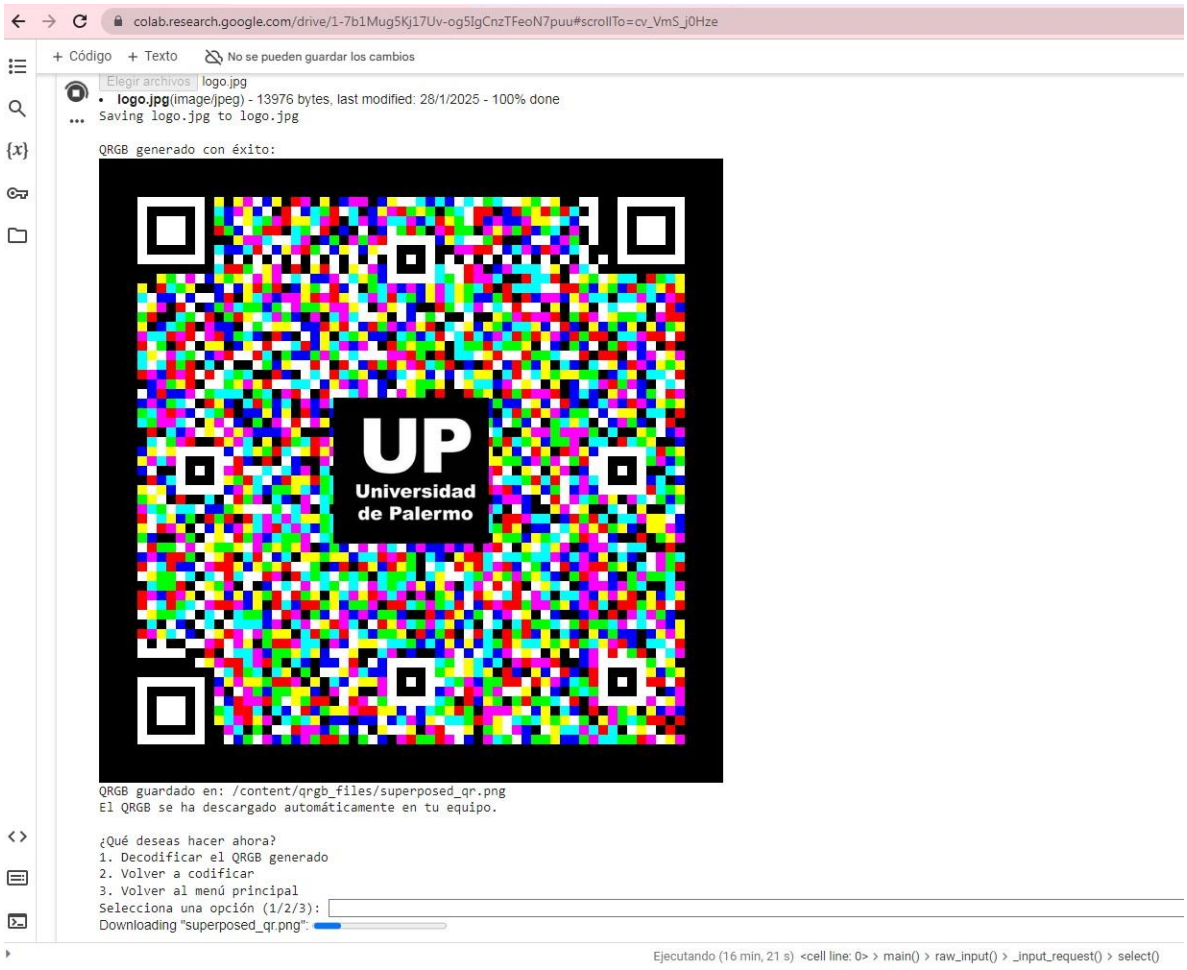


When you run it you will notice that it is working by the square symbol (top left where the red arrow is indicating it) and you will have to scroll the page (go down with the mouse, towards the end of the script or Python program) until you see the following:

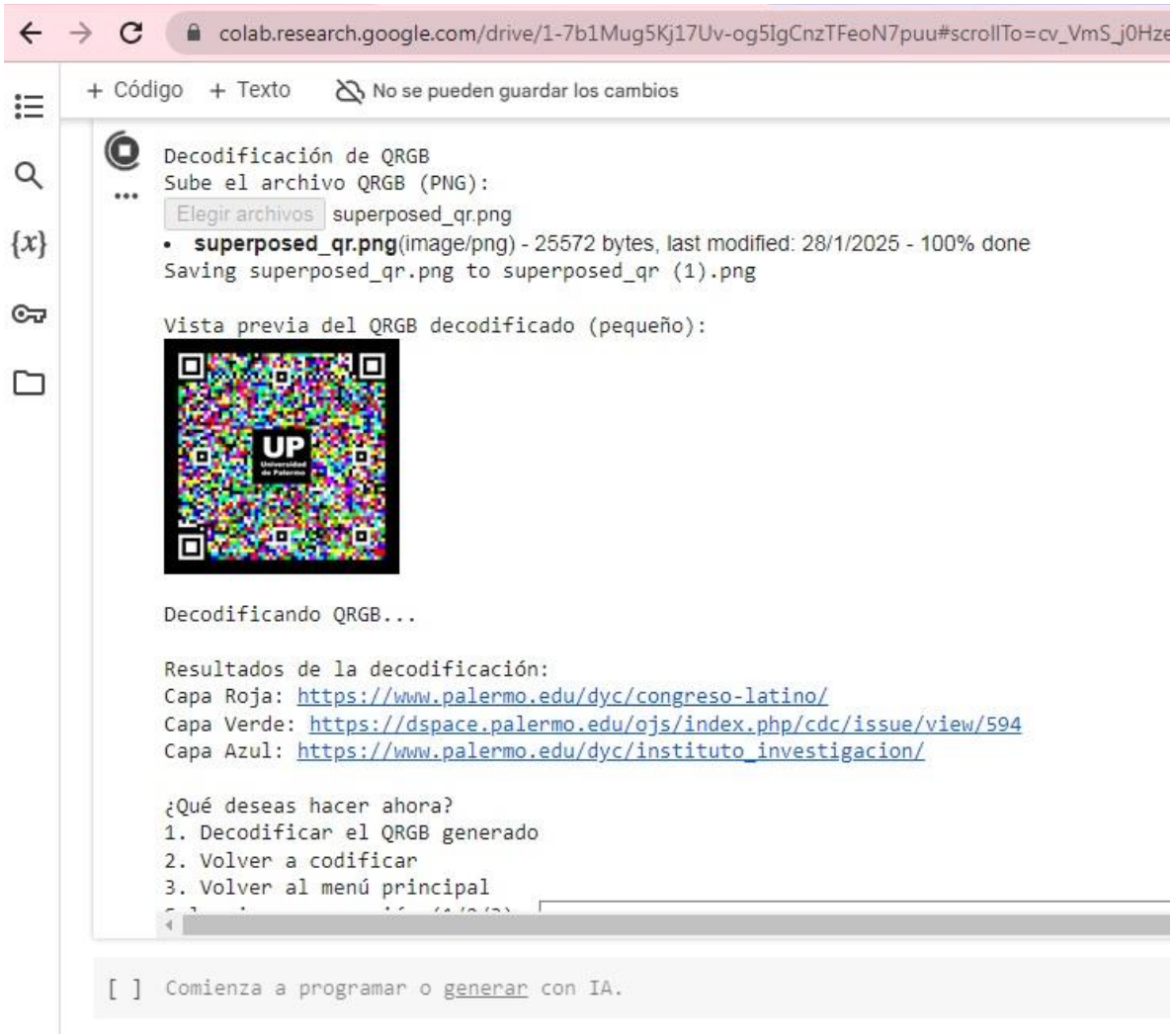
It will ask you if you want to: (1) Encode QRGB, (2) Decode QRGB or (3) Exit. In my case I will select the option (1) Encode QRGB and enter it with Enter. Immediately three data entry sites (inputs) open for each of the three (3) channels (RGB readers) for entering information (there you can enter the data you want to encode). In my case I will enter the three links of the University of Palermo and they are the following for each of the layers (red, green and blue):
https://www.palermo.edu/dyc/congreso-latino/
https://dspace.palermo.edu/ojs/index.php/cdc/issue/view/594
https://www.palermo.edu/dyc/instituto_investigacion/
When it asks me if I want to enter a logo (or image) I will select “yes” (but you can select the “no” option), in my case I will select the Logo of the University of Palermo.



Once the logo (or image) is uploaded, the QRGB Code is generated and can be downloaded or saved in a folder on your computer, as an image file in (.png) format. This is how it will look on your PC monitor:



Next, it will ask you again what you want to do, in my case how am I going to Decode the QRGB Code, previously generated or created, for which I will select the option (1) Decode the generated QRGB and press Enter (again the program will ask me to choose the code, for which I will select the QRGB Code saved previously). Then the monitor of my PC will show me a thumbnail with all the encoded information (which in my case were the links to the web pages of the University of Palermo):



Thank you for reading this tutorial, I hope you find it useful. If you have any questions, please email me.

6.Conclusions.

The analyzed script is a robust and well-structured example of how Python can be used for advanced QR code generation and decoding tasks, specifically designed to run on Google Colab. Its main purpose is to generate and decode composite QR codes (QRGB), which combine three layers of information (red, green, and blue) into a single image. Each function in the script is designed to perform a specific task, making the code easy to understand, maintain, and scale. Furthermore, popular and powerful libraries such as qrcode, PIL, OpenCV are leveraged, demonstrating a good understanding of the tools available in the Python ecosystem. These libraries allow complex tasks such as QR generation, image manipulation, and code decoding to be performed efficiently.

The script is designed to be used in Google Colab, making it accessible to users without the need for additional configuration. The main() function provides an interactive menu that guides the user through the encoding and decoding process, thus improving the user experience. Additionally, the script allows customizing the generated QR codes, such as adding a logo in the center or combining multiple layers of information, and supports both text and links as input data, making it versatile for different use cases. In terms of good programming practices, the code follows clear principles, such

as using static typing to improve clarity and reduce errors, and the functions are well documented with docstrings, making them easy to understand and use. Special cases are also handled, such as checking for file existence and validating image sizes.

However, there are opportunities for improvement that could further elevate the quality of the script. For example, while some errors are handled, such as file verification, more robust exception handling could be implemented, especially in file uploads and QR decoding. Also, some operations, such as image resizing and pixel blending, could be optimized to improve performance, especially when working with large images. Regarding the graphical interface, although the script is designed for Colab and uses basic interaction tools, it could be significantly improved if implemented in an environment such as Visual Studio Code with frameworks such as Kivy or Tkinter. These frameworks would allow for a more advanced and customizable graphical interface, which would improve the user experience, especially for those who are not familiar with console-based programming environments.

As for potential applications, this script can be useful in a variety of scenarios. For example, in marketing, it could be used to create custom QR codes with logos for advertising campaigns; in education, it could be used to teach concepts of image processing and data encoding; and in automation, it could be integrated into larger workflows to programmatically generate and decode QRs. The ability to combine multiple layers of information into a single QR code opens up interesting possibilities, such as encoding additional data or creating interactive QR codes.

In conclusion, this script is a solid example of how Python can be used to solve complex problems in an efficient and accessible way. It combines advanced functionality, such as QR Code generation, image manipulation, and decoding, with an interface (which is not as user-friendly as the original Python script for VS Code)[2], making it suitable for both technical and non-technical users.

When comparing Google Colab to Visual Studio Code (VS Code), one can identify key differences and limitations, especially in the context of developing applications with frameworks such as Kivy. For example, Google Colab is primarily designed to run Python code in the cloud and display results in the browser; however, it does not have direct access to local graphical interfaces (GUIs), as it lacks a physical screen to render windows or interactive elements. This means that frameworks such as Kivy, which are designed to create applications with advanced graphical interfaces, cannot fully run on Colab because there is no way to view or interact with the windows generated by Kivy. In contrast, VS Code allows you to develop and test GUI applications locally, as it has access to your operating system and can open graphical windows in real time; furthermore, with suitable extensions, you can use VS Code to develop full applications with Kivy, including creating, debugging, and viewing graphical interfaces.

Another important difference lies in the installation of plugins and advanced tools. Google Colab is a preconfigured cloud environment with popular libraries already installed, such as numpy, pandas, and matplotlib. While you can install additional libraries using commands like !pip install or !apt-get install, you can't add plugins or extensions to the environment like you would in VS Code. Additionally, it lacks native support for advanced tools like visual debuggers, graphical variable explorers, or integration with GUI frameworks like Kivy. On the other hand, VS Code offers a wide range of extensions that enhance the development experience, such as support for frameworks like Kivy, PyQt, or Tkinter, advanced debugging tools, and integration with Git, Docker, databases, and cloud services, making it a more versatile tool for complex projects.

Access to local resources is also a key factor. Google Colab does not have direct access to local files or physical devices, such as cameras or microphones, without manually uploading files or using specific APIs, which limits its use for applications that require constant interaction with local resources. In contrast, VS Code has full access to your local system, allowing you to work with

[2]To see the full Python script for VS Code (with user-friendly GUI frameworks like Kivy) see this link:https://osf.io/g3ame_v1

files, devices, and custom configurations without restrictions. This makes VS Code more flexible for developing applications that rely on local resources.

Despite these limitations, Google Colab has specific advantages. It is accessible, as it requires no local configuration or software installation, making it ideal for non-technical users. In addition, it runs in the cloud, allowing you to work from any device with internet access. It also offers free access to GPUs and TPUs, which is useful for intensive tasks such as machine learning or image processing. Finally, it facilitates real-time collaboration, something that VS Code does not offer natively.

In conclusion, Google Colab is limited compared to VS Code for certain advanced tasks; however, these limitations are not inherent to Colab as a platform, but rather to the fact that it is designed to run in a browser and lacks access to local resources.

7.Literature.

Anderson, IF (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." Preprints.org, pp. 1-47.https://doi.org/10.20944/preprints202407.1384.v2

Anderson, IF (2024). "QRGB: App for QR code generation (method: 3 in 1), or additive color generation method (RGB), applying open source Python libraries, to increase the accumulated information density". EdArXiv Preprints, pp. 1–29.https://doi.org/10.35542/osf.io/hy2em

Anderson, IF (2024). "QRGB+: Advanced QR Code Generator with RGB Color Method in Python to Expand Data Capacity." Journal of Sensor Networks and Data Communications, Vol. 4, No. 2, pp. 1-20. Handle:
http://sedici.unlp.edu.ar/handle/10915/169498

Anderson, IF (2024). "QRGB++ in Python running in Visual Studio Code with a graphical in-terface (pip install kivy) + pip install pillow + pip install qrcode[pil] + pip install opencv-python". OSF Preprints, pp. 1-17.https://doi.org/10.31219/osf.io/g3ame