ABSTRACT

This report provides an analysis of 12 works in which artists developed software-based art pieces for an event held at the Whitney Museum in 2002. Due to the time elapsed, many of the programming languages used have become obsolete or have undergone significant updates, leading to incompatibility issues with modules and libraries. This work aims to reproduce these artworks using current technologies, employing 3 different strategies:

1. **Execution in a modern environment**: For languages still in use, scripts were run on personal computers with updated versions.
2. **Transpilation**: Some scripts were translated into modern languages with equivalent features to preserve the artistic intent of the work.
3. **Use of virtual machines (VM)**: In cases where the languages are no longer in use, virtual machines were set up to faithfully replicate the original environment.

It is important to note that while cloud-based environments were not considered for code reproduction, this option is viewed as a potential strategy.

INTRODUCTION

In 2002, the Whitney Museum of American Art presented an exhibition of software-based artworks, highlighting how artists use code to create interactive and innovative pieces. These works, known as software-based artworks, use software as a creative tool, merging programming, interactivity, and aesthetics in new forms of artistic expression.

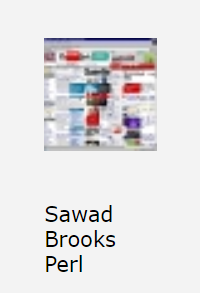
However, the preservation of these pieces presents a significant challenge, as over time, technologies, languages, libraries, and execution environments are constantly evolving and, in some cases, become obsolete, threatening both the preservation and the ability to experience the works in their original form.

This report analyzes the 12 pieces presented at the event, evaluating a set of strategies for each specific case, including: execution in the original language, transpilation to modern languages, use of virtual machines (VM) emulating original platforms, and packaging in isolated containers.

The report documents the entire research and execution process. The content is organized into sections that include: a development section for each artwork, detailing the process followed; a summary of the results obtained; conclusions derived from the analysis; and the references consulted.

DEVELOPMENT

1. ****Sawad Brooks****

 Picture taken from [2]

**Analysing the code:**

The artwork by Sawad Brooks is implemented in Perl 5, a programming language that was very popular during the 1990s and early 2000s. Perl was known for its ease in processing text and performing network-related tasks. Although it is still used in specialized fields such as bioinformatics and system administration, its popularity has declined due to the rise of more modern and versatile languages like Python.

The main goal of the script is to generate an HTML page (global.html) that simulates the front page of a newspaper by extracting content from three websites: ["[www.nytimes.com](http://www.nytimes.com" \t "_new)", "[www.theguardian.com](http://www.theguardian.com" \t "_new)", "[www.asahi.com](http://www.asahi.com" \t "_new)"].  
The code uses the GetHTTP function to connect to each URL and retrieve the full HTTP response. However, it lacks advanced content selection logic; it simply reads the complete HTML of the web pages, including text and HTML elements, saves them to a file, and then displays them. The original code is stored in the file globalcityOriginalCode.pl.

**Strategies: Preparing the working environment:**

**DEBUGGING**

To run the code, I first installed Strawberry Perl on my Windows 11 operating system. Strawberry Perl is a distribution of Perl for Windows that includes the necessary tools and libraries to run Perl scripts easily. I then proceeded to fix some issues in the code due to incompatibilities with newer versions of Perl.

To execute the file in the Strawberry Perl console, I used the following command:  
perl C:\path\to\your\file\script.pl

In all three Perl code cases, it was necessary to modify the original URLs due to access policy restrictions that blocked the websites, preventing the extraction of information. This could be due to changes in the security rules of the web pages since 2002 or how HTTP methods are implemented in Perl. After modifying the URLs, the collage was generated correctly, but the result varied each time the script was run and did not exactly match with the original artwork image.  
To resolve this issue, I made the following changes to the code:

**CASE 1:**

Script Name: globalcityCase1.pl  
Result: globalCase1.html

In this case, I kept the idea of using sockets for the connections but made the following adjustments in the code:

* **URL change**: I modified the complete URLs in the @space list to only include domain names, as the inet\_aton function does not work with full URLs.
* **Domain resolution check**: I added a check to ensure that inet\_aton could resolve the domain:

my $paddr = sockaddr\_in(80, inet\_aton($hostname)) or die "inet\_aton(): Could not resolve $hostname\n";

* **HTTP headers**: I updated the HTTP headers to ensure compatibility with modern servers and added the line Connection: close:

print $sock "GET $doc HTTP/1.0\r\nHost: $hostname\r\nConnection: close\r\n\r\n";

* **Variable handling**: I added my to the local variables to avoid conflicts and improve code clarity.

**CASE 2:**

Script Name: globalcityCase2.pl  
Result: globalCase2.html

In this case, I switched from using sockets to the *LWP::Simple* module to handle HTTP requests in Perl. To do this, I first installed the module in the Strawberry Perl console with the following command:  
cpan LWP::Simple

After running the script, the result was saved in the file globalCase2.html. I used the following test URLs:

* [https://www.example.com](https://www.example.com" \t "_new)
* [https://www.wikipedia.org](https://www.wikipedia.org" \t "_new)
* [https://www.perl.org](https://www.perl.org" \t "_new)  
  I did not use the original URLs from the code because they could not connect properly.

**CASE 3:**

Script Name: globalcityCase3.pl  
Result: globalCase3.html

For this case, I replaced the use of Socket with *LWP::UserAgent*, a more modern and robust library for handling HTTPS requests in Perl.

**TRANSPILATION**

**To Python:** For the migration of the original code, I chose Python as the new language due to its wide popularity and ease of handling HTTPS requests. In this process, I used the requests library to handle HTTP requests and BeautifulSoup (via the bs4 library) to parse and clean the HTML content. These libraries allowed me to omit unnecessary headers and properly handle redirects.  
One of the advantages of Python is its readability, as it is a high-level language. This made it easier to iterate through the URLs, handle errors, and manipulate the HTML content.

To install the necessary libraries, I used the following command:  
*pip install beautifulsoup4*

Script Name: *globalcity.py*  
Result: *globalPython.html*

In this case, I used the original code's websites, and the result was quite similar to the one obtained originally.

**Use of Virtual Machines (VM):**

To emulate the original environment in which the code was executed, I downloaded and installed Oracle VirtualBox and an ISO image of Windows XP, which would serve as the operating system in the virtual machine. This approach would allow for a more faithful replication of the original execution environment.  
Status: Pending testing.  
If the virtual machine solution is not viable, I will consider other alternatives, such as setting up a VM that emulates an appropriate environment to run the original code.

1. **Mary Flanagan**

 Picture taken from [2]

**Analysing the code:**

**The original code** is written in Lingo, a scripting language designed for Macromedia Director, primarily used for creating interactive multimedia content such as games and animations. However, Macromedia Director stopped receiving support in 2017 after being acquired by Adobe, and more modern technologies like HTML5, JavaScript, and game engines such as Unity have replaced its functionality.

The original project, titled [remotion], was developed by Mary Flanagan using Macromedia Director. This project utilizes the TrackThemColors extension to track and process images from a video captured by a webcam.

The goal of the code is to capture video from the webcam, detect frame-to-frame changes (specifically in colored areas called blobs), and move certain sprites on the screen based on the detected changes in the video areas.

The file 100kstream.mov is the video generated by the original code, downloaded from [1].

**Strategies: Preparing the working environment:**

#### ****1. Transpilation to Python:****

Python was chosen to migrate the original code due to its widespread use in video processing and computer vision projects, leveraging libraries like OpenCV, which allow access to capture devices (such as webcams) and video file reading.  
To install the required libraries, I used the following commands:

***pip install opencv-python***

***pip install --upgrade opencv-python opencv-python-headless***

***pip install notebook***

In the original code, the author worked with a video camera to apply points and rectangles over the detected blobs. However, due to issues with the laptop camera, I decided to work with a music video file to test the code. The result is shown in video.mp4.

**Laptop Camera Issues:**  
When attempting to use the local camera, I encountered several errors, likely due to incompatibility between the camera and the OpenCV backend. Some of the issues included:

- Invalid resolutions (0x0).

- FPS detected as 0.0.

- Incorrect backend configuration for recording.

To open the local camera, I used:

***cap = cv2.VideoCapture(0, cv2.CAP\_DSHOW)***

This command forces the use of the CAP\_DSHOW backend to resolve issues when opening the camera.

**Using a Local Video File:**  
The only successful scenario occurred when using a locally stored video instead of the camera. The code used to open the video was:

***cap = cv2.VideoCapture('path/to/video.mp4')***

This enabled motion detection in the video and saved the result to the file outputVideo.mp4. Pressing the ‘q’ key stops the execution.

**Script Name:** *code.py*

**Output:** *outputVideo.mp4*

In this case, blobs are areas of change detected in the video, calculated from the difference between the reference frame and the current frame. Blobs are found using contours in the binarized image and are drawn as rectangles and circles on the video.

**Sprites in Python:**  
While the original code handled sprites more explicitly, in the Python version, drawing rectangles and circles over the blobs can be considered a visual equivalent of moving the sprites. These "sprites" represent the largest detected areas of change and their positions in the video.

In the video, red dots and green rectangles indicate the detected areas of change:

- **Red Dots:** Represent the center of each region where a significant change in the video pixels was detected.

- **Green Rectangles:** Enclose the areas identified as regions of change, showing the extent and location of the changes.

#### ****2. Transpilation to JavaScript (In Progress):****

As a modern and viable option, JavaScript combined with HTML5 and the WebRTC API is an excellent choice to replicate the functionality of the original code. This combination allows video capture directly from the browser and motion processing using more up-to-date web technologies.

**Tasks to Complete:** Finalize the JavaScript implementation and conduct tests to verify correct execution in a modern environment.

**Use of Virtual Machines (VM):**

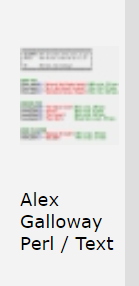
To emulate the environment in which the original code was executed, a virtual machine (VM) with Windows XP was considered. This setup allows Macromedia Director and its dependencies to run. Oracle VirtualBox and a Windows XP ISO image were downloaded for this purpose.

Once the system is configured, Macromedia Director (compatible with that version of Windows) and the TrackThemColors extension need to be installed. The extension, along with the demo files, should be placed in the same folder as the application. To ensure proper functionality, the camera must have drivers compatible with the virtual environment.

**Status:** Pending testing.

**Steps to Follow:**

1. Install Oracle VM VirtualBox.
2. Download Windows XP ISO.
3. Install VirtualBox Guest Additions.
4. Download and install Macromedia Director.
5. Download and install the TrackThemColors extension.
6. **Alexander R. Galloway**

 Picture taken from [2]

**Analysing the code:**

The original code is written in Perl and focuses on abstract concepts related to cybersecurity and digital vulnerabilities.  
It defines three arrays (*$first\_point, $second\_point, $third\_point*) containing titles of hacking tools or exploits, but it does not perform any additional actions. The primary purpose of the code seems to be conceptual. Its intention aligns more with illustrating the relationship between users, interfaces, and potential computer vulnerabilities.

Rather than being a functional tool, the code represents a metaphor for the power and risks associated with executing online scripts without understanding their implications.

**Strategies: Preparing the working environment:**

**DEBUGGING**

Using Strawberry Perl, the script was executed and did not perform any operation beyond defining the arrays.

**TRANSPILATION**

To modernize the code, it was migrated to Python using Flask. An application was designed to display the original lists as interactive links. Clicking a link redirects to another page that displays related content.

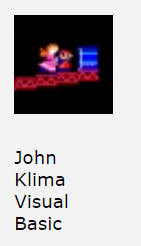
**Steps to Execute:**

1. Navigate to the app folder within the project.
2. Run the file app.py.
3. Access the URL: [http://127.0.0.1:5000](http://127.0.0.1:5000" \t "_new) in your browser.

**Use of Virtual Machines (VM):**

If it becomes necessary to execute the code in an environment closer to the original, the use of a VM with a Windows XP operating system is considered.

1. **John Klima**

 Picture taken from [2]

**Analysing the code:**

This code was written in Visual Basic 6.0, a popular visual development tool from the 1990s and early 2000s that allowed for the creation of desktop applications with graphical interfaces. Although Visual Basic 6.0 is now obsolete and no longer officially supported by Microsoft, its successor, Visual Basic .NET, remains in use within the ecosystem of modern .NET-based applications.

The code simulates an interactive animation where the characters Jack and Jill move across an inclined terrain, with their actions and behaviors influenced by an emotional state. As they interact with the hill and the environment, their movements and positions are controlled through graphical functions that manipulate pixels on the screen.

Additionally, the code logic allows for changing the characters' motivations and restarting the game. From a technical perspective, the code uses timing functions to synchronize the animation and a buffering system to improve performance by preventing flickering.

**Strategies: Preparing the working environment:**

**DEBUGGING**

To execute and adapt the code, it is necessary to configure the appropriate environment. Two main strategies were employed: transpiling the code to Python and using a virtual machine.

**TRANSPILATION**

The pygame library was used to handle graphics and animation in Python. The library was installed using the command:

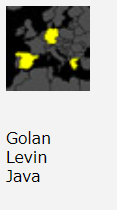
*pip install pygame*

Despite the progress made, the adaptation process has not yet been completed.

**Use of Virtual Machines (VM):**

It is recommended to use Oracle VM VirtualBox with Windows XP and Microsoft Visual Studio 6.0 to execute the original code in an environment closer to the project’s original design.

1. **Golan Levin**

 Picture taken from [2]

**Analysing the code:**

The *"AxisApplet"* project by Golan Levin, created in 2002, explores the concept of political "axes" in an artistic and satirical context. This artwork is developed in Java, using classic structures such as Graphics and image handling via the draw function—features commonly found in applets and graphical applications of that era. While Java remains a widely used language, the AWT and Swing libraries employed in this project are now outdated and have been largely replaced by JavaFX in more recent versions.

**Strategies: Preparing the working environment:**

**DEBUGGING**

To ensure compatibility with modern Java versions, such as Java 17 (an LTS version), it would be necessary to update the graphics handling to be compatible with JavaFX and replace deprecated methods with modern alternatives.

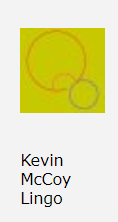
**TRANSPILATION**

One option would be to transpile the code into other languages, such as Python, using graphical libraries like tkinter or pygame, or into JavaScript to create an interactive web application.

**Use of Virtual Machines (VM):**

To run the original Java code, a virtual machine with a compatible version of the Java Runtime Environment (JRE), such as Java 1.4 or later, could be used. This setup would allow the code to execute without significant issues.

1. **Kevin McCoy**

 Picture taken from [2]

**Analysing the code:**

The code is designed to generate an interactive animation using the Lingo scripting language, which was employed in Macromedia Director. The primary purpose of this code is to manage and animate graphical objects (sprites) on the screen, creating a dynamic visual experience driven by the interaction between these objects.

The code handles the movement and animation of sprites within a scene. Each sprite moves according to random parameters and dynamic calculations. Sprites interact with one another, and their movement is influenced by the average position of other sprites in the scene, resulting in a complex animation. The movement cycles of the sprites repeat until all reach their final position, which resets the scene.

**Strategies: Preparing the working environment:**

**DEBUGGING**

**Not applicable**

**TRANSPILATION**

##### ****Python****: Using Pygame for graphics and animations.

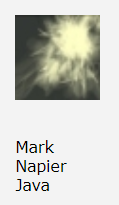
### ****Important Notes****

1. **Sprite Simulation**: A Sprite class was created to encapsulate the properties and methods related to each sprite.
2. **Pygame**: Pygame was used as a suitable library for graphics and sprite simulation.
3. **Preserved Logic**: Most operations related to angles, movements, and collisions were directly carried over.
4. **Adaptation to a Modern Environment**: Management of global properties and flow control in Pygame replaces Lingo's specific functionalities.

**Use of Virtual Machines (VM):**

Use a VM with Windows XP

1. **Mark Napier**

 Picture taken from [2]

**Analysing the code:**

The code is implemented in Java and consists of two files: *Bitmap255.java*, which is the library responsible for screen rendering, and *SpringyDotsApplet.java*, which contains the main logic. The code generates interactive 2D graphics with fading and opacity effects, using a color palette created through gradients. It allows drawing lines and rectangles on an adjustable background, where pixel colors blend translucently with the background, producing smooth visual effects. Additionally, it manages real-time image rendering, resulting in a dynamic and visually appealing representation.

**Strategies: Preparing the working environment:**

**DEBUGGING**

It was not carried out due to installation issues and time constraints.

**Transpilation**

**Python**  
The original code was successfully transpiled to Python, adapting its functionality to a modern environment. The following modifications were made:

1. **Color Palette:** Numpy was used to handle color palette data, simplifying gradient operations and blending calculations.
2. **Graphics:** The Pillow library replaced Java’s graphic classes, enabling efficient image creation and manipulation.
3. **Opacity Calculations:** Implemented through Numpy’s vectorized operations, optimizing the processing of large datasets.
4. **Rendering:** The final generated image can be saved as a .png file using Pillow’s capabilities.

This adaptation ensures compatibility with modern systems and eliminates dependency on outdated technologies.

**Use of Virtual Machines (VM):**

The original code can be executed in a virtual machine configured with the appropriate environment, such as a Windows XP OS running Java version 5.

1. **W. Bradford Paley**

 Picture taken from [2]

**Analysing the code:**

The code, written in Java by W. Bradford Paley, visualizes and reflects on the process of writing and executing a program. It represents three interactive points: the insertion point (white), which follows the order in which the code was written; the fixation point (amber), simulating human word-by-word reading; and the execution point (green), showing how the computer interprets the code. As lines of code are executed, their brightness increases, highlighting their frequency of use. Additionally, the program reads its own source code and displays it on the screen, prioritizing aesthetics and interaction over technical efficiency.

**Strategies: Preparing the working environment:**

**DEBUGGING**

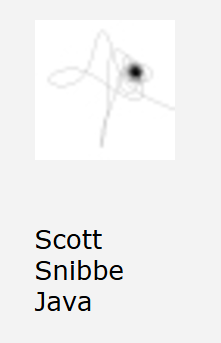
Status: Pending to do

**Transpilation**  
To Python. Status: Pending to finish

**Use of Virtual Machines (VM):**

Status: Pending to do

1. **Scott Snibbe**

 Picture taken from [2]

**Analysing the code:**

The code, originally written in an older version of Java, simulates a chaotic system where a pendulum interacts with three magnets. The pendulum's trajectory, determined by its initial position and the system's parameters, is visually represented on the screen. This chaotic behavior is characterized by its sensitivity to initial conditions, where small variations lead to unpredictable results.

**Main Features:**

**- Chaotic Simulation:**

**-** The pendulum is subject to gravitational and magnetic forces generated by three magnets.

- The trajectory is chaotic, exploring intermediate points between pixels based on the mouse position.

**- Drawing and Animation:**

- A double buffering system is used to smooth the graphical animation.

- The visual representation of the trajectories generates fractal-like patterns.

**- Sensitivity:**

**-** Parameters such as gravity and damping drastically affect the system's behavior.

The result is a graphical representation of chaotic trajectories that blends scientific and artistic elements.

**Strategies: Preparing the working environment:**

**DEBUGGING**

The code uses the java.applet API and deprecated graphical components (Graphics, Event), requiring adaptations to work in modern environments like Java 8 or higher. To achieve this:

- The use of Applet was replaced with JPanel within the Swing framework.

- The logic was updated to ensure compatibility with current Java versions.

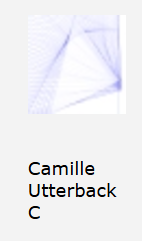
**Transpilation**

To adapt the code to Python, PyGame was used as the graphical interface, allowing for a modern and compatible representation of the original system.

**Use of Virtual Machines (VM):**

A virtual machine was used to run old Java environments, ensuring the preservation of the original code and its functionality before migration or adaptation. Additionally, the VM allows experimentation with older versions of the environment without compromising the stability of the main system.

1. **Camille Utterback**

 Picture taken from [2]

**Analysing the code:**

The program linescape.cpp, written in C, is an interactive graphical application that uses OpenGL to generate dynamic visual art based on the movement of three points (dots) within rectangular trajectories. The points are connected by a translucent white triangle, while their past trajectories are represented by blue triangles that fade over time. When interacting with the program, the user can click to move a random vertex of a rectangle to the click position, altering the speed and trajectory of a point, which generates new visual patterns. The program closes when the ENTER key is pressed.

**Strategies: Work Environment Setup**

**Debugging**

To run the C script on my local machine, it was necessary to install MinGW, an environment that allows for the compilation and execution of programs in this language.

Adjustments were made to resolve issues related to dependencies and the compatibility of the original code with modern systems.

**Transpilation:**

The original code was adapted to Python, using the Pygame and PyOpenGL libraries. This allowed for replicating the graphical functionality of the initial program in a more accessible way for modern environments.

**Dependencies Installation:**

*pip install pygame PyOpenGL*

**Code Features:**

**- Graphical Window:** Pygame is used to create the window and manage events.

- **Rectangle Generation:**

- draw\_random\_rectangles(): Generates rectangles with random sizes and positions.

- draw\_rectangles(): Draws the generated rectangles on the screen.

I**nteractivity:**

**-** handle\_events(): Detects mouse events, such as clicks, and manages user interaction with the graphical elements.

**Main Execution:**

**-** main(): Manages the main program loop, ensuring that the window is continuously updated and that interactive functionality is maintained.

**Use of Virtual Machines (VM):**

A virtual machine was implemented to replicate old environments when necessary, ensuring compatibility with obsolete technologies. This was especially useful for running the original code or making comparisons between the transpiled version and the initial functionality.

1. **Martin Wattenberg**

 Picture taken from [2]

**Analysing the code:**

The code, written in Java, implements an interactive graphical applet that allows drawing lines on the screen based on the user's mouse interaction. It uses three reference points that can be moved by clicking or dragging. The animation is managed by a thread that continuously updates the image in memory and draws it on the screen. The main methods include *init()* to initialize the applet, *run()* to manage the animation cycle, and *paint()* to render the graphics. It also overrides mouse events like *mouseDown()* and *mouseDrag()* to select and move points. The generated lines have colors calculated using geometric operations related to the positions of the points, simulating a dynamic and visually appealing effect.

**Strategies: Work Environment Setup**

**Debugging**

To modernize and run the original code written in Java, the following changes were made:

- **Applet Replacement:** Replaced with JFrame or JPanel from the Swing library to handle the graphical interface since Applet is obsolete.

- **Obsolete Methods:**

- size() was replaced with getWidth() and getHeight() to get the component's dimensions.

- The Event class was replaced with the MouseListener and MouseMotionListener interfaces to handle mouse events in an updated manner.

- **Compilation:**

**-** The javac ConnectApplet.java command was used to compile the file and generate the .class files.

- Attempting to compile with javac -target 1.8 ConnectApplet.java resulted in issues due to incompatibilities between Java versions.

**- Outcome:** Execution was not possible in Java due to version compatibility issues.

**Transpilation:**

The code was migrated to Python using the tkinter library to implement the graphical interface in a functional and modern way.

***Script Name*:** The transpiled program is located in a file called *code.py*.

**Use of Virtual Machines (VM):**

A virtualized environment was used to ensure the execution of the code in a controlled environment compatible with the necessary dependencies.

1. **Maciej Wisniewski**

 Picture taken from [2]

**Analysing the code:**

The code, written in an old version of Java, implements an applet called Couplet that adjusts the background color and displays text based on provided parameters. It uses outdated classes such as *javax.servlet* and requires external libraries to run in modern environments. The main functionality includes reading a parameter called "but wait" to determine the background's shade of gray (ranging from 0 to 255) and displaying three lines of text on the screen. Conceptually, the program is a visual metaphor for a dynamic sundial, representing the passage of time, geography, and history through gray tones and text. Although it does not interact with physical laws, it adapts its display based on geographical and temporal parameters, offering an abstract representation of time and space.

**Strategies: Work Environment Setup**

**Debugging:**

**Issues running the code on modern Java versions:**

1. **Obsolete Applets:** Java Applet was deprecated starting from Java 9 and completely removed in Java 11. Modern browsers no longer support applets.
2. **Use of** getParameter**:** This method has no direct equivalent outside the context of applets and should be replaced with another form of input, such as arguments or a graphical user interface.
3. **Outdated Code:**
   1. Integer.parseInt(getParameter("but wait")) depends on external parameters that now need to be simulated or replaced.
   2. Graphics is an old API that can be replaced with modern libraries like JavaFX or Swing.

**Transpilation:**

The project consists of 2 files: app.py, which contains the program logic, and index.html, which holds the information used to build the HTML response. Run the app.py script:

**Time Zones:** The 6 original time zones are kept in arrays.

**Colors **and** Phrases:** These change dynamically based on the current system time.

**Table Design:** Four rows and six columns are dynamically created in the HTML.

**Result:** A *life\_support.html* file is generated.

**Use of Virtual Machines (VM):**

A virtualized environment was used to ensure the execution of the code in a controlled environment compatible with the necessary dependencies.

**RESULTS**

The following are the results obtained for each artwork analyzed:

1. The code was successfully executed in its original language after making updates to modernize it. Additionally, the logic was migrated to Python, achieving accurate and consistent results with the original behavior.
2. The code was adapted to Python to process a locally stored music video.
3. Running the code with Strawberry Perl did not yield significant results. However, using Python, a webpage displaying the related .txt files was generated.
4. Necessary variables were not completed in Python, preventing results from being obtained.
5. Although the code was migrated to Python, it has errors that require correction.
6. Full execution was achieved in Python.
7. The Python-migrated code requires additional adjustments to be completed.
8. The migration to Python is incomplete and needs debugging.
9. The code was successfully executed in Python.
10. Migration to Python is pending completion.
11. Successfully completed in Python.
12. The migration and execution of the code in Python were successful.

Several cases remain pending and require adjustments, particularly those dependent on virtual machine (VM) configurations. While migrating to Python has been the primary strategy to overcome the limitations of obsolete platforms, the value of exploring alternative approaches is recognized. It is essential to continue researching methods to reproduce scripts reliant on discontinued libraries and technologies, as new tools and approaches may emerge to address these challenges in the future.

**CONCLUSIONS**

After working on and researching each artwork from the event, it can be concluded that this is a fascinating topic that requires time and dedication. Among the strategies proposed and applied in this project, transpilation proved to be the most feasible, efficient, and swift. Migrating the original artwork code to a modern programming language that is well-understood and equipped with suitable libraries for replicating the program's logic significantly facilitated the adaptation and execution of the scripts. This strategy not only preserved the original functionality but also opened up opportunities for exploring new implementations and improvements.

**REFERENCES**

[1] - [CODeDOC | Whitney Museum of American Art](https://whitney.org/exhibitions/codedoc" \l "inbox/_blank)

[2] - [Whitney Artport: Past Exhibitions](https://artport.whitney.org/v2/exhibitions/past-exhibitions.shtml)

[3] - [Perl Documentation - Perldoc Browser](https://perldoc.perl.org/)