Future Academy
معهد المستقبل العالي
للدراسات التكنولوجية المتخصصة

وزارة التعليم العالي و البحث العلمي
Ministry of Higher Education
and Scientific Research

Desktop Application

Vision Visual Scripting

Graduation project as a requirements completion for bachelor's degree in management information systems

written by

Sherif Hany Mostafa Husain

Khyria Abd-Al-Baset Mohamed

Abd-Al-Rahaman Mahmoud

Mennat Allah Mahmoud

Ahmed Attia Tawfik Ahmed

Alaa Ehab Kamel Attia

Under the Supervision of

Management Information Systems

June - 2022

بِسْمِ اللَّـهِ الرَّحْمَـٰنِ الرَّحِيمِ

(قَالُواْ سُبْحَٰنَكَ لَا عِلْمَ لَنَآ إِلَّا مَا عَلَّمْتَنَآ إِنَّكَ أَنتَ ٱلْعَلِيمُ ٱلْحَكِيمُ)

سورة البقرة آية (32)

English - Sahih International

They said, "Exalted are You; we have no knowledge except what You have taught us. Indeed, it is You who is the Knowing, the Wise".

Tafheem-ul-Quran by Syed Abu-al-A'la Maududi

(2:32)They replied, "Glory be to You. You alone are free from defect. We possess only that much knowledge which You have given us 43* .Indeed You alone are All-Knowing and All-Wise".

Al-baqarah (32)

صدق اللّهُ العظيم

The Acadimy's Vision

The Higher Future Institute for Specialized Technological Studies should be a leading and distinct scientific institution in private university education and an active and supportive element for community service and environmental development.

The Acadimy's Message

Preparation of specialized and distinguished scientific cadres in the fields of administrative science, information and computer sciences capable of competing in the fields of work through faculty members holding degrees from prestigious universities, providing an advanced and distinguished educational environment, and developing students’ scientific and practical skills to make them able to innovate, self-learn and and work collectively.

Summery

Vision Visual Scripting (VVS) forms a new vision in the field of programing as it's design targets improving programing workflow through multiple essential components, dampening the learning curve, reducing efforts, limiting time delays and enabling the programmer to interact with any programming language without needing to have previous deep knowlage of it.

Visual programming generally depends on the formation of a logic for a program by using digital graphics, which in turn replaces the written method of coding and enables developers to extract a book code based on the logic that was written using graphic code.

Through having VVS apply the rules and following the syntax of each different programing language, repeated process of writing and adjusting code syntax could be automated which in return reduces chances of human error. Respectivly VVS would be the only software that has this feature which would enable the programmer to selected the desired syntax and generate decent code for it without needing to have previous knowlage about it, instead of relativly running the code in a single language in comparison to other softwares.

VVS targets most programmers starting from beginners and to professionals, where the nature of the programme system allows for its use as an advanced programming tool or a professional development tool. The VVS development methodology depends on being open source, which enables users to contribute to the development of the program system, which in turn enhances the dependence of the use of the program.

To our beloved…

This book is dedicated to the ones who helped us take our first steps, work hard and grow stronger, thanks for being there when we needed you, thanks for supporting, guiding and loving us… our Parents and our Siblings, we appreciate, adore and love every single moment you held our hands through… Thank you.

Thanks…

To the ones who gave us a chance when we most needed it, to our doctors:

Dr. Mohamed Amer

Dr. Mohamed Dosuky

Dr. Saffaa Mohamed

قال رســـول الله (صـــلى الله عليه وســـلم) : "مَنْ صَنَعَ إِلَيْكُمْ مَعْرُوفًا فَكَافِئُوهُ، فَإِنْ لَمْ تَجِدُوا مَا تُكَافِئُونَهُ فَادْعُوا لَهُ حَتَّى تَرَوْا أَنَّكُمْ قَدْ كَافَأْتُمُوهُ". (رواه أبو داوود)

Table of Contents

List of Shapes

Chapter one

Challenges and methodology for research and development

1 Visual programming in general

Learning in software languages requires mastery, investment of time and effort, and keeping up with their developments forms challenges and difficulties in learning those languages. It is certainly essential to use programming languages to build programmes. Many learners find that programming is a skill that is difficult to learn and gain experience in that there are different challenges and difficulties. Vision Visual Scripting (VVS) studies targeting and analysing these challenges, to propose and create an effective, effective and impactive solution that can be widely applied with most of the coding languages.

Several studies have shown that the communication and interaction between the program and the user based on graphics are more efficient in two essential processes of human processing information. Which are storing information and retrieving information. Visual programming enables the user to see the process steps through digital visual graphics, which makes it easier for any user to understand the direction, order and function of each operation and its relative to each others and with variables.

The core VVS mission is centered around enabling the user to form the desired code through visual programming and, then giving the user the ability to choose any programing language that the software supports, to then output written code represented by the visual code made by the user, which creates a flexible framework that has never been present in the programing.

2 Challenges of written programming

- it's relatively difficult for beginners to learn programing, due to having too many rules to follow.
- Beginners are distracted in choosing the coding languages for their similarity of functionality.
- Correction of code mistakes drains a lot of time and effort.
- It's difficult to adapt to the advancement of programming languages and the emergence of new programming languages.
- It's difficult to understand theoretical code in practical applications.
- Some programming languages have no sources of education for different cultures.
- It's difficult to move between the code lines and determine their functions in practice.
- The emergence of code errors when the user doesn't know the right way to write using the coding language.
- The need for constant thinking of the programming process causes mental exhaustion and reduces the programmer's productivity.
- Some programing languages don't support all operating systems.

3 Suggested solutions

1- Introducing and explaining the concepts of programming using drawings.

2- introducing beginners to the concepts and principles of programming before they learn programming languages.

3- Depending on softwares that are not based on a programming language in itself, but on the general basis of programming in a comprehensive manner.

4- Improving the programming workflow so that it's easy to learn and use in a short time.

5- Use graphics to represent code in terms of process sequences and variables, in simplified, flexible and rapidly adjustable form.

6- separate the logical code from the actual code (syntax).

4 Software objectives

- Simplify the programming process, provided that the efficiency of the proposed solution (using graphic code) is similar to the efficiency of the written code method.
- Make the code easy to read no matter the culture or language of the user, through enabling the user to change the UI language of the software.
- Dampening the programming learning curve by reducing effort, limiting time delays, reducing minor tweaks and fixing mistakes.
- Covering a large number of most commonly used programming languages and empowering users to convert code into any language they want.
- Developing a new instrument for programming without getting far from the foundations and principles of programming languages, and develop in the same direction.
- Enabling programmers to handle any programming language easily and effectively without having to study it in depth.

5 Software importance

- Developing the field of programming for future generations by providing tools and instruments to improve the programming process and making it less time consuming and more dependent for users.

- Removing some basic barriers that pose challenges and difficulties, such as clarifying the practical concepts of software in virtual texts by visualizing them in digital graphics.

- Enable developers to use pre-made logical codes regardless of the targeted code.

- Uniting the general concepts of programming principles for new users and enable them to practice programming based on the logical functions of the code without having to learn one language deeply.

- Innovating the format of programming from routine writing to interactive graphics format which urges thinking of the code in a broader perspective and reduces the mental and visual stress of users.

6 The foundations and principles of development

VVS aims to find an appropriate solution that shortens and make manual programming challenges simpler and more reliable, aimed at shortening time and reducing efforts for users and giving them more space to focus on innovation and creativity instead of routine error fixing and repeating writing codes. VVS focuses on improving the programming Work Flow, with a view to dampening the learning curve for the users through utilizing graphics to improve communication and interaction between the program and the user, taking into account the different cultures and the mother language of the user but keeping in mind the progress made in the programming field, working with it, utilizing and building on it, and making sure to work in the overall direction of the programming field

Directional thinking

With the differences between programming languages and the method of programming, we note in different languages that programming is limited to two essential elements, namely, operations and variables. This was not just a coincidence, but different languages were designed to take into account fixed foundations in all the programming languages that worked at the same level, which what the directional thinking is in the programing field, which VVS is taking into consideration. The objective of the software is to build a comprehensive software instrument that can utilize and build on what has been developed from programing principles, while maintaining the efficiency , harmonization and functionality of previously written coding methods and not creating a replacement solution for the previously written code method

7 Similar Software

7.1 Unreal Engine Blueprint System

Unreal Engine is an engine for building and developing 3D games based on Blueprint System visual programming to write code and then turn it into C++ to be processed and implemented. Blueprint System is a graphic language that doesn't contain codes in its visible code and it's classified as Object Oriented Programing, it enables users to build some logic by building a group of entities and linking them to each other to implement a set of common instructions.

7.2 3Ds Max Slate Material Editor

It's a graphic screen that gives the user a set of graphic entities that can be moved and linked to other entities by following a logic determined by the user. The product of execution of these entities is representation of material such as fabrics, metals or wood used with a 3D object to produce a realistic graphics image format.

7.3 Blender Geometry & Material System

Blender is free and Open-source program that creates 3D objects. It was provided with the object-building system through the use of node editor after the success of its previously implemented materials-building node editor system for it's ease of use, and since blender is open-source this decision was made collectively between the Blender community of over 9,000 users and developed by the 2022 Discord statistics.

8 Foundations of the VVS system

After analyzing and studying the above systems, it was necessary to identify and effective framework, after a research, comparisons and trade-offs between several possibilities it was decided this framework would best serve this project in the long and short run.

(Python - HTML - PyQt5 GUI library - PyCharm - GitHub - Trello - Diagrams.net)

During the research and study journey, an open-source software of a graphic system called Node Editor Made by Pavel Krupala is identified to the required specifications and it's written in the Python programming language, which was used to be a solid foundation for building, after updating the libraries used and the Python language to the latest version some errors appeared which had to be solved to be able to the development of VVS, after relentless search and a lot of trial and error, the team was able to work the errors and the journey began develop a prototype version of VVS.

Pavel Krupala:

Pavel Krupala is a programmer and a youtuber who built the (open source Node Editor), who documented its construction in a free training course on the YouTube platform and then shared it on GitLab platform as an open source project with the details necessary to understand the program as a whole.

At the end of this course, Pavel wrote and explained a sample calculator program that can perform basic calculations (addition, subtraction, multiplication and division) using his program to show the possibilities of developing with it.

Chapter two

Analysis of VVS software requirements

1 Functional requirements

- **Identifying the programing language**

  The software development had to use flexible, easy-to-write, high-performance programing language that supports graphical user interface Libraries and, takes into consideration the direction of going open-source, Python was chosen as the primary language with some other languages such as HTML.

- **Supporting the visual programming system**

  The software is based on the principle of programming with digital graphical representations and this type of programming does not depend on learning programming languages at all but on understanding the logical relationships of a function to produce a working programme, which requires the software to deal with a flexible graphical interface that illustrates the logic and order and flow of operation in the software.

- **Providing code generation feature**

  Establishing a sub-system that manages the process of generating text codes from graphical codes created by the users automatically of the software language that the user may choose at any stage during the programming process.

- **ability of saving and loading files**

  Implementing saving graphical codes and loading them using the default OS file system enabling the ability to share them.

- **support for built-in libraries**

  Depending on the fact that the software is open-source, basic functions must be supported in the default libraries of VVS, in addition to enabling users to create their own graphical codes libraries and share them.

2 Non-functional requirements

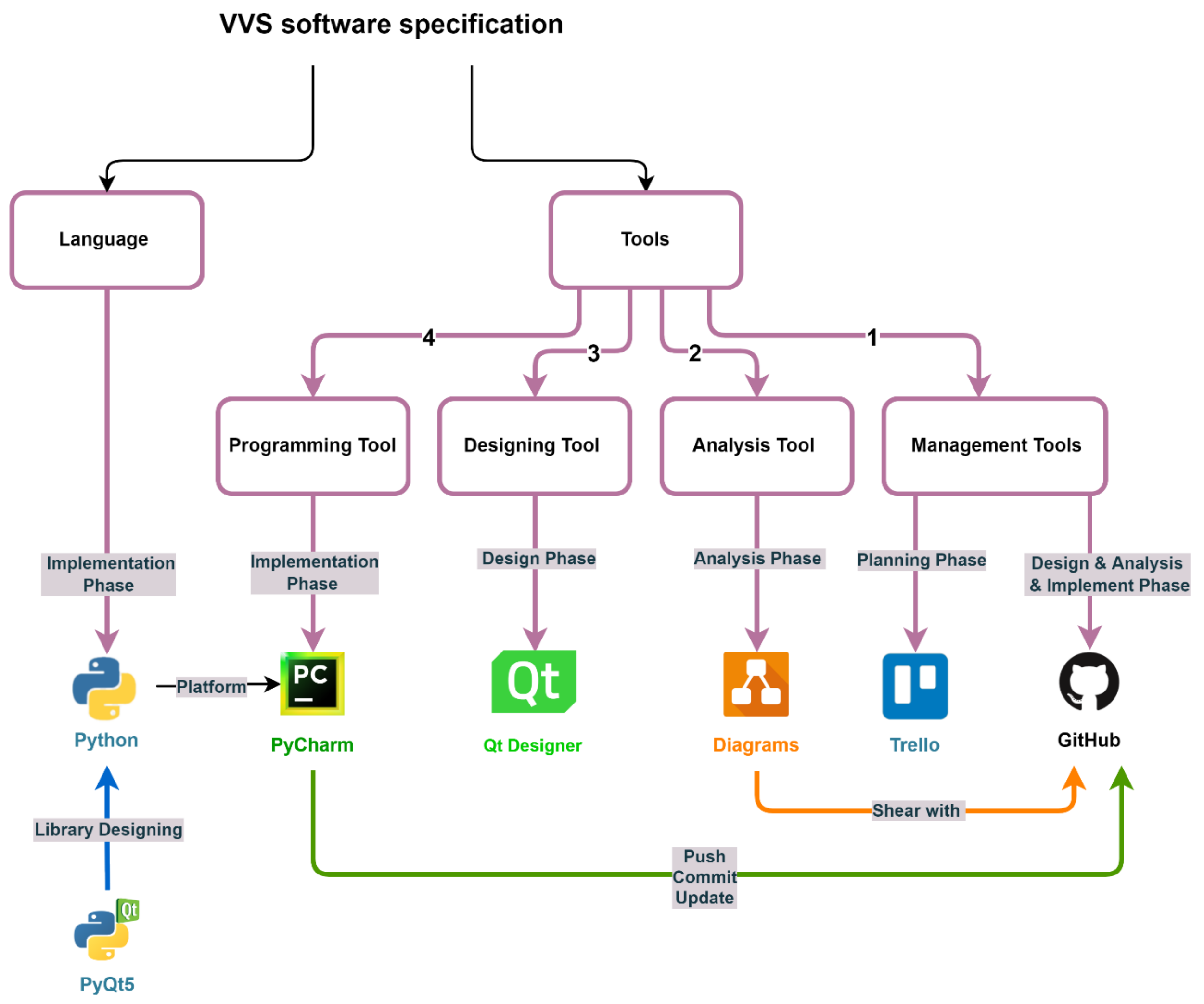
- **The ability to link the system of VVS to other systems**

  According to the VVSs development principles and methodologies, it must be flexible enough to work in harmony with previous and modern programming tools and software.

- **Supporting the most used operating systems**

  In the prototype version of VVS, it supports only Windows OS, but the software development plan seeks to support Mac OS, Linux.

3 VVS Framework



shape1 - VVS Development framework

Trello

Trello is used in the management development process in terms of organizing tasks, monitoring the workflow, tasks accomplished, record and plan for necessary adjustments and future plans.

GitHub

GitHub is used to share and update all of the code through PyCharm and software charts and schemas from the Diagrams.net platform.

Python

Python has been chosen for its wide spread and use in many fields and the availability of powerful graphic user interface libraries like PyQt5, and for easy handling and continuous updates of the language.

PyQt5 Library

A flexible and famous graphical user interface library that exists in several other programing languages, known in the software development field as the most common and comprehensive GUI library, and also has an official documentation site with all the functions and uses of the library and it's regularly updated.

PyCharm

PyCharm is a free integrated development environment used in programming with several languages including Python, which helps with code analysis, detecting code errors, and could be used as a compact modular testing tool.

4 The interface and the users

4.1 Users

The user interacts with the system through a flexible graphical interface that enables it to adjust and utilize the software characteristics to take out a graphical code that can be directly operated or used outside the software.

4.2 VVS's interface

It's the interface which the users interact through to enable them to practice visual programming and adjust graphical code.

- **Text Editor**: A window that shows the generated text code from the graphical codes, and through it the graphical code can be converted into text code format with the programing language that the users select.

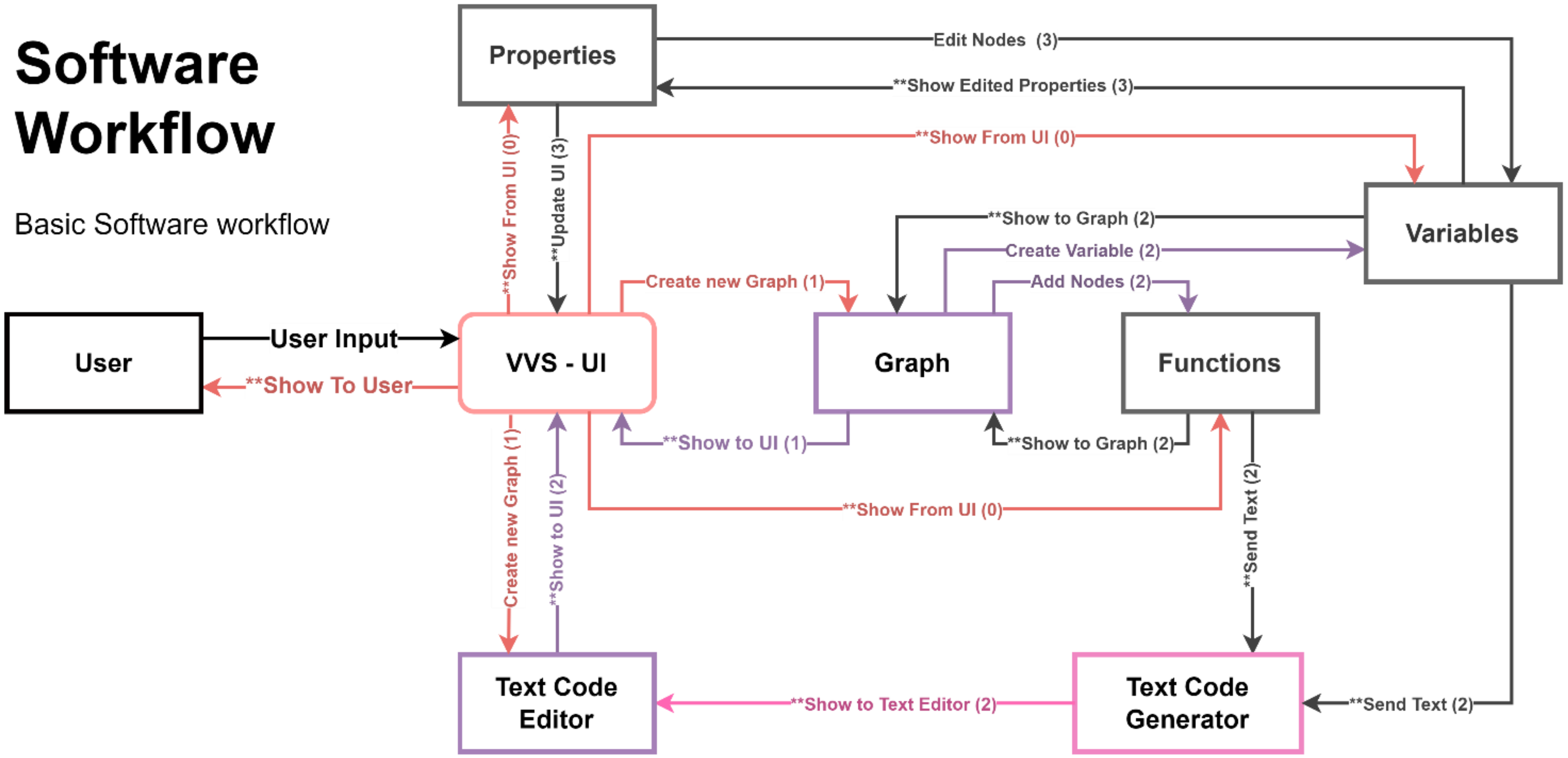
- **Node Editor**: The user can create and modify graphical entities and their interrelationship with the text code window to then do code generation.

- **Functions list**: A list of graphic codes each containing a specific text code has a constant core function among the different programming languages.

- **Variables List**: Enables the user to create variables of different types and use them in the process of building logical graphical codes.

- **Properties List**: Enables users to control the settings or properties of all interactive graphical entities in the software.

5 Software Workflow



shape 2 - Software Workflow

1- Automatic processes happen first (when the program starts)

2- Three lists appear on the graphic interface

- Functions List
- User Variables and User Functions Window
- Properties window

3- Creating New Graph from the File menu shows on the interface

- Graph
- Text Code Editor
- User Variables and User Functions List

4- The Graph is ready to receive Nodes that including

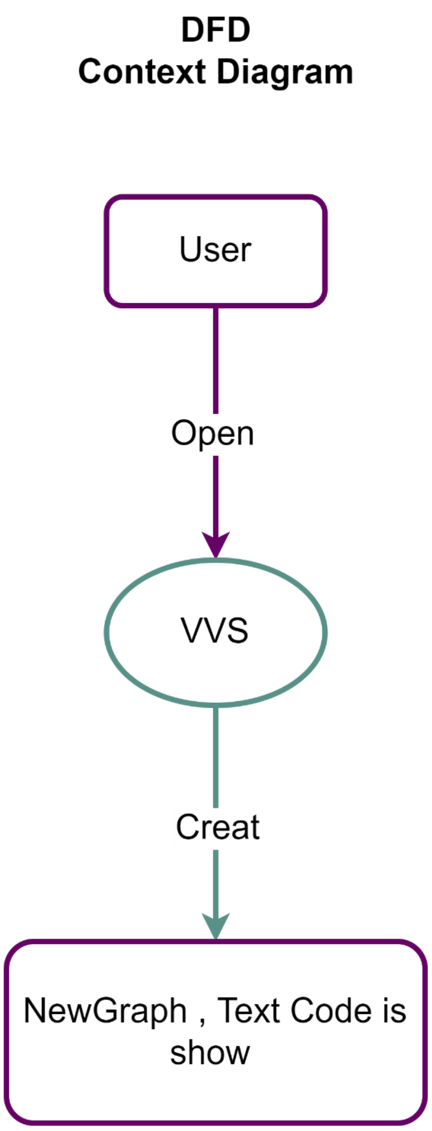
- Functions
- User Variables
- User Functions

By doing Drag and Drop, it shows on the Graph window followed by the code generation operation automatically and the text code shows in the Text Code window

5- User Variables and User Functions can be adjusted through the Properties Window.

6 Data flow Diagram

6.1 Context Level Diagram

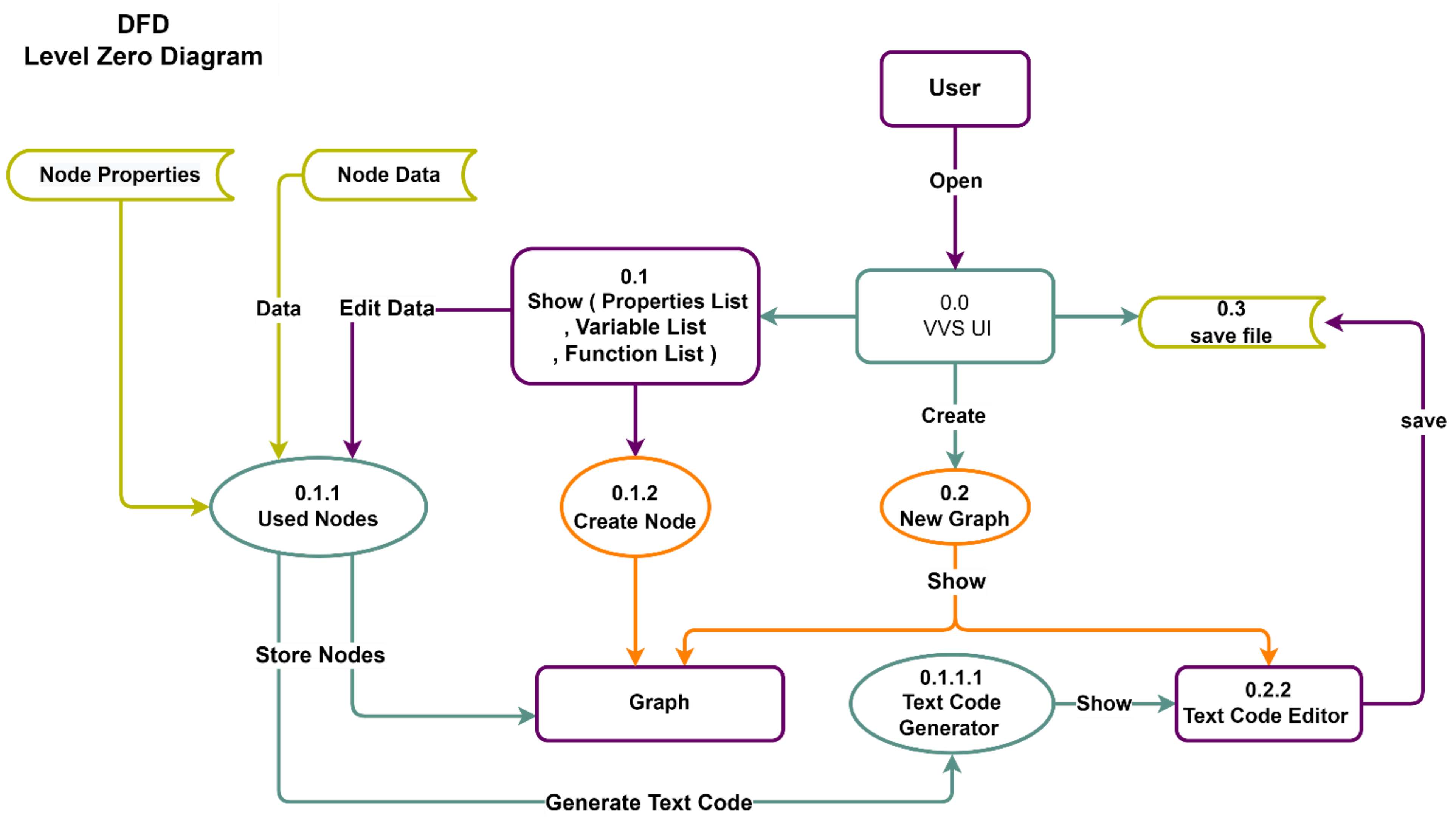


shape 3 - Context Level Diagram

Processes of context level:

1- The user opens the software.
2- Creating a new graphical interface.
3- Graph shows up and the Text Code Editor window appears automatically.

6.2 Level Zero Diagram



shape 4 - Level Zero Diagram

The flow of operations is described at zero level and the Data Stored showed for each node addition, adjustment and saving

1- The user opens the software then the three lists of Functions, Variables, Properties are shown.

2- The user creates a new graph.

3- The Graph window appears then the Text Code window.

4- Nodes can be dragged and dropped and adjusted from the Properties window.

5- The script code appears in the Text Code Editor window, so the user can save the script in a File and copy the code to use it.

Chapter Three

VVS Design

One of the most important systems in VVS is the system of creating codes in digital graphics, and this part is represented in a system known as Node Editor in the GUI development field, which includes key elements common to all visual programming software.

- **Connections**: connections that ties in to each other to create logical relations that determine the order of each entity's function in relation to other entities.

- **Nodes**: Digital graphic representation that store a set of written codes that are connected using graphic connections within the graph to create an ordered logical processes.

- **Graphs**: An empty space for the presence and manipulation of entities and its relations, it represents Code Text Editors but in graphic format.

- **Node Editor**: It represents the control panel that enables the user to create and modify graphical entities and their relations .

To build the Node Editor system, a number of systems developed in other software have been studied and used which have proven node editors to be effective, these programs rule the field with their technologies, such as:

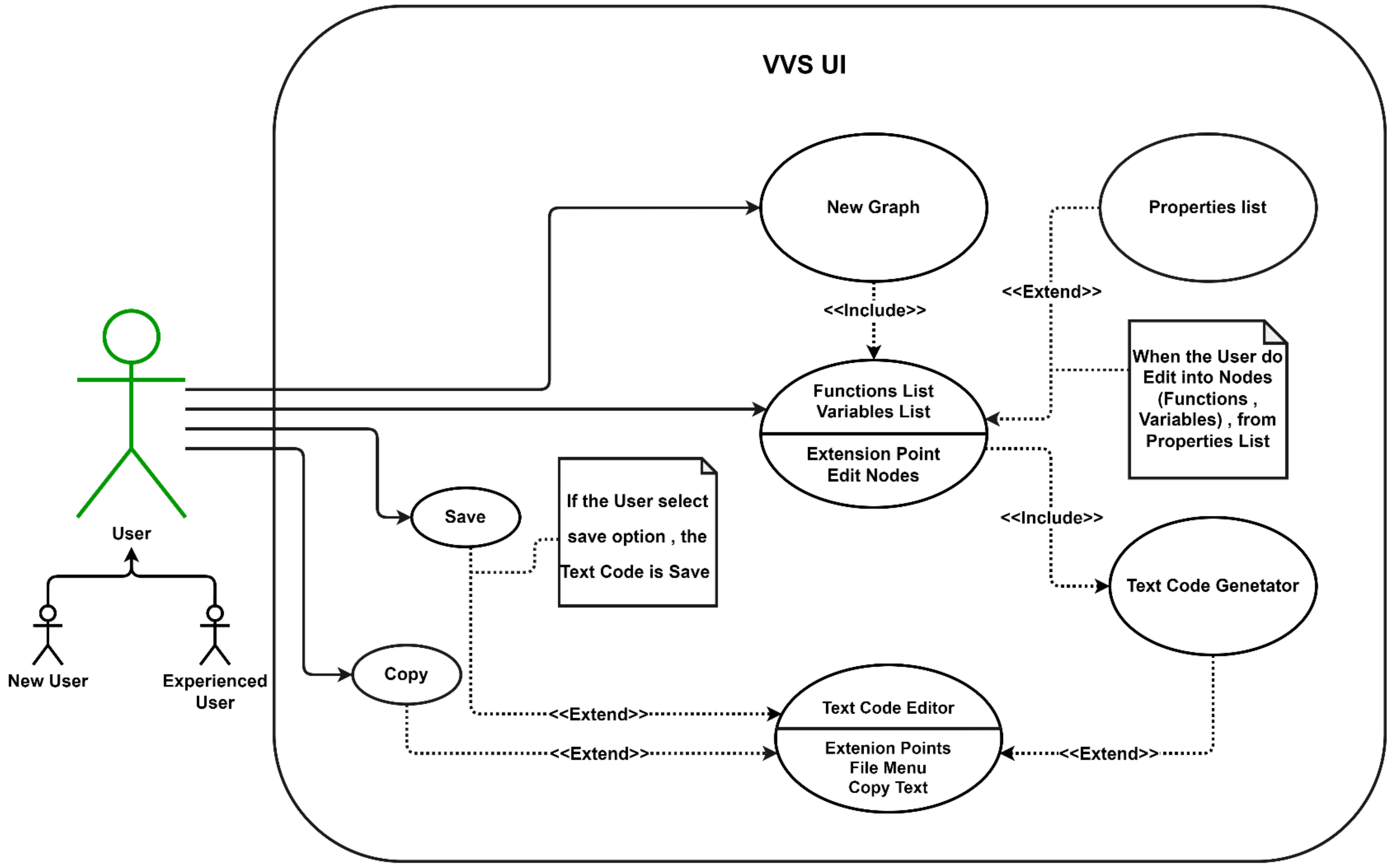
- Unreal Engine**: Blueprint System**
- 3Ds Max**: Slate Material Editor**
- Blender**: Geometry & Material System**

1 Users and use cases

1.1 **Users**: users are key elements that interact with VVS, they don't require to have programming experience, they can be in junior level and still easily interact with VVS systems.

1.2 **Software interface**: The lists and output of user interactions are shown through it, it includes the following

- **Lists**: It is a set of orders divided into several lists to provide the user with the possibilities of adjusting and adding graphical entities.
- **Operations Output**: It represents the end product of the functions and orders that the program processes to then be used with other software.



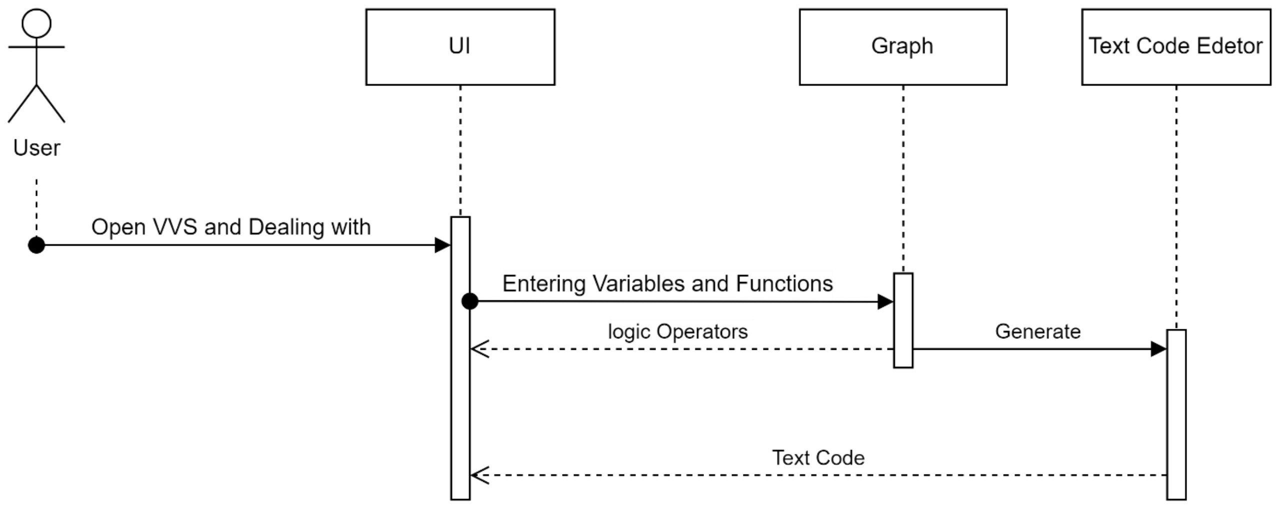
shape 5 - Use Case Diagram

2 Operations Sequence Diagram

1- The user opens the software and shows this process on the program's UI.

2- Through the user interface, New Graph is created and Functions Variables are used within the Graph.

3- A logical transformation of graphical operations occurs to generate a text code that appears in the Text Code Editor, which is reflected to the user through information from the software interface.

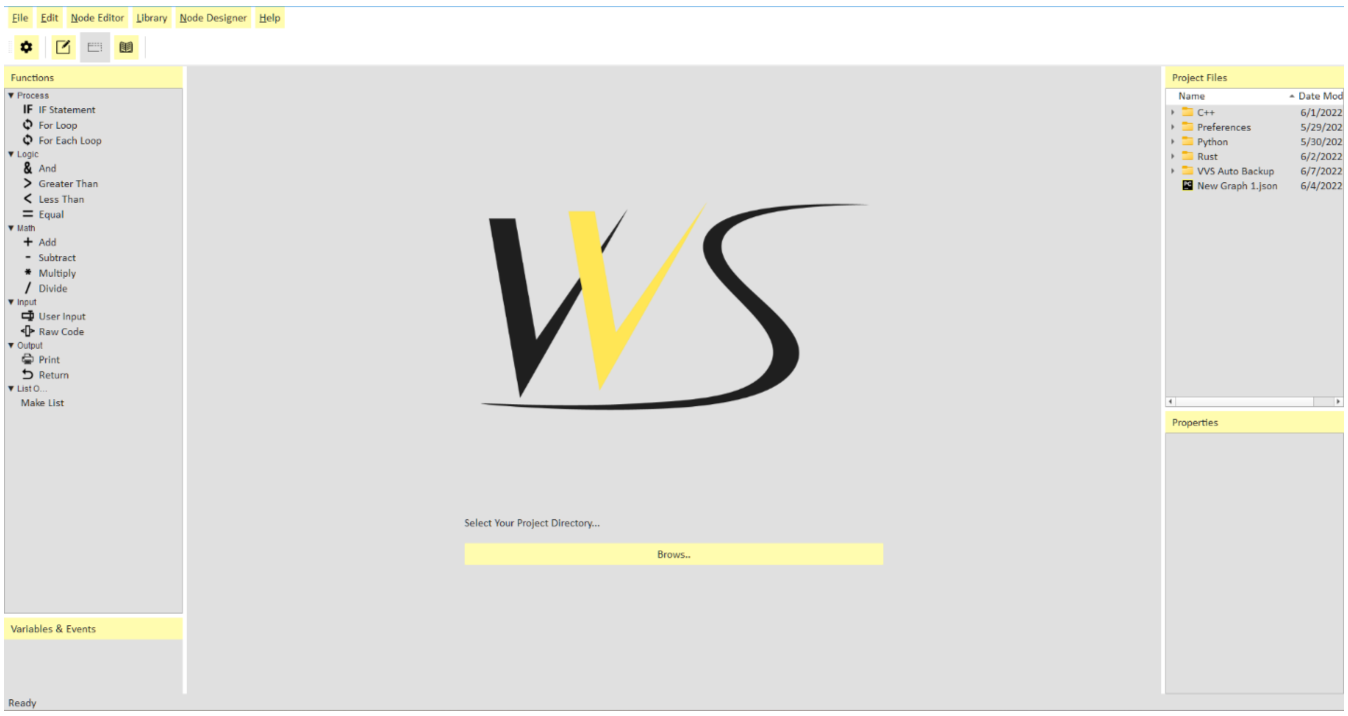


shape 6 - Operations Sequence Diagram

3 Design and construction of the User interface

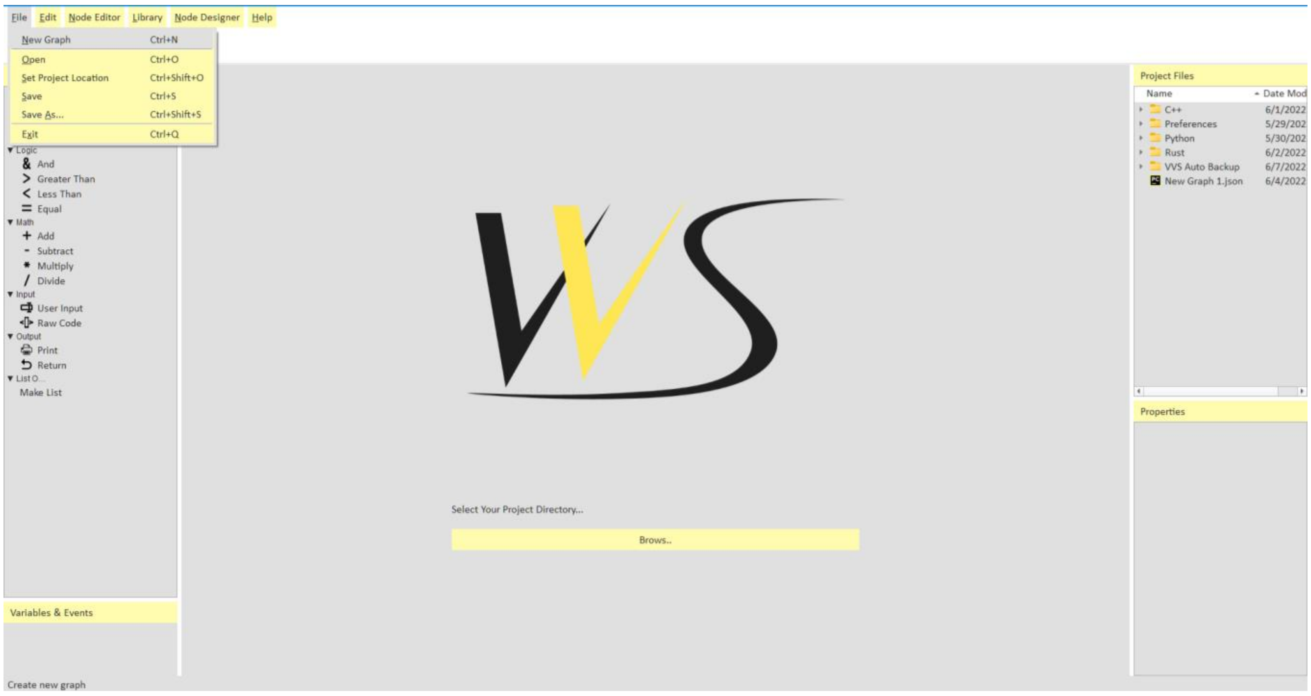
In the next example, an application that shows how the prototype version works and its effectiveness, an illustrative example of how to design a simple program that prints out a message that indicates whether a person is young or old, we create a variable, and enter a value for a person's age if Ahmed is younger than 50, the script prints the word Young, otherwise it prints out Old.

1- This is software interface that the user interacts with, when the user first starts the software:



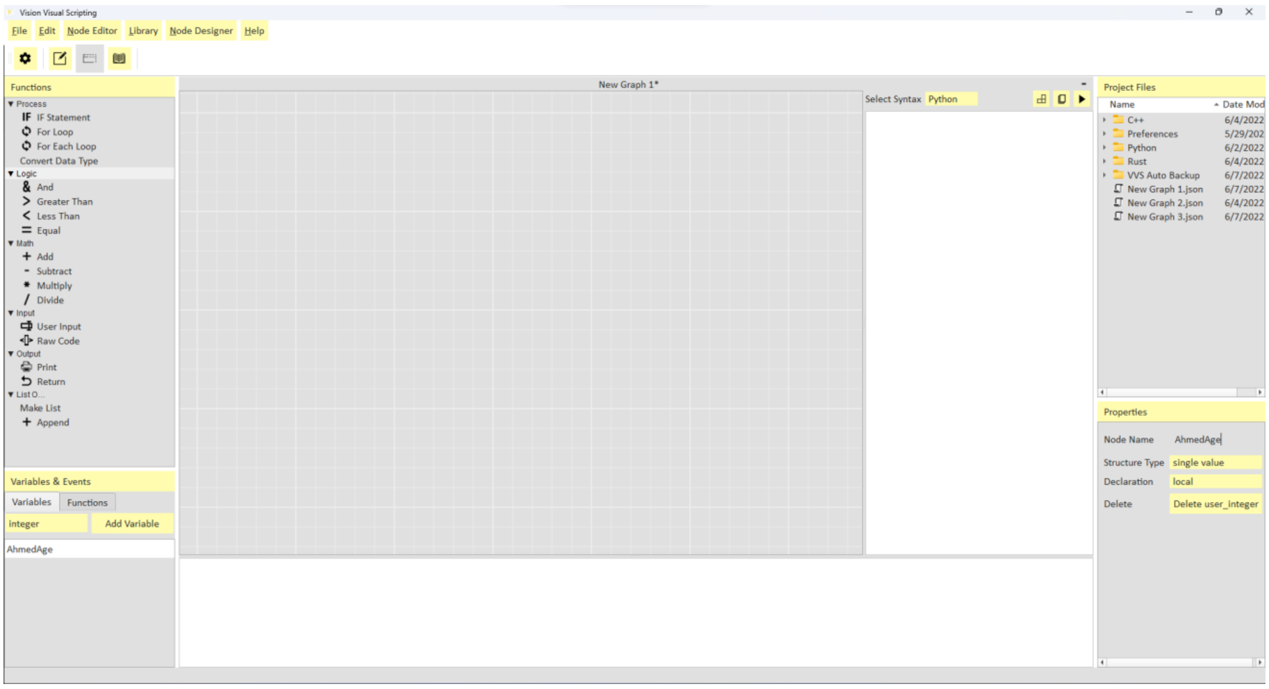
shape 7 - Software User Interface

2- The user creates a New Graph through the File menu or opens a Graph that was already saved:



shape 8 - Creating a new graph

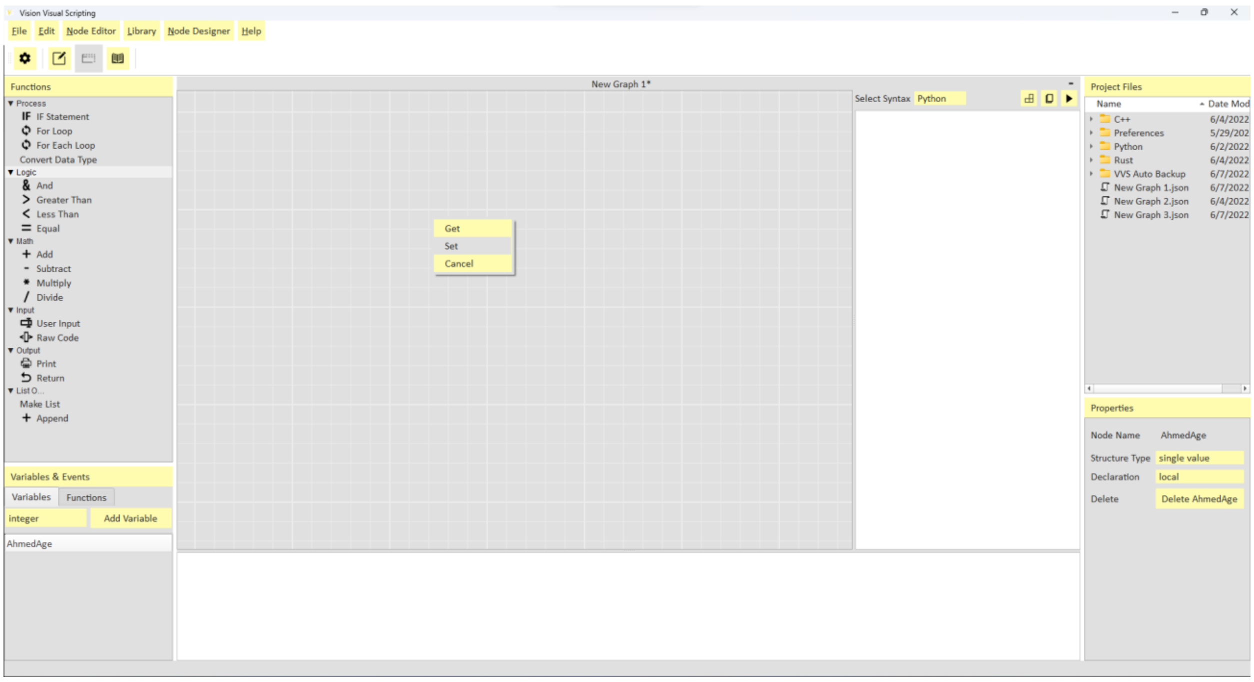
3- Adding integer to the file by pressing the Add Variables button, and then renaming it from the Properties panel:



Node Name	AhmedAge
Structure Type	single value
Declaration	local
Delete	Delete user_integer

shape 9 - Creating a variable and renaming it

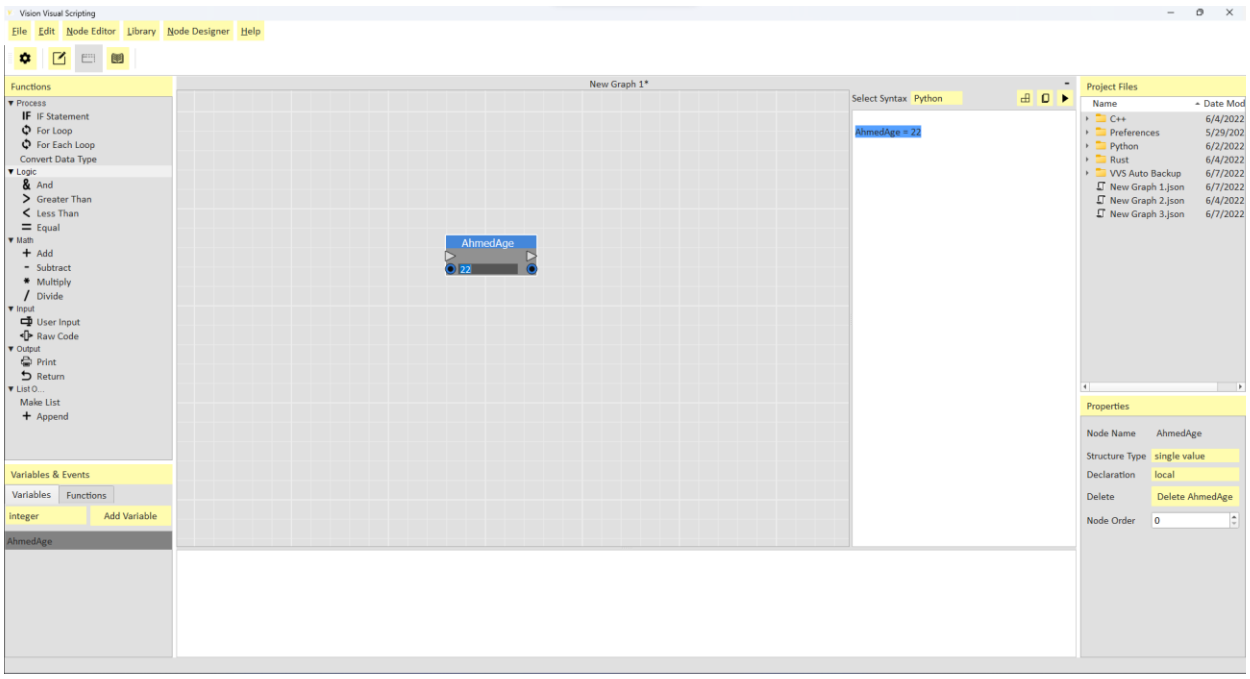
4- After the variable is renamed to Ahmed_age, it is inserted into the Graph by dragging it and calling Set from the menu that appears, which is the process setting the variables value:



Node Name AhmedAge
Structure Type single value
Declaration local
Delete Delete AhmedAge

shape 10 - Setting Variable Value

5- Entering a new value to the variable, it then automatically appears in the text code in the Text Code**:**

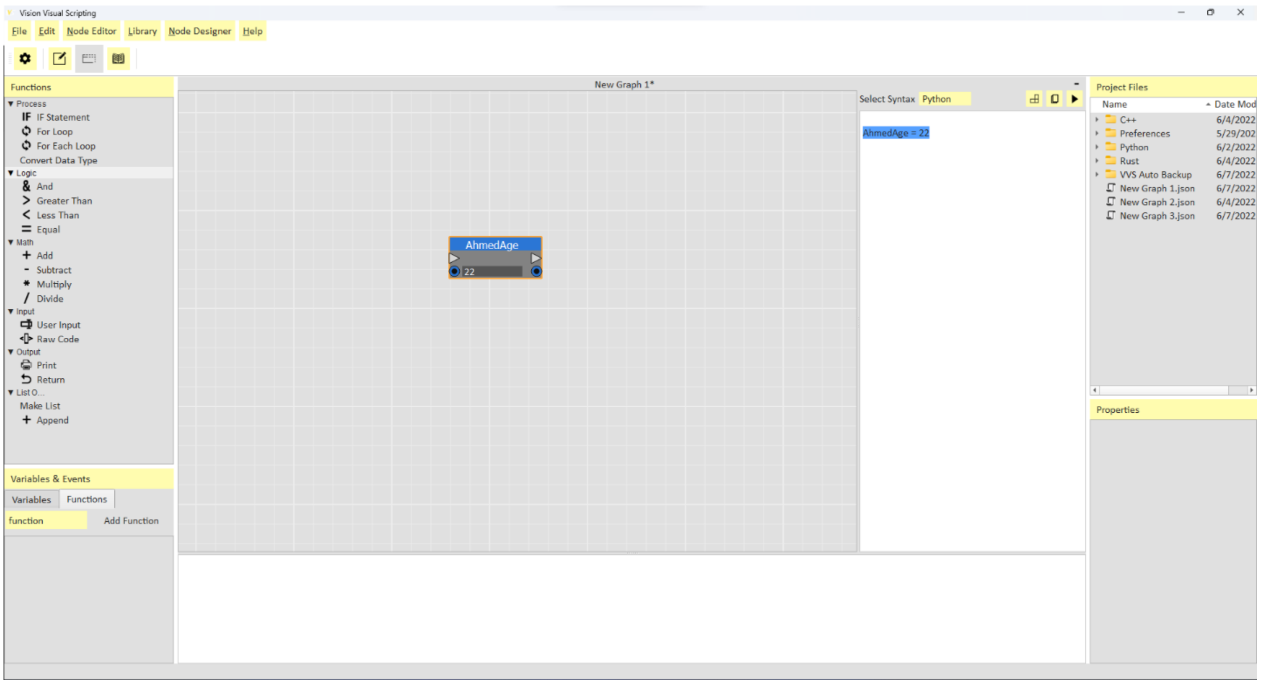


```
AhmedAge = 22
```

Node Name AhmedAge
Structure Type single value
Declaration local
Delete Delete AhmedAge
Node Order 0

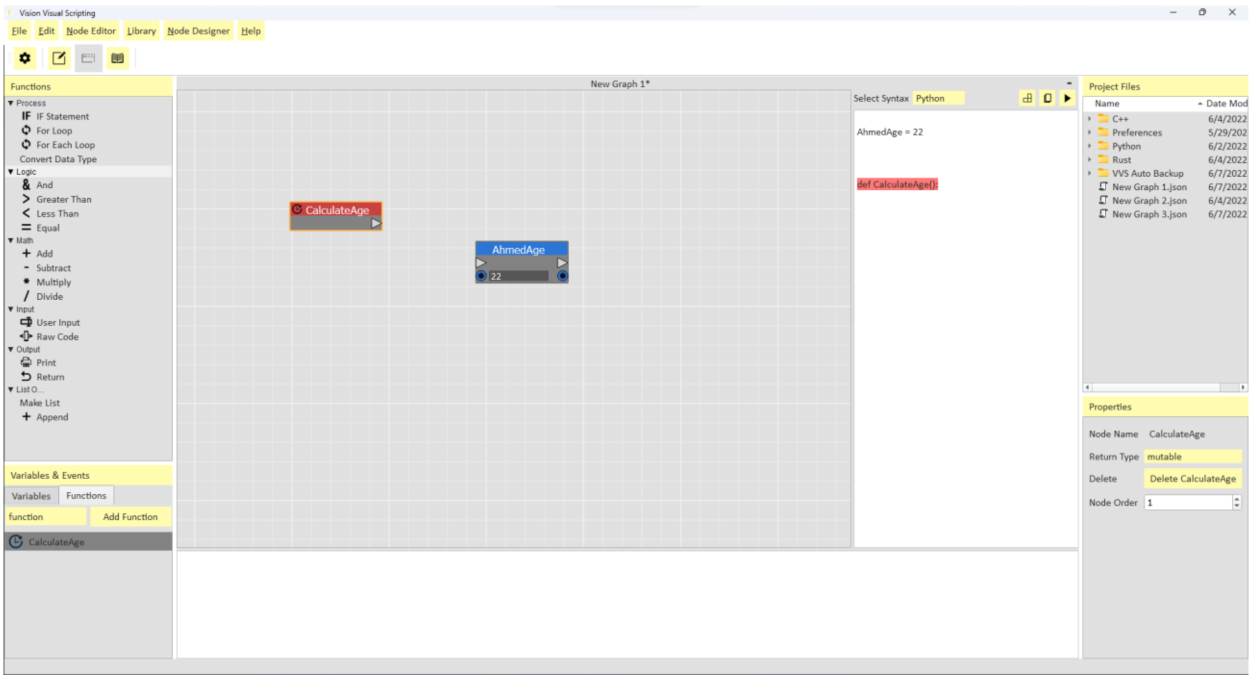
shape 11 - Entering a new value for the variable

6- Creating a new User Function and renamed it:



```
AhmedAge = 22
```

shape 12 - Creating a enw user function



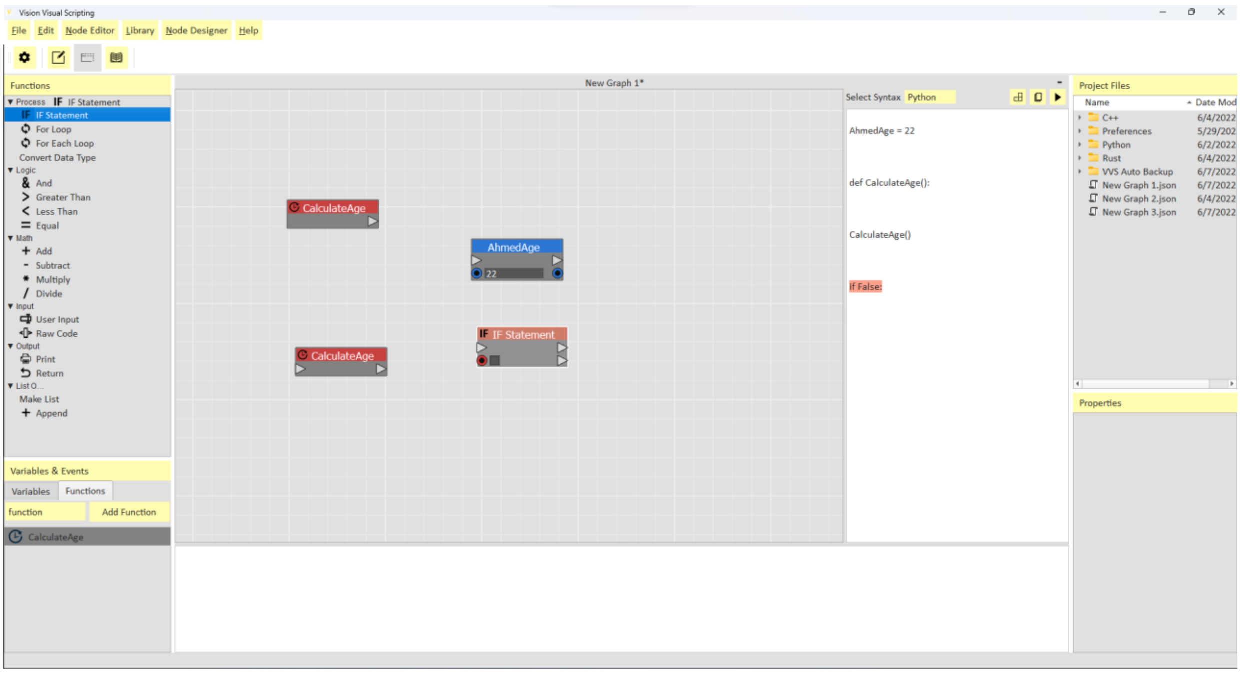
```
AhmedAge = 22

def CalculateAge():
```

Node Name	CalculateAge
Return Type	mutable
Delete	Delete CalculateAge
Node Order	1

shape 13 - Renaming the user function

7- Doing Drag and Drop and selecting the Write order to later use it to contain logical code.

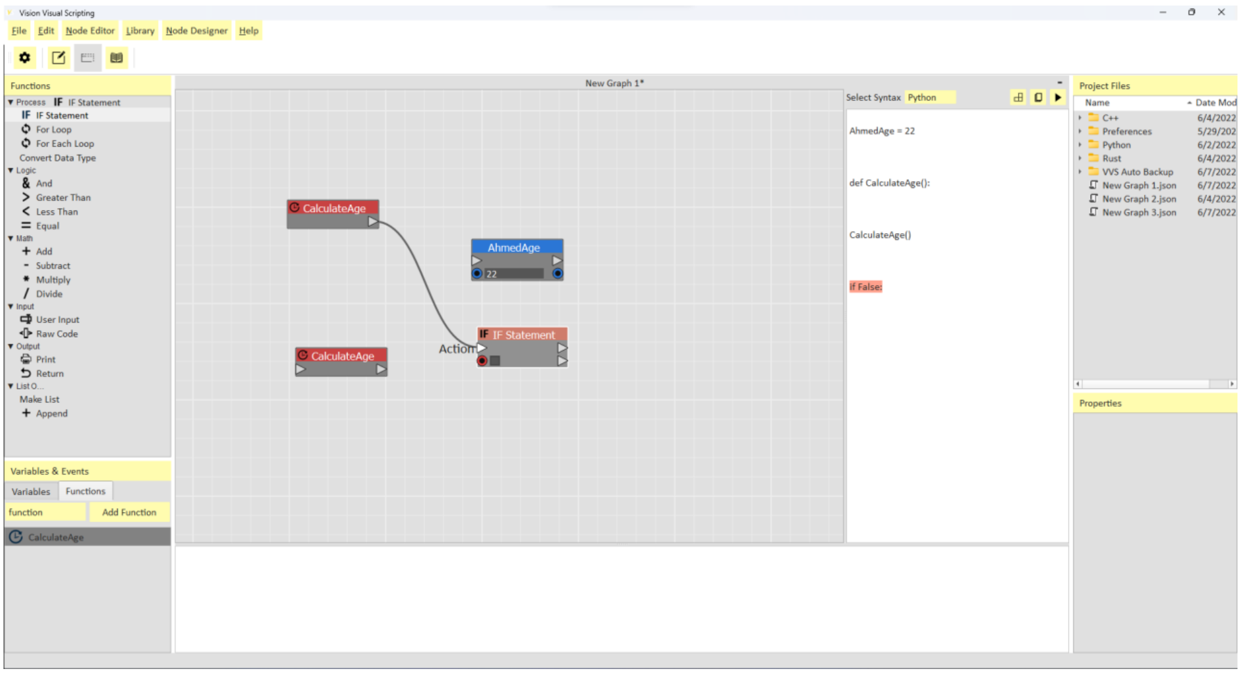
8- Calling a Function, which is the logical IF Statement:



```
AhmedAge = 22

def CalculateAge():

CalculateAge()

If False:
```

shape 14 - Calling an IF Statement operation

9- The IF Statement operation is connected to Calculate:

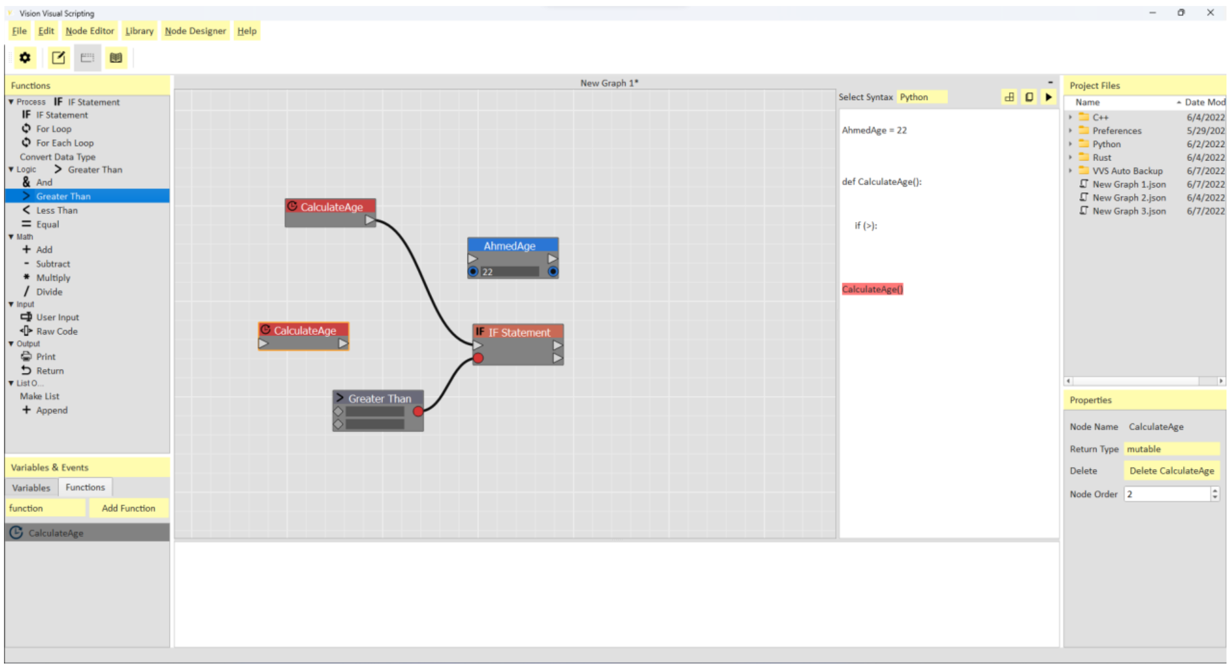


```
AhmedAge = 22

def CalculateAge():

CalculateAge()

If False:
```

shape 15 - Establishing a relationship between graphical entities

10- Using a Larger Than node, through dragging & dropping it to the Graph from the Function list**.**

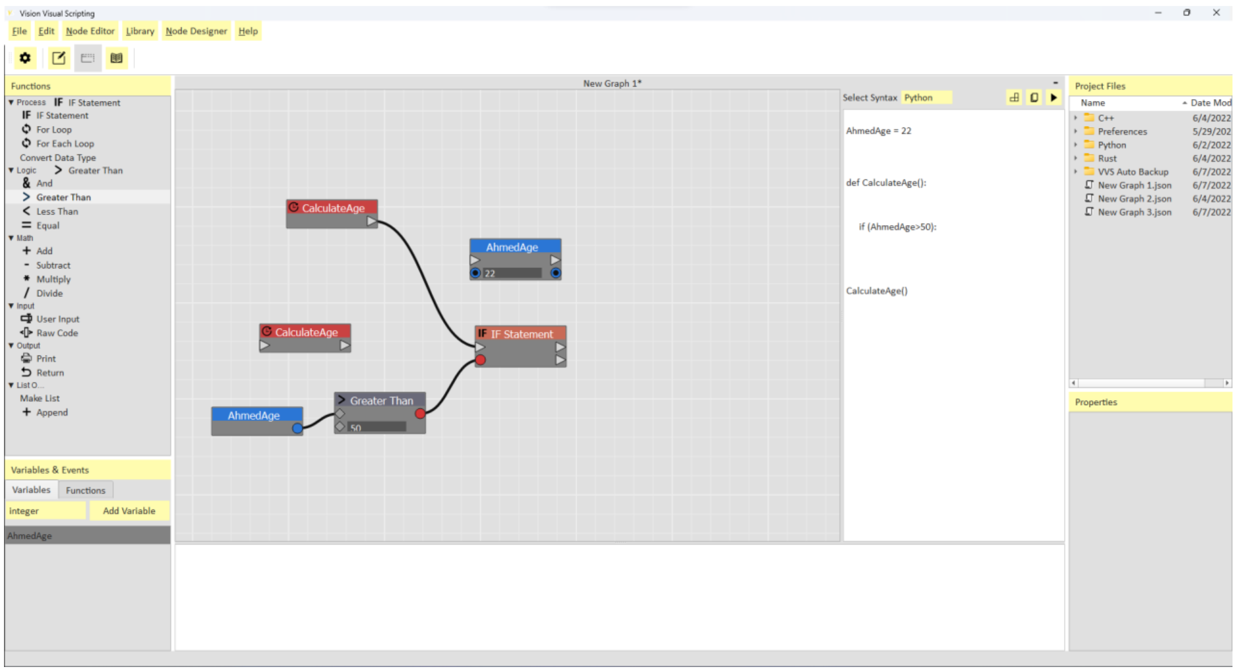
11- Connecting Larger Than Operation to the IF Statement:



```
AhmedAge = 22

def CalculateAge():

    if (>):

CalculateAge()
```

shape 16 - Using a Bigger Than Operation

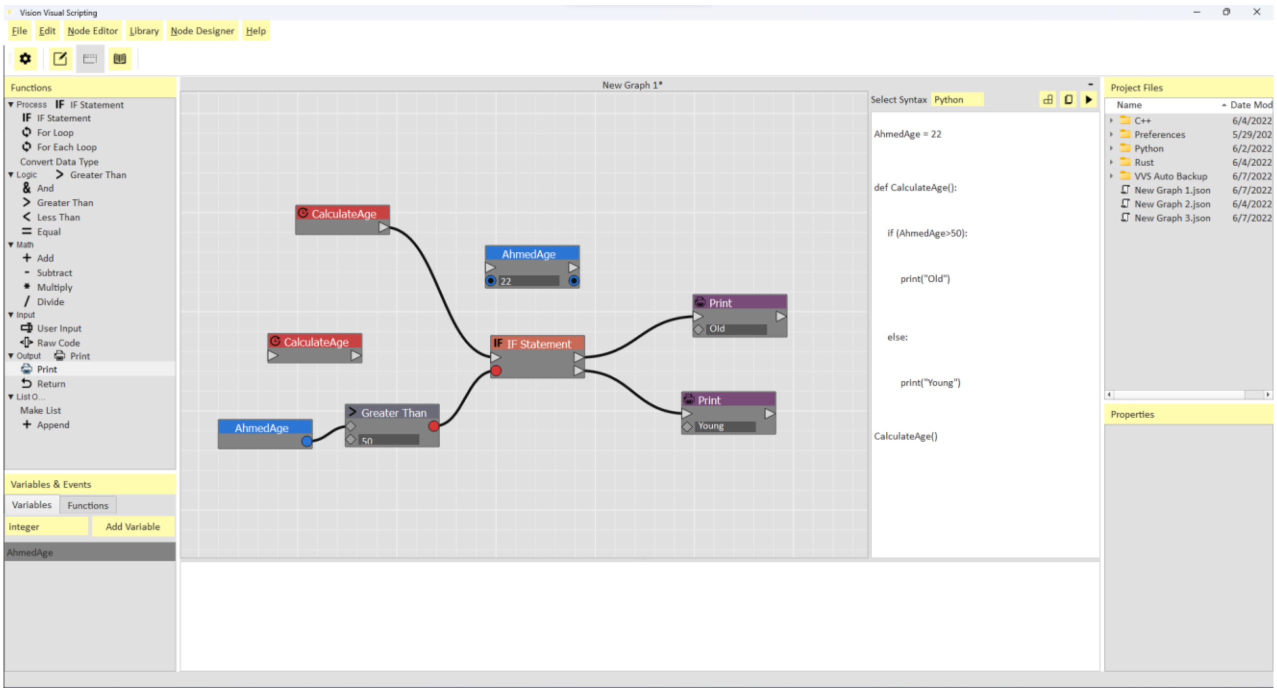
12- Calling the variable Ahmed_age using the Get menu selection and connecting it to the Greater node:



```
AhmedAge = 22

def CalculateAge():

    if (AhmedAge>50):

CalculateAge()
```

shape 17 - Calling variable value

13- We introduce two Print nodes to print out two messages, then connect each of them to the logical output of IF Statement node:



```
AhmedAge = 22

def CalculateAge():

    if (AhmedAge>50):

        print("Old")

    else:

        print("Young")

CalculateAge()
```

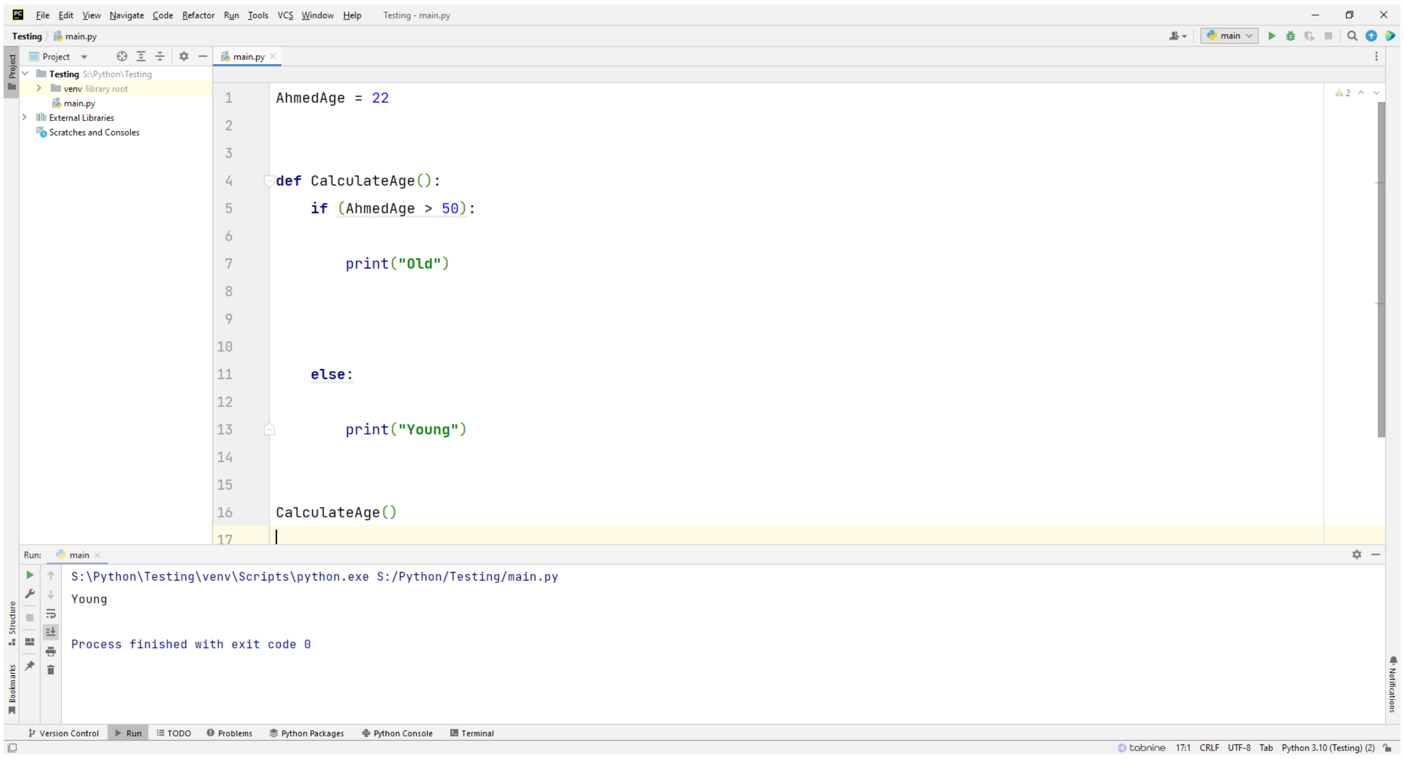
shape 18 - Useing the printing node

14- We note that:

- For every entry of Variables & Functions, the software translates all of the Nodes into Text Code in the Text Code window in real time.

- After connecting Variables & Functions Nodes to each other and building Logical Code, the output is a text code script that can be used in any other software.

15- After the final code is copied to PyCharm and running it Run, the output print was "Young".



```
AhmedAge = 22


def CalculateAge():
    if (AhmedAge > 50):

        print("Old")



    else:

        print("Young")


CalculateAge()
```

```
S:\Python\Testing\venv\Scripts\python.exe S:/Python/Testing/main.py
Young

Process finished with exit code 0
```

shape 19 - Runnig the generated code

16- We note that the workflow of the software didn't get modified by the user after it was exported to PyCharm, which is one of the most important objectives of the experimental version.

Chapter Four

Development and maintenance

- As it was clarified that the nature of VVS development depends on the original developers and software supporters through making it open source, it is necessary to follow the official GitHub page, review, accept and ensure that the commits pushed are reliant efficient.

- The official website of the program must be maintained up and updated to the latest stable version of the software.

- Following the base principles for maintaining software and developing according to future changes and variables in the programing field.

    - Corrective maintenance
    - Adaptive maintenance
    - Perfective maintenance
    - Preventive maintenance

Conclusions

- Software development is a difficult field to learn and requires a lot of time and effort to produce a professional program in a short time, but there are other ways and solutions that can comprehensively improve the software field with a wide scope of evolution.

- Visual programming is an instrument to produce code and its not a programming language itself.

- The existence of multiple programing languages has led the programmers to focus on finding solutions to problems that exist in these languages, not focus on developing in the principles and foundations of programming itself, which has led to thinking of new ways to approach programing, which is programming using graphics.

- Clarifying the unity in the principles and foundations of programming languages shows its easy to learn software development, resulting in user investment time in learning logic instead of programing language syntax which in turn leaves wide space for innovation and creativity in the programmer's time.

- Graphics and visuals have the ability to simplify most forms of information and its usage forms an intuitive understanding quick to recall and practice.

- The way every syntax is written can be completely dispensed with, but the software logic of processes and variables cannot be dispensed with.

- Tackling the basis and nature of something to improve its productivity can dramatically affect it in the short and long term

Recommendations

- The roadmap of the software includes many important technical features that will have a radical impact on the efficiency, effectiveness and scope of VVS usage, the most important of which are:
  - Empowering the users to change the interface language of their mother language.
  - Enable the user to change the direction of the code to be compatible with the language used for the interface.
  - Developing the software to enable it to Run the code for all supported programming languages.
  - Developing the library system to enable sharing graphical logical codes among users online.

- Developers are allowed to participate in adding their ideas of new features by making the software development open-source and accessible to all Open Source.

- The programme development plan must include covering all most used programming languages.

- Enable the user to create and share graphical logical codes(nodes) and libraries within the program using the Node Designer feature to make the software more flexible, more usable, excel and improve the development process.

- periodical maintenance must fundamentally touch upon the development of the software's graphical interface aiming to make it User Friendly and not stacked with information. (Using only the methodology of information only shown when only needed).

- Availability of a community platform to receive comments, questions and responses, to increase user turnout and to effectively target and improve the development of VVS.

Sources

Websites related to VVS

- **Official** website for **Visin visual Scripting** software

  vvscodes.com

- **Official** software development website on **GitHub**:

  https://github.com/Sheriff99yt/Vision_Visual_Scripting

Online Sources (7 - 6 – 2022)

1- https://quran.com/al-baqarah/32
2- https://surahquran.org/english-aya-32-sora-2.html

3- eLearning Industry - By Stamatis Olika (May 25, 2017 - Primary Teacher) & Johnny Hamilton (May 11, 2016 - Online Instructional Designer at Providence Health & Services) & Ayesha Habeeb Omer (November 14, 2016 -) & Christopher Pappas (December 1, 2014 & May 15, 2016 - Founder) & Satyagraha Das (January 11, 2020 - Founder & CEO at Hexalen) & Dorian Peters (May 3, 2014 - designer and author who specializes in UX and Interface Design for learning and wellbeing)

4- Ecole Globule (Copyright 2022 - Ecole Globule | Schools in Dehradun)

5- ResearchGate - By (Journal of Education and Practice)www.iiste.org ISSN 2222-1735 (Paper) ISSN 2222-288X (Online) Vol.7, No.24, 2016) / Assessment of Adaptive PBL's Impact on HOT Development of Computer Science Students (Authors: Jamal Rainy) / The Impact of Alice on the Attitudes of High School Students Toward Computing (Authors: Eileen Peluso & Gene Screeching) / Engaging middle school teachers and students with alike in a diverse set of subjects (Authors: Susan H. Rodger & Jenna Hayes & Gatemen's Lezen & Henry Qin) / Evaluation of computer games developed by primary school children to gauge understanding of programming concepts (Authors: Amanda Ford & Amanda Ford & Thomas Connolly) / Expressing computer science concepts through Kodu game lab (Authors: Kathryn T. Stole & Teale Fristoe)

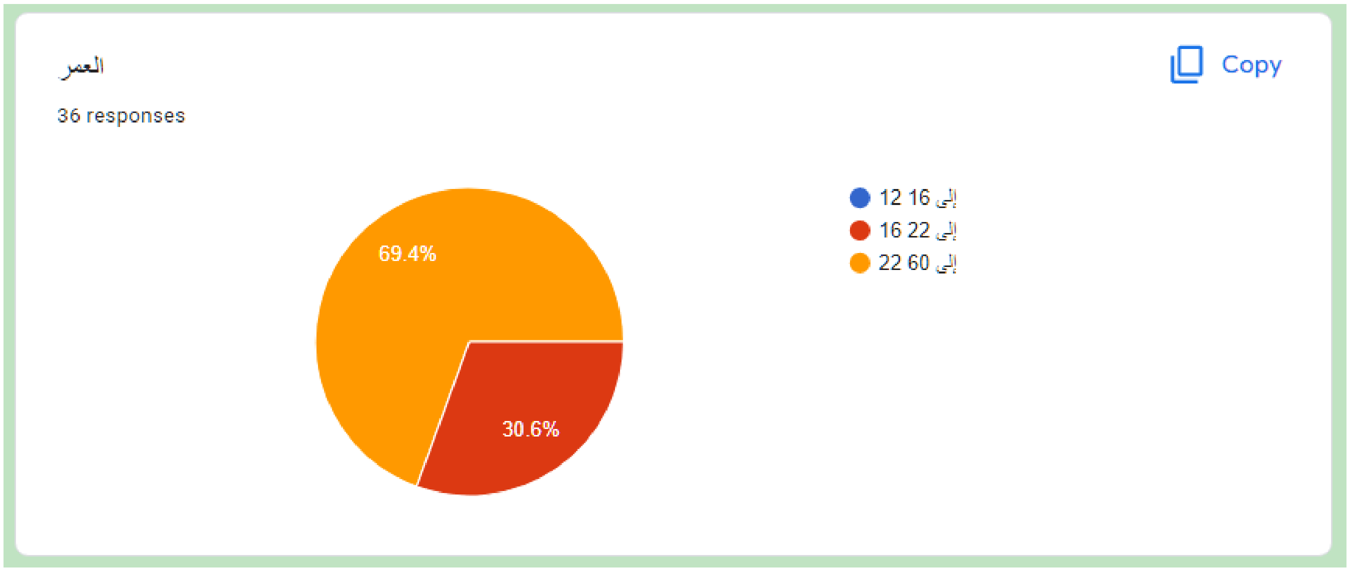
6- Benefits of Stimulus Congruency for Multisensory Facilitation of Visual Learning – By Robyn S. Kim, Aaron R. Seitz, Ladan Shams

Attachments

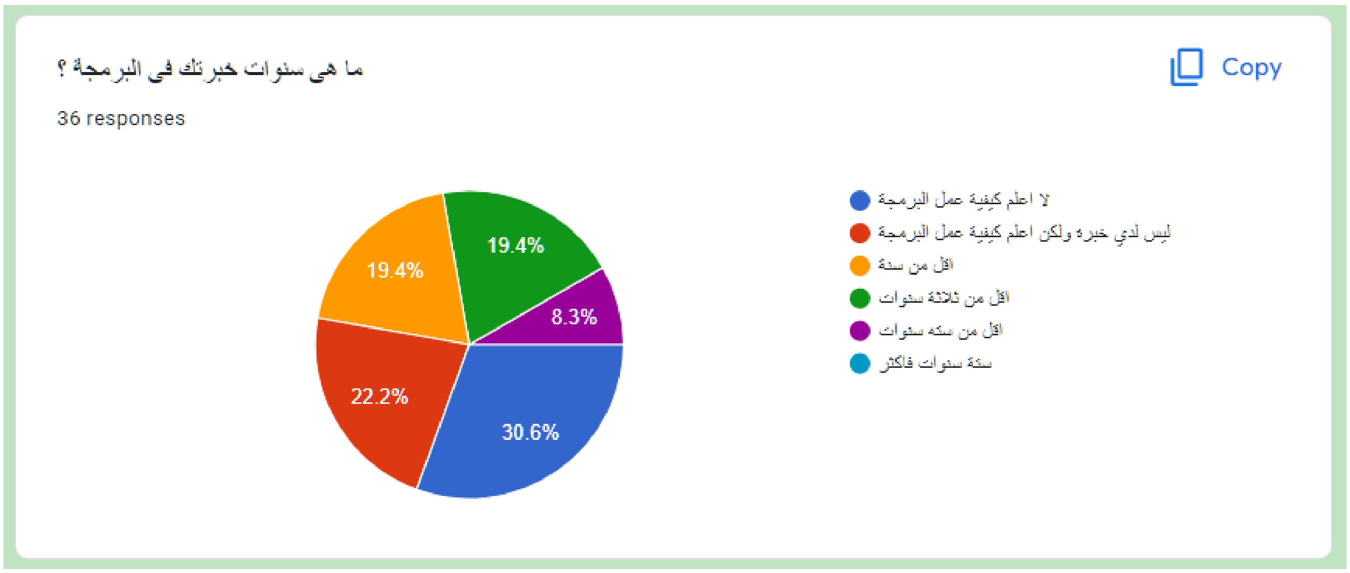
الملحق

Survey on the topic of visual programing

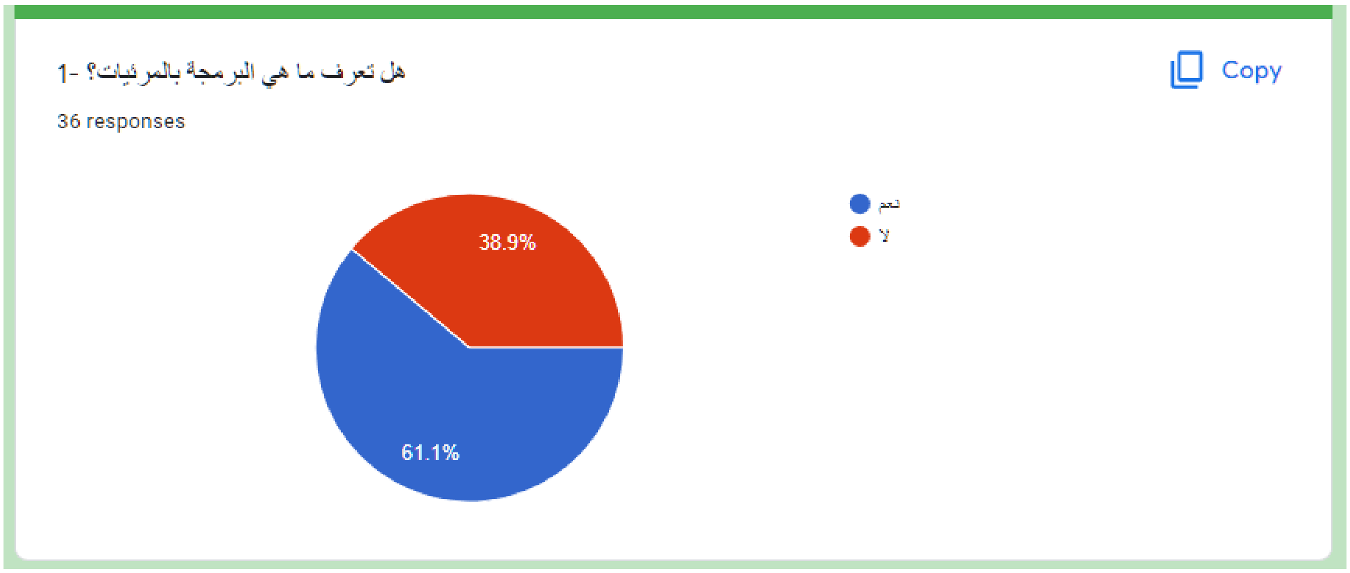
The survey made using Google Forms :



Age ratio

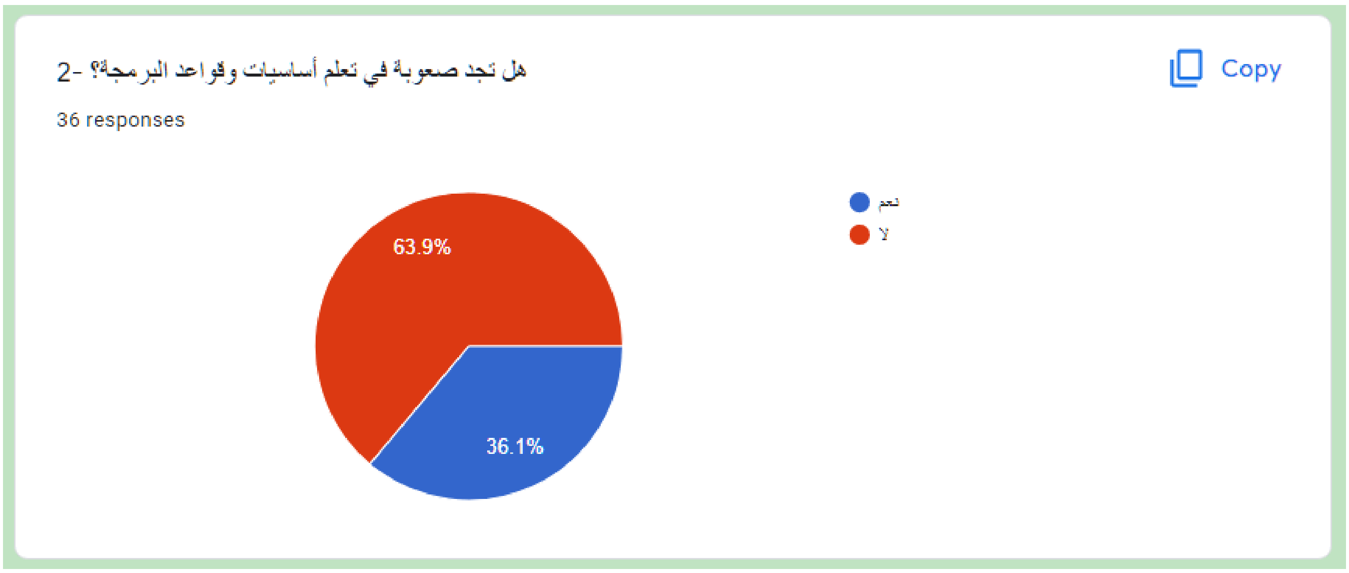


Years of experience


1- هل تعرف ما هي البرمجة بالمرئيات؟

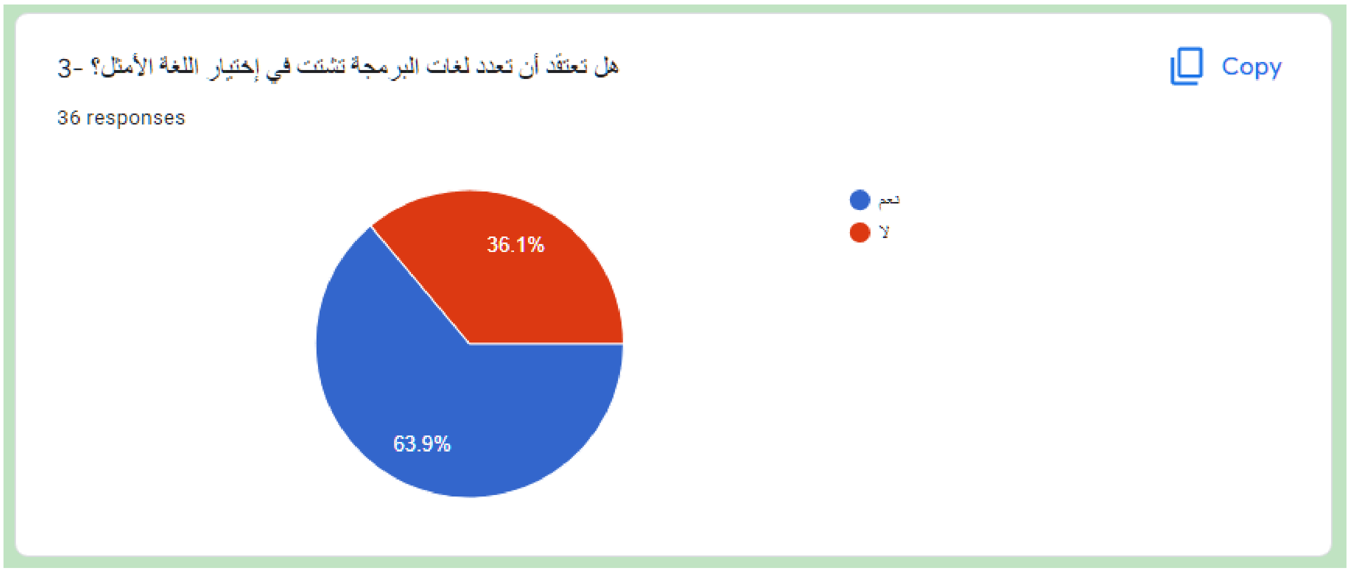
36 responses

Ratio of people who recognized visual programming


2- هل تجد صعوبة في تعلم أساسيات وقواعد البرمجة؟

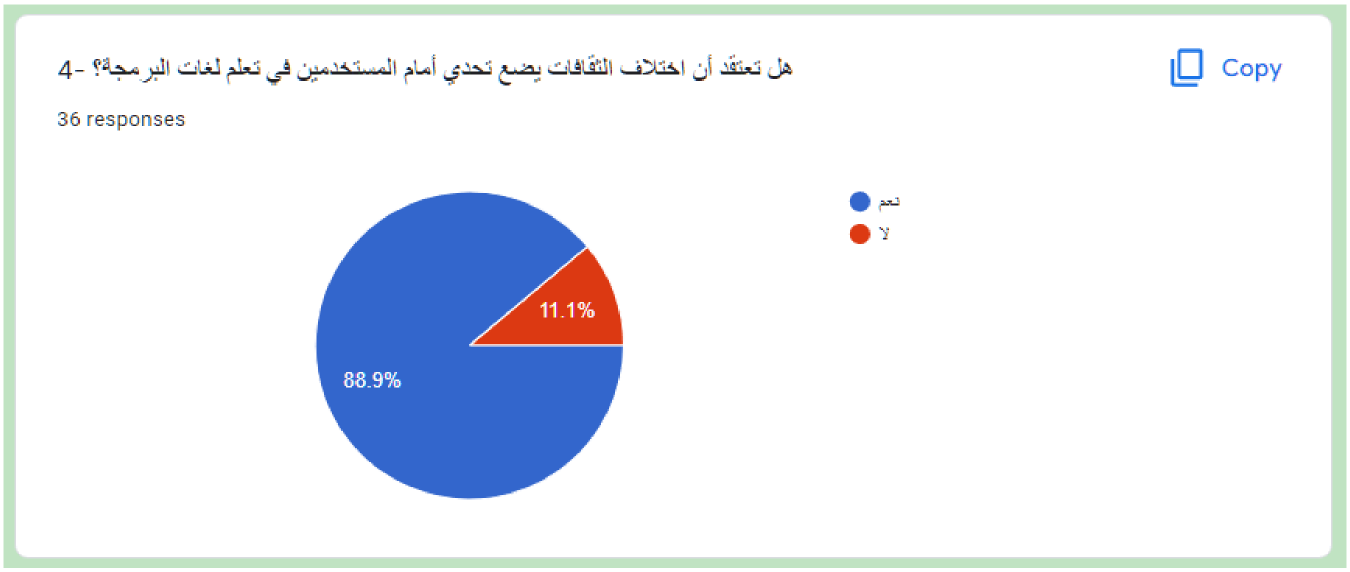
36 responses

Ratio of people that find difficulty in learning the foundation and principles of programming languages


3- هل تعتقد أن تعدد لغات البرمجة تشتت في إختيار اللغة الأمثل؟

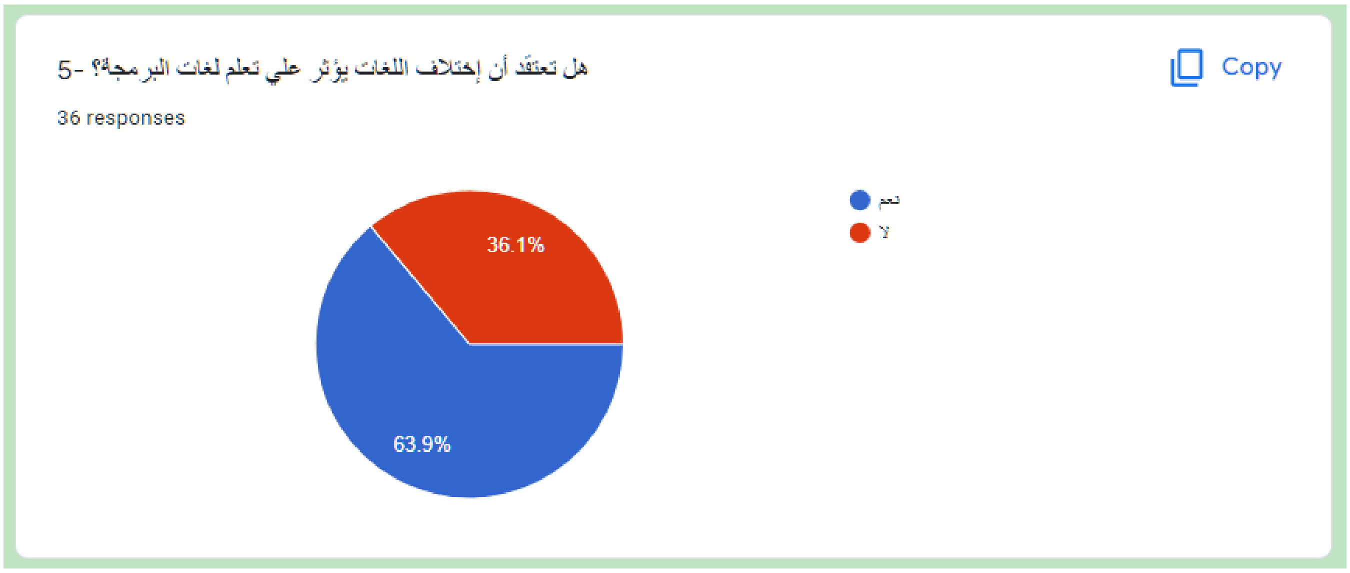
36 responses

Ratio of believing that the optimal code language selection is distracting

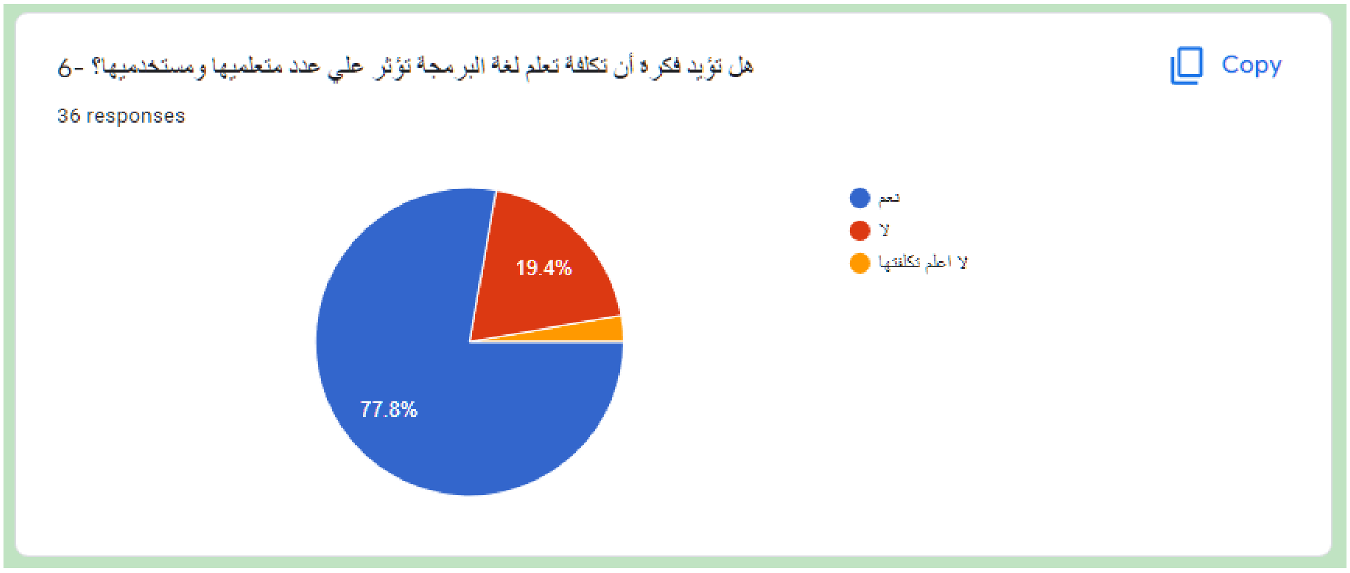

4- هل تعتقد أن اختلاف الثقافات يضع تحدي أمام المستخدمين في تعلم لغات البرمجة؟

36 responses

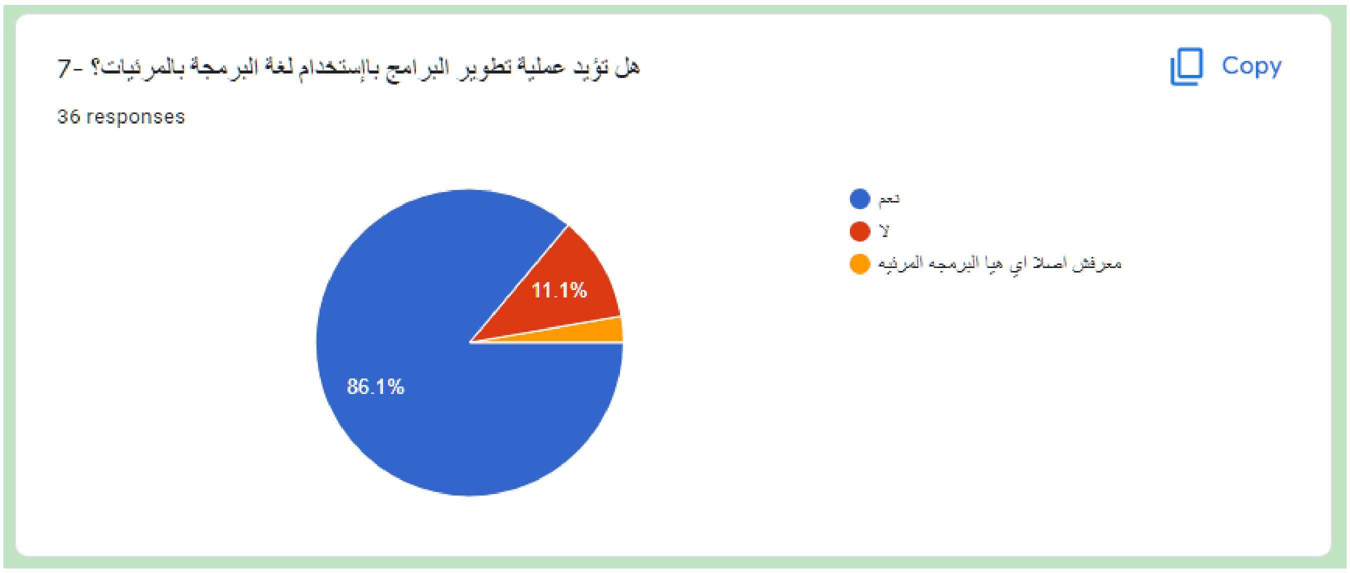
Ratio of believing different cultures poses a challenge in programming



Ratio of believing that programming languages are affected by the different mother languages



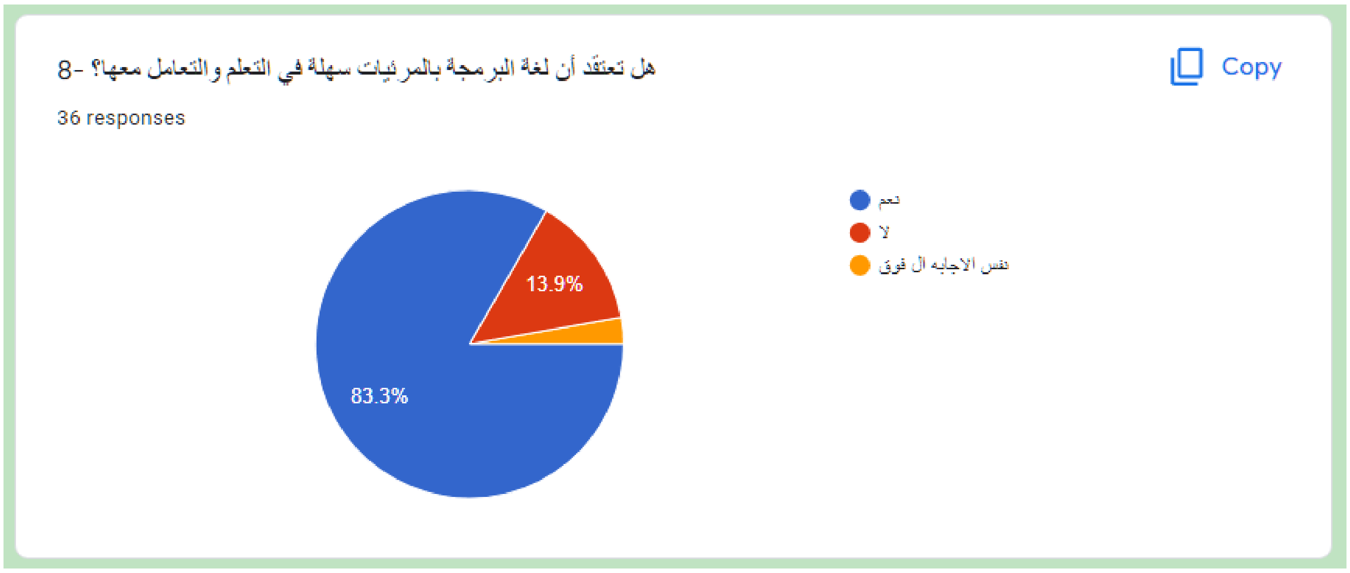
Ratio of believing that learning a programing language is expensive



7- هل تؤيد عملية تطوير البرامج بإستخدام لغة البرمجة بالمرئيات؟

36 responses

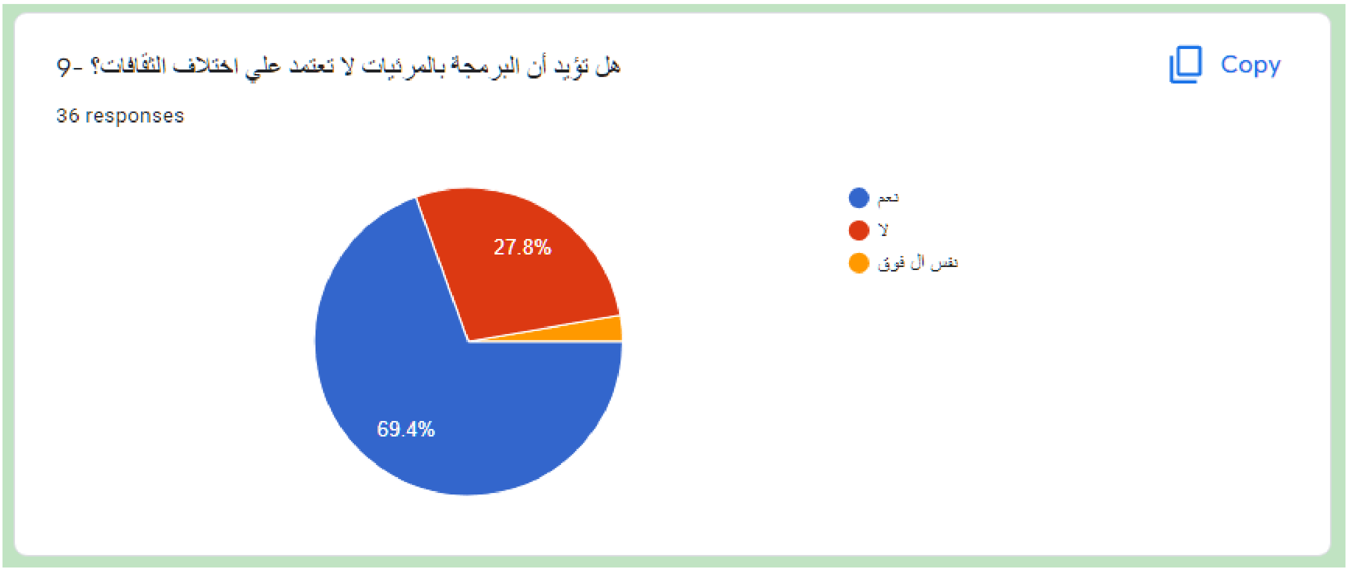
Ratio of people who are pro-software development using visual programming



8- هل تعتقد أن لغة البرمجة بالمرئيات سهلة في التعلم و التعامل معها؟

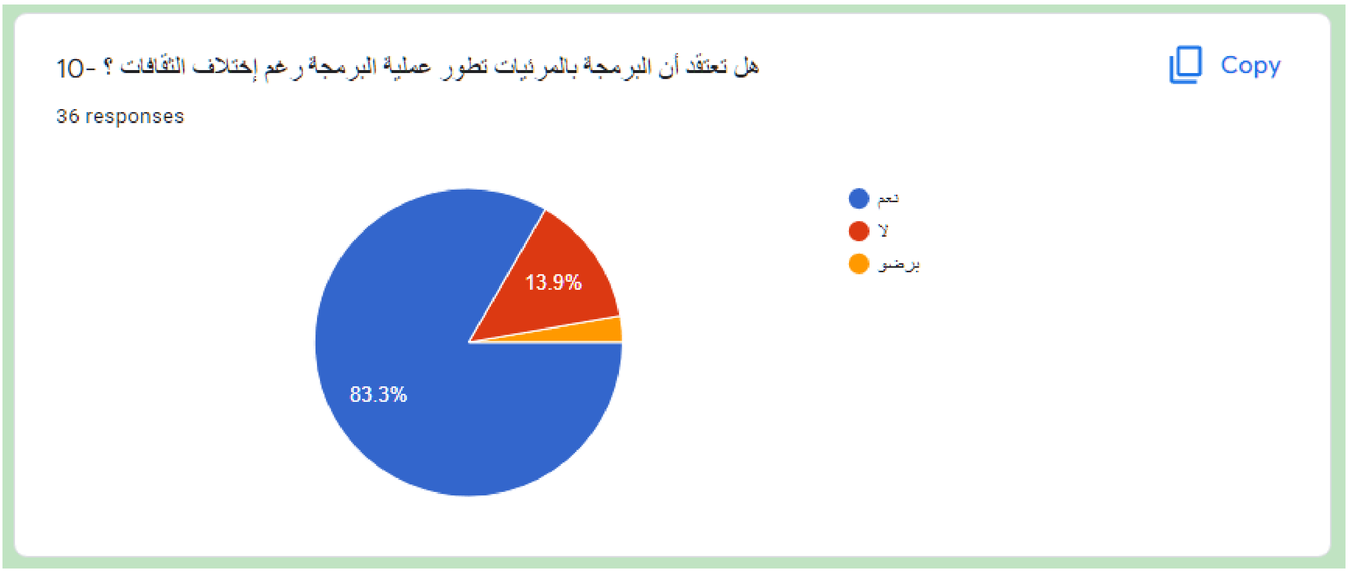
36 responses

Ratio of beliving that visual programming is easier on users


هل تؤيد أن البرمجة بالمرئيات لا تعتمد علي اختلاف الثقافات؟ -9

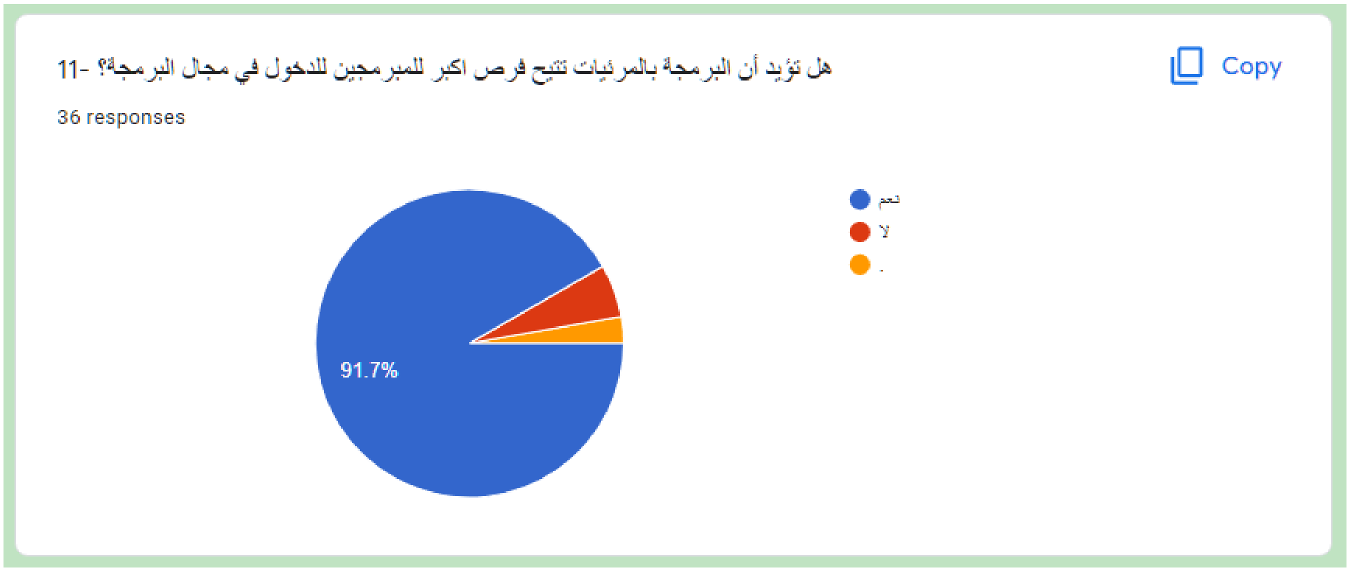
36 responses

Ratio of proponents visual programming is not dependence on different cultures


هل تعتقد أن البرمجة بالمرئيات تطور عملية البرمجة رغم إختلاف الثقافات ؟ -10

36 responses

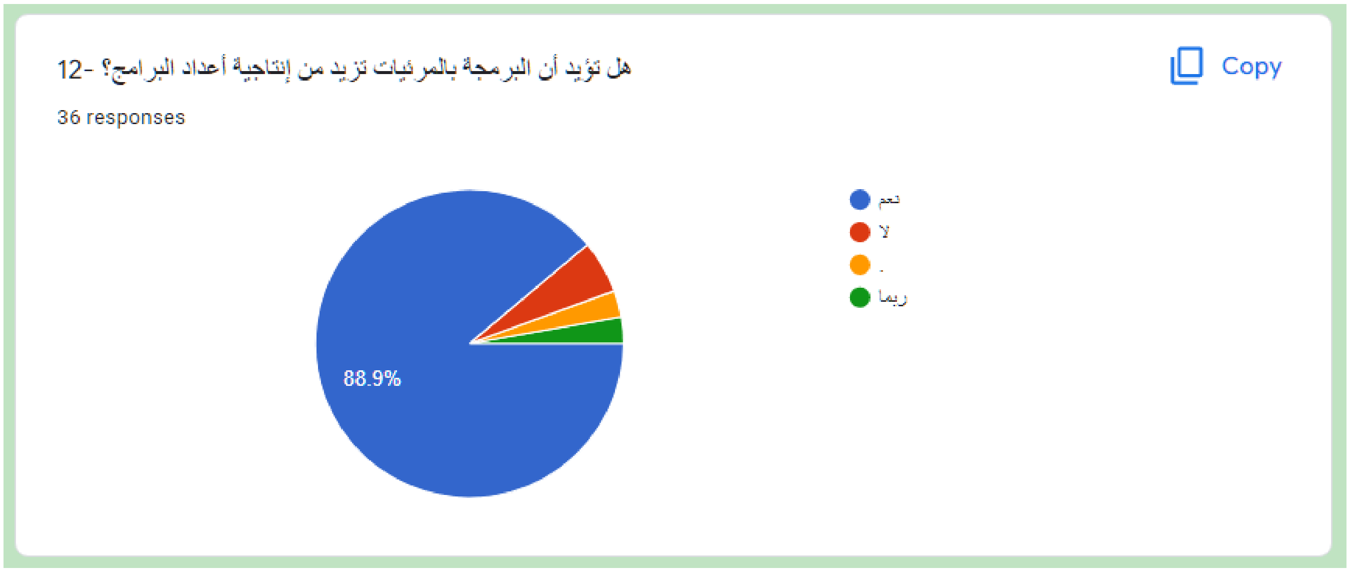
Ratio of believing that visual programming evolves the programming prosses despite different cultures



11- هل تؤيد أن البرمجة بالمرئيات تتيح فرص اكبر للمبرمجين للدخول في مجال البرمجة؟

36 responses

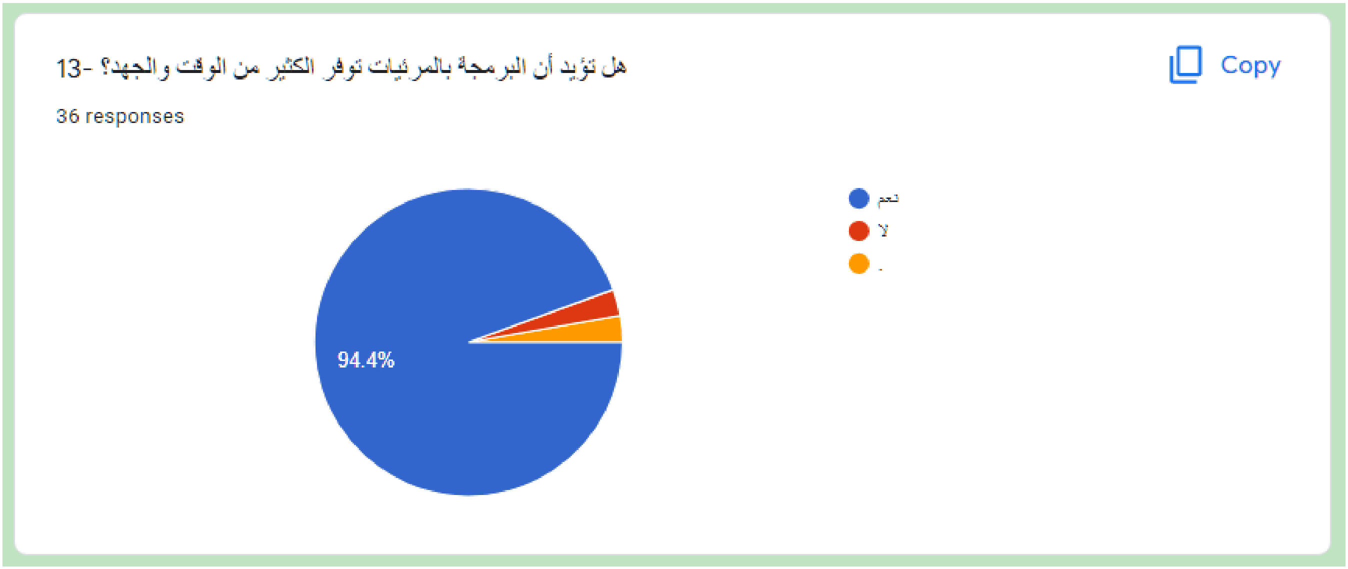
Ratio of believing that visual programming allows for bigger opportunities for entering the programing field



12- هل تؤيد أن البرمجة بالمرئيات تزيد من إنتاجية أعداد البرامج؟

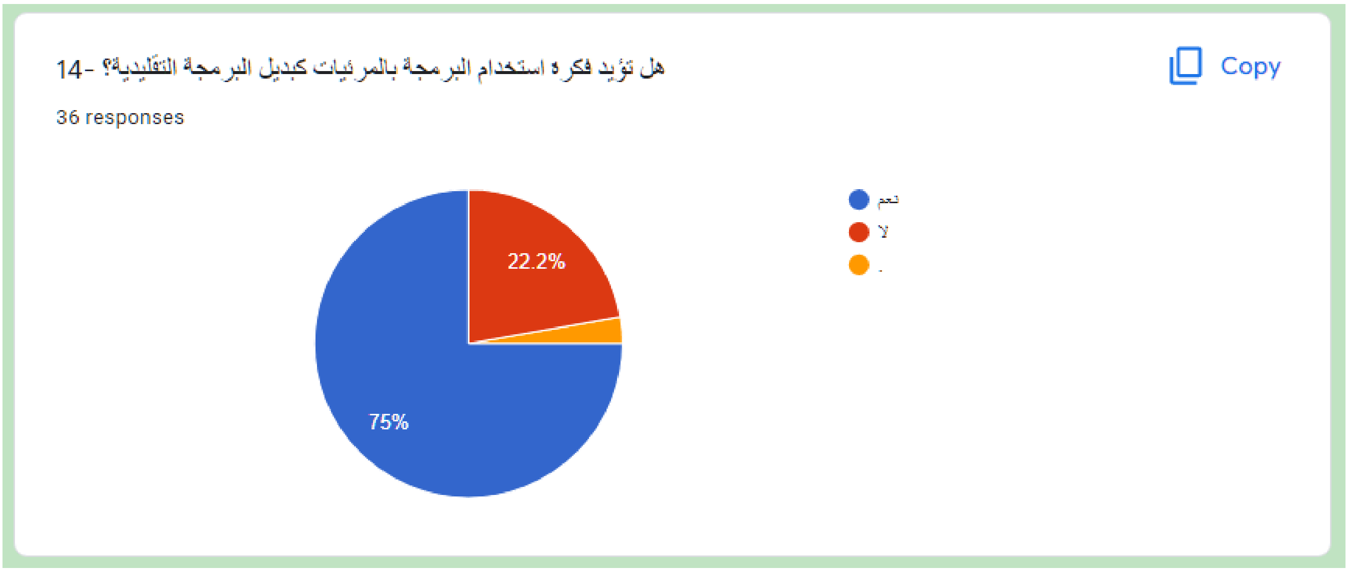
36 responses

Ratio of proponents that visual programming increases the productivity of programming


13- هل تؤيد أن البرمجة بالمرئيات توفر الكثير من الوقت والجهد؟

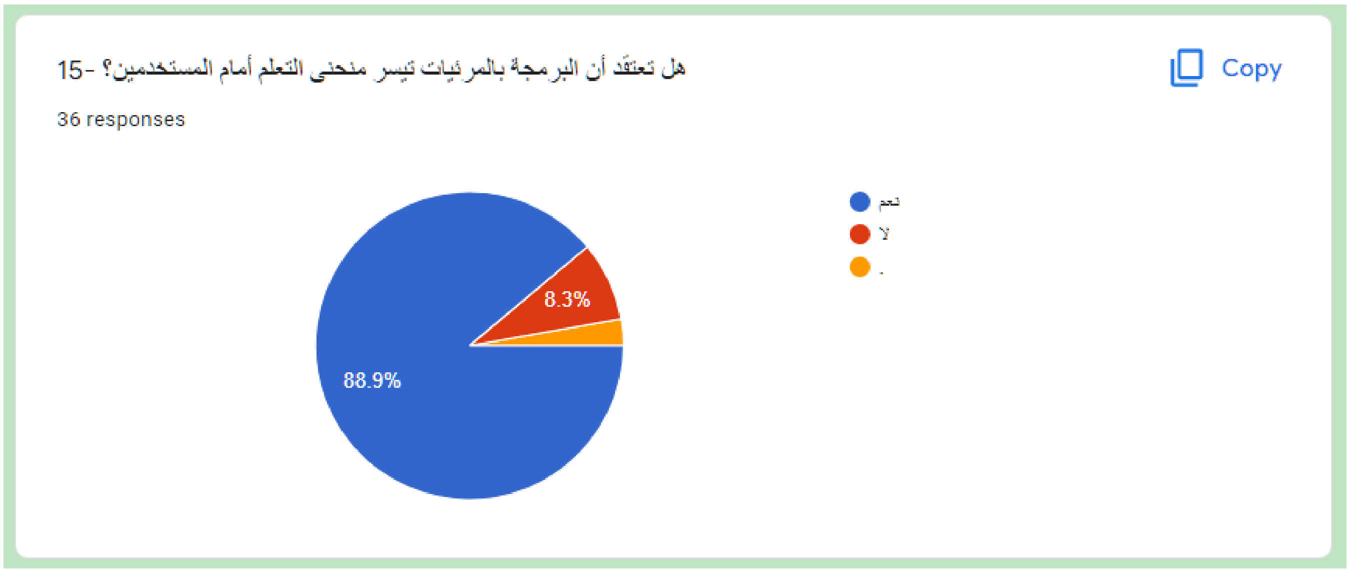
36 responses

Ratio of how much visual programing saves time and effort

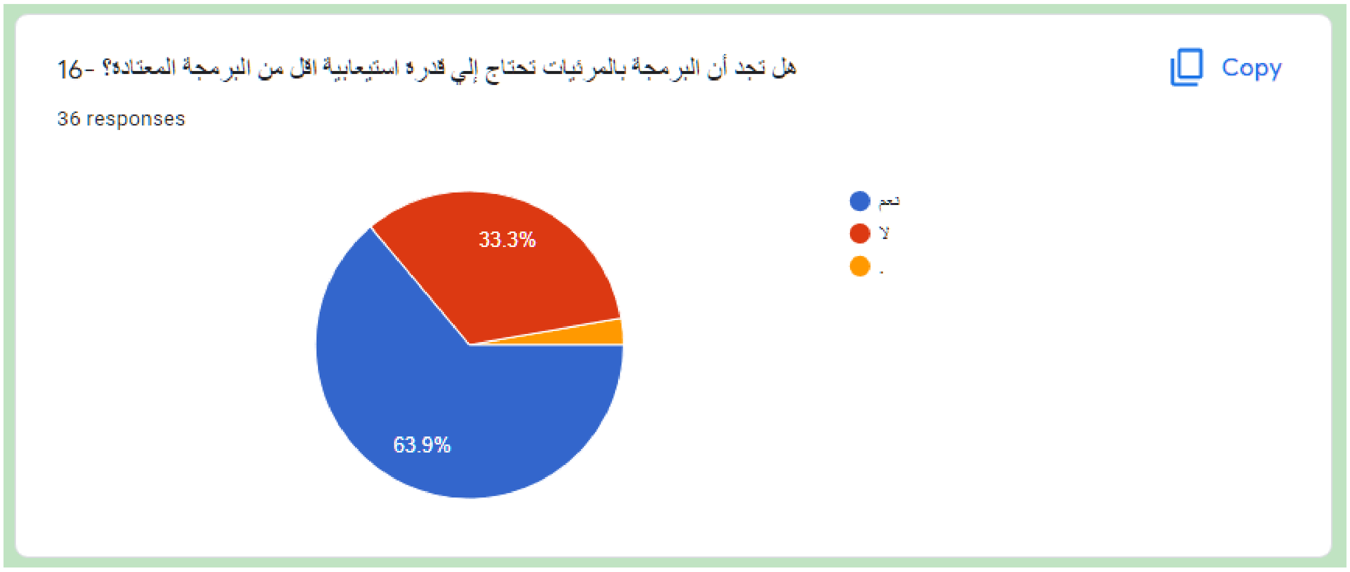

14- هل تؤيد فكرة استخدام البرمجة بالمرئيات كبديل البرمجة التقليدية؟

36 responses

Ratio of proponents of using visual programming as an alternative to traditional programming

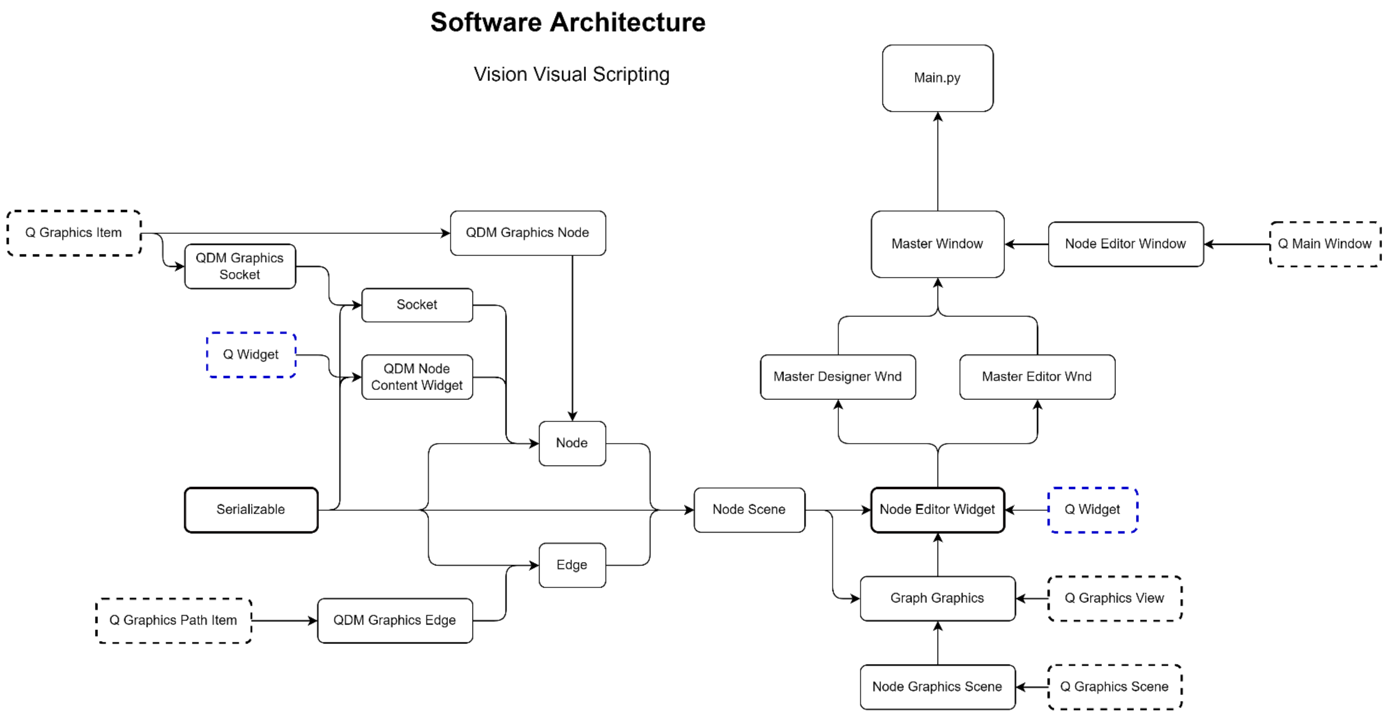


Ratio of proponents that visual programming dampens the learning curve



Ratio of reponders thinkink that visual programming needs less absorptive capacity

Software Architecture



Codes

Main.py

```
import ctypes
import os
import sys
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *
from qtpy.QtWidgets import QApplication
from vvs_app.master_window import MasterWindow, Splash

sys.path.insert(0, os.path.join(os.path.dirname(__file__), "", ".."))
if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setStyle('Fusion')
    app.setWindowIcon(QIcon("icons/Dark/VVS_Logo_Thick.png"))

    # Show app Icon In Task Manager
    myappid = 'mycompany.myproduct.subproduct.version'

ctypes.windll.shell32.SetCurrentProcessExplicitAppUserModelID(myappid)

    splash = Splash()
    wnd = MasterWindow()
    splash.run(wnd)
    sys.exit(app.exec_())
```

MasterWindow

```
import os
import subprocess
import sys

from qtpy.QtGui import *
from qtpy.QtWidgets import *
from qtpy.QtCore import *

from nodeeditor.utils import loadStylesheets
from nodeeditor.node_editor_window import NodeEditorWindow
from vvs_app.editor_settings_wnd import SettingsWidget
from vvs_app.master_editor_wnd import NodeEditorTab
from vvs_app.master_designer_wnd import MasterDesignerWnd
from vvs_app.editor_node_list import NodeList
from vvs_app.editor_files_wdg import FilesWDG
from vvs_app.editor_user_nodes_list import UserNodesList
from vvs_app.editor_properties_list import PropertiesList
from vvs_app.global_switches import *

from nodeeditor.utils import dumpException
# from vvs_app.nodes_configuration import FUNCTIONS

# Enabling edge validators
from nodeeditor.node_edge import Edge
from nodeeditor.node_edge_validators import (
    edge_cannot_connect_two_outputs_or_two_inputs,
    edge_cannot_connect_input_and_output_of_same_node
)

# Edge.registerEdgeValidator(edge_validator_debug)
from vvs_app.master_node import MasterNode
from vvs_app.nodes.nodes_configuration import register_Node

Edge.registerEdgeValidator(edge_cannot_connect_two_outputs_or_two_inputs
)
Edge.registerEdgeValidator(edge_cannot_connect_input_and_output_of_same_
node)

# images for the dark skin

DEBUG = False

class Splash(QWidget):
    def __init__(self):
        super().__init__()
        self.setStyleSheet("background-color: transparent")
        self.setAttribute(Qt.WA_TranslucentBackground, on=True)
        self.setWindowFlag(Qt.FramelessWindowHint)

        lo = QVBoxLayout()
        self.setLayout(lo)
        Logo = QLabel()
        pixmap = QPixmap("icons/Dark/VVS_White2.png")
        Logo.setPixmap(pixmap)
        lo.addWidget(Logo)

        self.Loading_Label = QLabel("Loading")
        self.Loading_Label.setStyleSheet("font: 20px; color: white") #
font-family: Calibri;
        lo.addWidget(self.Loading_Label)

        self.timer = QTimer()

    def mousePressEvent(self, event):
        self.oldPosition = event.globalPos()

    def mouseMoveEvent(self, event):
        delta = QPoint(event.globalPos() - self.oldPosition)
        self.move(self.x() + delta.x(), self.y() + delta.y())
        self.oldPosition = event.globalPos()

    def run(self, Main):
        self.show()
        self.timer.start(300)
        self.times = 0
        self.timer.timeout.connect(lambda: self.run_timeout(Main))

    def run_timeout(self, Main):
        if self.times >= 3:
            Main.showMaximized()
            self.close()
            self.timer.stop()
        else:
            self.times += 1
            self.Loading_Label.setText(self.Loading_Label.text() + " .")

class MasterWindow(NodeEditorWindow):
    def __init__(self):
        super().__init__()

        self.initUI()

    def initUI(self):
        # self.qss_theme = "qss/nodeeditor-light.qss"
        self.qss_theme =
self.global_switches.themes[self.global_switches.switches_Dict["Appearan
ce"]["Theme"][0]] # ["Theme"][0]

        self.stylesheet_filename =
os.path.join(os.path.dirname(__file__), self.qss_theme)

        loadStylesheets(
            os.path.join(os.path.dirname(__file__), self.qss_theme),
self.stylesheet_filename)
```

```
        self.empty_icon = QIcon(".")

        if DEBUG: print("Registered nodes:")

        self.stackedDisplay = QStackedWidget()

        self.graphs_parent_wdg = QMdiArea()

        self.CreateLibraryWnd()

        # Create Node Designer Window
        self.node_designer = MasterDesignerWnd(self)

        self.stackedDisplay.addWidget(self.graphs_parent_wdg)

        self.stackedDisplay.addWidget(self.node_designer)


self.graphs_parent_wdg.setHorizontalScrollBarPolicy(Qt.ScrollBarAsNeeded
)

self.graphs_parent_wdg.setVerticalScrollBarPolicy(Qt.ScrollBarAsNeeded)
        self.graphs_parent_wdg.setTabsClosable(True)
        self.graphs_parent_wdg.setTabsMovable(True)

self.graphs_parent_wdg.subWindowActivated.connect(self.active_graph_swit
ched)

        self.setCentralWidget(self.stackedDisplay)

        self.windowMapper = QSignalMapper(self)

self.windowMapper.mapped[QWidget].connect(self.setActiveSubWindow)

        # Create Welcome Screen and allow user to set the project
Directory
        self.create_welcome_screen()

        # Create Nodes List
        self.create_functions_dock()

        # Create Files Dock
        self.create_files_dock()

        # Create Details List Window
        self.create_properties_dock()

        # Create Variable List
        self.create_user_nodes_dock()

        self.createActions()
        self.create_menus()
        self.createStatusBar()
        self.update_menus()
        self.readSettings()

        self.CreateToolBar()

        self.setWindowTitle("Vision Visual Scripting")
        self.update_libraries_wnd()
        self.library_menu.setEnabled(False)
        self.node_designer_menu.setEnabled(False)

        self.set_actions_shortcuts()

    def create_welcome_screen(self):
        Elayout = QVBoxLayout()
        Elayout.setAlignment(Qt.AlignCenter)
        Elayout.setSpacing(20)

        self.empty_screen = QWidget()
        self.empty_screen.setLayout(Elayout)

        user_text = QLabel("Select Your Project Directory...")
        user_text.setFont(QFont("Roboto", 14))
        w_image =
QPixmap(f"icons/{self.global_switches.switches_Dict['Appearance']['Theme
'][0]}/VVS_White1.png")

        welcome_image = QLabel()
        welcome_image.setPixmap(w_image)
        welcome_image.setScaledContents(True)

        self.brows_btn = QPushButton("Brows..")

        Elayout.addWidget(welcome_image)
        Elayout.addItem(QSpacerItem(120, 120))
        Elayout.addWidget(user_text)
        Elayout.addWidget(self.brows_btn)

        self.stackedDisplay.addWidget(self.empty_screen)
        self.switch_display(Welcome=True)

    def CreateOfflineDir(self):
        self.Offline_Dir =
f"C:/Users/{os.getlogin()}/AppData/Roaming/VVS/Offline Library"
        if os.path.exists(self.Offline_Dir):
            pass
        else:
            self.Offline_Dir = os.makedirs(os.getenv('AppData') +
"/VVS/Offline Library")


self.library_offline_list.setRootIndex(self.Model.index(self.Offline_Dir
))

    def CreateLibraryWnd(self):
        self.librariesDock = QDockWidget("Libraries")
        self.library_subwnd = QTabWidget()

        self.librariesDock.setWidget(self.library_subwnd)

self.librariesDock.setFeatures(self.librariesDock.DockWidgetMovable)
        self.addDockWidget(Qt.RightDockWidgetArea, self.librariesDock)

        offline_Vlayout = QVBoxLayout()
        offline_Vlayout.setContentsMargins(0, 0, 0, 0)

        self.Model = QFileSystemModel()
        self.Model.setRootPath("")

        self.library_offline_list = QTreeView()

self.library_offline_list.setSelectionMode(QAbstractItemView.ExtendedSel
ection)
        self.library_offline_list.setModel(self.Model)
        self.library_offline_list.setSortingEnabled(True)
        self.library_offline_list.setColumnWidth(0, 130)
        self.library_offline_list.sortByColumn(0, Qt.AscendingOrder)
        self.library_offline_list.hideColumn(1)
        self.library_offline_list.hideColumn(2)
        self.library_offline_list.setStyleSheet("color: white")

self.library_offline_list.setSizePolicy(QSizePolicy.MinimumExpanding,
QSizePolicy.MinimumExpanding)

        offline_Vlayout.addWidget(self.library_offline_list)

        self.library_online_list = QListWidget()

        topVlayout = QVBoxLayout()
        search_bar_layout = QHBoxLayout()

        self.search_line_edit = QLineEdit()
        self.search_btn = QPushButton()

        search_bar_layout.addWidget(self.search_line_edit)
        search_bar_layout.addWidget(self.search_btn)

        self.search_btn.setMaximumSize(30, 30)
        self.search_btn.setIcon(QIcon("icons/Light/search.png"))
        self.search_line_edit.setMinimumHeight(30)

        topVlayout.addLayout(search_bar_layout)
        topVlayout.addWidget(self.library_online_list)

        online_widget = QWidget()
        online_widget.setLayout(topVlayout)

        offline_widget = QWidget()
        offline_widget.setLayout(offline_Vlayout)
        self.library_subwnd.addTab(offline_widget, "    Offline    ")
        self.library_subwnd.addTab(online_widget, "    Online    ")

        self.CreateOfflineDir()

self.library_offline_list.clicked.connect(self.ViewSelectedFiles)

    def ViewSelectedFiles(self):
        all_files = []

        selected_files = self.library_offline_list.selectedIndexes()

        for file_name in selected_files:
            file_path = QFileSystemModel().filePath(file_name)

            if file_path.endswith(".json"):
                if not all_files.__contains__(file_path):
                    all_files.append(file_path)
                    # print(all_files)

        self.on_file_open(all_files)

    def active_graph_switched(self):
        self.update_menus()
        if self.currentNodeEditor():

self.VEStackedWdg.setCurrentWidget(self.currentNodeEditor().scene.user_n
odes_wdg)

    def switch_display(self, Welcome=False, Editor=False,
Designer=False, Library=False):
        # Use the Argument To Force Activate the Specified Window

        if Editor:
            self.stackedDisplay.setCurrentIndex(0)
            self.library_btn.setChecked(False)
            self.node_editor_btn.setChecked(True)
            self.node_designer_btn.setChecked(False)
            return

        if Library:
            self.stackedDisplay.setCurrentIndex(0)
            self.library_btn.setChecked(True)
            self.node_editor_btn.setChecked(False)
            self.node_designer_btn.setChecked(False)
            return

        elif Designer:
            self.stackedDisplay.setCurrentIndex(1)
            self.library_btn.setChecked(False)
            self.node_editor_btn.setChecked(False)
            self.node_designer_btn.setChecked(True)
            self.node_editor_menu.setEnabled(False)
            self.library_menu.setEnabled(False)

            self.toolbar_library.setChecked(False)

self.librariesDock.setVisible(self.toolbar_library.isChecked())

            return

        elif Welcome:
            self.stackedDisplay.setCurrentIndex(2)
            return

    def CreateToolBar(self):
        # Create Tools self.tools_bar
        self.tools_bar = QToolBar("Tools", self)
        self.tools_bar.setIconSize(QSize(20, 20))
        self.tools_bar.setFloatable(False)

        # Add self.tools_bar To Main Window
        self.addToolBar(self.tools_bar)

        # Add and connect self.settingsBtn
        self.settingsBtn =
QAction(QIcon(self.global_switches.get_icon("settings.png")), "&Open
Settings Window", self)
        self.settingsBtn.setIconText("settings.png")
        self.settingsBtn.setCheckable(True)
        self.settingsBtn.triggered.connect(self.onSettingsOpen)

self.settingsBtn.setShortcut(QKeySequence(self.global_switches.switches_
Dict["Key Mapping"]["Settings Window"]))
        self.tools_bar.addAction(self.settingsBtn)
        self.actions_creation_dict["UI"]["Settings Window"] =
[self.settingsBtn]

        # Add Separator
        self.tools_bar.addSeparator()

        self.node_editor_btn =
QAction(QIcon(self.global_switches.get_icon("edit.png")), "&Node
Editor", self)
        self.node_editor_btn.setIconText("edit.png")
        self.node_editor_btn.setCheckable(True)

self.node_editor_btn.triggered.connect(self.activate_editor_mode)
        self.tools_bar.addAction(self.node_editor_btn)
        self.actions_creation_dict["UI"]["Node Editor Window"] =
[self.node_editor_btn]

        # Add and connect self.node_designer_btn
        self.node_designer_btn =
QAction(QIcon(self.global_switches.get_icon("node design.png")), "&Node
Designer", self)
```

```
        self.node_designer_btn.setIconText("node design.png")
        self.node_designer_btn.setEnabled(False)
        self.node_designer_btn.setCheckable(True)

self.node_designer_btn.triggered.connect(self.activate_designer_mode)
        self.tools_bar.addAction(self.node_designer_btn)
        self.actions_creation_dict["UI"]["Node Designer Window"] =
[self.node_designer_btn]

        # Add and connect self.library_btn
        self.library_btn =
QAction(QIcon(self.global_switches.get_icon("library.png")), "&Library",
self)
        self.library_btn.setIconText("library.png")
        self.library_btn.setCheckable(True)
        self.library_btn.triggered.connect(self.activate_library_mode)
        self.library_btn.setShortcut(QKeySequence("`"))
        self.tools_bar.addAction(self.library_btn)
        self.actions_creation_dict["UI"]["Library Window"] =
[self.library_btn]

        # Add Separator
        self.tools_bar.addSeparator()

    def onSettingsOpen(self):
        if self.__dict__.__contains__("settingsWidget"):
            if self.settingsWidget.isHidden():
                self.settingsWidget.show()
                self.settingsBtn.setChecked(True)
            else:
                self.settingsWidget.hide()
        else:
            self.settingsWidget = SettingsWidget(masterRef=self)


self.global_switches.update_font_size(self.global_switches.switches_Dict
["Appearance"]["Font Size"])

            self.settingsWidget.show()
            self.settingsBtn.setChecked(True)

            self.settingsWidget.setWindowTitle("Settings")
            self.settingsWidget.setGeometry(300, 150, 500, 500)

    def closeEvent(self, event):
        self.graphs_parent_wdg.closeAllSubWindows()
        if self.graphs_parent_wdg.currentSubWindow():
            event.ignore()
        else:
            self.writeSettings()
            event.accept()

            # hacky fix for PyQt 5.14.x
            import sys
            sys.exit(0)

    def createActions(self):
        self.actions_creation_dict = \
            {
                "File Menu":
                    {
                        "New Graph": [None, "Create new graph",
self.on_new_graph_tab, '&New Graph'],
                        "addSeparator 1": [],
                        "Open": [None, "Open file", self.on_file_open,
'&Open'],
                        "Set Project Location": [None, "Set a Folder For
Your Project", self.files_widget.set_project_folder, '&Set Project
Location'],
                        "Save": [None, "Save file", self.onFileSave,
'&Save'],
                        "Save As": [None, "Save file as...",
self.on_file_save_as, 'Save &As...'],
                        "addSeparator 2": [],
                        "Exit": [None, "Exit application", self.close,
'E&xit']
                    }
                ,
                "Edit Menu":
                    {
                        "Undo": [None, "Undo last operation",
self.onEditUndo, '&Undo'],
                        "Redo": [None, "Redo last operation",
self.onEditRedo, "&Redo"],
                        "addSeparator 1": [],
                        "Select All": [None, "Select's All Nodes",
self.selectAllNodes, 'Select&All'],
                        "Cut": [None, "Cut to clipboard",
self.onEditCut, 'Cu&t'],
                        "Copy": [None, "Copy to clipboard",
self.onEditCopy, '&Copy'],
                        "Paste": [None, "Paste from clipboard",
self.onEditPaste, '&Paste'],
                        "addSeparator 2": [],
                        "Delete": [None, "Delete selected items",
self.onEditDelete, "&Delete"]
                    }
                ,
                "Node Editor Menu":
                    {
                        "Close": [None, "Close the active window",
self.graphs_parent_wdg.closeActiveSubWindow, "Cl&ose"],
                        "Close All": [None, "Close all the windows",
self.graphs_parent_wdg.closeAllSubWindows, "Close &All"],
                        "addSeparator 2": [],
                        "Tile": [None, "Tile the windows",
self.graphs_parent_wdg.tileSubWindows, "&Tile"],
                        "addSeparator 3": [],
                        "Next": [None, "Move the focus to the next
window", self.graphs_parent_wdg.activateNextSubWindow, "Ne&xt"],
                        "Previous": [None, "Move the focus to the
previous window", self.graphs_parent_wdg.activatePreviousSubWindow,
"Pre&vious"]
                    }
                ,
                "Help":
                    {
                        "About": [None, "Show the application's About
box", self.about, "&About"],
                        "Doc": [None, "Program Documentation",
self.open_doc, "&Documentation"]
                    }
                ,
                "UI":
                    {}
            }
        self.actSeparator = QAction(self)
        self.actSeparator.setSeparator(True)

    def set_actions_shortcuts(self):
        shortcuts = self.global_switches.switches_Dict["Key Mapping"]
        for menu in self.actions_creation_dict:
            for act in self.actions_creation_dict[menu]:
                menu_vals = self.actions_creation_dict[menu]
                if not act.__contains__("addSeparator"):
                    if shortcuts.__contains__(act):
                        menu_vals[act][0].setShortcut(shortcuts[act])

    def open_doc(self):
        subprocess.Popen('hh.exe "VVS-Help.chm"')

    def currentNodeEditor(self):
        """ we're returning NodeEditorWidget here... """
        activeSubWindow = self.graphs_parent_wdg.activeSubWindow()

        if activeSubWindow:
            return activeSubWindow.widget()
        else:
            return None

    def on_new_graph_tab(self):
        # Overrides Node Editor Window > actNew action
        try:
            subwnd = self.new_graph_tab()

            all_names = []
            for item in self.graphs_parent_wdg.subWindowList():
                all_names.append(item.widget().windowTitle())

            self.files_widget.new_graph_name(subwnd, all_names)

        except Exception as e:
            dumpException(e)

    def on_file_open(self, all_files=False):

        if all_files == False:
            file_names, filter = QFileDialog.getOpenFileNames(self,
'Open graph from file',

self.files_widget.Project_Directory,

self.getFileDialogFilter())

        else:
            file_names = all_files

        try:
            for file_name in file_names:
                if file_name:
                    if self.findMdiChild(file_name):
                        subwnd = self.findMdiChild(file_name)

self.graphs_parent_wdg.setActiveSubWindow(subwnd)

                    else:
                        # We need to create new subWindow and open the
file

                        subwnd = self.new_graph_tab()
                        node_editor = subwnd.widget()

                        if node_editor.fileLoad(file_name):
                            self.statusBar().showMessage("File %s
loaded" % file_name, 5000)

node_editor.setWindowTitle(os.path.splitext(os.path.basename(file_name))
[0])
                        else:
                            node_editor.close()

        except Exception as e:
            dumpException(e)

    def create_menus(self):
        super().create_menus()

        self.node_editor_menu = self.menuBar().addMenu("&Node Editor")

        self.library_menu = self.menuBar().addMenu("&Library")

        self.node_designer_menu = self.menuBar().addMenu("&Node
Designer")

        self.update_window_menu()

        self.helpMenu = self.menuBar().addMenu("&Help")
        for i in self.actions_creation_dict["Help"]:
            if i.__contains__("addSeparator"):
                self.helpMenu.addSeparator()
            else:
                mylist = self.actions_creation_dict["Help"][i]
                act = QAction(mylist[3], parent=self,
statusTip=mylist[1], triggered=mylist[2])
                self.helpMenu.addAction(act)
                self.actions_creation_dict["Help"][i][0] = act

        self.editMenu.aboutToShow.connect(self.update_edit_menu)

    def update_menus(self):
        active = self.currentNodeEditor()
        hasMdiChild = (active is not None)

        Switchs = {"File Menu": ["Save", "Save As"], "Edit Menu":
["Paste", "Select All"], "Node Editor Menu": ["Close", "Close All",
"Tile", "Next", "Previous"]}
        for menu in Switchs:
            for act_name in Switchs[menu]:

self.actions_creation_dict[menu][act_name][0].setEnabled(hasMdiChild)

        # Update Edit Menu
        self.update_edit_menu()

    def update_edit_menu(self):
        try:
            active = self.currentNodeEditor()
            hasMdiChild = (active is not None)

            self.actions_creation_dict["Edit
Menu"]["Cut"][0].setEnabled(hasMdiChild and active.hasSelectedItems())
            self.actions_creation_dict["Edit
Menu"]["Copy"][0].setEnabled(hasMdiChild and active.hasSelectedItems())
            self.actions_creation_dict["Edit
Menu"]["Delete"][0].setEnabled(hasMdiChild and
active.hasSelectedItems())
            self.actions_creation_dict["Edit
Menu"]["Undo"][0].setEnabled(hasMdiChild and active.canUndo())
            self.actions_creation_dict["Edit
Menu"]["Redo"][0].setEnabled(hasMdiChild and active.canRedo())

        except Exception as e:
            dumpException(e)

    def update_window_menu(self):
        self.toolbar_library = self.library_menu.addAction("Libraries
Window")
        self.toolbar_library.setCheckable(True)

self.toolbar_library.triggered.connect(self.update_libraries_wnd)
```

```
        self.toolbar_library.setChecked(False)

        self.toolbar_properties =
self.node_editor_menu.addAction("Properties Window")
        self.toolbar_properties.setCheckable(True)

self.toolbar_properties.triggered.connect(self.update_properties_wnd)
        self.toolbar_properties.setChecked(True)

        self.toolbar_files = self.node_editor_menu.addAction("Project
Files Window")
        self.toolbar_files.setCheckable(True)
        self.toolbar_files.triggered.connect(self.update_files_wnd)
        self.toolbar_files.setChecked(True)

        self.toolbar_events_vars =
self.node_editor_menu.addAction("Variables & Events Window")
        self.toolbar_events_vars.setCheckable(True)

self.toolbar_events_vars.triggered.connect(self.update_events_vars_wnd)
        self.toolbar_events_vars.setChecked(True)

        self.toolbar_functions =
self.node_editor_menu.addAction("Functions Window")
        self.toolbar_functions.setCheckable(True)

self.toolbar_functions.triggered.connect(self.update_functions_wnd)
        self.toolbar_functions.setChecked(True)

        self.node_editor_menu.addSeparator()

        for i in self.actions_creation_dict["Node Editor Menu"]:
            if i.__contains__("addSeparator"):
                self.node_editor_menu.addSeparator()
            else:
                mylist = self.actions_creation_dict["Node Editor
Menu"][i]
                act = QAction(mylist[3], parent=self,
statusTip=mylist[1], triggered=mylist[2])
                self.node_editor_menu.addAction(act)
                self.actions_creation_dict["Node Editor Menu"][i][0] =
act
        windows = self.graphs_parent_wdg.subWindowList()
        self.actSeparator.setVisible(len(windows) != 0)

        for i, window in enumerate(windows):
            child = window.widget()

            text = "%d %s" % (i + 1, child.getUserFriendlyFilename())
            if i < 9:
                text = '&' + text

            action = self.node_editor_menu.addAction(text)
            action.setCheckable(True)
            action.setChecked(child is self.currentNodeEditor())
            action.triggered.connect(self.windowMapper.map)
            self.windowMapper.setMapping(action, window)

    def update_functions_wnd(self):

self.toolbar_functions.setChecked(self.toolbar_functions.isChecked())

self.functionsDock.setVisible(self.toolbar_functions.isChecked())

    def update_events_vars_wnd(self):

self.toolbar_events_vars.setChecked(self.toolbar_events_vars.isChecked()
)

self.varsEventsDock.setVisible(self.toolbar_events_vars.isChecked())

    def update_properties_wnd(self):

self.toolbar_properties.setChecked(self.toolbar_properties.isChecked())

self.proprietiesDock.setVisible(self.toolbar_properties.isChecked())

    def update_libraries_wnd(self):

self.toolbar_library.setChecked(self.toolbar_library.isChecked())
        self.librariesDock.setVisible(self.toolbar_library.isChecked())

    def update_files_wnd(self):
        self.toolbar_files.setChecked(self.toolbar_files.isChecked())
        self.filesDock.setVisible(self.toolbar_files.isChecked())

    def activate_editor_mode(self):
        if self.graphs_parent_wdg.subWindowList():
            self.switch_display(Editor=True)
        else:
            self.switch_display(Welcome=True)


        self.node_editor_menu.setEnabled(True)
        self.library_menu.setEnabled(False)

        self.toolbar_library.setChecked(False)
        self.librariesDock.setVisible(self.toolbar_library.isChecked())

        self.toolbar_functions.setChecked(True)

self.functionsDock.setVisible(self.toolbar_functions.isChecked())

        self.toolbar_files.setChecked(True)
        self.filesDock.setVisible(self.toolbar_files.isChecked())

        self.toolbar_properties.setChecked(True)
self.proprietiesDock.setVisible(self.toolbar_properties.isChecked())

    def activate_designer_mode(self):
        self.switch_display(Designer=True)

    def activate_library_mode(self):
        if self.graphs_parent_wdg.subWindowList():
            self.switch_display(Library=True)
        else:
            self.switch_display(Welcome=True)

        # Handel buttons State
        self.node_editor_menu.setEnabled(False)
        self.library_menu.setEnabled(True)

        self.toolbar_library.setChecked(True)
        self.librariesDock.setVisible(self.toolbar_library.isChecked())

        self.toolbar_files.setChecked(False)
        self.filesDock.setVisible(self.toolbar_files.isChecked())

    def create_functions_dock(self):
        self.functionsDock = QDockWidget("Functions")
        self.nodesListWidget = NodeList()

        self.functionsDock.setWidget(self.nodesListWidget)

self.functionsDock.setFeatures(self.functionsDock.DockWidgetMovable)
        self.addDockWidget(Qt.LeftDockWidgetArea, self.functionsDock)

    def create_files_dock(self):

self.brows_btn.clicked.connect(self.files_widget.set_project_folder)
        # self.files_widget.setSizePolicy(QSizePolicy.Maximum,
QSizePolicy.Maximum)

        self.filesDock = QDockWidget("Project Files")
        self.filesDock.setWidget(self.files_widget)
        self.filesDock.setFeatures(self.filesDock.DockWidgetMovable)
        self.addDockWidget(Qt.RightDockWidgetArea, self.filesDock)

    def create_properties_dock(self):
        self.proprietiesWdg = PropertiesList(master_ref=self)

        self.proprietiesDock = QDockWidget("Properties")
        self.proprietiesDock.setWidget(self.proprietiesWdg)

self.proprietiesDock.setFeatures(self.proprietiesDock.DockWidgetMovable)

        self.addDockWidget(Qt.RightDockWidgetArea, self.proprietiesDock)

    def create_user_nodes_dock(self):
        self.varsEventsDock = QDockWidget("Variables & Events")
        self.VEStackedWdg = QStackedWidget()
        self.VEStackedWdg.setSizePolicy(QSizePolicy.MinimumExpanding,
QSizePolicy.MinimumExpanding)
        self.varsEventsDock.setWidget(self.VEStackedWdg)

self.varsEventsDock.setFeatures(self.varsEventsDock.DockWidgetMovable)
        self.addDockWidget(Qt.LeftDockWidgetArea, self.varsEventsDock)

    def delete_user_nodes_wgd(self, ref):
        self.VEStackedWdg.removeWidget(ref)

    def createStatusBar(self):
        self.statusBar().showMessage("Ready")

    def before_window_close(self):
        self.proprietiesWdg.clear_properties()

    def on_before_save_file(self):
        self.proprietiesWdg.clear_properties()

    def new_graph_tab(self):
        # This Check Prevents The Parent graph from opening in Cascade
view-mode
        if not self.graphs_parent_wdg.subWindowList():
            self.switch_display(Editor=True)

        node_editor = NodeEditorTab(masterRef=self)

        VEL = UserNodesList(scene=node_editor.scene,
propertiesWdg=self.proprietiesWdg)
        self.VEStackedWdg.addWidget(VEL)
        self.VEStackedWdg.setCurrentWidget(VEL)

        node_editor.scene.user_nodes_wdg = VEL

        subwnd = QMdiSubWindow()
        subwnd.setAttribute(Qt.WA_DeleteOnClose, True)
        subwnd.setWidget(node_editor)

        self.graphs_parent_wdg.addSubWindow(subwnd)
        subwnd.setWindowIcon(self.empty_icon)

        node_editor.scene.addItemSelectedListener(self.update_edit_menu)

node_editor.scene.addItemsDeselectedListener(self.update_edit_menu)


node_editor.scene.history.addHistoryModifiedListener(self.update_edit_me
nu)
        node_editor.addCloseEventListener(self.on_sub_wnd_close)

        self.graphs_parent_wdg.setViewMode(QMdiArea.TabbedView)

        subwnd.show()
        return subwnd

    def on_sub_wnd_close(self, widget, event):
        existing = self.findMdiChild(widget.filename)
        self.graphs_parent_wdg.setActiveSubWindow(existing)
        if self.ask_save():
            event.accept()
            self.delete_user_nodes_wgd(widget.scene.user_nodes_wdg)
            if (len(self.graphs_parent_wdg.subWindowList())-1) == 0:
                self.switch_display(Welcome=True)
            else:
                self.switch_display(Editor=True)
            self.before_window_close()
        else:
            event.ignore()

    def findMdiChild(self, filename):
        for window in self.graphs_parent_wdg.subWindowList():
            if window.widget().filename == filename:
                return window
        return None

    def setActiveSubWindow(self, window):
        if window:
            self.graphs_parent_wdg.setActiveSubWindow(window)
    def get_QWidget_content(self, widget):
        if [QKeySequenceEdit].__contains__(type(widget)):
            return widget.keySequence().toString()
        elif [QSpinBox, QDoubleSpinBox].__contains__(type(widget)):
            return widget.value()
        elif [QLineEdit, QLabel].__contains__(type(widget)):
            return widget.text()
        elif [QTextEdit].__contains__(type(widget)):
            return widget.toPlainText()
        elif [QRadioButton, QCheckBox].__contains__(type(widget)):
            return widget.isChecked()
        elif [QComboBox].__contains__(type(widget)):
            current = widget.currentText()
            widget.removeItem(widget.currentIndex())
            content_list = [current]
            for index in range(widget.__len__()):
                content_list.append(widget.itemText(index))
            widget.clear()
            widget.addItems(content_list)
            return content_list
        else:
            print(widget, "Widget Not Supported << Get")
            return None

    def set_QWidget_content(self, widget, new_value):
        if [QKeySequenceEdit].__contains__(type(widget)):
            widget.setKeySequence(new_value)
        elif [QSpinBox, QDoubleSpinBox].__contains__(type(widget)):
            widget.setValue(new_value)
        elif [QLineEdit, QLabel, QTextEdit].__contains__(type(widget)):
            widget.setText(new_value)
        elif [QRadioButton, QCheckBox].__contains__(type(widget)):
            widget.setChecked(new_value)
        elif [QComboBox].__contains__(type(widget)):
            widget.clear()
            widget.addItems(new_value)
        else:
            print(widget, "Widget Not Supported << Set")

    def about(self):
        QMessageBox.about(self, "About Calculator NodeEditor Example",
                          "The <b>Calculator NodeEditor</b> example
demonstrates how to write multiple "
                          "document interface applications using PyQt5
and NodeEditor. For more information visit: "
                          "<a
href='https://www.blenderfreak.com/'>www.BlenderFreak.com</a>")
```

UserNodesList

```
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

from nodeeditor.node_scene import NodeScene
from vvs_app.editor_properties_list import PropertiesList
from vvs_app.nodes.default_functions import *
from vvs_app.nodes.user_functions_nodes import UserFunction
from vvs_app.nodes.nodes_configuration import VARIABLES, get_class_by_type,
LISTBOX_MIMETYPE
from nodeeditor.utils import dumpException
from vvs_app.nodes.variables_nodes import UserVar


class UserNodesList(QTabWidget):

    def __init__(self, parent=None, scene: NodeScene = None, propertiesWdg:
PropertiesList = None):
        super().__init__(parent)
        self.user_nodes_data = []
        self.USER_NODES = {}

        self.scene = scene
        self.proprietiesWdg = propertiesWdg
        self.InitUI()

    def InitUI(self):
        # Add QTabWidget and add Both Variables Tab and Events Tab
        tab1 = QWidget()
        tab2 = QWidget()

        self.addTab(tab1, "Variables")
        self.addTab(tab2, "Functions")

        # Create Variables List
        self.var_list = QListWidget()
        self.function_list = QListWidget()

        # Create Variables and Events WNDs layout
        self.varLayout = QVBoxLayout()
        self.varLayout.setContentsMargins(0, 0, 0, 0)

        self.eventLayout = QVBoxLayout()
        self.eventLayout.setContentsMargins(0, 0, 0, 0)

        # Set Layouts For both
        tab1.setLayout(self.varLayout)
        tab2.setLayout(self.eventLayout)

        ## Setup CompoBox and add Button For Vars
        self.varCompoBox = QComboBox()
        self.varAddBtn = QPushButton("Add Variable")
        self.varHlayout = QHBoxLayout()
        self.varCompoBox.setMinimumHeight(28)
        self.varAddBtn.setMinimumHeight(28)
        self.var_list.setIconSize(QSize(28, 28))
        self.varHlayout.setContentsMargins(2, 2, 2, 2)
        self.varHlayout.addWidget(self.varCompoBox)
        self.varHlayout.addWidget(self.varAddBtn)
        self.varLayout.addLayout(self.varHlayout)
        self.varLayout.addWidget(self.var_list)
        self.var_list.setDragEnabled(True)

        # Setup CompoBox and add Button For Vars
        self.function_compo_box = QComboBox()
        self.event_add_btn = QPushButton("Add Function")
        self.eventHlayout = QHBoxLayout()
        self.function_compo_box.setMinimumHeight(28)
        self.event_add_btn.setMinimumHeight(28)
        self.function_list.setIconSize(QSize(28, 28))
        self.eventHlayout.setContentsMargins(2, 2, 2, 2)
        self.eventHlayout.addWidget(self.function_compo_box)
        self.eventHlayout.addWidget(self.event_add_btn)
        self.eventLayout.addLayout(self.eventHlayout)
        self.eventLayout.addWidget(self.function_list)
        self.function_list.setDragEnabled(True)

        self.var_list.startDrag = self.VarStartDrag
        self.function_list.startDrag = self.EventStartDrag

        # self.VarList.startDrag.connect()

        self.var_list.itemClicked.connect(lambda:
self.list_selection_changed(is_var=True))
        self.function_list.itemClicked.connect(lambda:
self.list_selection_changed(is_var=False))

        # self.VarList.itemSelectionChanged.connect(lambda :
self.list_selection_changed(var=True))
        # self.EventList.itemSelectionChanged.connect(lambda :
self.list_selection_changed(var=False))

        self.InitList()

    def set_user_node_Id_now(self, class_reference):
        id = 0
        while id in self.USER_NODES:
            id = id+1
        else:
            self.USER_NODES[id] = class_reference
            return id

    def get_user_node_by_id(self, node_id):
        if node_id not in self.USER_NODES:
            raise NodeTypeNotRegistered("node_type '%d' is not registered"
% node_id)
        else:
            return self.USER_NODES[node_id]

    def MakeCopyOfClass(self, node):
        class NewNode(node):
            pass
        return NewNode

    def InitList(self):
        self.function_compo_box.addItem('function',
userData=UserFunction.node_type)

        self.varCompoBox.addItem('float', userData=UserVar.node_type)
        self.varCompoBox.addItem('integer', userData=UserVar.node_type)
        self.varCompoBox.addItem('boolean', userData=UserVar.node_type)
        self.varCompoBox.addItem('string', userData=UserVar.node_type)

        # self.loadVars(self.userData.LoadData())
        self.varAddBtn.clicked.connect(lambda: self.add_new_node(var=True))
        self.event_add_btn.clicked.connect(lambda:
self.add_new_node(var=False))

    def add_new_node(self, var):
        if var:
            usage = self.varCompoBox.currentText()
            type =
self.varCompoBox.itemData(self.varCompoBox.currentIndex())
            node_name = f'user_{usage}'
        else:
            node_name = 'user_function'
            type =
self.function_compo_box.itemData(self.function_compo_box.currentIndex())
            usage = 'function'

        self.create_user_node(self.autoNodeRename(node_name), node_id=None,
type=type, user=True, node_usage=usage, node_structure='single value',
node_return='mutable')

    def create_user_node(self, name, node_id, type, node_return,
node_structure, node_usage, user=False):
        if type == UserVar.node_type:
            node = UserVar
        elif type == UserFunction.node_type:
            node = UserFunction

        # Get new Variable type and construct new Variable object
        node = get_class_by_type(type)
        new_node = self.MakeCopyOfClass(node)
        new_node.node_return = node_return
        new_node.node_structure = node_structure
        new_node.node_usage = node_usage
        node_data = {'node_name':name,
                     'node_id':node_id,
                     'node_usage':node_usage,
                     'node_type':type,
                     'node_return':node_return,
                     'node_structure':node_structure}

        # Add new copy of Var class Info to Dict of USER_VARS
        new_id = self.set_user_node_Id_now(new_node)
        new_node.nodeID = node_data['node_id'] = new_id
        new_node.name = name
        # Save new Var to list of vars with [name ,node_id ,type
,node_return ,node_structure]
        self.user_nodes_data.append(node_data)

        if type == UserFunction.node_type:
            A_list = self.function_list
        else:
            A_list = self.var_list

        # Add new QListItem to the UI List using Init Data
        self.addMyItem(new_node.name, new_node.icon, new_id,
node.node_type, A_list)

        if user:
            self.scene.history.storeHistory("Created User Node ",
setModified=True)
        self.scene.node_editor.UpdateTextCode()

    def addMyItem(self, name, icon=None, new_node_ID=int, node_type=int,
List=QListWidget):
        item = QListWidgetItem(name, List)  # can be (icon, text, parent,
<int>type)

        pixmap = QPixmap(icon if icon is not None else "")
        item.setIcon(QIcon(pixmap))
        item.setSizeHint(QSize(28, 28))
        item.setFlags(Qt.ItemIsEnabled | Qt.ItemIsSelectable |
Qt.ItemIsDragEnabled)

        # setup data
        item.setData(Qt.UserRole, pixmap)

        item.setData(80, node_type)

        item.setData(90, new_node_ID)

        item.setData(91, name)

    def list_selection_changed(self, is_var, *args, **kwargs):
        # Name line edite setup
        if is_var:
            item = self.var_list.currentItem()
        else:
            item = self.function_list.currentItem()

        self.proprietiesWdg.clear_properties()
        self.create_wdg_for_selection(item, is_var)

    def create_wdg_for_selection(self, item, is_var):
        # Create name widget
        self.node_name_input = QLineEdit()
        self.node_name_input.setValidator(QRegExpValidator(QRegExp("[A-Za-
z0-9_]+")))
        self.node_name_input.setText(f"{item.data(91)}")
        self.node_name_input.returnPressed.connect(lambda:
self.update_node_name(is_var))
        self.proprietiesWdg.create_properties_widget("Node Name",
self.node_name_input)

        # Create user_function return type widget
        if item.data(80) == UserFunction.node_type:
            self.return_type = QComboBox()
            return_types = list(self.scene.node_editor.return_types.keys())
            return_types.remove('Languages')
            self.return_type.addItems(return_types)


self.return_type.setCurrentText(self.get_user_node_by_id(item.data(90)).nod
e_return)
            self.return_type.currentIndexChanged.connect(lambda:
self.update_node_return(item.data(91), item.data(90)))
            self.proprietiesWdg.create_properties_widget("Return Type",
self.return_type)

        elif UserVar.node_type == item.data(80):
            self.structure_type = QComboBox()

            self.structure_type.addItems(["single value", "array"])


self.structure_type.setCurrentText(self.get_user_node_by_id(item.data(90)).
node_structure)
            self.structure_type.currentIndexChanged.connect(lambda:
self.update_node_structure_type(item.data(91), item.data(90)))
            self.proprietiesWdg.create_properties_widget("Structure Type",
self.structure_type)

        # Create user_node Delete button
        self.delete_btn = QPushButton(f"Delete {item.data(91)}")
        self.delete_btn.clicked.connect(lambda:
self.delete_node(item.data(91), user=True))
        self.delete_btn.setShortcut(

QKeySequence(f"Shift+{self.scene.masterRef.global_switches.switches_Dict['K
ey Mapping']['Delete']}"))
        self.proprietiesWdg.create_properties_widget("Delete",
self.delete_btn)

    def update_node_structure_type(self, node_name, node_id):
        structure_type = self.structure_type.currentText()
        node_ref = self.get_user_node_by_id(node_id)
        node_ref.node_structure = structure_type

        for item in self.user_nodes_data:
            if item['node_name'] == node_name:
                item['node_structure'] = structure_type

        for node in self.scene.nodes:
            if node.name == node_name:
                node.node_structure = structure_type

                for socket in node.inputs + node.outputs:
                    if socket.socket_type != 0:
                        socket.changeSocketType(5 if structure_type ==
'array' else socket.original_socket_type)

        self.scene.node_editor.UpdateTextCode()

    def update_node_return(self, node_name, node_id):
        return_type = self.return_type.currentText()
        for data in self.user_nodes_data:
            if data['node_name'] == node_name:
                data['node_return'] = return_type

        for node in self.scene.nodes:
            if node.name == node_name:
                node.node_return = return_type

        node_ref = self.get_user_node_by_id(node_id)
        node_ref.node_return = return_type

        self.scene.node_editor.UpdateTextCode()

    def VarStartDrag(self, *args, **kwargs):
        try:
            self.list_selection_changed(True)
            item = self.var_list.currentItem()
            var_ID = item.data(90)

            pixmap = QPixmap(item.data(Qt.UserRole))
            itemData = QByteArray()
            dataStream = QDataStream(itemData, QIODevice.WriteOnly)
            mimeData = QMimeData()
            drag = QDrag(self)

            dataStream << pixmap
            dataStream.writeInt(var_ID)
            dataStream.writeQString(item.text())
            dataStream.writeQStringList(["V"])

            mimeData.setData(LISTBOX_MIMETYPE, itemData)

            drag.setMimeData(mimeData)
            drag.setHotSpot(QPoint(pixmap.width() // 2, pixmap.height() //
2))
            drag.setPixmap(pixmap)

            drag.exec_(Qt.MoveAction)

        except Exception as e:
            dumpException(e)

    def EventStartDrag(self, *args, **kwargs):
        try:
            self.list_selection_changed(False)
            item = self.function_list.currentItem()
            event_ID = item.data(90)
            pixmap = QPixmap(item.data(Qt.UserRole))
            itemData = QByteArray()
            dataStream = QDataStream(itemData, QIODevice.WriteOnly)
            mimeData = QMimeData()
            drag = QDrag(self)

            dataStream << pixmap
            dataStream.writeInt(event_ID)
            dataStream.writeQString(item.text())
            dataStream.writeQStringList(["E"])

            mimeData.setData(LISTBOX_MIMETYPE, itemData)

            drag.setMimeData(mimeData)
            drag.setHotSpot(QPoint(pixmap.width() // 2, pixmap.height() //
2))
            drag.setPixmap(pixmap)

            drag.exec_(Qt.MoveAction)

        except Exception as e:
            dumpException(e)

    def update_node_name(self, is_var):
        if is_var:
            item = self.var_list.currentItem()
        else:
            item = self.function_list.currentItem()

        oldName = item.data(91)
        tryName = self.node_name_input.text()
        newName = self.userRename(oldName=oldName, tryName=tryName)
        if newName is None:
            return
        else:

            # get ref to user variable copy
            node_ref = self.get_user_node_by_id(item.data(90))

            # set item text to new name
            item.setText(newName)
            item.setData(91, newName)

            # set name of parent var
            node_ref.name = newName

            # rename all children grNode vars that have the old name
            for node in self.scene.nodes:
                if node.name == oldName:
                    node.name = newName
                    node.grNode.name = newName

            self.scene.node_editor.UpdateTextCode()

    def findListItem(self, selectedNodes: 'Nodes'):
        if selectedNodes != []:
            for item in range(self.var_list.count()):
                list_item = self.var_list.item(item)
                if list_item.text() == selectedNodes[0].name:
                    self.var_list.setCurrentItem(list_item)
                    self.list_selection_changed(is_var=True)
                    self.setCurrentIndex(0)
                    self.proprietiesWdg.create_order_wdg()
                    return list_item

            for item in range(self.function_list.count()):
                list_item = self.function_list.item(item)
                if list_item.text() == selectedNodes[0].name:
                    self.function_list.setCurrentItem(list_item)
                    self.list_selection_changed(is_var=False)
                    self.setCurrentIndex(1)
                    self.proprietiesWdg.create_order_wdg()

                    return list_item

            self.proprietiesWdg.clear_properties()
        else:
            self.proprietiesWdg.clear_properties()

###################################
    # Data
###################################

    def userRename(self, oldName, tryName: str):
        names = []
        for item in self.user_nodes_data:
            names.append(item['node_name'])

        if names.__contains__(tryName):
            return None
        else:
            for item in self.user_nodes_data:
                if item['node_name'] == oldName:
                    item['node_name'] = tryName
                    return tryName

    def autoNodeRename(self, name: 'Node'):
        x = 0
        newName = name

        # does a variable already has this name ?
        names = []
        for item in self.user_nodes_data:
            names.append(item['node_name'])
        # print(names)
        while names.__contains__(newName):
            x += 1
            newName = f"{name}{x}"

        else:
            return newName

    def delete_node(self, item_name, user=False):
        item_ref = None
        if self.var_list.findItems(item_name, Qt.MatchExactly):
            list_ref = self.var_list
            item_ref = self.var_list.findItems(item_name,
Qt.MatchExactly)[0]
```

```
        else:
            list_ref = self.function_list
            item_ref = self.function_list.findItems(item_name,
Qt.MatchExactly)[0]

        if item_ref:

            self.USER_NODES.pop(item_ref.data(90))
            selected = []
            for item in self.user_nodes_data:
                if item['node_name'] == item_name:
                    self.user_nodes_data.remove(item)

            for node in self.scene.nodes:
                if node.name == item_name:
                    selected.append(node)

            # This is split into two loops to prevent bugs that happen
while deleting nodes
            for node in selected:
                node.remove()

            list_ref.setCurrentItem(item_ref)

            list_ref.takeItem(list_ref.currentRow())
            list_ref.clearSelection()
            self.proprietiesWdg.clear_properties()
            self.scene.node_editor.UpdateTextCode()

            if user:
                self.scene.history.storeHistory("Delete User Node ",
setModified=True)

            self.scene.node_editor.UpdateTextCode()

        else:
            print("List Item Doesn't Exist")
```