
PWCT Documentation

Release 2.0

Mahmoud Fayed

Jan 09, 2025

CONTENTS

1	Introduction	2
1.1	Usage	2
1.2	Quotes about PWCT	3
1.3	History	4
1.4	Features	5
1.5	License	5
2	The Concept	6
2.1	The First Generation	6
2.1.1	The Concept of The First Generation	6
2.1.2	The Disadvantages of the First Generation	8
2.2	The Second Generation	9
2.2.1	Improving the Concept	9
2.2.2	Interactive Visualizations	9
2.2.3	Import Textual Source Code Files	13
2.2.4	Improving the Implementation	14
3	Getting Started	16
3.1	The Main Window	16
3.1.1	Components Browser	17
3.1.2	Project Files	19
3.1.3	Form Designer	20
3.1.4	Output Window	21
3.1.5	Goal Designer	23
3.2	Hello World Program	24
3.2.1	Hello World Program (Using the Mouse)	24
3.2.2	Hello World Program (Steps Summary)	26
3.2.3	Hello World Program (Using the Keyboard)	26
3.3	Using the Goal Designer	26
3.3.1	Copy & Paste the Steps	27
3.3.2	Modify the Steps	29
3.3.3	Using the Time Machine	30
3.3.4	Moving Steps Up & Down	32
3.3.5	Cut & Paste Steps	33
3.3.6	Inserting Steps	34
3.3.7	Comment/Uncomment Steps	35
3.3.8	Deleting Steps	37
3.3.9	Search and Replace	38
3.3.10	Using the Again Button	41
3.3.11	Undo	43

3.4	Visual Source Files	43
3.4.1	Saving the file	43
3.4.2	Opening the file	45
3.4.3	Starting new file	46
3.4.4	Save As	46
3.4.5	Printing the file	47
3.4.6	Go to line	48
3.5	Customization	49
3.5.1	View Menu	49
3.5.2	Customization Window	51
3.6	Run Programs	53
3.6.1	Program Menu	53
3.6.2	Main File toolbar	56
3.7	More Options	57
3.7.1	Tools Menu	57
3.7.2	Distribute Menu	60
3.7.3	Help Menu	62
4	Rich Comments	63
4.1	Using the Header Component	63
4.2	Using the New Line Component	65
4.3	Using the Comment Component	67
4.4	Using the Image Component	69
4.5	Printing some text	72
5	Basic Program component	75
5.1	Introduction	75
5.2	Selecting the Component	75
5.3	Steps Tree	76
6	Quick Start component	77
6.1	Introduction	77
6.2	Selecting the Component	77
6.3	The Interaction Page	78
6.4	Steps Tree	78
6.5	Adding Comments	79
7	Say Hello program	80
7.1	Introduction	80
7.2	Program Steps	81
7.3	Creating the Program	81
8	Using Variables	92
8.1	Introduction	92
8.2	Program Steps	93
8.3	Creating the Program	94
9	Deep Copy	127
9.1	Introduction	127
9.2	Program Steps	128
9.3	Creating the Program	128
10	Implicit Conversion	141
10.1	Introduction	141
10.2	Program Steps	141

10.3	Creating the Program	142
11	Operators	153
11.1	Arithmetic Operators	153
11.2	Relational Operators	153
11.3	Logical Operators	154
11.4	Bitwise Operators	154
11.5	Assignment Operators	154
11.6	Misc Operators	155
11.7	Operators Precedence	155
12	Operators Precedence	157
12.1	Introduction	157
12.2	Program Steps	157
12.3	Creating the Program	158
13	Loop and Condition	160
13.1	Introduction	160
13.2	Program Steps	160
13.3	Creating the Program	161
14	Main Menu	168
14.1	Introduction	168
14.2	Program Steps	169
14.3	Creating the Program	170
15	Dynamic Loop	208
15.1	Introduction	208
15.2	Program Steps	209
15.3	Creating the Program	209
16	Modify Lists	224
16.1	Introduction	224
16.2	Program Steps	225
16.3	Creating the Program	226
17	Exit from two loops	254
17.1	Introduction	254
17.2	Program Steps	255
17.3	Creating the Program	255
18	The Loop Command	265
18.1	Introduction	265
18.2	Program Steps	266
18.3	Creating the Program	266
19	Short Circuit Evaluation	276
19.1	Introduction	276
19.2	Program Steps	277
19.3	Creating the Program	278
20	Using Functions	306
20.1	Introduction	306
20.2	Program Steps	307
20.3	Creating the Program	308

21 Variables Scope	343
21.1 Introduction	343
21.2 Program Steps	344
21.3 Creating the Program	344
22 RingPWCT Visual Components	358
22.1 General/Comments	360
22.2 General/Templates	361
22.3 Console	362
22.4 Control Structures	364
22.5 Variables and Operators	366
22.6 Functions	367
22.7 Program Structure	368
22.8 Lists	370
22.9 Strings	372
22.10 Date and Time	373
22.11 Check Data Type and Conversion	375
22.12 Math	376
22.13 Files	378
22.14 System	380
22.15 Dynamic Code and Debugging	381
22.16 Database/ODBC	383
22.17 Database/MySQL	385
22.18 Database/SQLite	386
22.19 Security and Internet Functions	388
22.20 Object Oriented Programming	390
22.21 Functional Programming	391
22.22 Reflection and Meta-programming	393
22.23 StdLib/Functions	394
22.24 StdLib/Classes	402
22.25 WebLib/General	403
22.26 WebLib/Classes	405
22.27 LibCurl	406
22.28 GUI/Add Components	406
22.29 GUI/Classes	417
22.30 GUI/Dialogs	442
22.31 GUI/More	443
23 Resources	444
23.1 Ring Language Website	444
Index	445

Contents:

**CHAPTER
ONE**

INTRODUCTION

Welcome to the PWCT visual programming language!



1.1 Usage

PWCT is a visual programming language that focuses on Productivity and Ease-of-Use!

If you want to learn programming, create applications/systems or get some new ideas about visual programming in the practice then you are in the right place. The goal of this project is to present programming to every computer users, whether they are beginners or professionals. Beginners means that the tools of programming must be accessible – must be easy. So I decided to take coding out of programming. And presenting programming to professional developers requires a tool that is productive and unlimited and can be extended.

The domain of the problem is called “Visual Programming Languages.” There are many projects in this domain, but most of these languages are domain-specific languages that are used in education, But with respect to general-purpose visual programming languages, there are few of them. PWCT don’t use the Drag-and-Drop method. PWCT provide a new method based on Automatic Steps Tree Generation and Update in response to interaction with components that provide to the user simple data entry forms. The idea behind this new method is to mix between programming

using Diagrammatic approach and programming using Form-based approach where the integration between the two approaches are done seamlessly through an Automatic Visual Representation Generation process. This is just the basic idea and many other ideas are developed around this concept to get a practical general purpose visual programming language for real world tasks.

1.2 Quotes about PWCT

“PWCT2 is transforming how we think about programming! This innovative visual programming language makes coding accessible and fun. Created in the Ring language, it supports importing and exporting code seamlessly.”

, Mudit Juneja (India)

“Programming language masterpiece! I have to develop an educational software for my PhD Thesis and i haven’t the time to learn a coding language now. Thumbs up for Mahmoud Samir Fayed and his research team. “

, jslafas (Sourceforge)

“This is great for making applications. I am a beginner who is still learning but with a few tutorial videos i’m sure i’ll get the hang of it soon enough”

, jamesh101 (Sourceforge)

“This Project is best way to learn programming”

, eternel0422 (Sourceforge)

“This is a great tool for development and strives to remove many of the problems coding with text based languages. It aides you in putting your program logic together and does not force you to use one style of programming (OO, procedural, event and others). Instead it lets your focus your attention on solving the problem. It also standardizes errors for debugging your program. A lot of work went into this project. “

, c_horne (Sourceforge)

“Nice implementation and interface. And it seems that it has many possibilities!! “

, stavros68 (Sourceforge)

“Nice Concept Keep Developing it!!!”

, aryaputrasrj (Sourceforge)

“Some years ago I tried to use programming tool also without coding(it used block-like structure), but I was dissapointed. But this tool is amazing!!! I can say that it is really alternative to coding! “

, bettinaf1986 (Sourceforge)

“I can agree with most of this reviews: this project is great! “

, davidshwinders (Sourceforge)

“Very smooth and non intrusive interface. “

, revenajs1983 (Sourceforge)

“Really great project for newbies and noobs who doesn’t know programming skills. “

, annhouge (Sourceforge)

“Very, very light program, and fully featured. “

, congworlniros (Sourceforge)

“Version to evaluate , like supernova 1.3 , this tool could be the last word in human computer interaction , no more difference Java and Visual Basic , my word is equal to a computer command . The best idea I’ve seen . “

, oid-3753088 (Sourceforge)

“Mahmoud Fayed, I have started used your Programming Without Coding Technology. Its just incredible to use your application and your programming language, I dont think anything is more worth to made this Programming Without Coding Technology . I dont think anything is more worth to made coding to beginners like me and to the common people. I dont think anything is more worth to made coding such faster ,easy and visualize. its just incredible and future of coding so good luck for future to change the coding technology and hence change the world of codinghence entire universe..... the spreading or marketing can change whole education industry and coding technology.... thank you “

, nikunjkavadia (Sourceforge)

1.3 History

PWCT 1.0 is released on October 18, 2008

PWCT 1.1 is released on February 20, 2009

PWCT 1.2 is released on May 4, 2009

PWCT 1.3 is released on May 30, 2009

PWCT 1.4 is released on August 28, 2009

PWCT 1.5 is released on March 27, 2010

PWCT 1.6 is released on May 16, 2010

PWCT 1.7 is released on September 15, 2010

PWCT 1.8 is released on October 18, 2011 (Latest update : 22 April 2013)

PWCT 1.9 is released on May 7, 2013 (Latest update : 20 November 2024)

PWCT 2.0 is released on March 1, 2023 (Latest update : 9 January 2025)

1.4 Features

The PWCT language comes with the next features

Tip: The language is ready for production!

- Visual Programming Language
- Program Visualization
- Support the Ring programming language
- Many Samples and Applications
- Complete Documentation.

1.5 License

The PWCT Visual Programming Language

Version 2.0

Free Software

Copyright (c) Mahmoud Fayed

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER TWO

THE CONCEPT

In this chapter we are going to discuss the goals behind the language design and implementation.

2.1 The First Generation

In this generation we did a lot of research to develop the main ideas behind the project

2.1.1 The Concept of The First Generation

In December 2005, I started to think about developing PWCT.

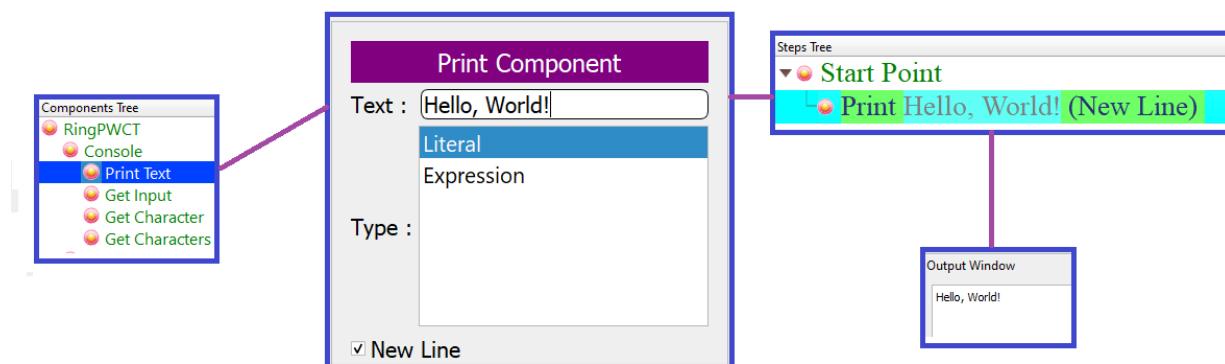
I started with many ideas and I tested each idea through practical development

In 2007, The core idea was very simple (Goal Designer instead of Code Editor)

We see this in the next screen shot, Instead of writing textual code directly, we follow a procedure of four simple steps

1. Select a component
2. Enter the data required for this component
3. Interact with the steps tree (If you need this)
4. Run the program

The next diagram demonstrates something similar to what I wrote using a pen and paper to explain the idea to my friends (The diagram uses screen shots from PWCT 2.0 that doesn't exist at that time).



At that time, the advantages were very clear

- Selecting components means we can explore the programming system and start using it without previous knowledge.
- Separating data from instructions through the data-entry forms provide the ability for quick reuse
- Using a GUI to program means easy support for translation (Arabic, English, French, etc.)
- No Syntax Errors (Happy experience for beginners)
- The generated steps tree (instead of source code) provides a chance for Maximum Readability
- The generated steps could replace many lines of source code (Higher Abstraction)
- The generated steps could be in many locations at the same time
- Interaction with a tree (Steps Tree) provide the ability to control many nodes (Parent & Children) together (Faster)
- The same component could generate different steps based on the data that we enter through data-entry forms

This design was enough to attract a few thousands of developers until 2009

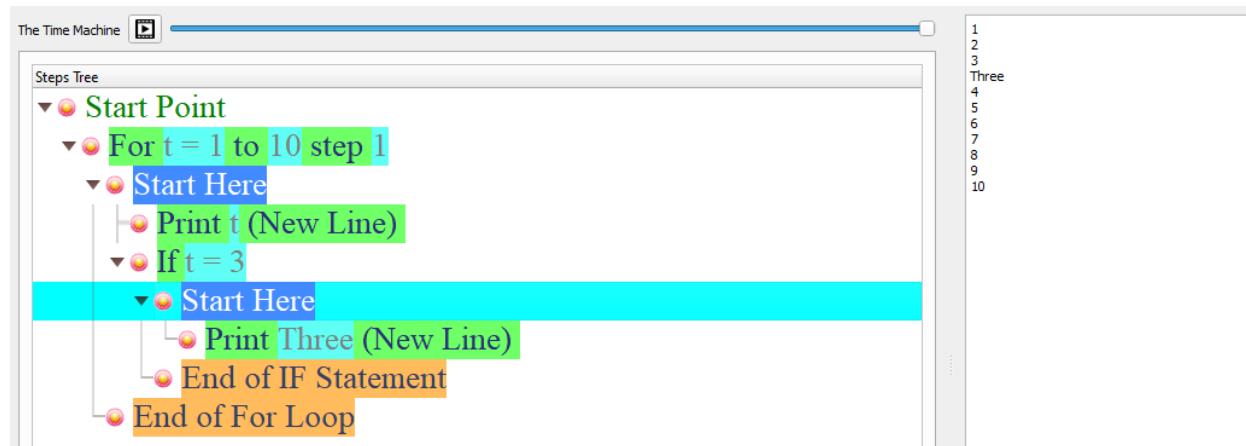
from 2009 to 2013, We improved the system with the next features

1. Using Keyboard Shortcuts to improve the writability
2. Adding Colors & Customization to the Steps Tree
3. Improving a domain-specific language that are used for developing the components
4. Supporting more programming languages in the code generation layer
5. Show/Hide steps based on the context
6. Two modes (Syntax Directed Editor OR Free Editor + VPL Compiler)
7. Automatic Documentation (Generate HTML Files from Visual Programs)
8. Optional automatic update of programs after components update
9. Adding the Time Dimension

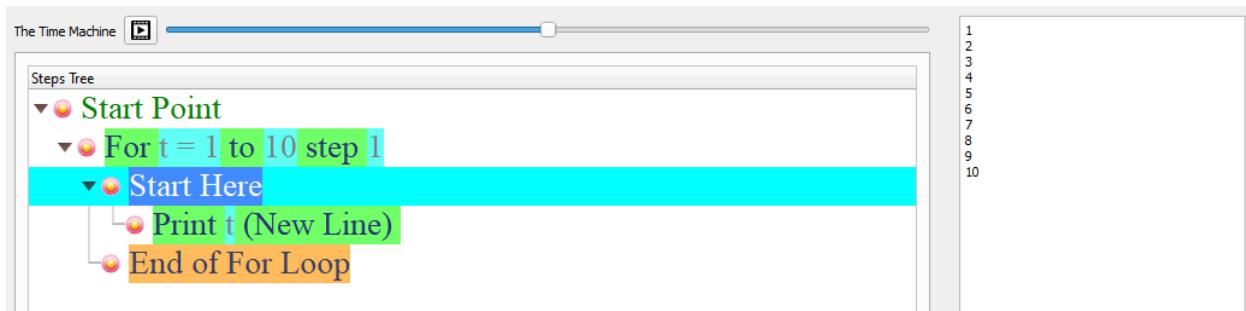
Using the Time Dimension we can

- Move to a point in the past during development and run the program
- Play programs as movies to learn how to create a program step by step

For example, the next program print numbers from 1 to 10 and print the Three message after the number 3.



Using the Time Machine slider, we can move to the past and run the program before adding the if statement that print the Three message.



The previous screen shots uses PWCT 2.0 but, the same idea is implemented in the first generation of PWCT from 2011 (PWCT 1.8)

These updates were enough to attract hundred thousands of developers to download the product and try it!

2.1.2 The Disadvantages of the First Generation

It's all about the implementation and the missing features!

The concept is very good, that attracted many developers with different background and they started asking about many things!

And Yes! some developers asked about improving the concept itself! (More innovation is required)

1. Some developers reported that using Code Editors is too much flexible and faster in some situations
2. Slow performance when using large visual source files
3. Visual Source Files are database files (not textual) which is not good for Version Control software
4. Support importing the source code (Convert source code files to Visual Source Files)
5. Opening many files and quick navigation
6. Multiplatform support (Windows, Linux & macOS)
7. Support Web & Mobile development
8. Support more programming languages
9. Translation & Unicode
10. Modern Form Designer
11. Support powerful libraries and frameworks
12. More educational resources!

I started to think about all of these issues and feature requests and how I can implement them.

I discovered that I need a new programming language (Ring) that could help me to do all of this in an efficient way

2.2 The Second Generation

The second generation (PWCT 2.0) is developed using the Ring programming language

Our goal is to provide a more powerful version of the software that satisfy two conditions

- (1) To be an easy tool to start learning about real programming concepts
- (2) To be a better replacement for Textual Code Editors

In the first generation we tried to achive the same goals but, we did a lot of mistakes, we learned from them and developed new ideas then we implemented these ideas in PWCT 2.0

I could be brave and say that after PWCT 2.0 development, I don't see any reason to use a Textual Code Editor when developing Ring applications. The same could be applied for other textual programming languages when we support them.

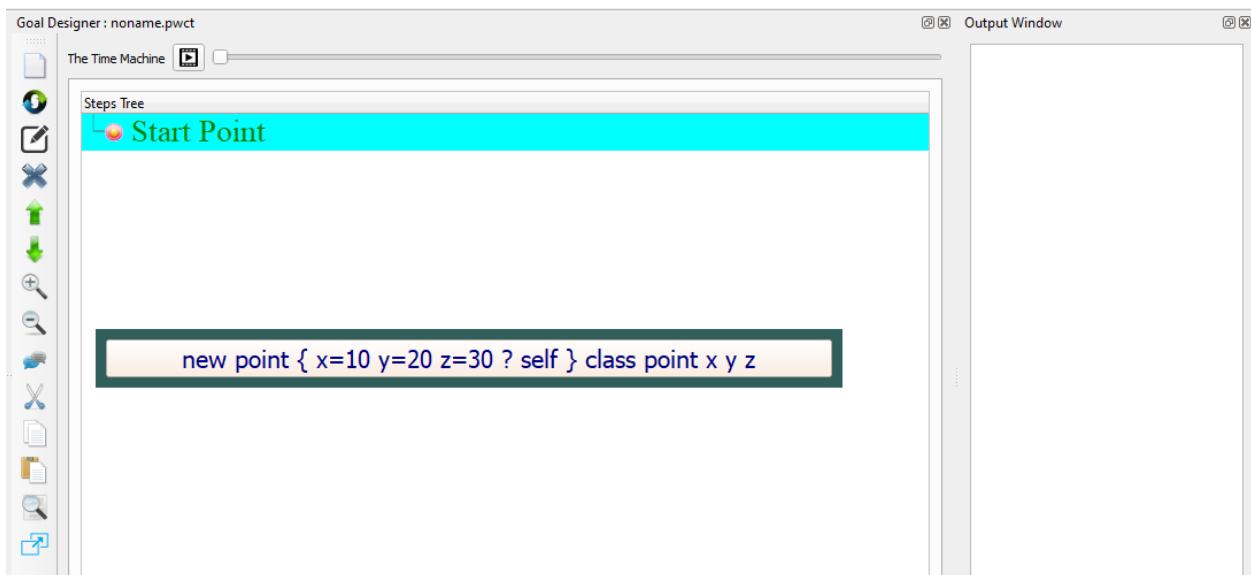
2.2.1 Improving the Concept

- 1- The Ring textual language is designed for high writability
- 2- The PWCT visual language is designed for high readability
- 3- We can use them together at the same time using (Ring2PWCT)
 - 3.1 - We can write Ring code directly and get the Visual Representation
 - 3.2 - We can import textual source code files and complete projects written in Ring

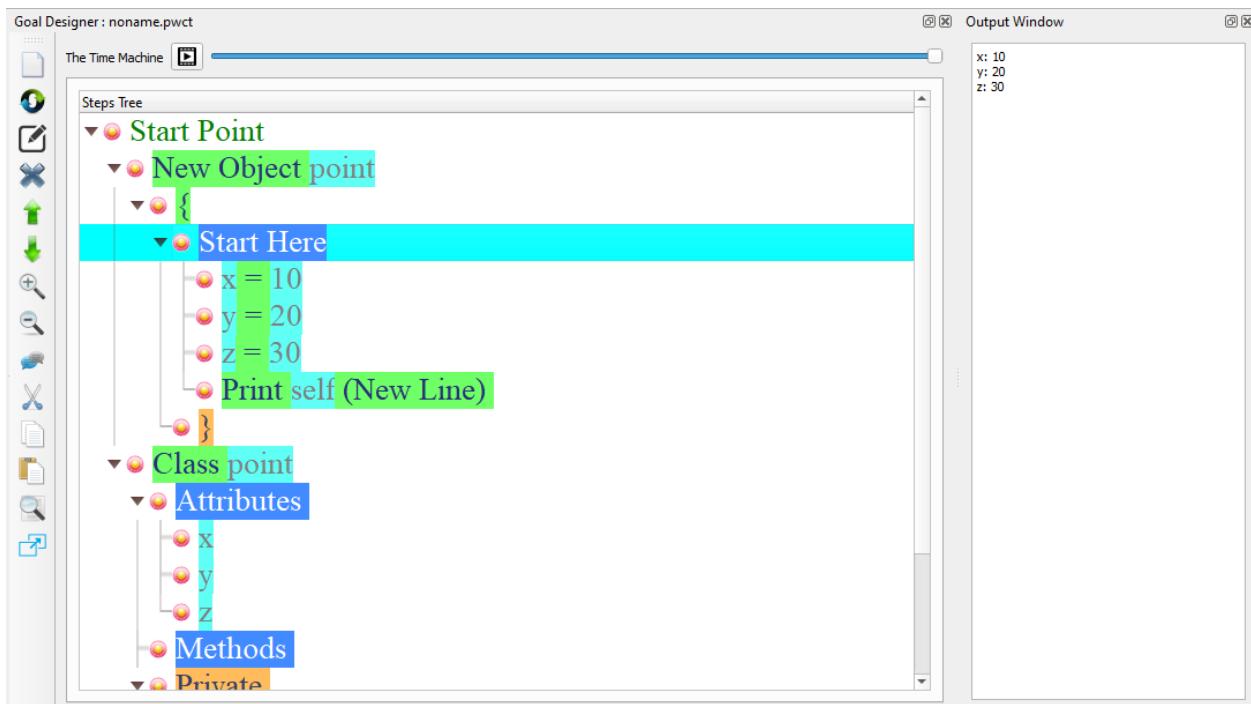
2.2.2 Interactive Visualizations

In the Goal Designer we can type Ring code directly and PWCT will do the visualization

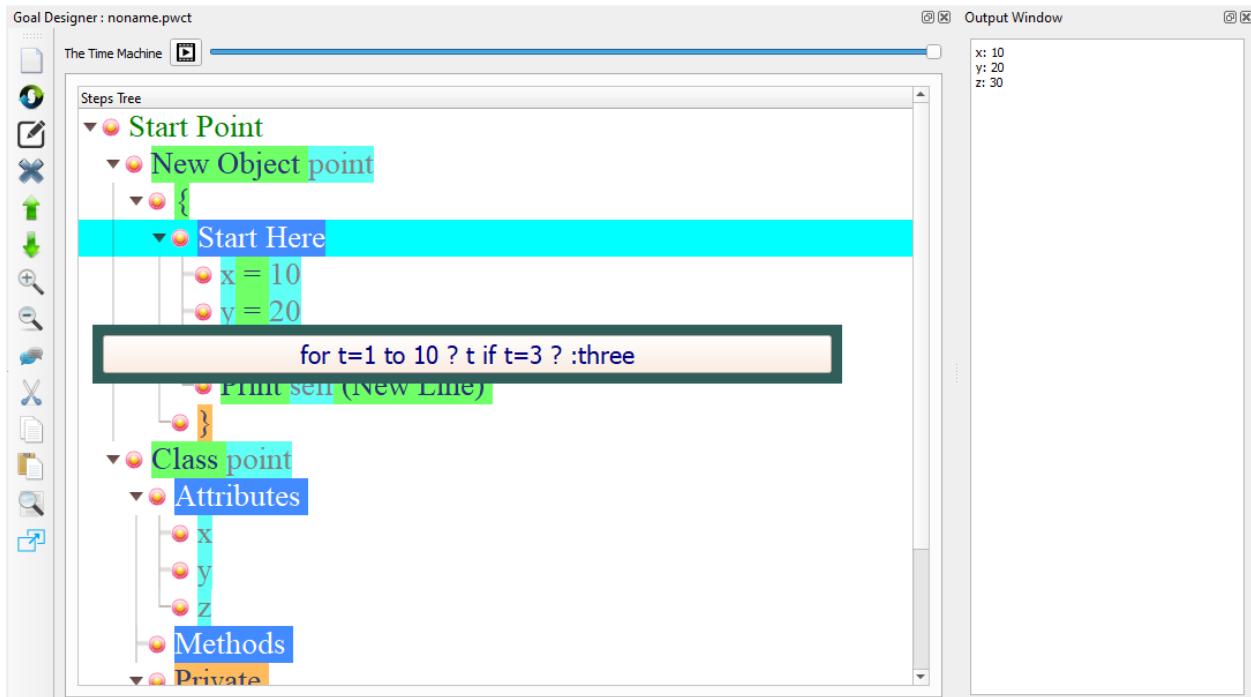
The next Ring code create an object and a class in one line.



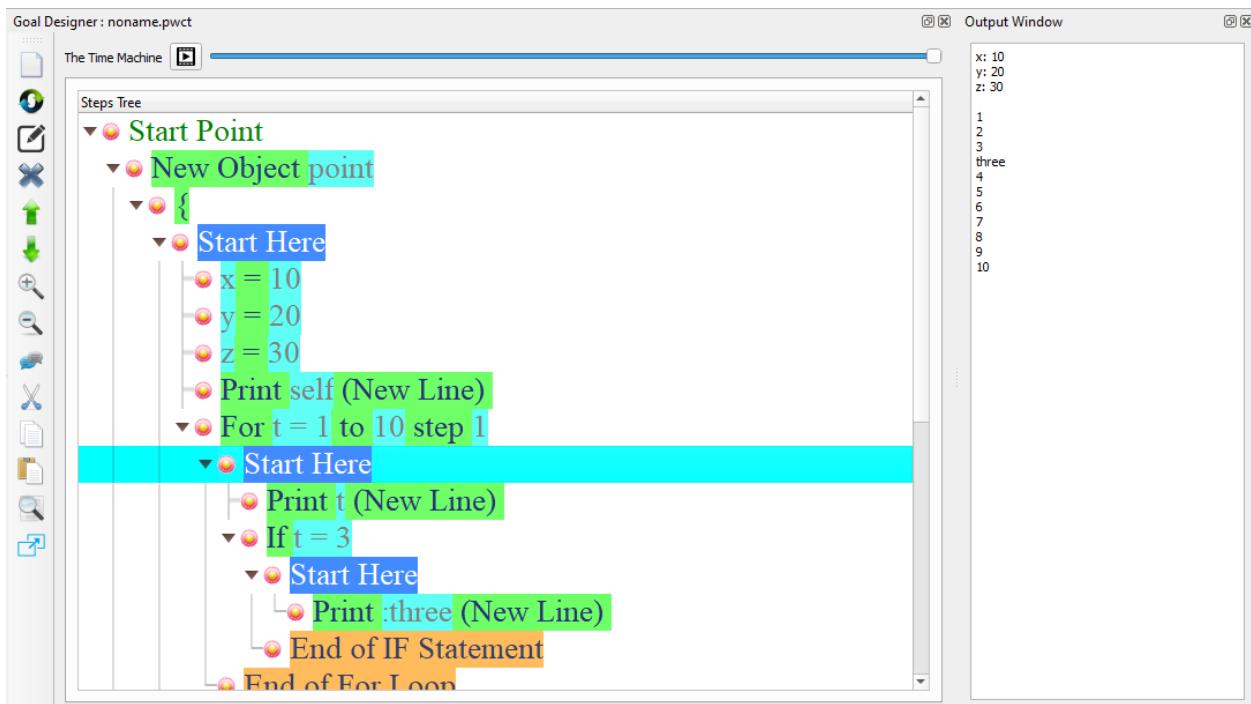
We get the visual output in the Steps Tree



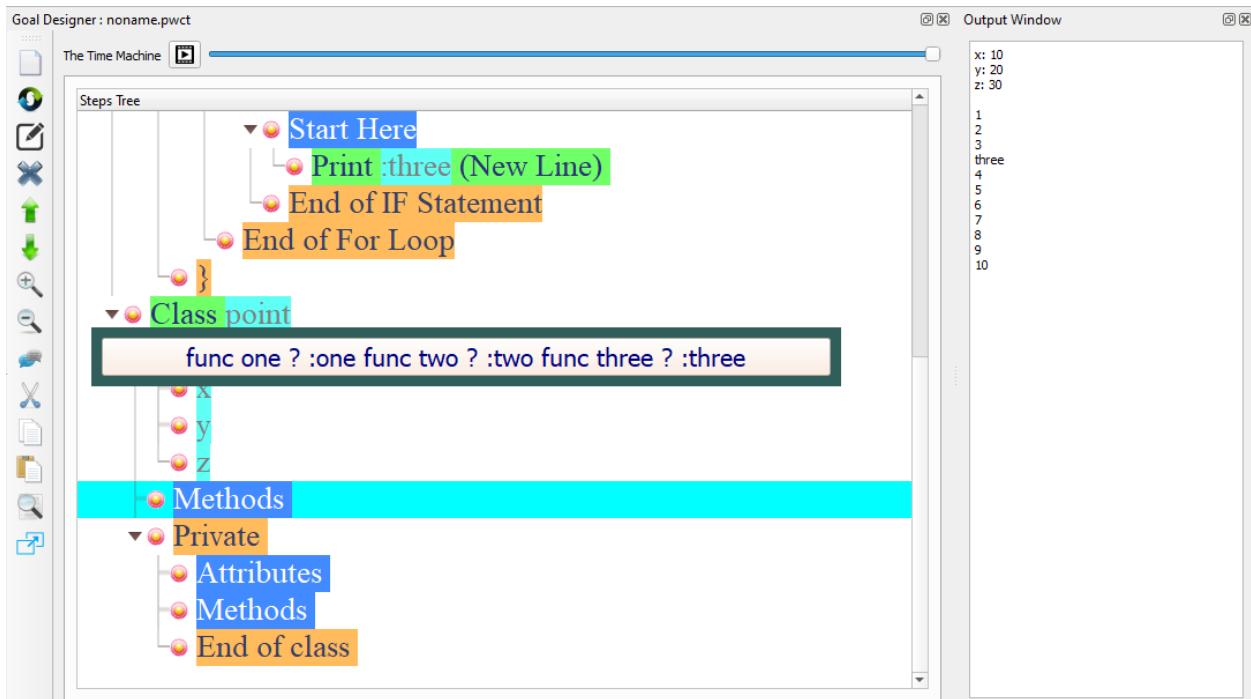
We can continue and write more code, for example using a (For Loop) and (If statement)



Again, we see the output in the steps tree



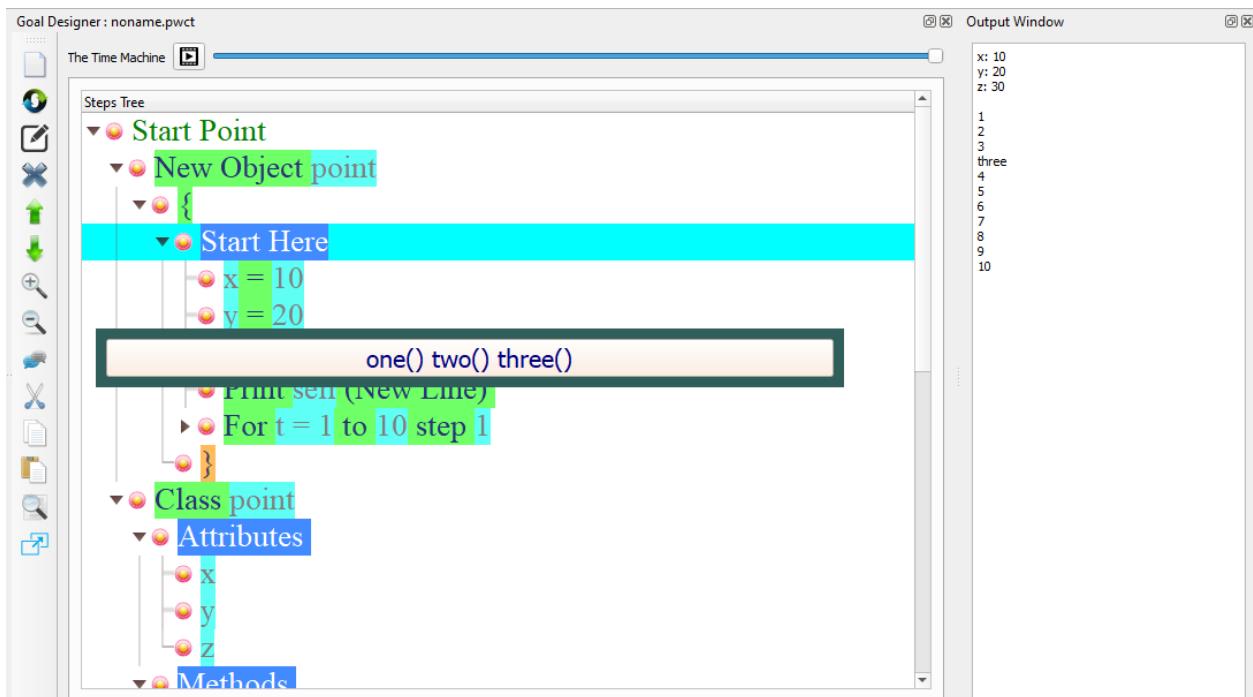
We can define three functions (one, two & three)



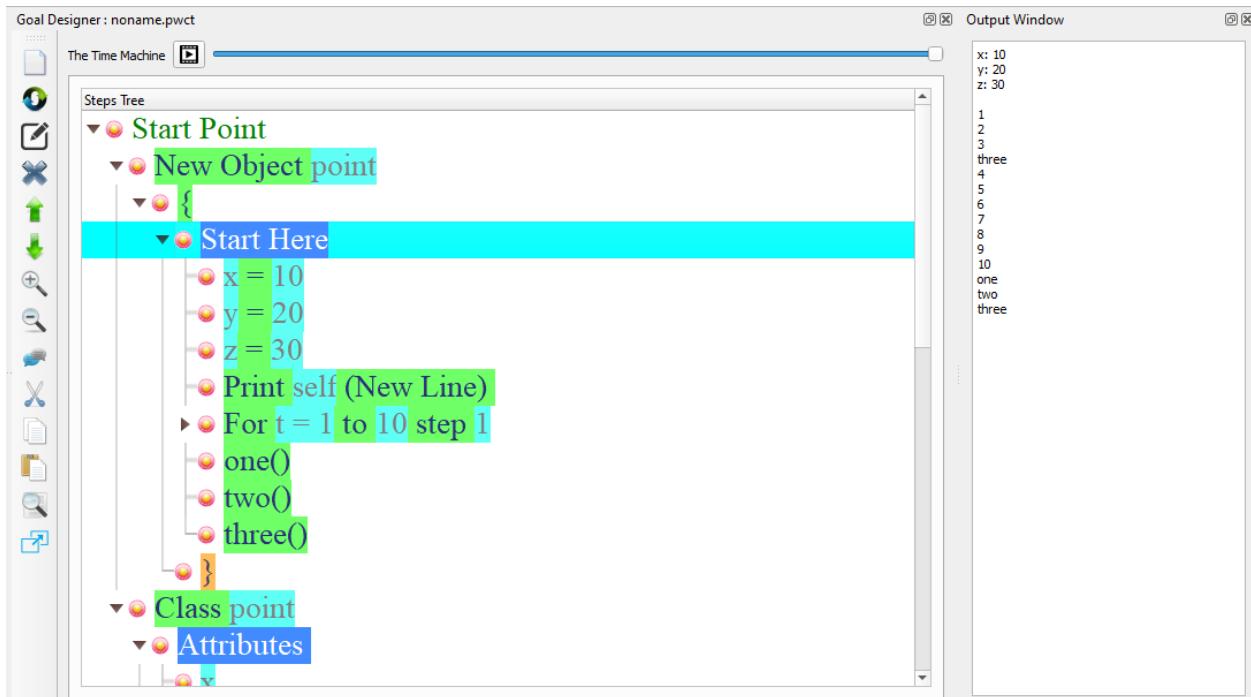
The steps tree are updated



We can call these functions



So, if you are a Ring programmer and know how to write textual Code, You can reuse this knowledge and PWCT will not get in your way!

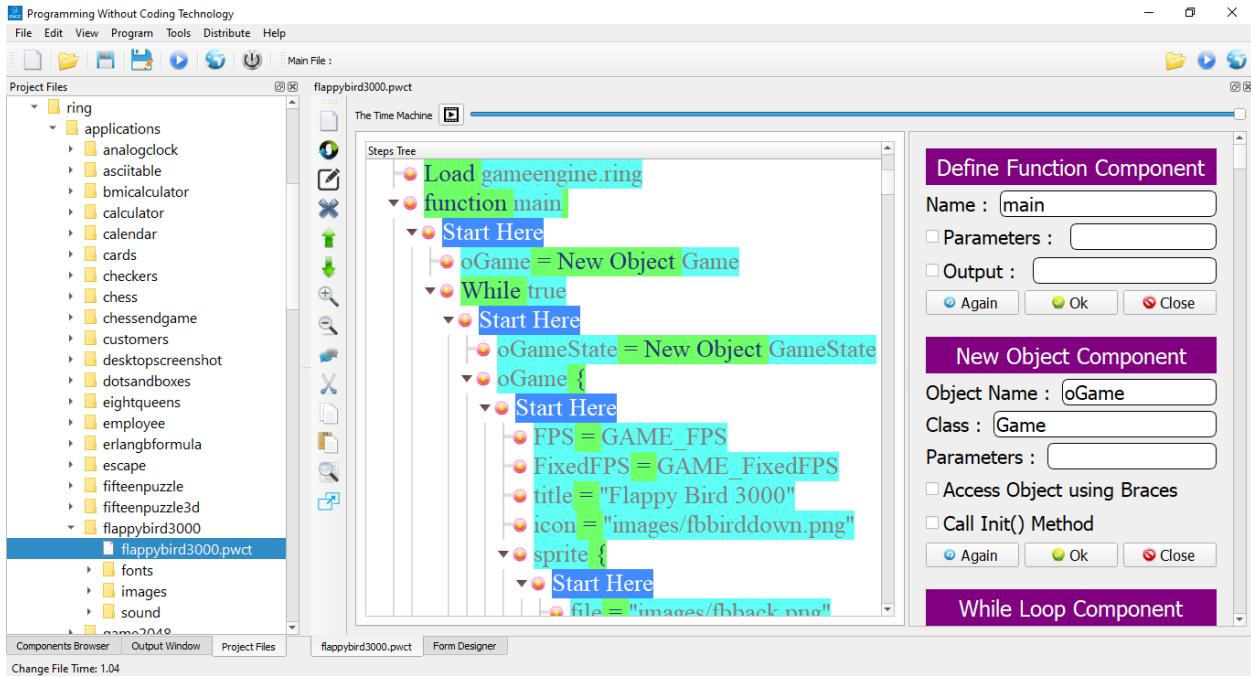


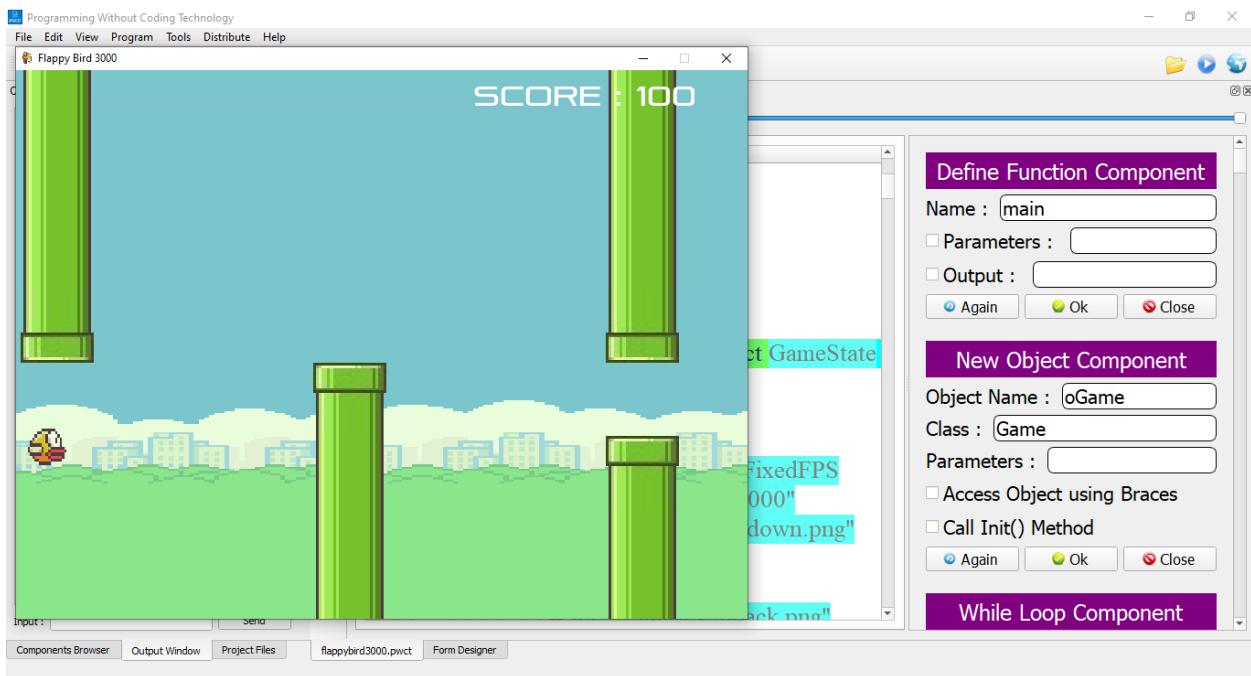
2.2.3 Import Textual Source Code Files

PWCT 2.0 comes with Ring2PWCT that can import any textual source code file.

We imported all of the Ring applications and samples

For example the next screen shots for the Flappy Bird 3000 game



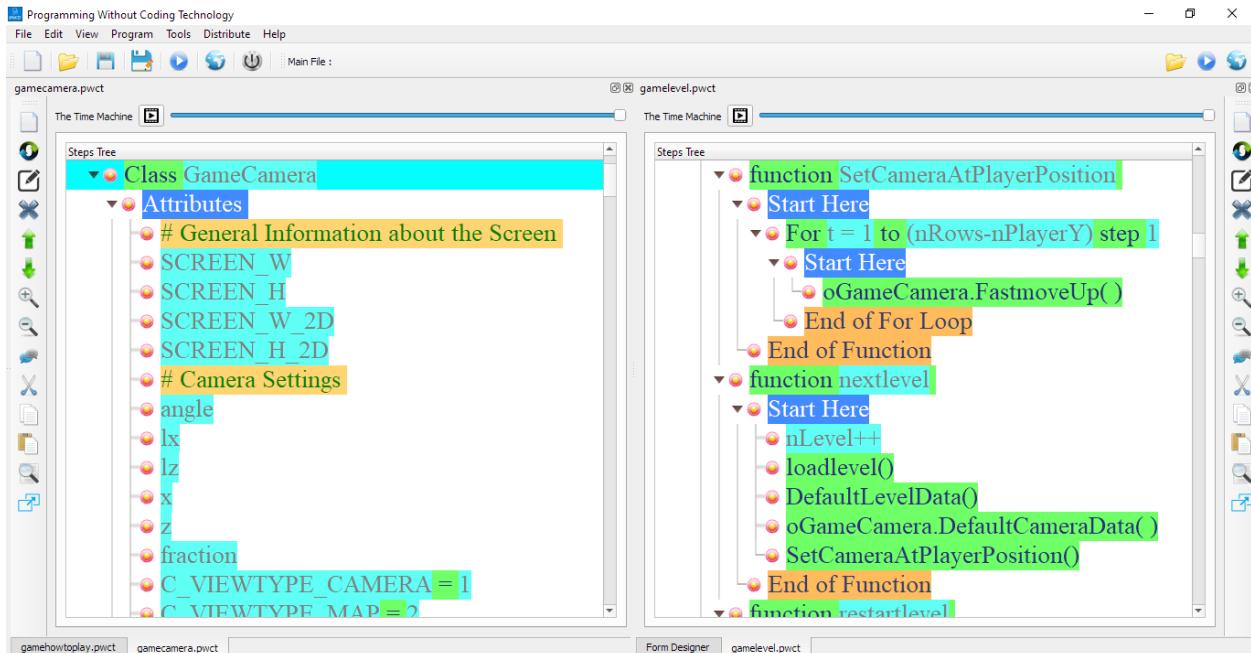


2.2.4 Improving the Implementation

In the second generation of PWCT, we worked on solving all of the reported problems in the design and implementation of the first generation. Also we implemented most feature requests that are useful and improve the flexibility of the product.

Open Many Files

PWCT 2.0 Support opening many visual source files at the same time



Inserting Steps

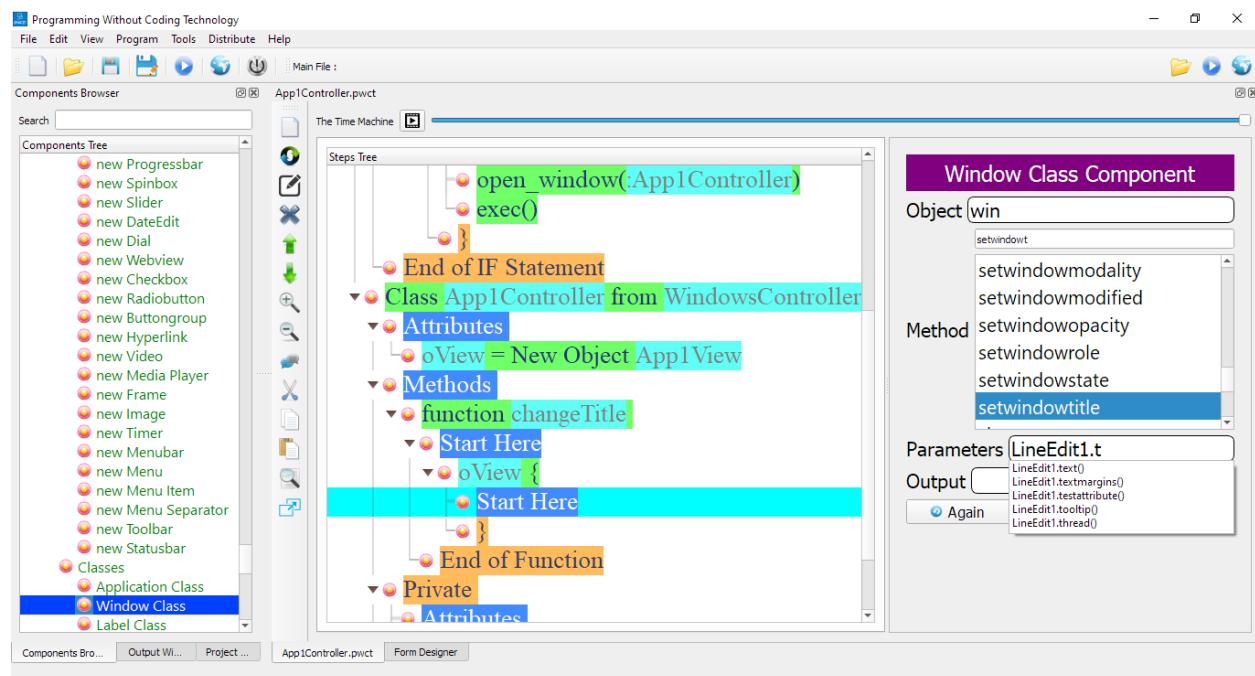
PWCT 2.0 support starting new interactions at any location in the steps tree (Not just Start Here steps)

The new steps will be inserted in the right location

This feature is supported while using components, Ring2PWCT and Paste operations

Auto-Complete

In PWCT 2.0 Interaction Pages, we have better usability through default values, search in listboxes and better auto-complete.



64-bit version

We have 64-bit version of the software (no longer limited to ~4gb of memory)

CHAPTER THREE

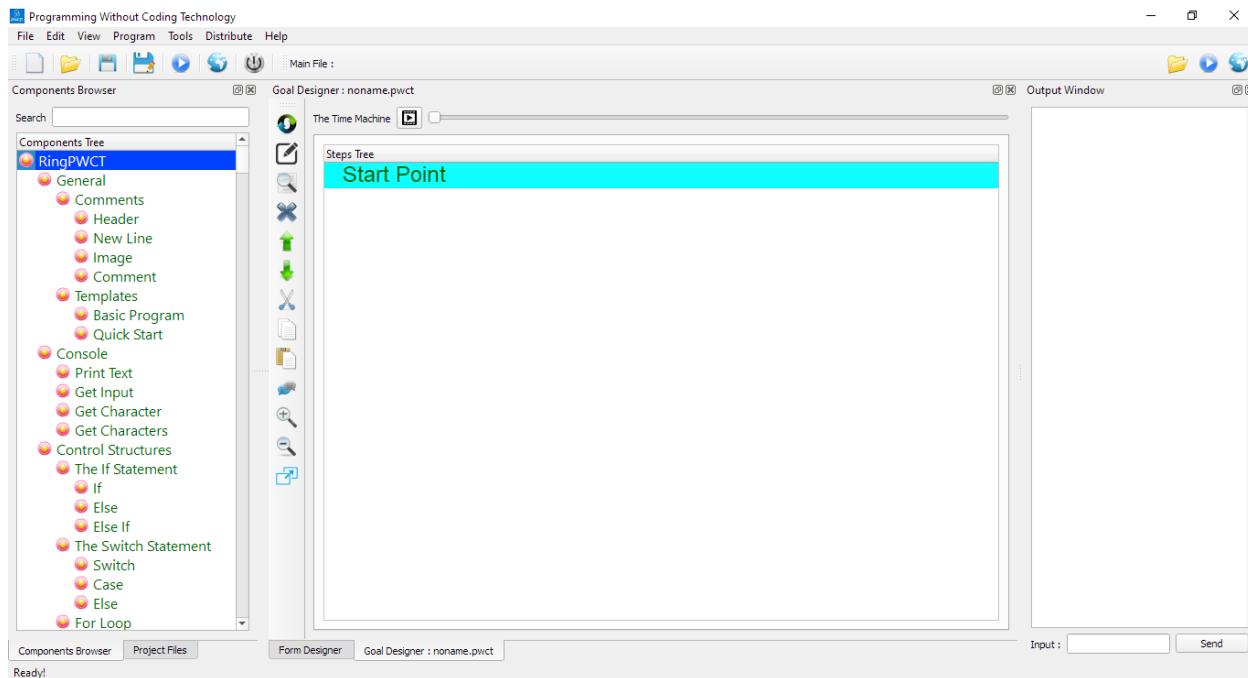
GETTING STARTED

In this chapter we are going to learn how to create our first application using PWCT

3.1 The Main Window

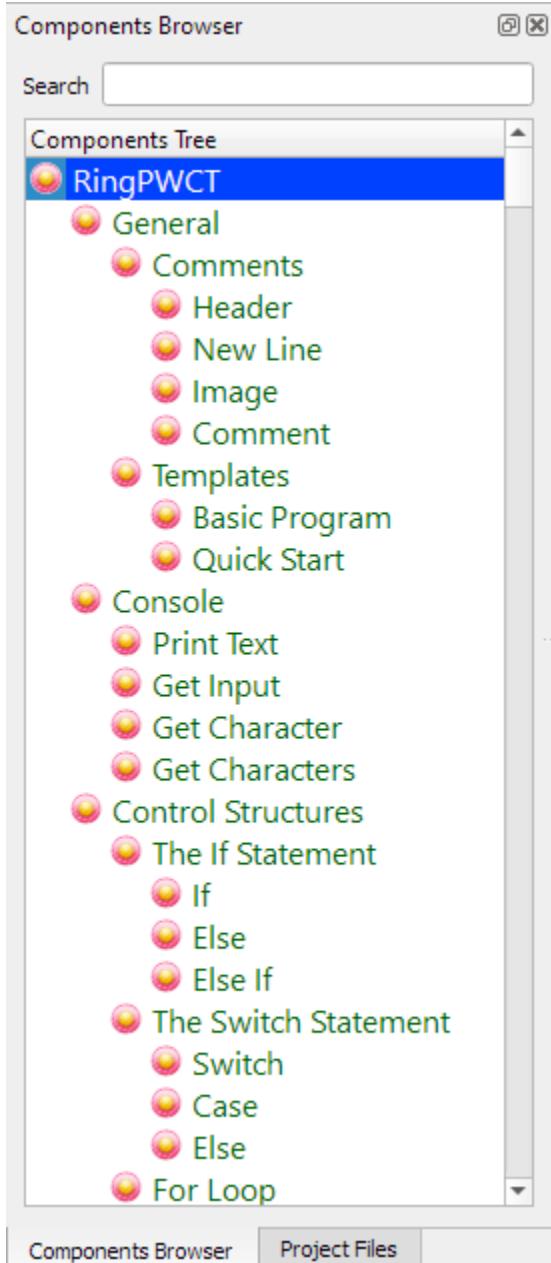
When we run PWCT, The Main Window contains the next dockable windows

- Components Browser
- Project Files
- Form Designer
- Output Window
- Goal Designer

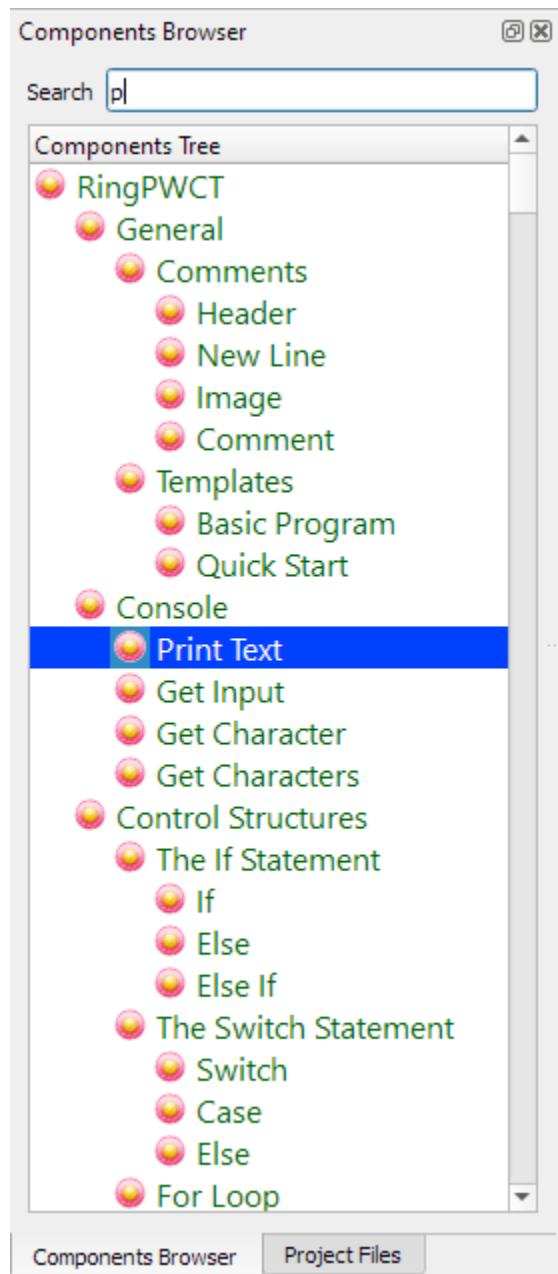


3.1.1 Components Browser

- 1 - Using the Components Browser, We can select a component to use
- 2 - Each component could provide an Interaction Page (Data-Entry Window)
- 3 - The Component lead the steps generation process in the Steps Tree (Inside the Goal Designer)
- 4 - The Component also generate the Textual source code in the background



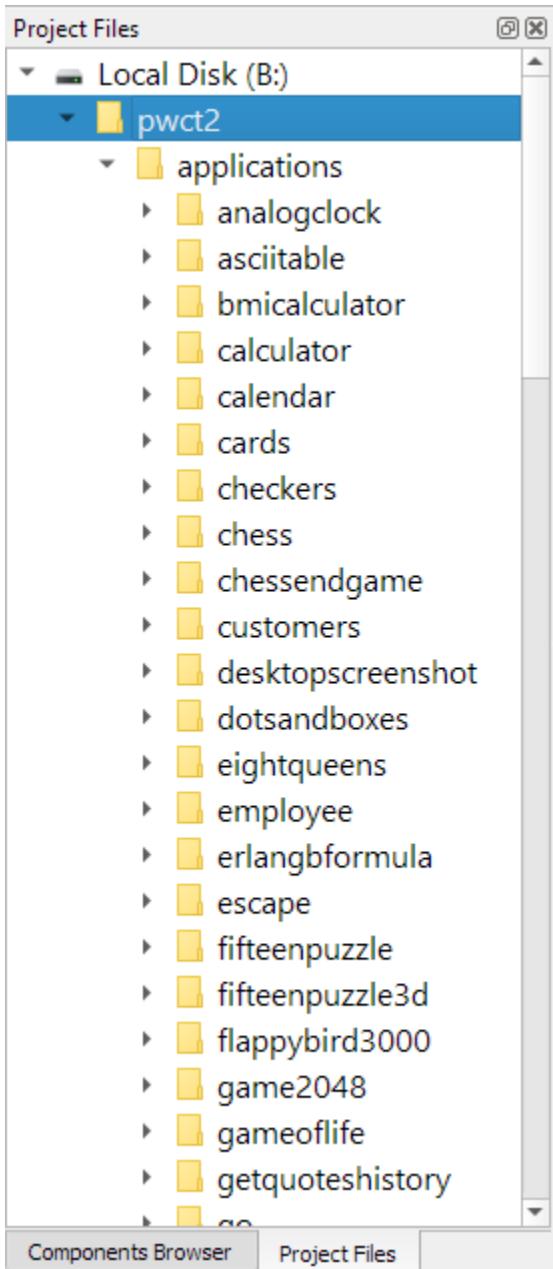
- 5 - We can search in the Components Tree by typing the Component name or some letters in the name
- For example, by typing the letter 'p' we can find the (Print Text) component



6 - To use the component after selecting it, Press (Enter) or (Double Click) using the Mouse

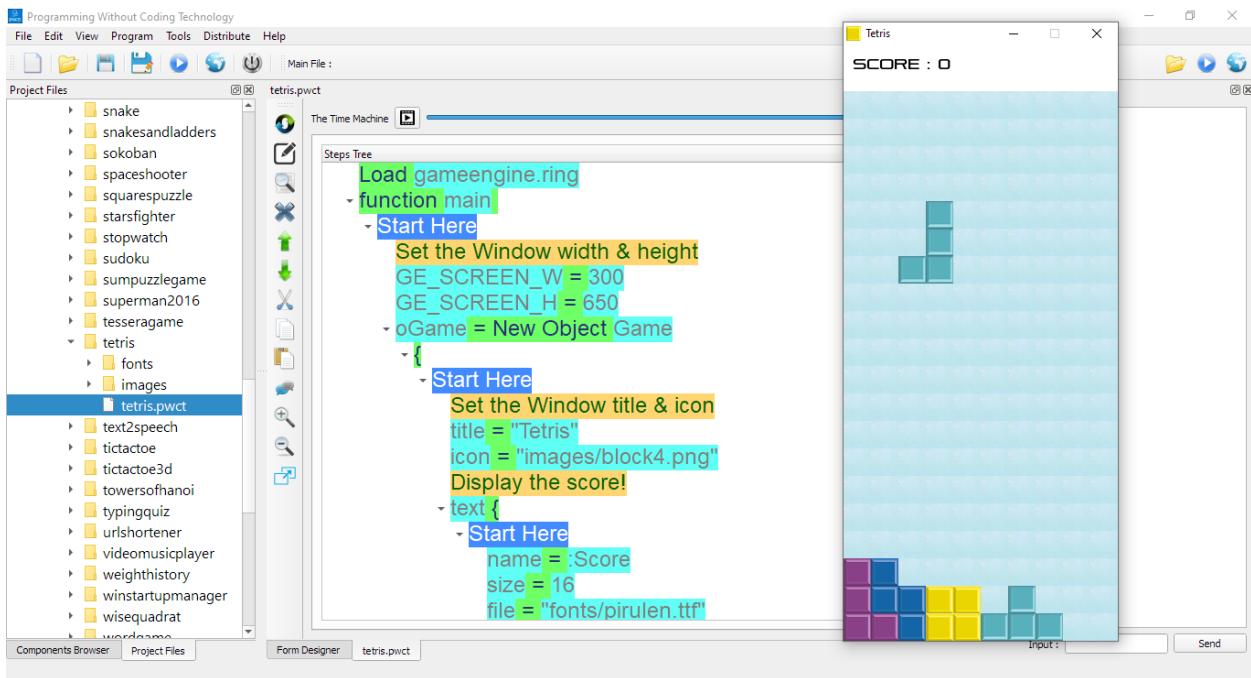
3.1.2 Project Files

Using the Project Files window, We can open visual source files quickly!



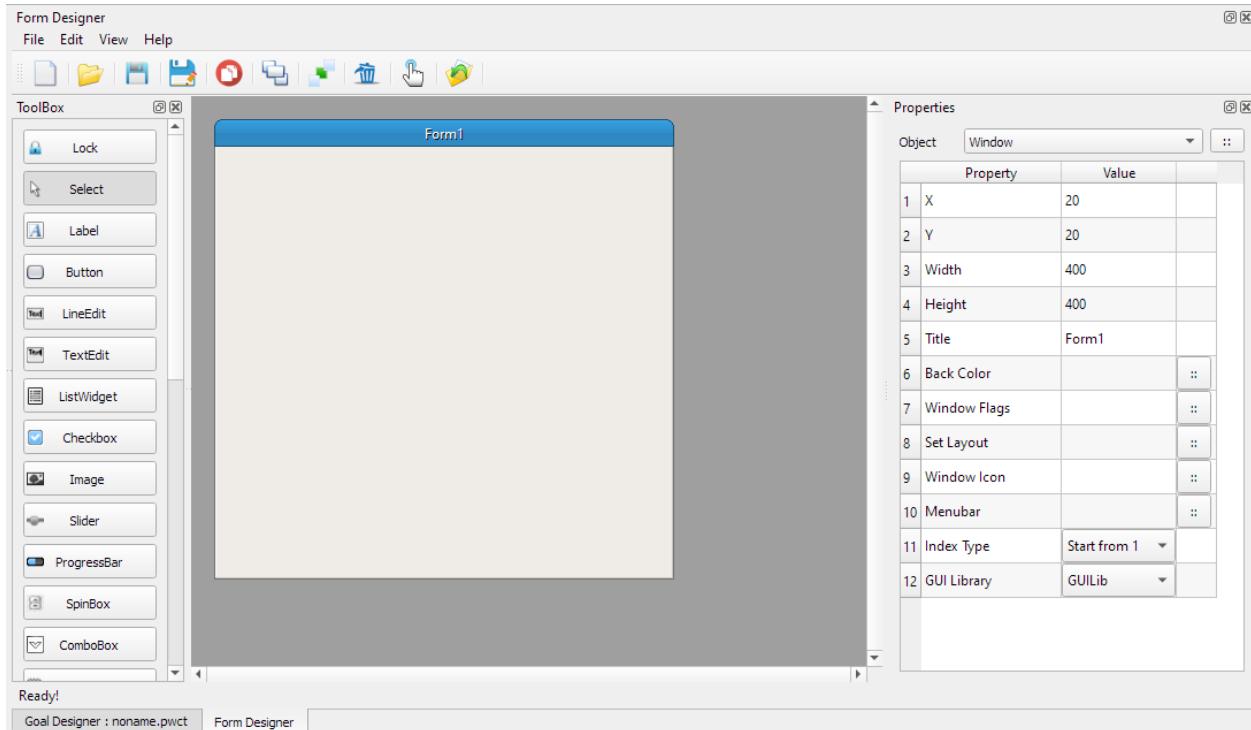
For example, We can open the file (tetris.pwct) which contains the Tetris game!

To run this game we can click on the Run button in the Main Toolbar (Ctrl+F5)

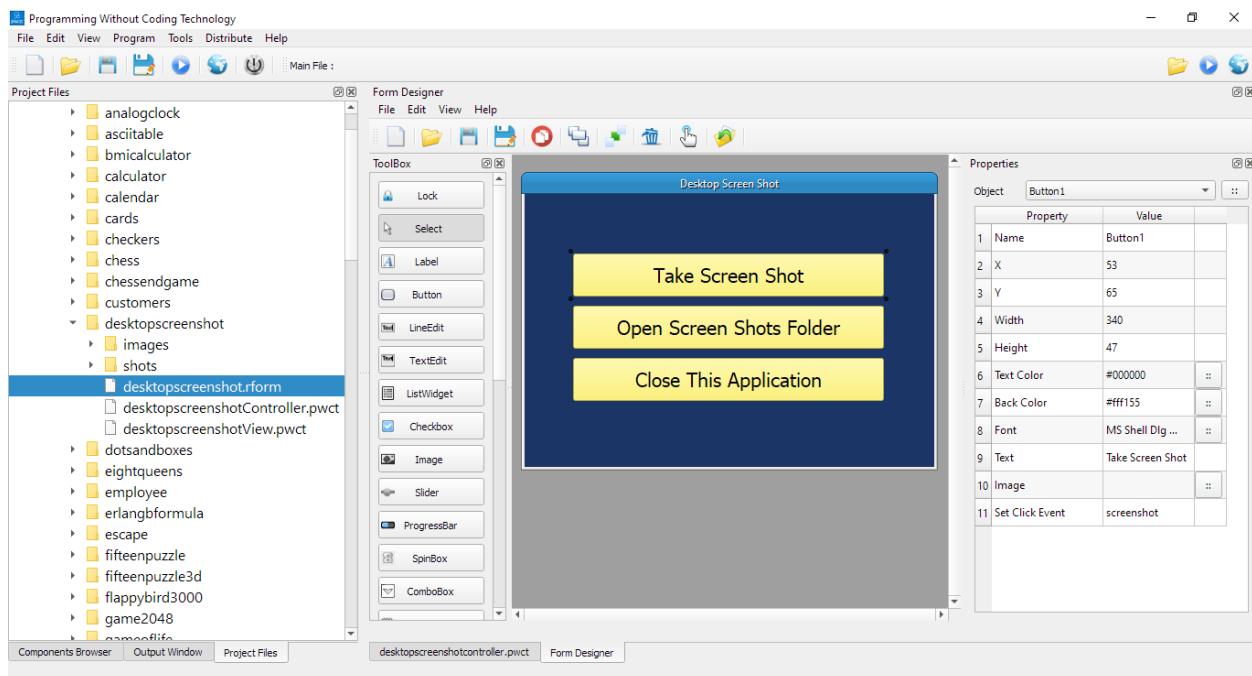


3.1.3 Form Designer

Using the Form Designer, We can design the application forms (User Interface)



For example, The User Interface of the Desktop Screen Shot application is designed using the Form Designer



3.1.4 Output Window

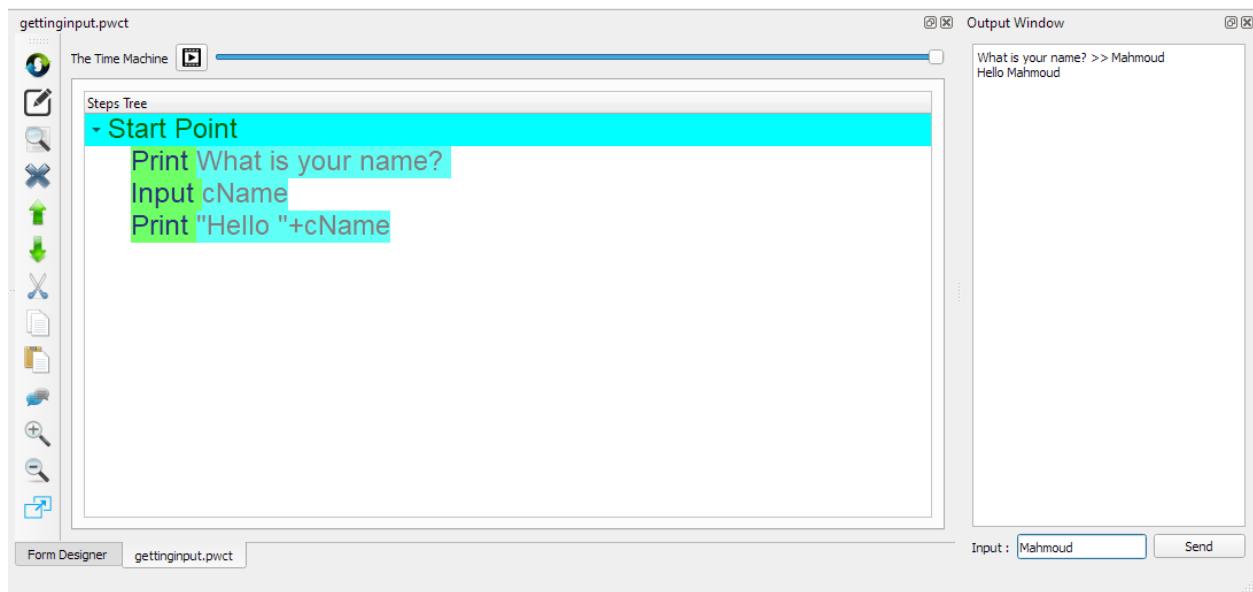
Using the Output Window, We can see the programs output!



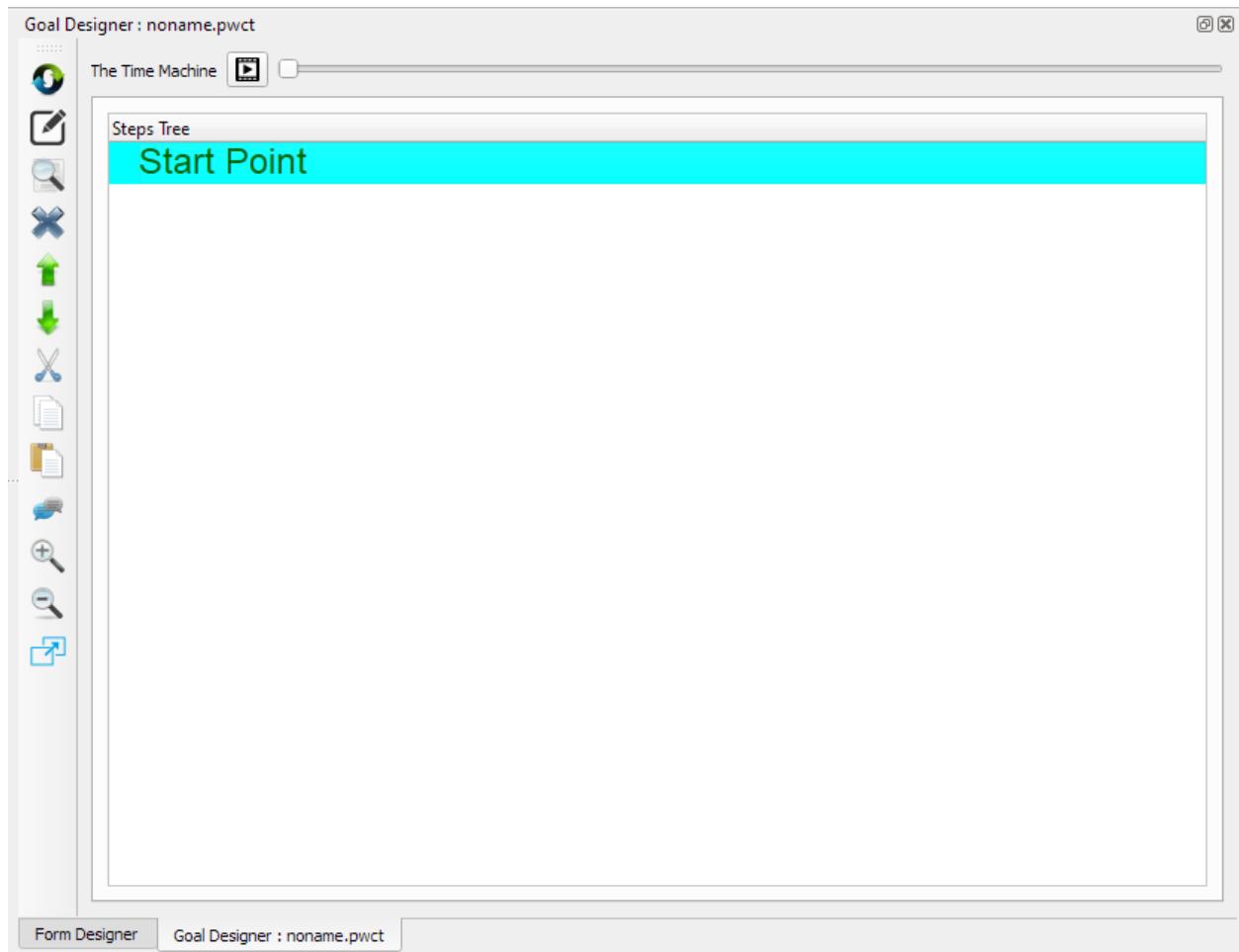
Also we can provide the required input for console applications

For example, The next console application ask about the User Name!

We can type the name in the Textbox then press (Enter) or click on the (Send) button



3.1.5 Goal Designer



- 1 - Contains the Steps Tree that represent the logic behind our program
- 2 - Contains buttons that we can use to control the steps tree
- 3 - Contains the Time Machine
 - 3.1 - Support running the program in the past
 - 3.2 - Support playing the program as movie

3.2 Hello World Program

In this section we will learn how to create the (Hello, World!) program

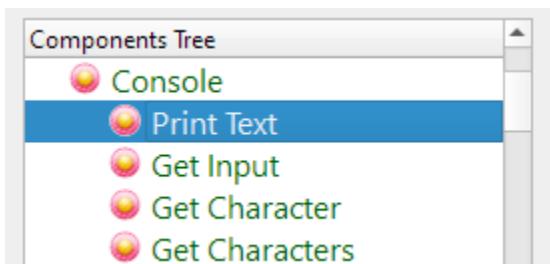
Section Contents:

- (1) How to create the program using the Mouse
- (2) The Steps Summary
- (3) How to create the program using the Keyboard Shortcuts

3.2.1 Hello World Program (Using the Mouse)

From the Components Browser, Select the (Print Text) component

Then double click using the Mouse

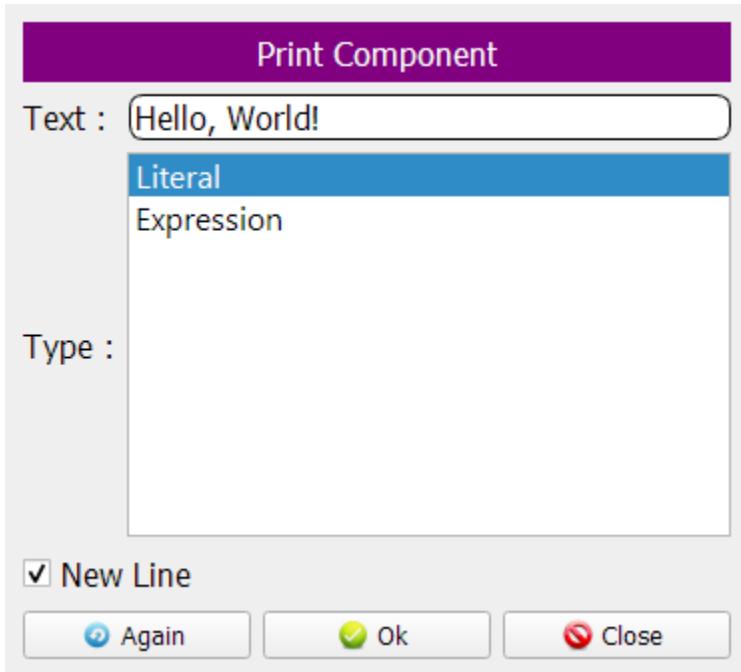


After selecting the component, an interaction page will appear in the Goal Designer

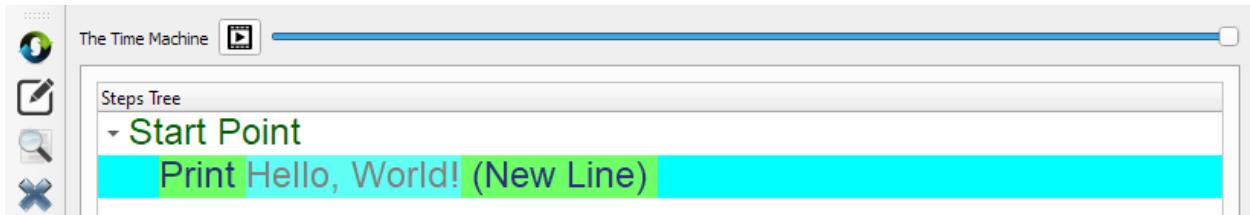
Using this Interaction Page we can determine the text that will be printed on the screen

Type: Hello, World!

Then click the (Ok) button

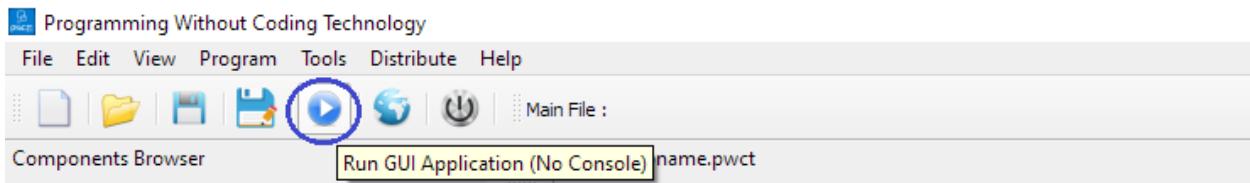


This will generate the next step in the Goal Designer Window

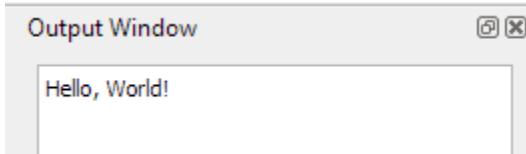


Now we can run the program and see the program output

To run the program click on the (Run) button from the Main Toolbar

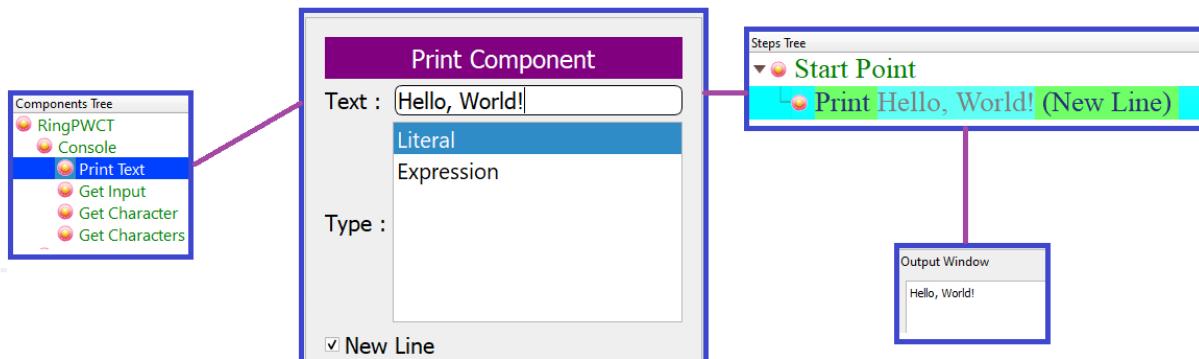


The next output will appear in the Output Window



3.2.2 Hello World Program (Steps Summary)

1. Select (Print Text) from the Components Browser
2. Enter (Hello, World!) then click (Ok) in the Interaction Page
3. Click (Run) from the Main Toolbar



3.2.3 Hello World Program (Using the Keyboard)

- Inside the Goal Designer window press the letter ‘p’
- The Search Textbox will be active, and the pressed letter will be written there
- This will select the (Print Text) component
- Press (Enter) to use the (Print Text) Component
- Write (Hello, World!) then press CTRL+W (Similar to clicking on the OK button)
- Press CTRL+F5 to run the program!

Tip: Pressing any letter inside the Goal Designer will move us to the Components Browser

3.3 Using the Goal Designer

In this section we will learn about using the Goal Designer

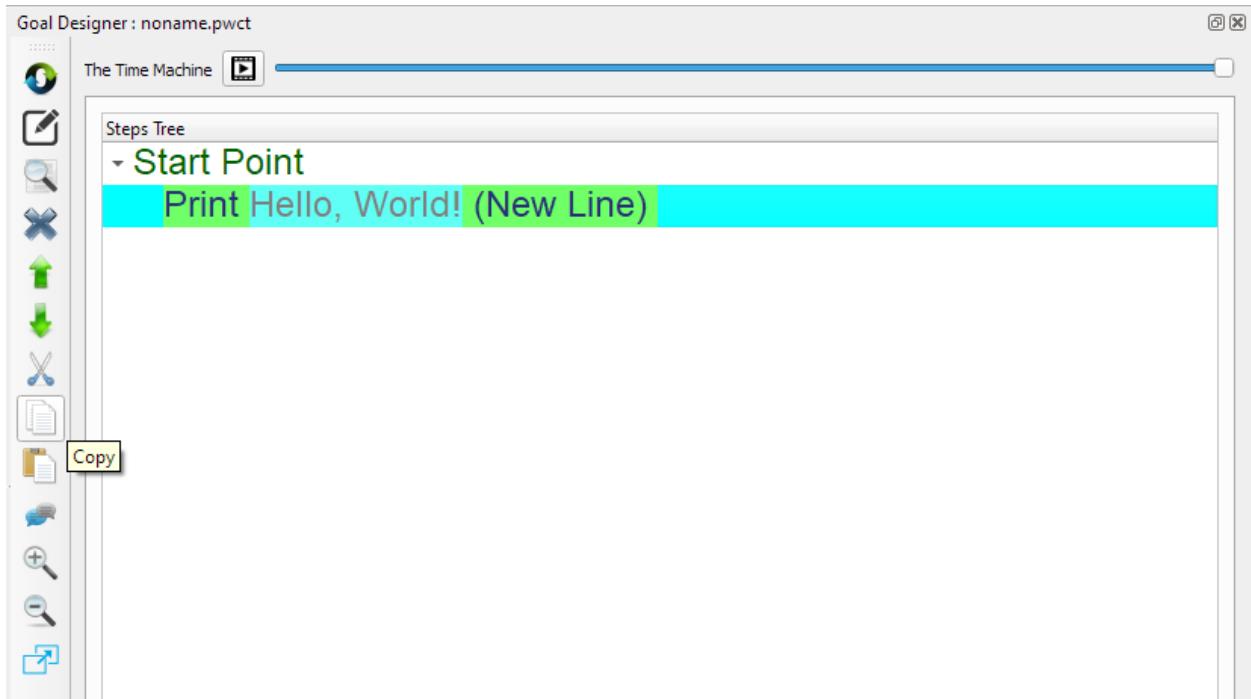
Section contents:

- Copy & Paste the Steps
- Modify the Steps
- Using the Time Machine
- Moving Steps Up & Down
- Cut & Paste Steps
- Inserting Steps
- Comment/Uncomment Steps
- Deleting Steps

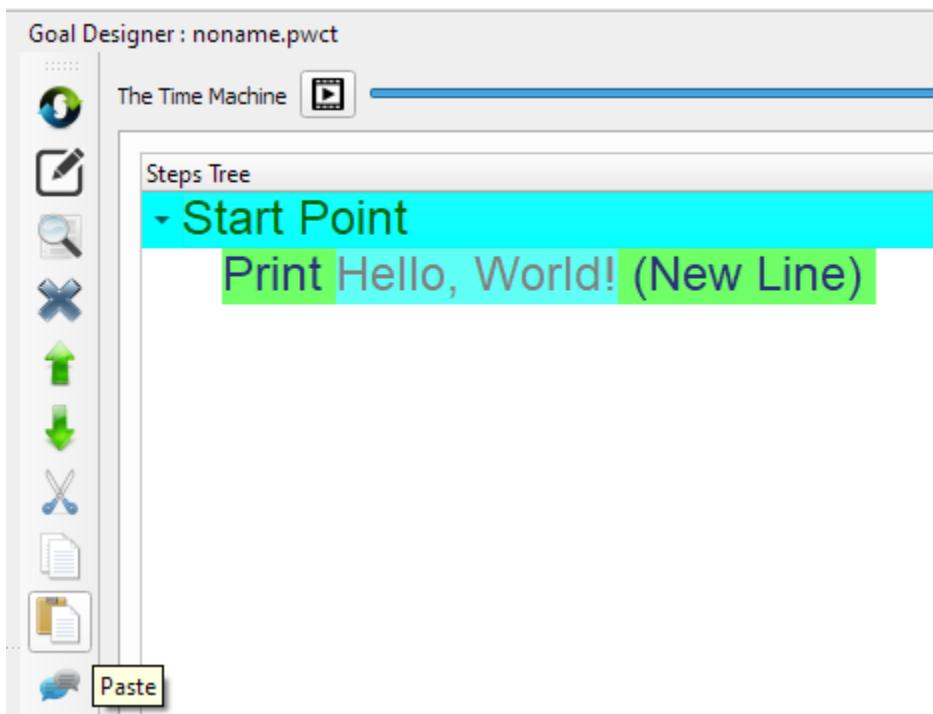
- Search and Replace
- Using the Again Button

3.3.1 Copy & Paste the Steps

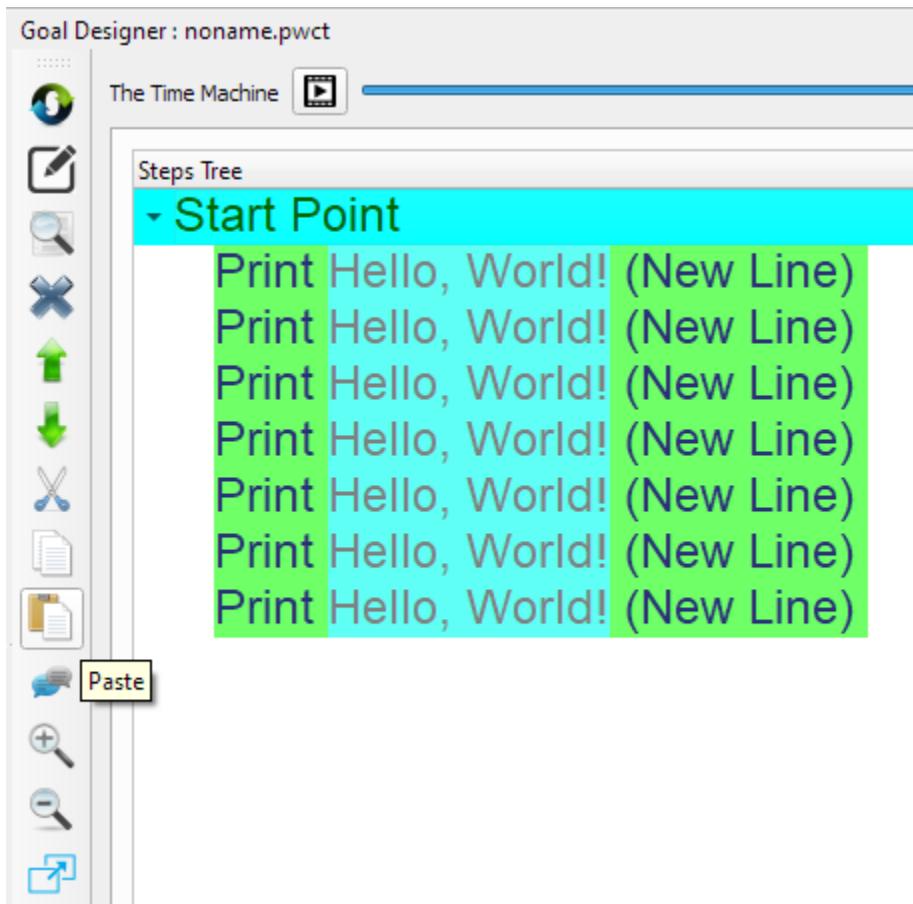
Select the step (Print Hello, World!) then click on the (Copy) button or press Ctrl+C



Now select the Start Point then click on the (Paste) button or press Ctrl+V

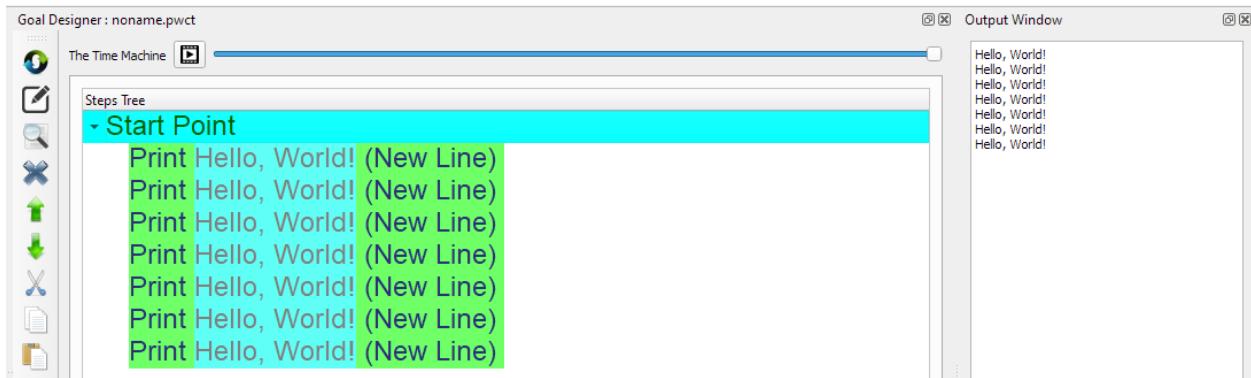


We can do the paste operation many times



Now when we run the program using Ctrl+F5 we can see the (Hello, World!) message printed seven times

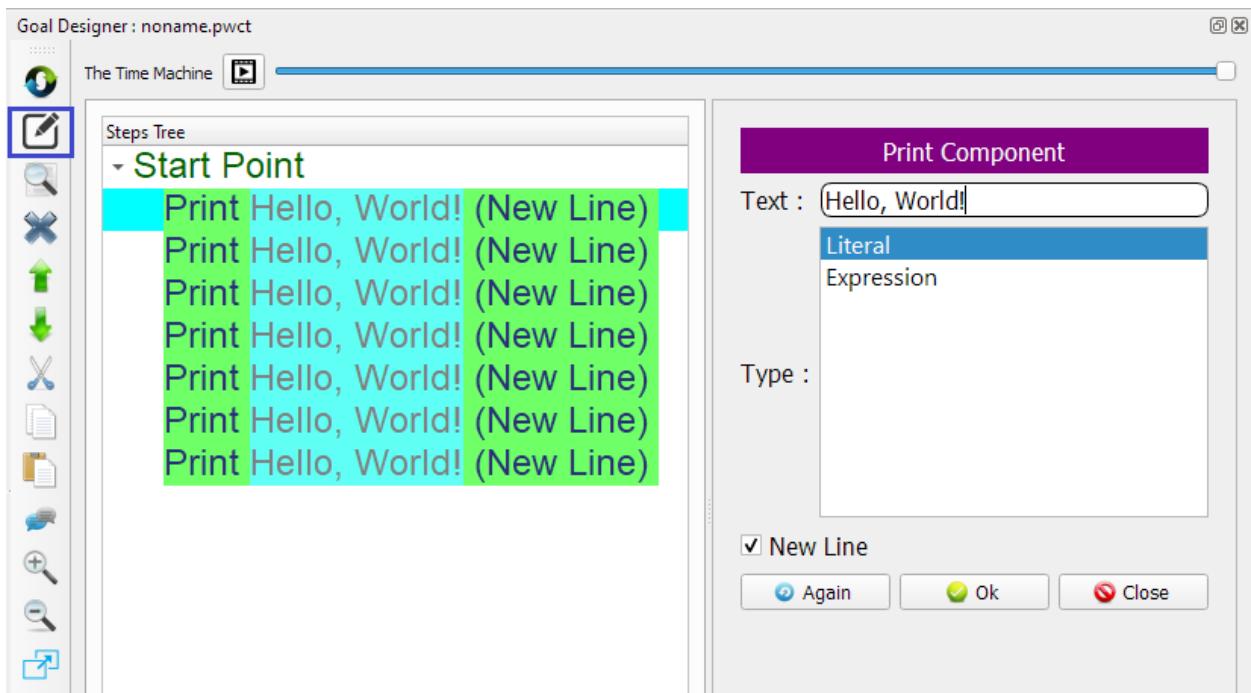
Sure it's not the right way to program something like this because it's better to use a For-Loop



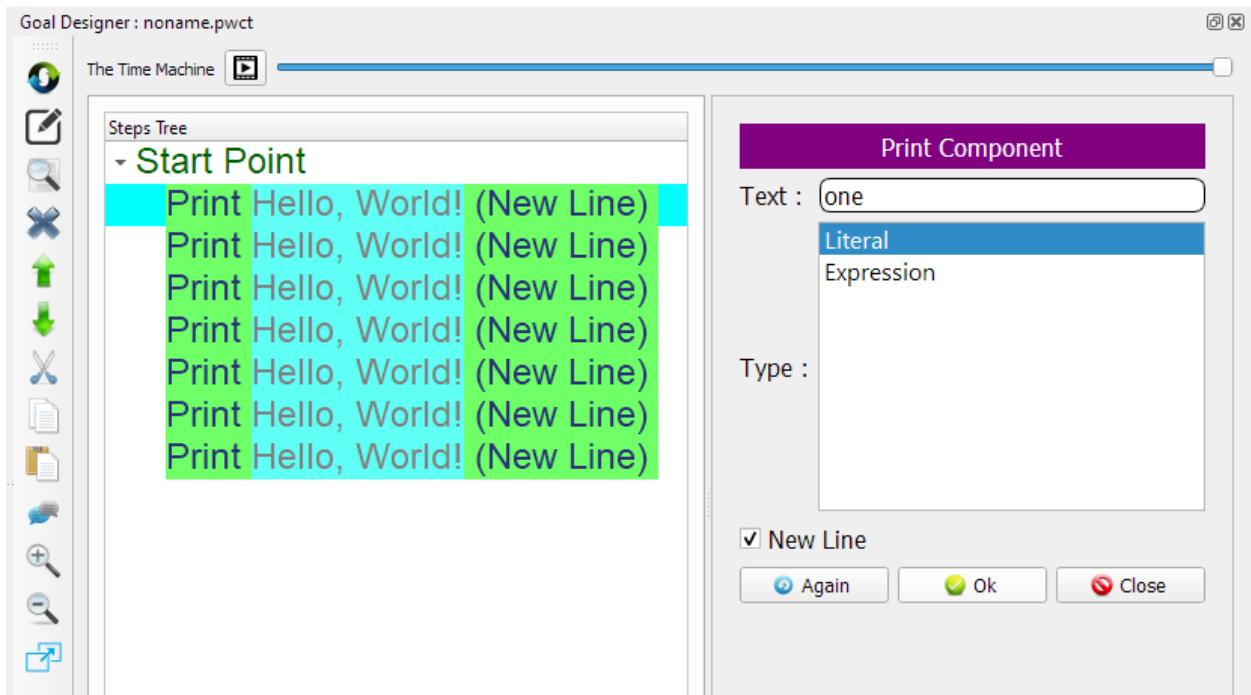
3.3.2 Modify the Steps

To modify a step, We can click on the (Modify) button or press Ctrl+E or double click on the step.

Select the first step, Then press Ctrl+E

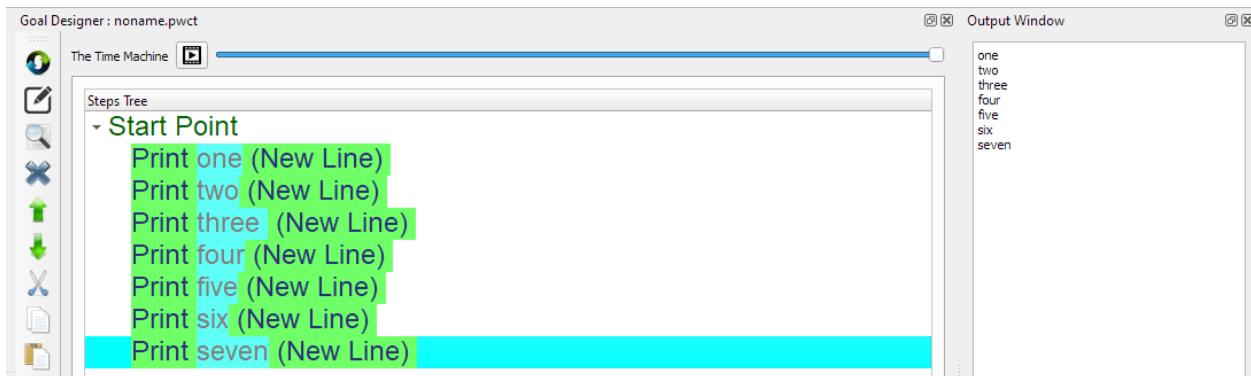


Change the Text from (Hello, World!) to (one) then press Ctrl+W



Change all of the steps to print the numbers from (one) to (seven)

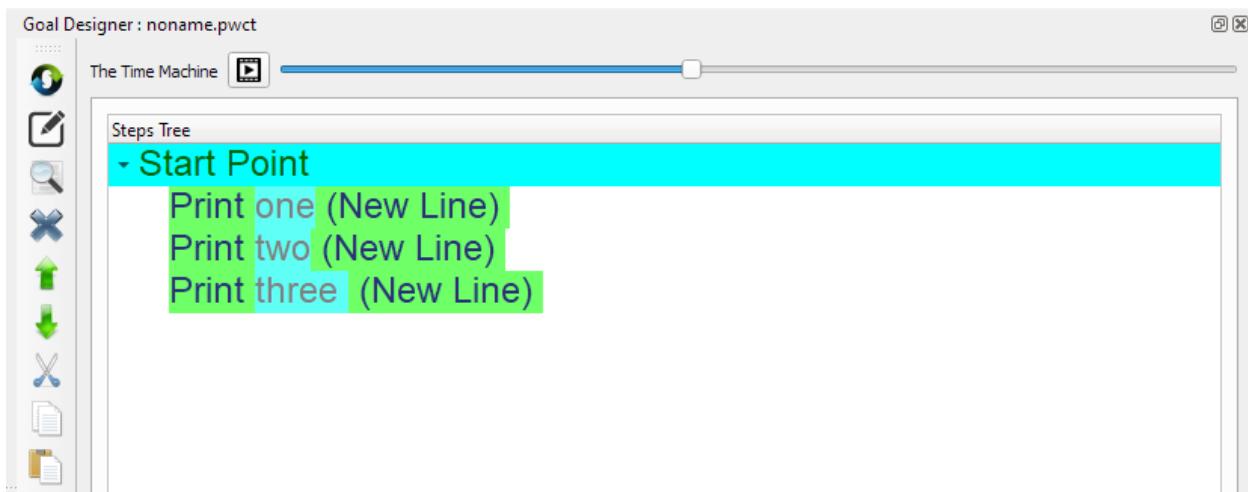
Then run the program using Ctrl+F5



3.3.3 Using the Time Machine

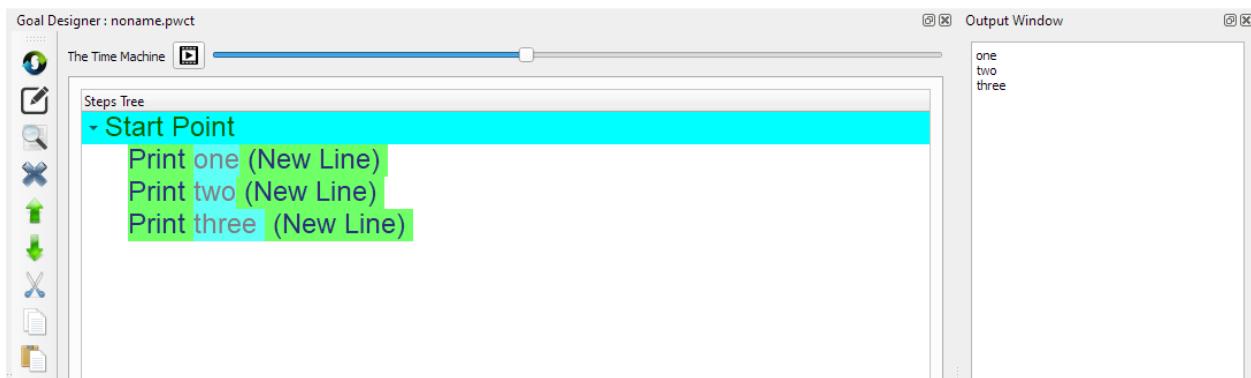
Inside the Goal Designer and using The Time Machine Slider we can move backward along the Time Dimension

For example, At this point in the past we have three steps only



We can run the program in the past using Ctrl+F5

This will print (one two three) - Each word in separate line

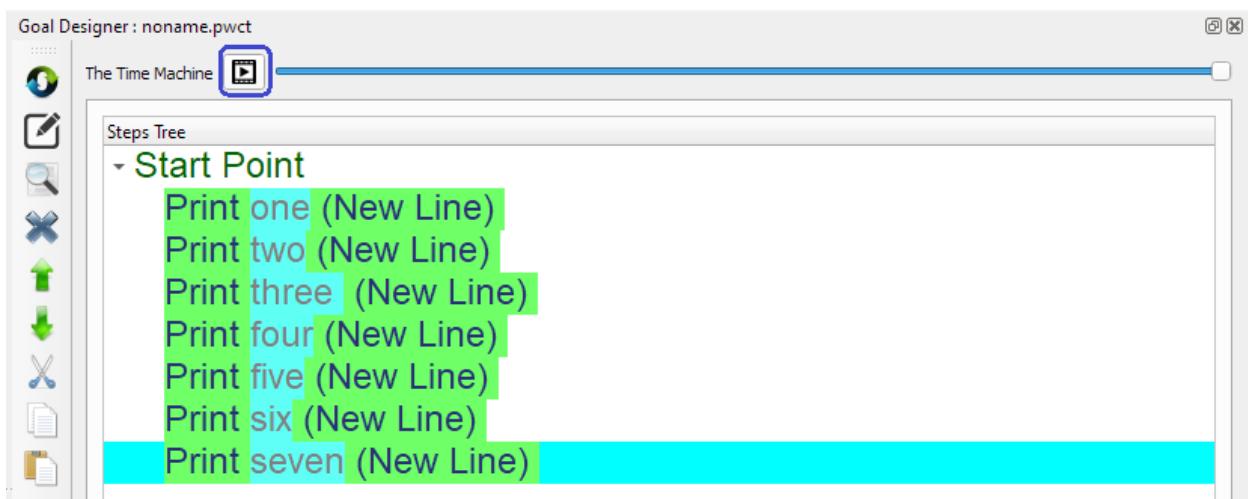


We can move forward and go to the present where we have all of the seven steps

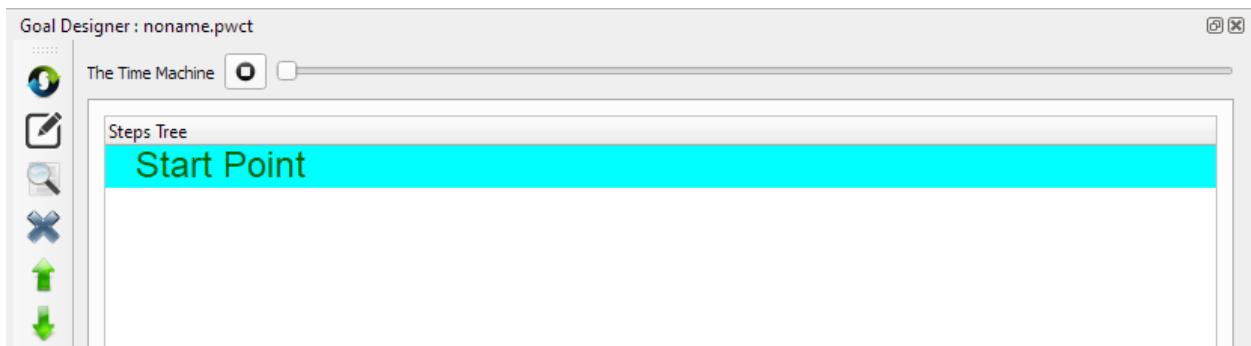
Also we can click on the (Play) button to play the program as movie

This will create the program step-by-step to learn how to generate each step

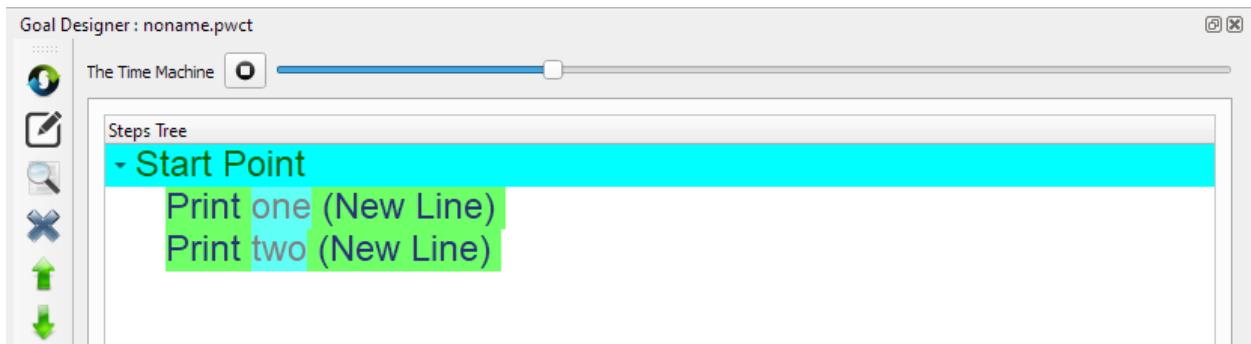
This will display the selected component in the Components Browser and the Interaction Pages too!



For example, at this point in the past, Nothing exist!



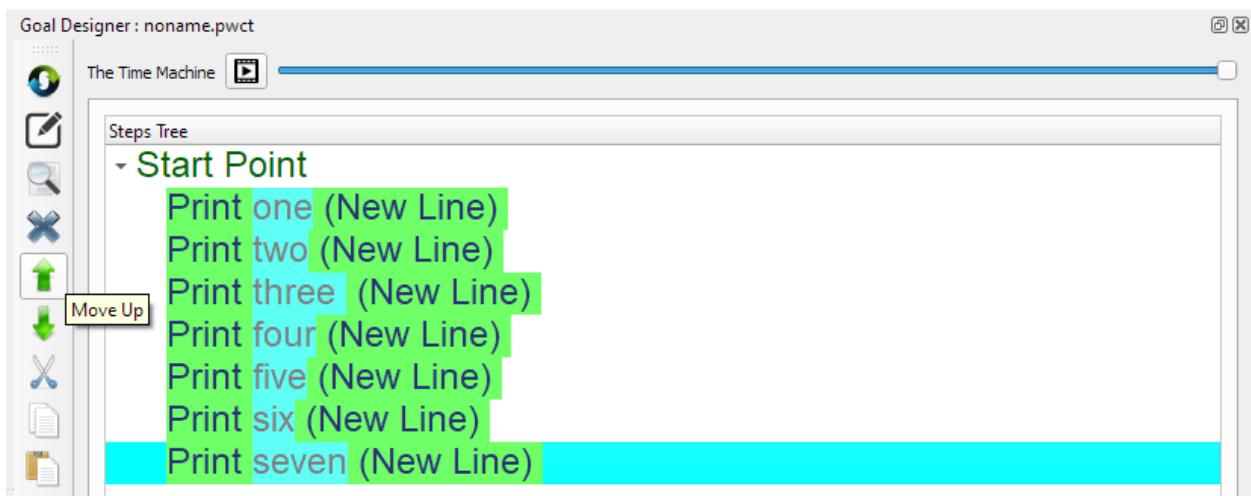
While at this point we have two steps



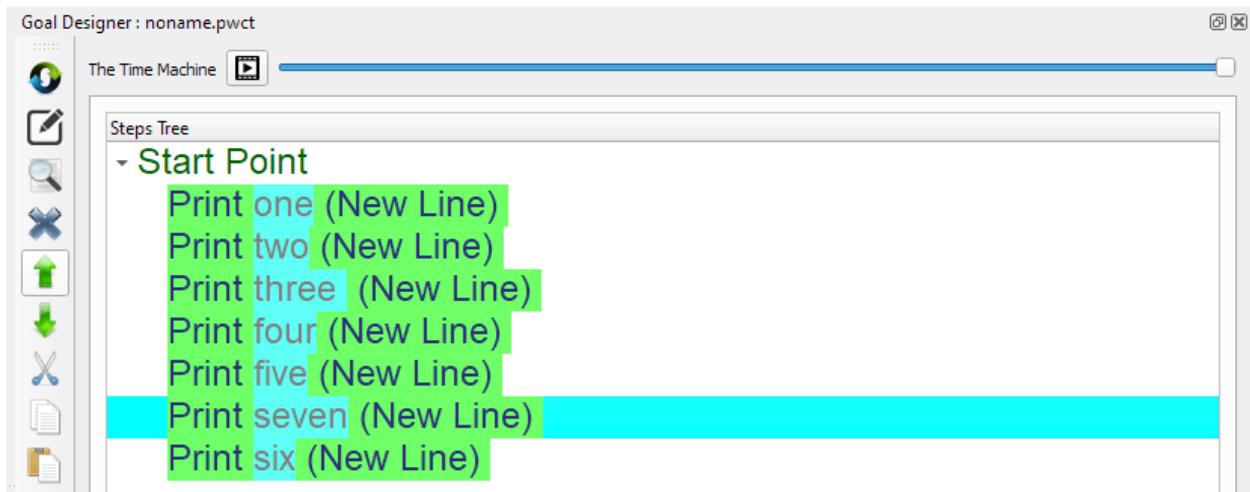
3.3.4 Moving Steps Up & Down

We can change the order of the steps inside the Steps Tree using the Up & Down buttons

For Example, Select the step (Print seven) then click on the Up button or press Ctrl+U

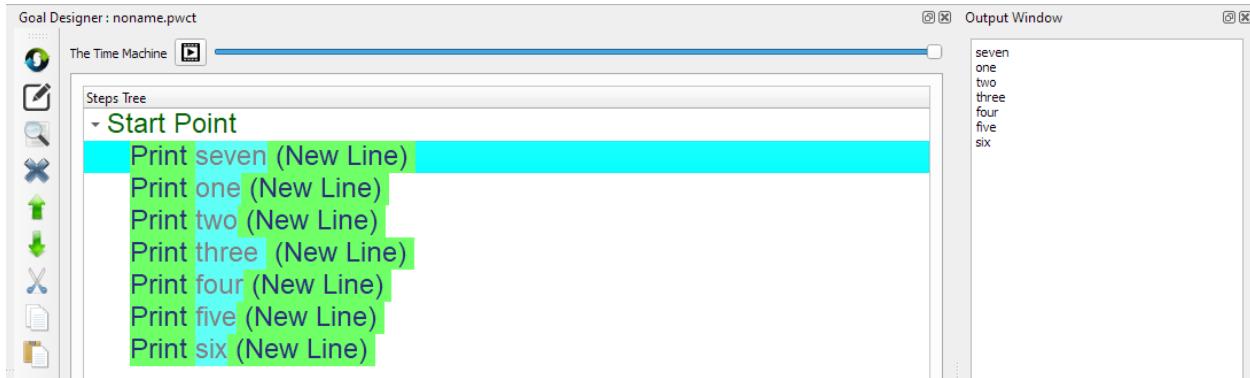


Now we see the (Print seven) comes before (Print six)



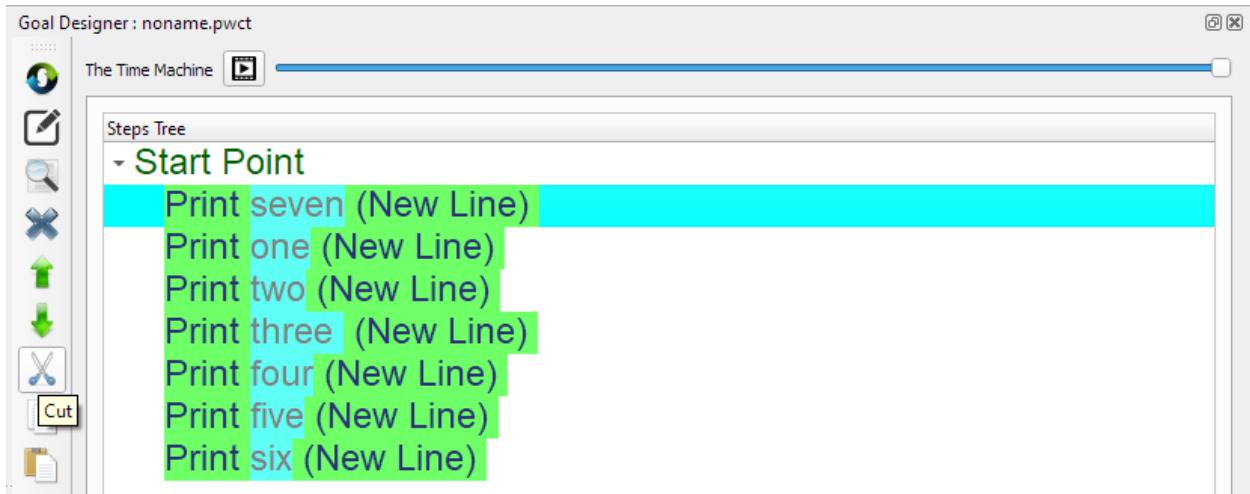
Click on the Up button many times until the (Print seven) step becomes the first step

Then run the program using Ctrl+F5

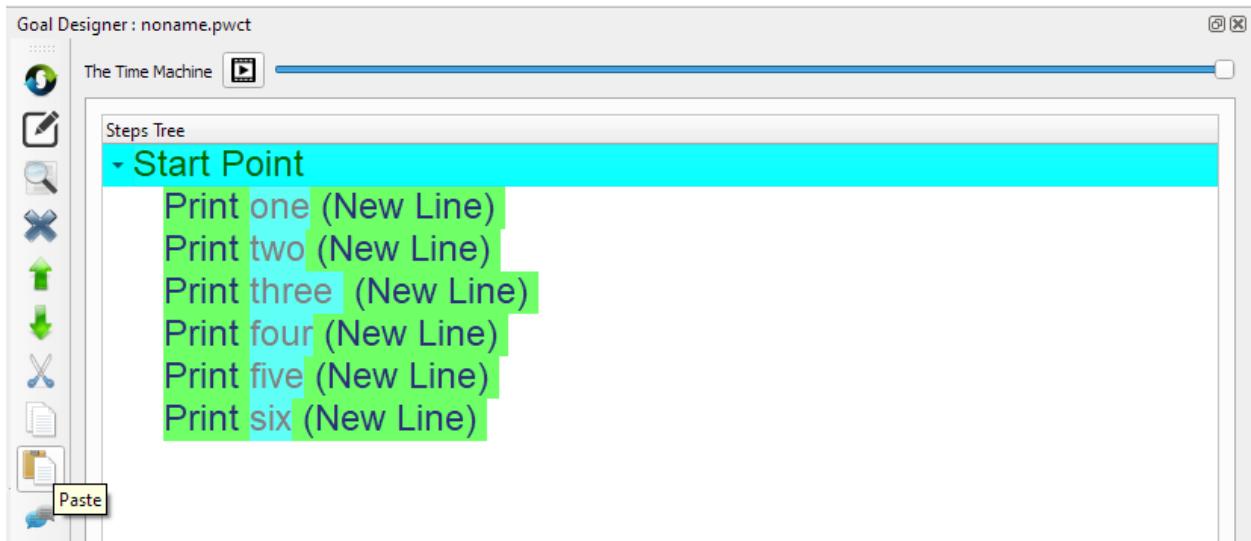


3.3.5 Cut & Paste Steps

Select the (Print seven) step then click on the (Cut) button or press Ctrl+X

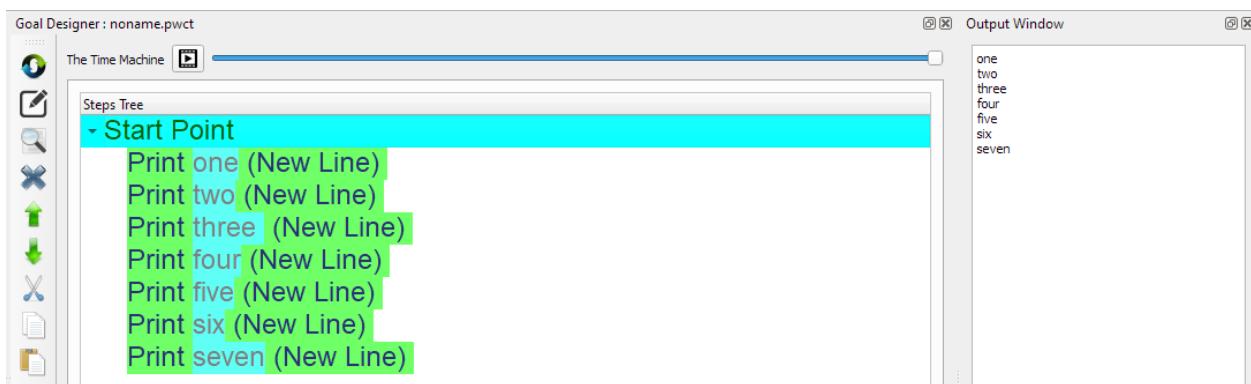


Select the (Start Point) then click on the (Paste) button or press Ctrl+V



This will paste the (Print seven) step after the (Print six) step

Run the program using Ctrl+F5

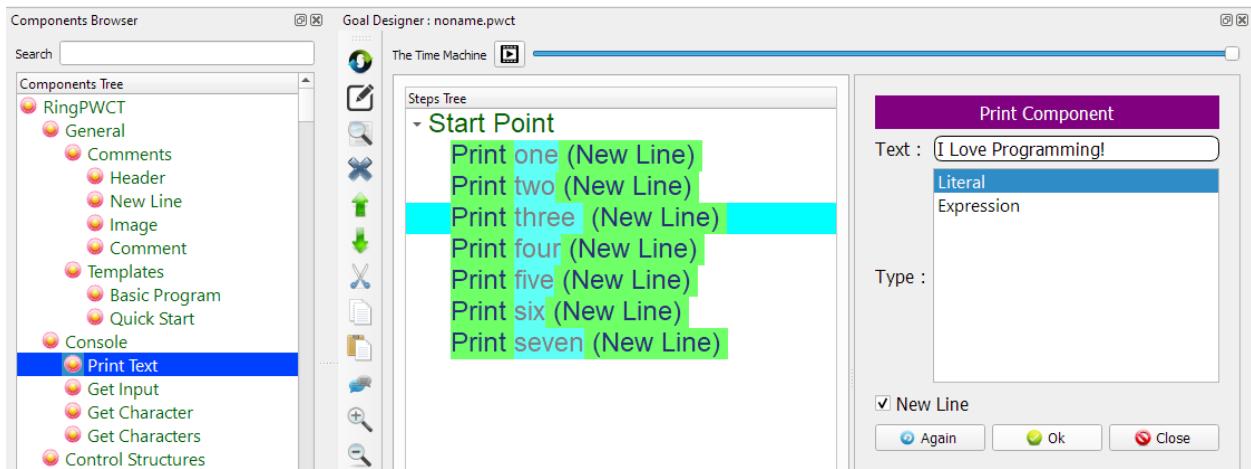


3.3.6 Inserting Steps

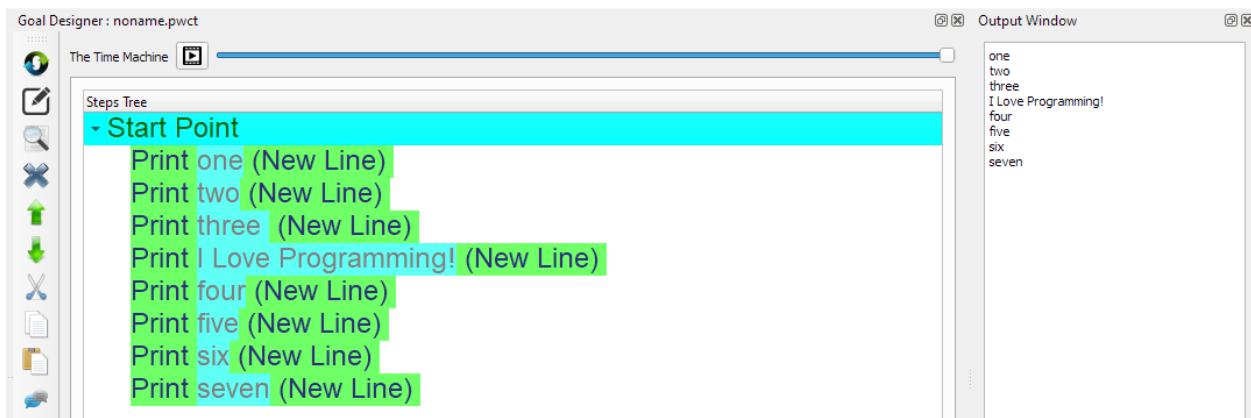
To insert a new step after the (Print three) step

At first select the (Print three) step then start a new interaction process

Start using the (Print Text) component

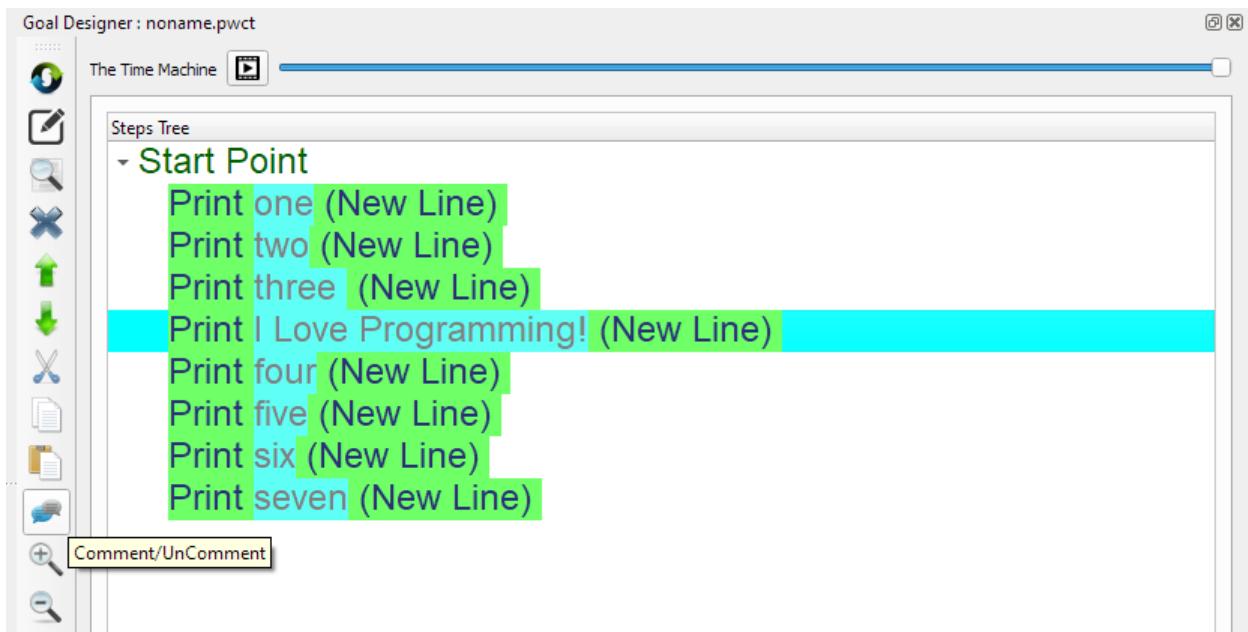


The new generated step will be added directly after the (Print three) step

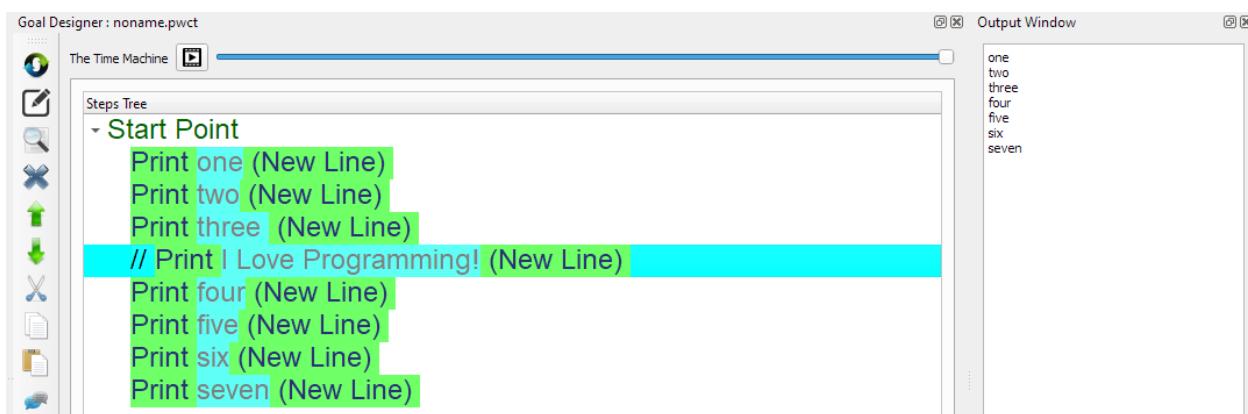


3.3.7 Comment/Uncomment Steps

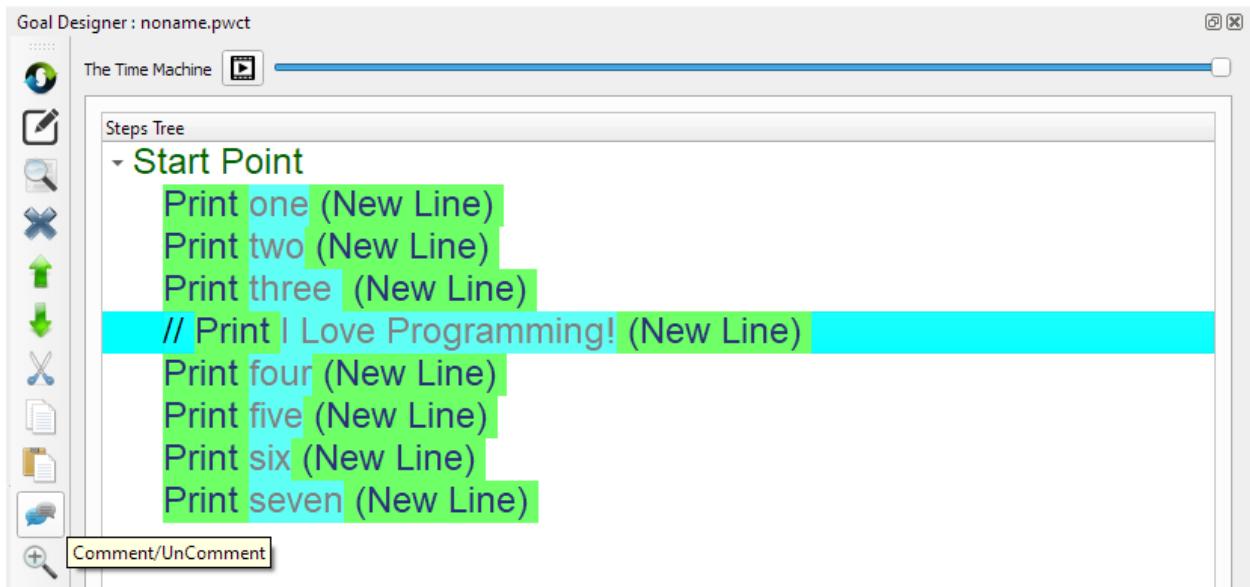
To comment a step, Select it first then click on the (Comment/Uncomment) button or press Ctrl+I



The commented step will not be executed when we run the program

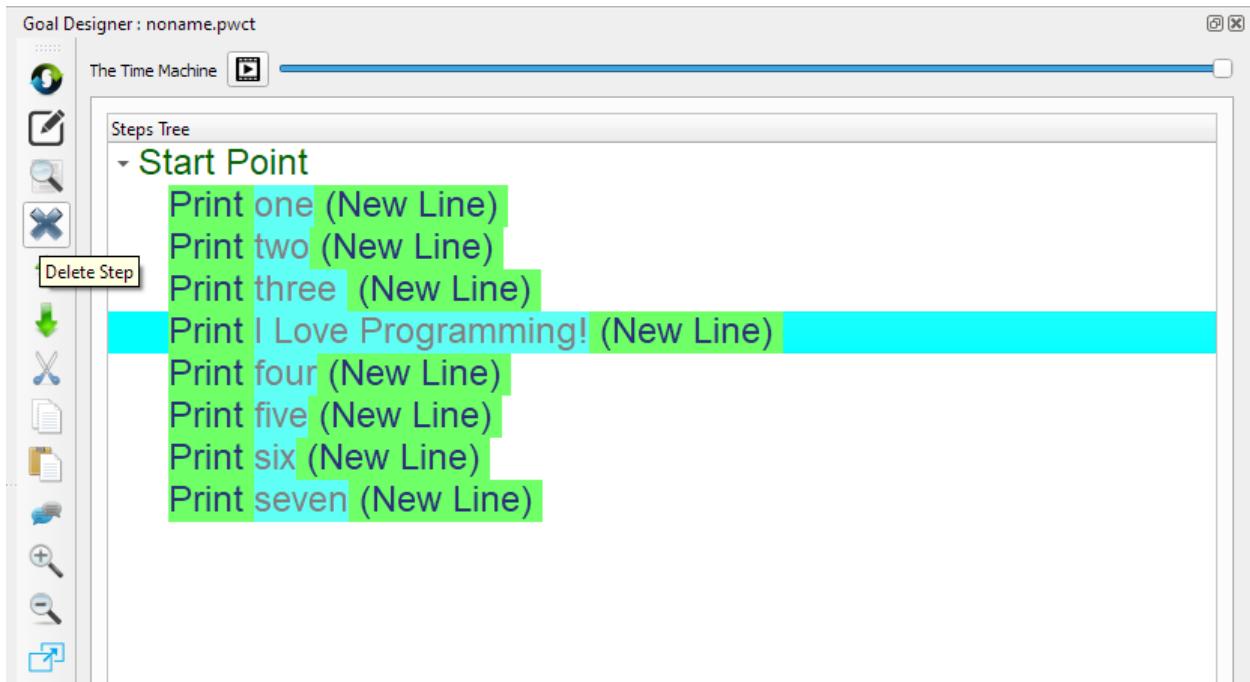


To enable the step again (Uncomment), select it then click on the (Comment/Uncomment) button

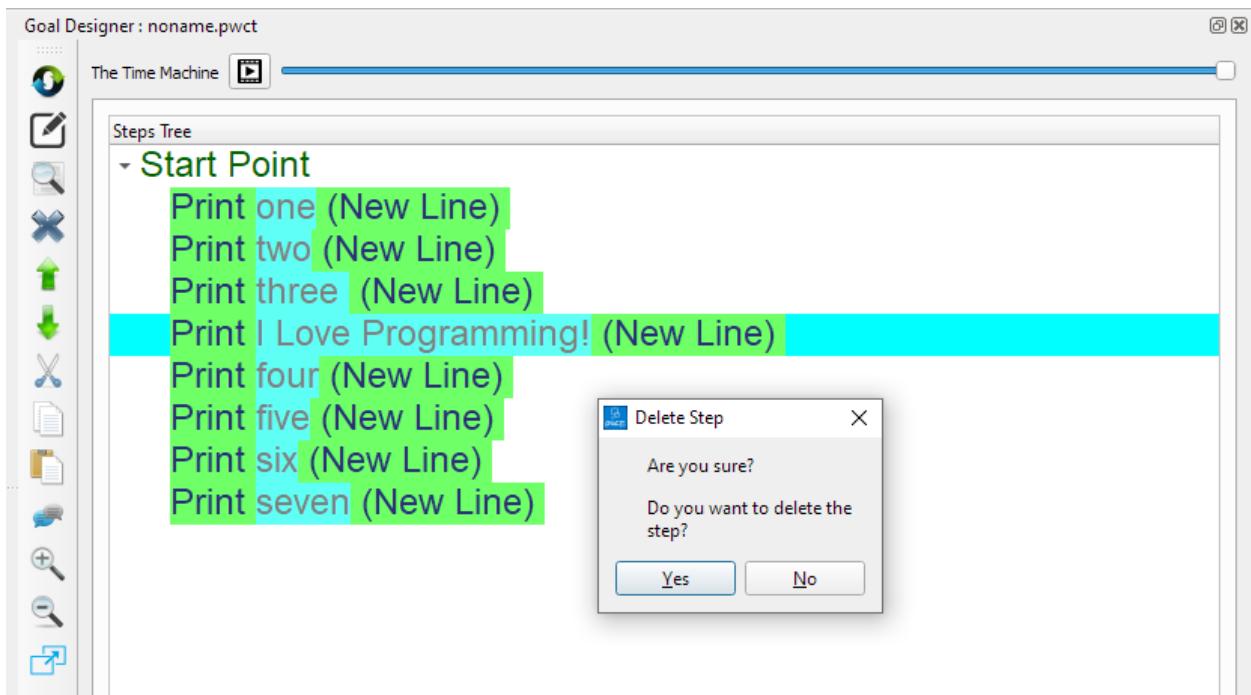


3.3.8 Deleting Steps

To delete a step, select it then click on the (Delete) button

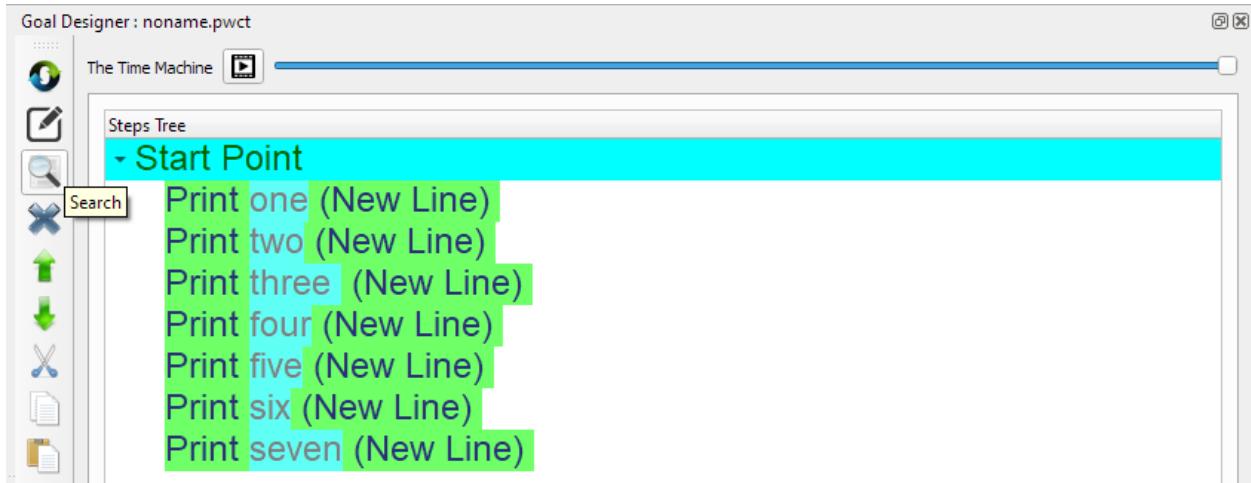


Click (Yes) or press (Enter) to delete the step



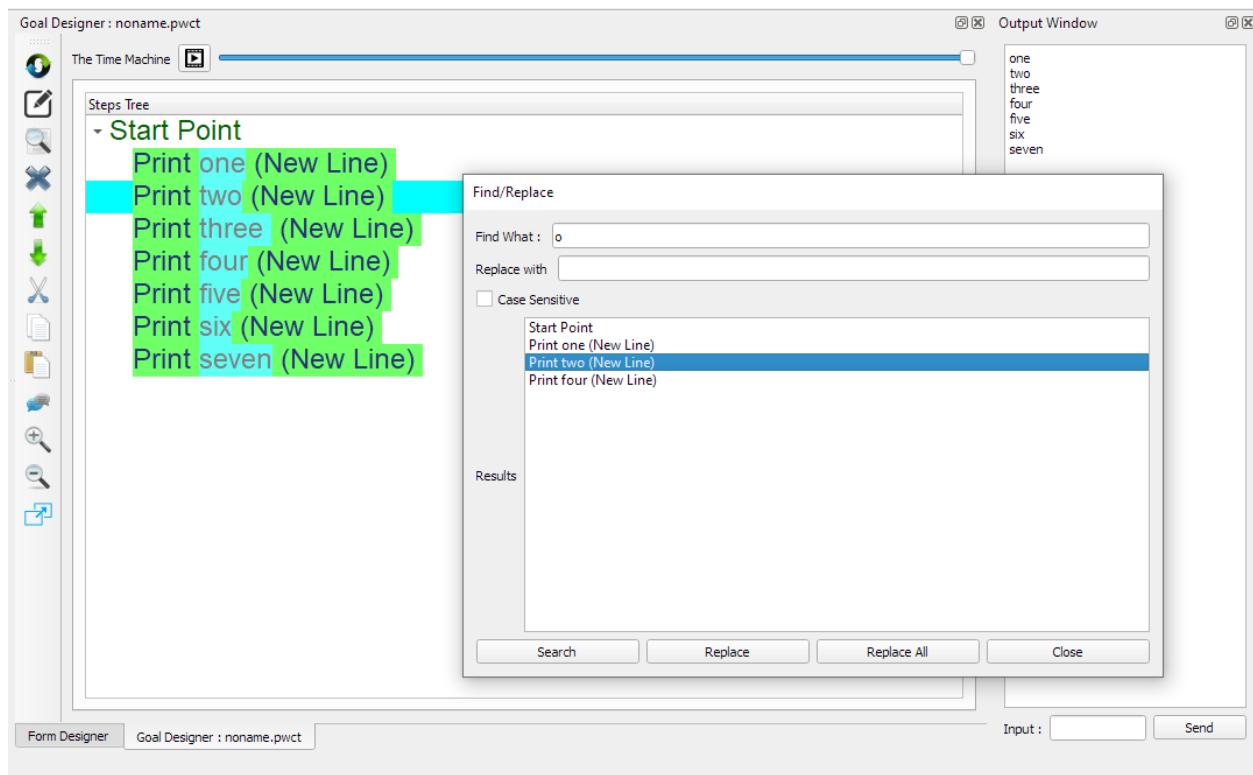
3.3.9 Search and Replace

To find a step, click the (Search) button or press Ctrl+F

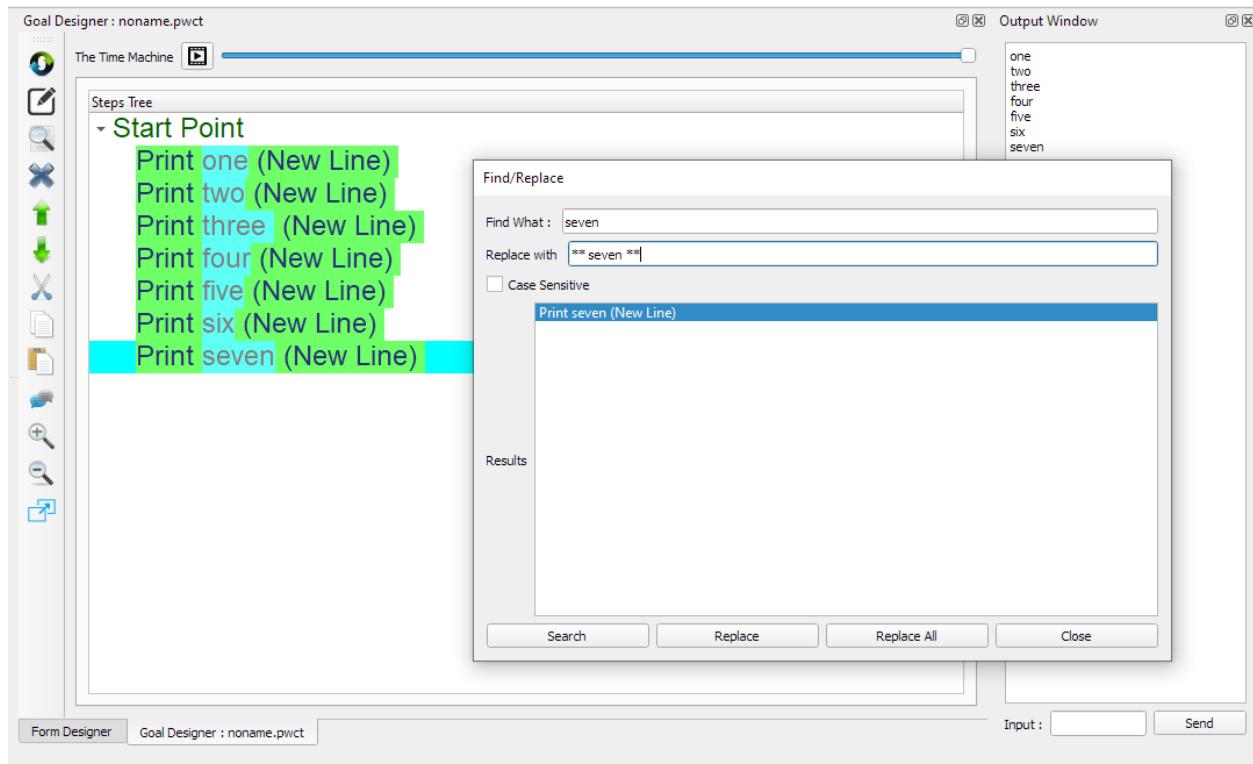


For example, In the Search window type the letter “o” then click (Search) or press Enter

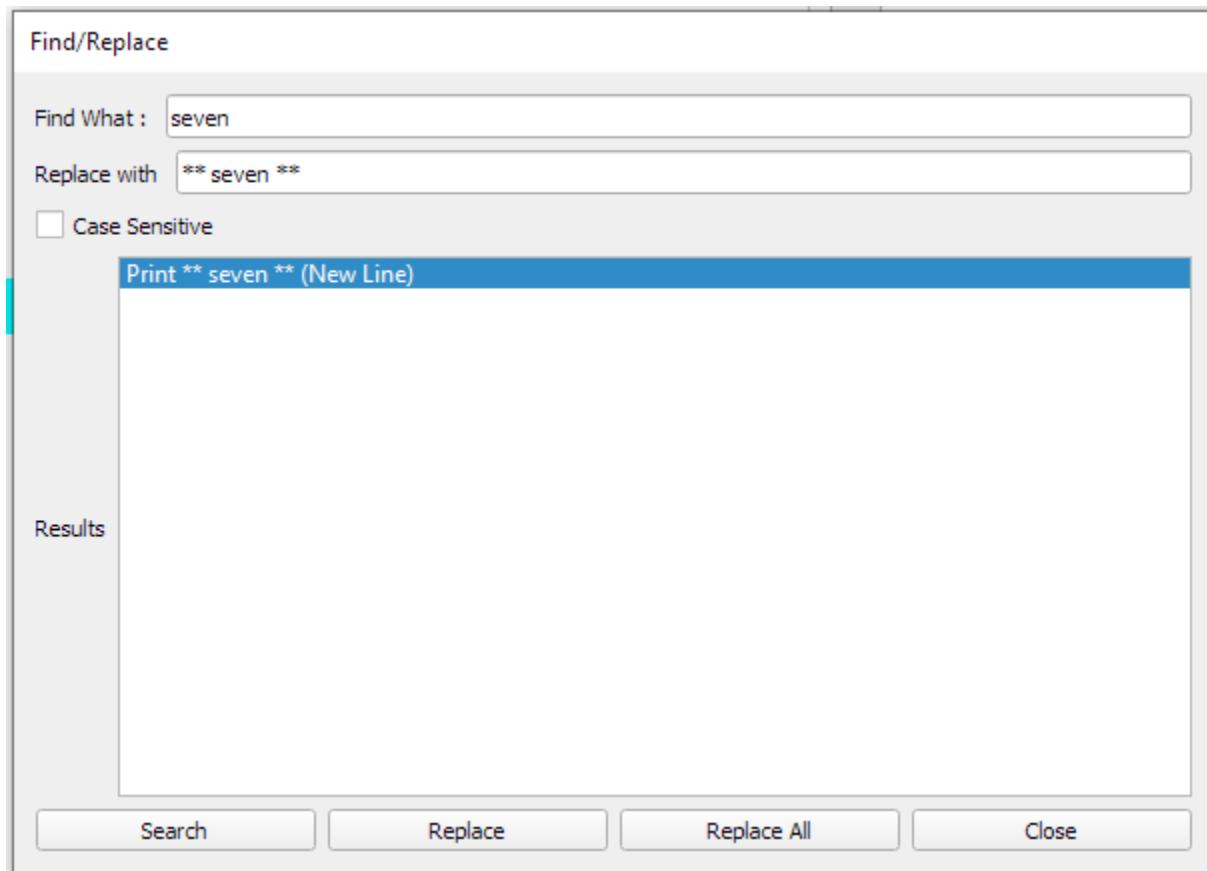
Once we select a step from the Results Listbox, this step will be the active step in the Steps Tree



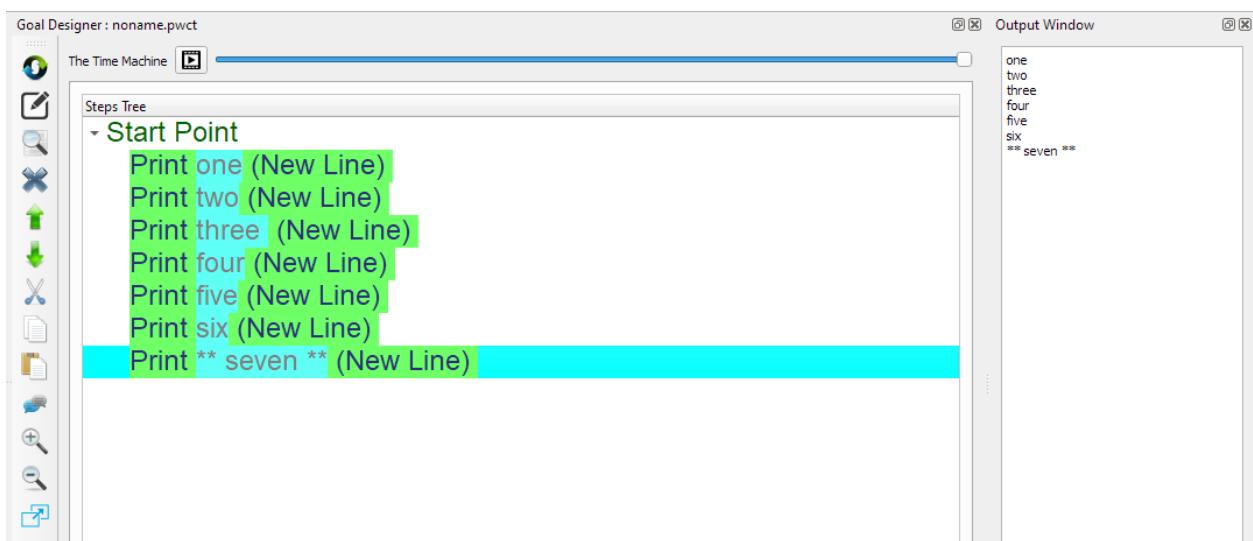
We can replace the text with another text



For example replace (seven) with (** seven **) then click the (Replace) button



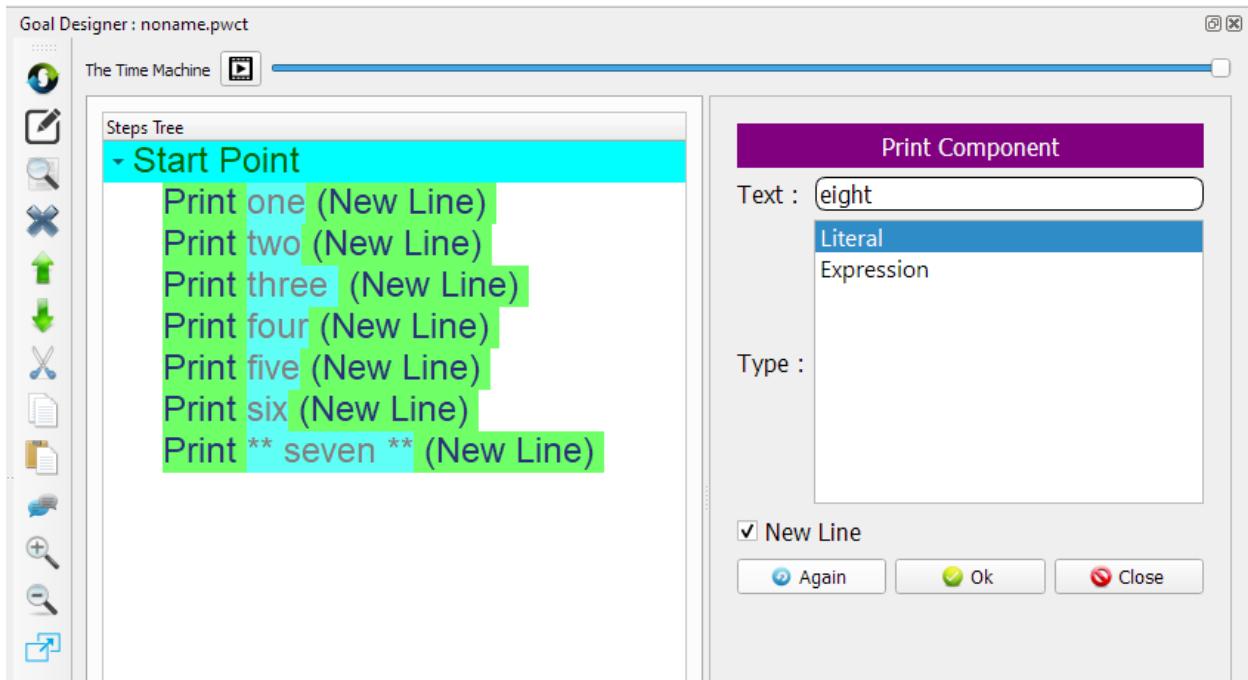
Then run the program using Ctrl+F5 to see the change in the output



3.3.10 Using the Again Button

We can use the same component many times using the (Again) button

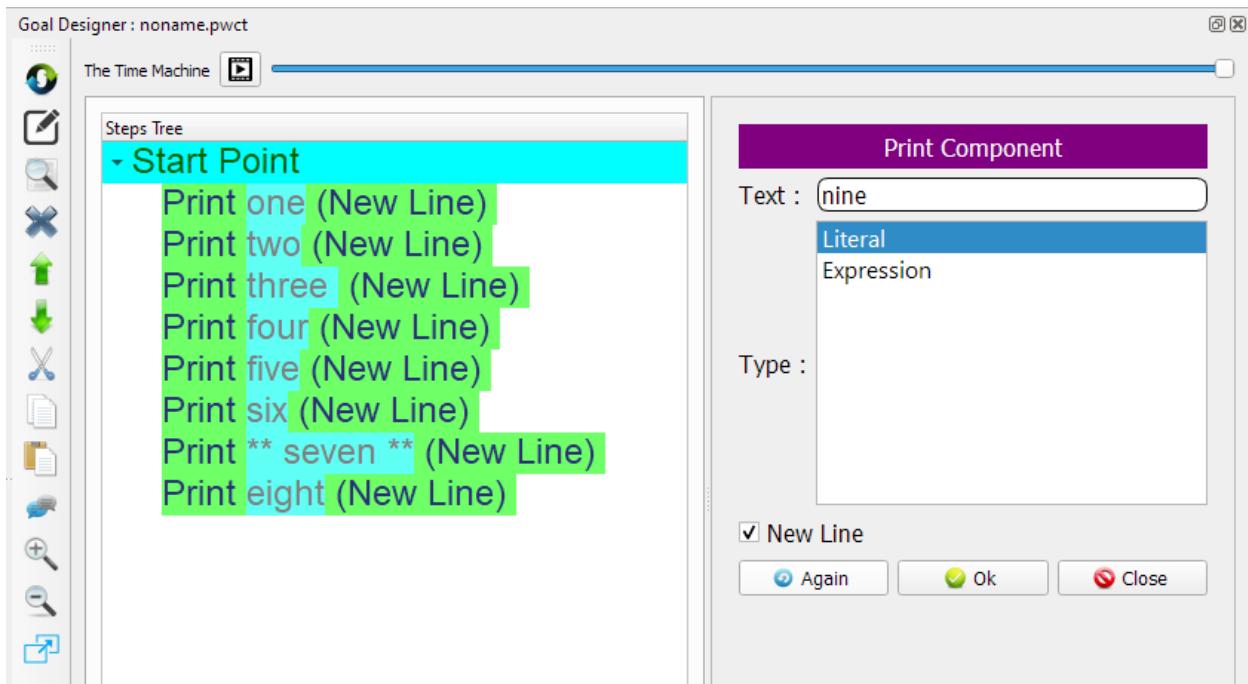
For example, when we use the (Print Text) component, and after writing the text (eight) click (Again) instead of (Ok)



This will generate the step (Print eight) in the steps tree

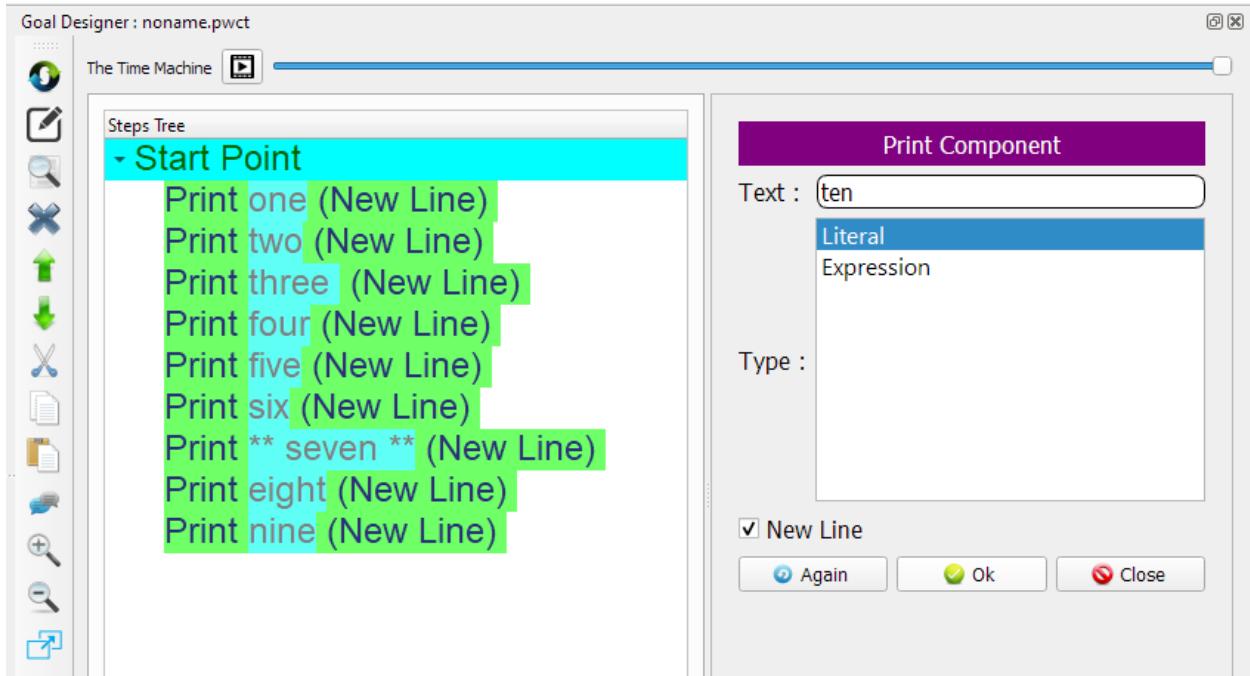
Also we can use the interaction page to write another text like (nine)

Click (again)

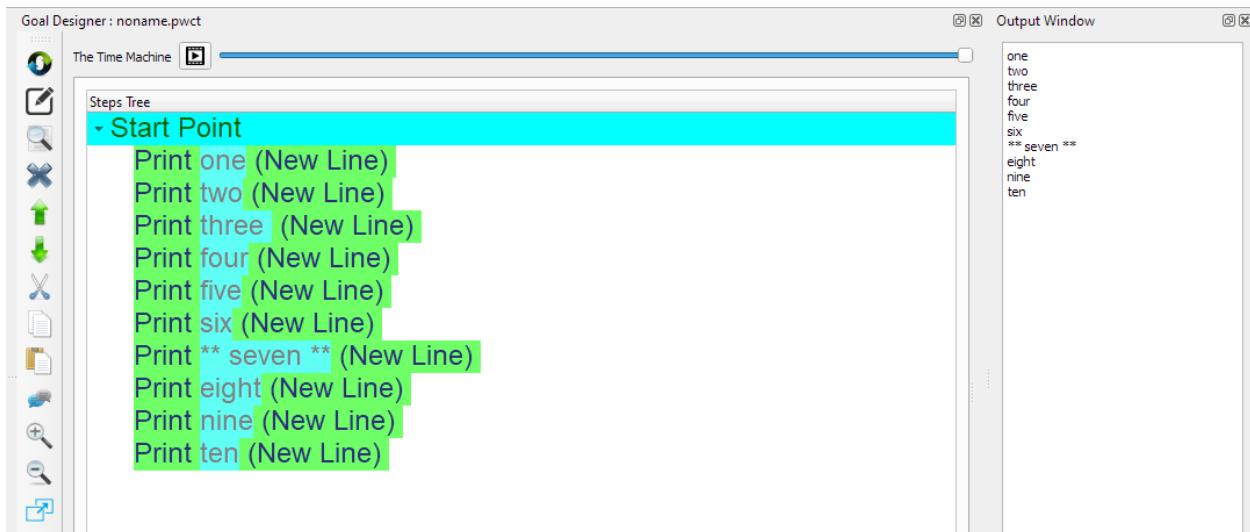


Now we have the step (Print nine) in the Steps Tree

This time we will write (ten) then click (Ok) or press Ctrl+W



Run the program using Ctrl+F5 to see the output



3.3.11 Undo

If we did something wrong in the Steps Tree (Like deleting a step) we can cancel this using Ctrl+Z

Also we can select (Undo) from the (Edit) menu instead of pressing Ctrl+Z

3.4 Visual Source Files

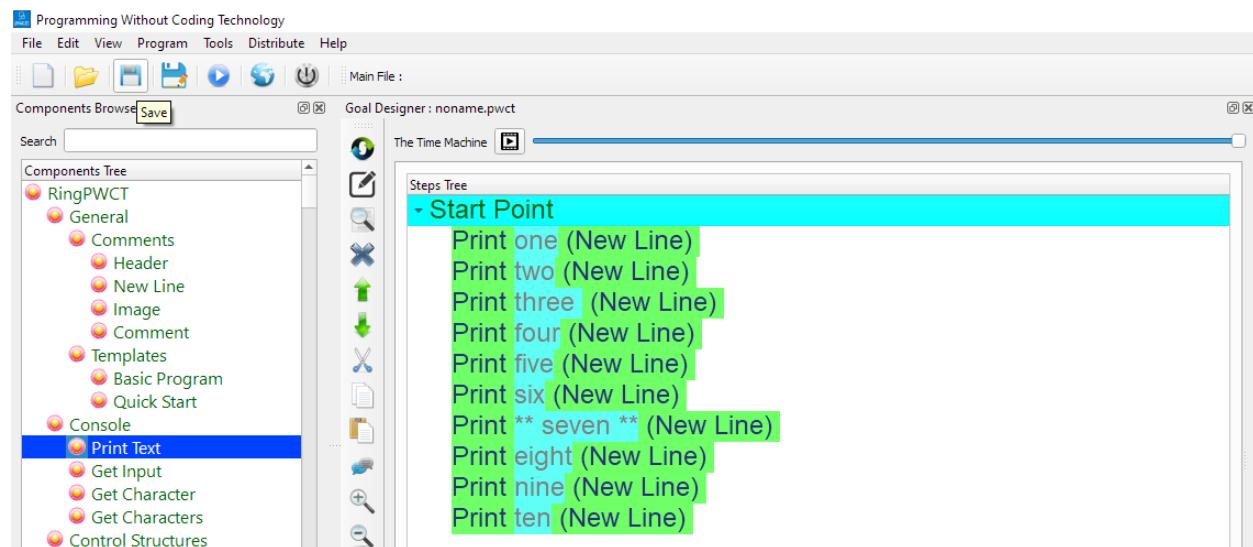
In this section we will learn about the Visual Source Files (*.pwct)

Section contents:

- Saving the file
- Opening the file
- Starting new file

3.4.1 Saving the file

To save the file, Click on the (Save) button from the Main Toolbar or press Ctrl+S

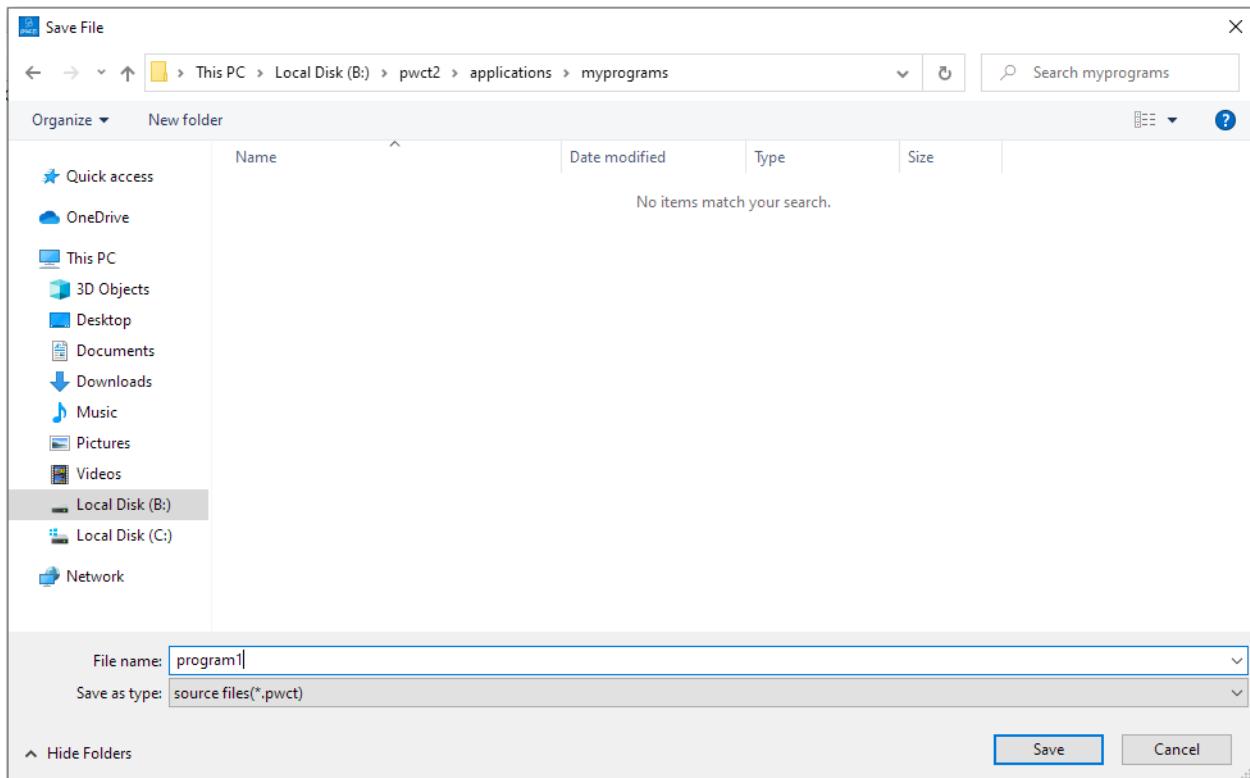


Select the folder, or create a new folder

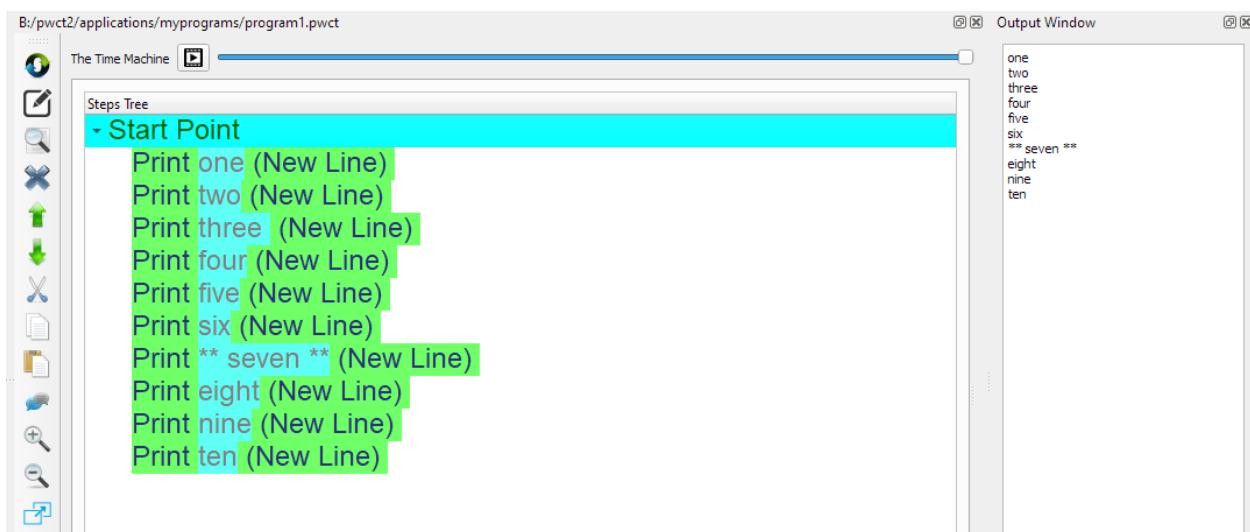
For example : pwct2/applications/myprograms

Then type the file name : program1

The saved file will be progam1.pwct



After saving the file, the file name will be used as the window title in the Goal Designer

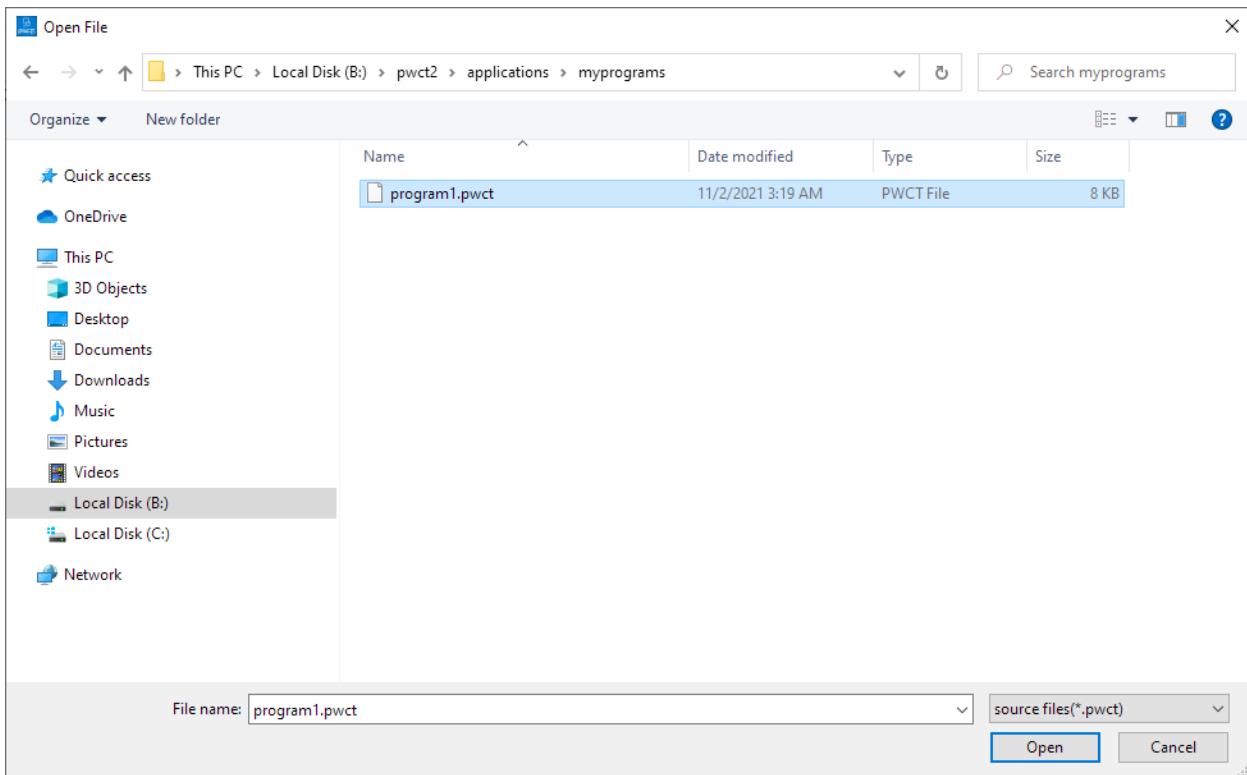


3.4.2 Opening the file

To open a file, click on the (Open) button from the Main Toolbar

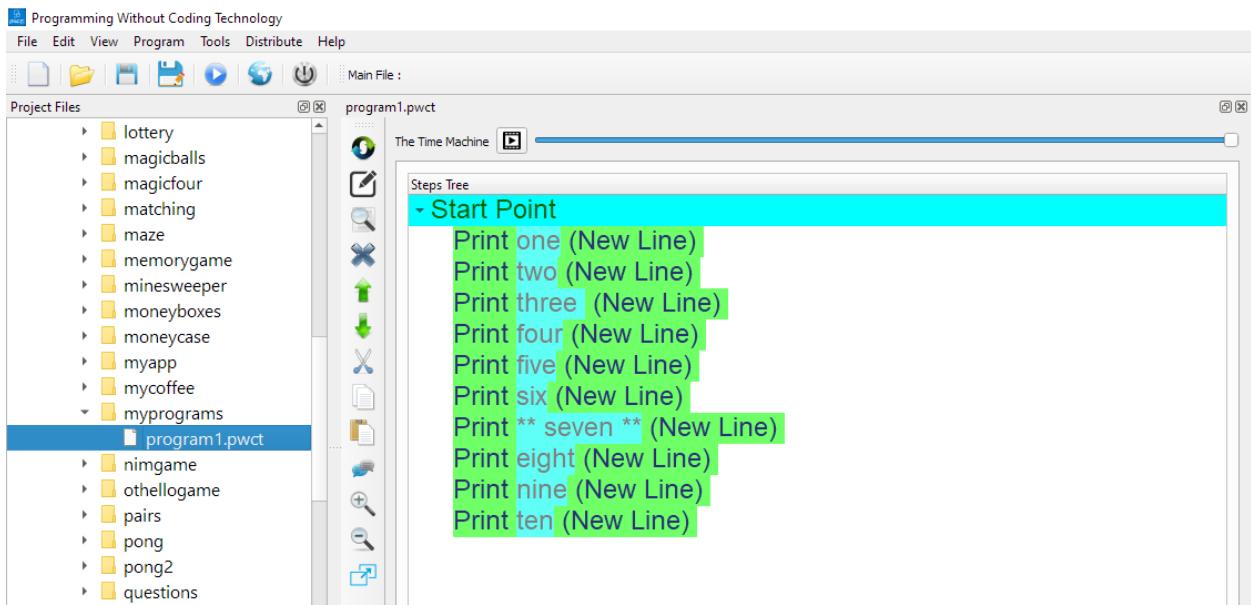


Select the file, for example: program1.pwct



Also we can open the files using the Project Files window

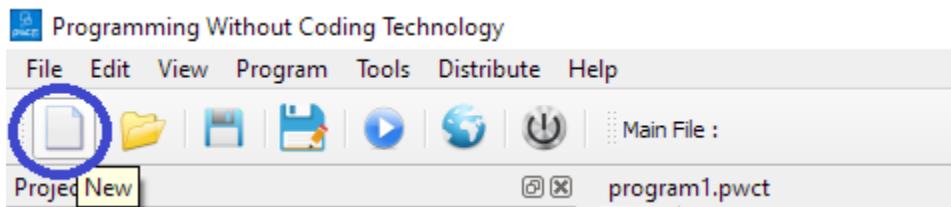
Just go to the folder and select the file



3.4.3 Starting new file

To start a new file, From the (File) menu, select (New) or press Ctrl+Alt+N

Another way is to click on the (New) button from the Main toolbar

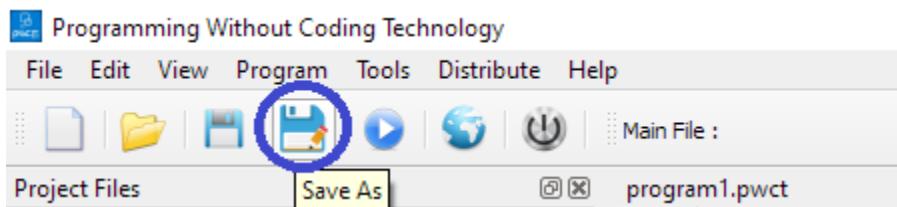


The new file name will be (noname.pwct)

To change the file name click (Save)

3.4.4 Save As

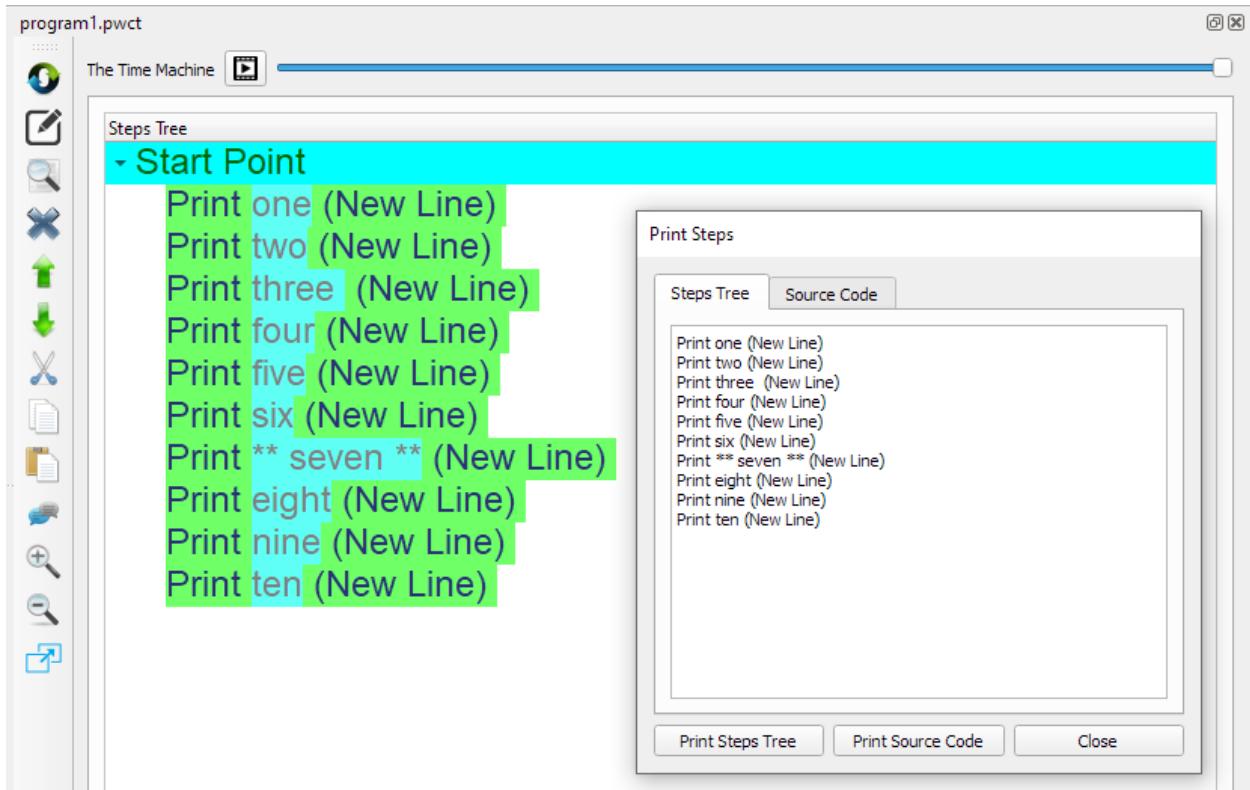
Using (Save As) we can save the file using another name



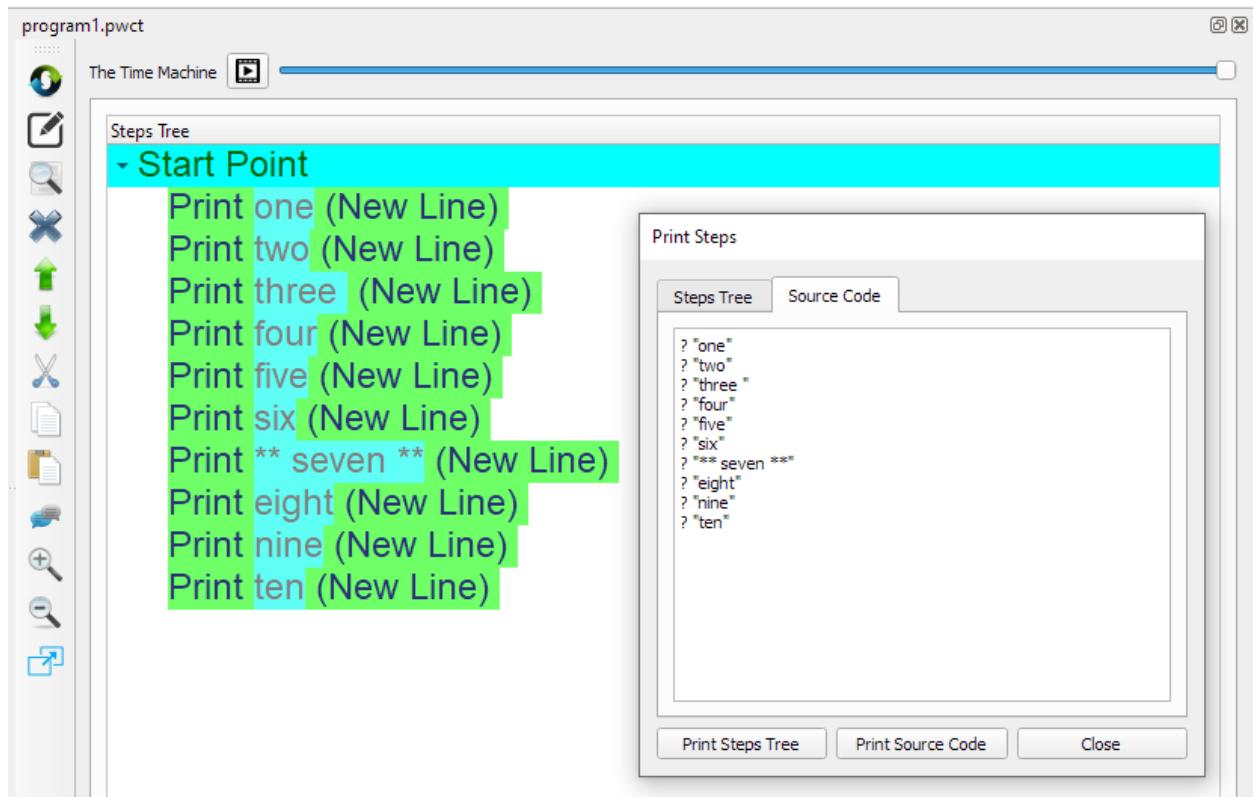
3.4.5 Printing the file

To print the file press Ctrl+P or select (Print to PDF) from the file menu

We can print the Steps Tree



Also we can print the textual source code



Tip: We can copy the text that represent the Steps Tree or the Source Code and paste it in our discussions in Online Forums

3.4.6 Go to line

Each visual source file (*.pwct) represent a textual source code file

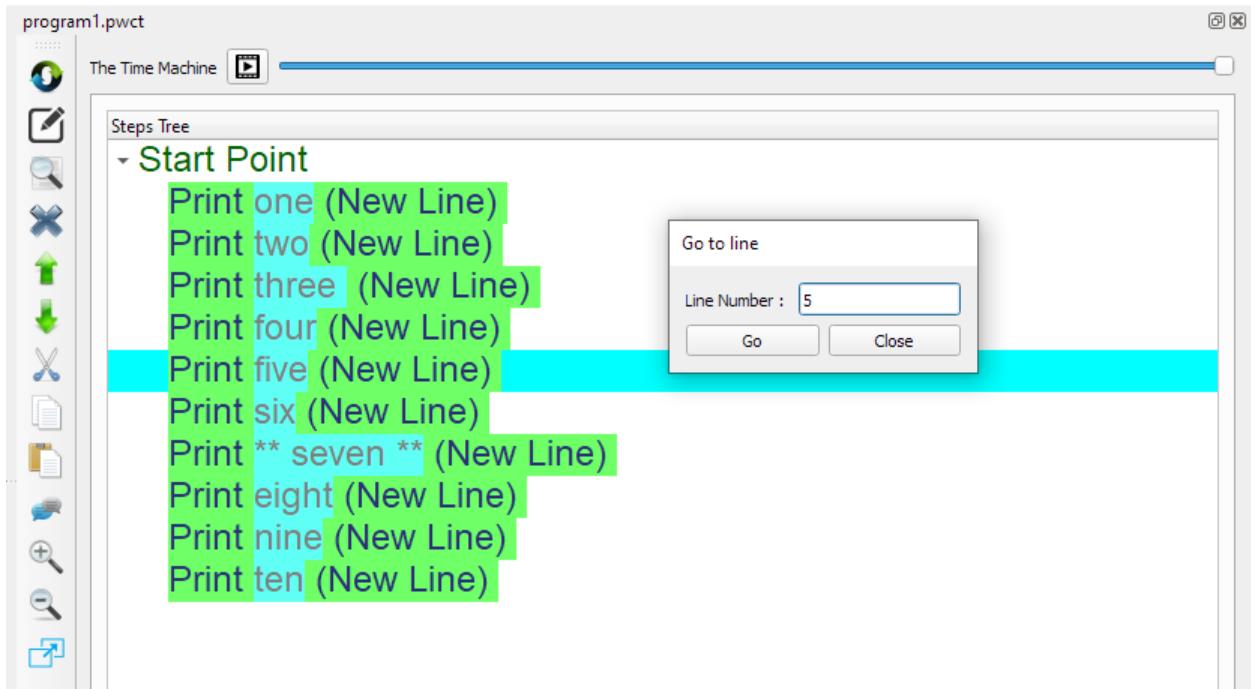
Sometimes we want to select a step inside the Steps Tree that represent a specific line in the textual source code

This is useful if we have a runtime error in a specific line of source code

Using (Go to line) from the (Edit) menu, we can do that

Tip: We can open the (Go to line) window using Ctrl+G

Note: Each step in the Step Tree could represent one or many of textual source code lines



3.5 Customization

In this section we will learn about the customization of the PWCT Environment

Section contents:

- View Menu
- Customization Window

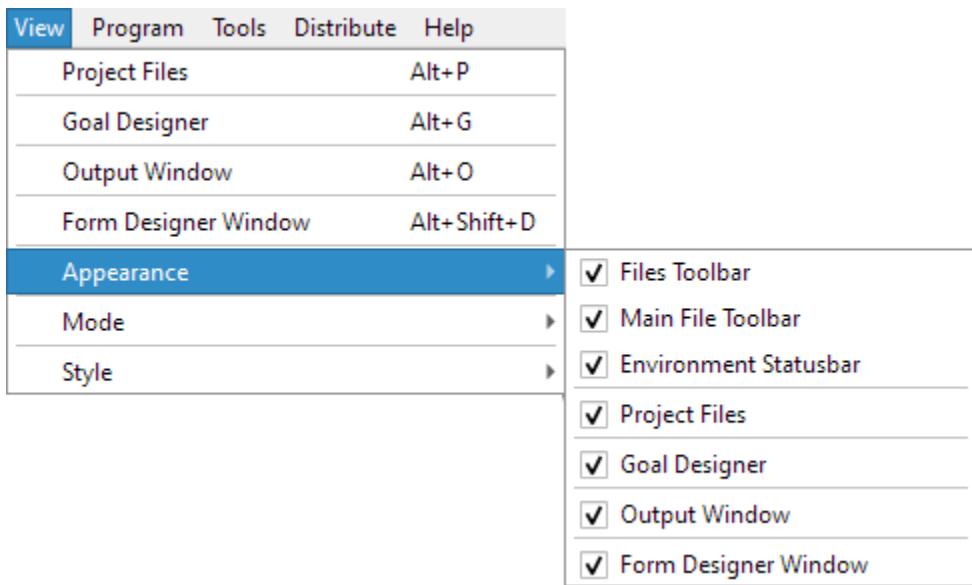
3.5.1 View Menu

From the View menu we can navigate quickly to the different dockable windows

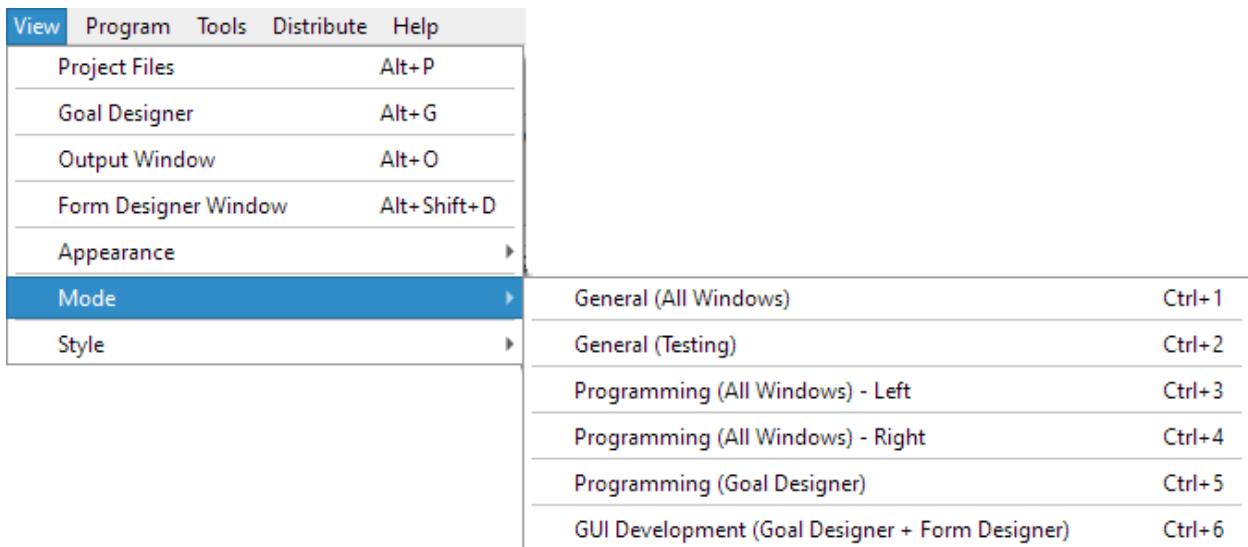
Also we have keyboard shortcuts to do the navigation quickly

- To activate the Project Files window press Alt+P
- To activate the Goal Designer window press Alt+G
- To activate the Output window press Alt+O

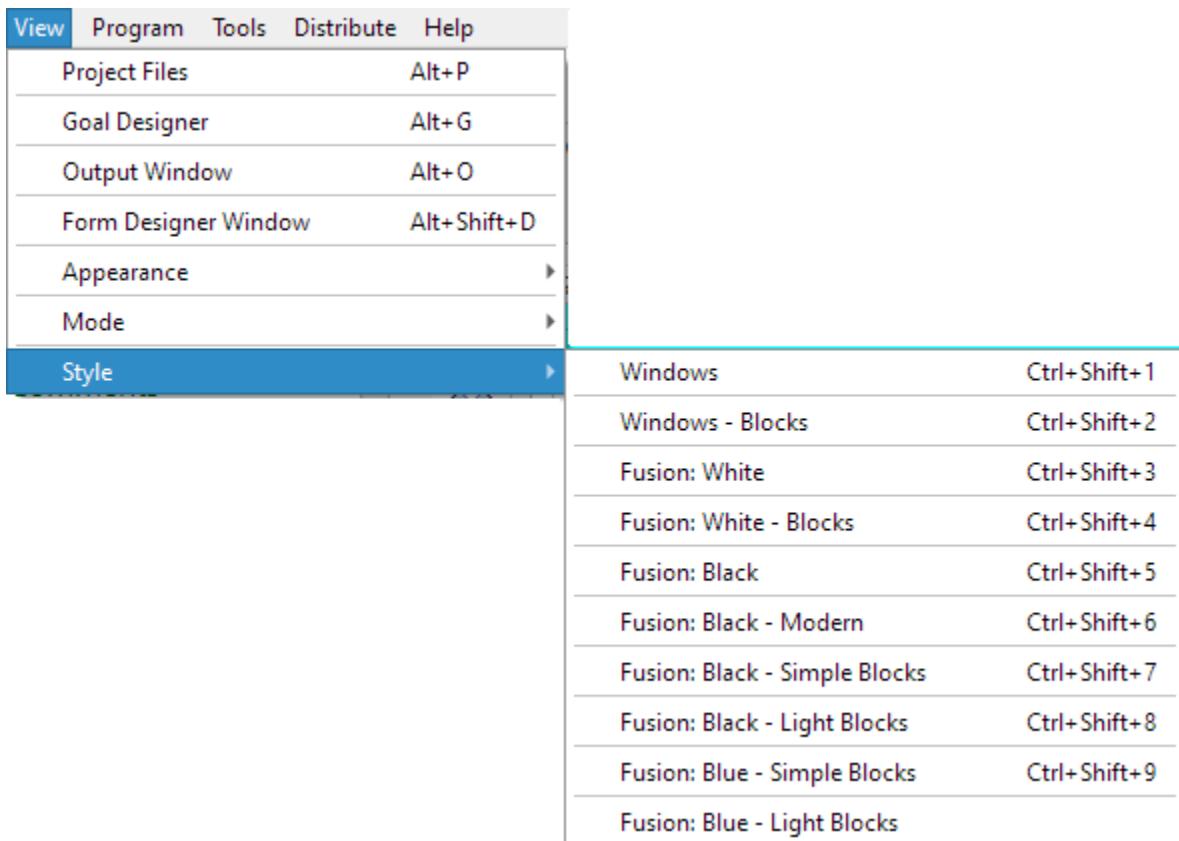
From the (Appearance) submenu we can show/hide the different windows and toolbars



From the (Mode) submenu we can quickly arrange the dockable windows



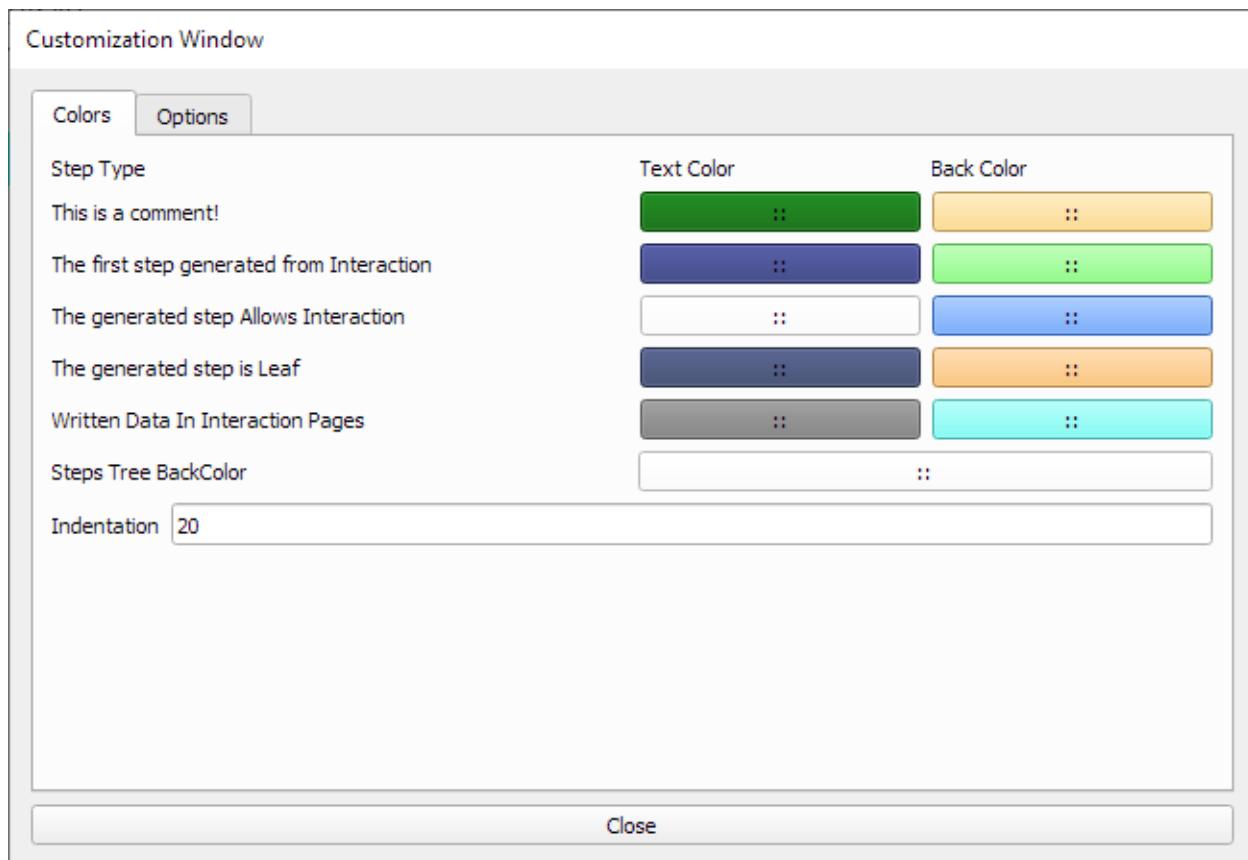
From the (Style) submenu we can change the theme



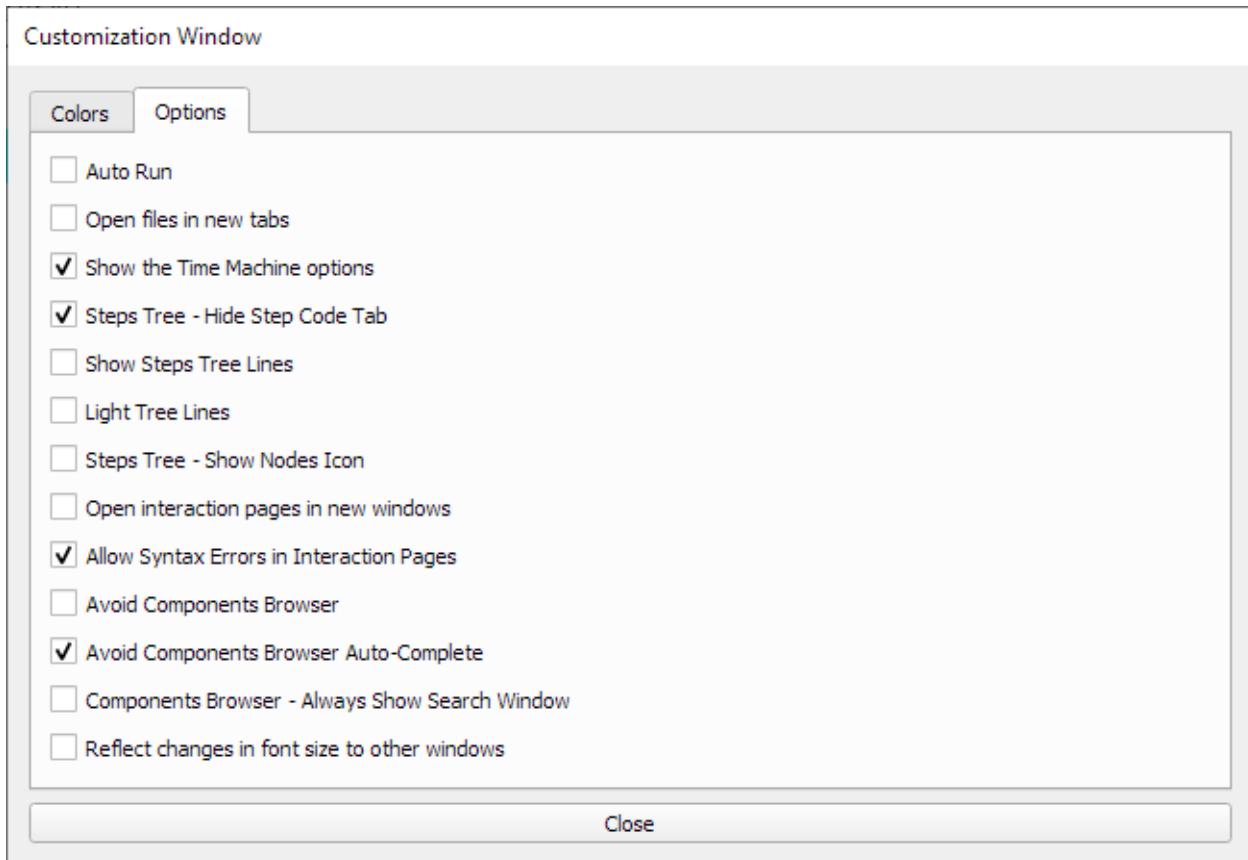
3.5.2 Customization Window

To open the Customization window, From the (Edit) menu select(Customization) or Press Alt+C

From this window we can change the Steps Tree colors



Also we have more options that control the behavior of the PWCT environment



3.6 Run Programs

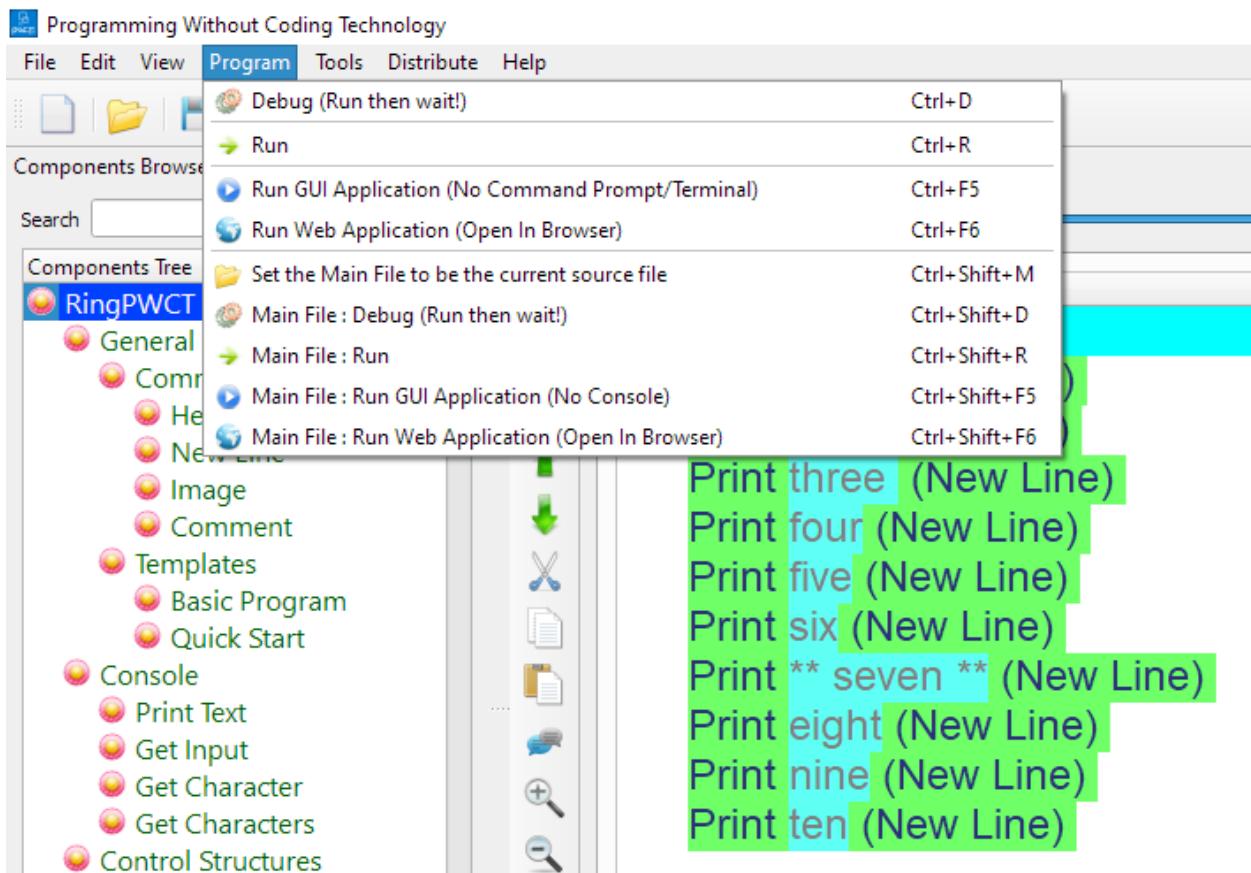
We can run programs using the Program menu or using the Toolbars

Also we can use the Keyboard shortcuts

3.6.1 Program Menu

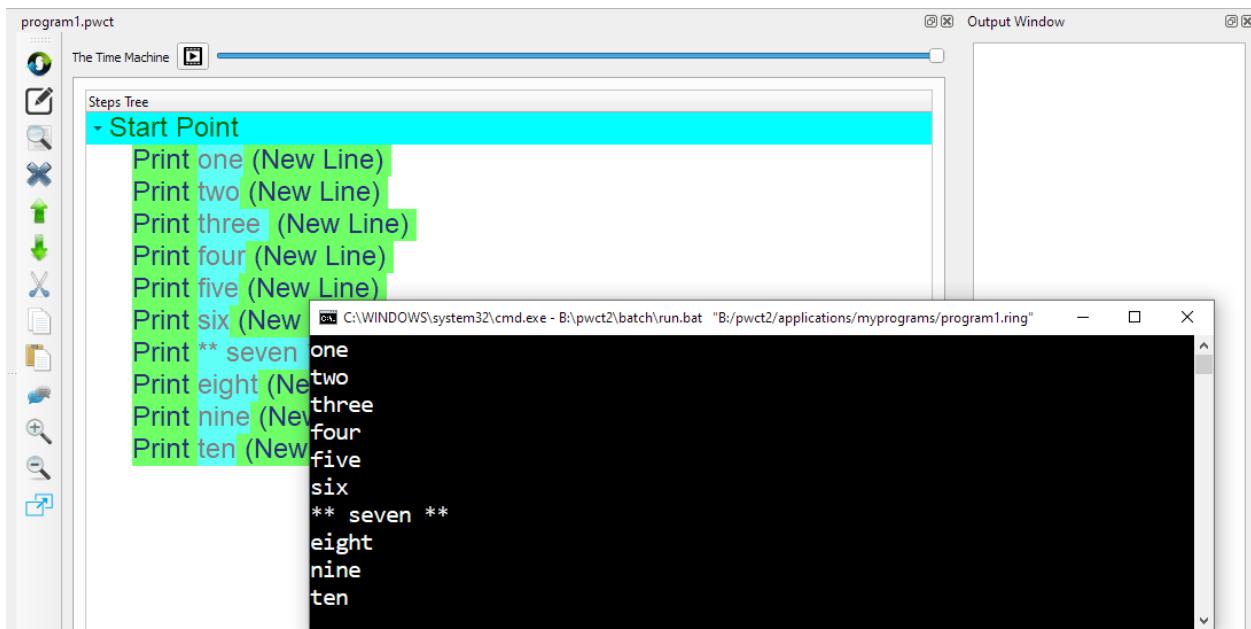
From the Program menu, we can run the current opened file using different options

Also we can run the Main file in the project



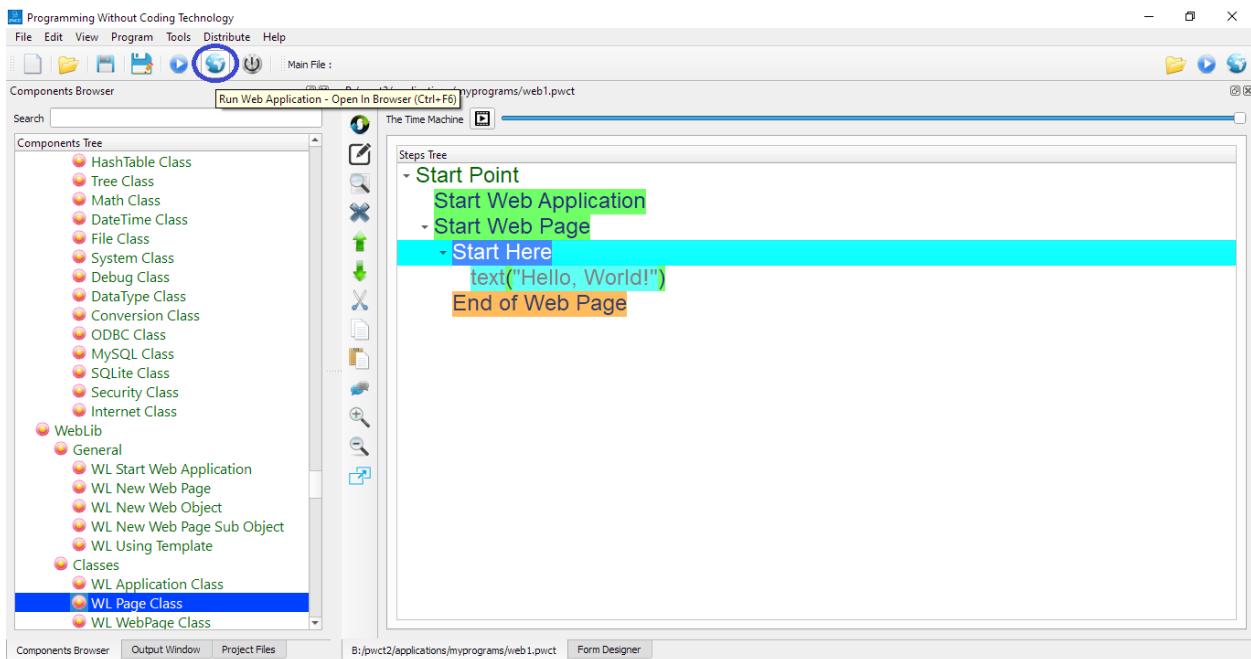
Selecting (Debug - Run then wait) will display the output in the command prompt window

After running the program, the pause command will be executed so we can see the output

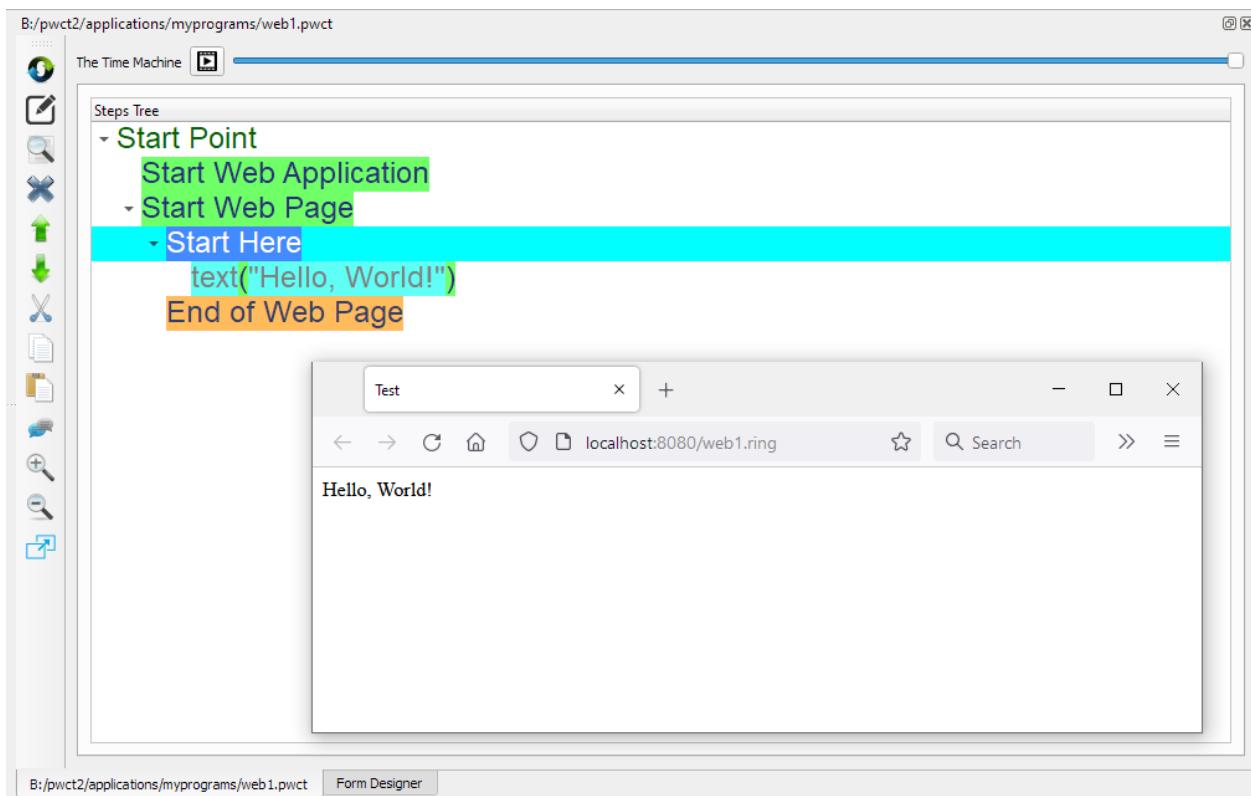


Also we have the (Run web application) option that we can use during web development

Also we have this option in the Main toolbar



For example, the next program display (Hello, World) in the web browser

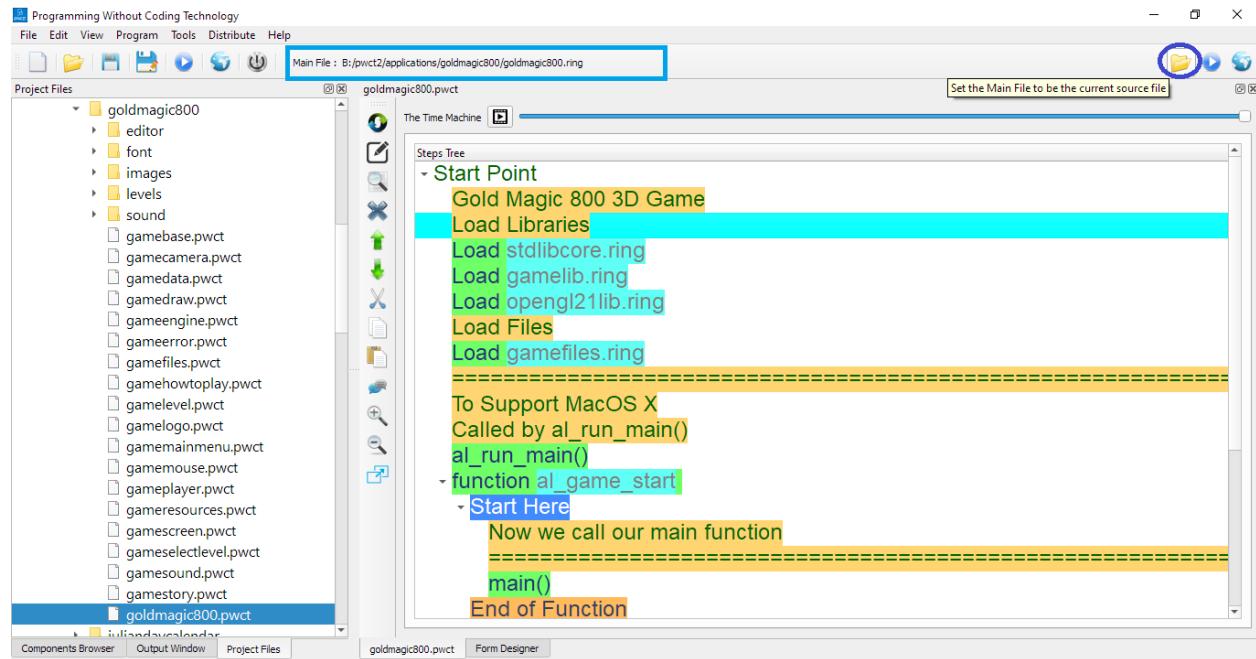


3.6.2 Main File toolbar

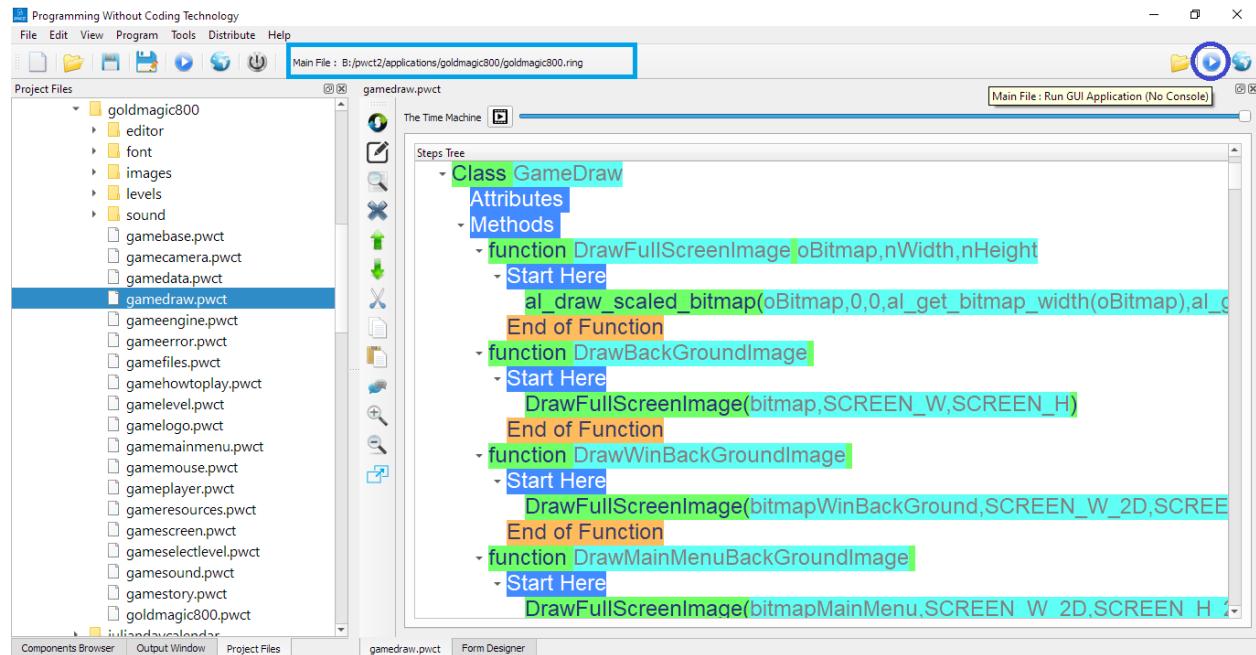
The program menu contains options for using the (Main File)

When we have a project that contains many visual source files, we can select one of these files as the Main file that we can use to run the project

For example we select (goldmagic800.pwct) as the main file, This file generate (goldmagic800.ring)



We can run the main file while opening other files in the project



3.7 More Options

In this section we will learn more about the features and resources provided by the PWCT Environment

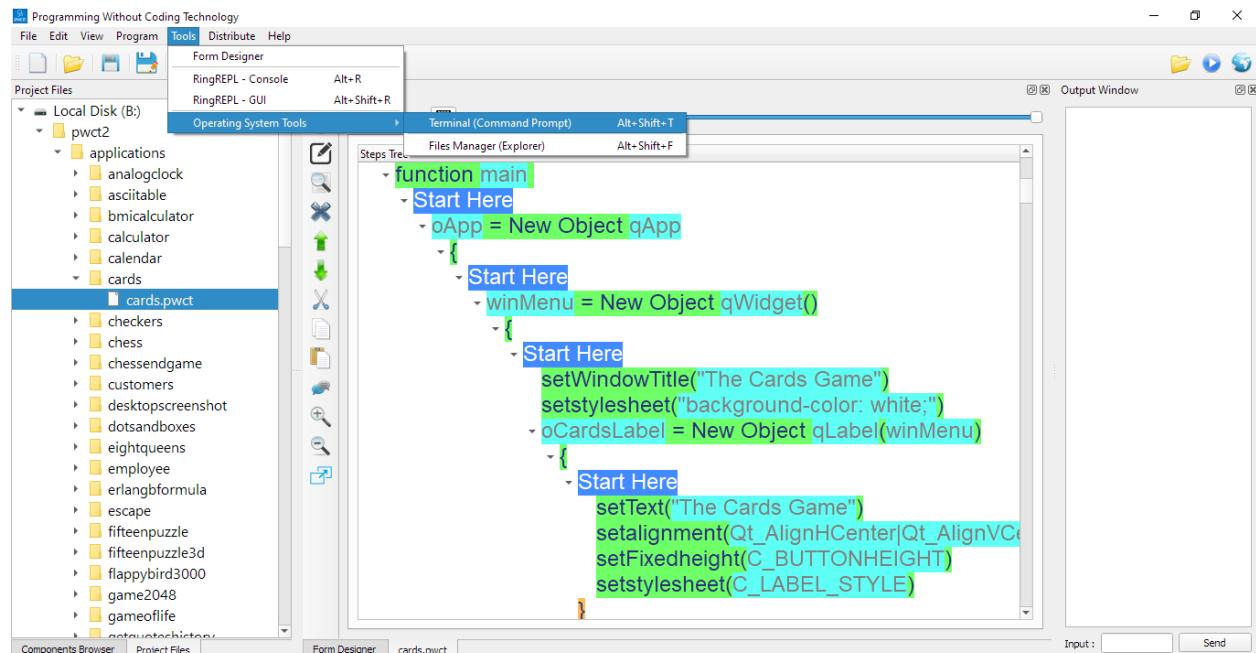
Section contents:

- Tools Menu
- Distribute Menu
- Help Menu

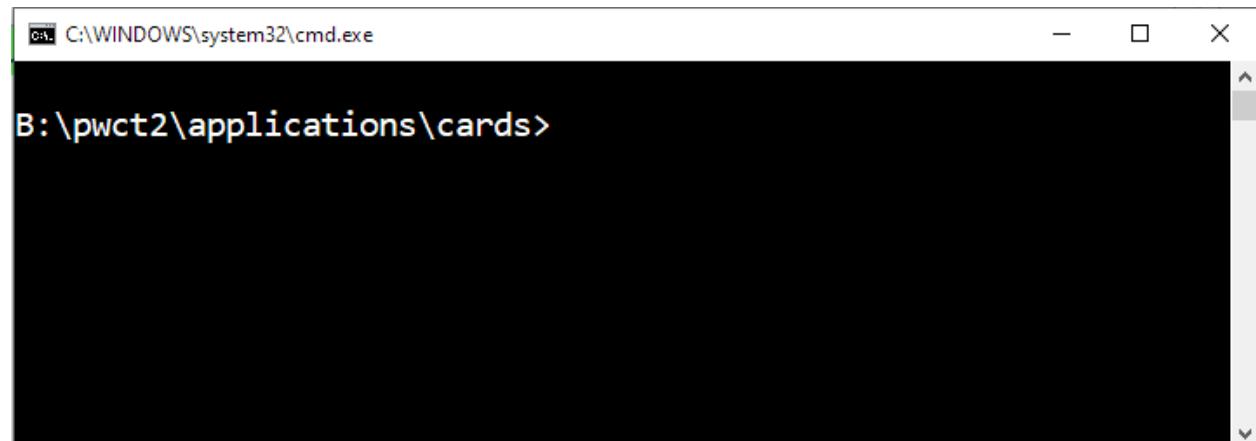
3.7.1 Tools Menu

From the tools menu we can run the Form Designer or the RingREPL (Read-Eval-Print-Loop)

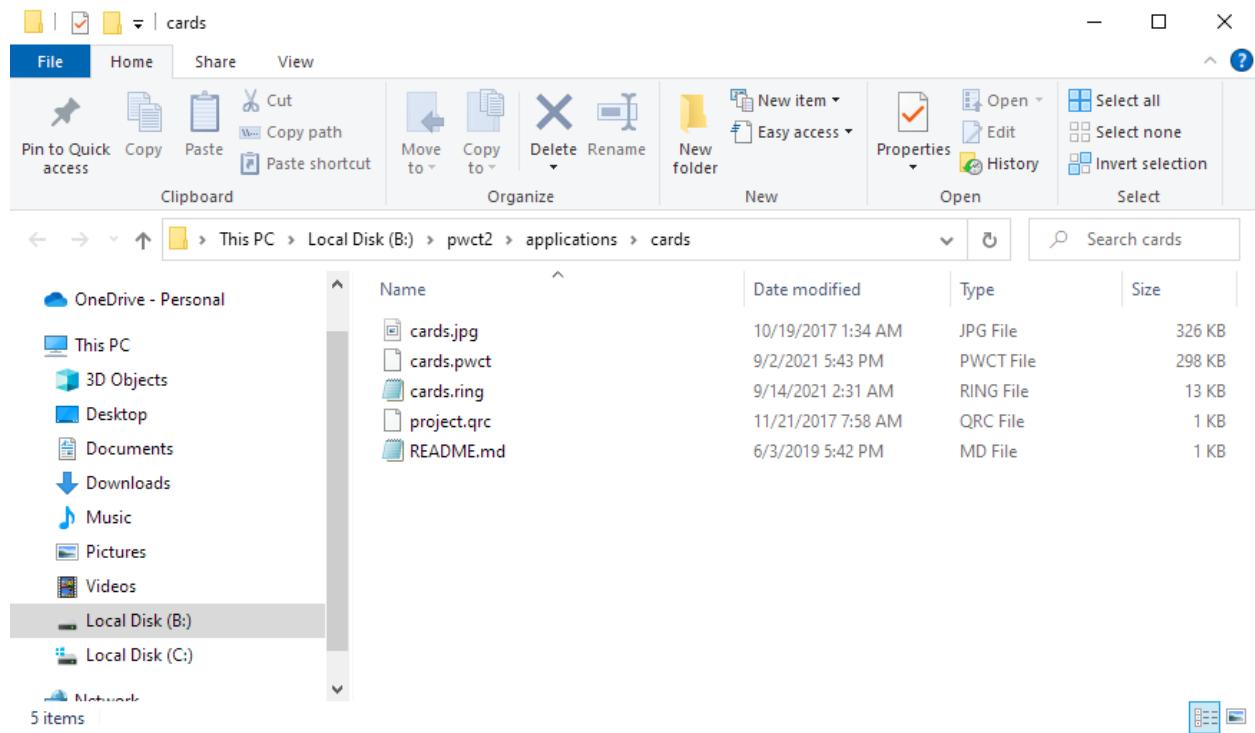
Also we can open the Command Prompt or the Files Explorer



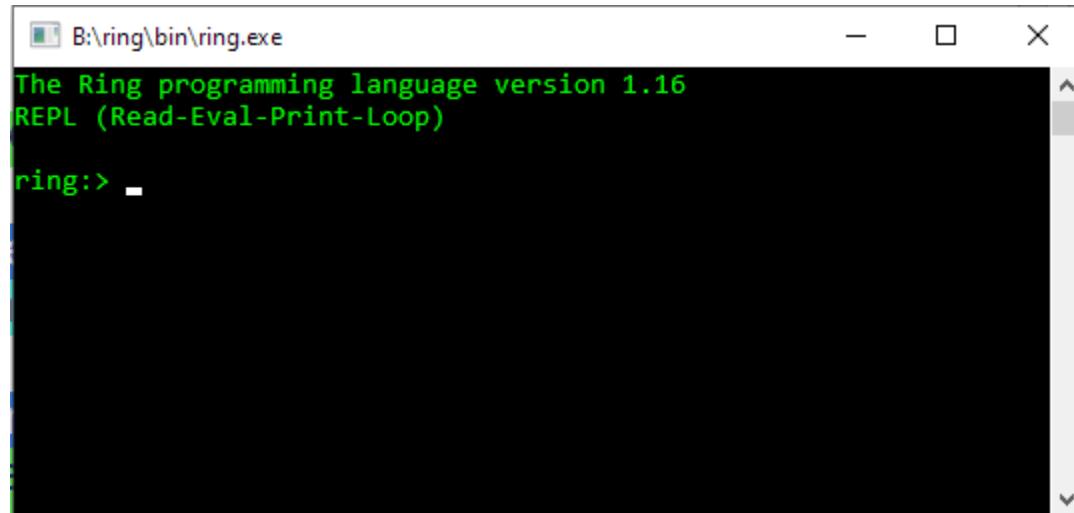
When we open the Command Prompt, the current directory will be the folder of the opened visual source file



Also when we open the Files Explorer, the current directory will be the folder of the opened file

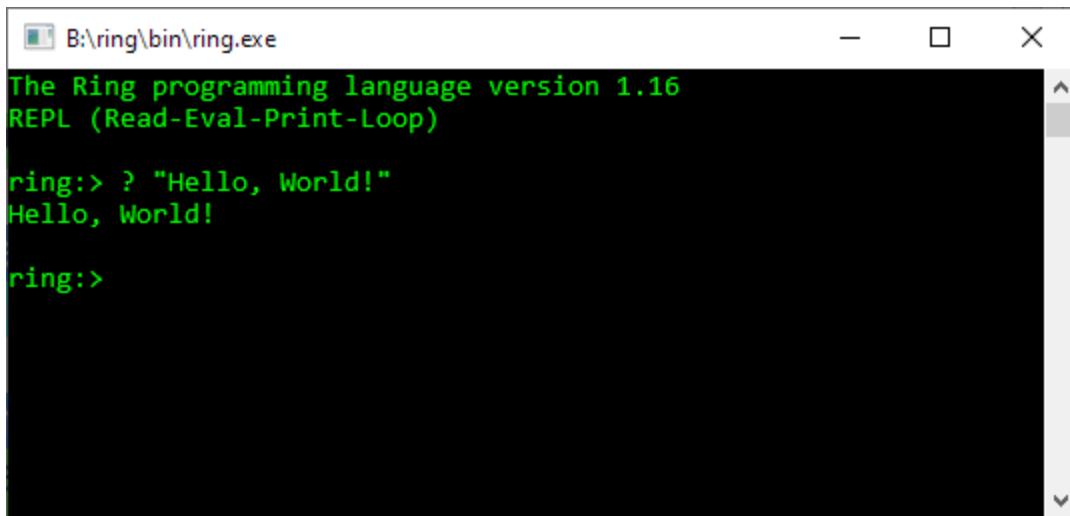


When we run RingREPL we can write and execute Ring code directly



For example, We can write and execute the next program that print a message on the screen

```
? "Hello, World!"
```

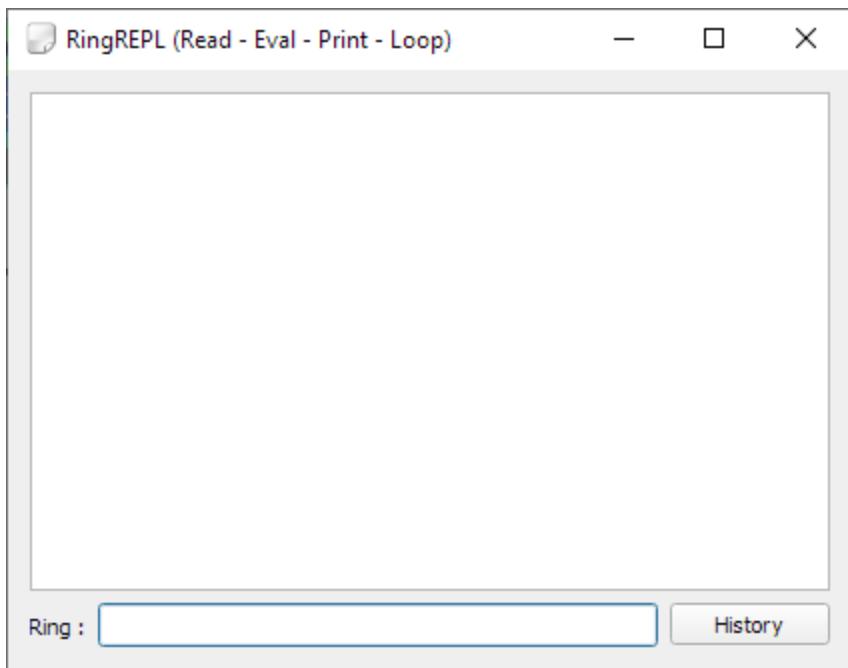


```
B:\ring\bin\ring.exe
The Ring programming language version 1.16
REPL (Read-Eval-Print-Loop)

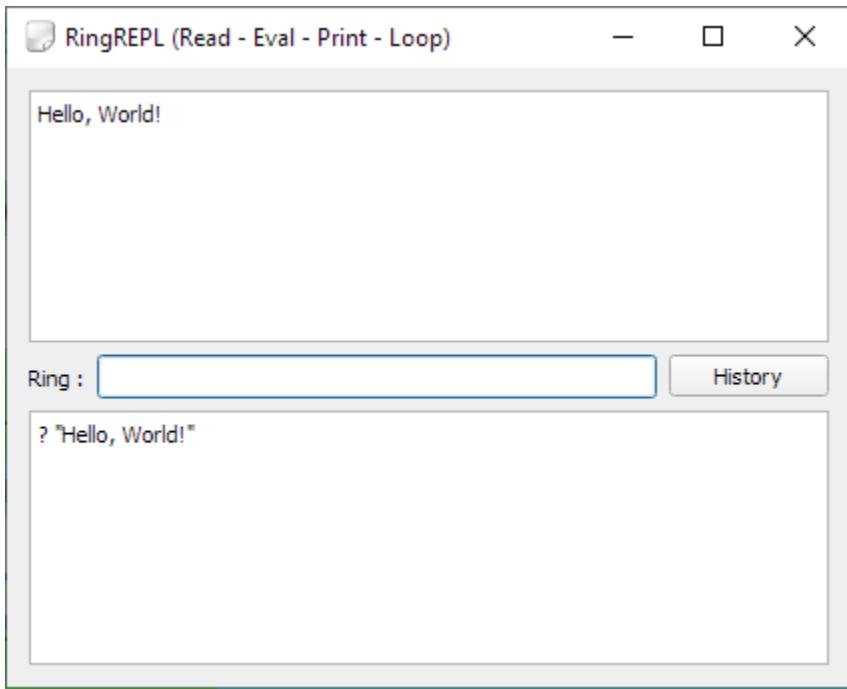
ring:> ? "Hello, World!"
Hello, World!

ring:>
```

We have also the GUI version of RingREPL

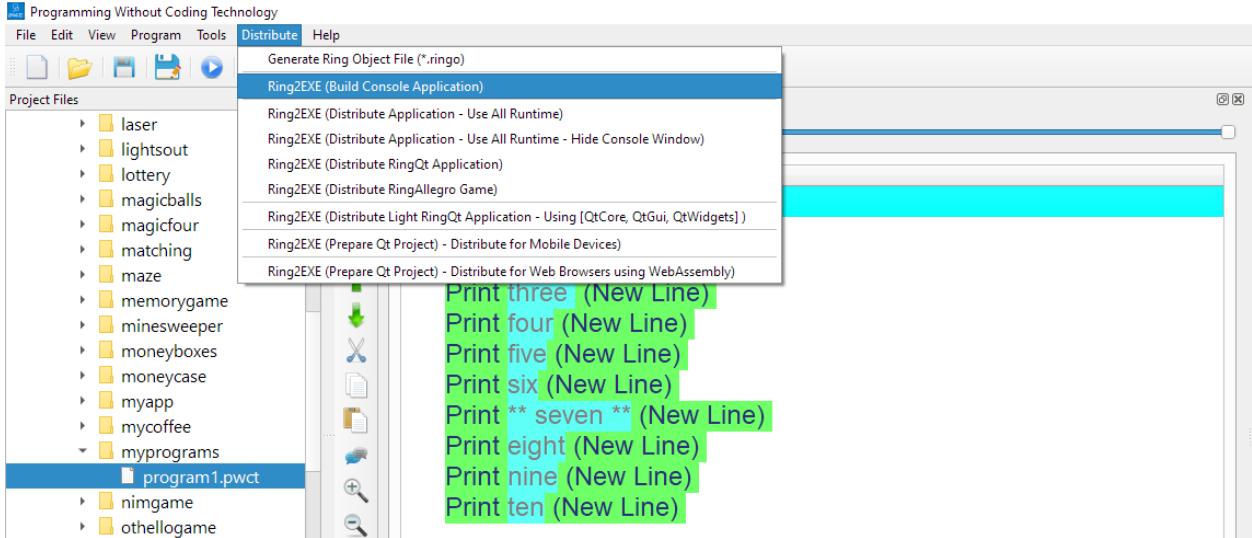


This version comes with the History option that contains the previous commands

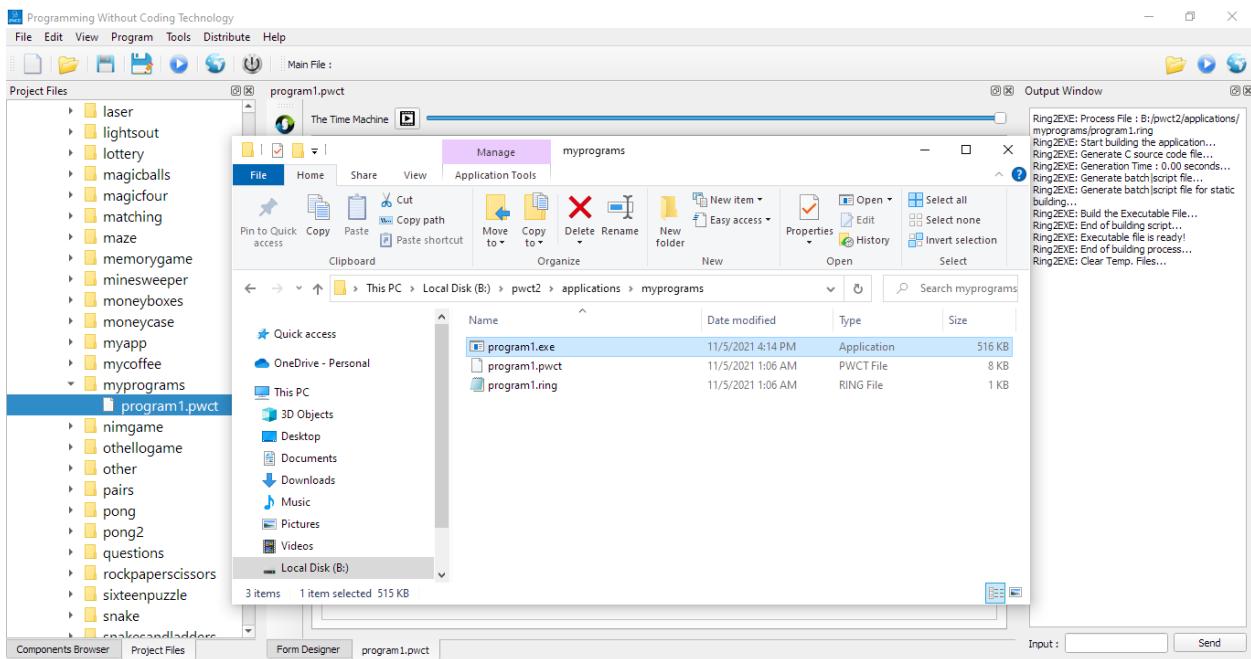


3.7.2 Distribute Menu

Using the Distribute menu we can distribute or applications and build executable files



For example, we can build (program1.exe) for our program



We can run (program1.exe) from the command prompt and see the output

```
C:\WINDOWS\system32\cmd.exe
B:\pwct2\applications\myprograms>dir
Volume in drive B has no label.
Volume Serial Number is 64DD-625D

Directory of B:\pwct2\applications\myprograms

11/05/2021  04:14 PM    <DIR>        .
11/05/2021  04:14 PM    <DIR>        ..
11/05/2021  04:14 PM           527,872 program1.exe
11/05/2021  01:06 AM           7,602 program1.pwct
11/05/2021  01:06 AM           106 program1.ring
               3 File(s)      535,580 bytes
               2 Dir(s)  172,018,782,208 bytes free

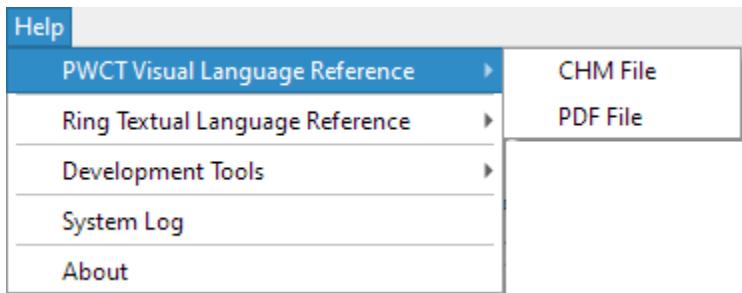
B:\pwct2\applications\myprograms>program1
one
two
three
four
five
six
** seven **
eight
nine
ten

B:\pwct2\applications\myprograms>
```

3.7.3 Help Menu

Using the Help menu we can open PWCT or Ring documentation

The documentation comes in different formats like CHM & PDF



CHAPTER
FOUR

RICH COMMENTS

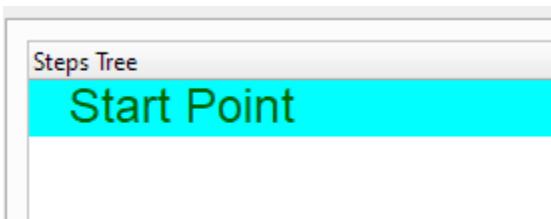
In this chapter we are going to learn how to add comments to our programs

PWCT support rich comments where we can mix between text, colors, lines, and images

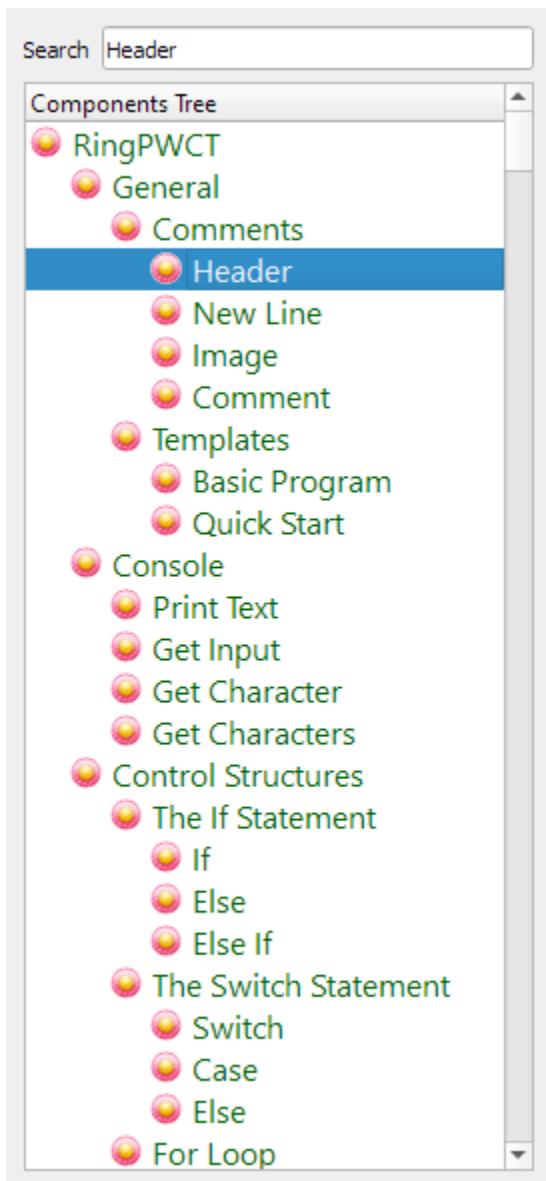
Also we can tables use HTML code

4.1 Using the Header Component

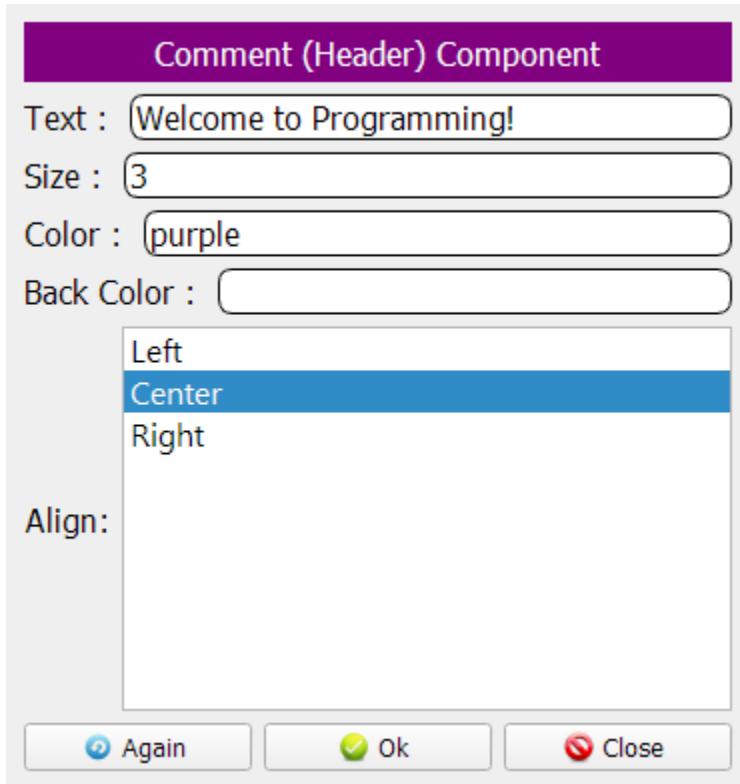
In the begining, No steps exist in our Steps Tree



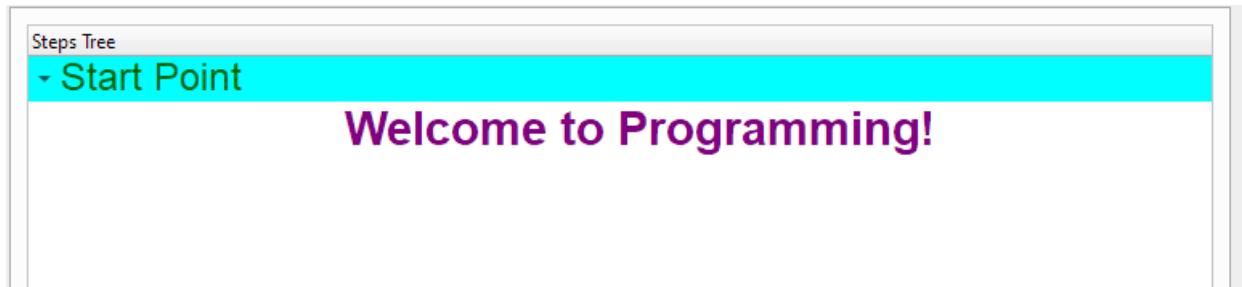
We will select the (Header) component



In the interaction page we will set the text to (Welcome to Programming!)

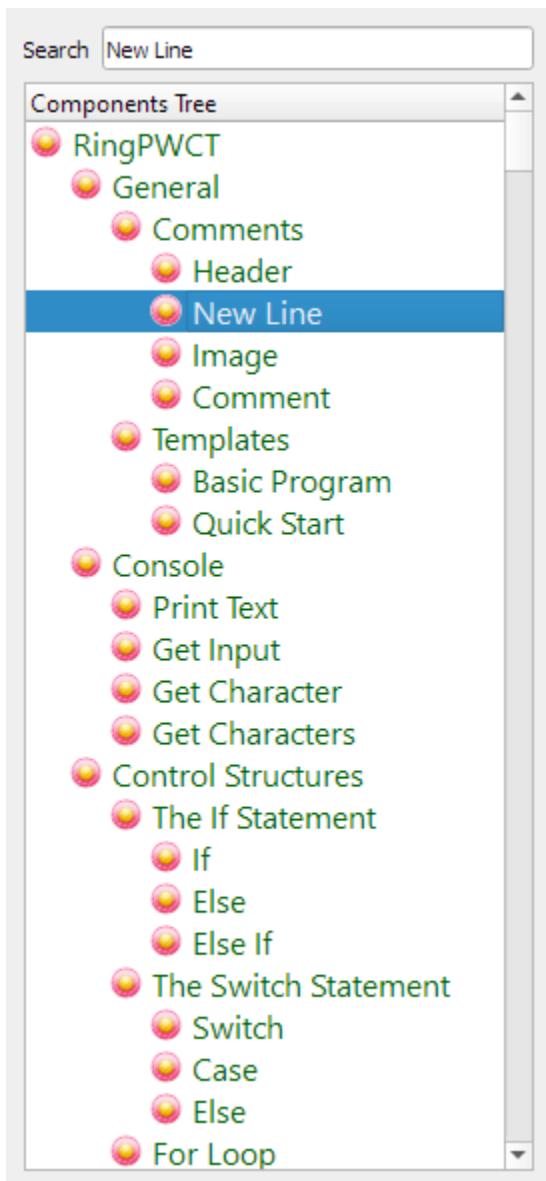


Now, we see this comment added to our Steps Tree

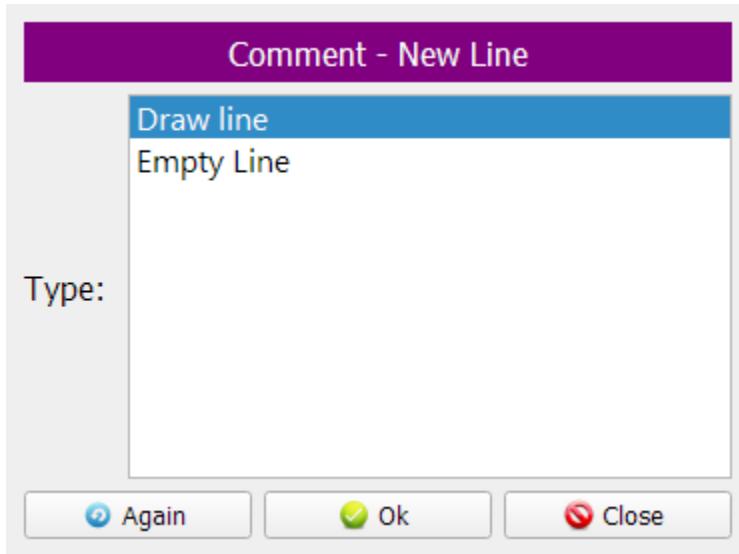


4.2 Using the New Line Component

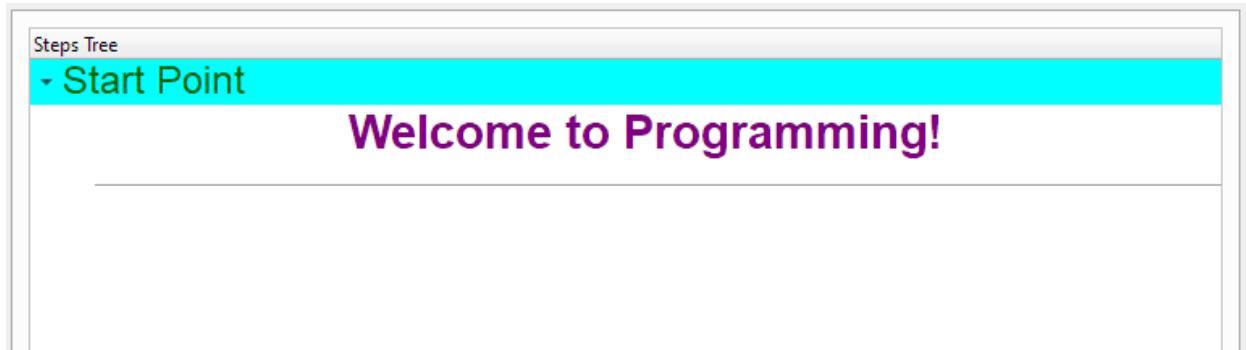
We will select the (New Line) component



From the Interaction Page, we will select (Draw Line)

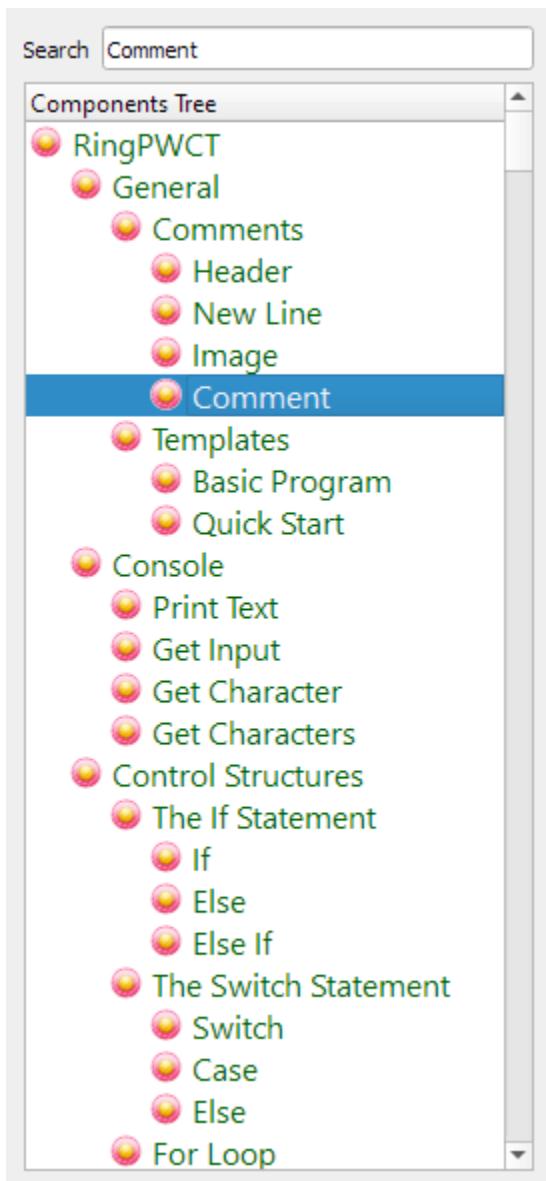


Now we see the Line is added to the Steps Tree

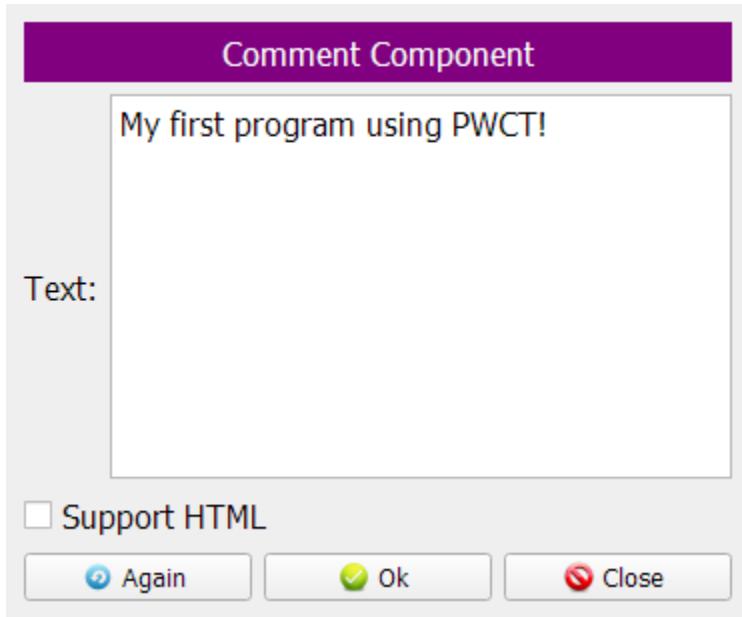


4.3 Using the Comment Component

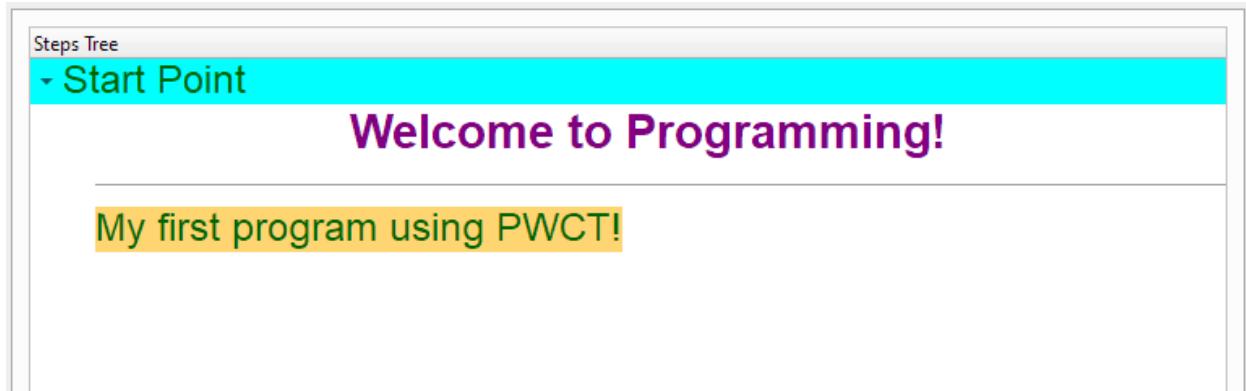
We will select the (Comment) component



In the text section we will write (My first program using PWCT!)

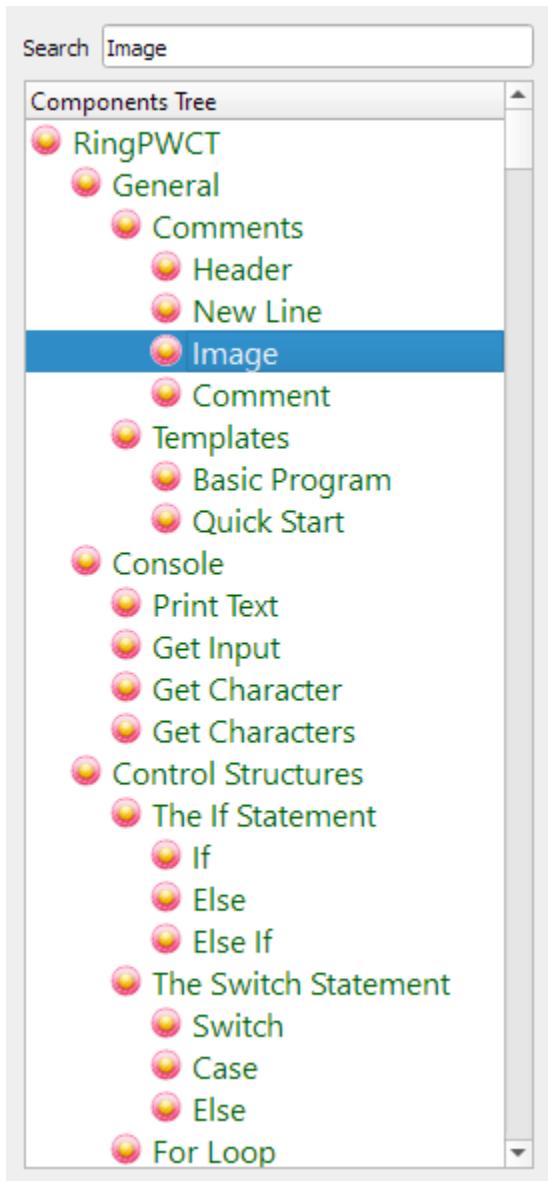


The comment is added to the Steps Tree



4.4 Using the Image Component

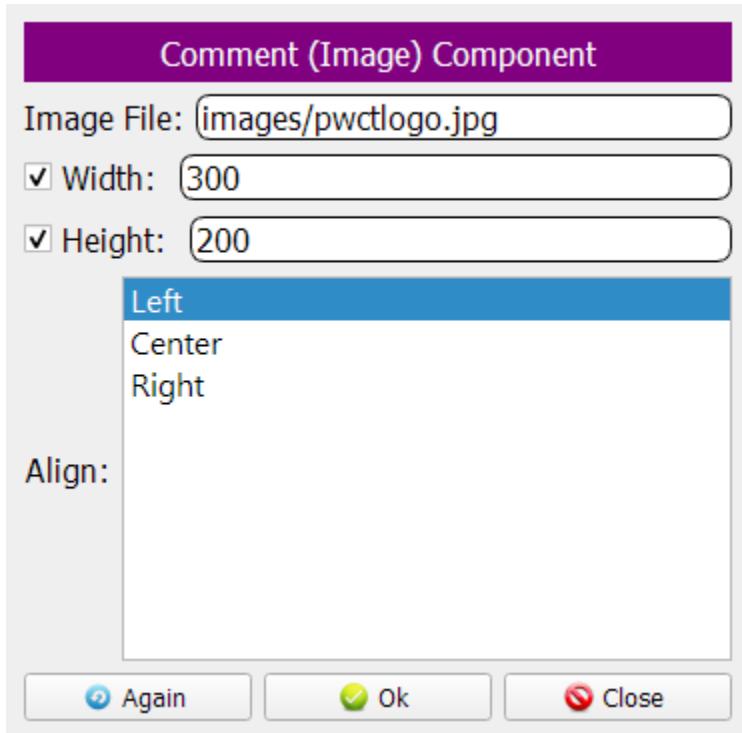
Time to add an image using the (Image) component



We will type the image file, width & height

The image exist in a relative path (images folder)

The images folder exist in the same folder as our visual source file (program2.pwct)



Now we see the Image is added to our Steps Tree

Steps Tree

- Start Point

Welcome to Programming!

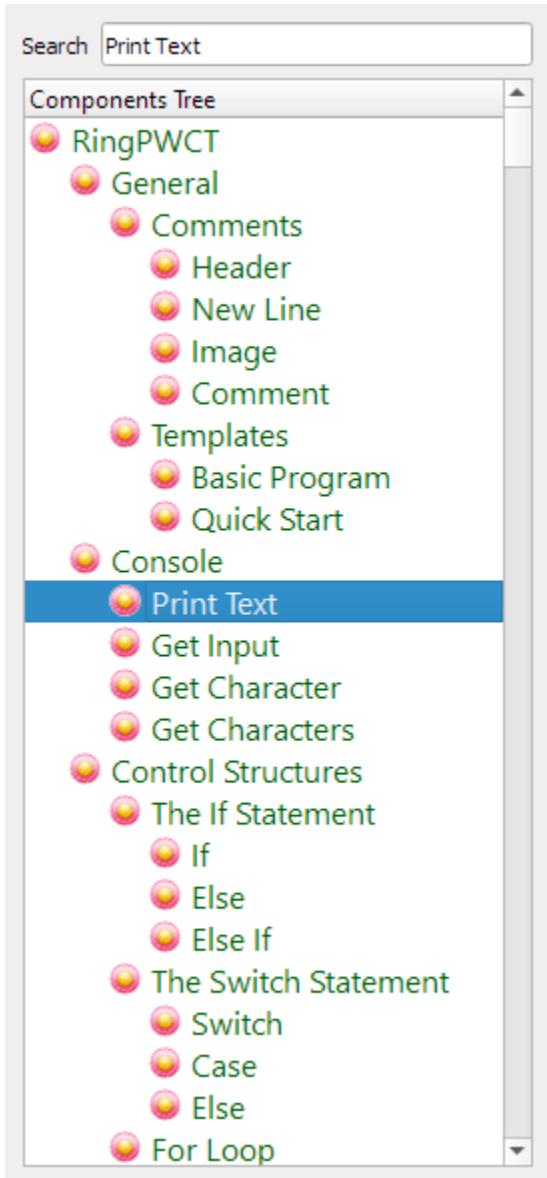
My first program using PWCT!



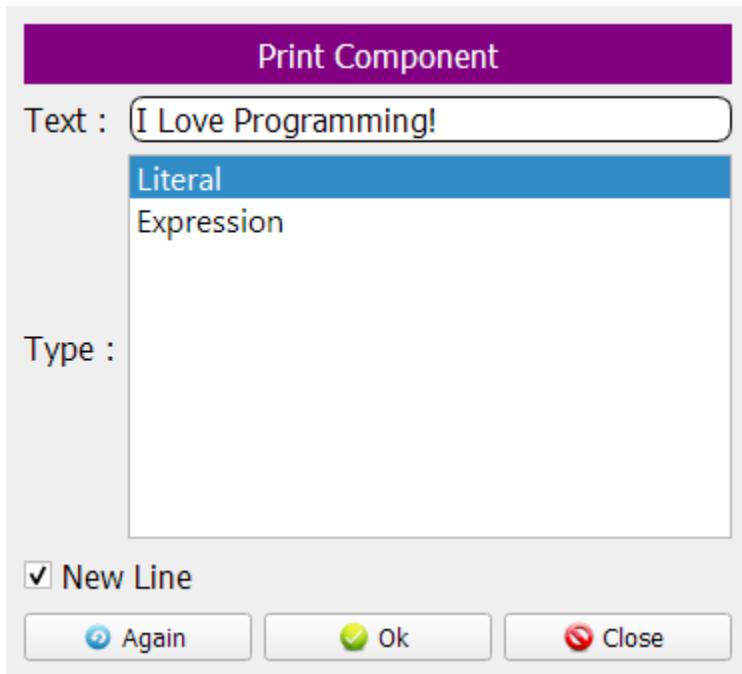
The PWCT logo features a blue background with the text "PWCT" in white. Below it, the tagline "PROGRAMMING WITHOUT CODING TECHNOLOGY" is written in smaller white text. To the right of the text, there are several 3D cubes of varying sizes.

4.5 Printing some text

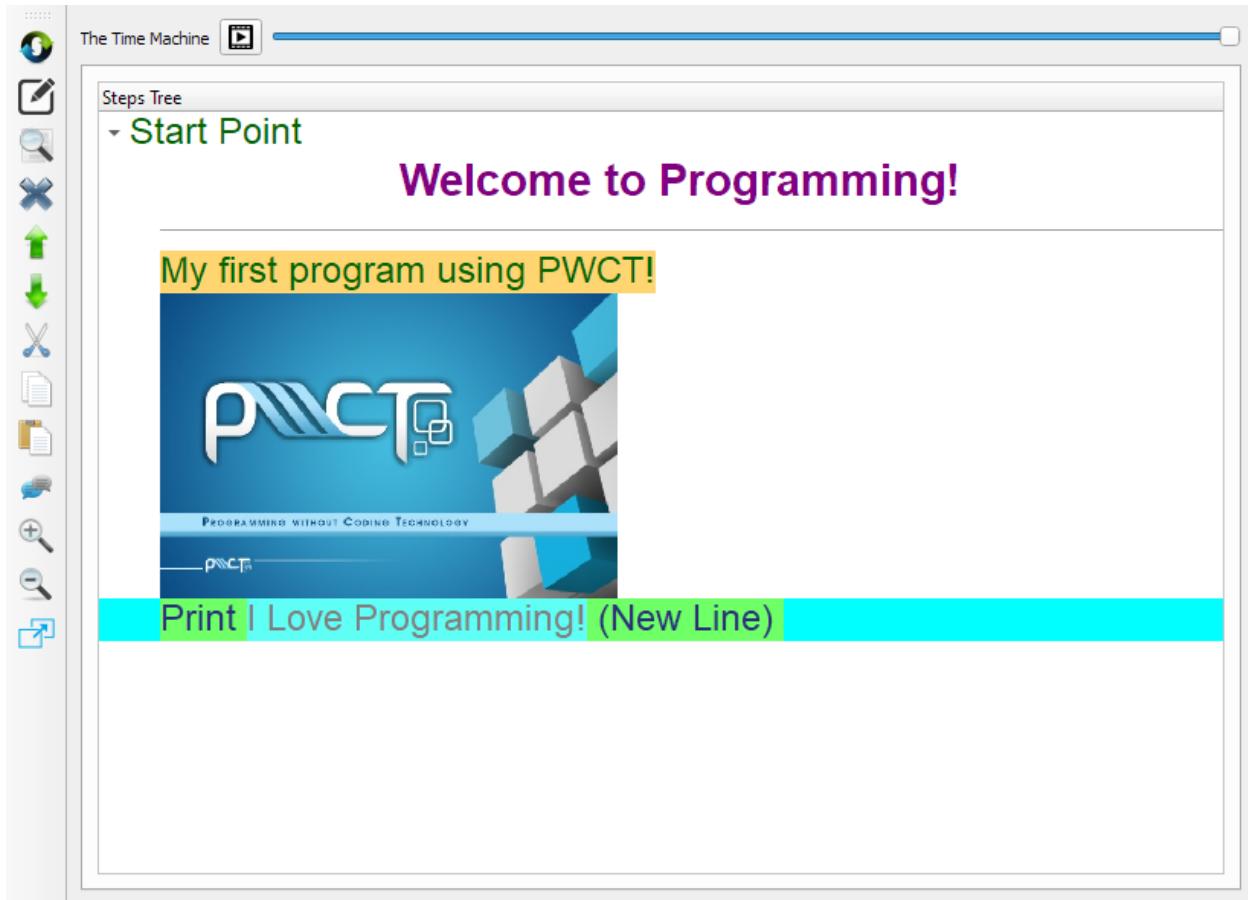
We can use the (Print Text) component



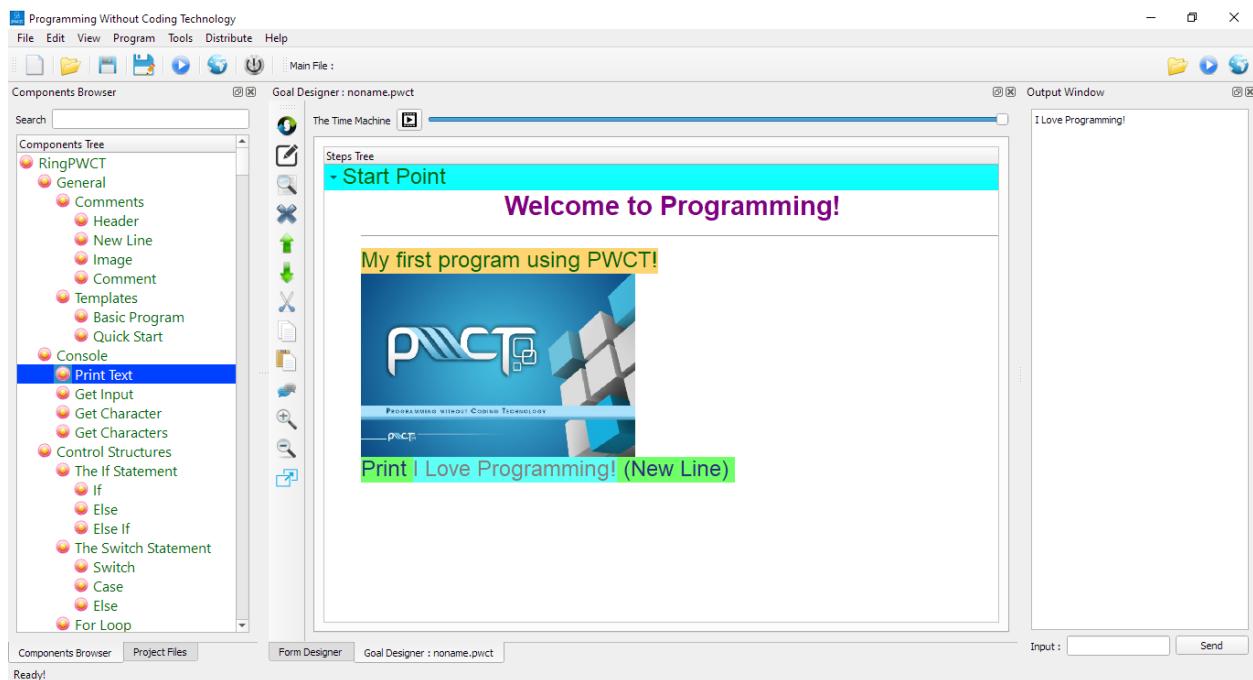
Set the text to (I Love Programming!)



The Print step is added to the Steps Tree



Now we can run the program and see the output in the Output Window



BASIC PROGRAM COMPONENT

In this chapter we are going to learn how to use the Basic Program Component

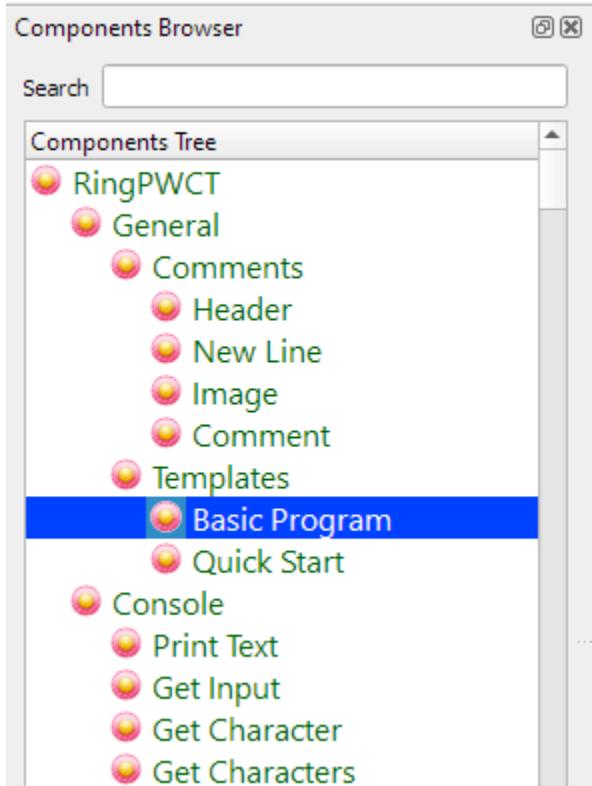
5.1 Introduction

Using the Basic Program component we can see big picture behind the structure of our visual source files

In each visual source file, We can Load other files, have statements, functions & classes

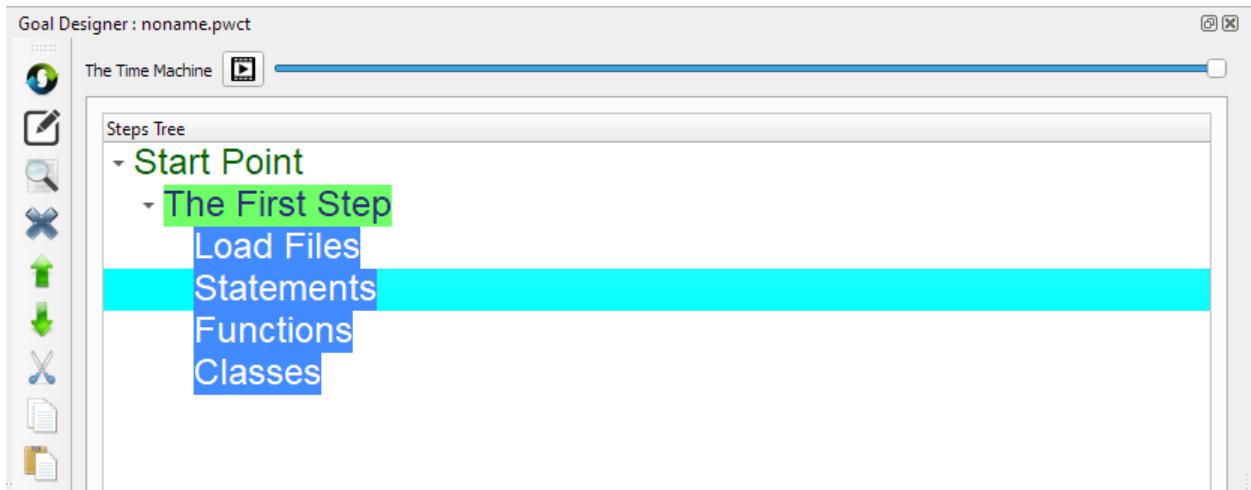
5.2 Selecting the Component

From the Components Browser select (Basic Program)



5.3 Steps Tree

After selecting the (Basic Program) component, The next steps will be generated in the Goal Designer



These steps are like Comments, i.e. using the (Basic Program) component is optional.

In our programs, we must follow this order with respect to the structure of our programs.

We start with loading files, then statements, then functions and finally our classes.

QUICK START COMPONENT

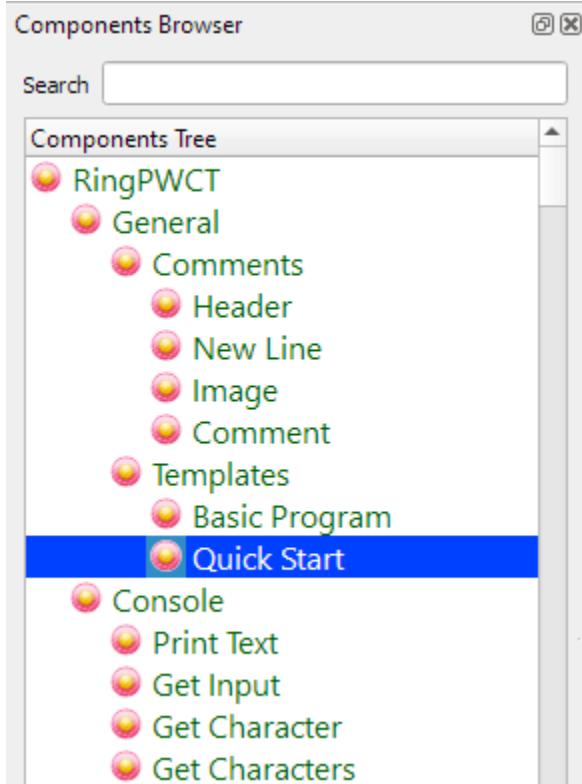
In this chapter we are going to learn how to use the Quick Start Component

6.1 Introduction

Using the Quick Start component we can try some samples quickly

6.2 Selecting the Component

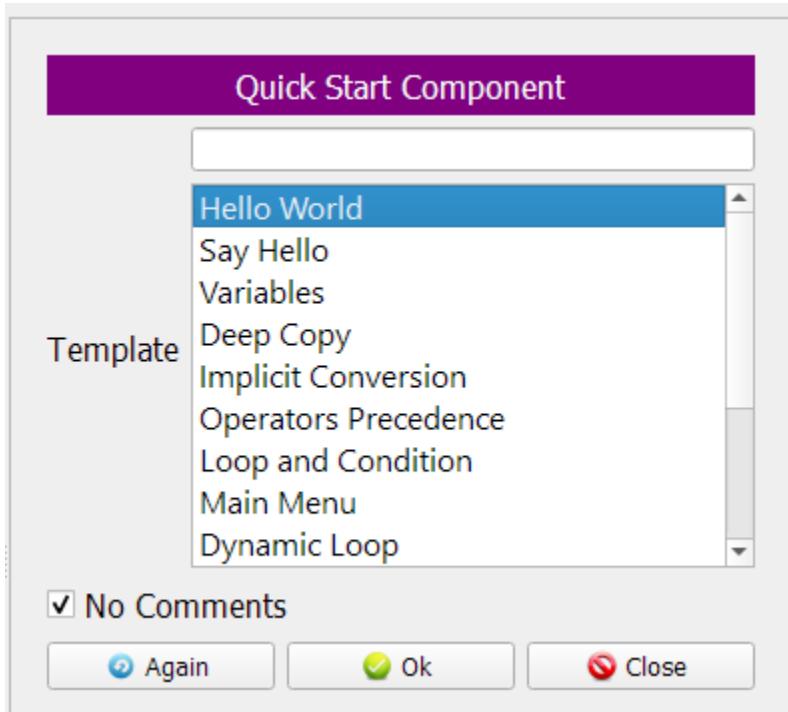
From the Components Browser select (Quick Start)



6.3 The Interaction Page

After selecting the (Quick Start) component

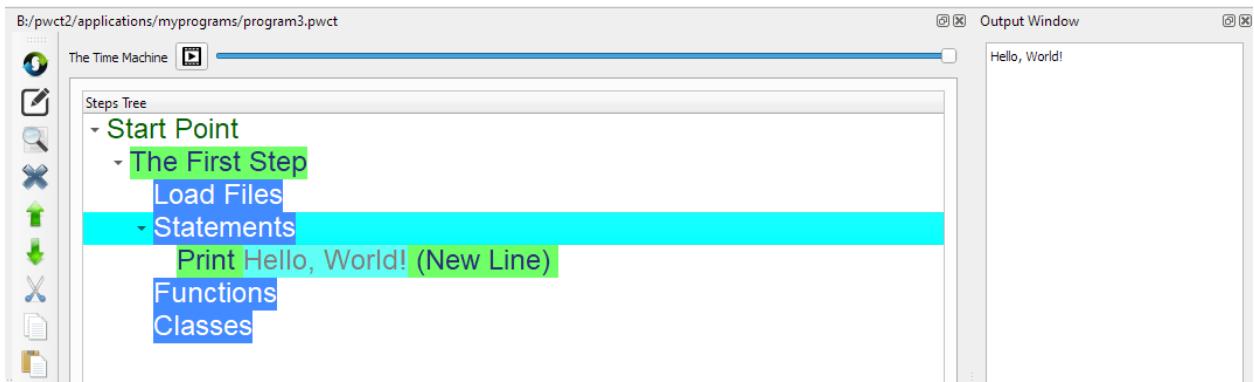
We will see the next interaction page



6.4 Steps Tree

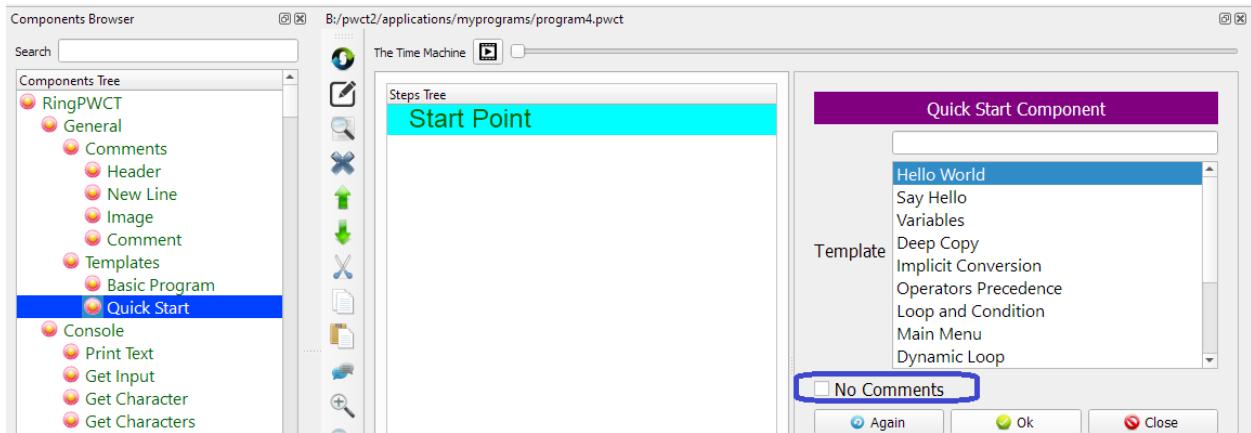
Selecting the (Hello World) Template will generate the next steps

No Comments checkbox : Active

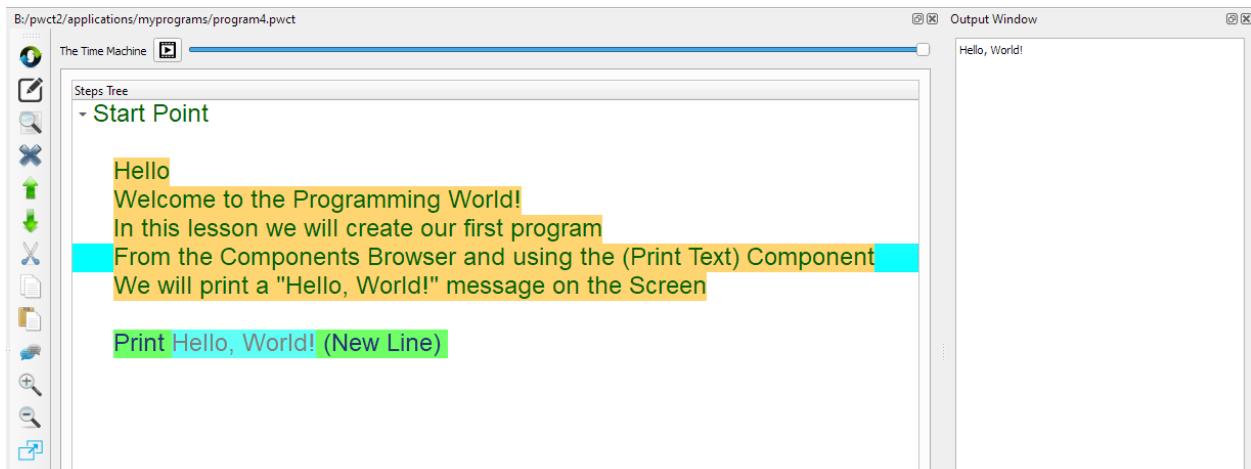


6.5 Adding Comments

When we use the Quick Start component we can disable the (No Comments) checkbox



This will generate the comments too



SAY HELLO PROGRAM

In this chapter we are going to learn how to create the Say Hello program

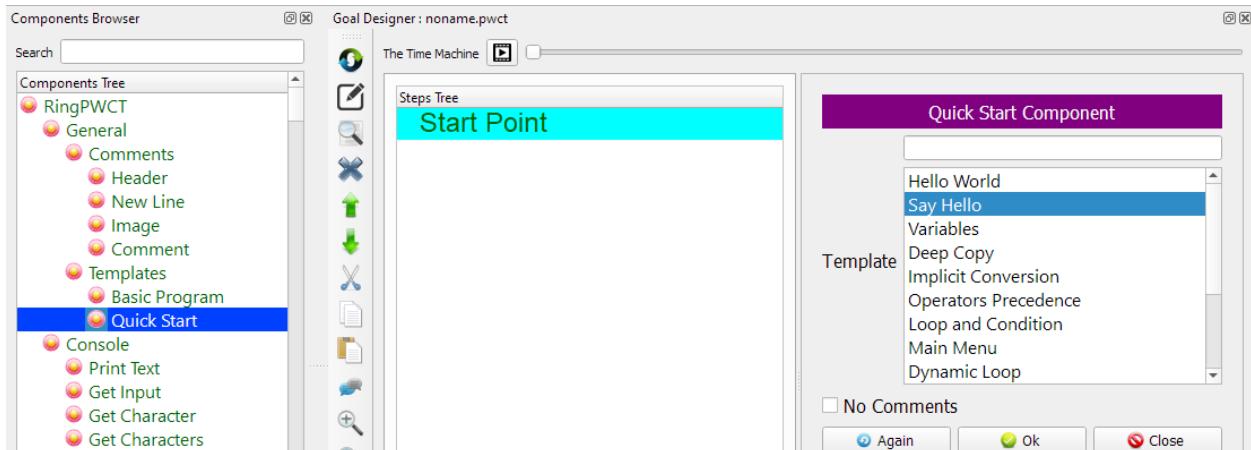
The User will enter his/her name then we will print (Hello UserName)

In this lesson we will learn about

- Variables
- Expressions
- Getting input using the Keyboard
- Printing Expressions that merge between two strings

7.1 Introduction

We can create this program quickly using the Quick Start component

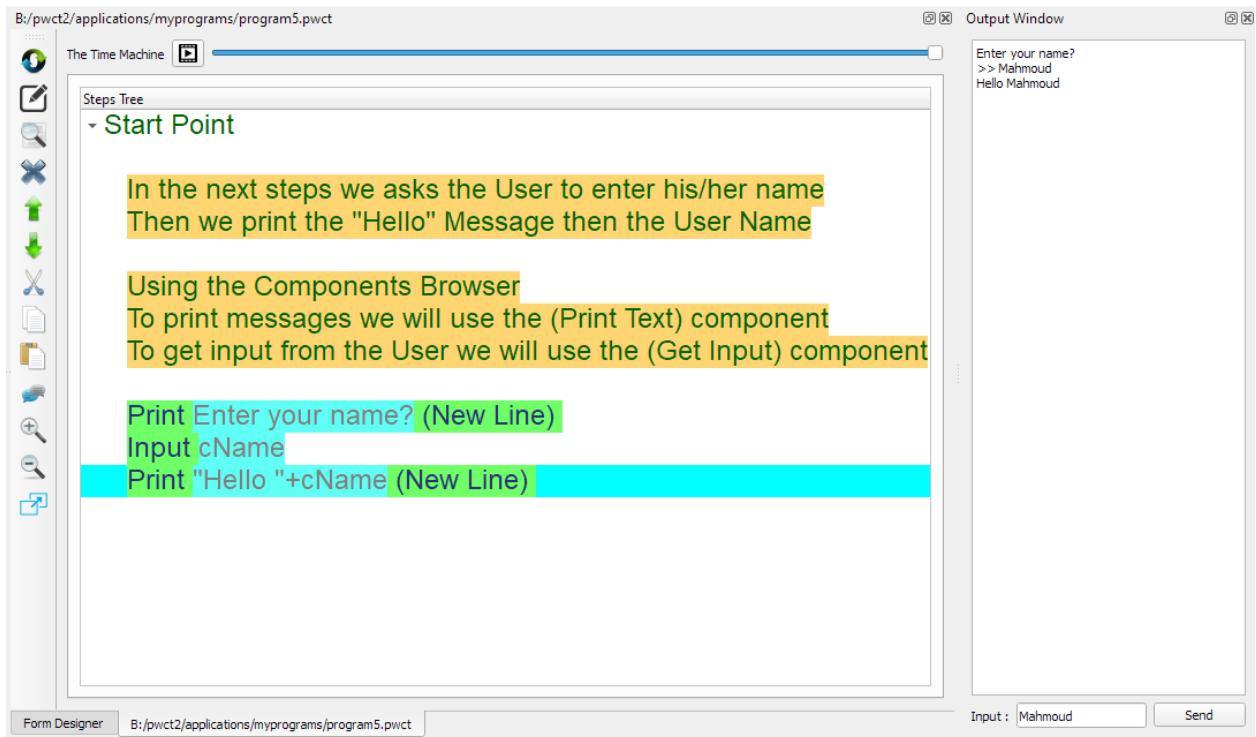


7.2 Program Steps

After selecting the (Say Hello) template, we will get the next steps in the Goal Designer

In the Output Window, The program prints the message (Enter your name?)

After typing (Mahmoud), We see the (Hello Mahmoud) message on the Screen

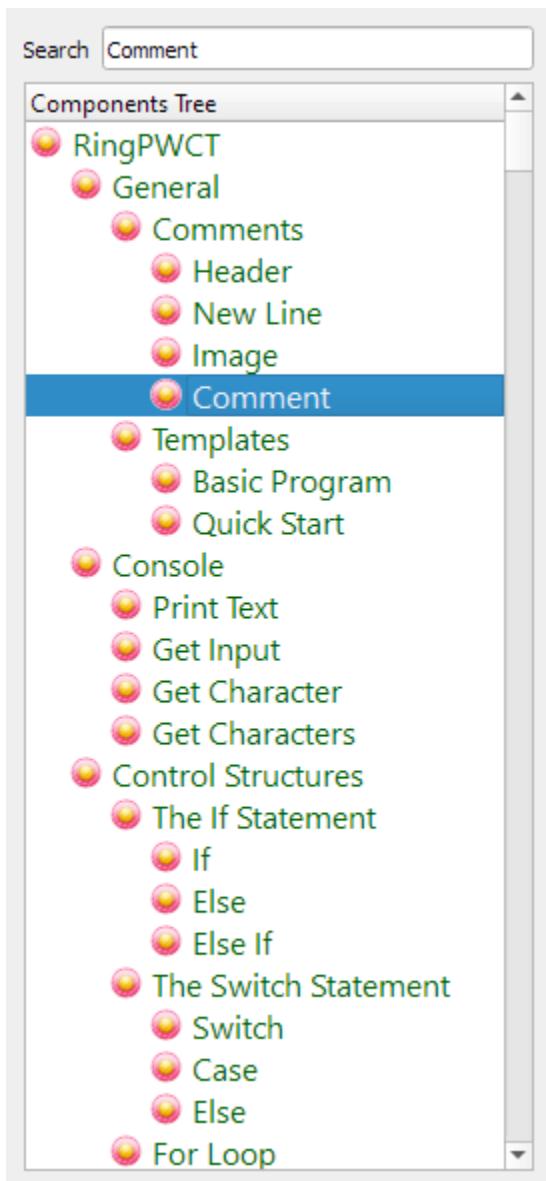


7.3 Creating the Program

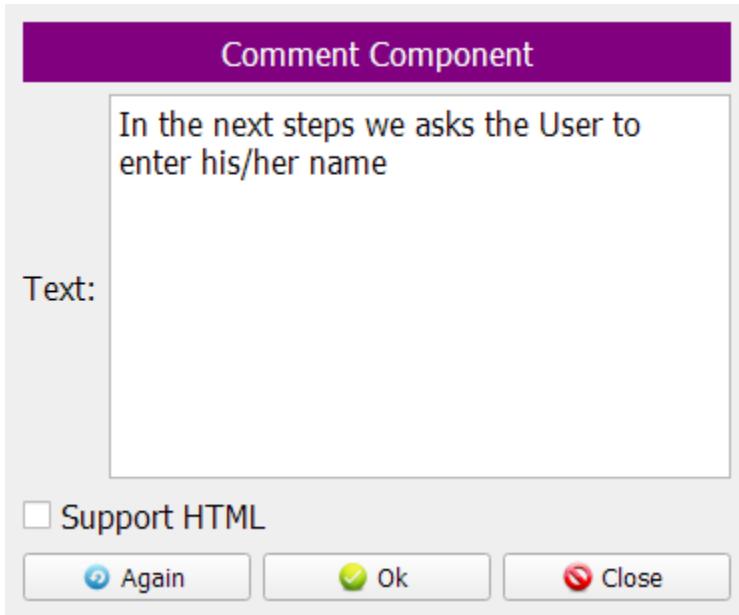
To create this program we will use the next components

- Comment (Optional)
- Print Text
- Get Input

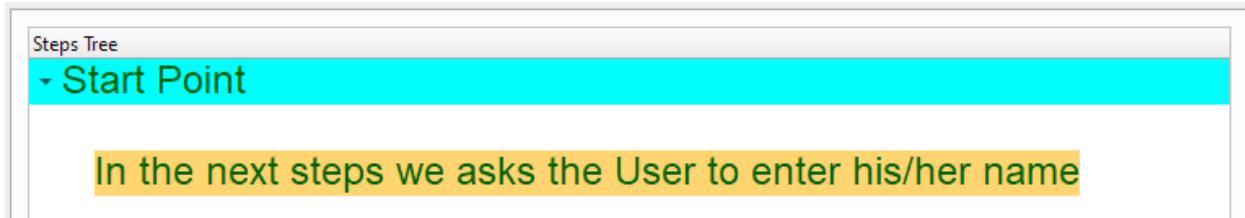
Using the Comment component, we will add some comments to our program!



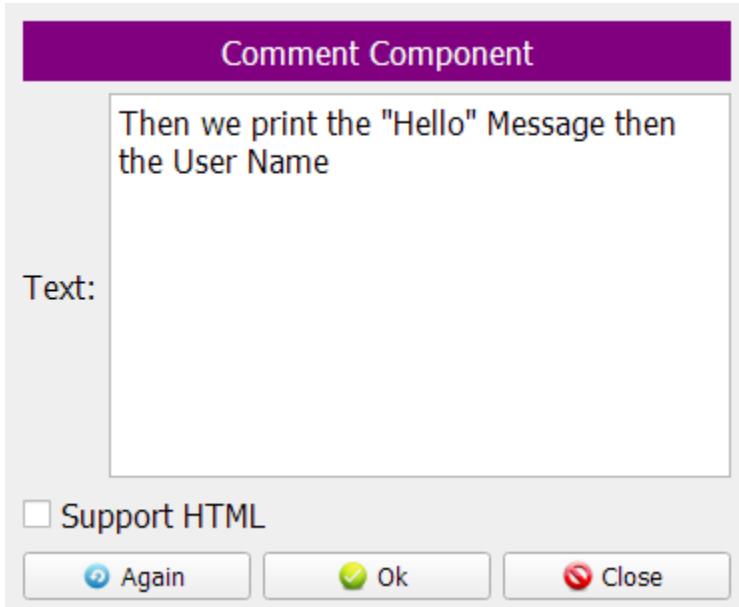
We can use the same component many times using the (Again) button



We will add more than one comment to the program



In the Interaction Page, We write the comment in the Text Field



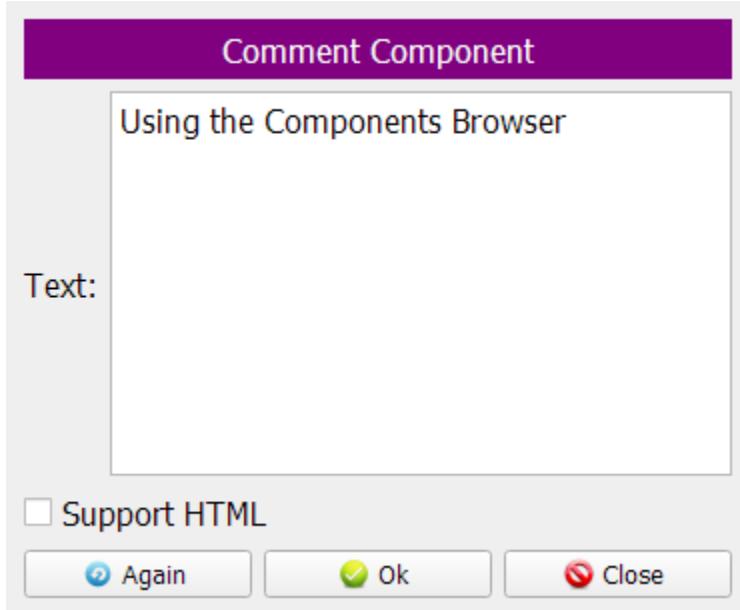
After writing the comment, The Steps Tree will be updated!

Steps Tree

- Start Point

In the next steps we asks the User to enter his/her name
Then we print the "Hello" Message then the User Name

Let's add the other comments



We can add empty lines to the Steps Tree by having empty comment

Empty lines could be a good separator between comments

Steps Tree

- Start Point

In the next steps we asks the User to enter his/her name
Then we print the "Hello" Message then the User Name

Using the Components Browser

Comment Component

To print messages we will use the (Print Text) component

Text:

Support HTML

 Again  Ok  Close

Steps Tree

- Start Point

In the next steps we asks the User to enter his/her name
Then we print the "Hello" Message then the User Name

Using the Components Browser
To print messages we will use the (Print Text) component

Comment Component

To get input from the User we will use the (Get Input) component

Text:

Support HTML

 Again  Ok  Close

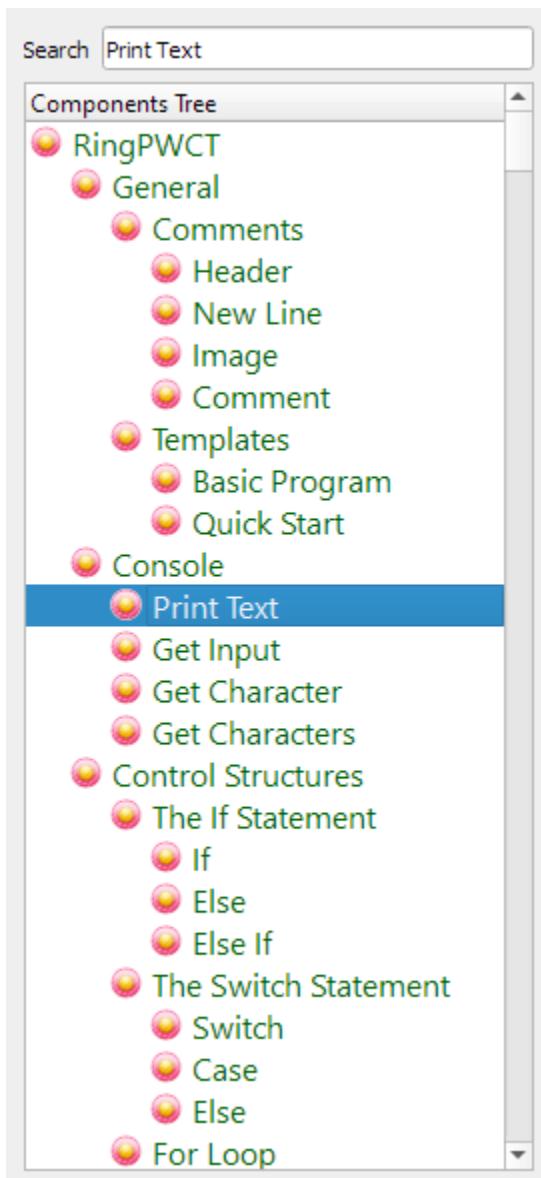
Steps Tree

- Start Point

In the next steps we ask the User to enter his/her name
Then we print the "Hello" Message then the User Name

Using the Components Browser
To print messages we will use the (Print Text) component
To get input from the User we will use the (Get Input) component

We will use the (Print Text) component



Print Component

Type : Text : Enter your name?

Type :

Type : New Line

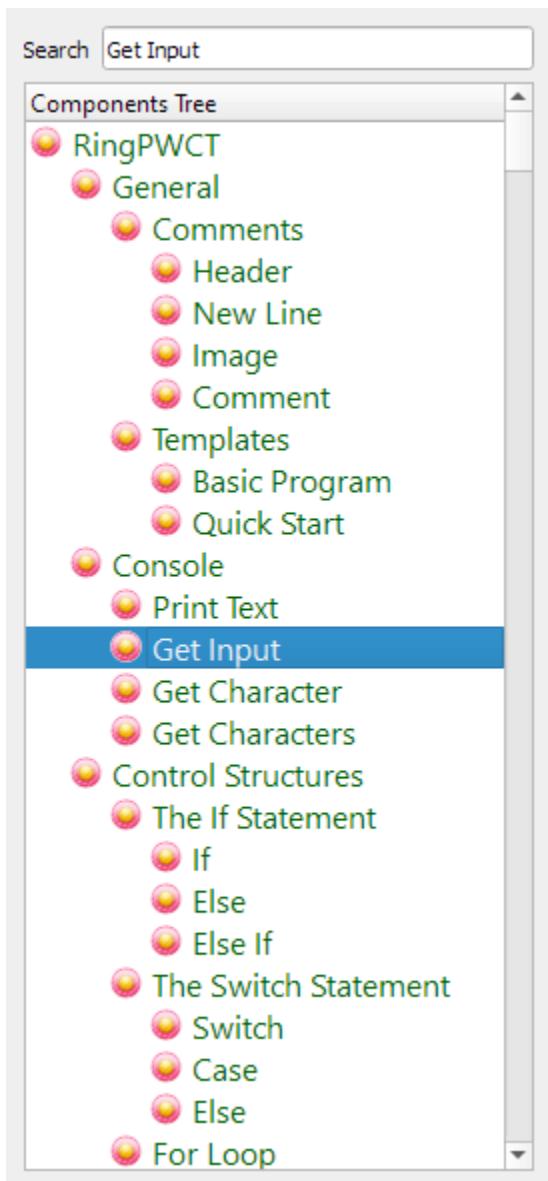
Steps Tree
- Start Point

In the next steps we asks the User to enter his/her name
Then we print the "Hello" Message then the User Name

Using the Components Browser
To print messages we will use the (Print Text) component
To get input from the User we will use the (Get Input) component

Print Enter your name? (New Line)

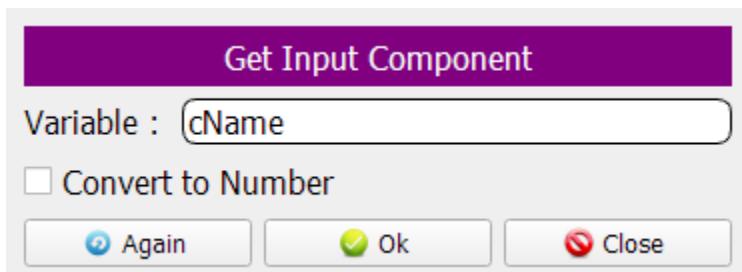
Select the (Get Input) Component



Type (cName) as the Variable Name

Variables are used to store data in the Computer Memory

Using the variable name we can set or get the variable value



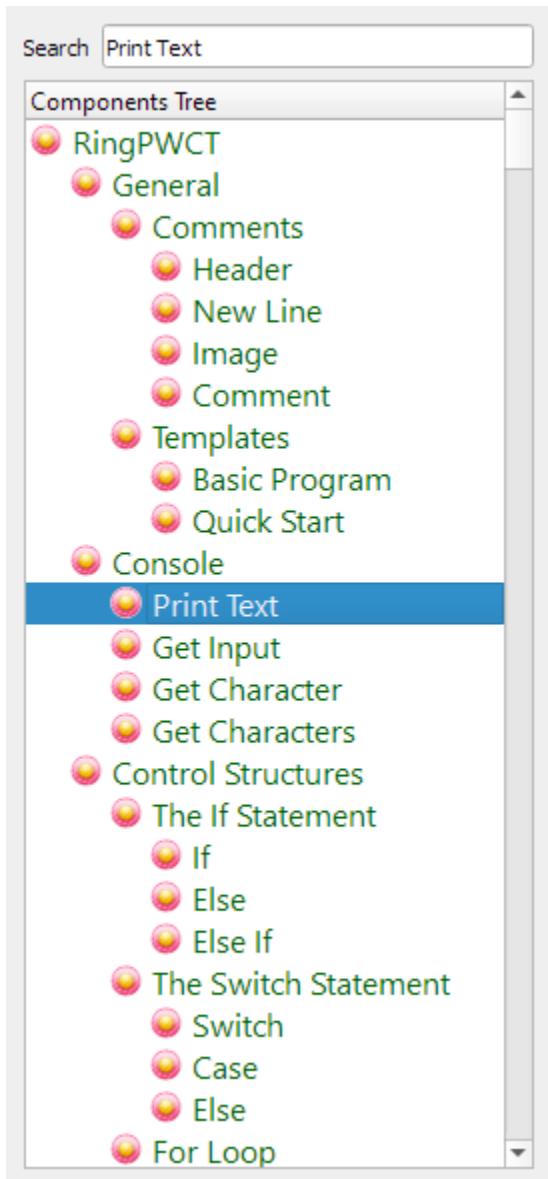
Steps Tree

- Start Point

In the next steps we asks the User to enter his/her name
Then we print the "Hello" Message then the User Name

Using the Components Browser
To print messages we will use the (Print Text) component
To get input from the User we will use the (Get Input) component

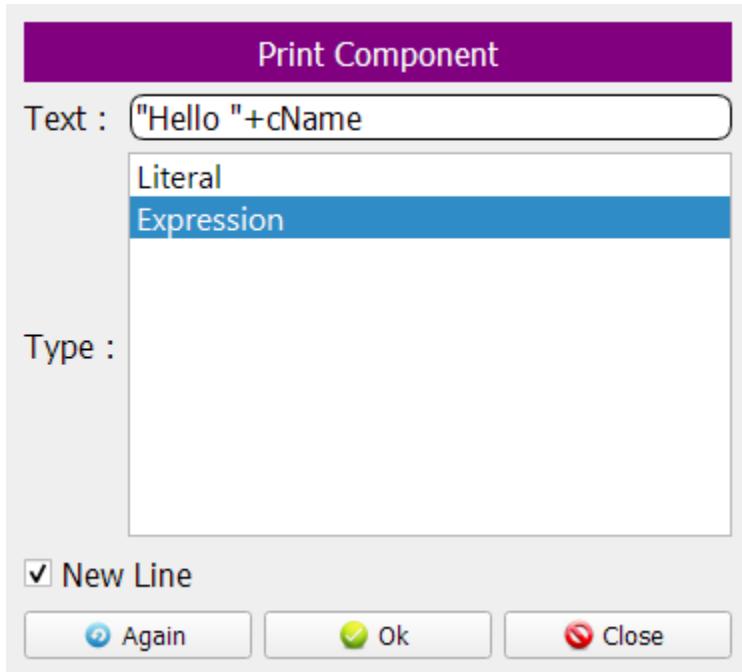
Print Enter your name? (New Line)
Input cName



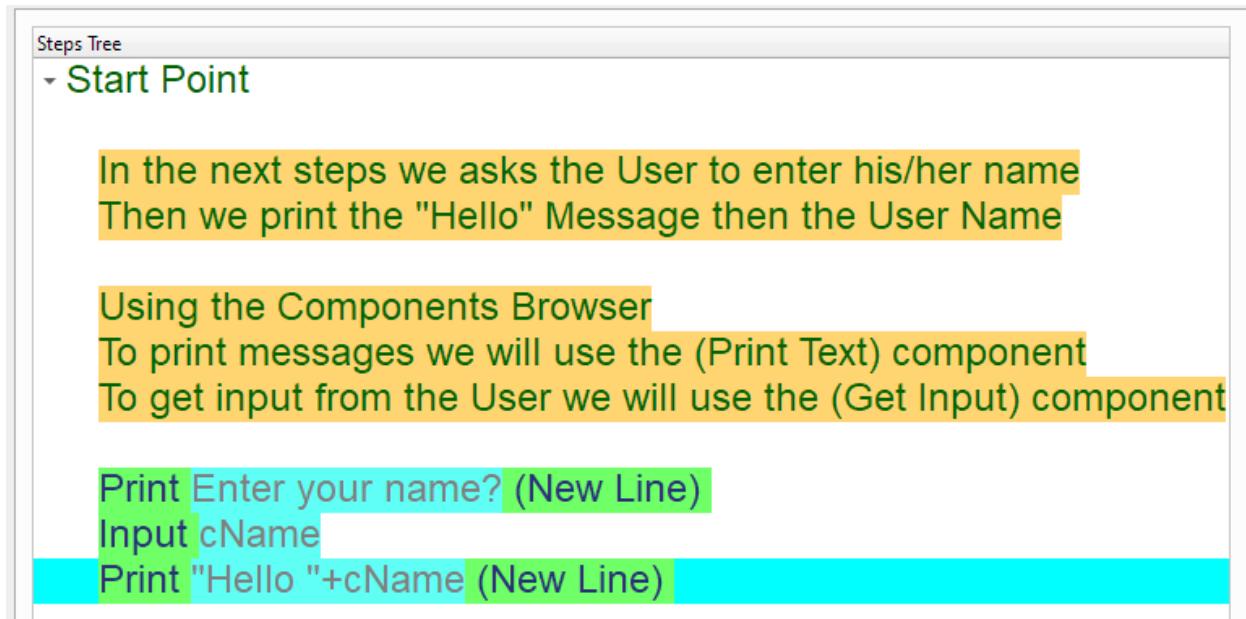
Now using the (Print Text) component we will print an Expression

The Expression is a mix of Variables, Values & Operators

In this example we use the Plus operator to merge between strings



Now we have the final Steps Tree in our program



**CHAPTER
EIGHT**

USING VARIABLES

In this chapter we are going to learn how to use the Variables

To create a new variable, you just need to determine the variable name & value. The value will determine the variable type and you can change the value to switch between the types using the same variable name.

Generated Step:

```
<Variable Name> = <Value>
```

Tip: The operator ‘=’ is used here as an Assignment operator and the same operator can be used in conditions, but for testing equality of expressions.

Note: The Variable will contains the real value (not a reference). This means that once you change the variable value, the old value will be removed from memory (even if the variable contains a list or object).

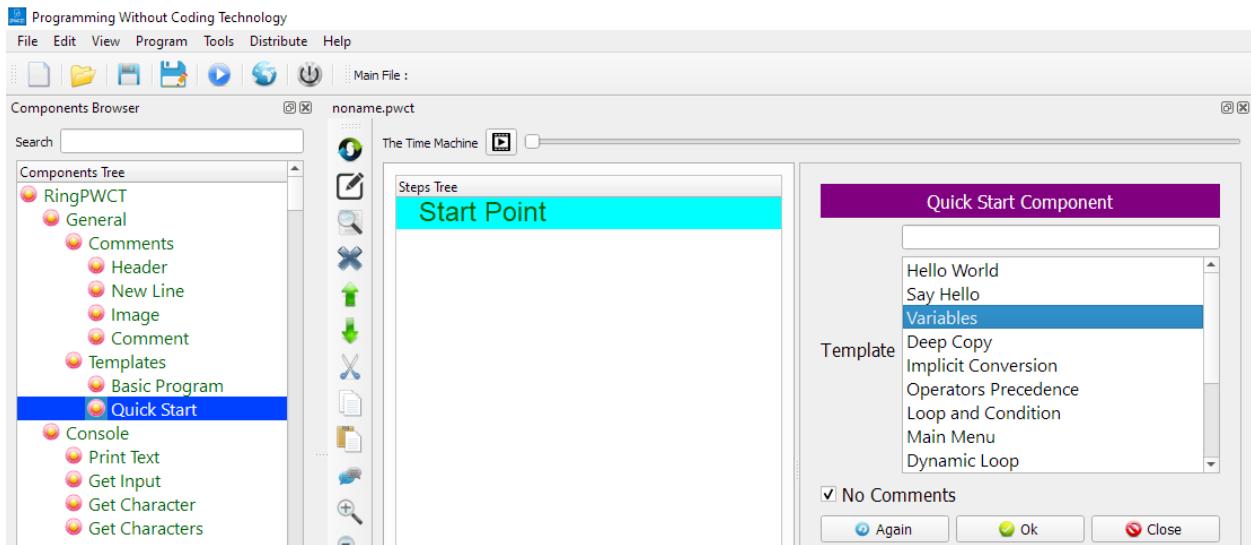
The variable type is based on the value, and this value could be

- String (One Character, Many Characters, Many Lines, Binary Data)
- Number (Signed Integer, Unsigned Integer, Double, Boolean)
- List (List of one type, List of many types, Nested Lists)
- Object

8.1 Introduction

We will create a simple program to learn how to use the Variables

We can create this program quickly using the Quick Start component



8.2 Program Steps

After selecting the (Variables) template, we will get the next steps in the Goal Designer



8.3 Creating the Program

To create this program we will use the next components

- Assignment
- Print Text

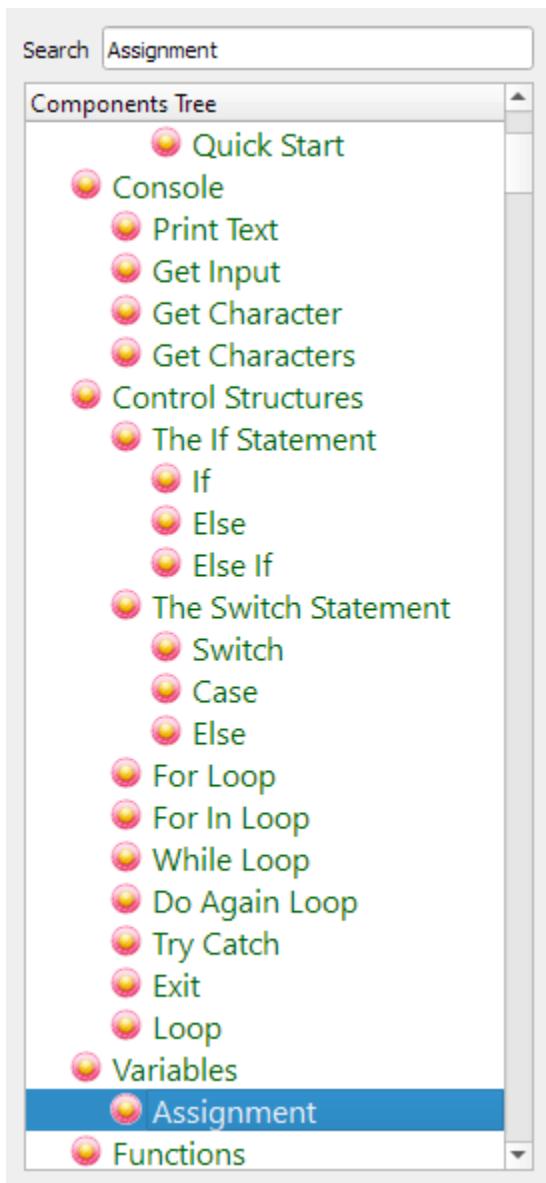
At first we will define a variable called X

The Variable value will be “Hello” which is a String

In the begining the Steps Tree is empty



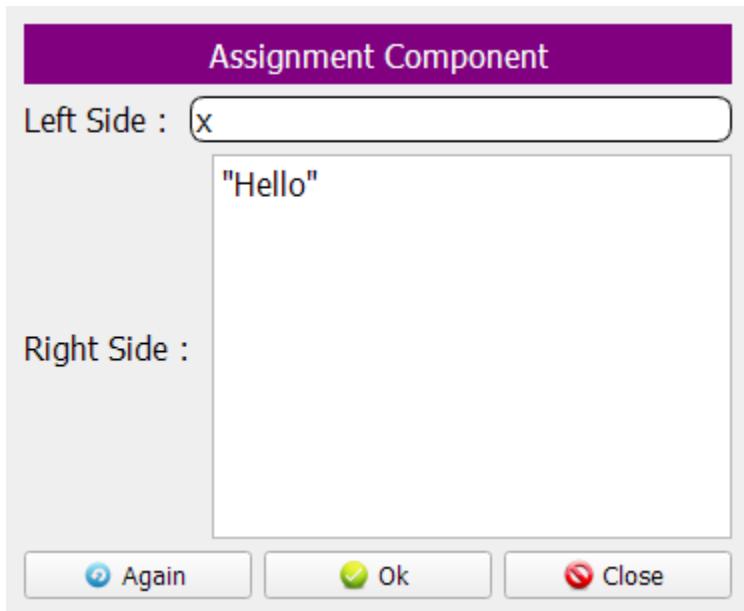
Select the (Assignment) component



Enter the data in the Interaction Page

Left Side: X

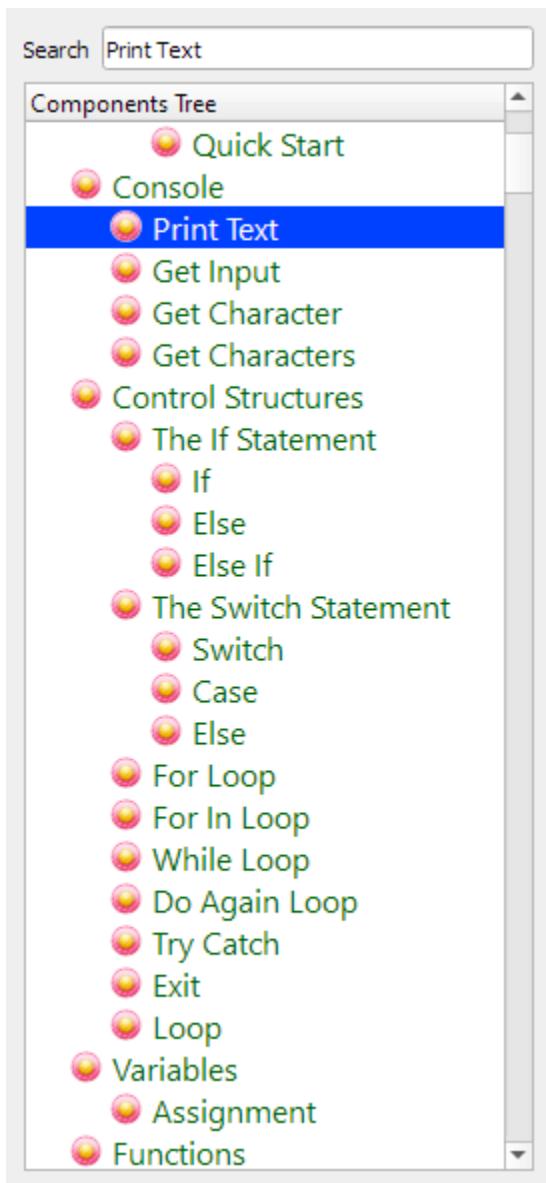
Right Side: "Hello"



A new step will be added to the Steps Tree



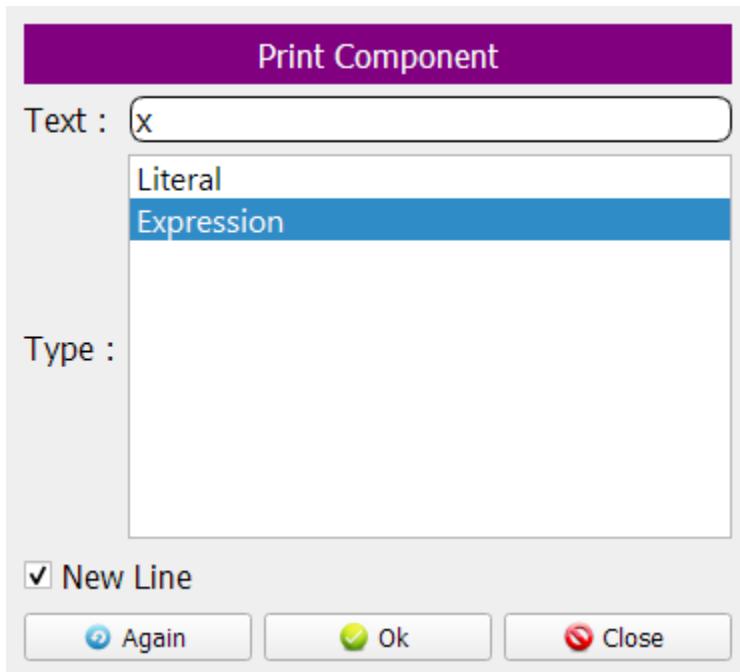
Let's print the variable using the (Print Text) component



Enter the data in the Interaction Page

Text: X

Type: Expression

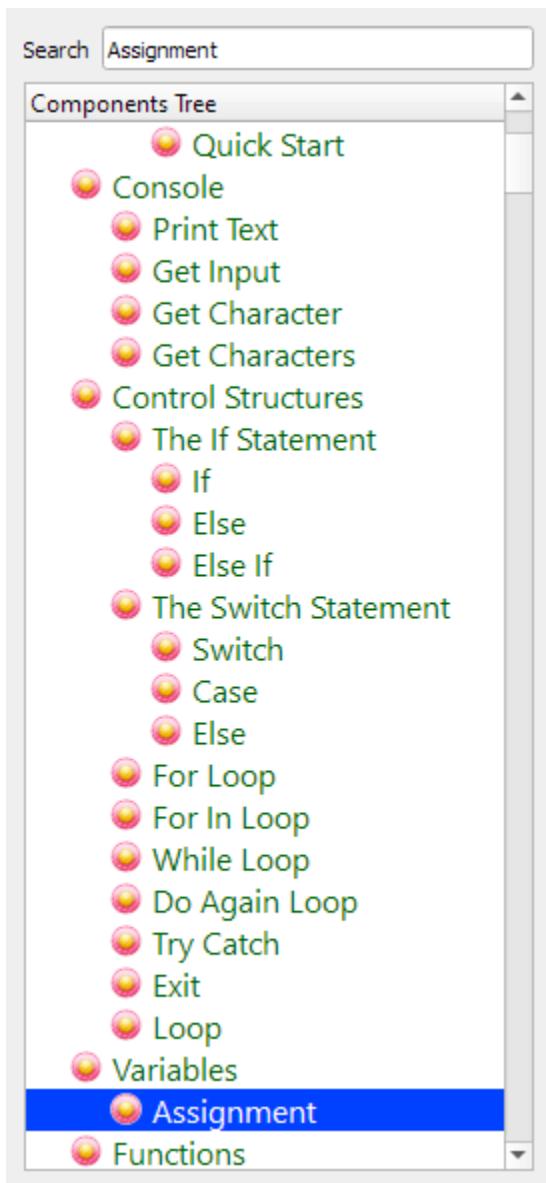


The Steps Tree is updated!



Now we will set X to 5

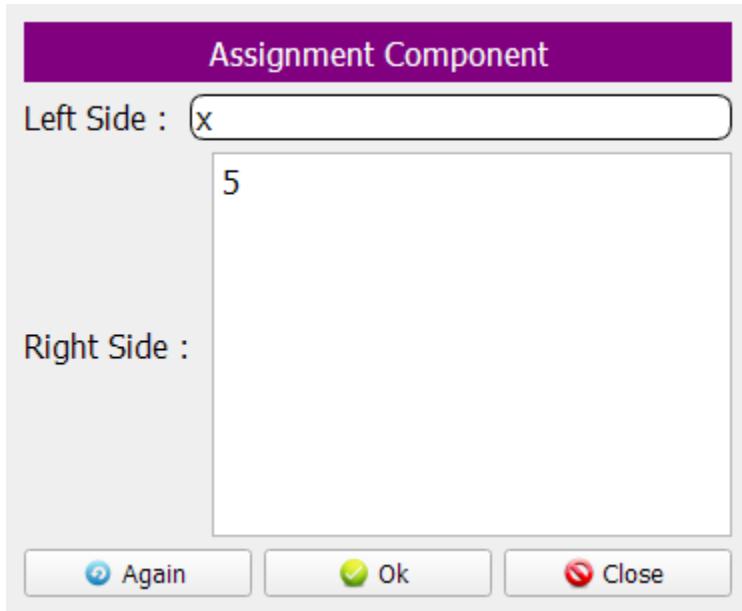
X type will be a Number (The type is based on the value)



Enter the data in the Interaction Page

Left Side: X

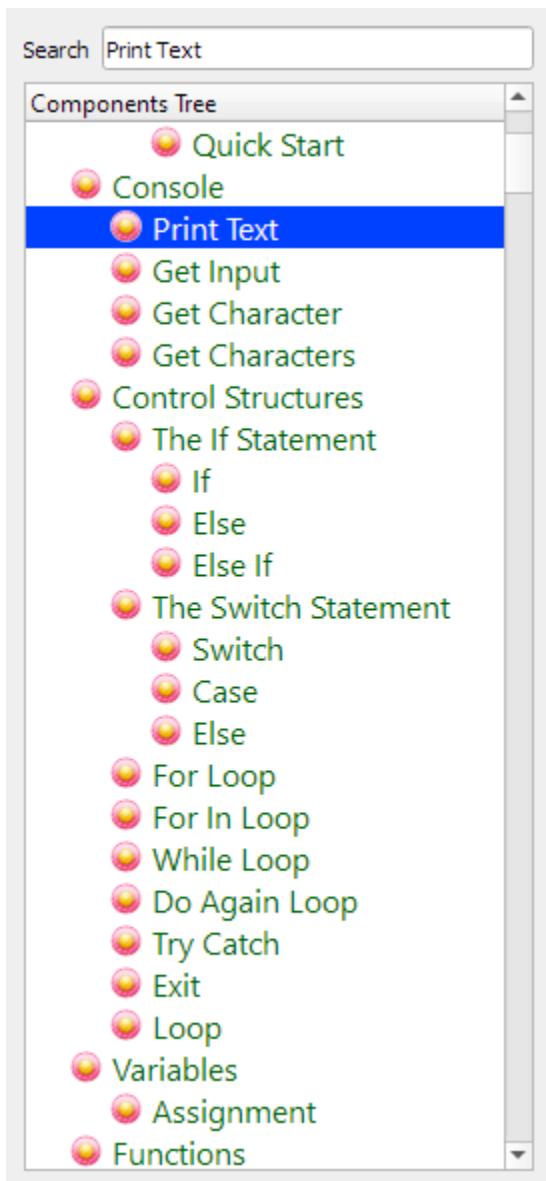
Right Side: 5



The Steps Tree is updated!



Let's print the variable using the (Print Text) component



Print Component

Text :

Type :

New Line

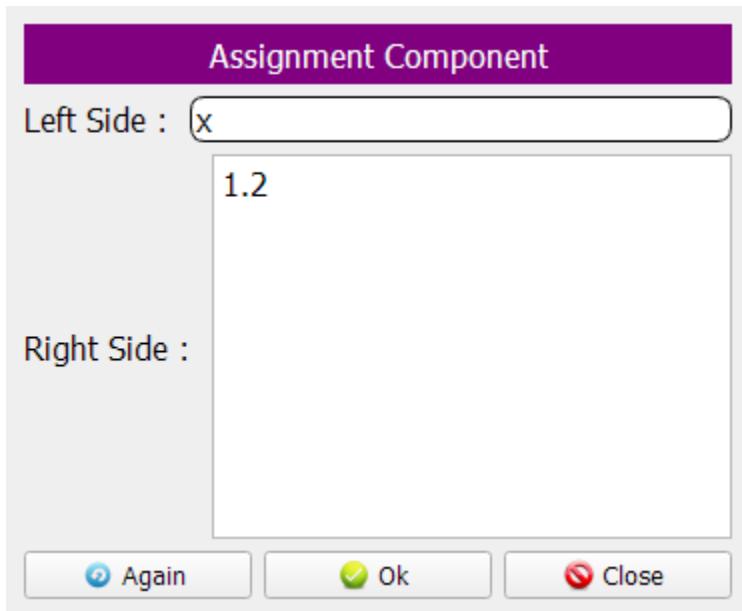
Steps Tree

- Start Point

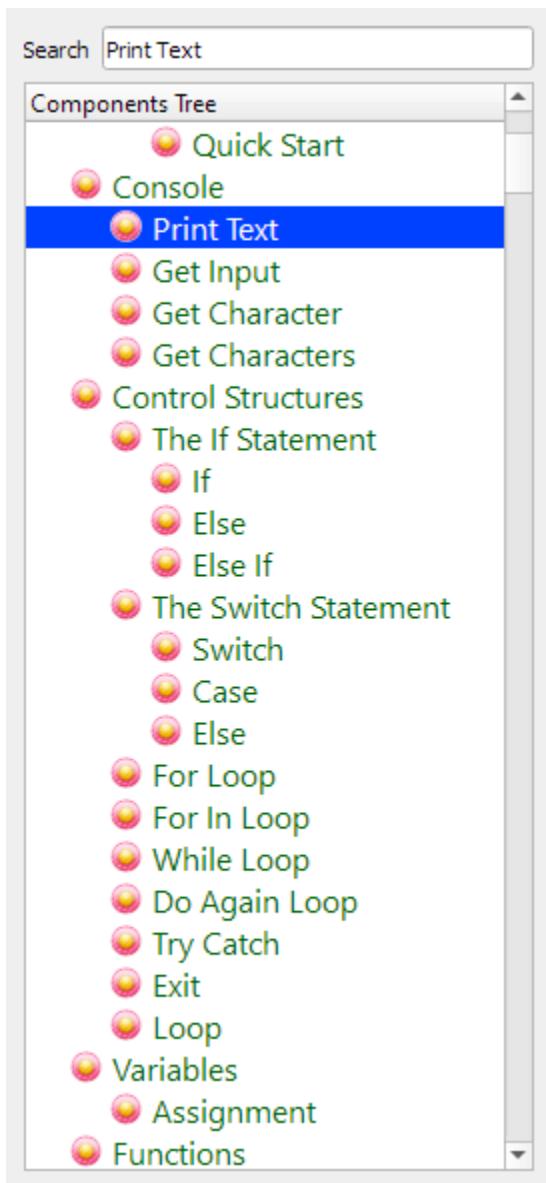
- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)

Set X to 1.2





We will print the X value



Print Component

Text :

Type :

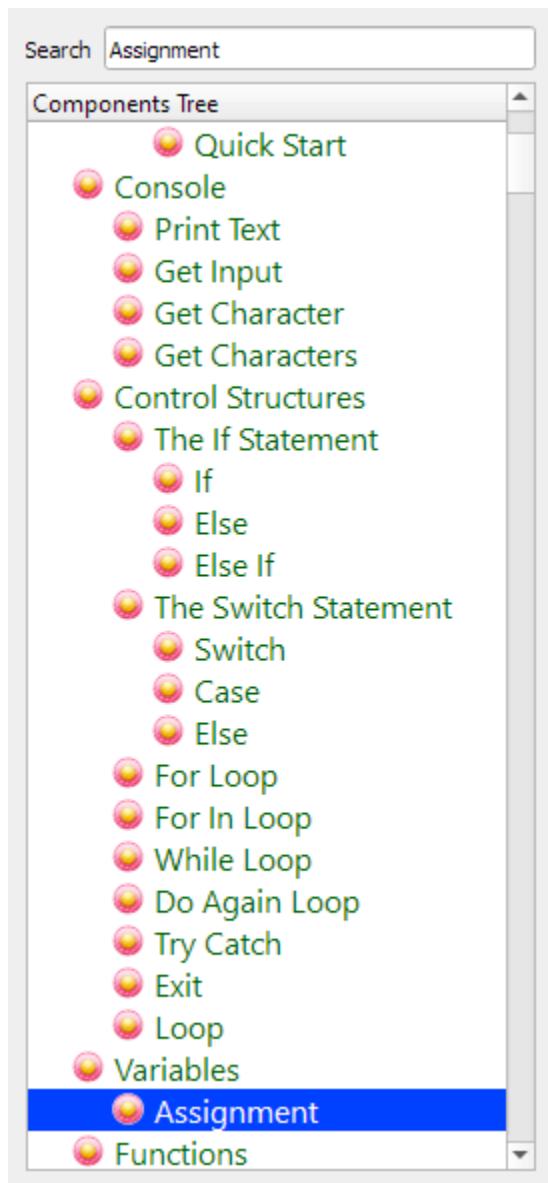
New Line

Steps Tree

- Start Point

- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)

X will be a List of four items [1,2,3,4]



Assignment Component

Left Side :

[1,2,3,4]

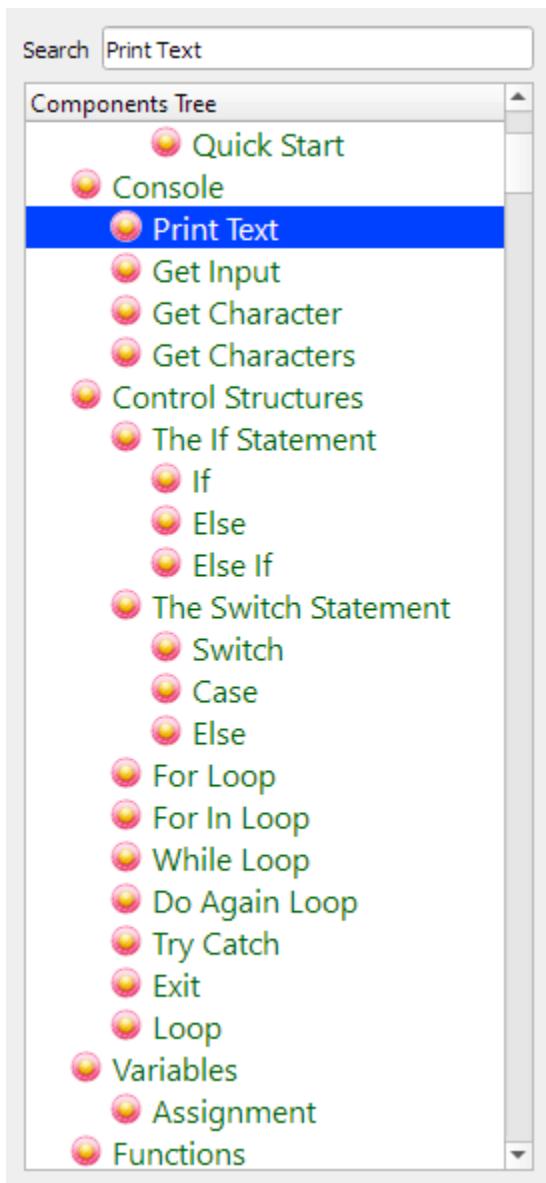
Right Side :

Steps Tree

- Start Point

```
x = "Hello"
Print x (New Line)
x = 5
Print x (New Line)
x = 1.2
Print x (New Line)
x = [1,2,3,4]
```

Print the X value



Print Component

Text :

Type :

New Line

Steps Tree

- Start Point

- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)
- x = [1,2,3,4]
- Print x (New Line)

X will be a String that contains the Date



Assignment Component

Left Side :

date()

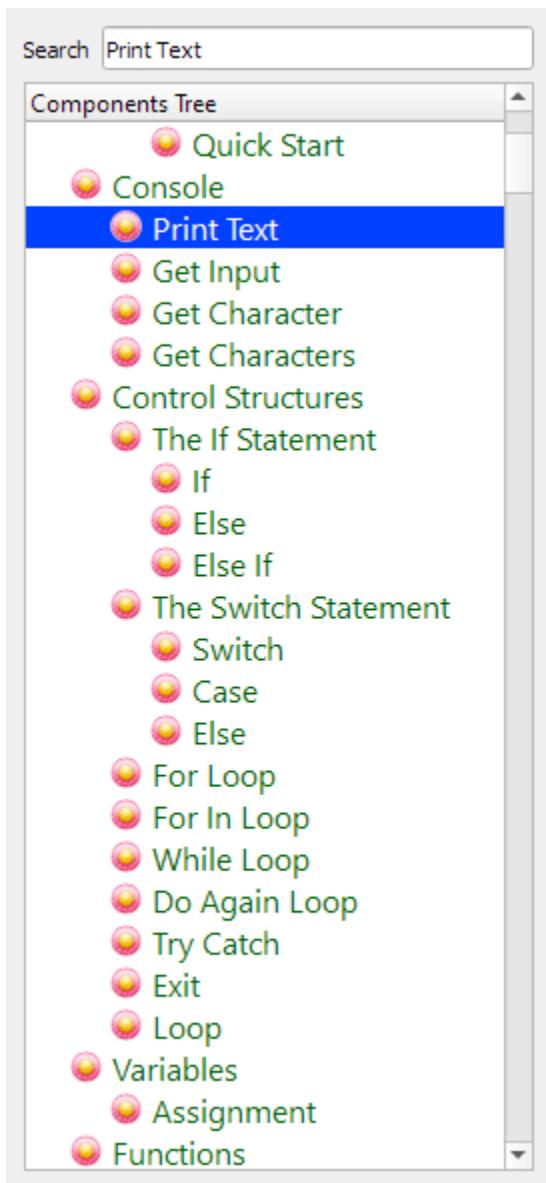
Right Side :

Steps Tree

- Start Point

- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)
- x = [1,2,3,4]
- Print x (New Line)
- x = date()

Print X value (The Date)



Print Component

Text :

Type :

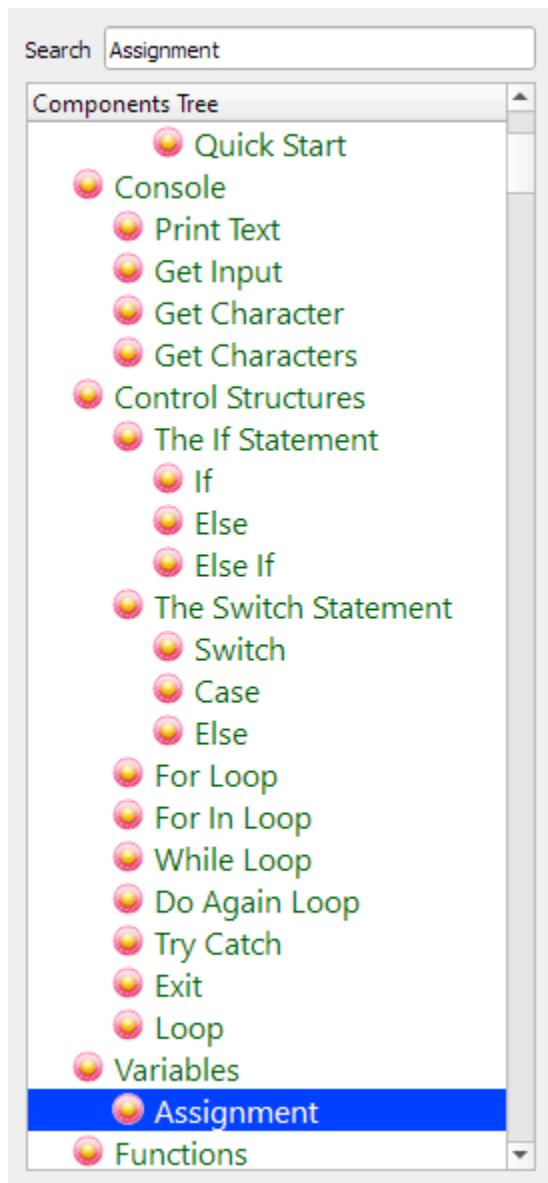
New Line

Steps Tree

- Start Point

- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)
- x = [1,2,3,4]
- Print x (New Line)
- x = date()
- Print x (New Line)

X will be a String contains the Time



Assignment Component

Left Side :

time()

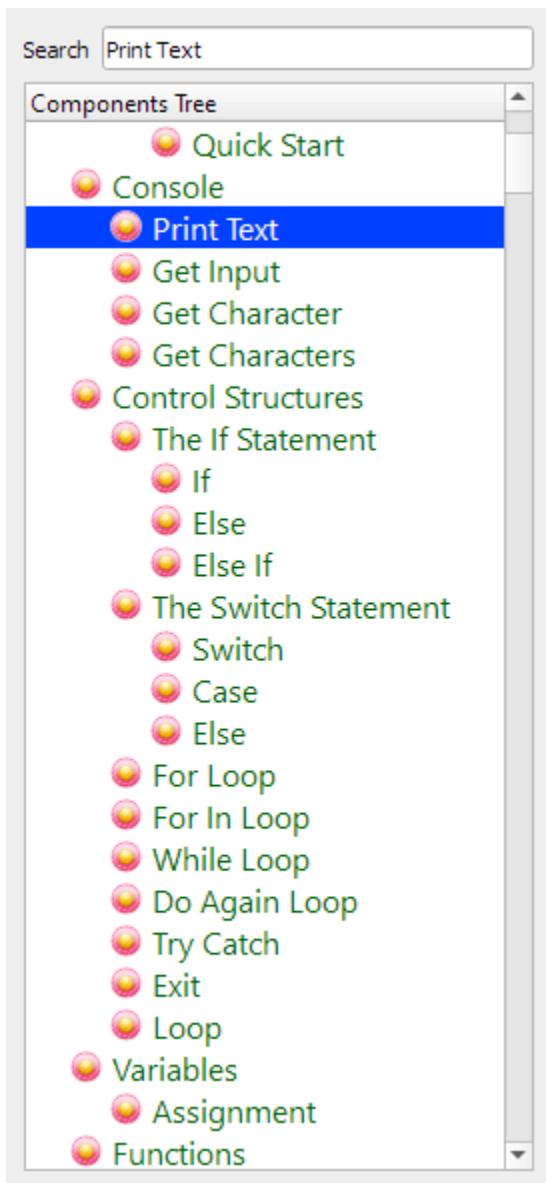
Right Side :

Steps Tree

- Start Point

- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)
- x = [1,2,3,4]
- Print x (New Line)
- x = date()
- Print x (New Line)
- x = time()

Print X value (The Time)



Print Component

Text :

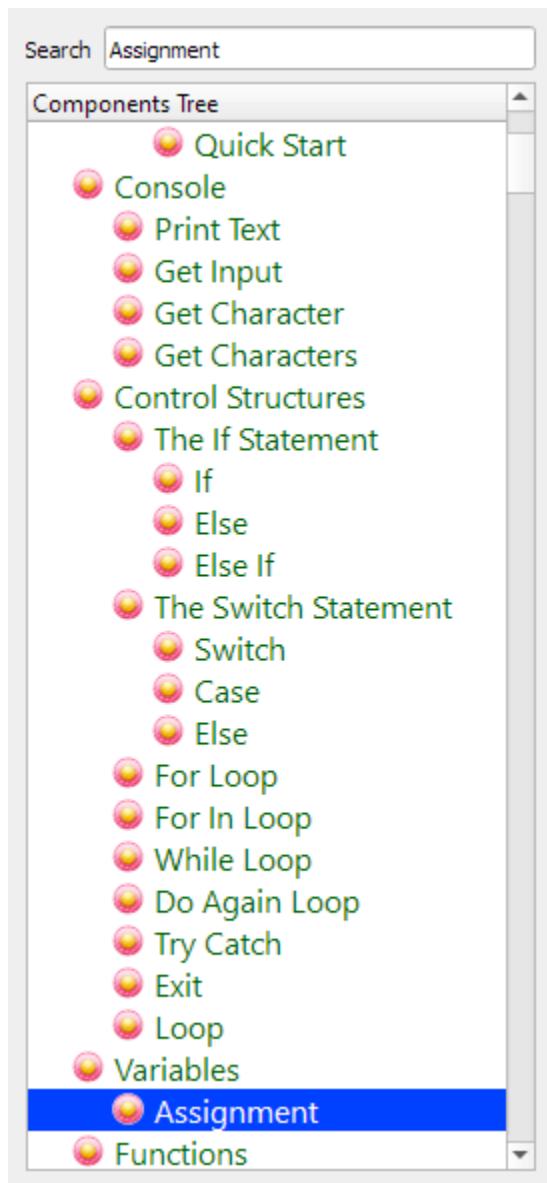
Type :

New Line

Steps Tree

- Start Point
 - x = "Hello"
 - Print x (New Line)
 - x = 5
 - Print x (New Line)
 - x = 1.2
 - Print x (New Line)
 - x = [1,2,3,4]
 - Print x (New Line)
 - x = date()
 - Print x (New Line)
 - x = time()
 - Print x (New Line)

X will be True (Number: 1)



Assignment Component

Left Side :

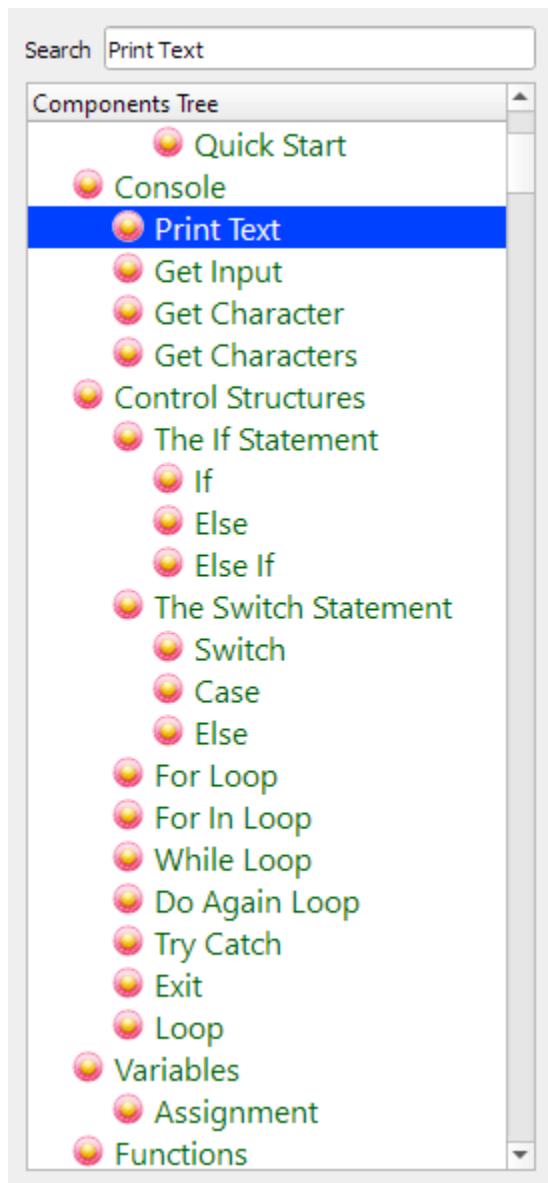
Right Side :

Steps Tree

- Start Point

- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)
- x = [1,2,3,4]
- Print x (New Line)
- x = date()
- Print x (New Line)
- x = time()
- Print x (New Line)
- x = true

Print X value (Will print 1)



Print Component

Text :

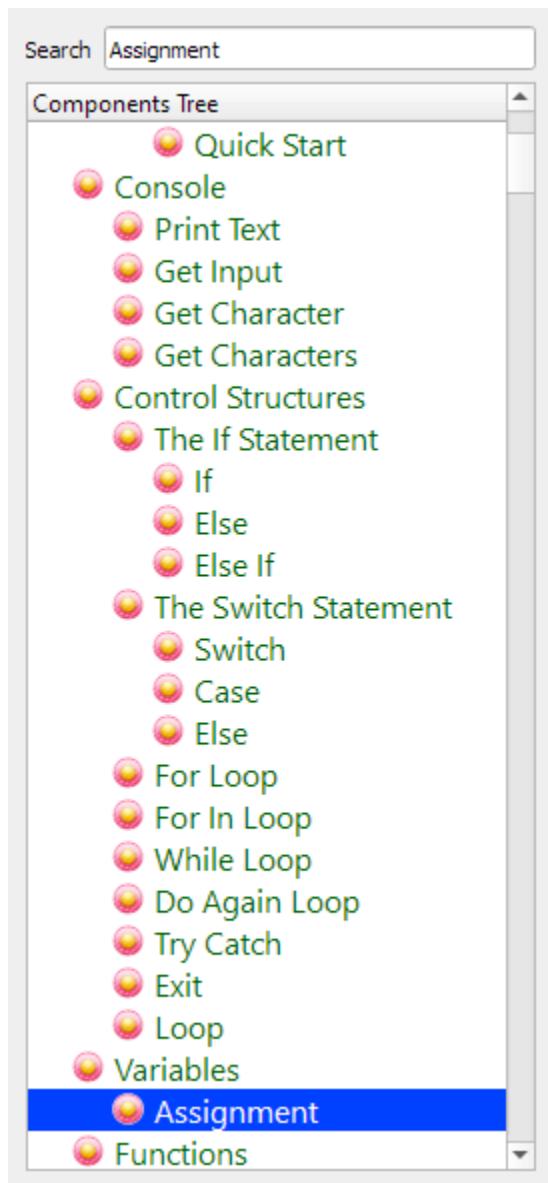
Type :

New Line

Steps Tree

- Start Point
 - x = "Hello"
 - Print x (New Line)
 - x = 5
 - Print x (New Line)
 - x = 1.2
 - Print x (New Line)
 - x = [1,2,3,4]
 - Print x (New Line)
 - x = date()
 - Print x (New Line)
 - x = time()
 - Print x (New Line)
 - x = true
 - Print x (New Line)

X will be False (Number: 0)



Assignment Component

Left Side :

false

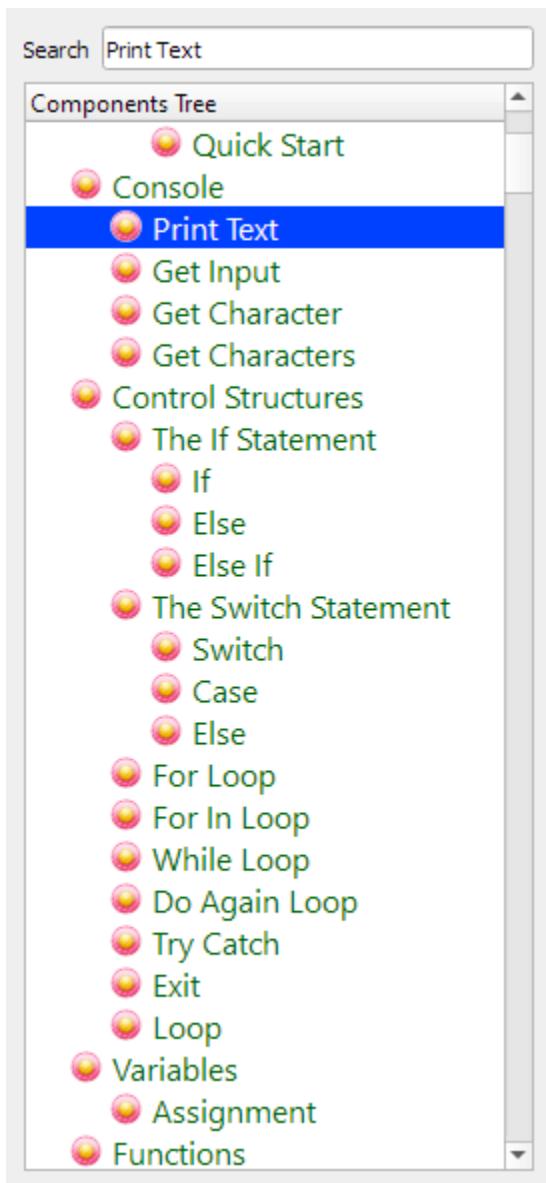
Right Side :

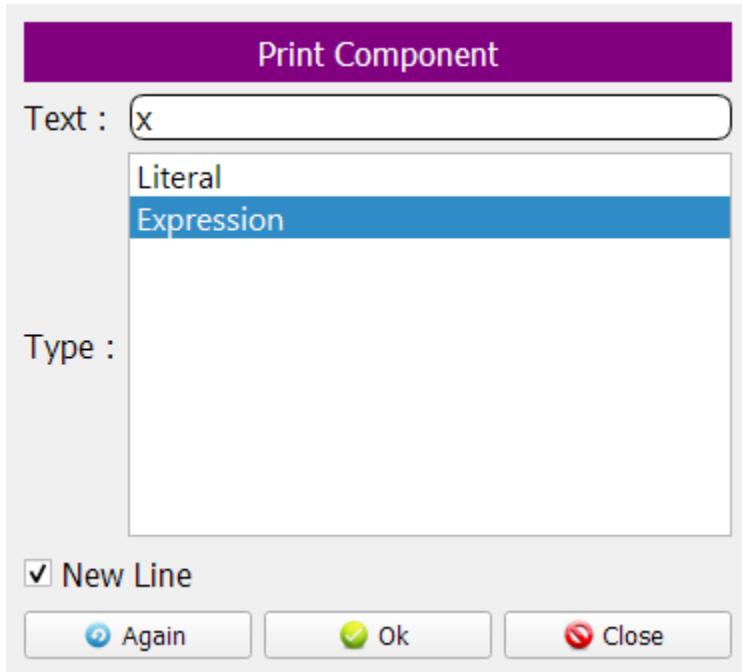
Steps Tree

- Start Point

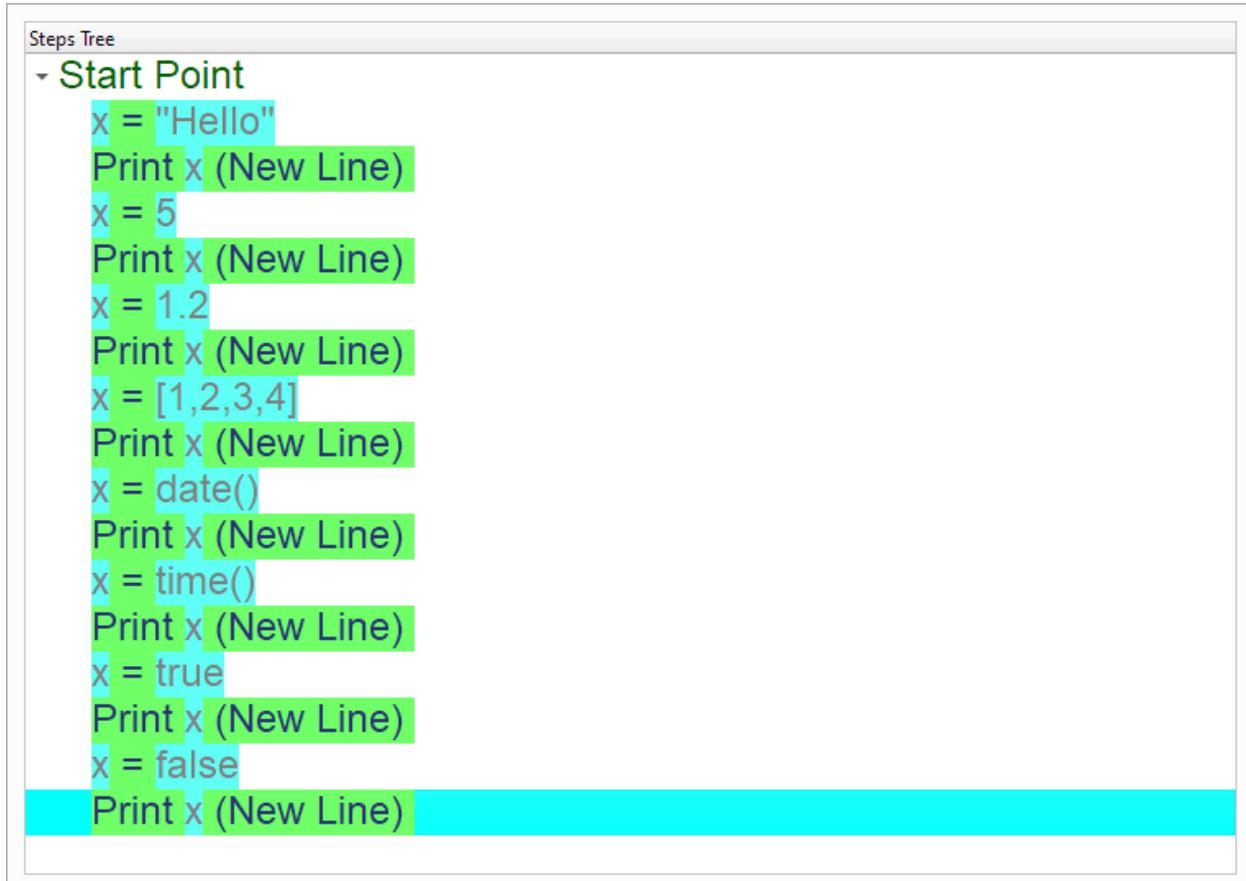
- x = "Hello"
- Print x (New Line)
- x = 5
- Print x (New Line)
- x = 1.2
- Print x (New Line)
- x = [1,2,3,4]
- Print x (New Line)
- x = date()
- Print x (New Line)
- x = time()
- Print x (New Line)
- x = true
- Print x (New Line)
- x = false

Print X value (Will print 0)





Now we have the final Steps Tree in our program



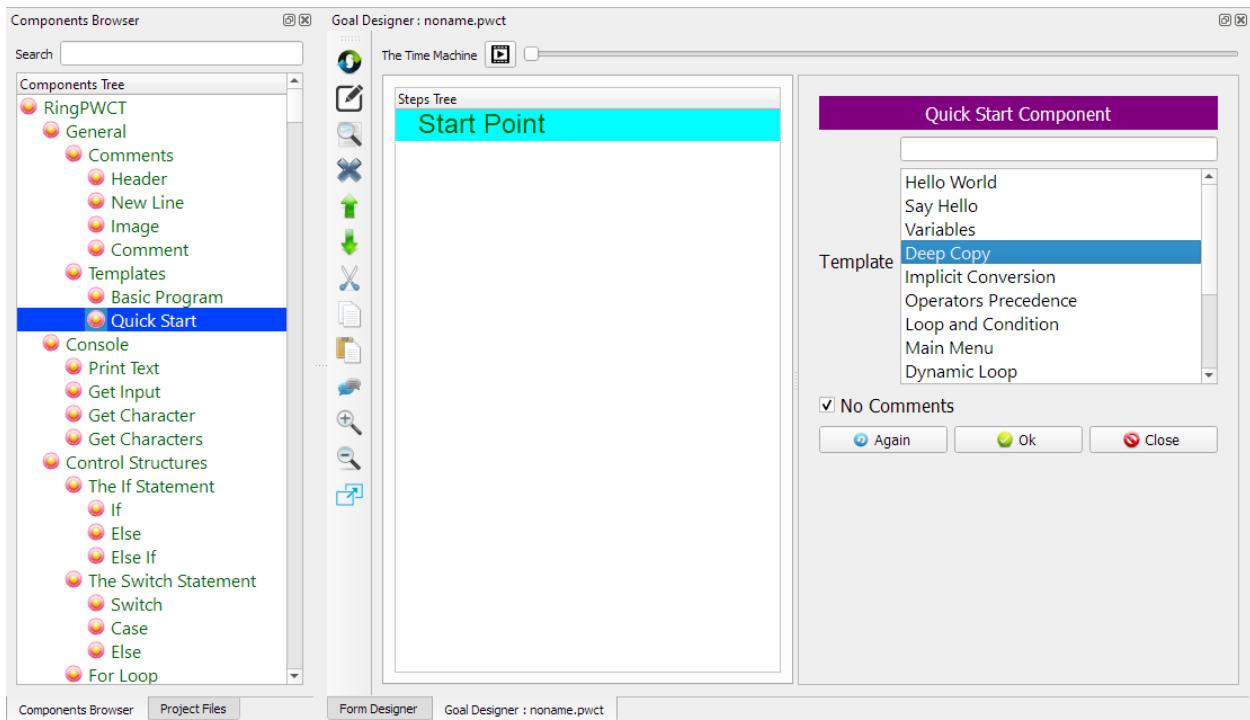
DEEP COPY

In this chapter we are going to learn how to copy lists using the Assignment component

When we copy a list we have a Deep Copy (Copy by Value & Each copy is isolated)

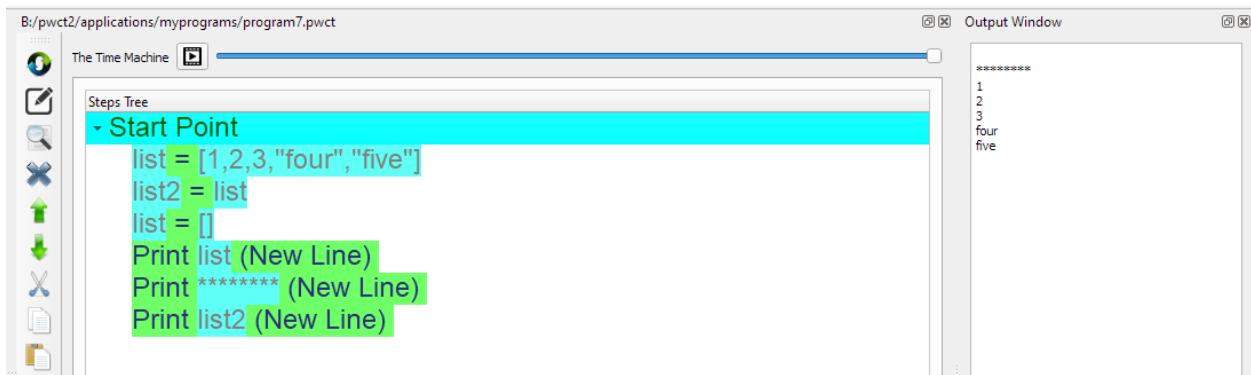
9.1 Introduction

We can create this program quickly using the Quick Start component



9.2 Program Steps

After selecting the (Deep Copy) template, we will get the next steps in the Goal Designer



The screenshot shows the PWCT Goal Designer interface. The title bar says "B:/pwct2/applications/myprograms/program7.pwct". The left sidebar has icons for Time Machine, Steps Tree, and other tools. The main area is titled "Steps Tree" and contains a tree view with a node labeled "Start Point". Under "Start Point", there is a list of code steps:

```

list = [1,2,3,"four","five"]
list2 = list
list = []
Print list (New Line)
Print ***** (New Line)
Print list2 (New Line)

```

To the right is an "Output Window" showing the results of the execution:

```

*****
1
2
3
four
five

```

9.3 Creating the Program

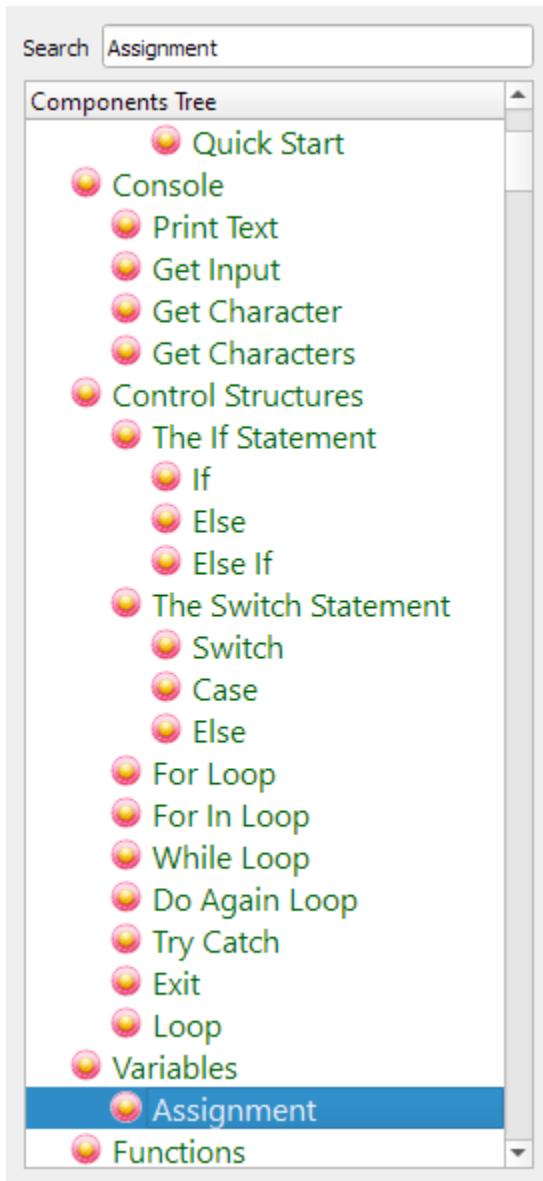
To create this program we will use the next components

- Assignment
- Print Text

In the Start, The Steps Tree will be Empty



Select the (Assignment) component



We will define a list called (list)

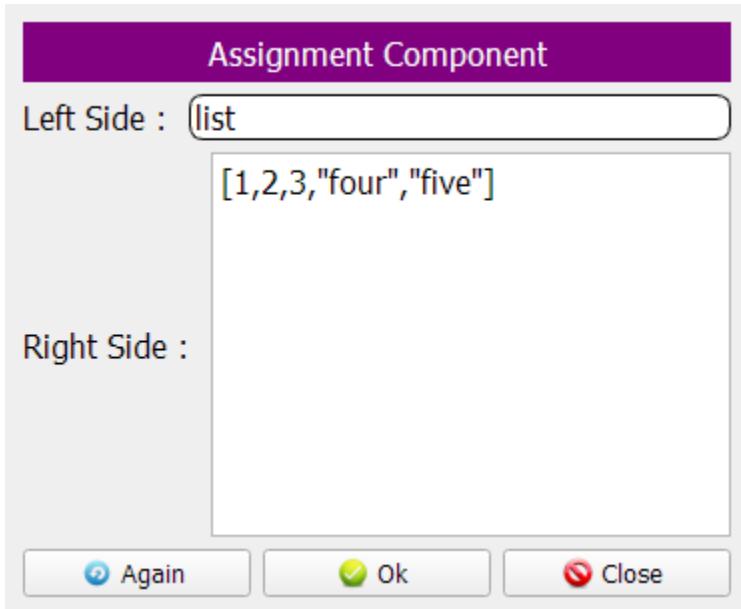
The list will contain five items

The first three items are numbers (1,2,3)

The last two items are strings ("four","five")

Left side: list

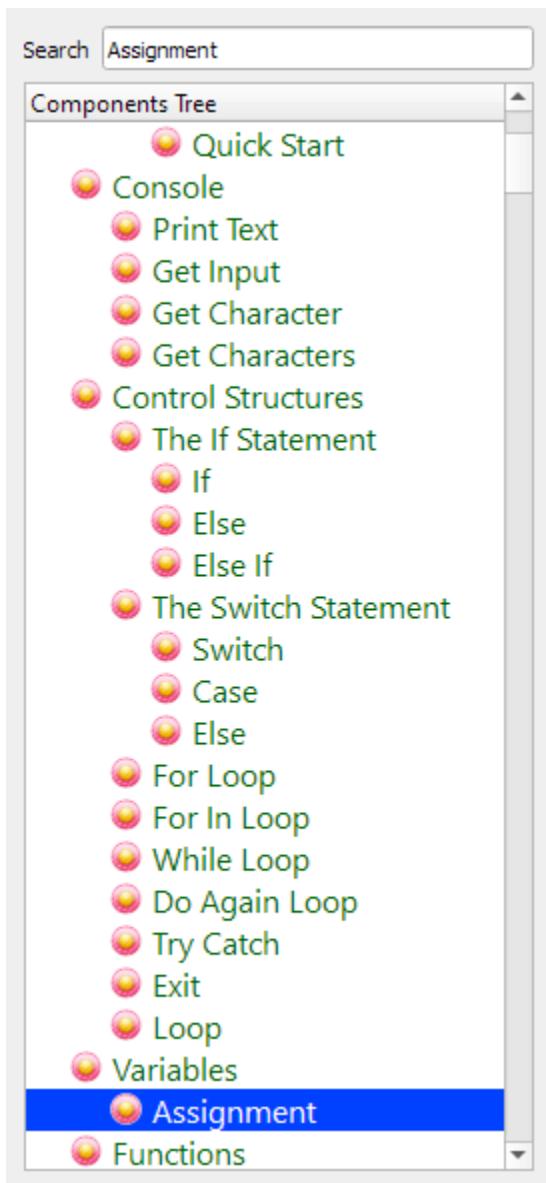
Right side: [1,2,3,"four","five"]



The Steps Tree will be updated



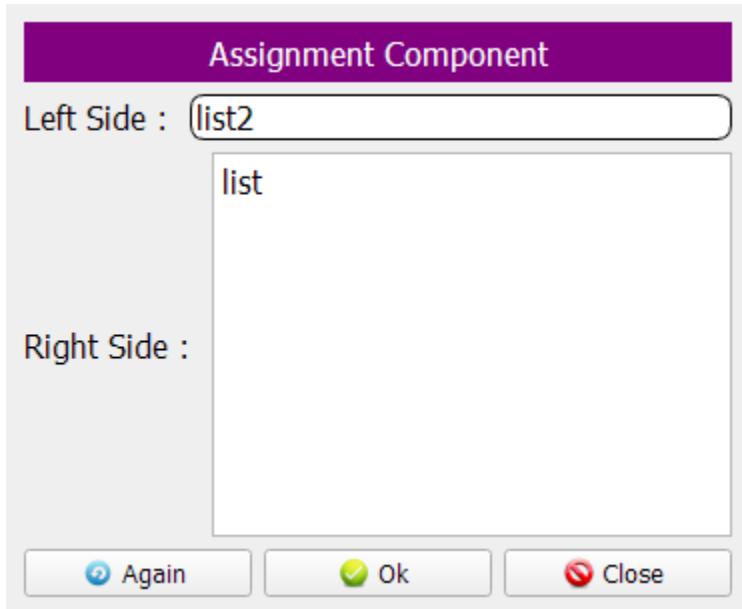
We will copy the List to another variable called (List2)



Enter the data to the Interaction Page

Left side: list2

Right side: list



The Steps Tree will be updated



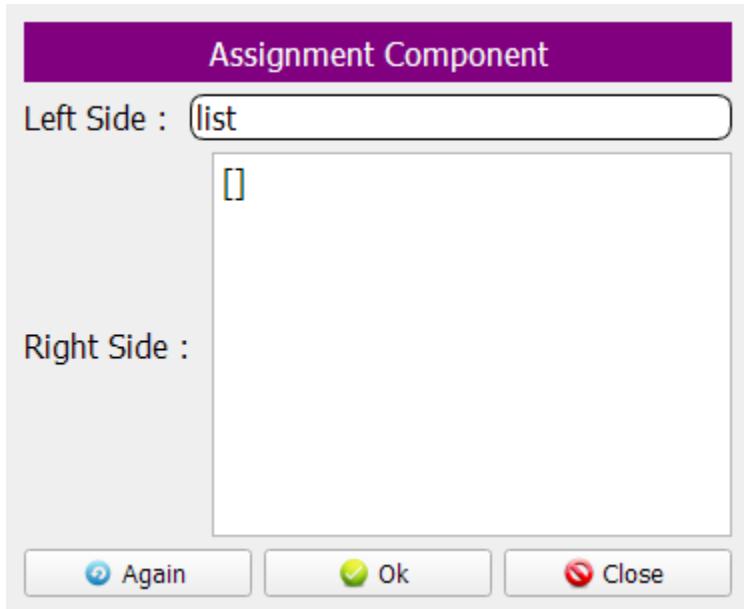
Now we will set the First List to an Empty List (All items will be deleted)



Enter the data in the Interaction Page

Left side: list

Right side: []



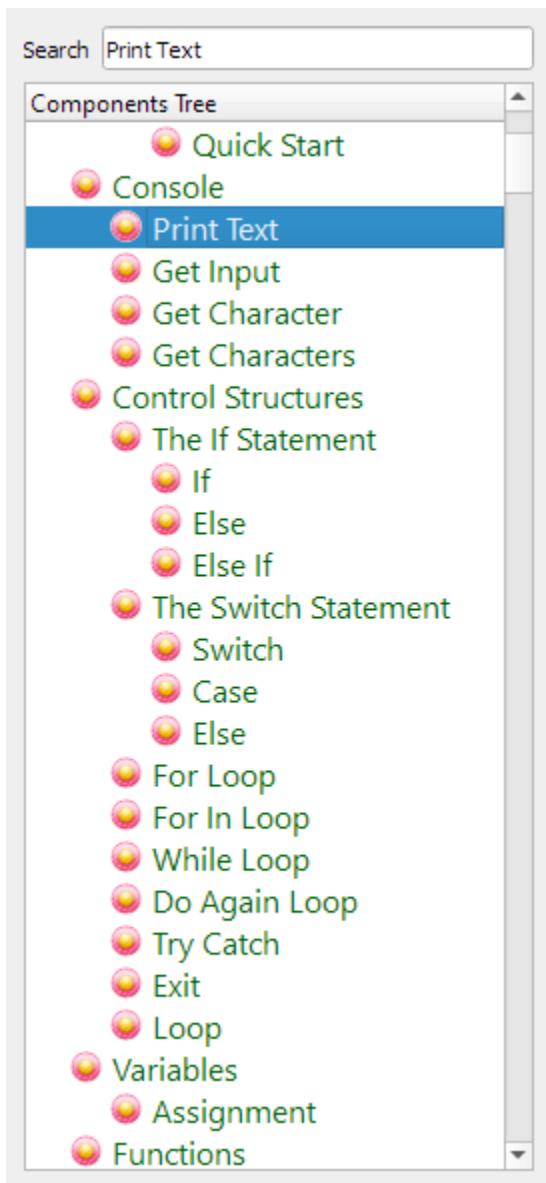
The Steps Tree will be updated

A screenshot of a "Steps Tree" window. It shows a tree structure with a single node expanded. The node is labeled "- Start Point" and contains the following Python-like code:

```
list = [1,2,3,"four","five"]
list2 = list
list = []
```

The first line "list = [1,2,3,"four","five"]" is highlighted with a green background.

We will print the First List (list)



Print Component

Text : `list`

Type :

New Line

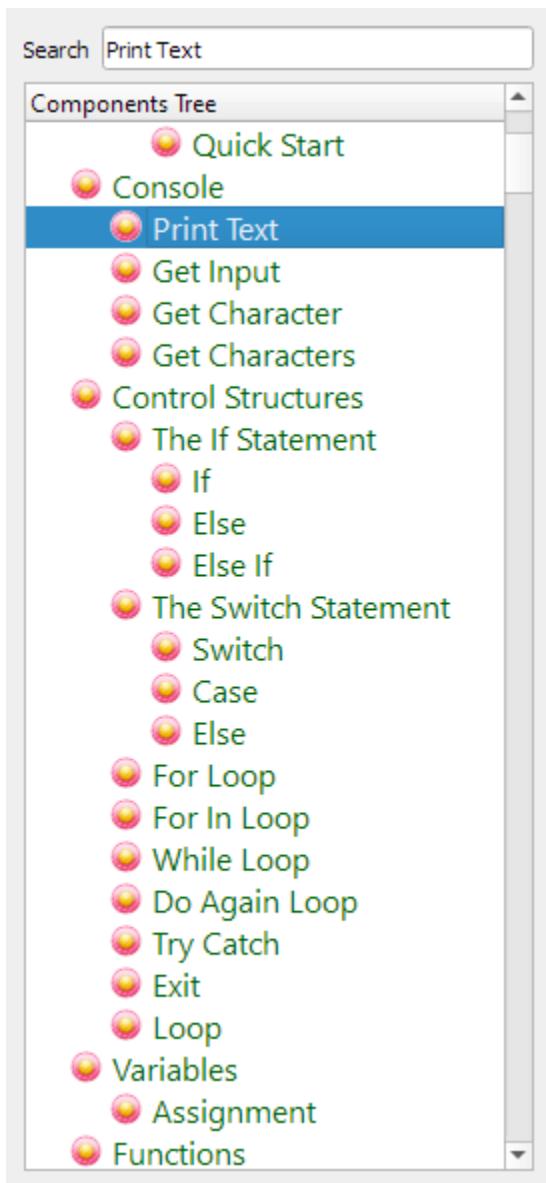
 Again  Ok  Close

Steps Tree

- Start Point

```
list = [1,2,3,"four","five"]
list2 = list
list = []
Print list (New Line)
```

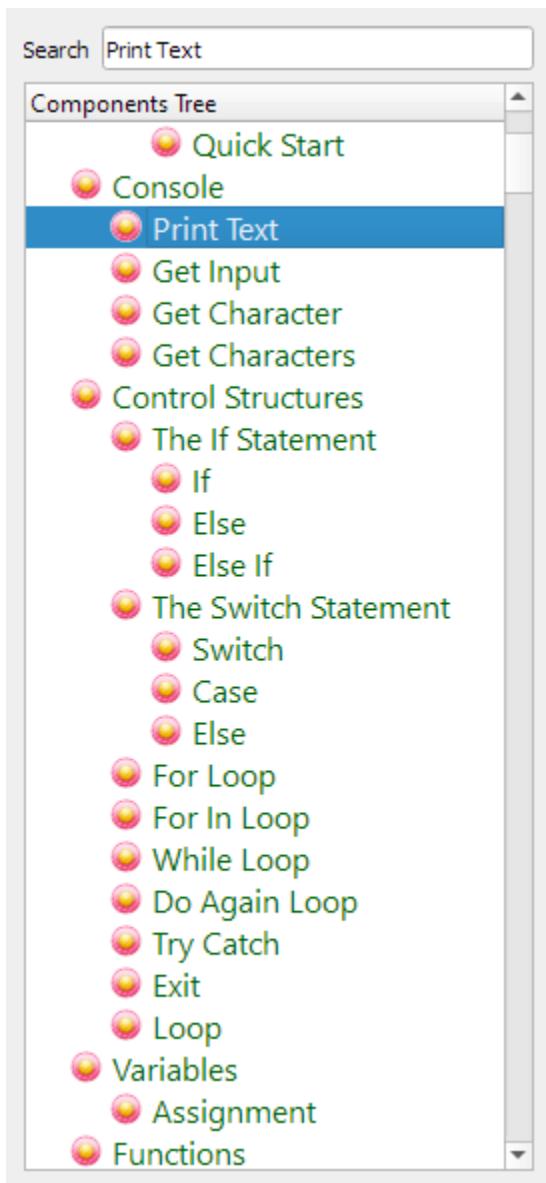
We will print the Second List too (list2)



The screenshot shows two windows from the PWCT interface. The top window is titled "Print Component". It has a text input field containing "*****" and a dropdown menu showing "Literal" and "Expression". A checkbox labeled "New Line" is checked. The bottom window is titled "Steps Tree" and shows a code editor with the following Python-like pseudocode:

```
list = [1,2,3,"four","five"]
list2 = list
list = []
Print list (New Line)
Print ***** (New Line)
```

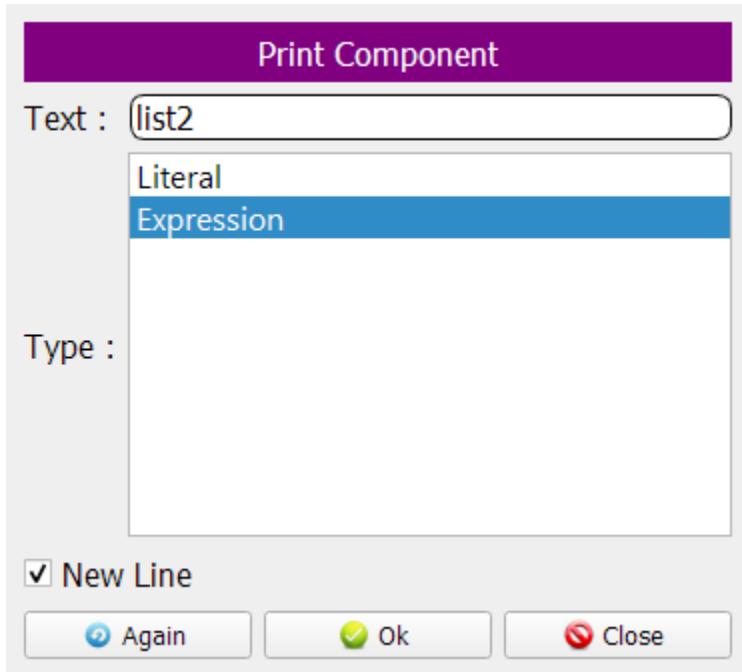
Select the (Print Text) component



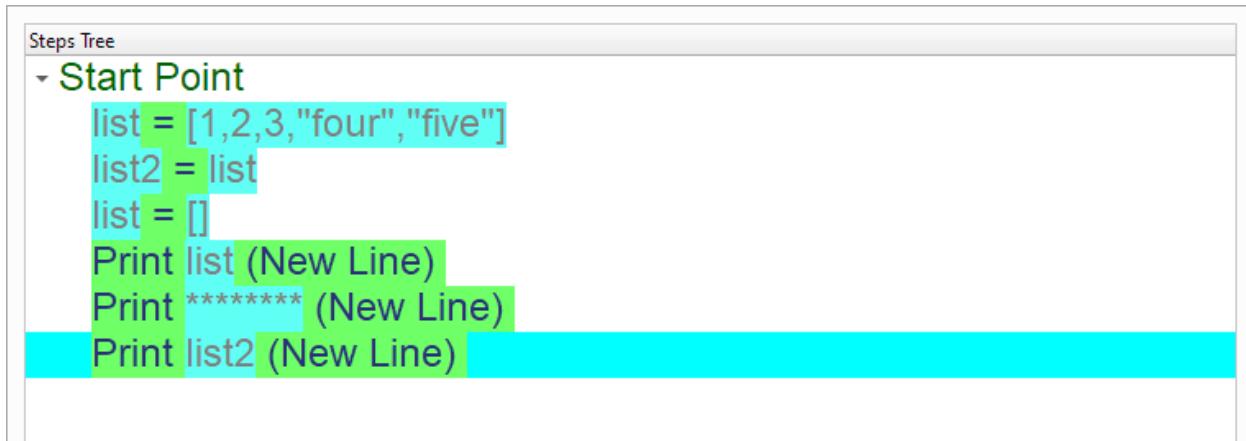
Enter the data in the Interaction Page

Text: List2

Type: Expression



The Final Steps Tree



CHAPTER TEN

IMPLICIT CONVERSION

In this chapter we are going to learn about the Implicit Conversion

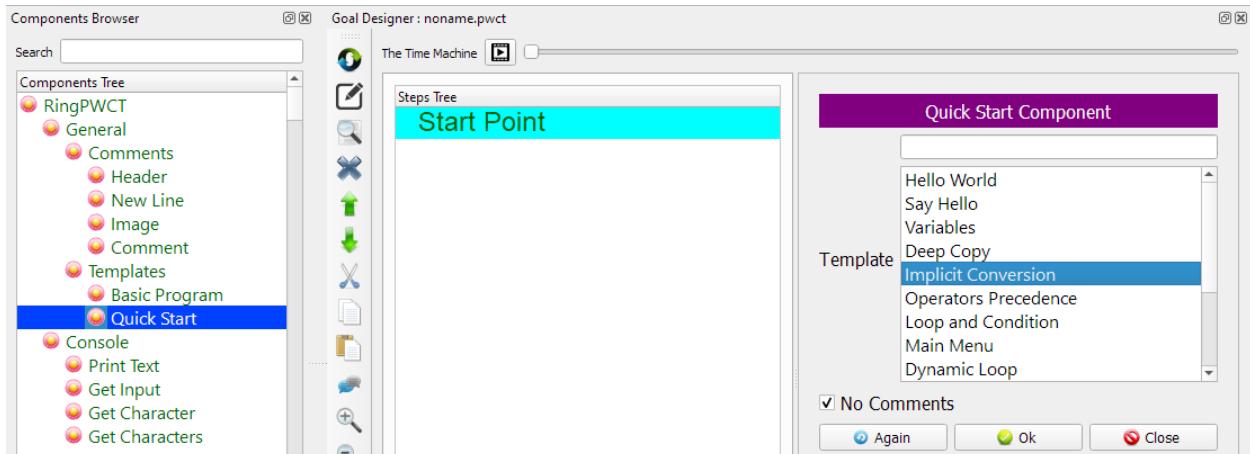
10.1 Introduction

Rules:

```
String + Number ---> String      # Merge Strings (Number will be converted to a String)
Number + String ---> Number       # Sum Numbers (String will be converted to a Number)
```

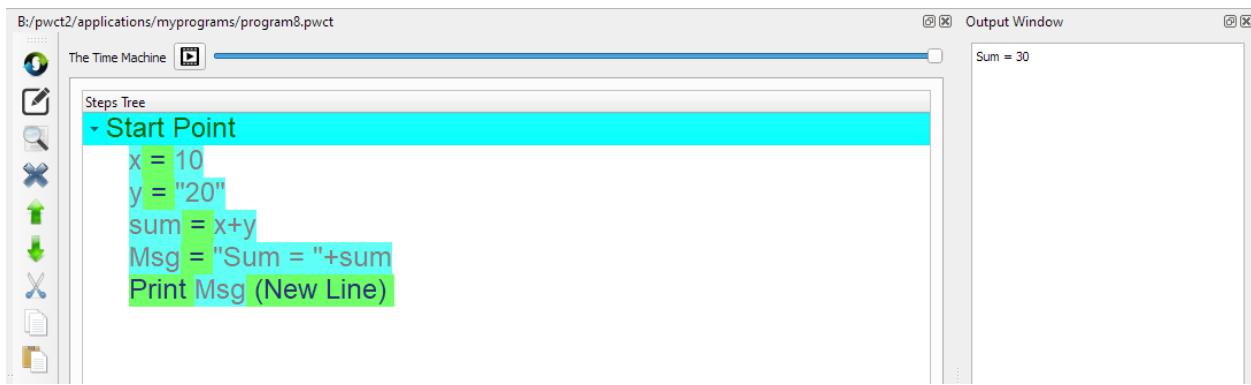
We will create a simple program to learn how to use the Implicit Conversion

We can create this program quickly using the Quick Start component



10.2 Program Steps

After selecting the (Implicit Conversion) template, we will get the next steps in the Goal Designer

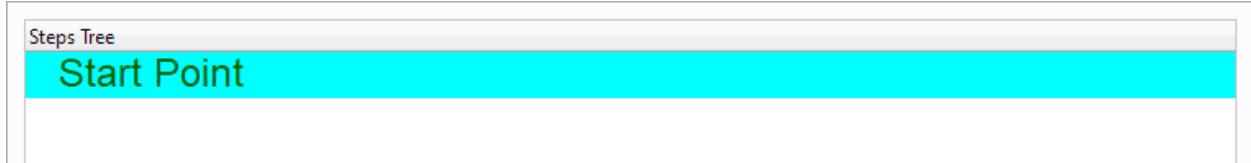


10.3 Creating the Program

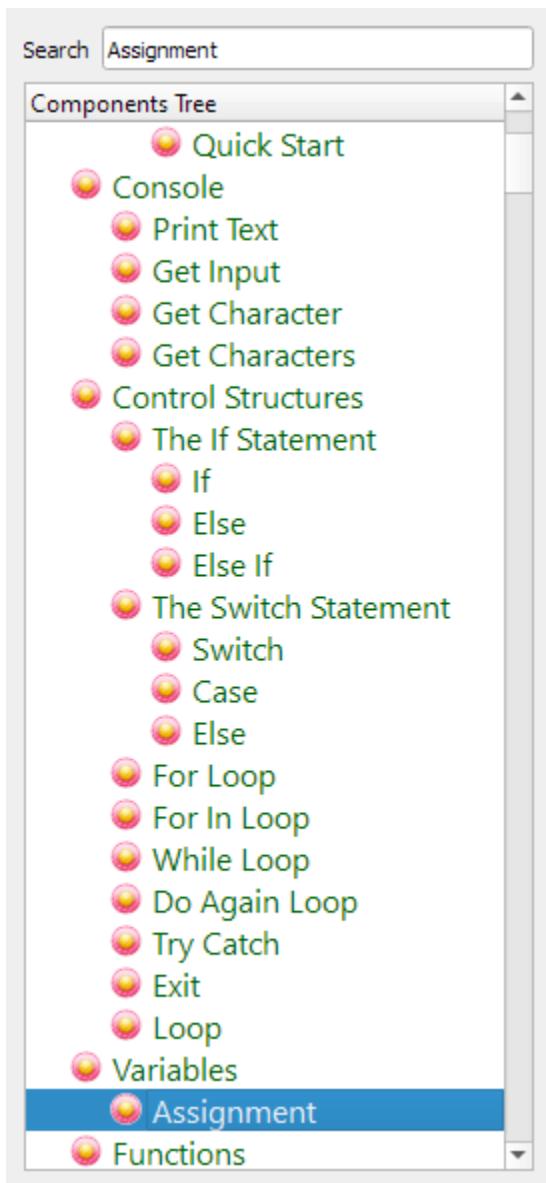
To create this program we will use the next components

- Assignment
- Print Text

In the begining the Steps Tree is empty



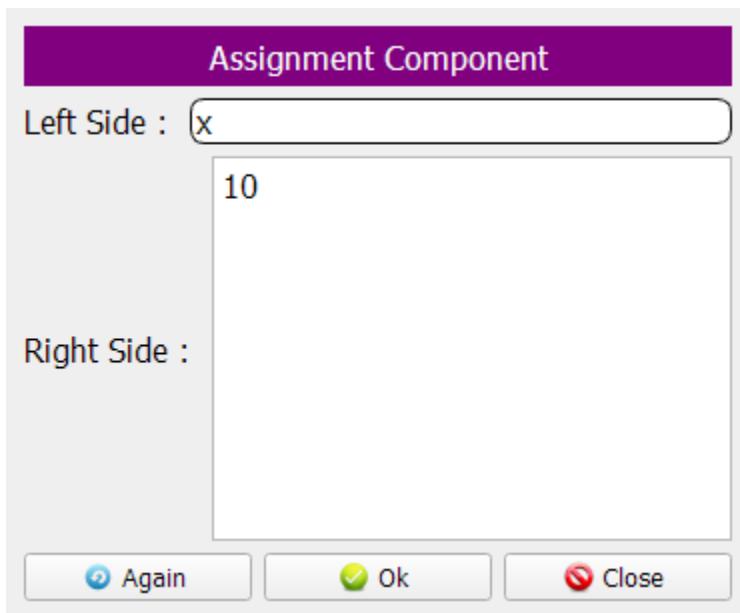
Select the (Assignment) component



Enter the data in the Interaction Page

Left side: x

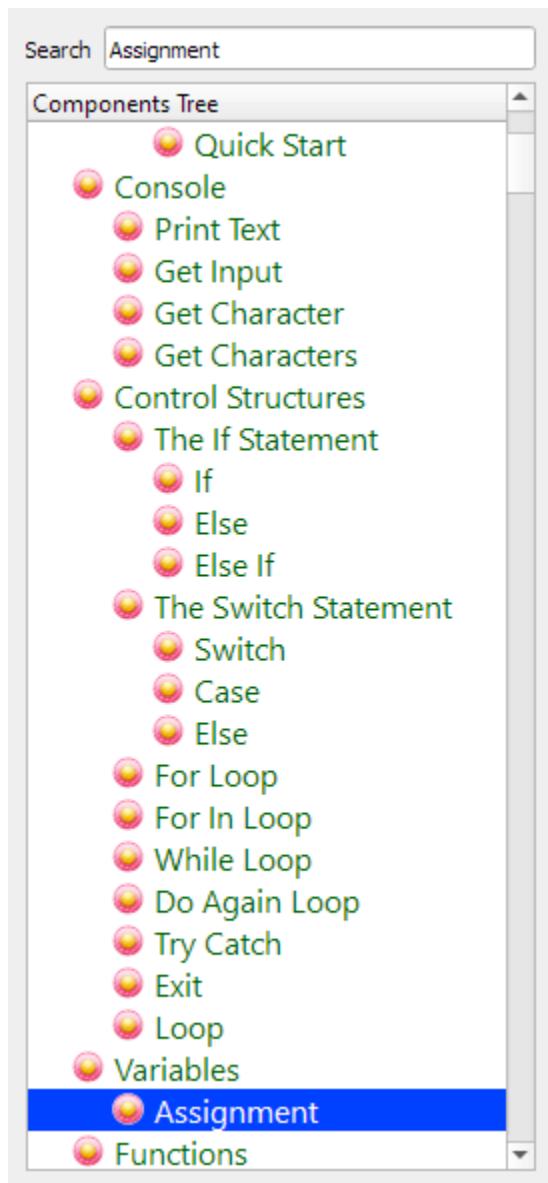
Right side: 10

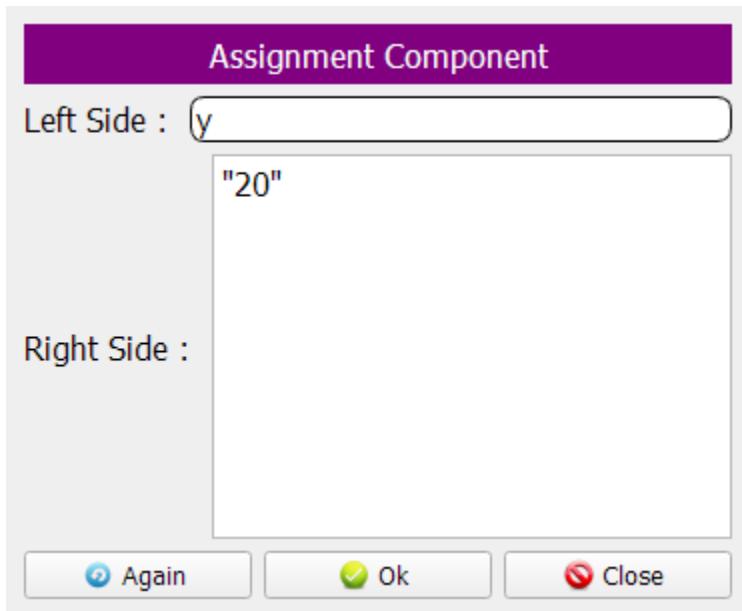


The Steps Tree will be updated



Set y = "20"





The Steps Tree will be updated

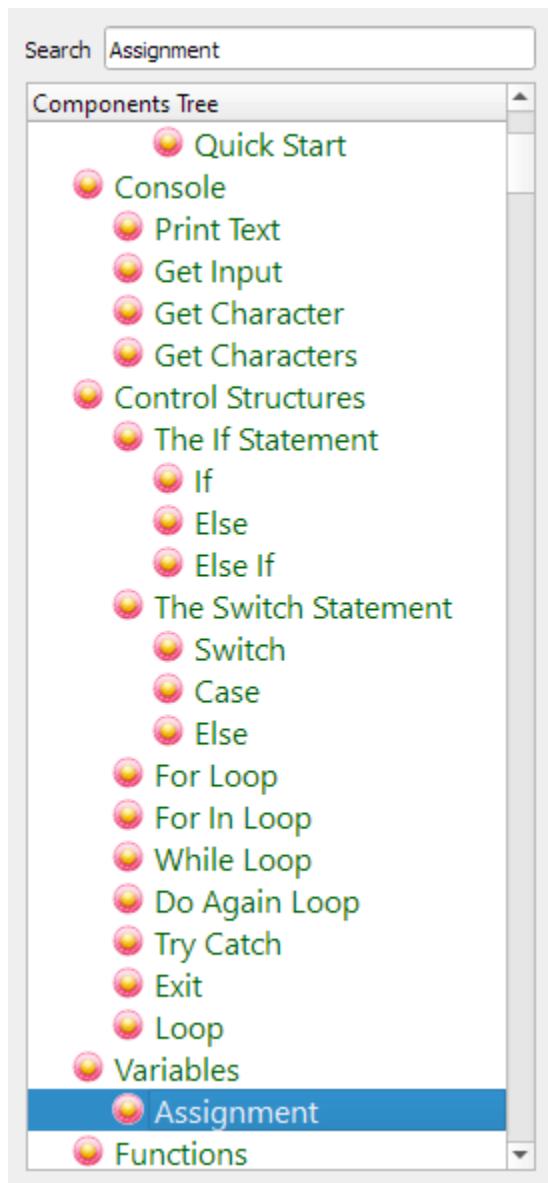


Set sum = x + y

X is a Number

Y is a String

X + Y → Number + String → Number



Assignment Component

Left Side : sum

x+y

Right Side :

Again Ok Close

Steps Tree

- Start Point

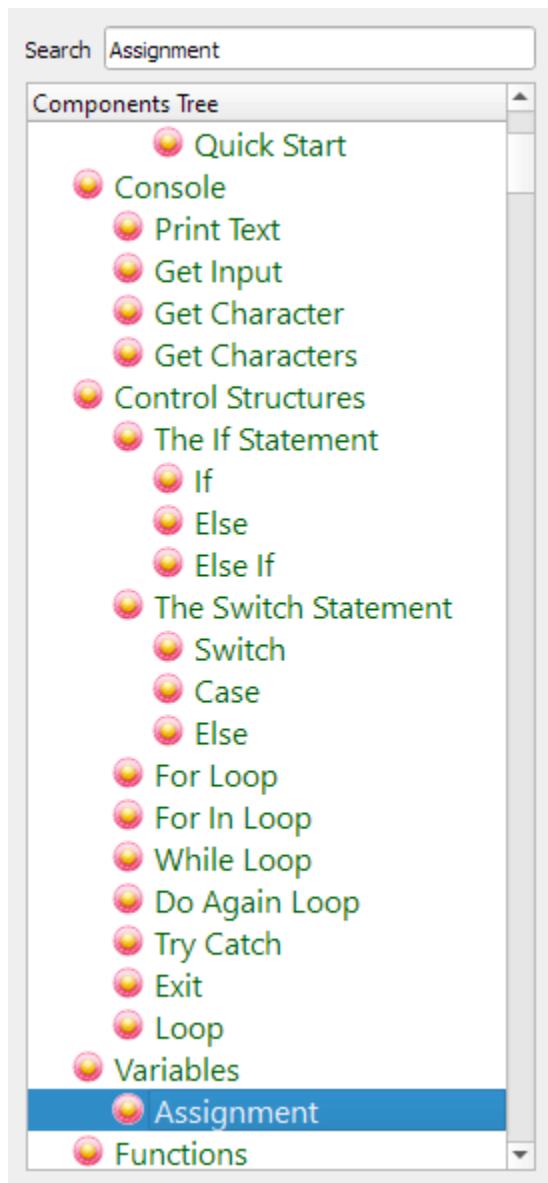
x = 10
y = "20"
sum = x+y

Set msg = "Sum = " + sum

"Sum = " is a String

sum is a Number

"Sum = " + sum —> String + Number —> String



Assignment Component

Left Side :

"Sum = "+sum

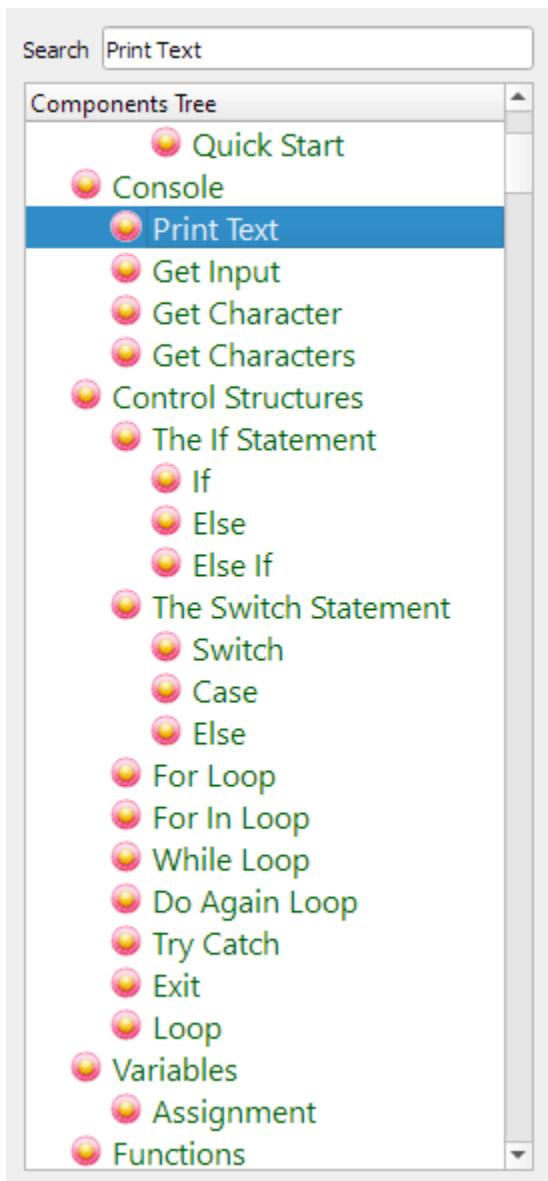
Right Side :

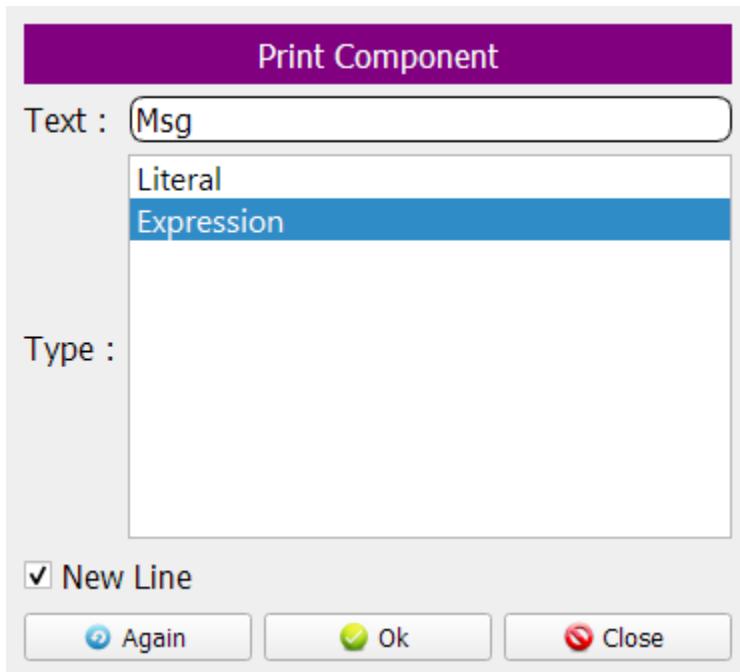
Steps Tree

- Start Point

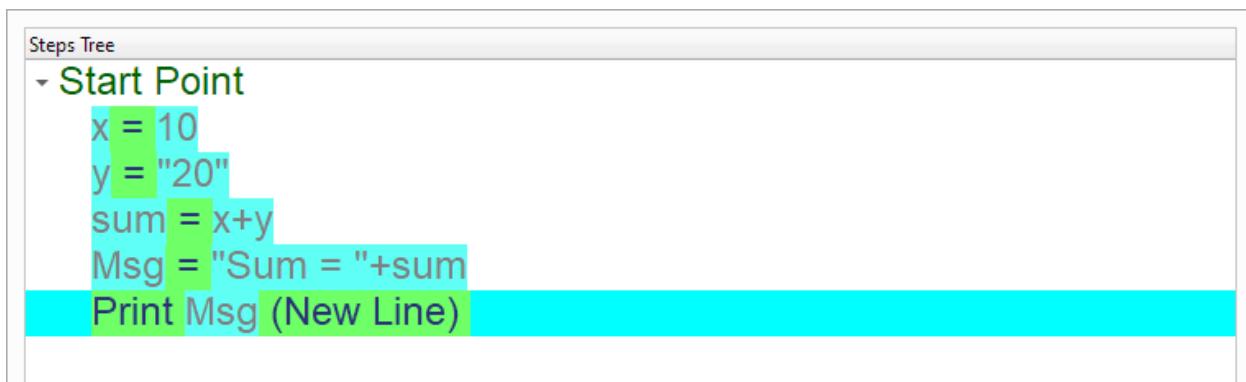
x = 10
y = "20"
sum = x+y
Msg = "Sum = "+sum

Print the Msg variable





Now we have the final Steps Tree in our program



CHAPTER
ELEVEN

OPERATORS

In this chapter we will introduce the different operators provided by the language.

11.1 Arithmetic Operators

The next table presents all of the arithmetic operators provided by the language. Assume variable X=50 and variable Y=10 then:

Operator	Description	Example	Result
+	Add	x+y	60
-	Subtract	x-y	40
*	Multiplies	x*y	500
/	Divide	x/y	5
%	Modulus	x%y	0
++	Increment	x++	51
--	Decrement	x--	49

11.2 Relational Operators

The next table presents all of the relational operators provided by the language. Assume variable X=50 and variable Y=10 then:

Operator	Description	Example	Result
=	Equal	x = y	False
!=	Not Equal	x != y	True
>	Greater than	x > y	True
<	Less than	x < y	False
>=	Greater or Equal	x >= y	True
<=	Less than or Equal	x <= y	False

11.3 Logical Operators

The next table presents all of the logical operators provided by the language. Assume variable X=True and variable Y=False then:

Operator	Description	Example	Result
and	Logical AND	x and y	False
or	Logical OR	x or y	True
not	Logical Not	not x	False

Another style

Operator	Description	Example	Result
&&	Logical AND	x && y	False
	Logical OR	x y	True
!	Logical Not	! x	False

11.4 Bitwise Operators

The next table presents all of the bitwise operators provided by the language. Assume variable X=8 and variable Y=2 then:

Operator	Description	Example	Result
&	Binary AND	x & y	0
	Binary OR	x y	10
^	Binary XOR	x ^ y	10
~	Binary Ones Complement	~x	-9
<<	Binary Left Shift	x << y	32
>>	Binary Right Shift	x >> y	2

11.5 Assignment Operators

The next table presents all of the assignment operators provided by the language.

Assume variable X=8 then:

Operator	Description	Example	Result
=	Assignment	x = 10	x=10
+=	Add AND assignment	x += 5	x=13
-=	Subtract AND assignment	x -= 3	x=5
*=	Multiply AND assignment	x *= 2	x=16
/=	Divide AND assignment	x /= 3	x=2.67
%=	Modulus AND assignment	x %= 2	x=0
<<=	Left shift AND assignment	x <<= 2	x=32
>>=	Right shift AND assignment	x >>= 2	x=2
&=	Bitwise AND assignment	x &= 4	x=0
=	Bitwise OR and assignment	x = 3	x=11
^=	Bitwise XOR and assignment	x ^= 4	x=12

11.6 Misc Operators

Operator	Description
:literal	using : before identifier mean literal
Start:End	create list contains items from start to end
[list items]	define list items
list[index]	access list item
obj.name	using the dot operator to access object members (attributes/methods).
obj {stmts}	execute statements with direct access to object attributes & methods
func(para,...)	call function using parameters separated by comma
? <expr>	Print expression then new line

11.7 Operators Precedence

The next table present operators from higher precedence (Evaluated first) to lower precedence.

Operator
. [] () {}
~ :Literal [list items]
++ --
- (Unary negative) + (Unary positive)
Start:End
* / %
+ -
<< >>
&
^
< > <= >=
= !=
not !
and &&
or
Assignment = += -= *= /= %=>>= <<= &= ^= =
?

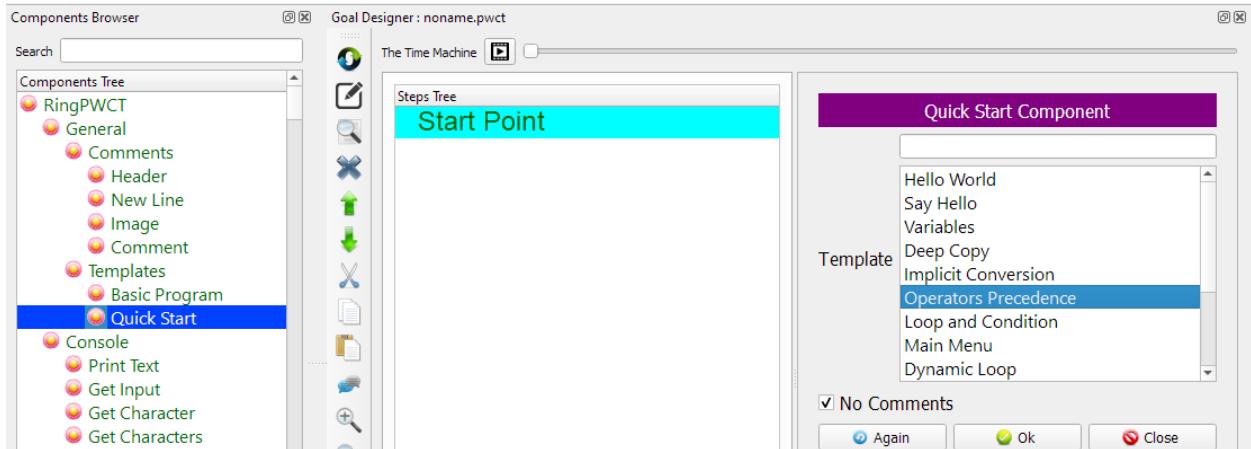
CHAPTER TWELVE

OPERATORS PRECEDENCE

In this chapter we are going to learn about the Operators Precedence

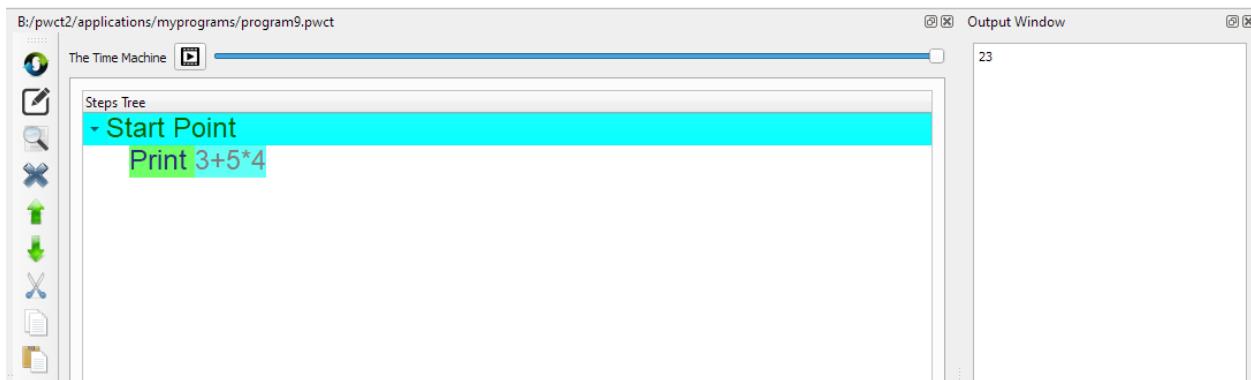
12.1 Introduction

We can create this program quickly using the Quick Start component



12.2 Program Steps

After selecting the (Operators Precedence) template, we will get the next steps in the Goal Designer

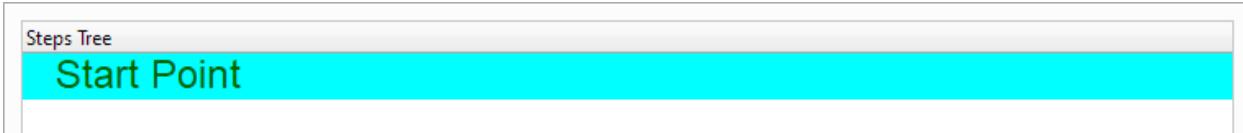


12.3 Creating the Program

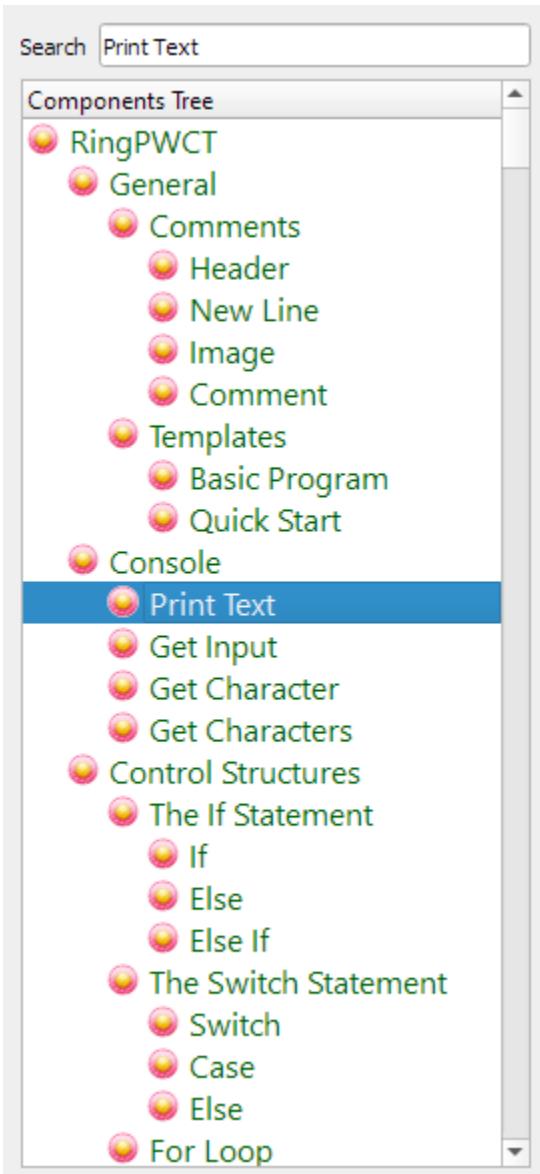
To create this program we will use the next components

- Print Text

In the begining the Steps Tree is empty



Select the (Print Text) component



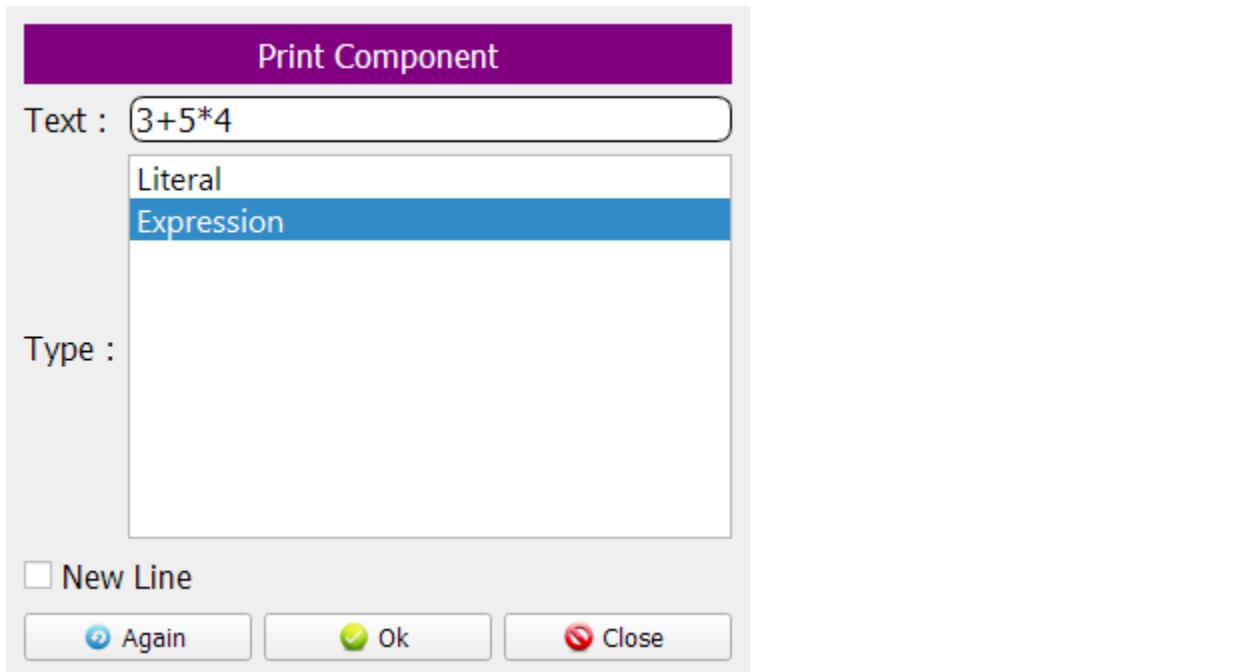
Enter the data in the Interaction Page

Text: $3+5*4$

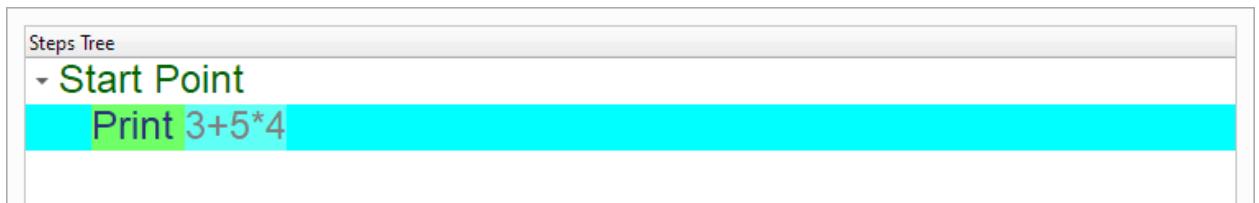
Type: Expression

Order of execution

```
5*4 --> 20  
3+20 --> 23
```



Now we have the final Steps Tree in our program



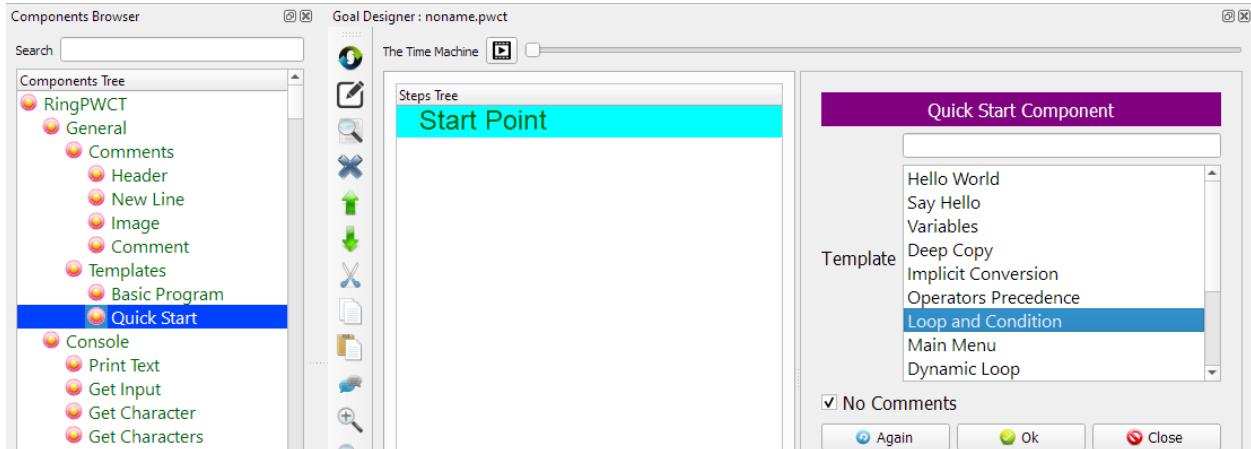
CHAPTER THIRTEEN

LOOP AND CONDITION

In this chapter we are going to learn about the Loop and Condition

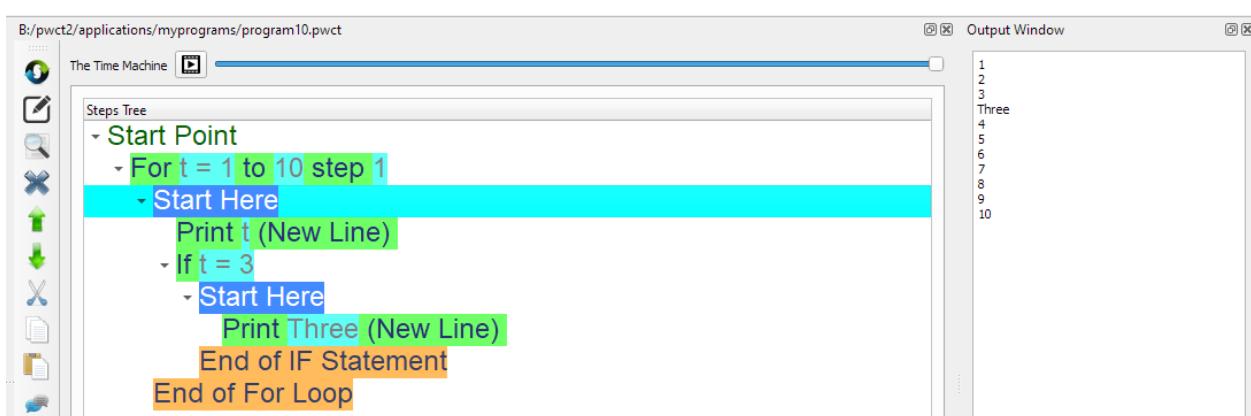
13.1 Introduction

We can create this program quickly using the Quick Start component



13.2 Program Steps

After selecting the (Loop and Condition) template, we will get the next steps in the Goal Designer



13.3 Creating the Program

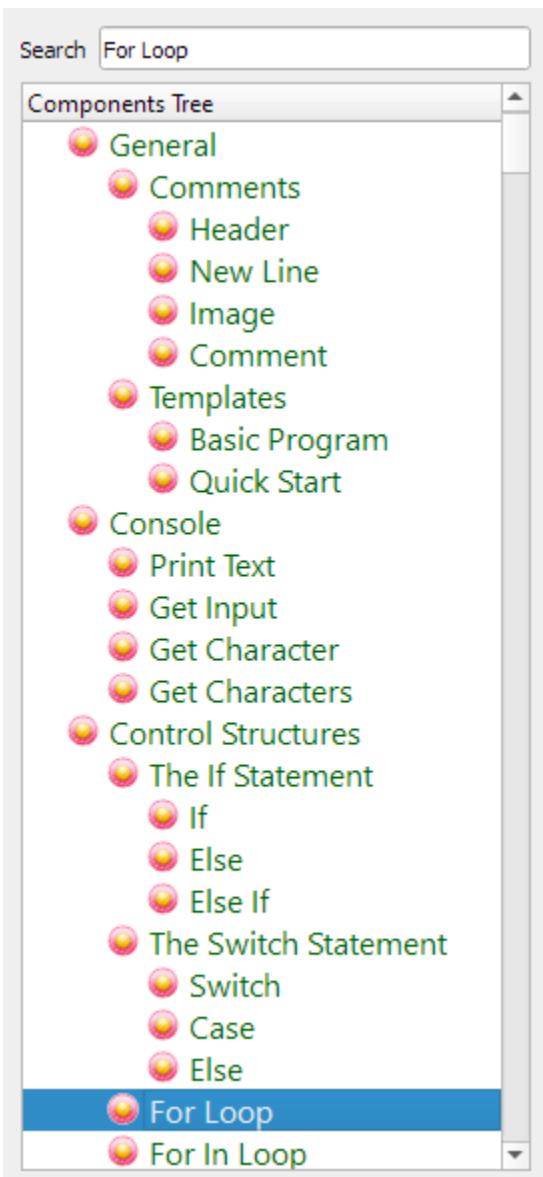
To create this program we will use the next components

- For Loop
- If Statement
- Print Text

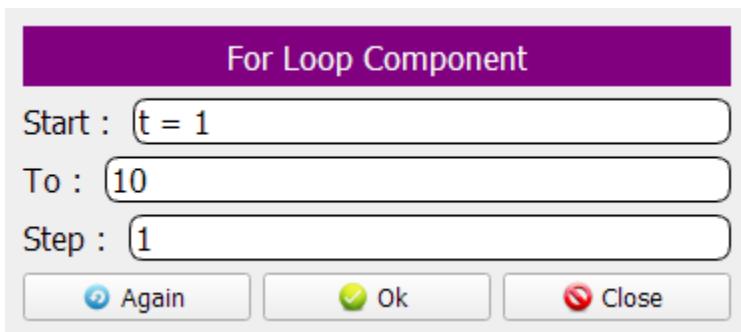
In the begining the Steps Tree is empty



Select the (For Loop) component



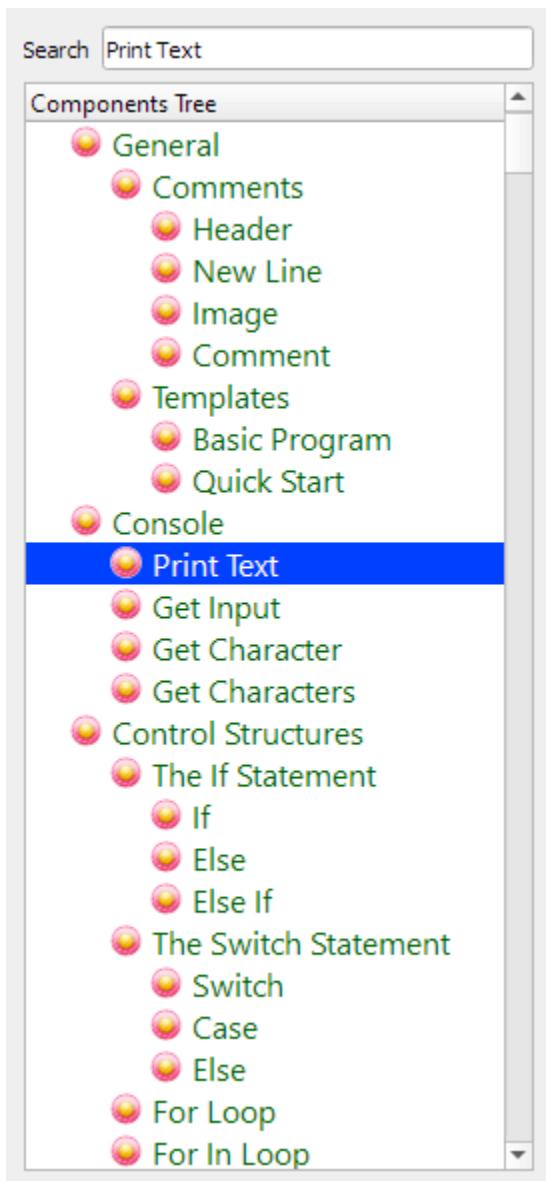
Enter the data in the Interaction Page



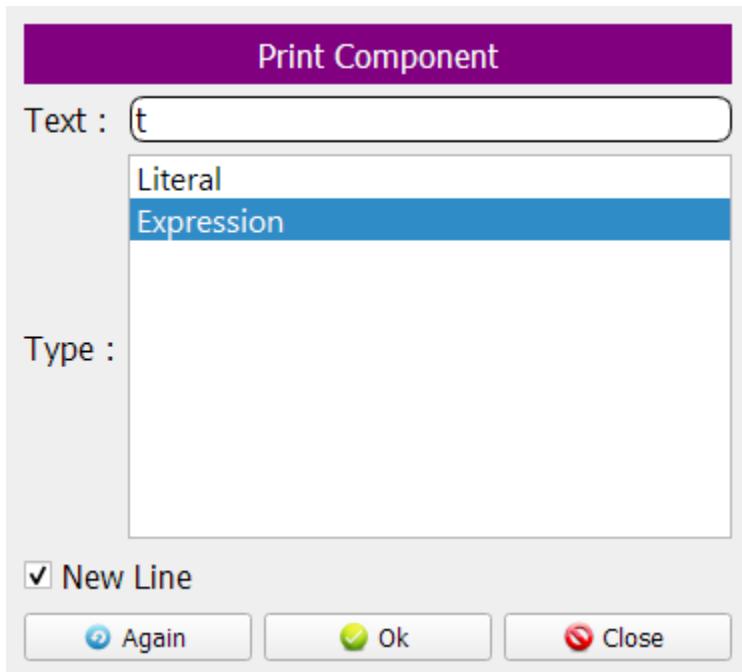
The Steps Tree will be updated



Select the (Print Text) component



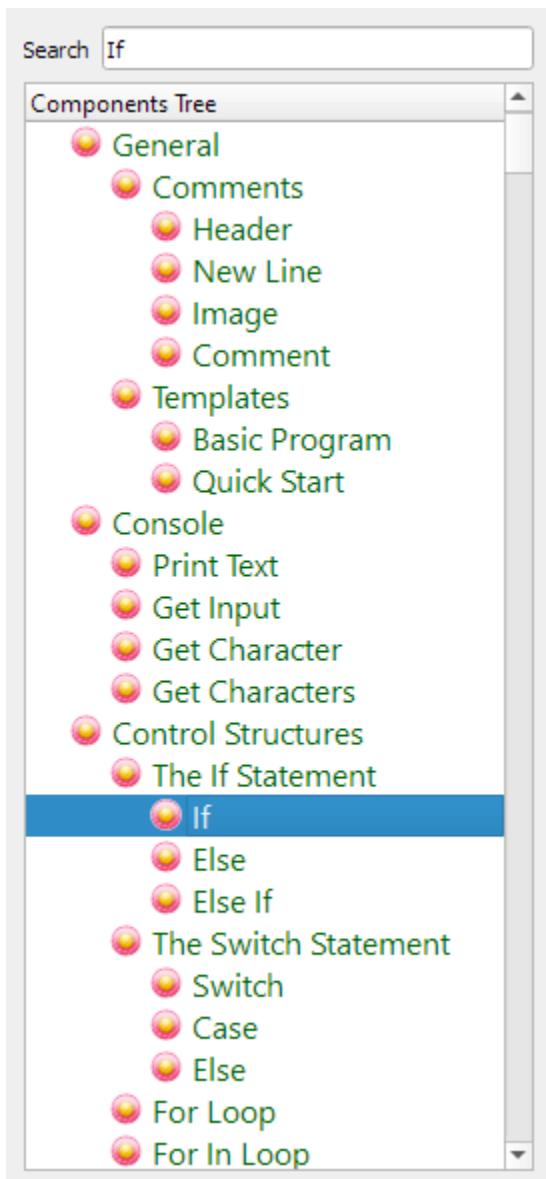
Enter the data in the Interaction Page



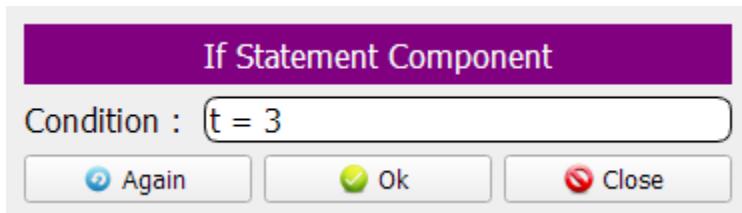
The Steps Tree will be updated



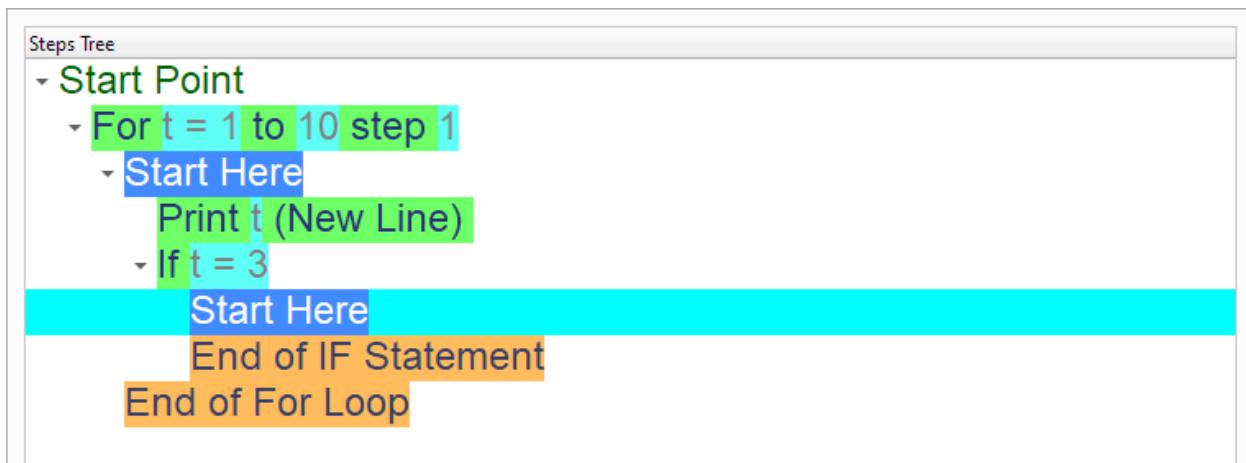
Select the (If Statement) component



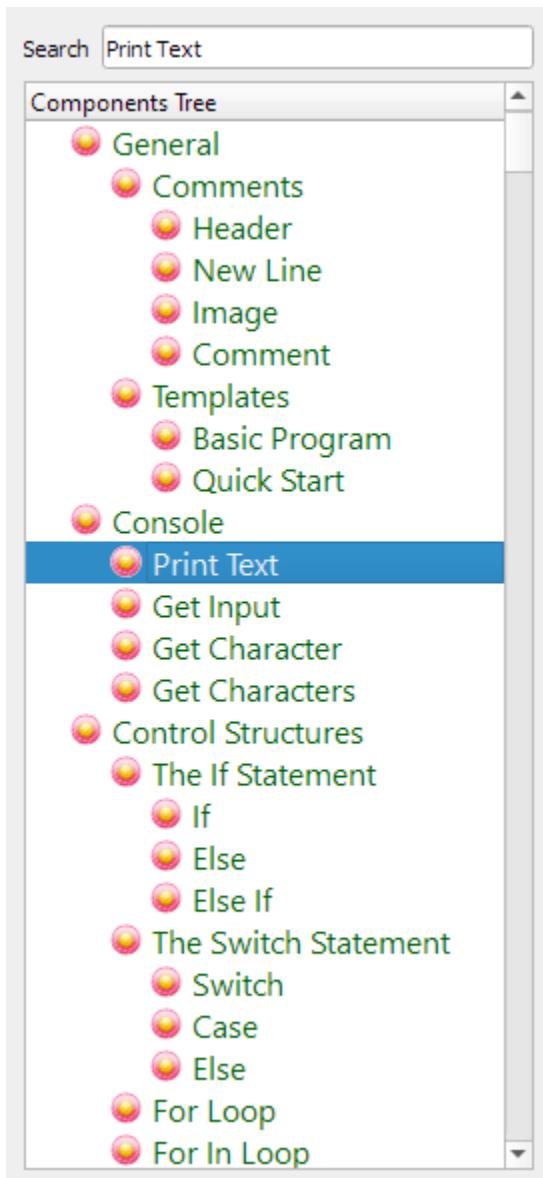
Enter the data in the Interaction Page



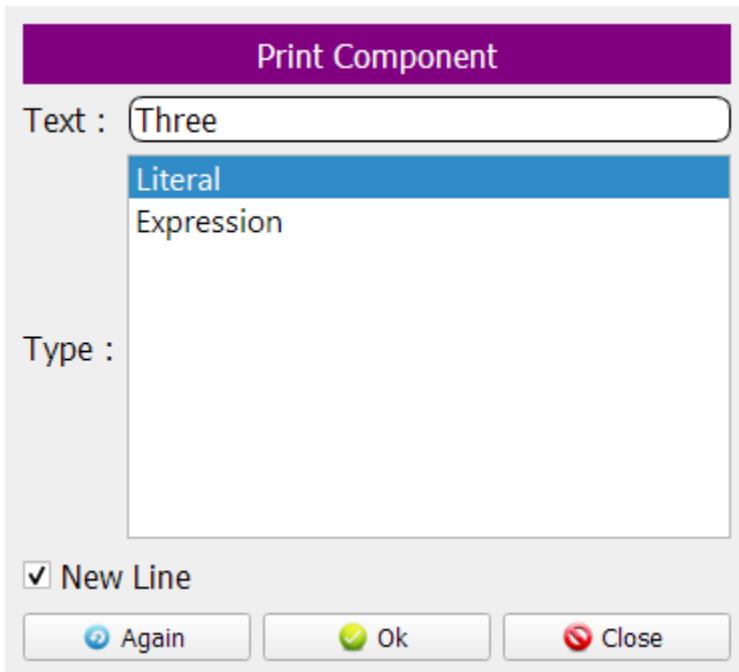
The Steps Tree will be updated



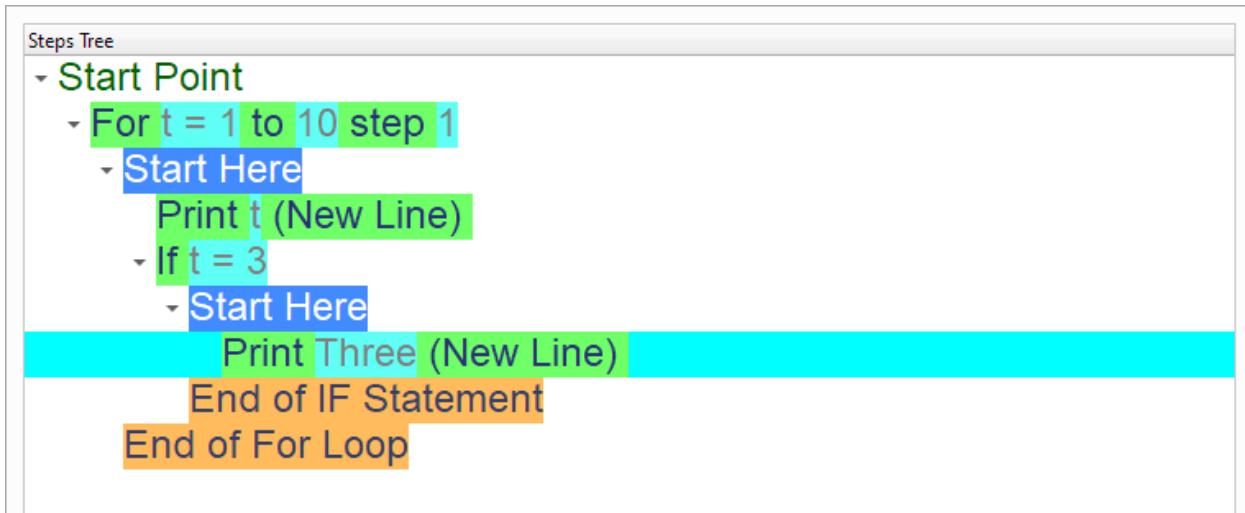
Select the (Print Text) component



Enter the data in the Interaction Page



Now we have the final Steps Tree in our program



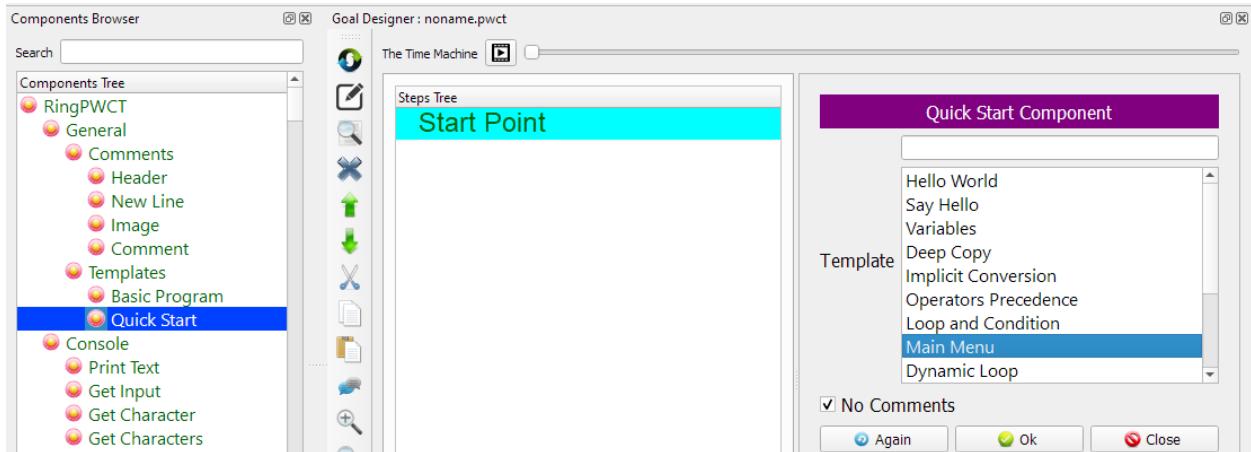
CHAPTER FOURTEEN

MAIN MENU

In this chapter we are going to learn about the Main Menu

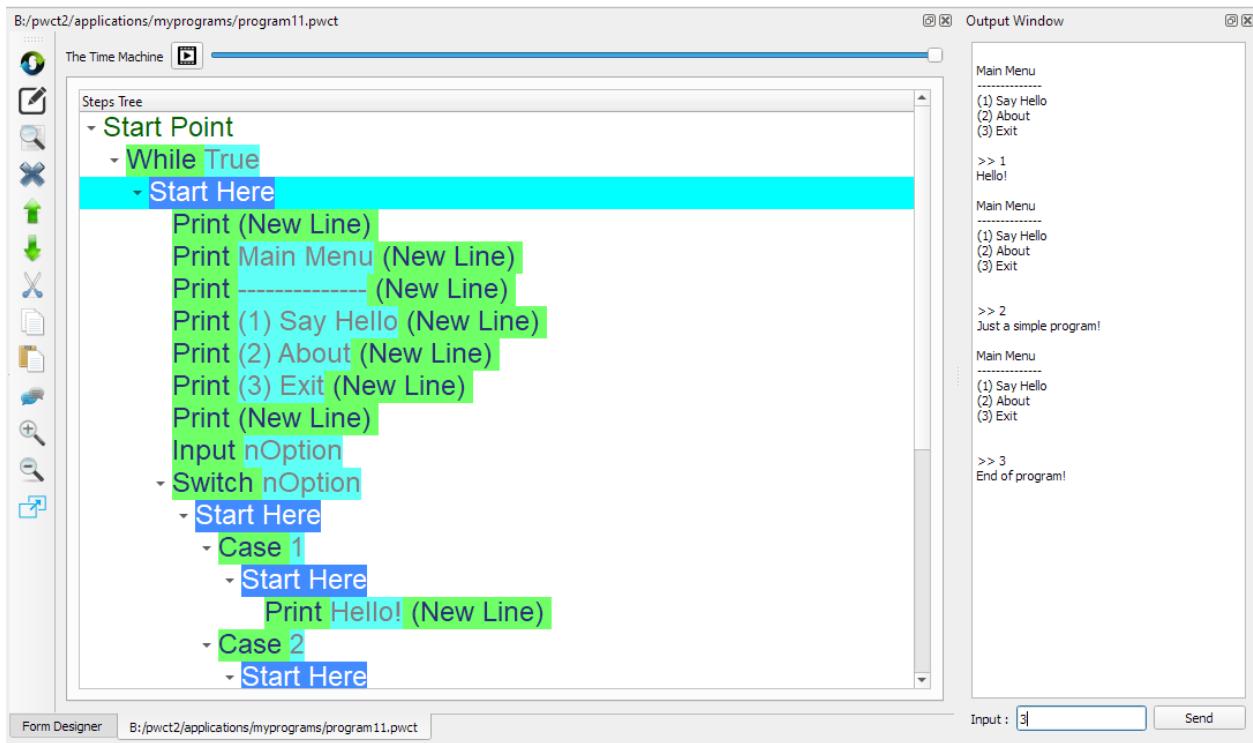
14.1 Introduction

We can create this program quickly using the Quick Start component



14.2 Program Steps

After selecting the (Main Menu) template, we will get the next steps in the Goal Designer



The Steps Tree:

```

While True
    Print  (New Line)
    Print Main Menu (New Line)
    Print ----- (New Line)
    Print (1) Say Hello (New Line)
    Print (2) About (New Line)
    Print (3) Exit (New Line)
    Print  (New Line)
    Input nOption
    Switch nOption
        Case 1
            Print Hello! (New Line)
        Case 2
            Print Just a simple program! (New Line)
        Case 3
            Print End of program! (New Line)
            Shutdown 0
        Else
            Print bad option... (New Line)
    End of Switch
End of While Loop
  
```

14.3 Creating the Program

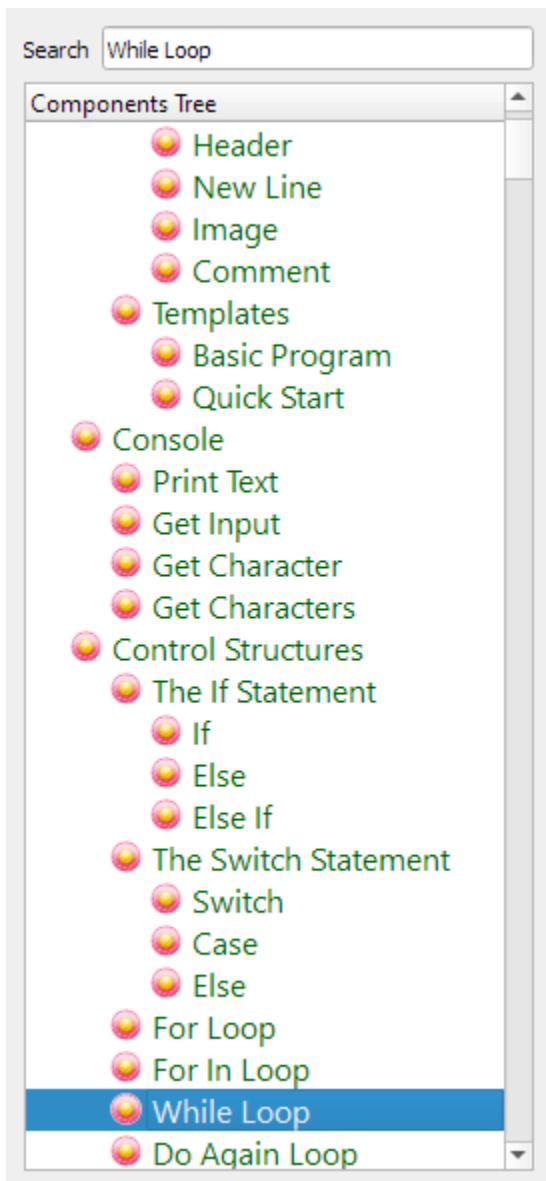
To create this program we will use the next components

- While
- Print Text
- Get Input
- Switch
- Case
- Shutdown
- Else

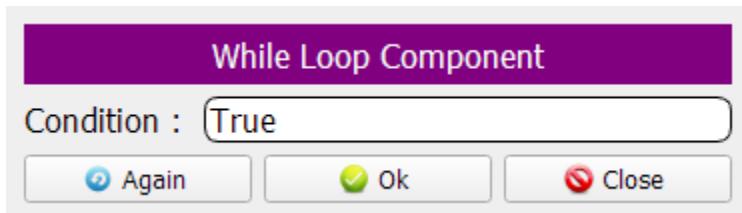
In the begining the Steps Tree is empty



Select the (While Loop) component



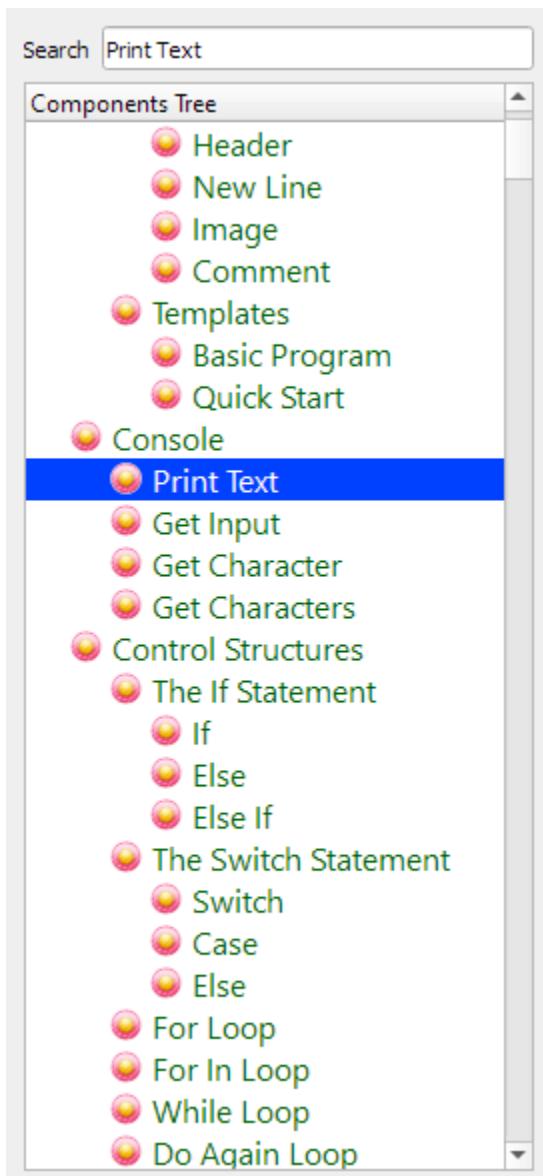
Enter the data in the Interaction Page



The Steps Tree will be updated

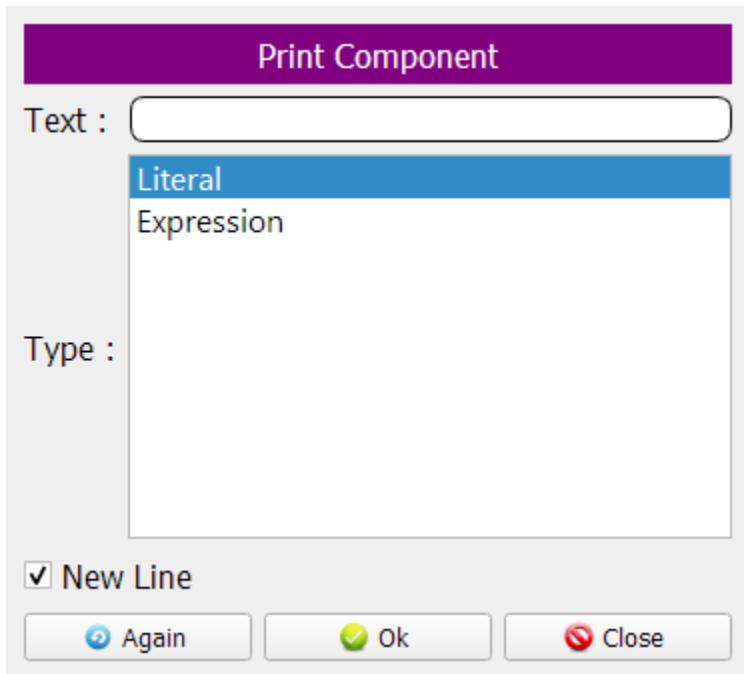


Select the (Print Text) component

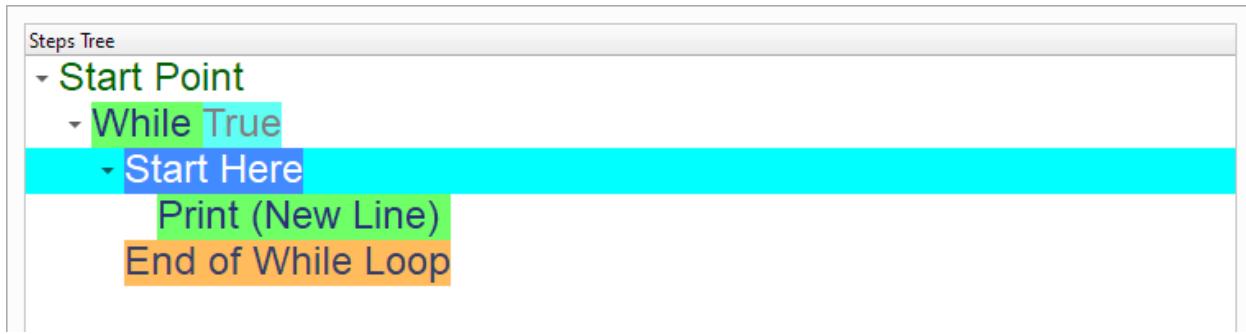


In the Interaction Page, Just click OK

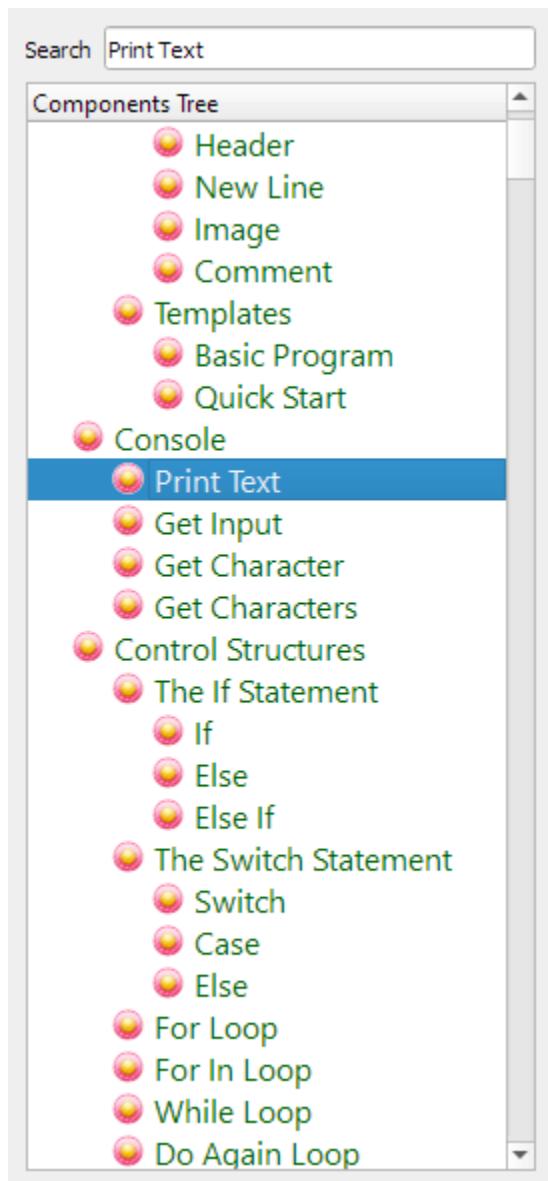
We will print an empty line!



The Steps Tree will be updated



We will print the Menu Items



Print Component

Text : Main Menu

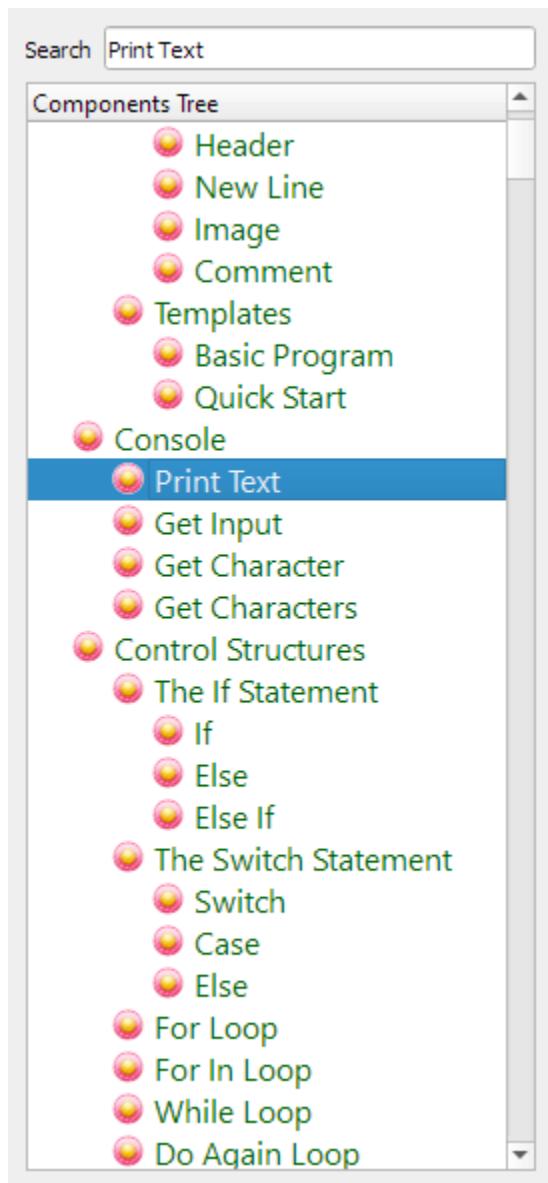
Type :

Literal
Expression

New Line

Steps Tree

- Start Point
 - While True
 - Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - End of While Loop



Print Component

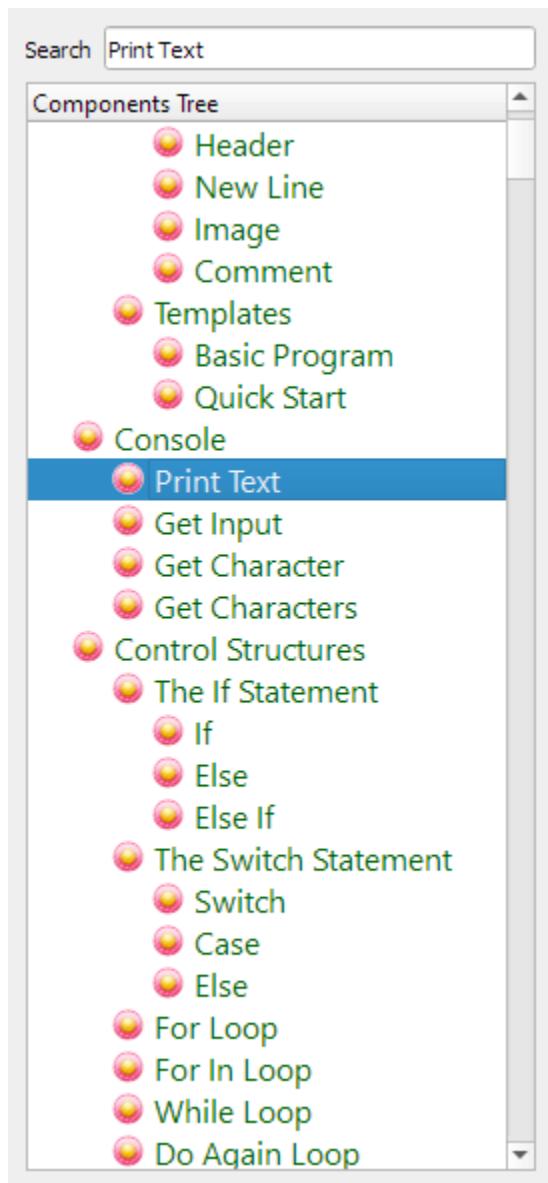
Text :

Type :

New Line

Steps Tree

- Start Point
 - While True
 - Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - Print ----- (New Line)
 - End of While Loop



Print Component

Text : (1) Say Hello

Literal
Expression

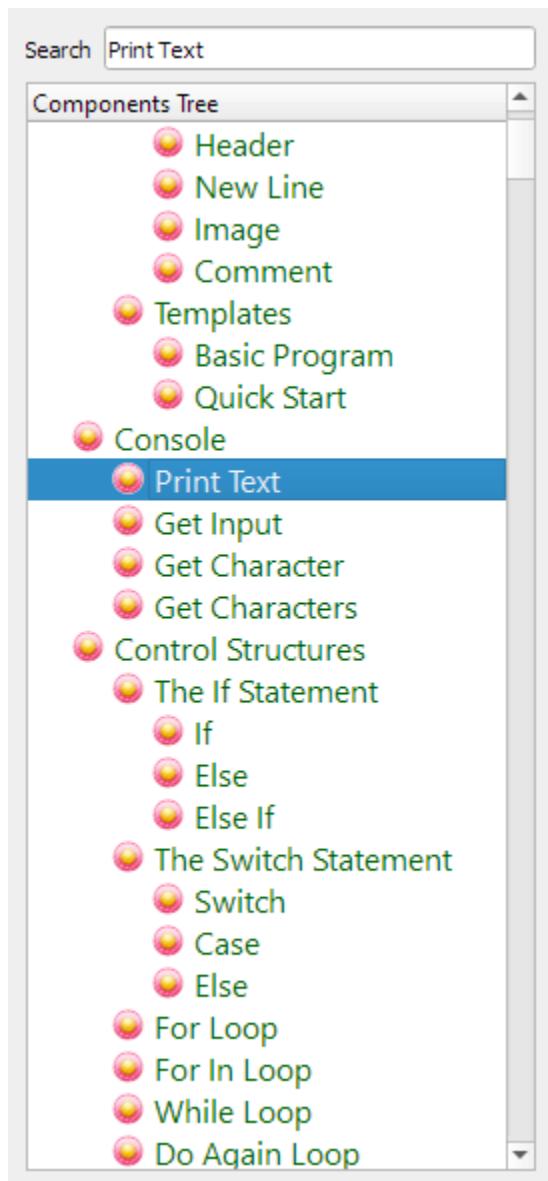
Type :

New Line

 Again  Ok  Close

Steps Tree

- Start Point
 - While True
 - Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - Print ----- (New Line)
 - Print (1) Say Hello (New Line)
 - End of While Loop



Print Component

Text : (2) About

Literal
Expression

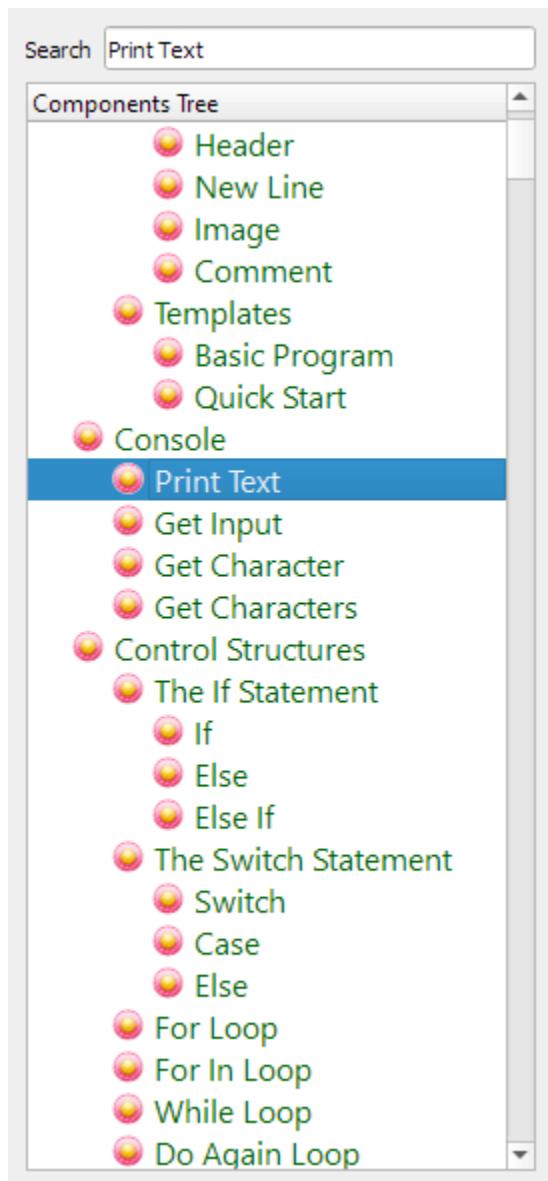
Type :

New Line

Again Ok Close

Steps Tree

- Start Point
 - While True
 - Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - Print ----- (New Line)
 - Print (1) Say Hello (New Line)
 - Print (2) About (New Line)
 - End of While Loop



Print Component

Text : (3) Exit

Literal
Expression

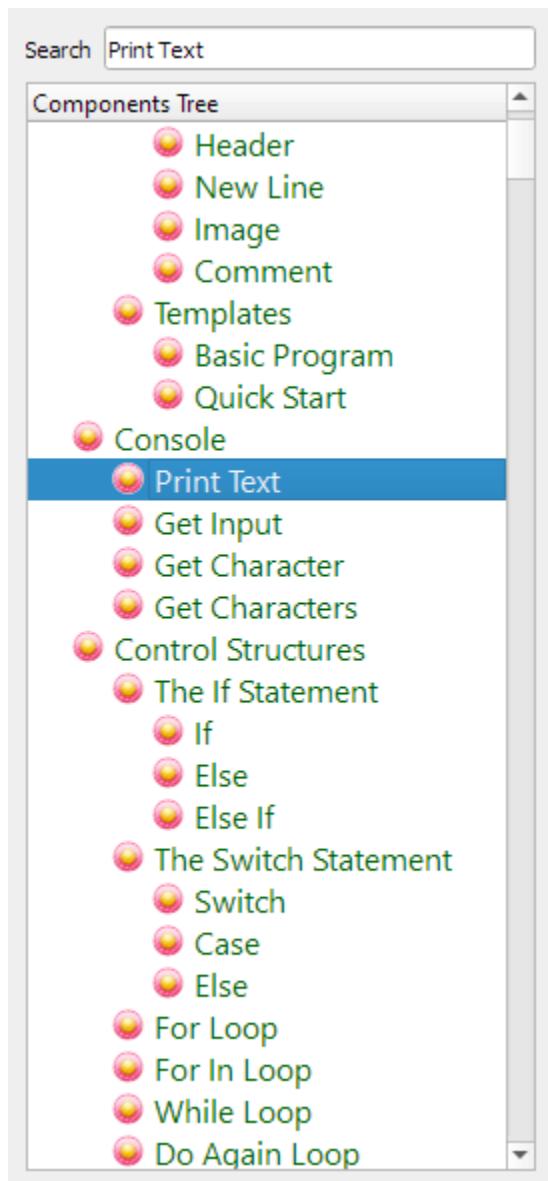
Type :

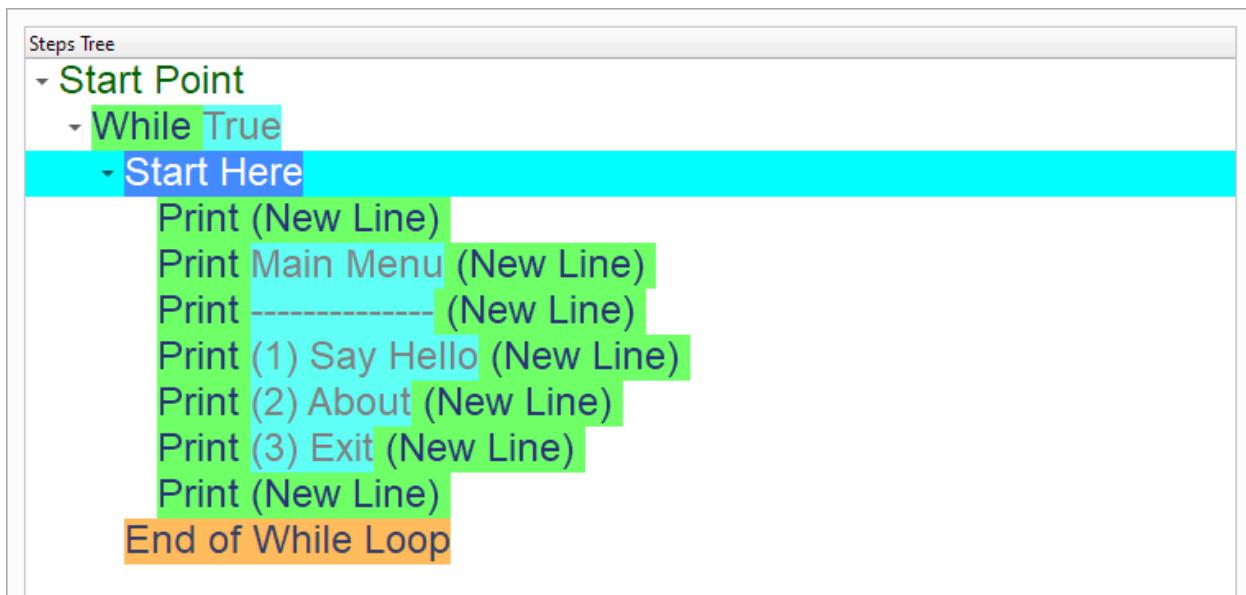
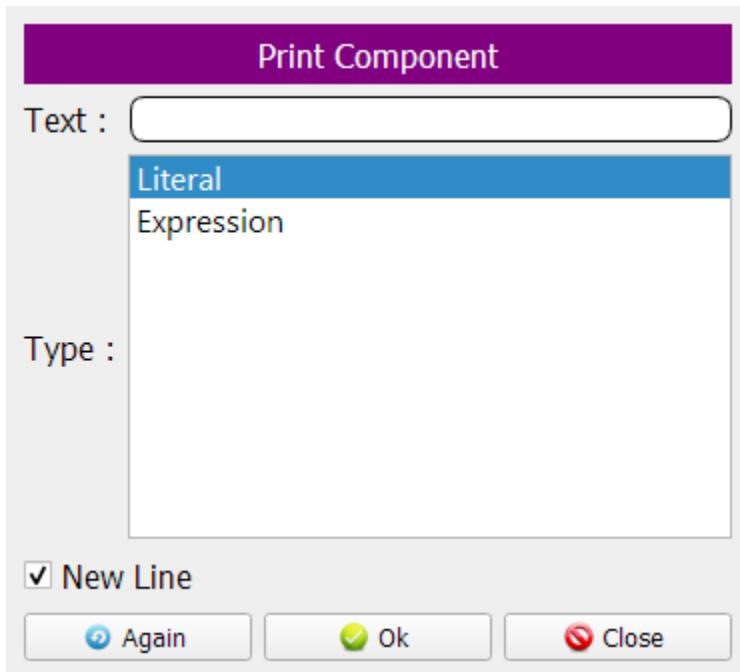
New Line

Again Ok Close

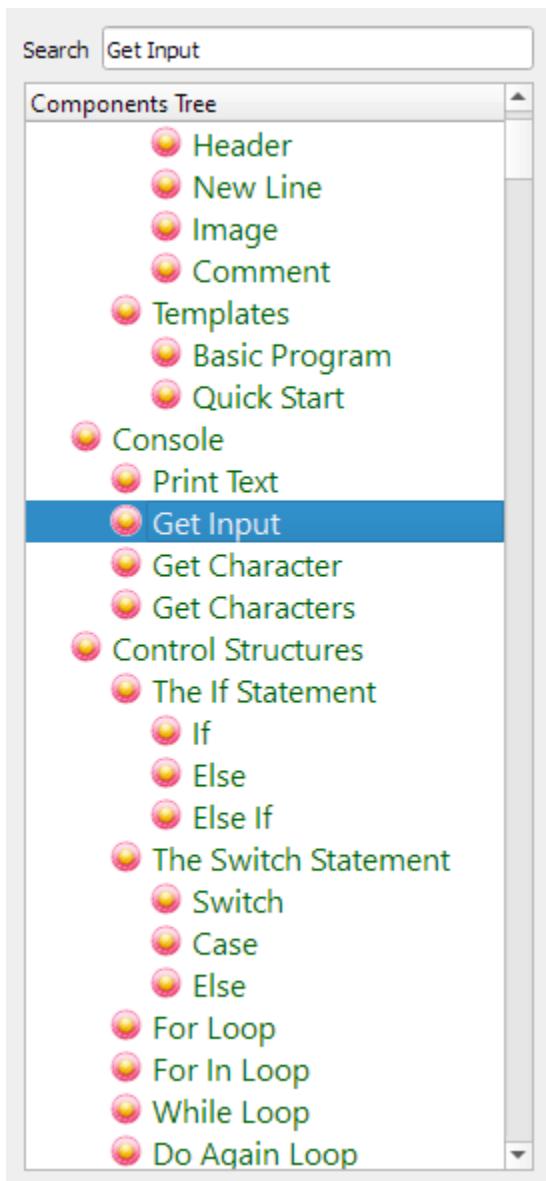
Steps Tree

- Start Point
 - While True
 - Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - Print ----- (New Line)
 - Print (1) Say Hello (New Line)
 - Print (2) About (New Line)
 - Print (3) Exit (New Line)
 - End of While Loop

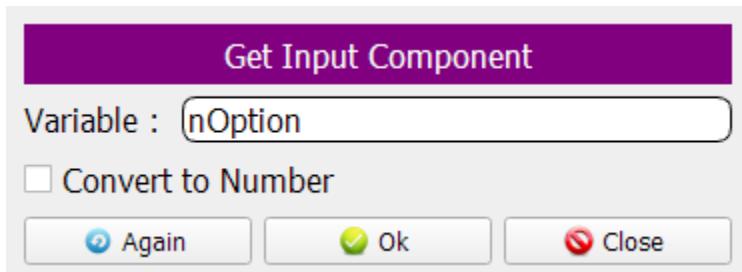




Select the (Get Input) component



We will use the (nOption) variable to get the input from the User!





Using (Switch) we will determine what to do based on the selected option

Search Switch

Components Tree

- Header
- New Line
- Image
- Comment
- Templates
- Basic Program
- Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop

Switch Component

Variable : nOption

 Again

 Ok

 Close



Search Case

Components Tree

- Header
- New Line
- Image
- Comment
- Templates
- Basic Program
- Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop

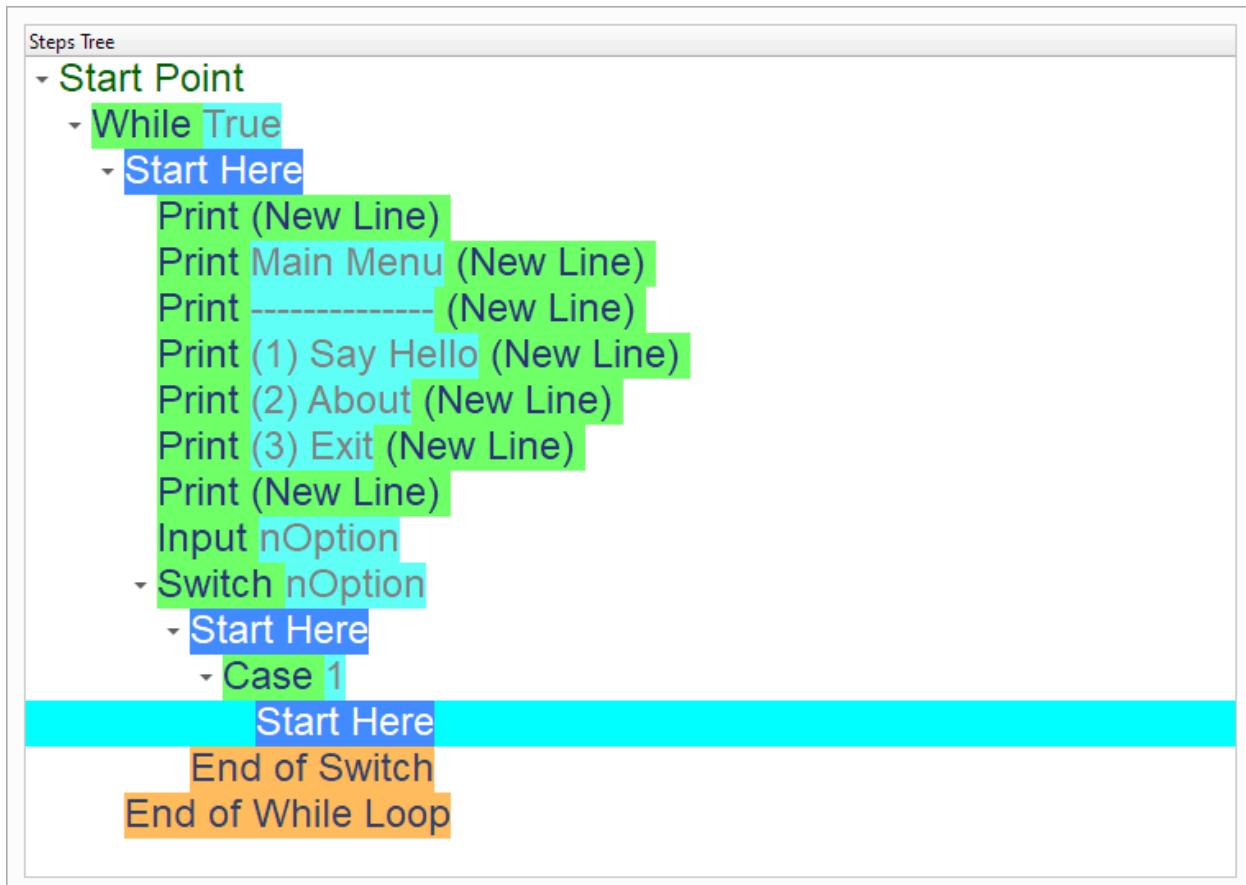
Case Component

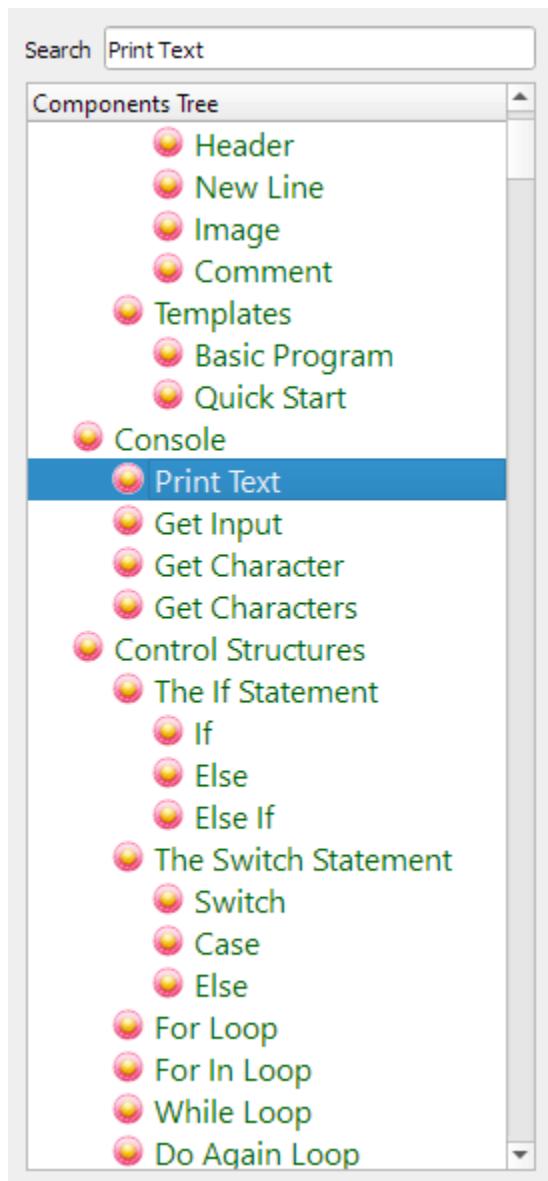
Value : 1

Again

Ok

Close





Print Component

Text : Hello!

Type :

Literal
Expression

New Line

 Again  Ok  Close

Steps Tree

- Start Point
 - While True
 - Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - Print ----- (New Line)
 - Print (1) Say Hello (New Line)
 - Print (2) About (New Line)
 - Print (3) Exit (New Line)
 - Print (New Line)
 - Input nOption
 - Switch nOption
 - Start Here
 - Case 1
 - Start Here
 - Print Hello! (New Line)
 - End of Switch
 - End of While Loop

Search Case

Components Tree

- Header
- New Line
- Image
- Comment
- Templates
- Basic Program
- Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop

Case Component

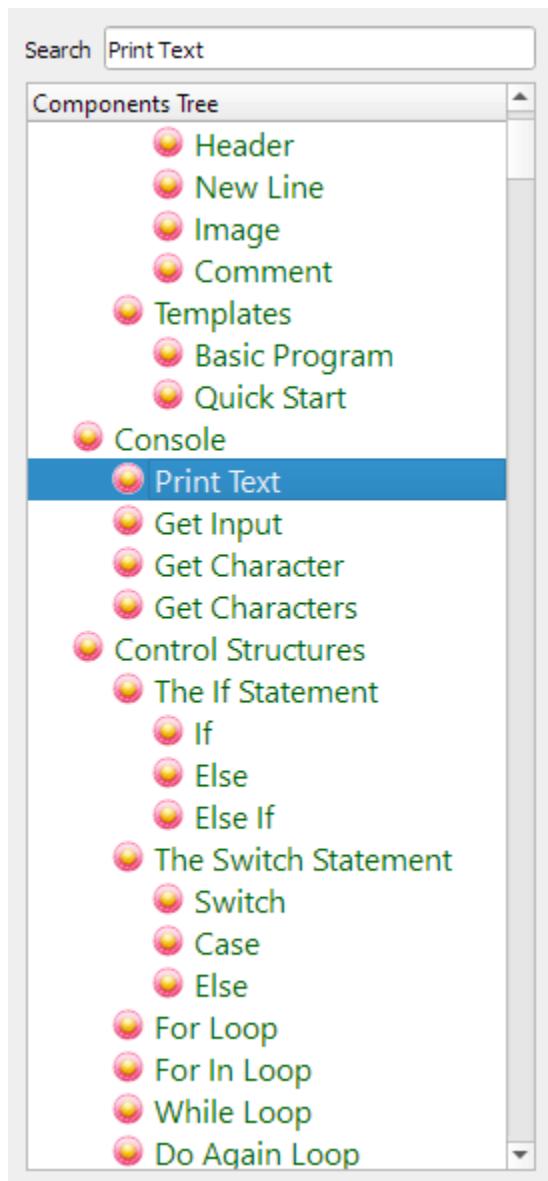
Value : 2

Again

Ok

Close





Print Component

Text : Just a simple program!

Type :

Literal
Expression

New Line

Steps Tree

- Start Here
 - Print (New Line)
 - Print Main Menu (New Line)
 - Print ----- (New Line)
 - Print (1) Say Hello (New Line)
 - Print (2) About (New Line)
 - Print (3) Exit (New Line)
 - Print (New Line)
 - Input nOption
 - Switch nOption
 - Start Here
 - Case 1
 - Start Here
 - Print Hello! (New Line)
 - Case 2
 - Start Here
 - Print Just a simple program! (New Line)
 - End of Switch

Search Case

Components Tree

- Header
- New Line
- Image
- Comment
- Templates
- Basic Program
- Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop

Case Component

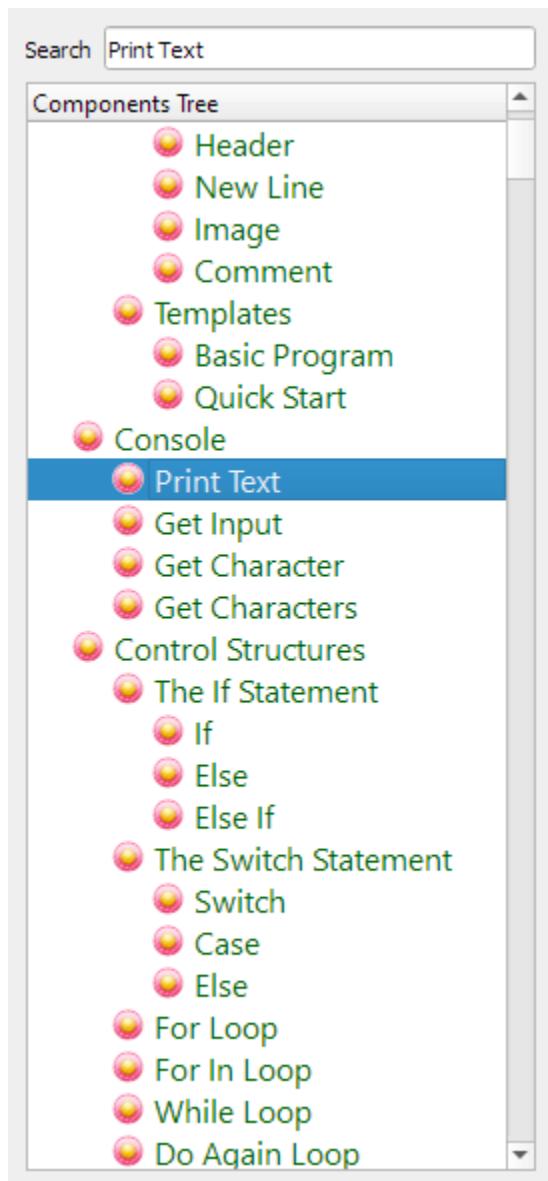
Value : 3

 Again

 Ok

 Close





Print Component

Text :

Type :

Literal
Expression

New Line

Steps Tree

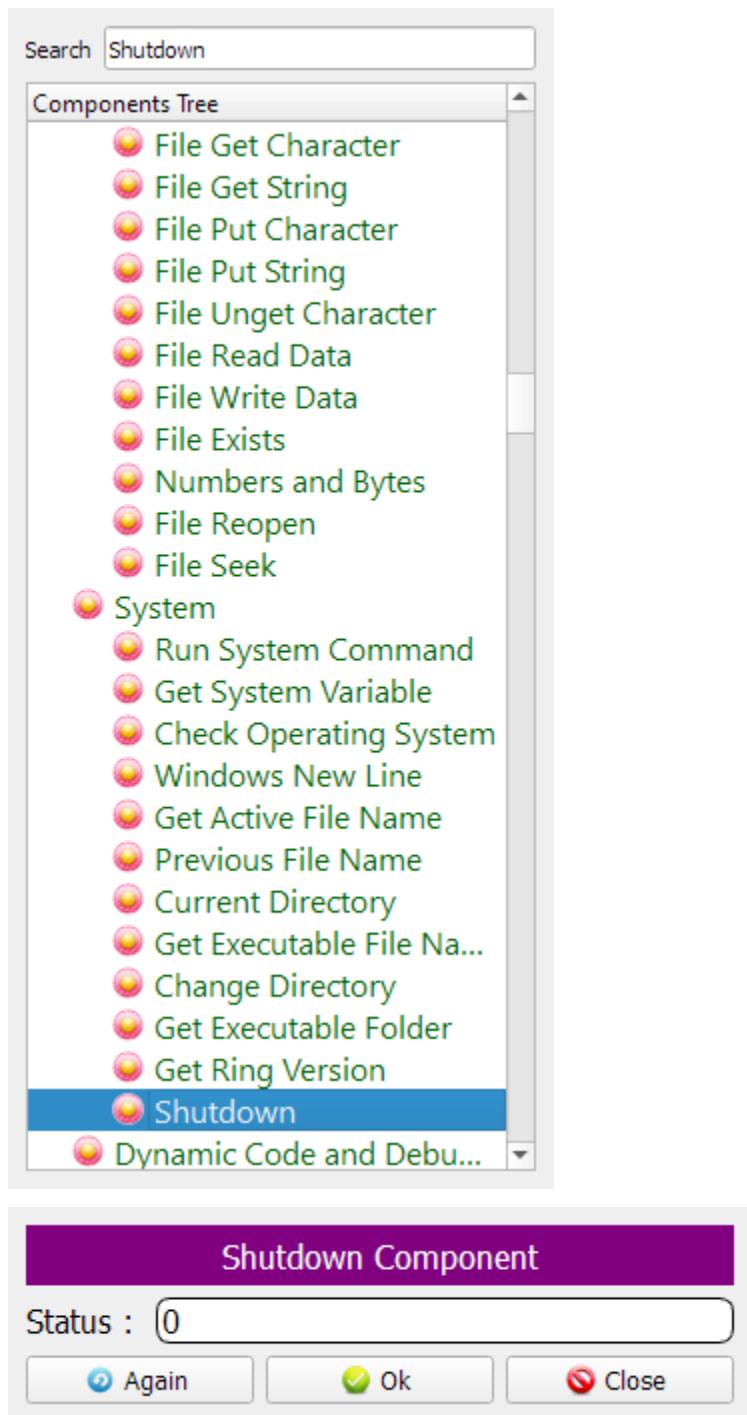
```

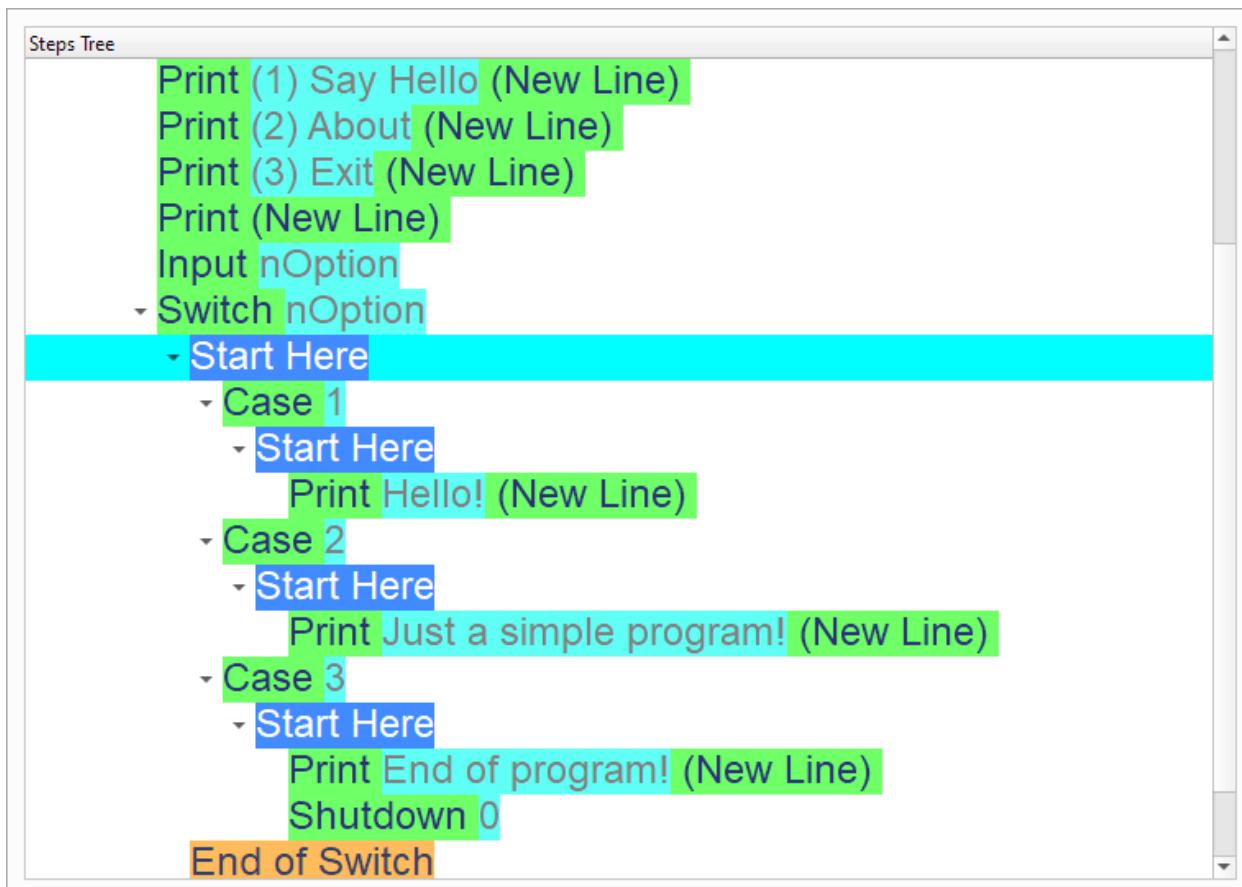
Print ----- (New Line)
Print (1) Say Hello (New Line)
Print (2) About (New Line)
Print (3) Exit (New Line)
Print (New Line)
Input nOption
Switch nOption
  - Start Here
    - Case 1
      - Start Here
        Print Hello! (New Line)
    - Case 2
      - Start Here
        Print Just a simple program! (New Line)
    - Case 3
      - Start Here
        Print End of program! (New Line)
  End of Switch

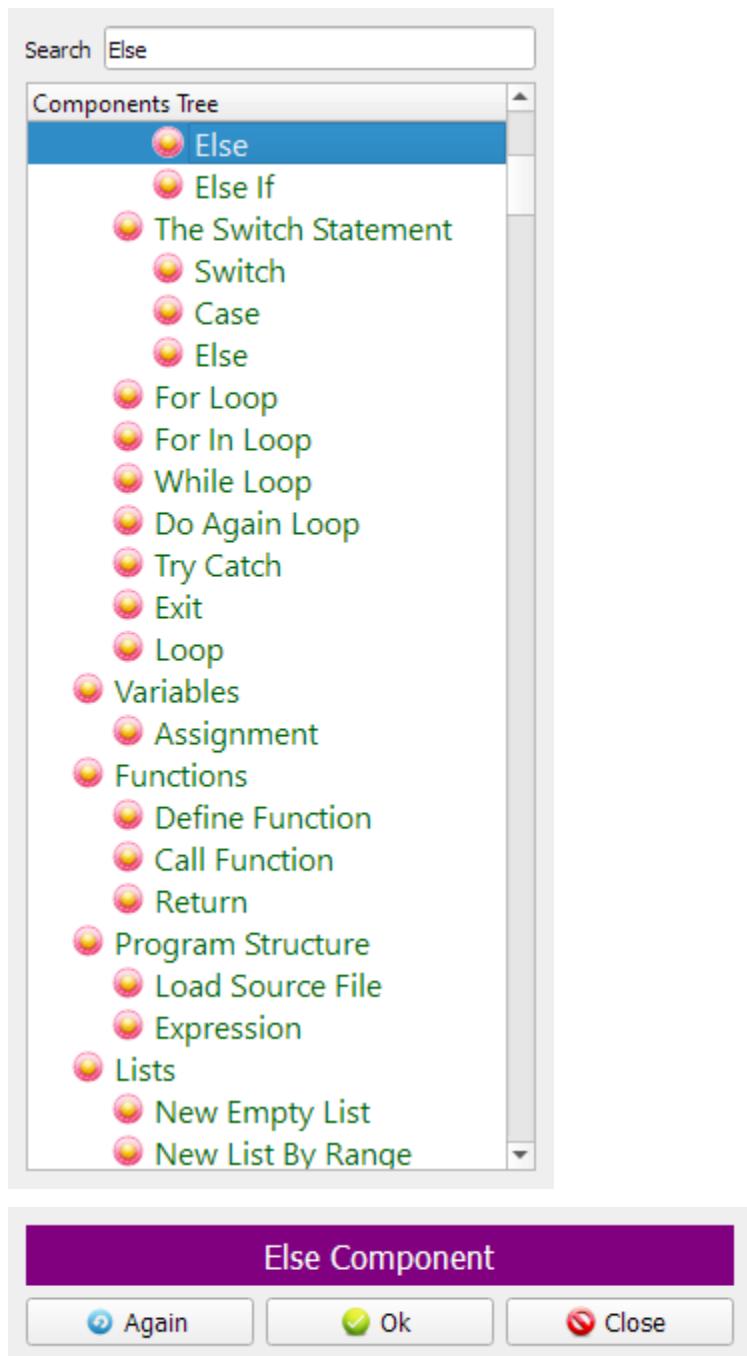
```

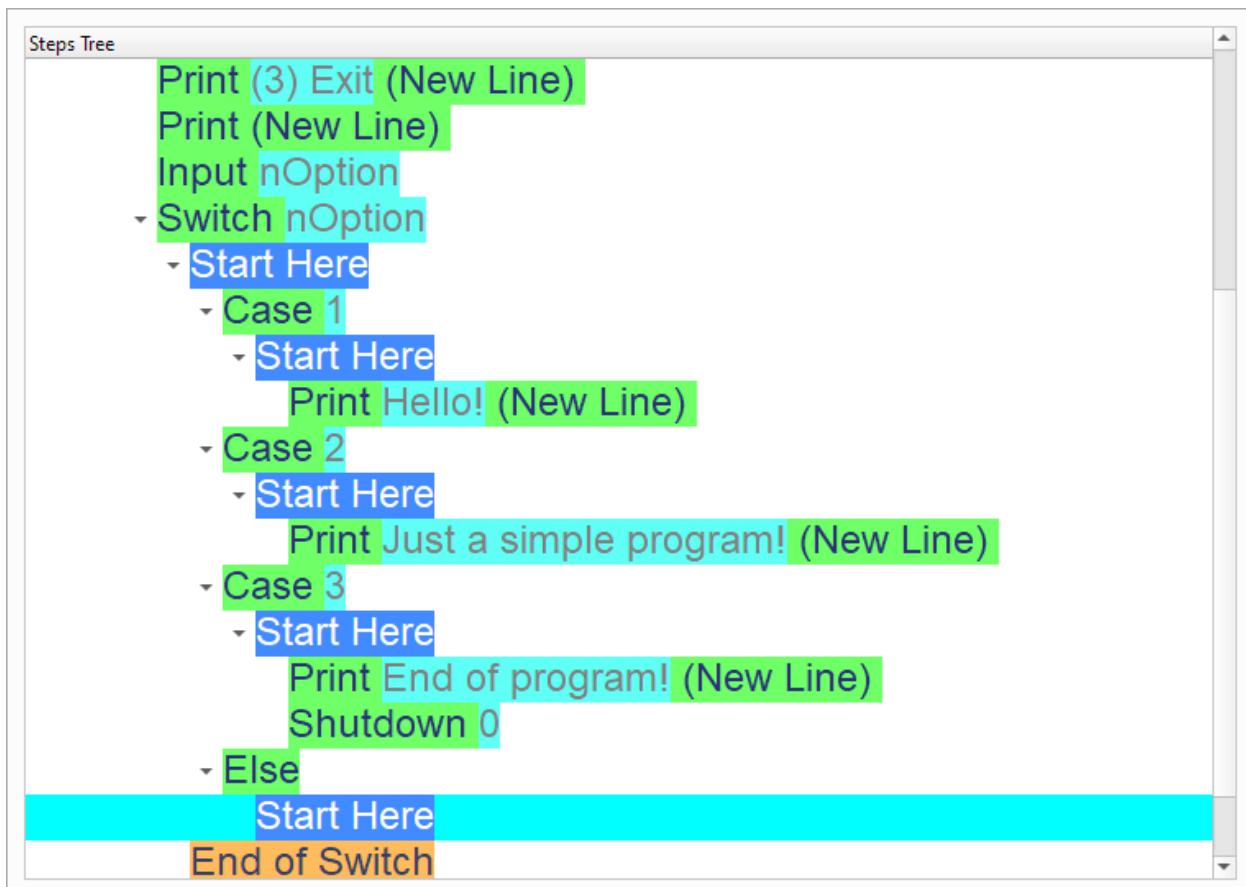
Select the (Shutdown) component

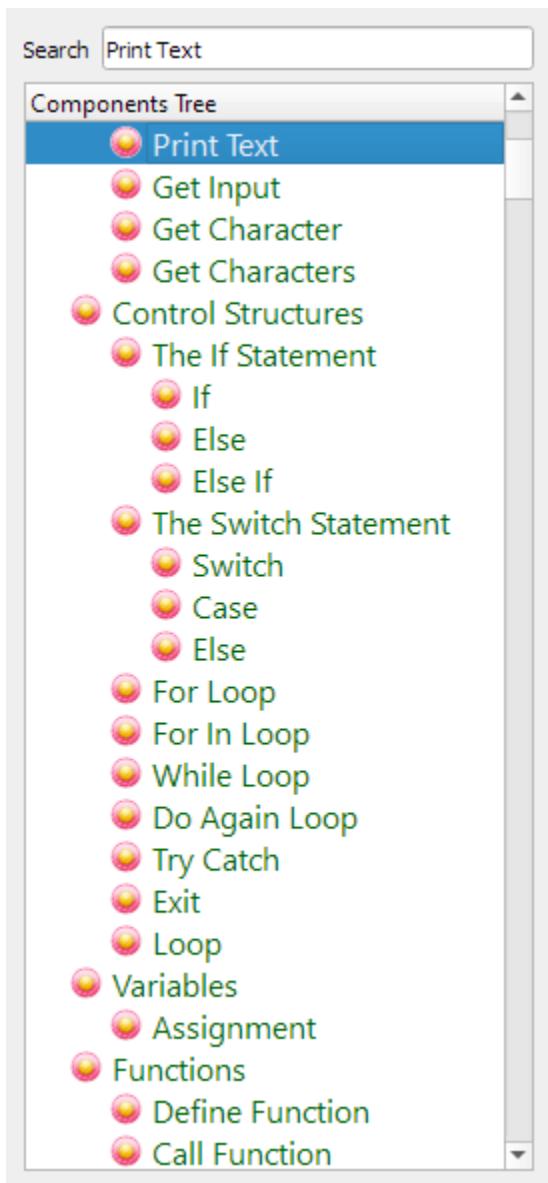
Using this component we can close our program!

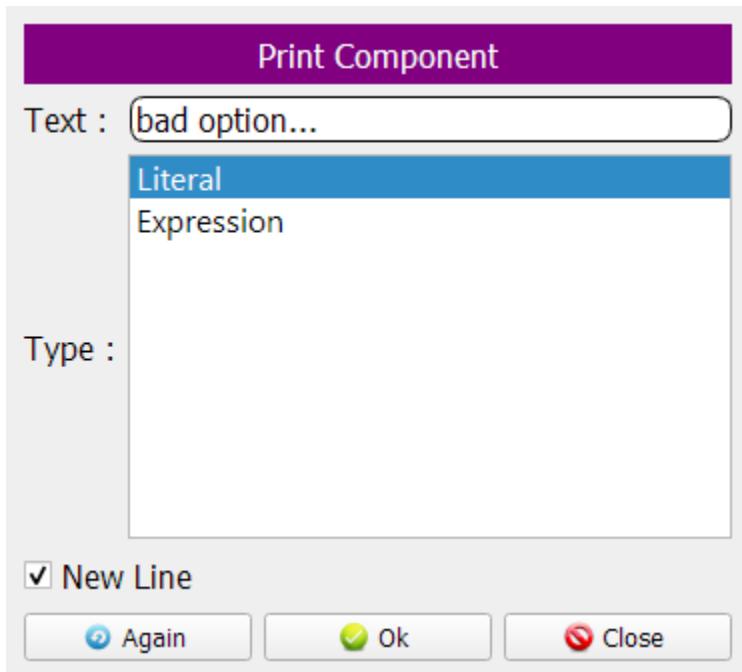




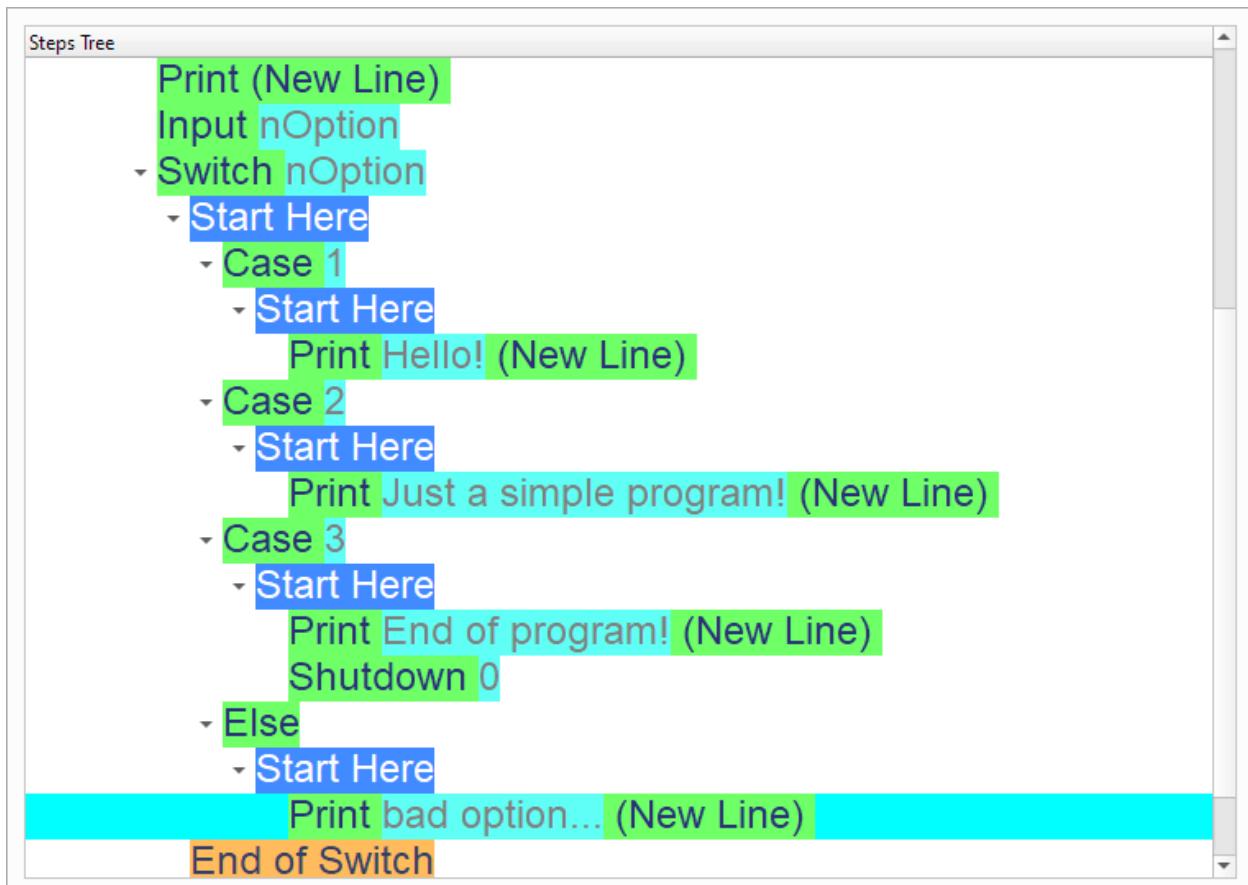








Now we have the final Steps Tree in our program



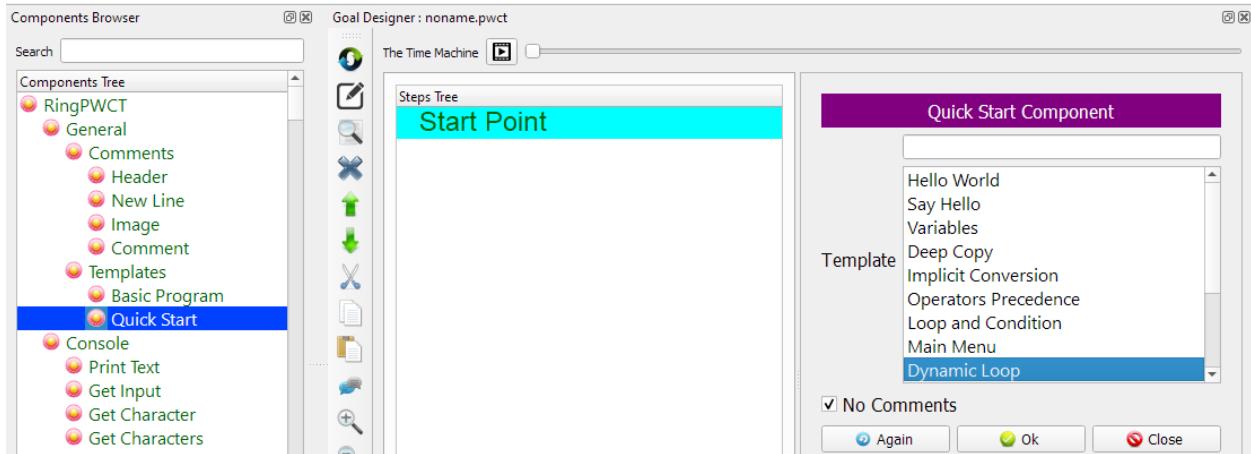
CHAPTER FIFTEEN

DYNAMIC LOOP

In this chapter we are going to create the Dynamic Loop sample

15.1 Introduction

We can create this program quickly using the Quick Start component



15.2 Program Steps

After selecting the (Dynamic Loop) template, we will get the next steps in the Goal Designer



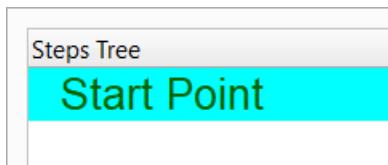
15.3 Creating the Program

To create this program we will use the next components

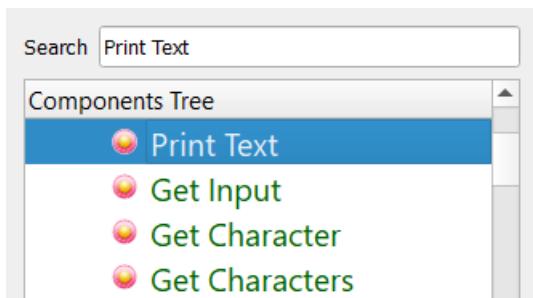
- Print Text
- Get Input
- Assignment
- For Loop

Note: This example uses the Assignment component to convert Strings to Numbers, but we can just let the Get Input component do the conversion for us.

In the begining the Steps Tree is empty



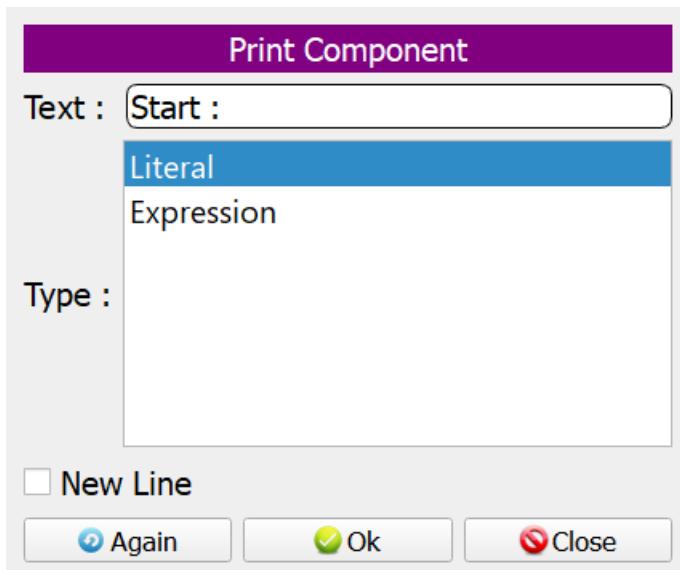
Select the (Print Text) component



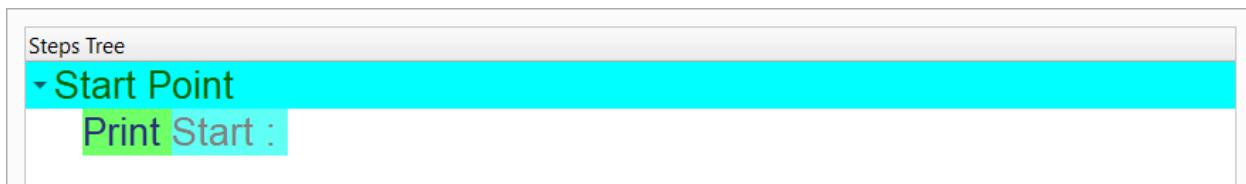
Enter the data in the Interaction Page

Text: Start :

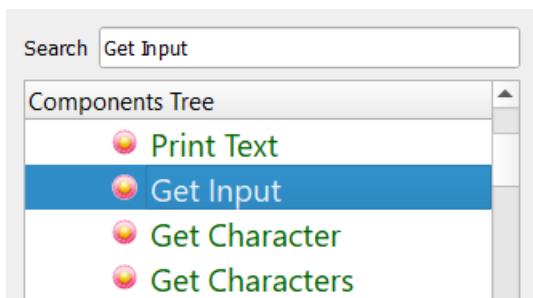
Type: Literal



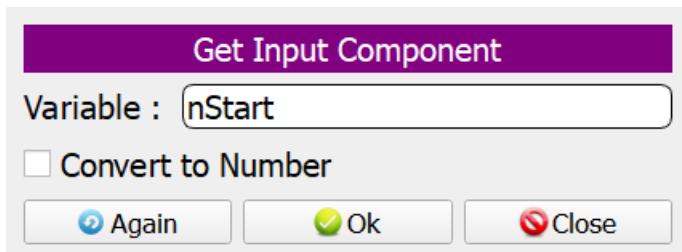
The Steps Tree will be updated



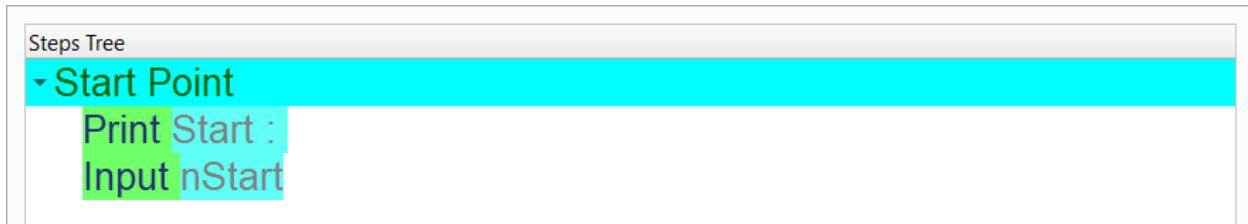
Select the (Get Input) component



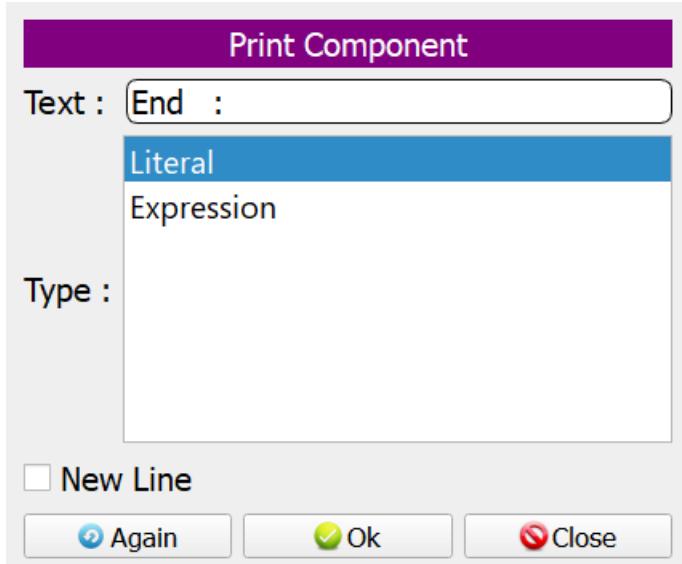
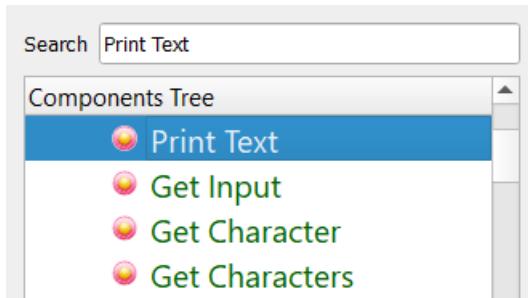
The variable name will be (nStart)



The Steps Tree will be updated

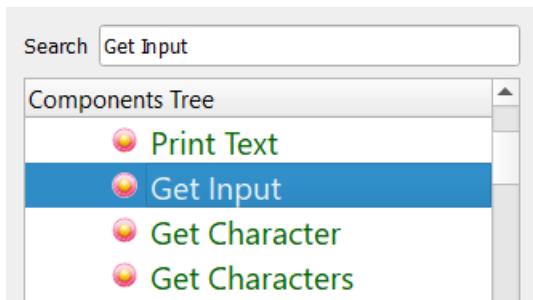


Print (End :)

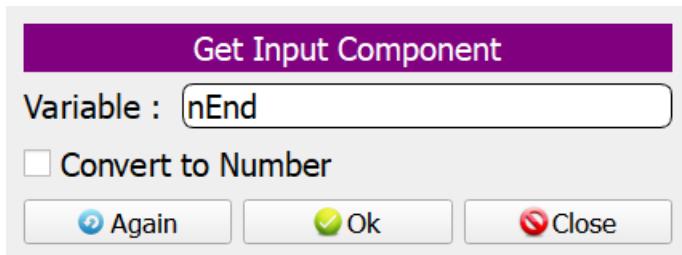




Use the (Get Input) component



The variable name will be (nEnd)



The Steps Tree will be updated



Print (Step :)

The screenshot shows the PWCT software interface. At the top, there is a search bar with the text "Print Text". Below it is a "Components Tree" pane containing the following components:

- Print Text (selected)
- Get Input
- Get Character
- Get Characters

Below the components tree is a dialog box titled "Print Component". It has the following fields:

- Text :** Step :
- Type :** Literal (selected), Expression
- New Line** checkbox (unchecked)

At the bottom of the dialog are three buttons: "Again" (refresh icon), "Ok" (checkmark icon), and "Close" (cross icon).

Below the dialog is a "Steps Tree" pane. The tree is expanded to show the "Start Point" node, which contains the following steps:

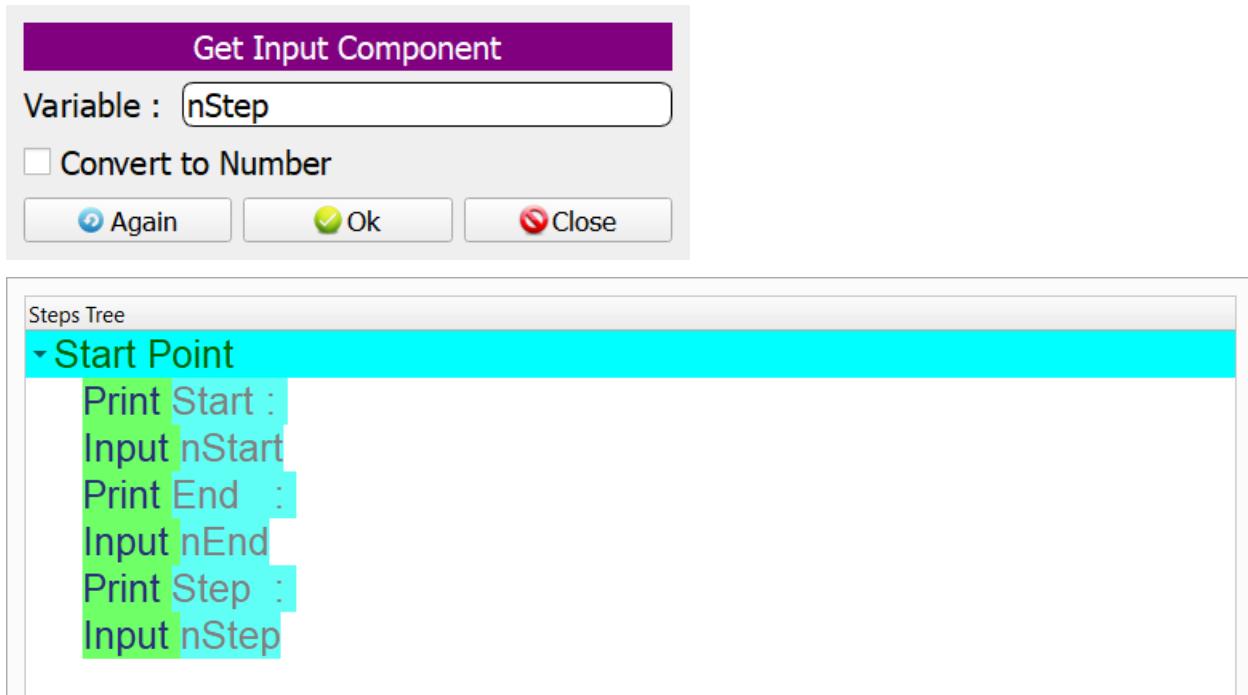
- Print Start :
- Input nStart
- Print End :
- Input nEnd
- Print Step :

Use the (Get Input) component

The screenshot shows the PWCT software interface. At the top, there is a search bar with the text "Get Input". Below it is a "Components Tree" pane containing the following components:

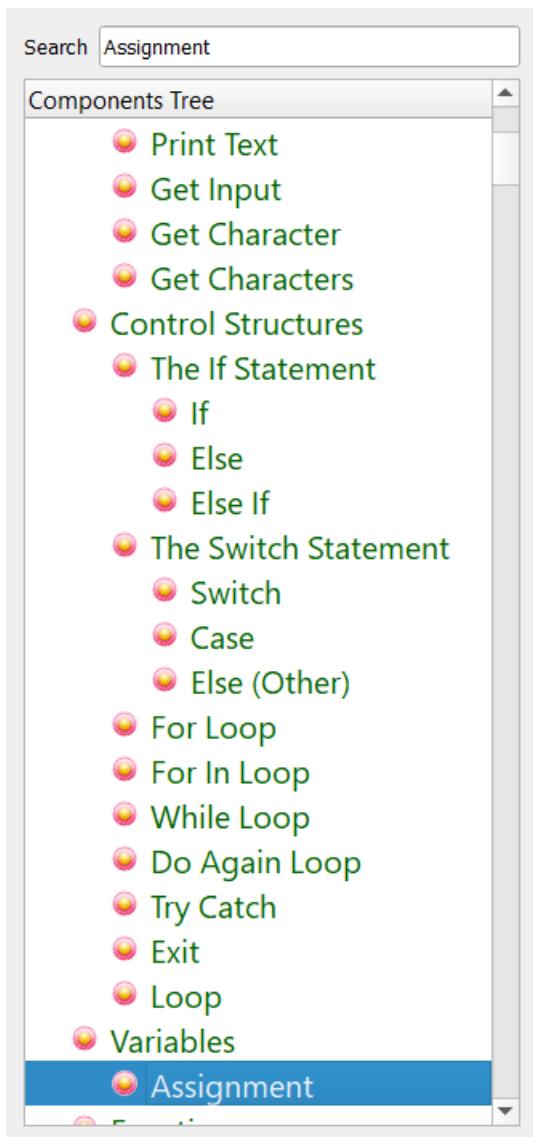
- Print Text
- Get Input (selected)
- Get Character
- Get Characters

The variable name will be (nStep)

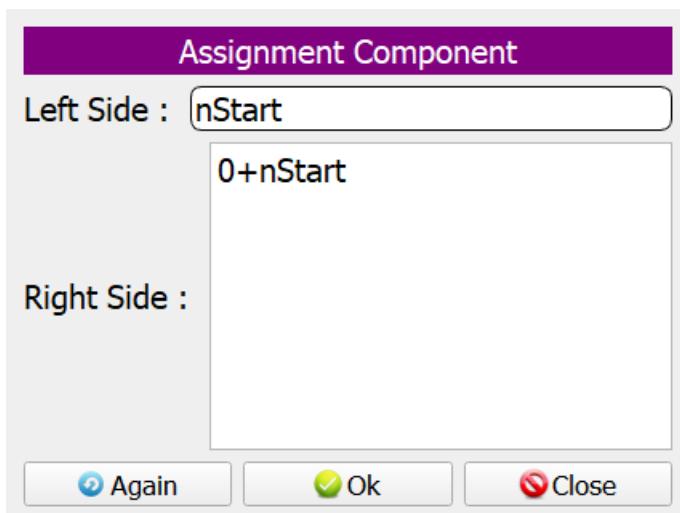


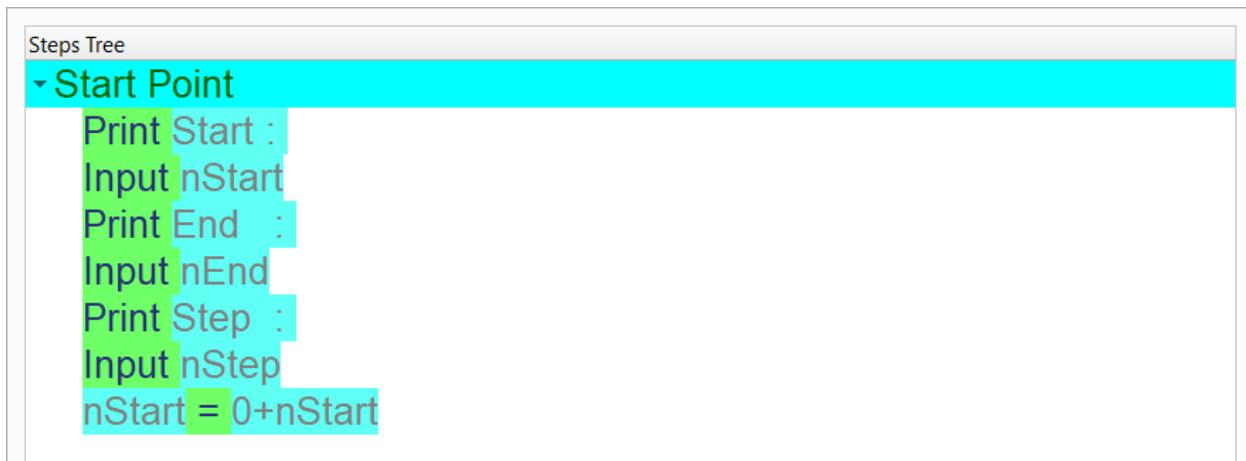
Convert the variables (nStart, nEnd and nStep) from String to Number

Use the (Assignment) component

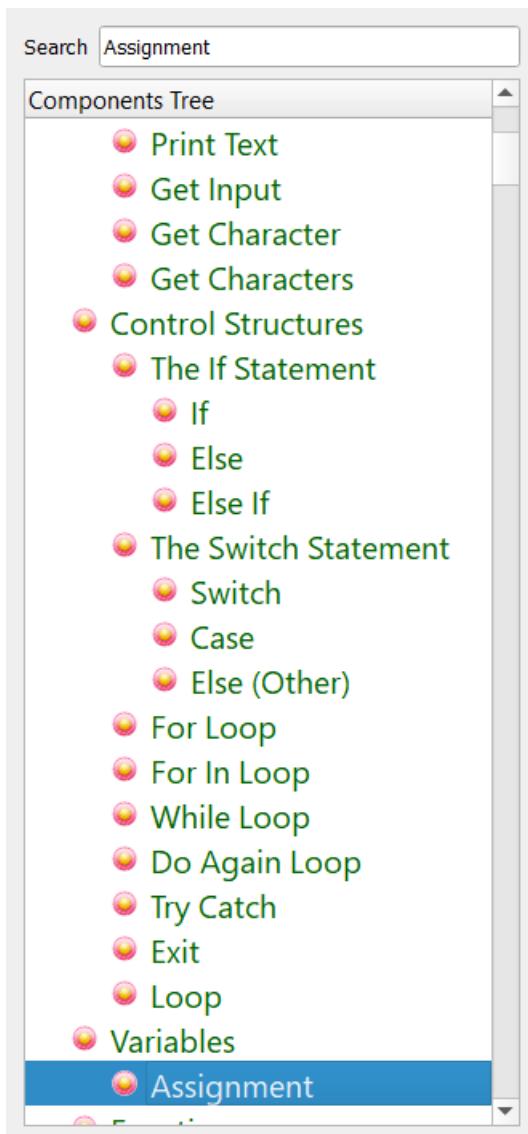


Convert the variable (nStart) from String to Number





Use the (Assignment) component



Convert the variable (nEnd) from String to Number

Assignment Component

Left Side : nEnd

Right Side :

0+nEnd

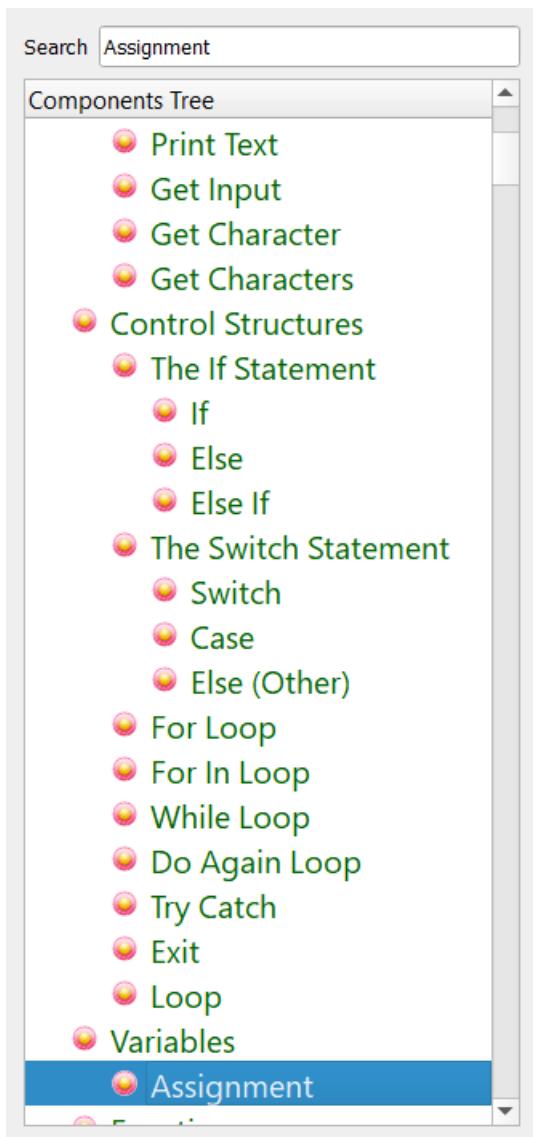
Again Ok Close

Steps Tree

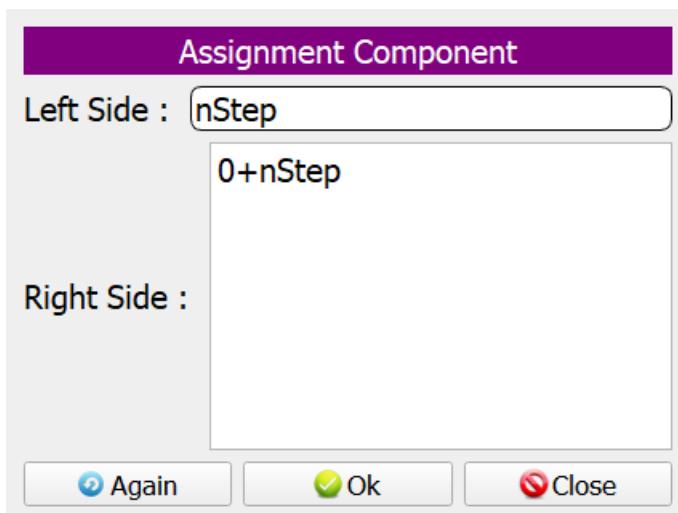
- Start Point

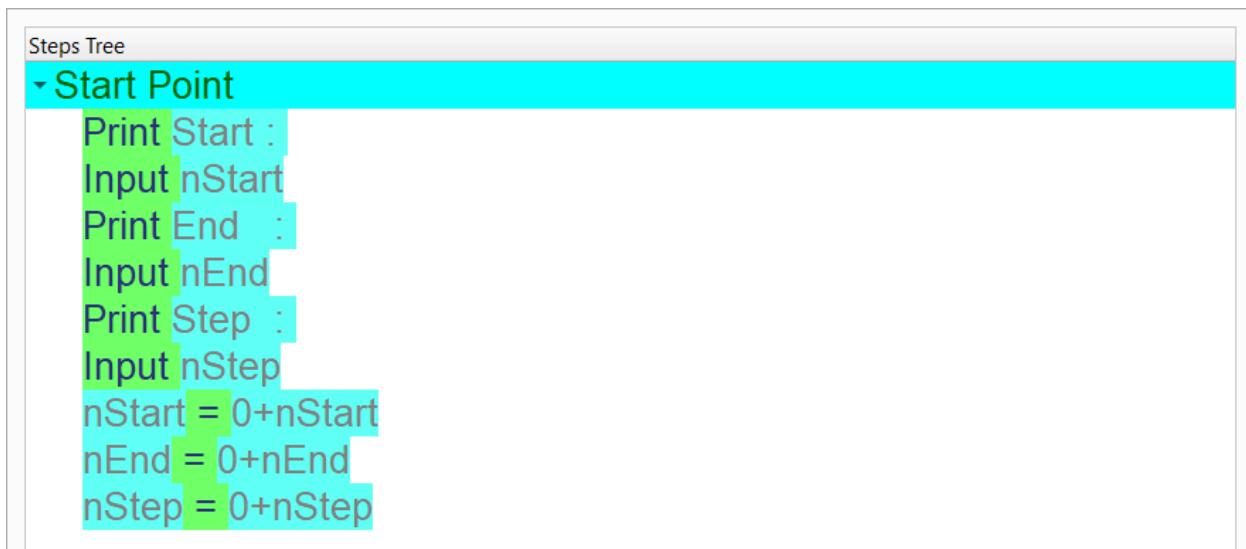
Print Start :
Input nStart
Print End :
Input nEnd
Print Step :
Input nStep
 $nStart = 0 + nStart$
 $nEnd = 0 + nEnd$

Use the (Assignment) component

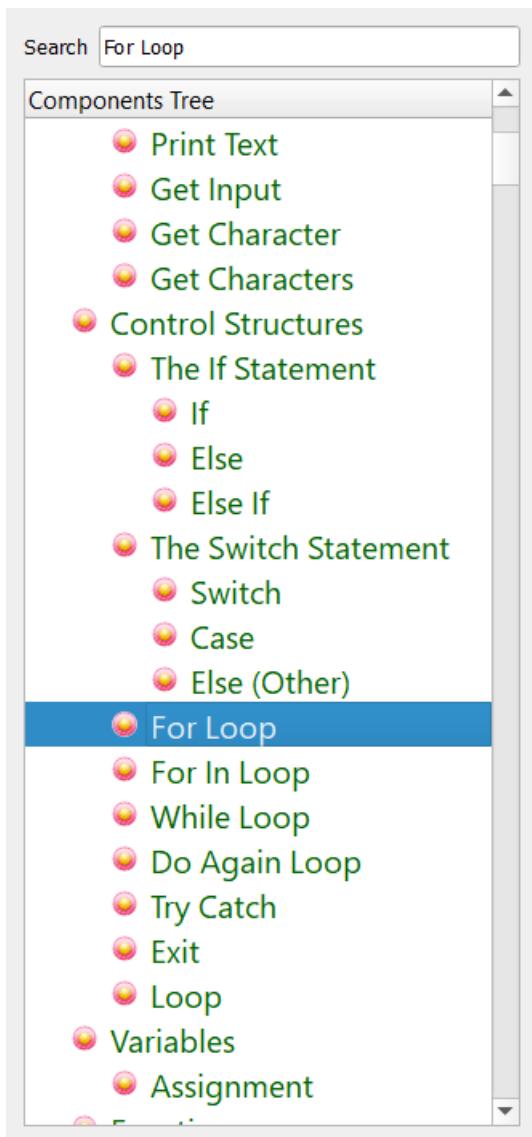


Convert the variable (nStep) from String to Number





Select the (For Loop) component



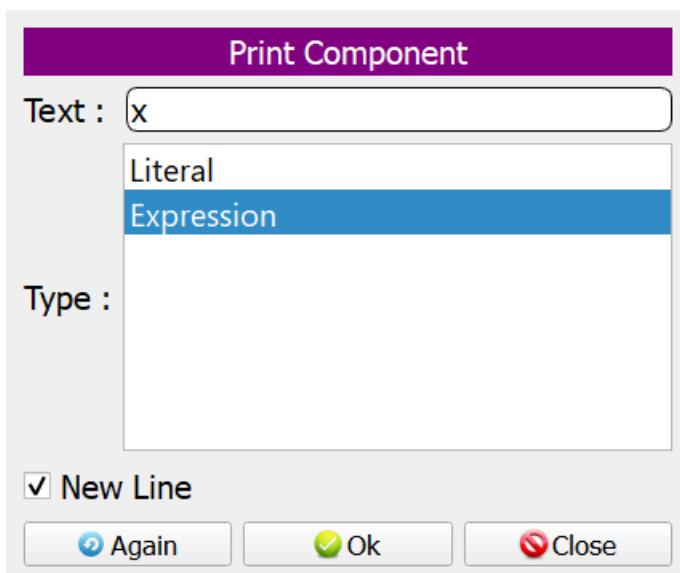
Enter the data in the Interaction Page

Start: x = nStart End: nEnd Step: nStep

For Loop Component	
Start :	<input type="text" value="x = nStart"/>
To :	<input type="text" value="nEnd"/>
Step :	<input type="text" value="nStep"/>
Again	Ok
Close	



Use the (Print Text) component to print the number



Now we have the final Steps Tree in our program



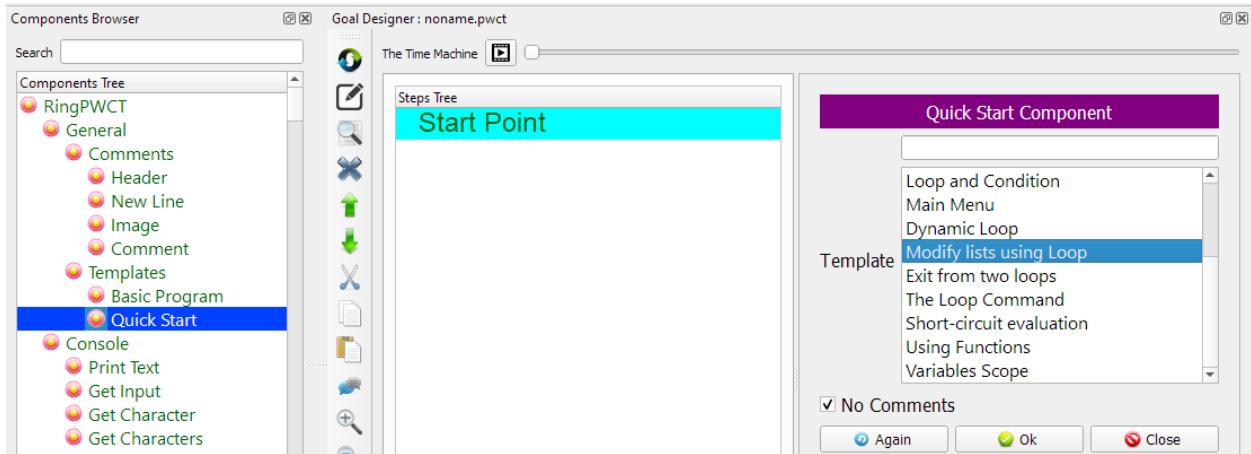
CHAPTER SIXTEEN

MODIFY LISTS

In this chapter we are going to learn about the Modify Lists

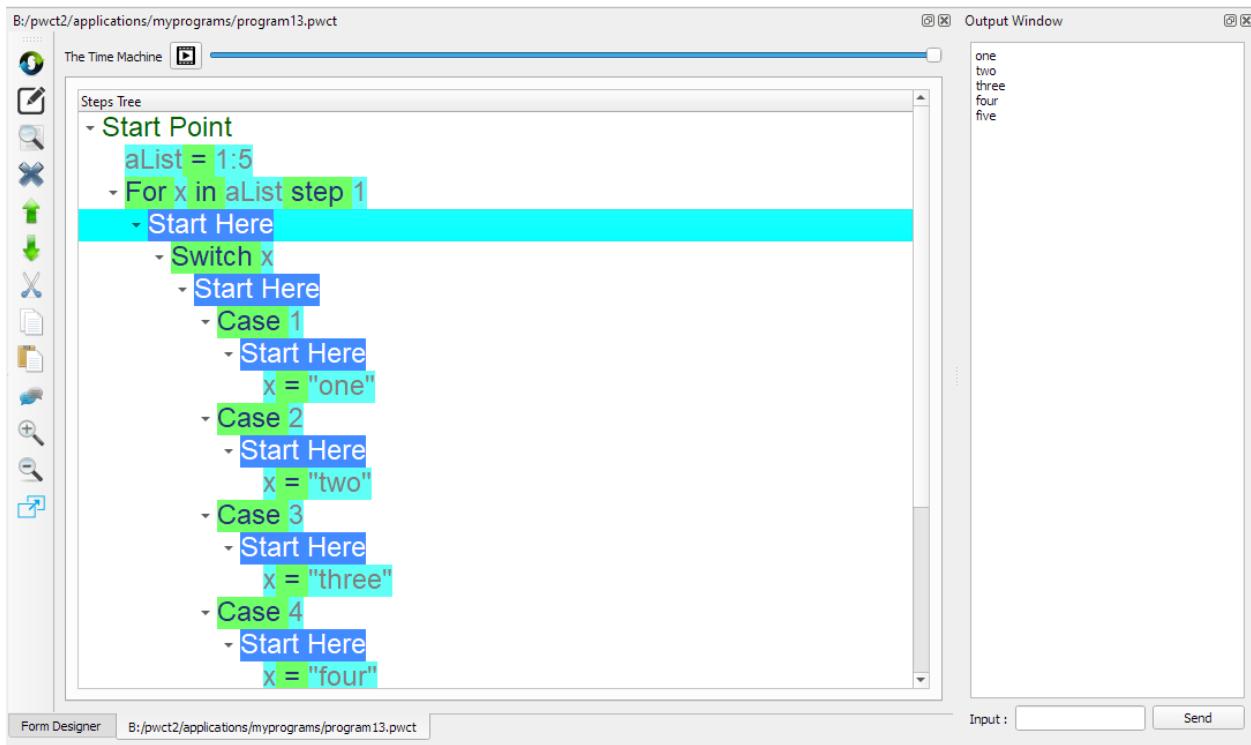
16.1 Introduction

We can create this program quickly using the Quick Start component



16.2 Program Steps

After selecting the (Modify Lists) template, we will get the next steps in the Goal Designer



The Steps Tree:

```
aList = 1:5
For x in aList step 1
    Switch x
        Case 1
            x = "one"
        Case 2
            x = "two"
        Case 3
            x = "three"
        Case 4
            x = "four"
        Case 5
            x = "five"
    End of Switch
End of For Loop
Print aList
```

16.3 Creating the Program

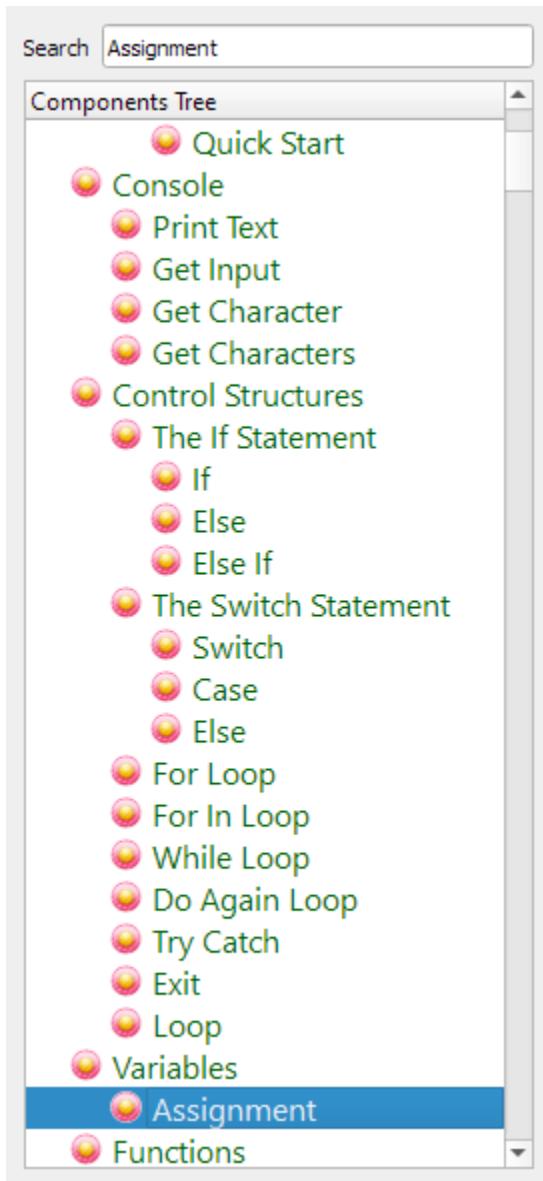
To create this program we will use the next components

- Assignment
- For In Loop
- Switch
- Case
- Print Text

In the begining the Steps Tree is empty



Select the (Assignment) component



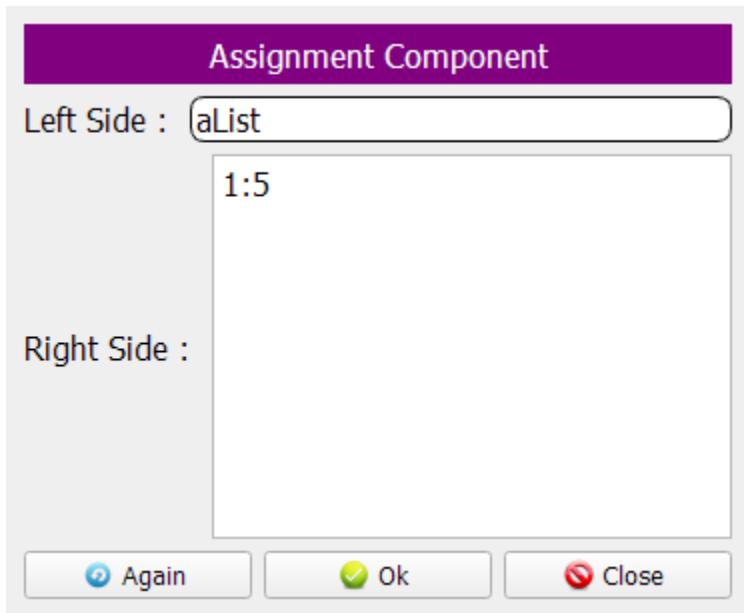
Enter the data in the Interaction Page

Left side: aList

Right side: 1:5

This will create a list contains the numbers from 1 to 5

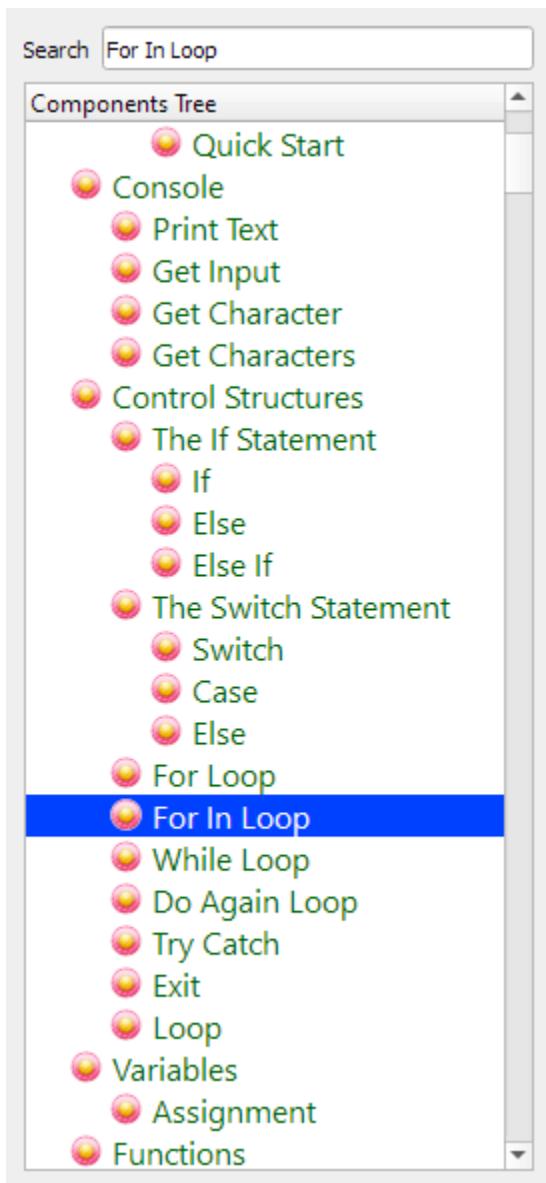
i.e. aList = [1,2,3,4,5]



The Steps Tree will be updated



Select the (For In) component



Enter the data in the Interaction Page

Variable: x

In: aList

Step: 1

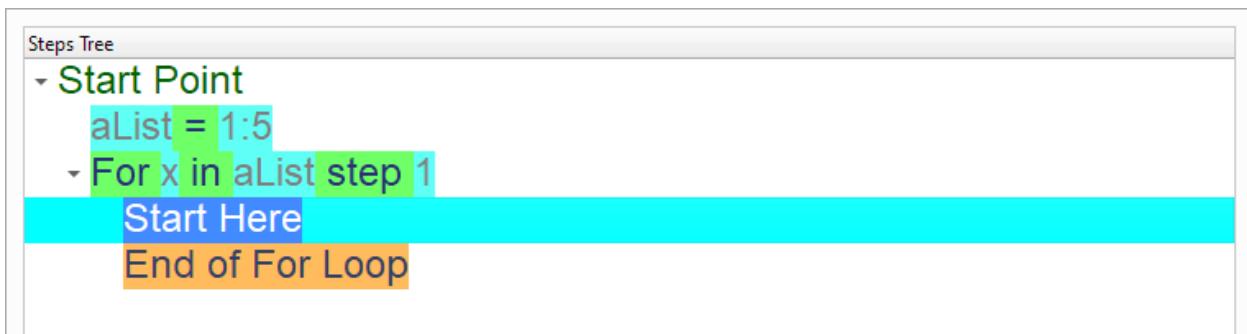
For In Component

Variable :

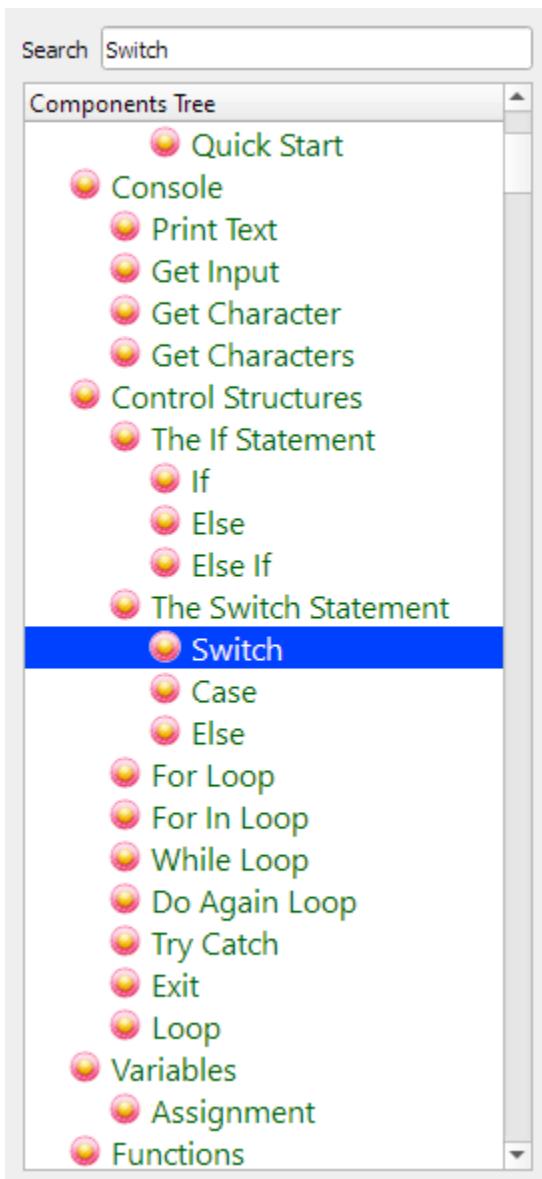
In :

Step :

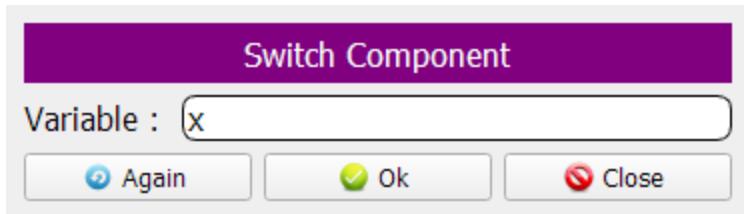
The Steps Tree will be updated



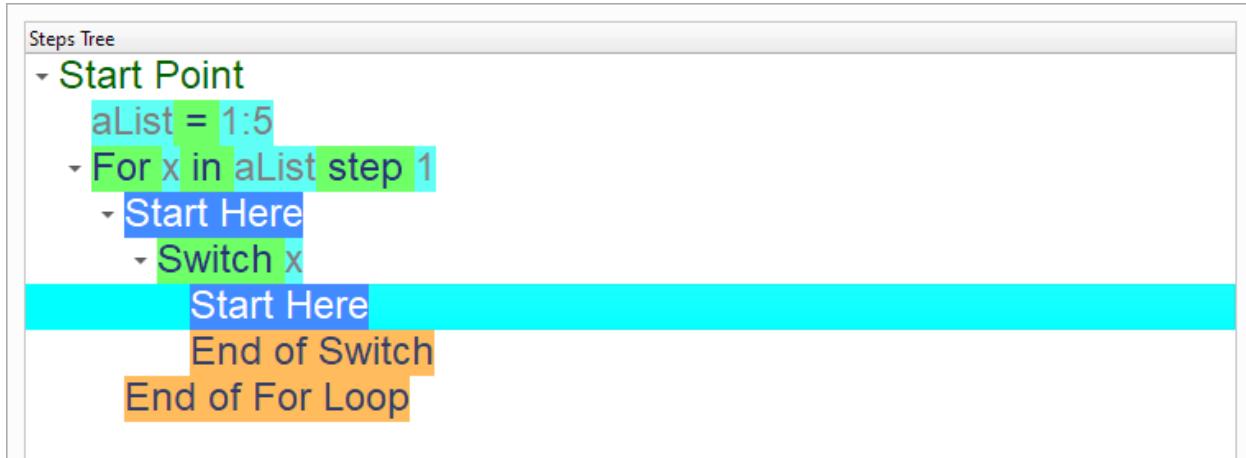
Select the (Switch) component



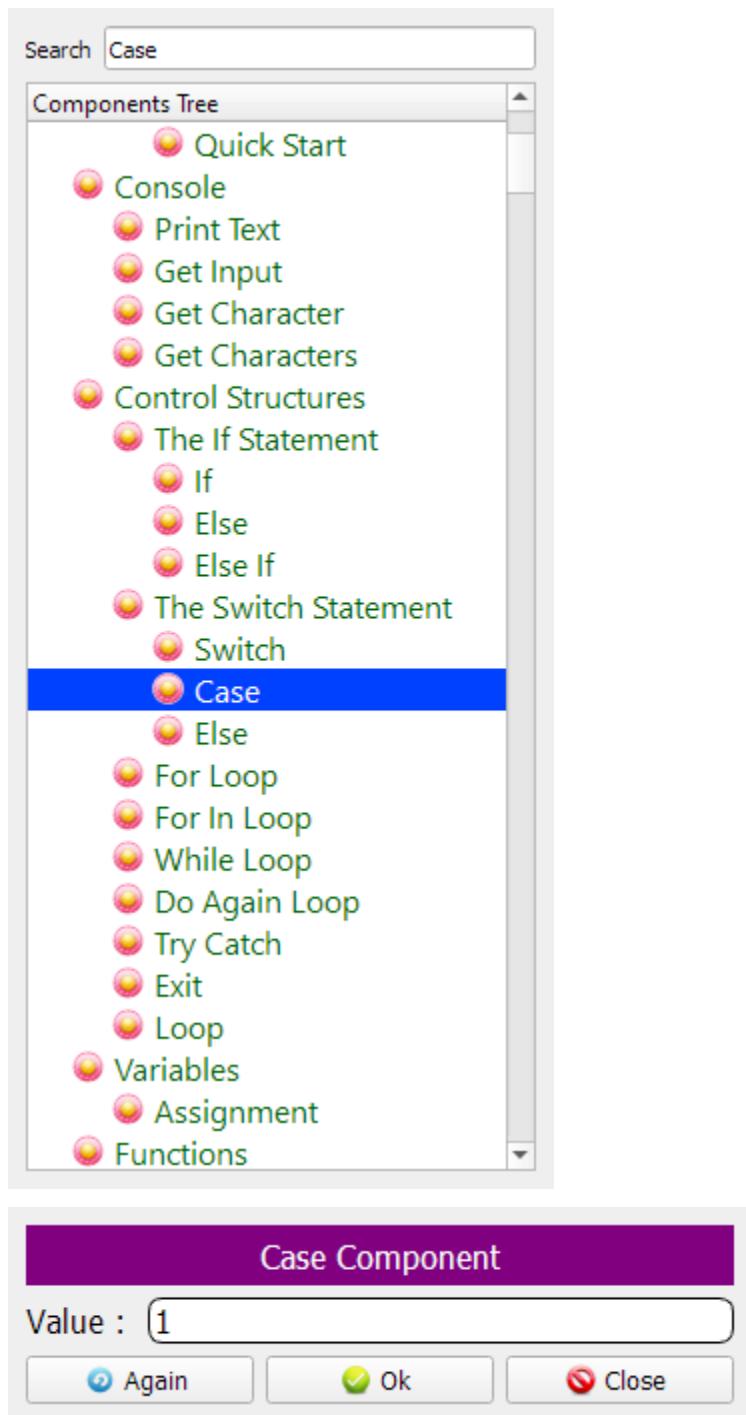
The variable will be (x)



We will update each list item based on the item number



if the item number is 1, set the item value to “one”



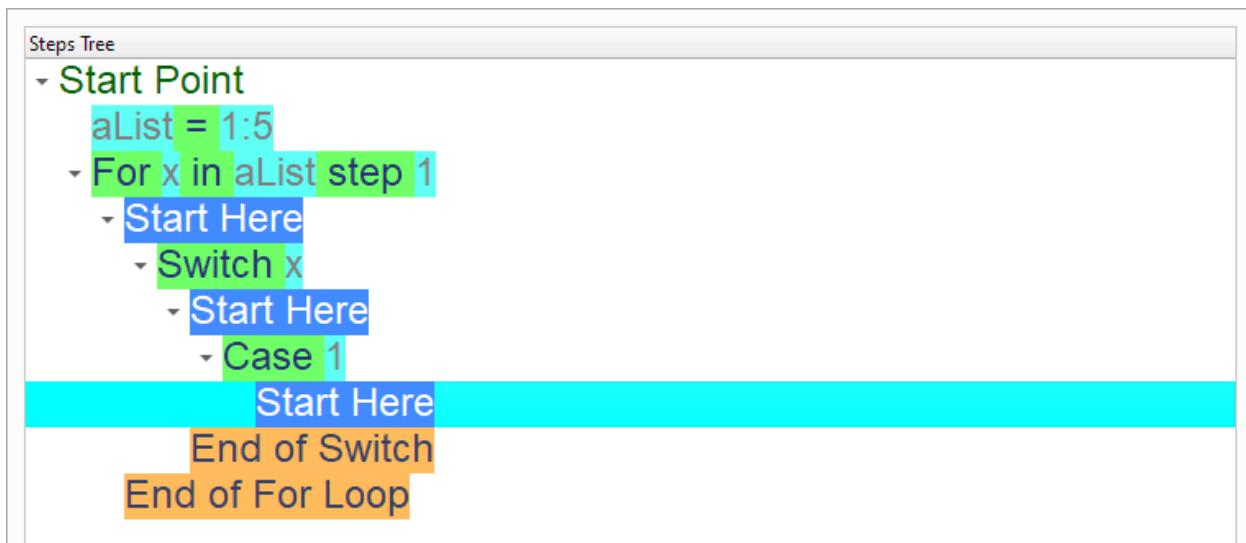
Case Component

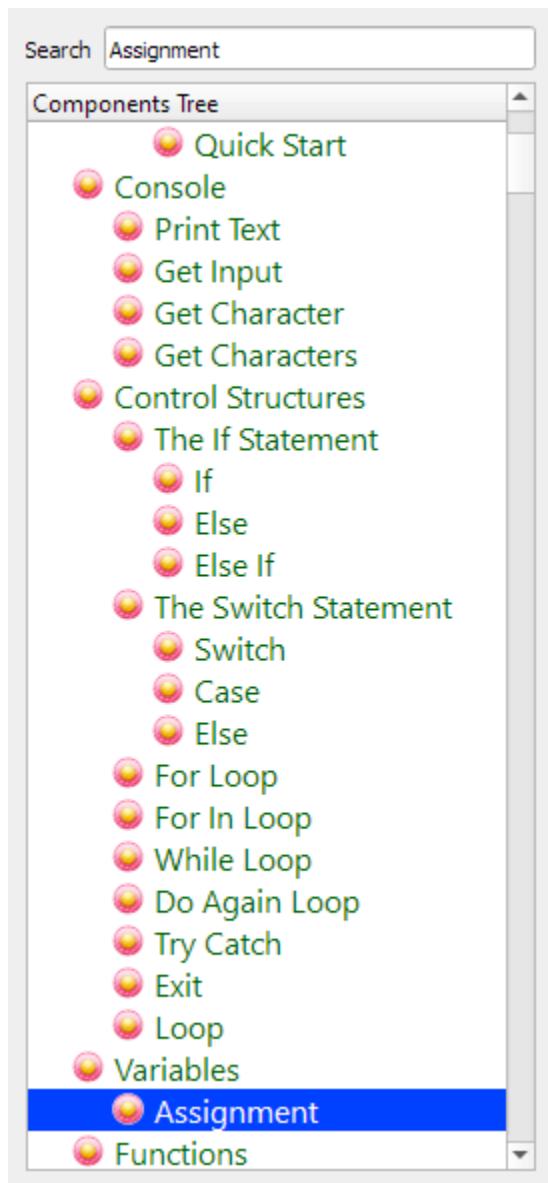
Value :

Again

Ok

Close





Assignment Component

Left Side :

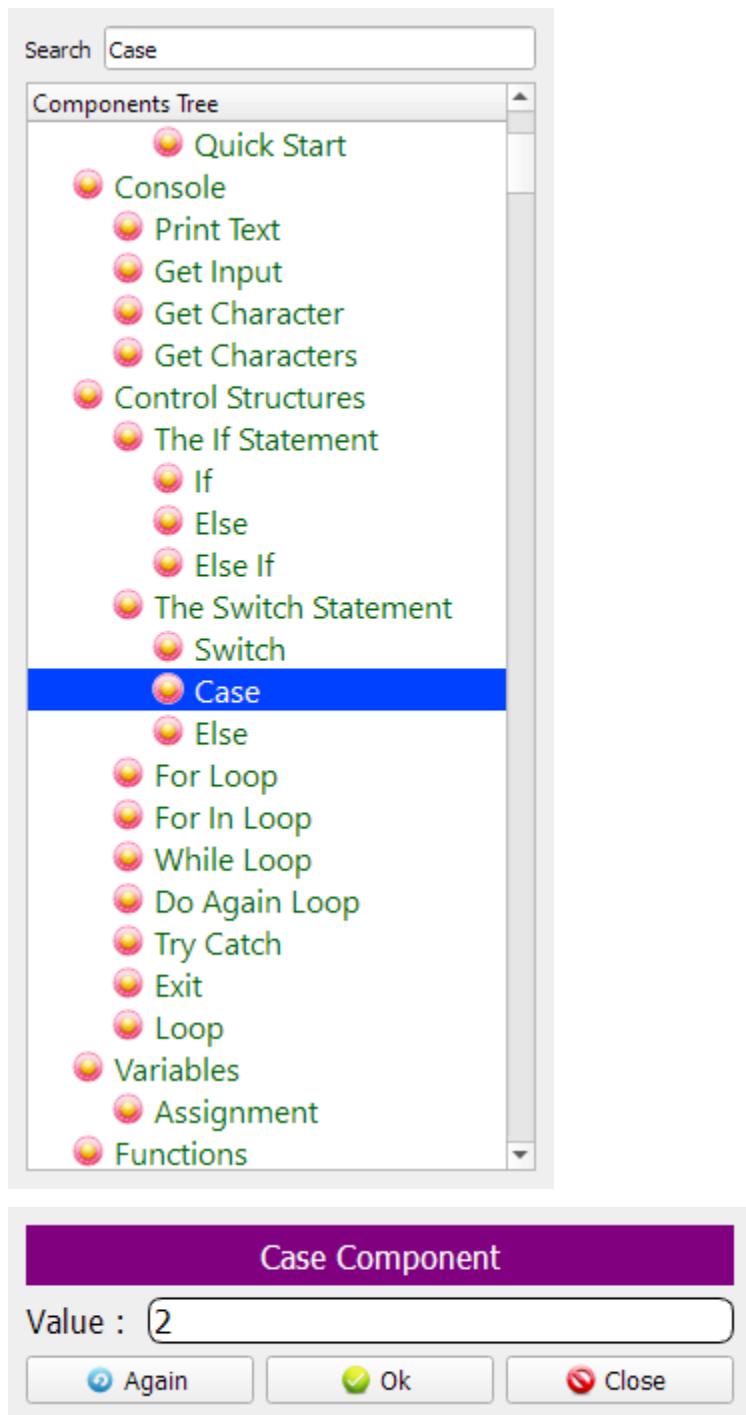
Right Side :

"one"

Steps Tree

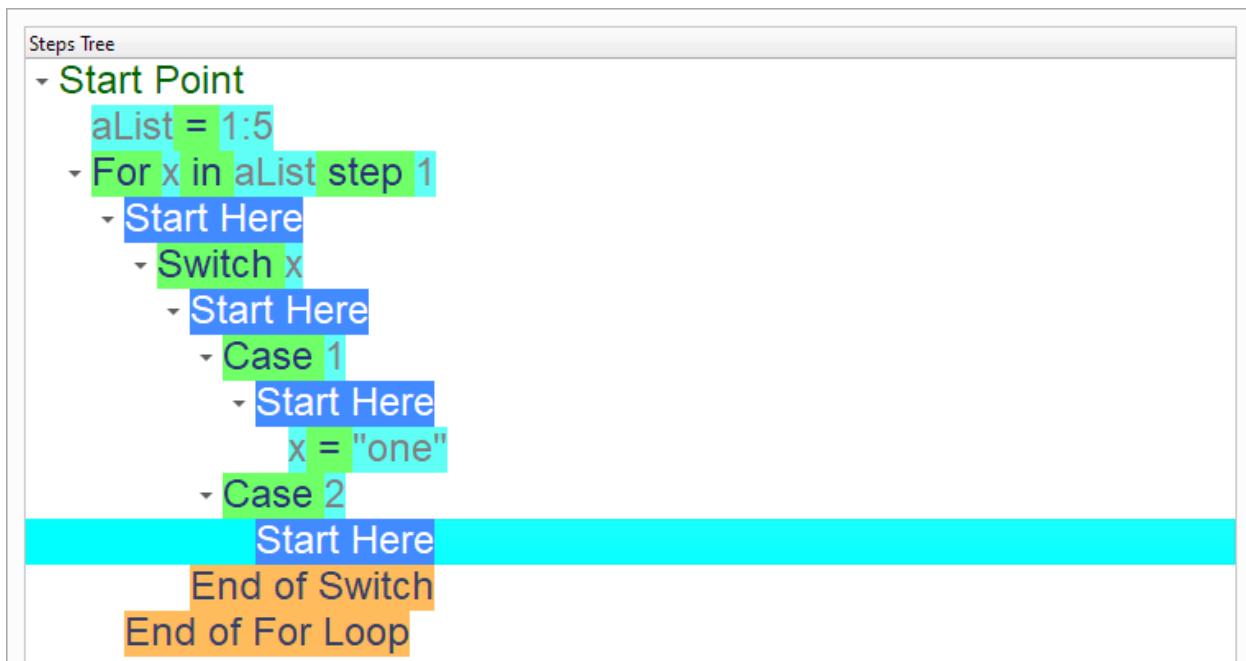
- Start Point
 - aList = 1:5
 - For x in aList step 1
 - Start Here
 - Switch x
 - Start Here
 - Case 1
 - Start Here
 - x = "one"
 - End of Switch
 - End of For Loop

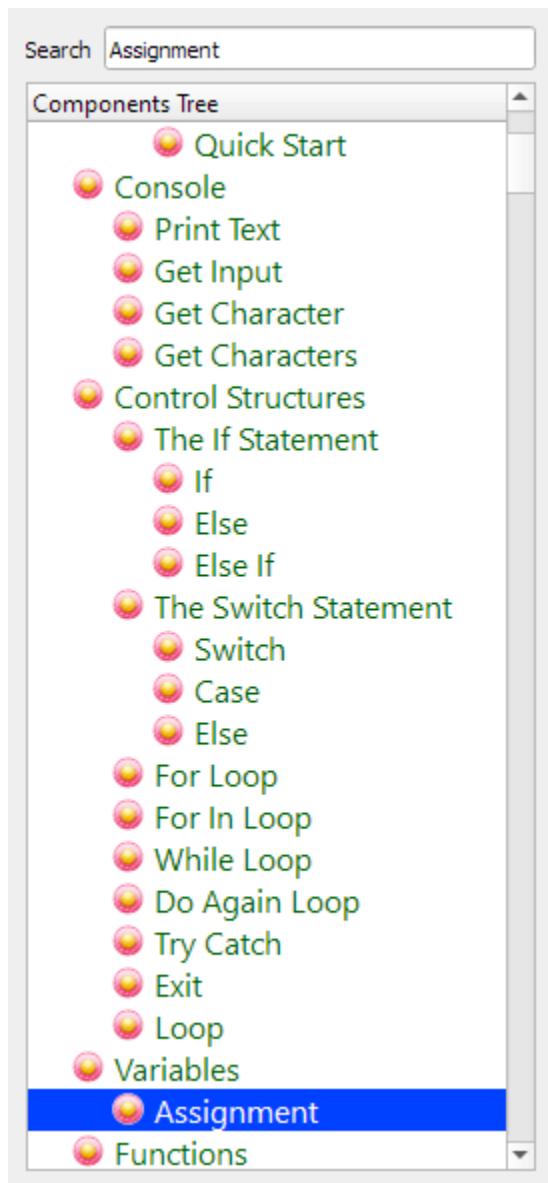
if the item number is 2, set the item value to “two”



Case Component

Value : 2





Assignment Component

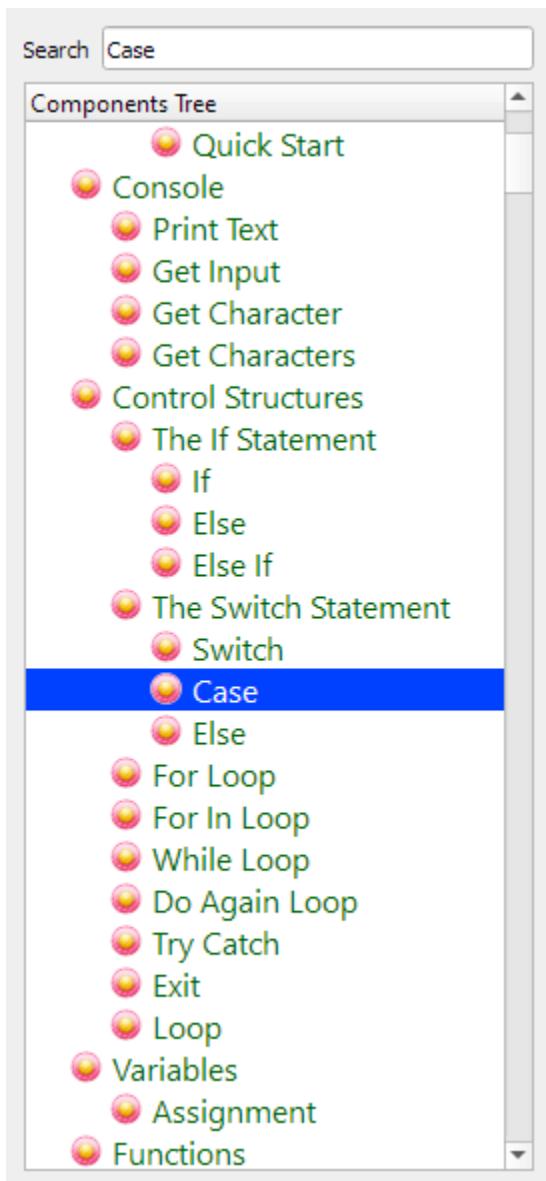
Left Side :

Right Side :

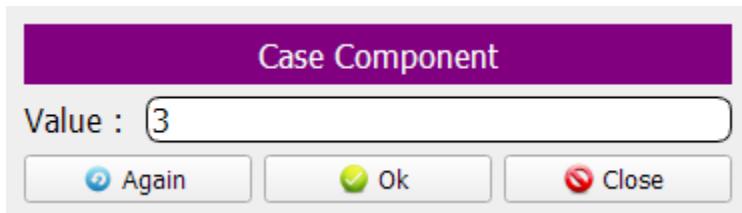
Again Ok Close

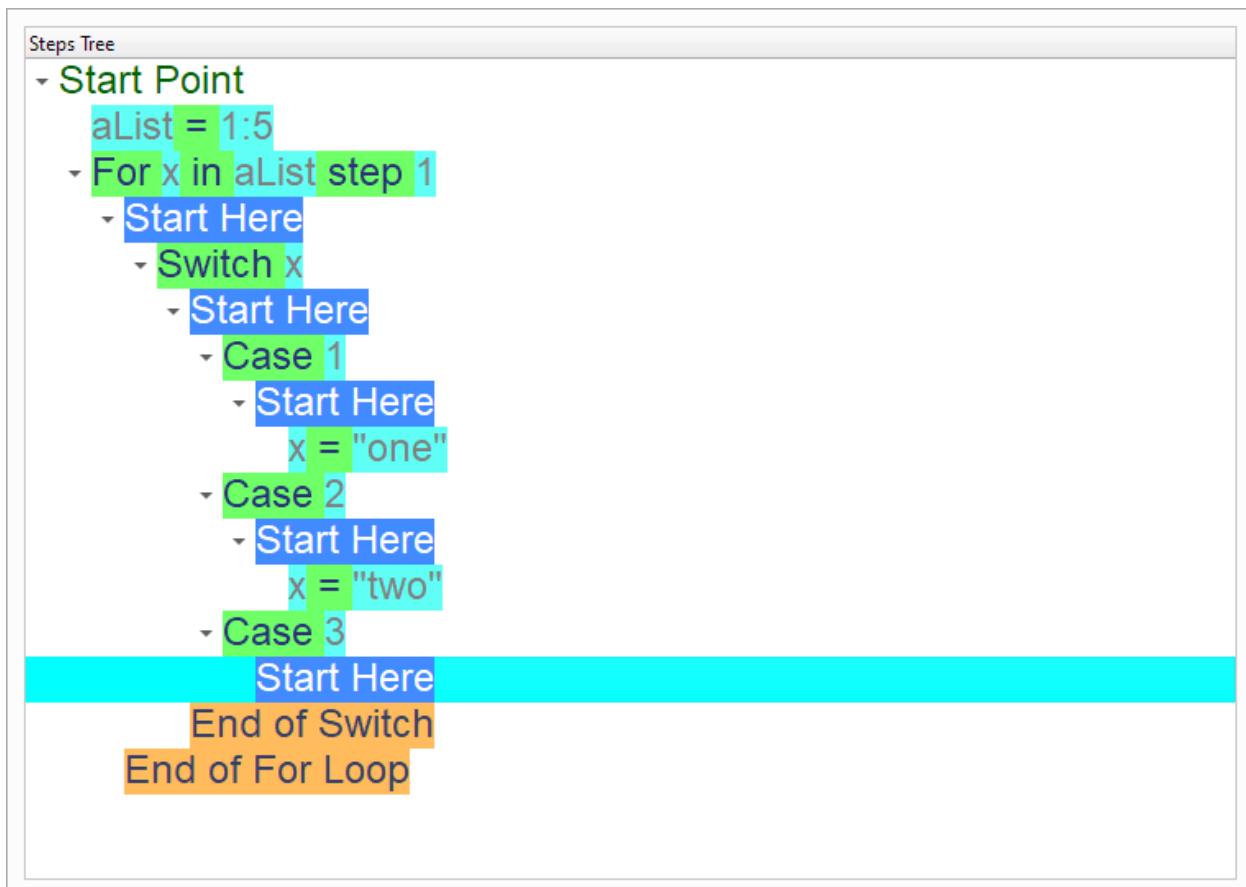
Steps Tree

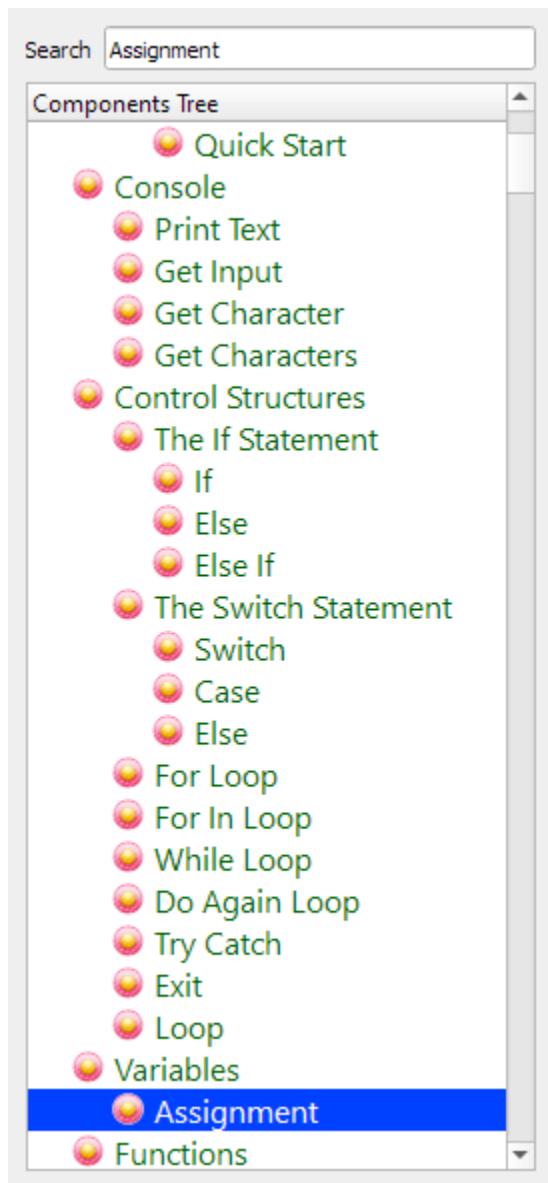
- Start Point
 - aList = 1:5
 - For x in aList step 1
 - Start Here
 - Switch x
 - Start Here
 - Case 1
 - Start Here
 - x = "one"
 - Case 2
 - Start Here
 - x = "two"
 - End of Switch
 - End of For Loop



if the item number is 3, set the item value to “three”







Assignment Component

Left Side :

"three"

Right Side :

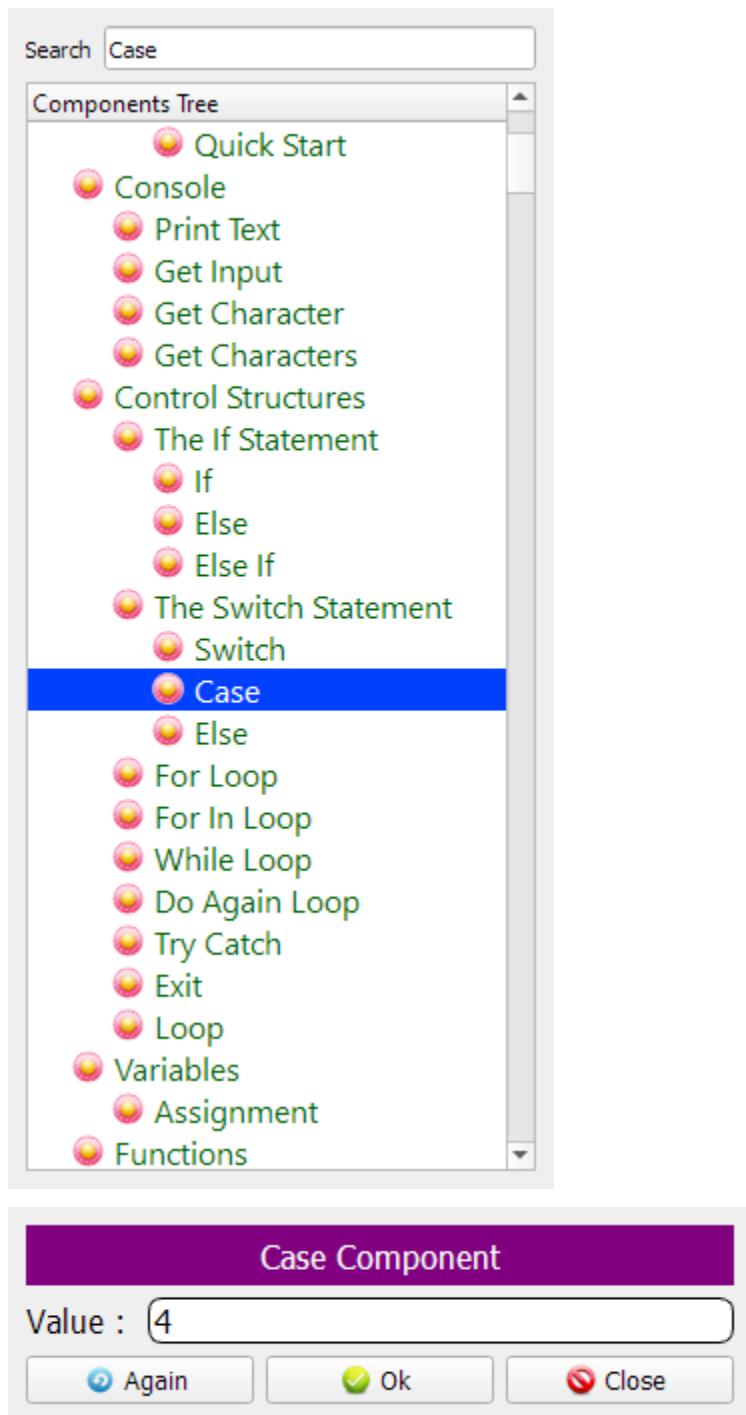
Steps Tree

- Start Point
 - aList = 1:5
 - For x in aList step 1
 - Start Here
 - Switch x
 - Start Here
 - Case 1
 - Start Here
 - x = "one"
 - Case 2
 - Start Here
 - x = "two"
 - Case 3
 - Start Here
 - x = "three"

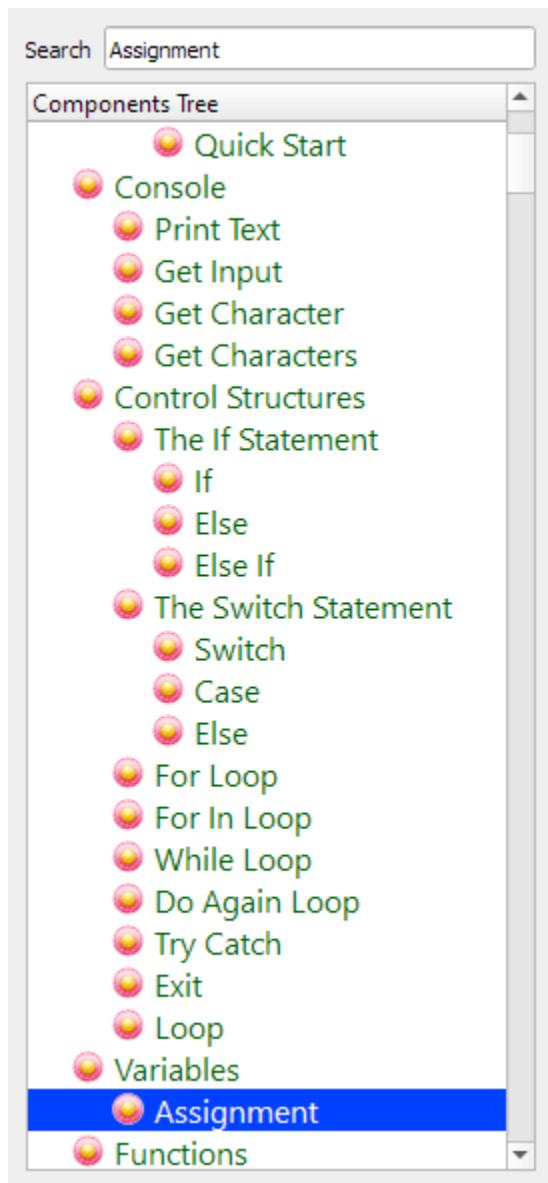
End of Switch

End of For Loop

if the item number is 4, set the item value to “four”







Assignment Component

Left Side :

"four"

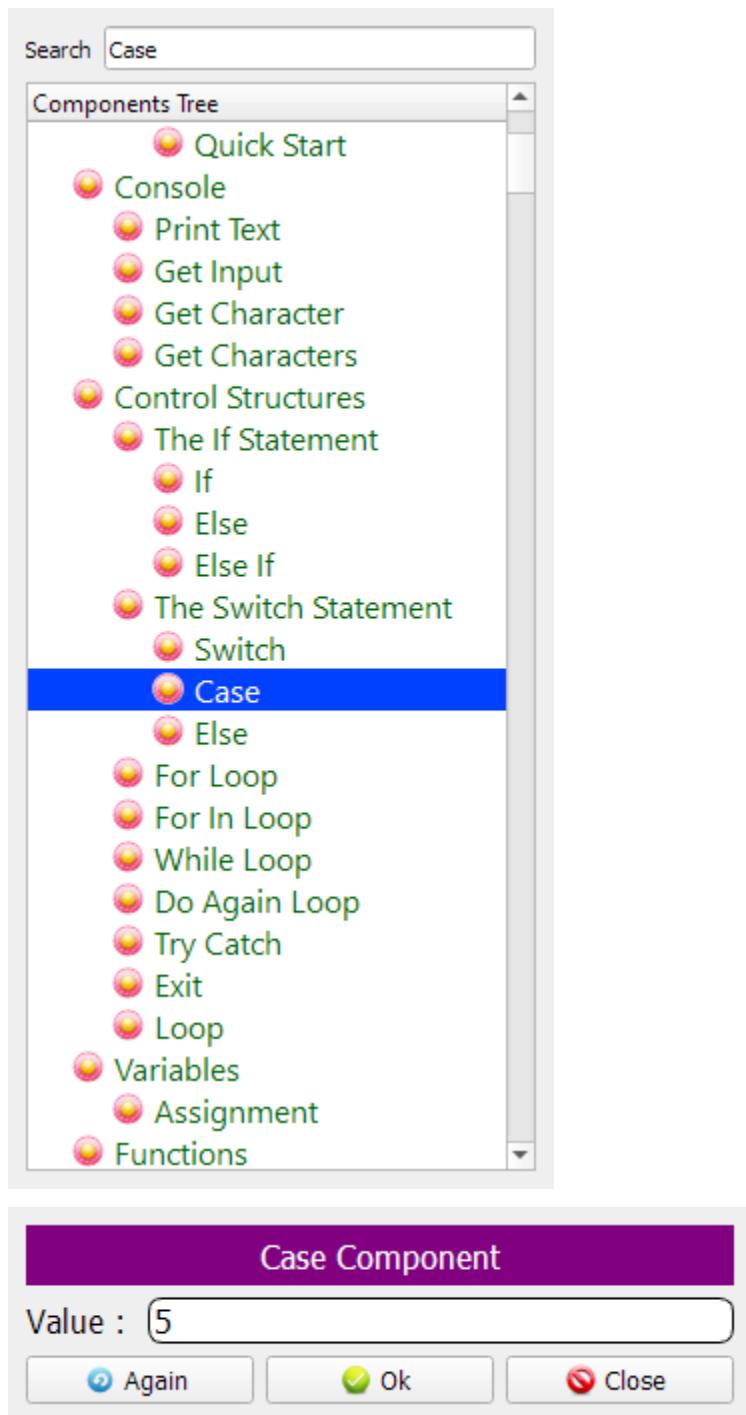
Right Side :

Steps Tree

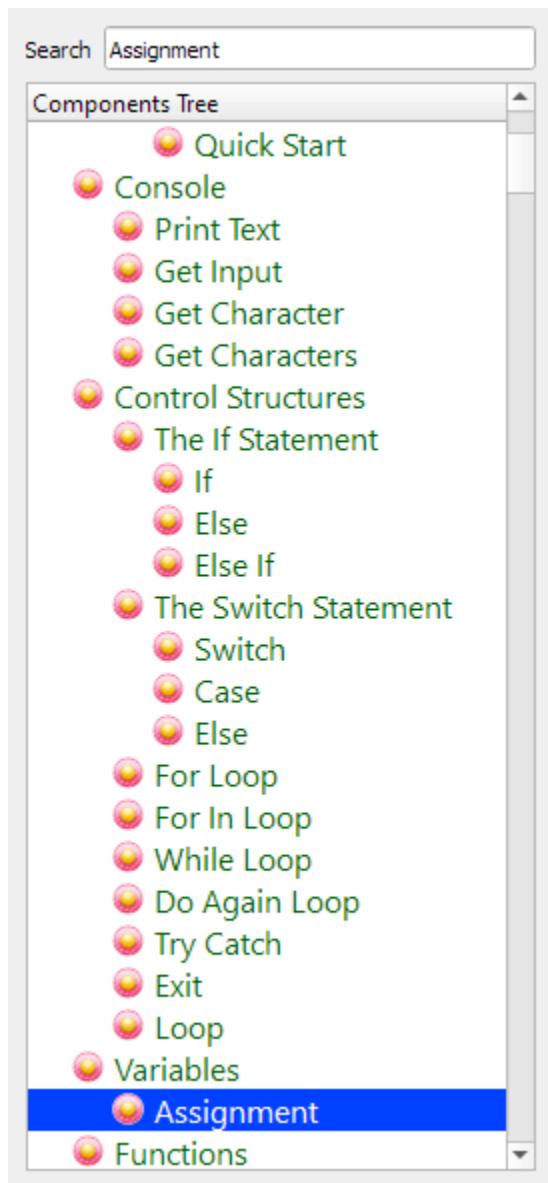
```

aList = 1:5
- For x in aList step 1
  - Start Here
  - Switch x
    - Start Here
      - Case 1
        - Start Here
          - x = "one"
      - Case 2
        - Start Here
          - x = "two"
      - Case 3
        - Start Here
          - x = "three"
      - Case 4
        - Start Here
          - x = "four"
    End of Switch
  
```

if the item number is 5, set the item value to “five”







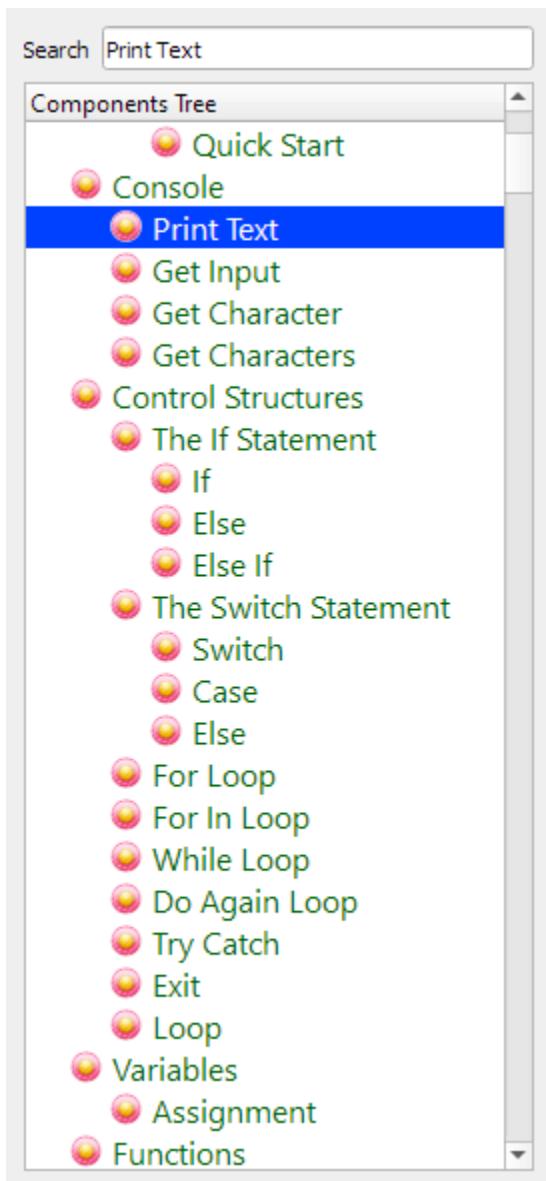
Assignment Component

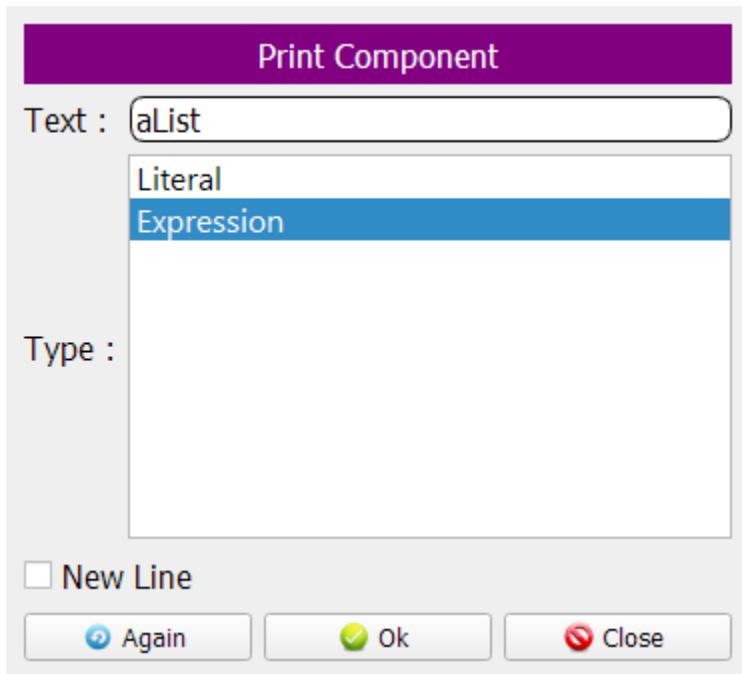
Left Side :

Right Side :

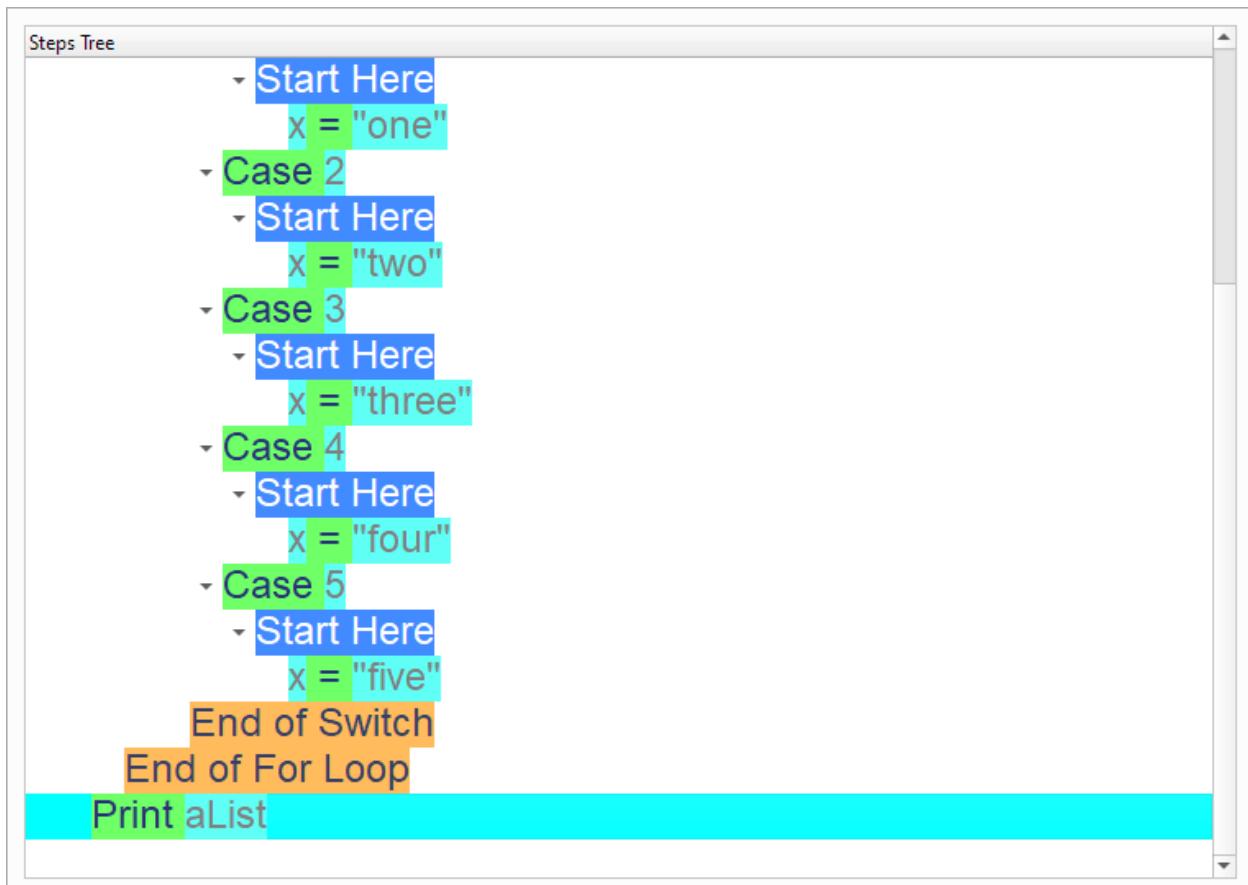
Steps Tree

- Case 1
 - Start Here
 - x = "one"
- Case 2
 - Start Here
 - x = "two"
- Case 3
 - Start Here
 - x = "three"
- Case 4
 - Start Here
 - x = "four"
- Case 5
 - Start Here
 - x = "five"
- End of Switch
- End of For Loop





Now we have the final Steps Tree in our program



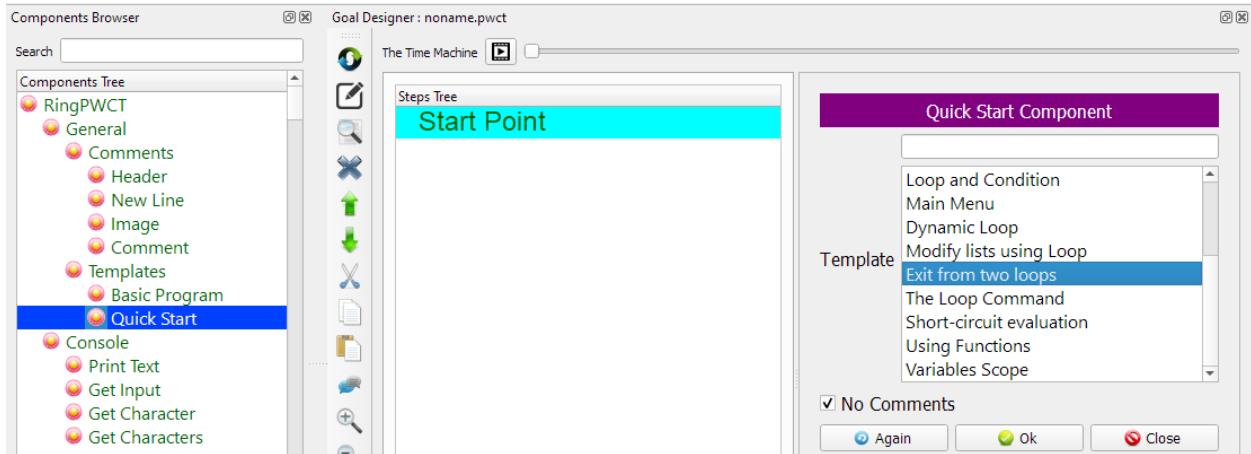
CHAPTER SEVENTEEN

EXIT FROM TWO LOOPS

In this chapter we are going to learn about the Exit from two loops

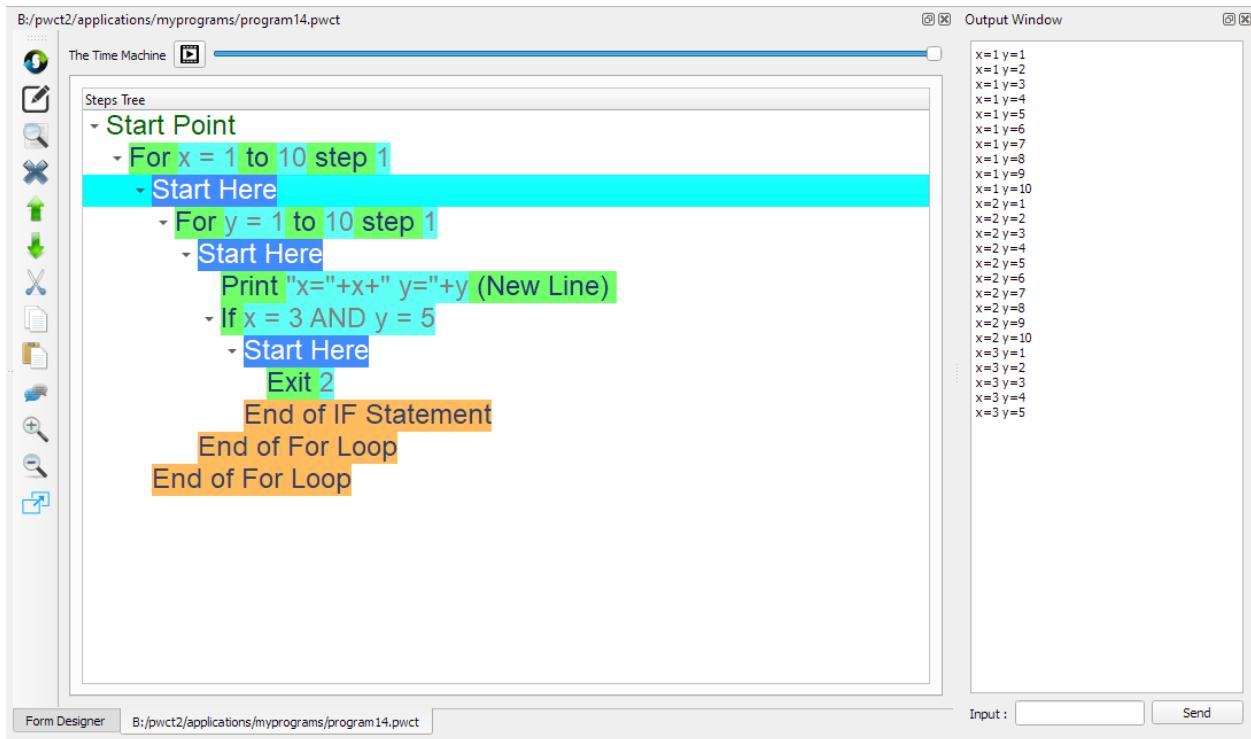
17.1 Introduction

We can create this program quickly using the Quick Start component



17.2 Program Steps

After selecting the (Exit from two loops) template, we will get the next steps in the Goal Designer

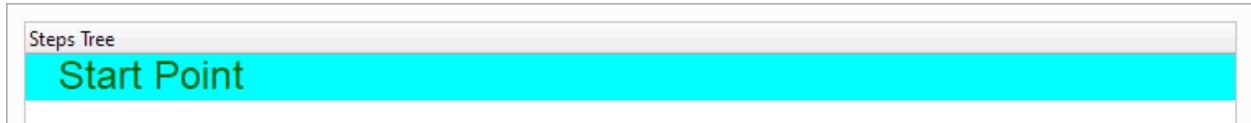


17.3 Creating the Program

To create this program we will use the next components

- For Loop
- Print Text
- If Statement
- Exit

In the begining the Steps Tree is empty



Select the (For Loop) component

Search For Loop

Components Tree

- General
 - Comments
 - Header
 - New Line
 - Image
 - Comment
 - Templates
 - Basic Program
 - Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop

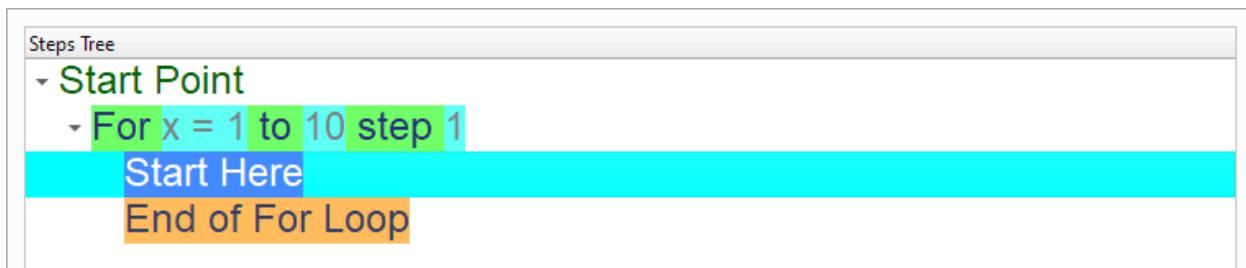
For Loop Component

Start :

To :

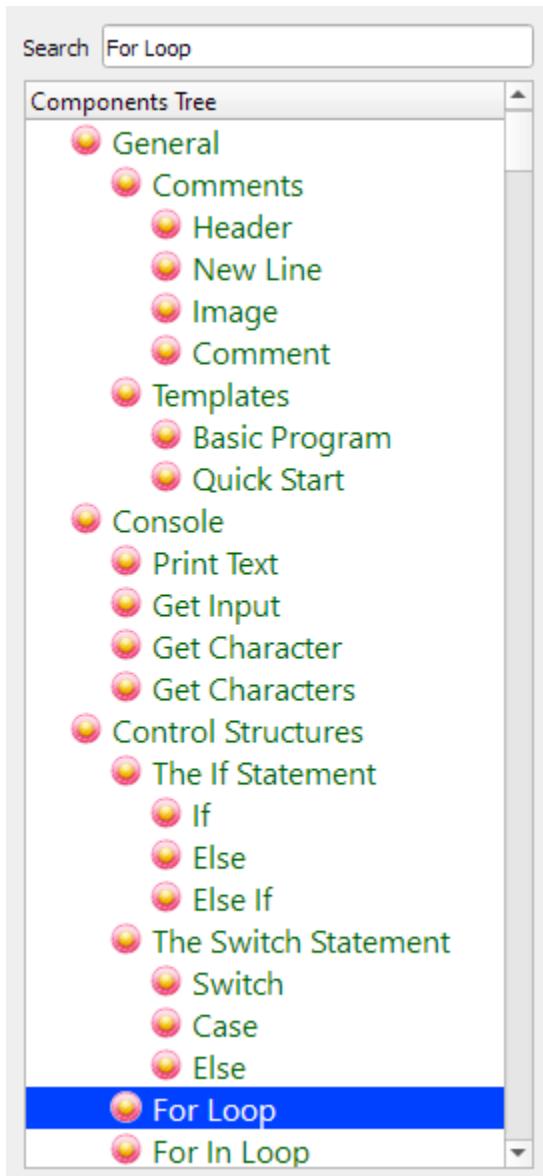
Step :

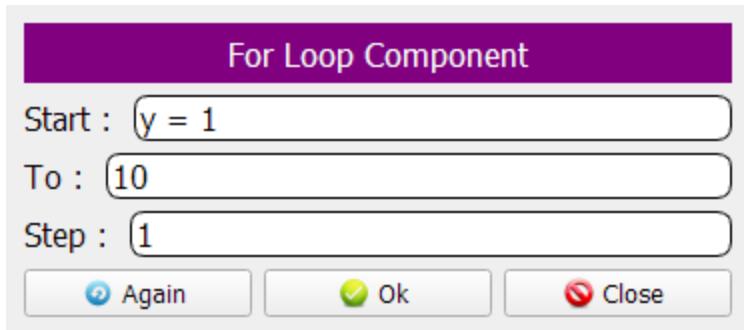
The Steps Tree will be updated



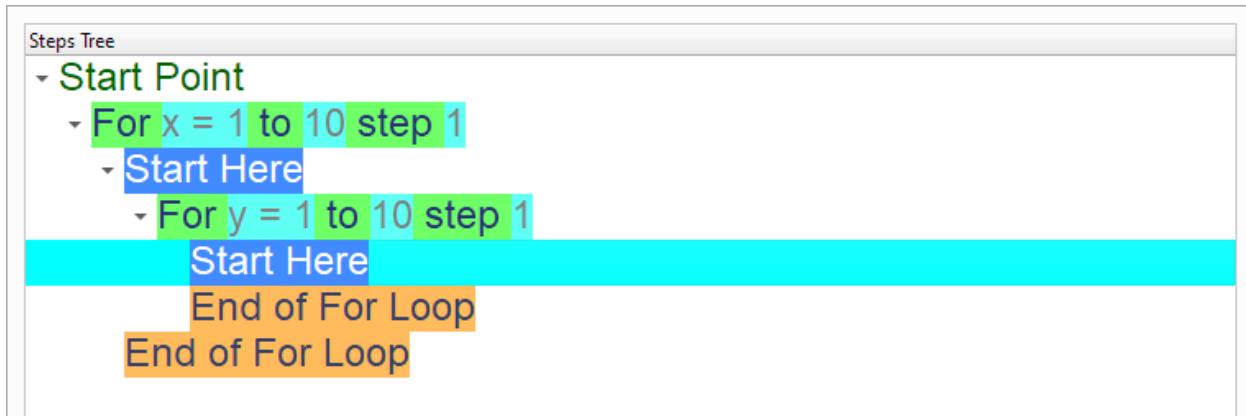
We will create an inner loop (i.e. for loop inside another for loop)

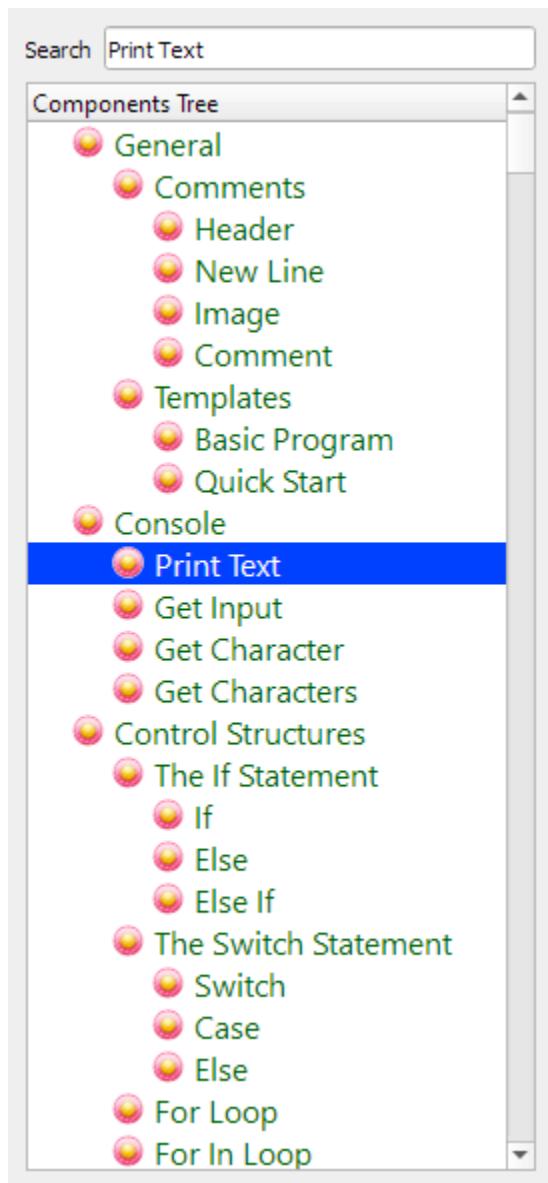
This time, The loop will use the (y) variable

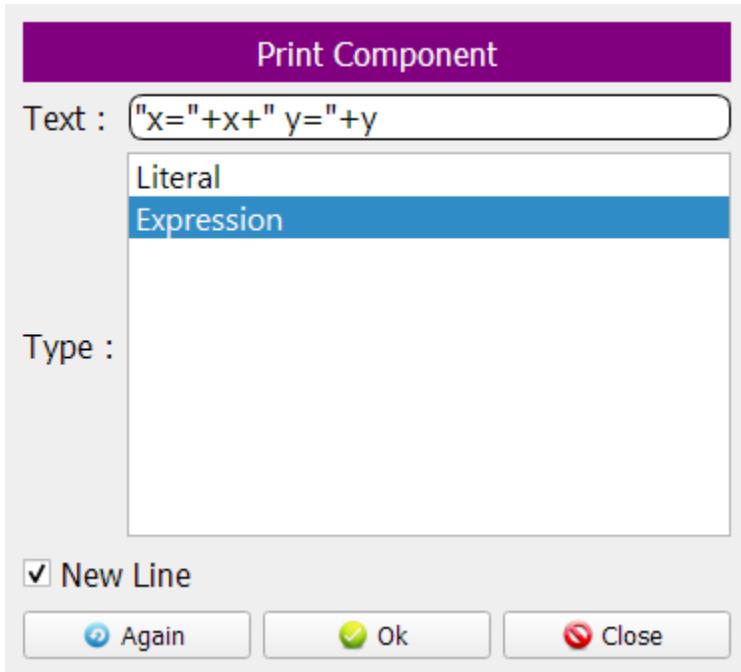




We will print the (X) and (Y) values using the (Print Text) component

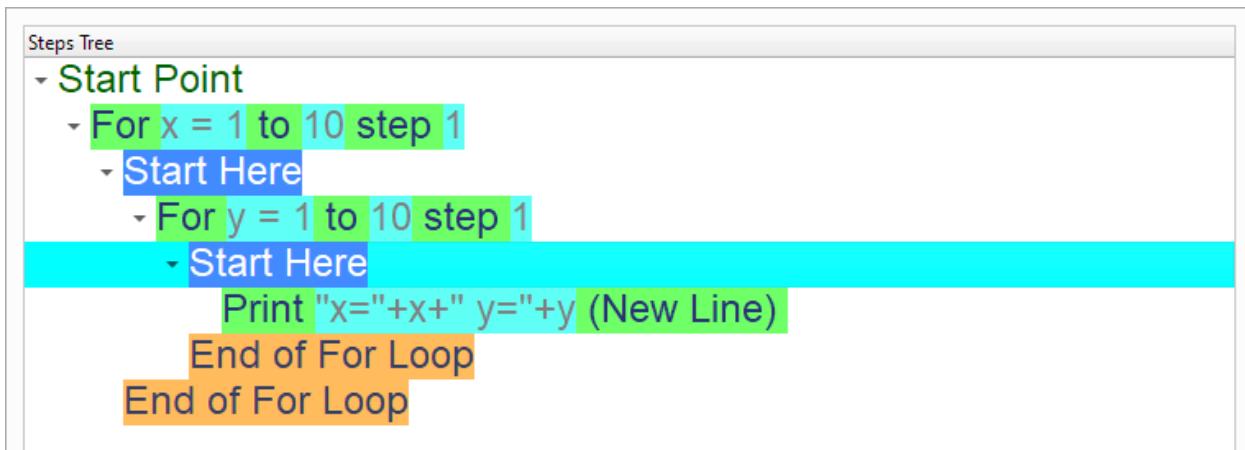






Using the (If Statement) component, we will check if $x=3$ and $y=5$

When this happens, We will end both loops using the (Exit) component



Search If

Components Tree

- General
 - Comments
 - Header
 - New Line
 - Image
 - Comment
 - Templates
 - Basic Program
 - Quick Start
 - Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
 - Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop

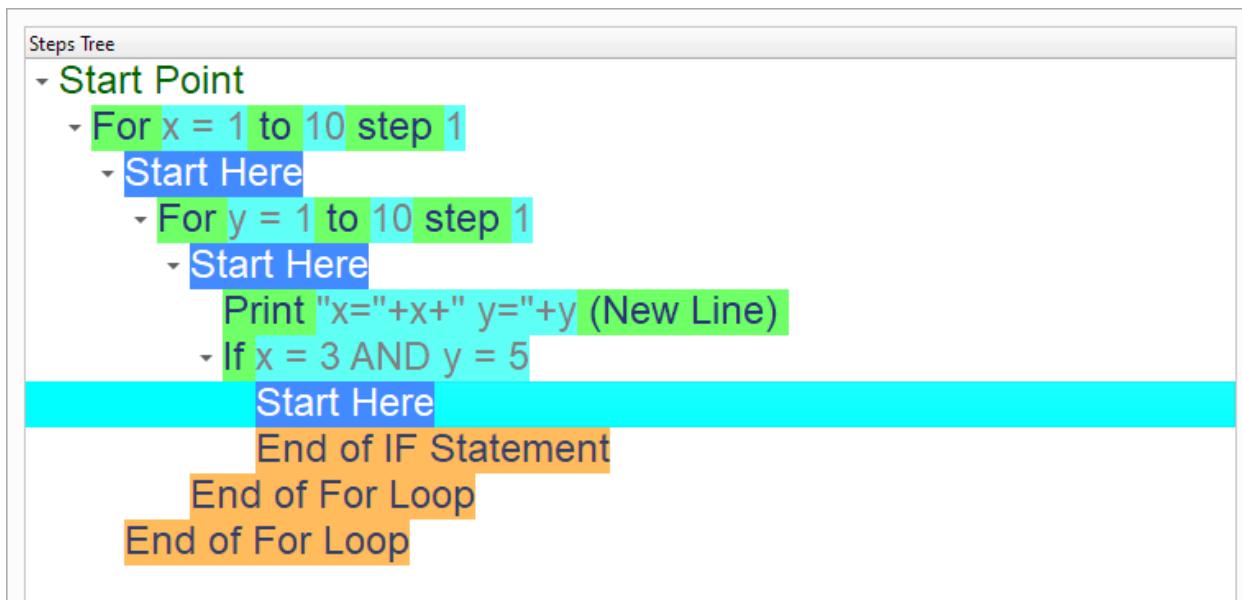
If Statement Component

Condition : $x = 3 \text{ AND } y = 5$

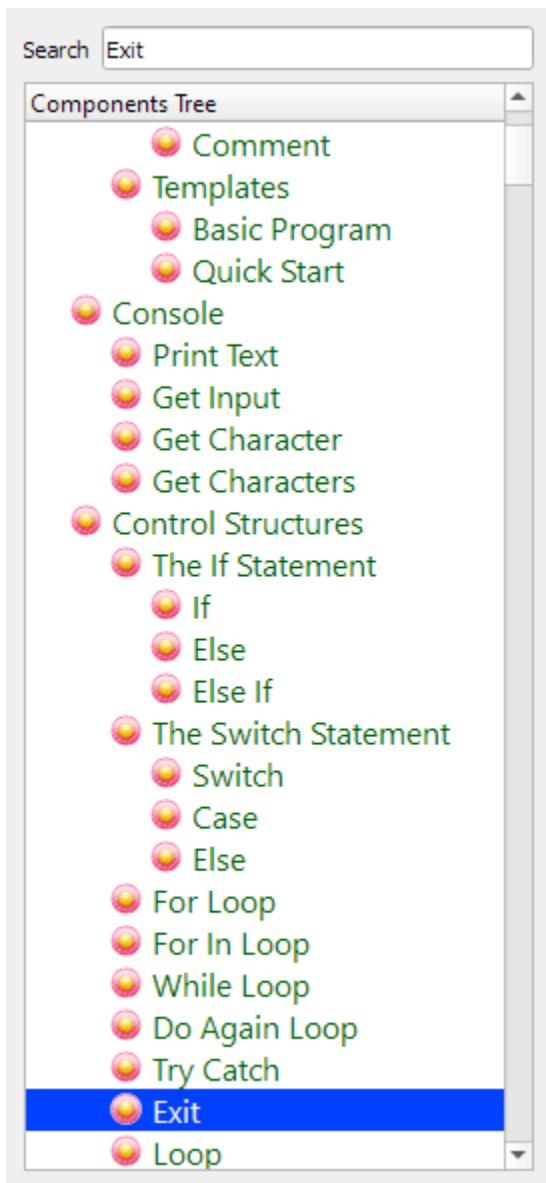
 Again

 Ok

 Close



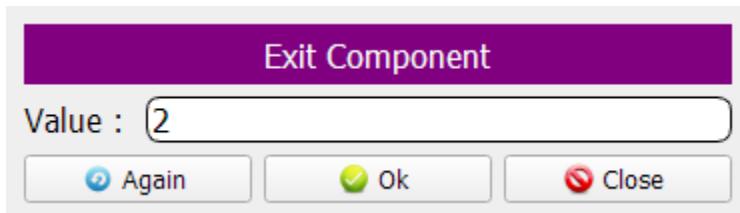
Select the (Exit) component



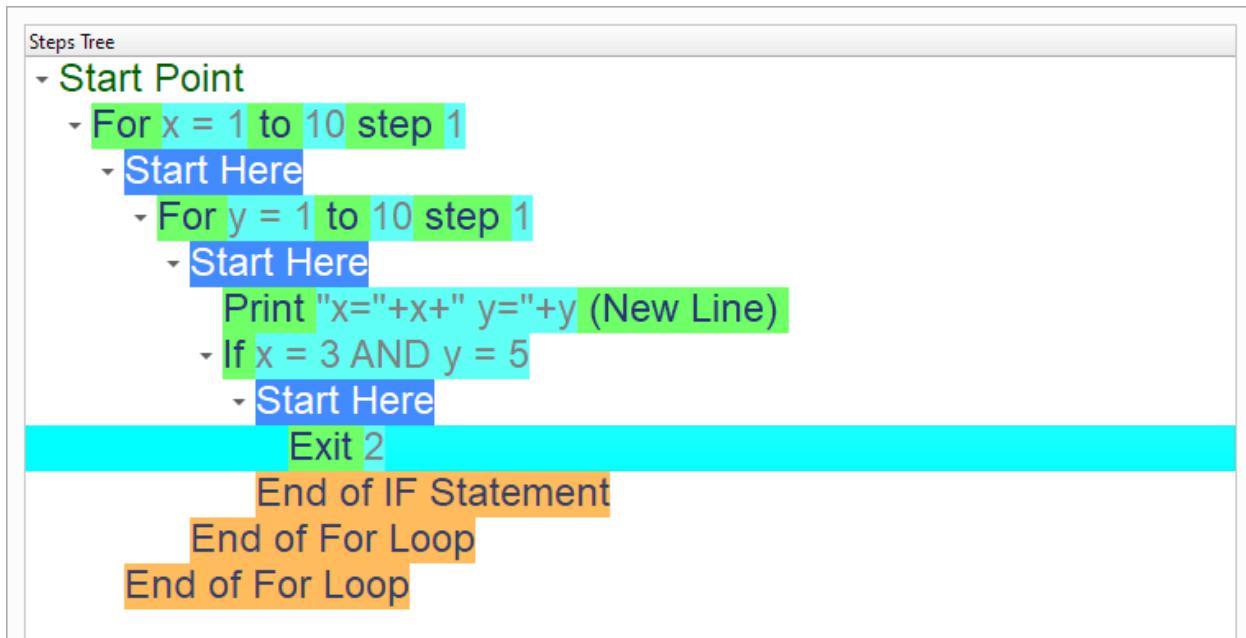
Enter the data in the Interaction Page

Value: 2

This means exit from two loops!



Now we have the final Steps Tree in our program



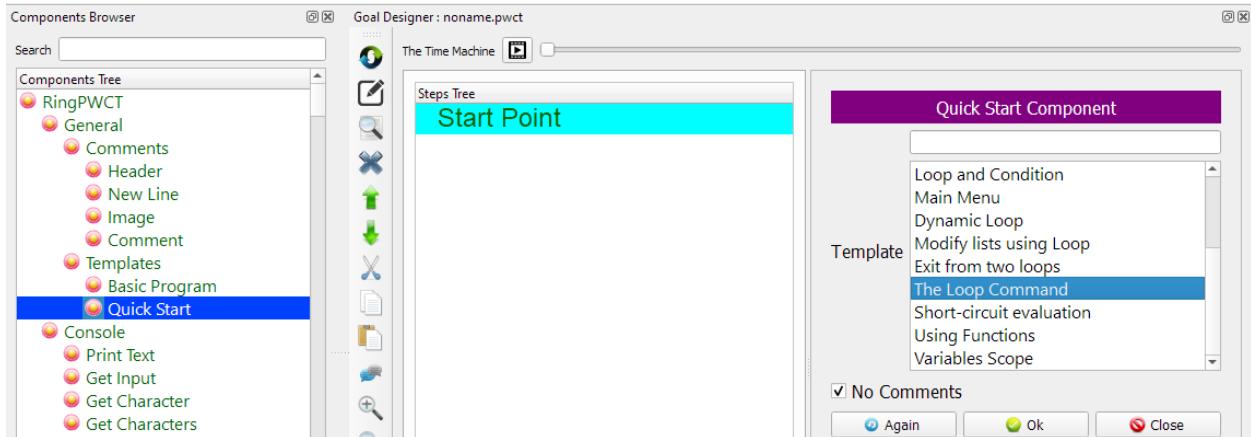
CHAPTER EIGHTEEN

THE LOOP COMMAND

In this chapter we are going to learn about the The Loop Command

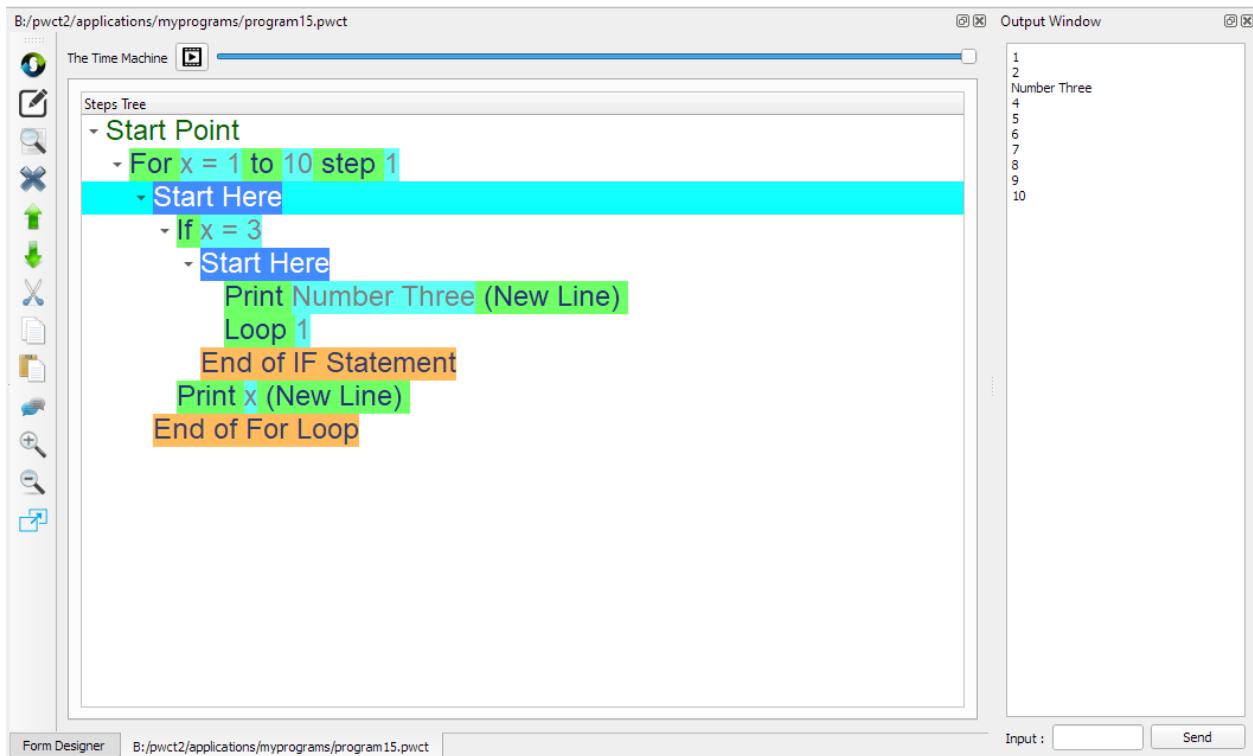
18.1 Introduction

We can create this program quickly using the Quick Start component



18.2 Program Steps

After selecting the (The Loop Command) template, we will get the next steps in the Goal Designer



18.3 Creating the Program

To create this program we will use the next components

- For Loop
- If Statement
- Print Text
- Loop

In the begining the Steps Tree is empty



Using the (For Loop) component we will create a loop from 1 to 10

The loop will use the X variable

Search For Loop

Components Tree

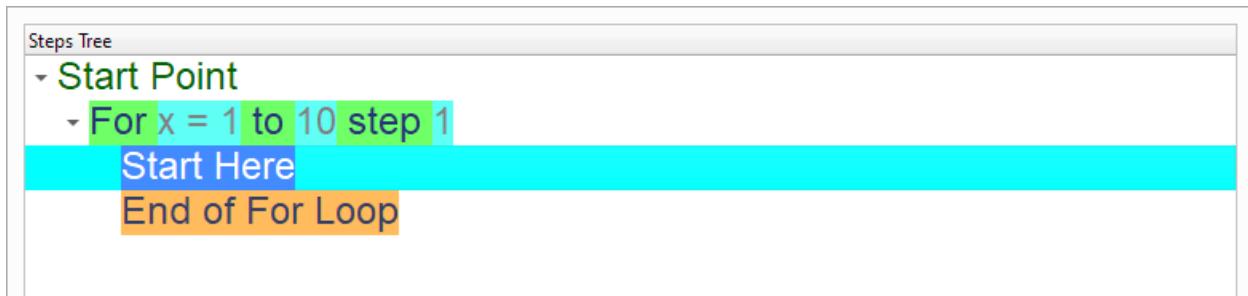
- General
 - Comments
 - Header
 - New Line
 - Image
 - Comment
 - Templates
 - Basic Program
 - Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop

For Loop Component

Start :

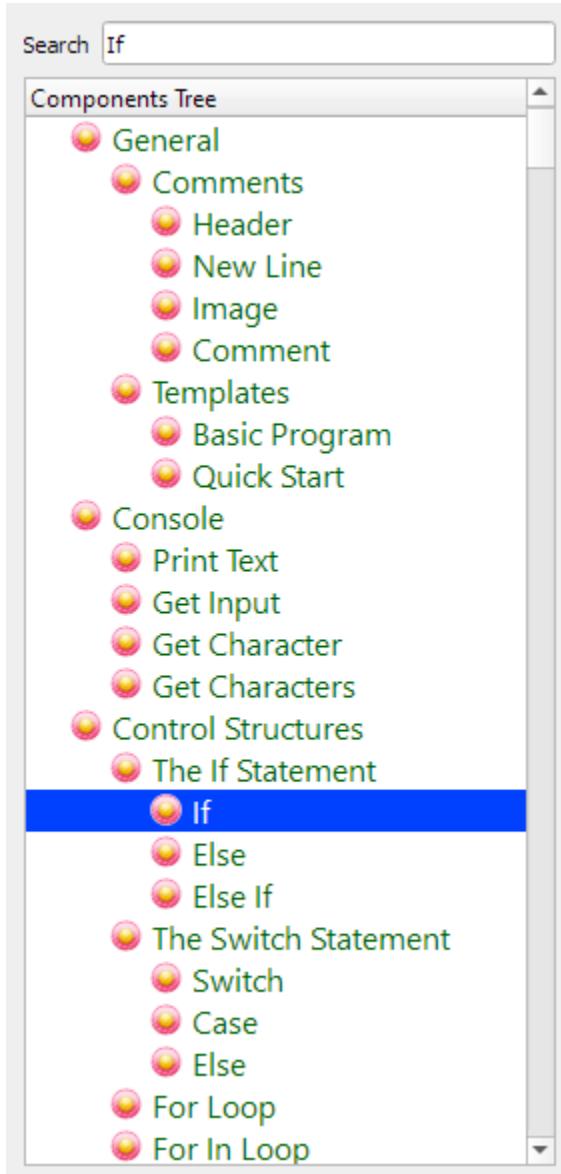
To :

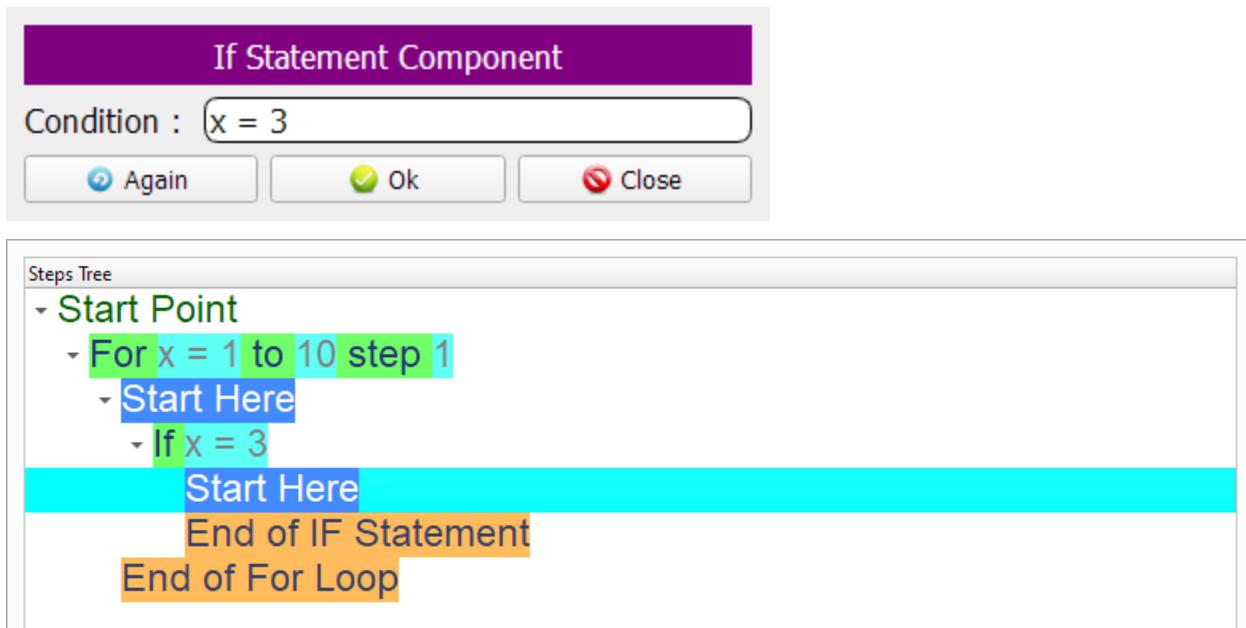
Step :

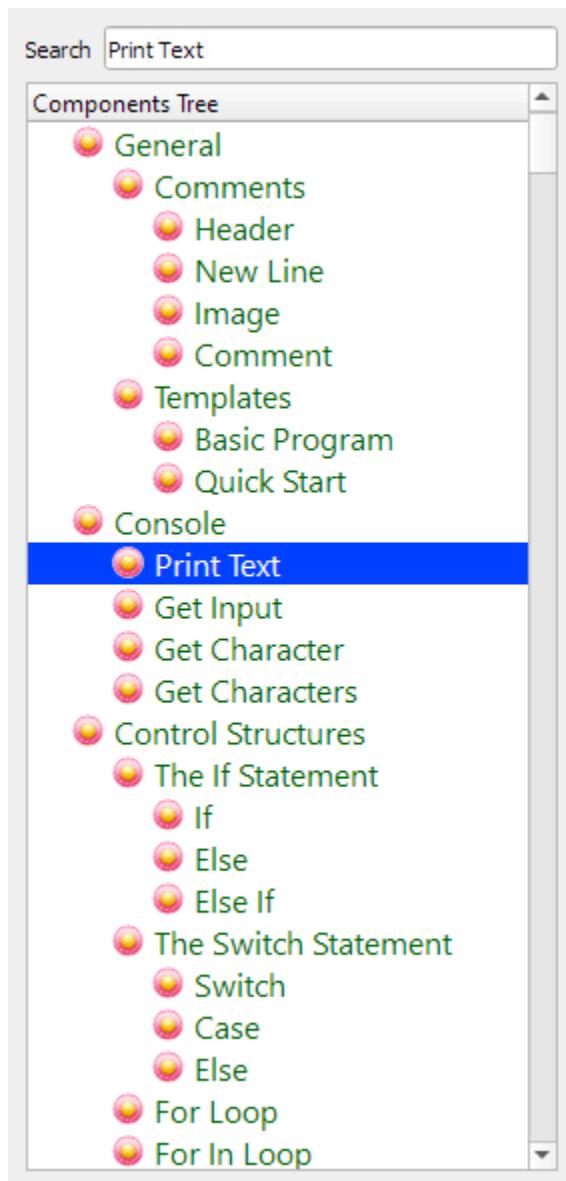


Using the (If Statement) component we will check if X=3

When this happens, we will print the (Three) message







The screenshot shows the PWCT software interface. At the top is a purple header bar with the text "Print Component". Below it is a dialog box titled "Text : Number Three". Inside the dialog, there are two tabs: "Literal" (selected) and "Expression". A "Type :" label is followed by a large empty text area. At the bottom of the dialog are three buttons: "Again" (with a circular arrow icon), "Ok" (with a checkmark icon), and "Close" (with a circular close icon).

Below the dialog is a "Steps Tree" window. It shows a hierarchical tree of steps:

- Start Point
 - For x = 1 to 10 step 1
 - Start Here
 - If x = 3
 - Start Here
 - Print Number Three (New Line)
 - End of IF Statement
 - End of For Loop

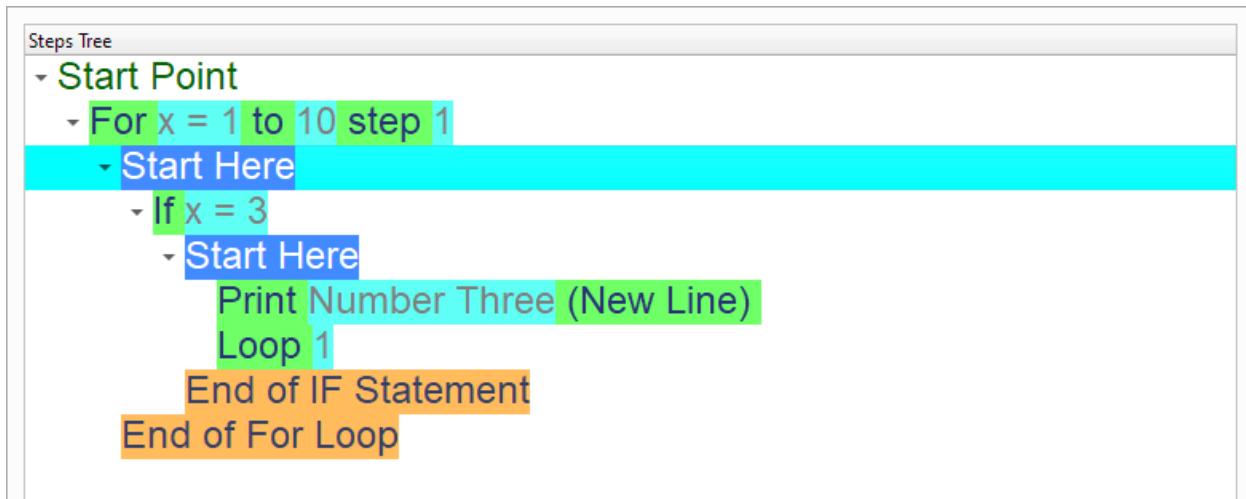
Using the (Loop) component, we will pass the next instructions in the loop
i.e. We will move to the next iteration in the loop

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar with the text "Loop". Below it is a tree view with the following structure:

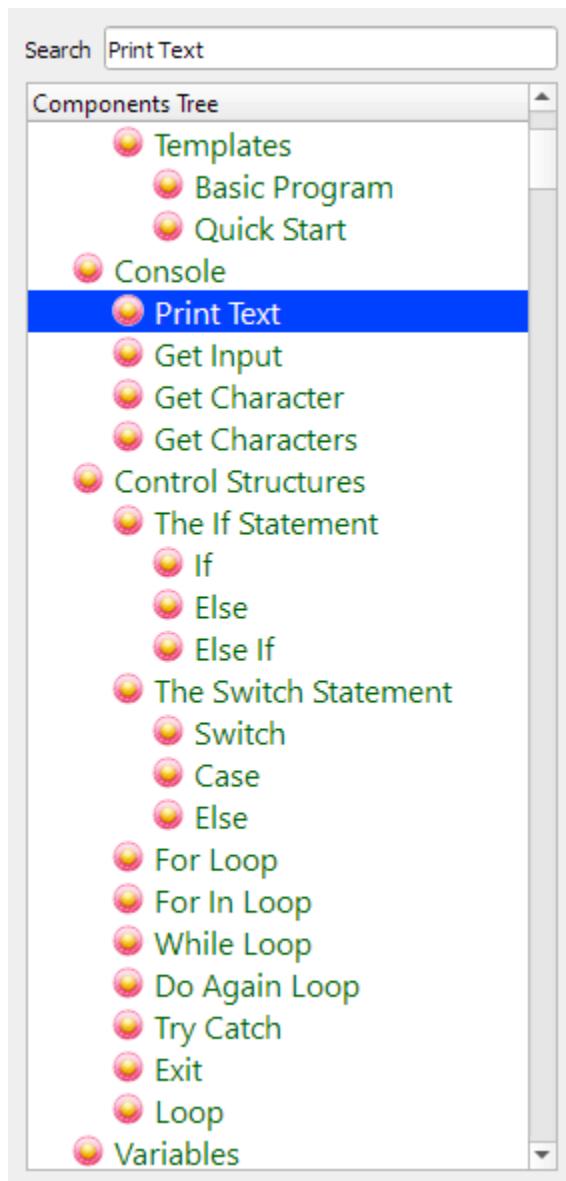
- Templates
- Basic Program
- Quick Start
- Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
- Loop
- Variables

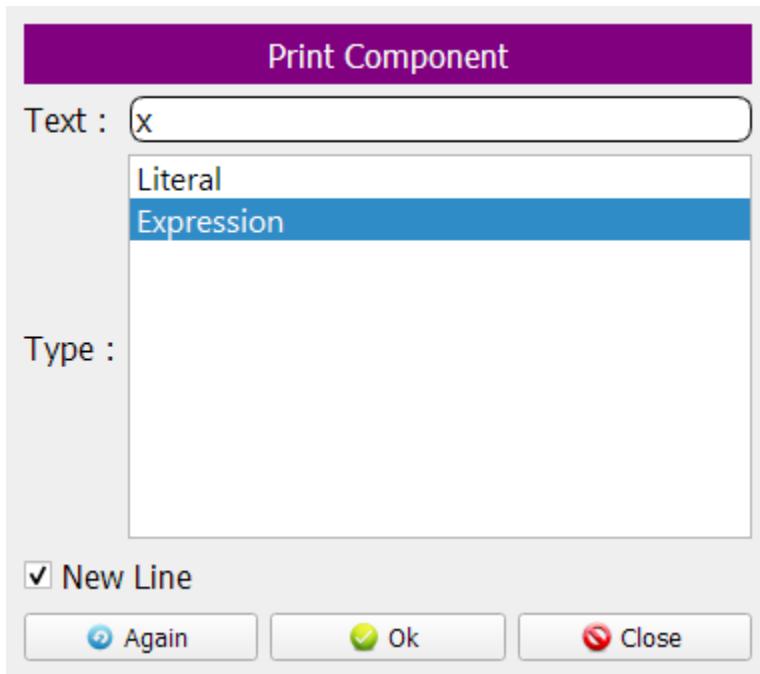
The "Loop" component is highlighted with a blue selection bar at the bottom of the tree.

Below the tree, a modal dialog box is open with the title "Loop Component". It contains a text input field labeled "Value : 1" and three buttons at the bottom: "Again" (with a circular arrow icon), "Ok" (with a checkmark icon), and "Close" (with a close/cross icon).

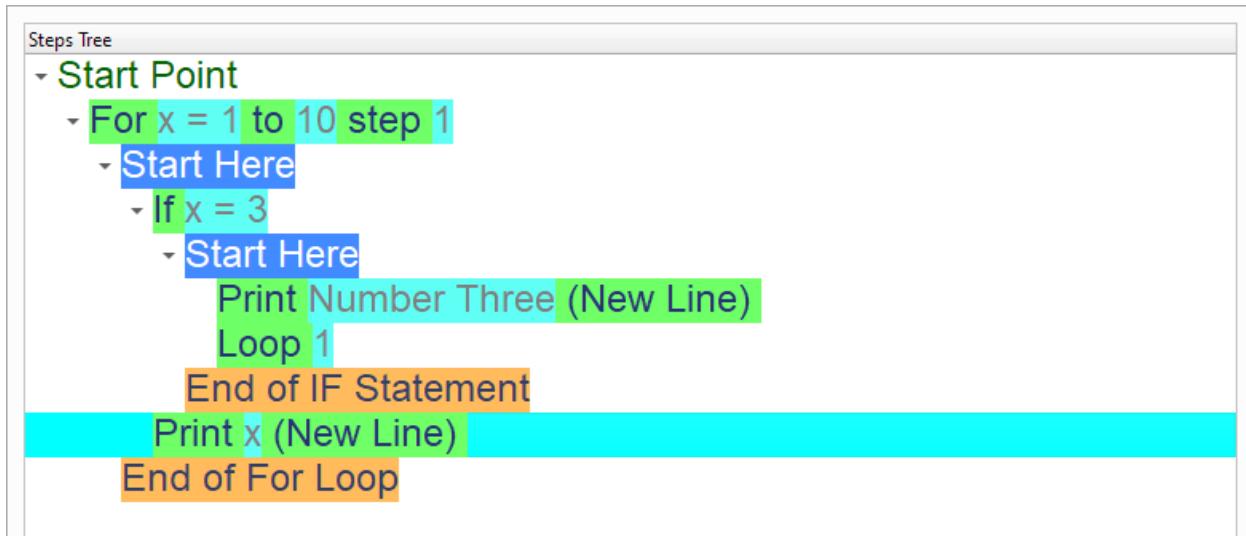


Using the (Print Text) component we will print the (X) variable





Now we have the final Steps Tree in our program



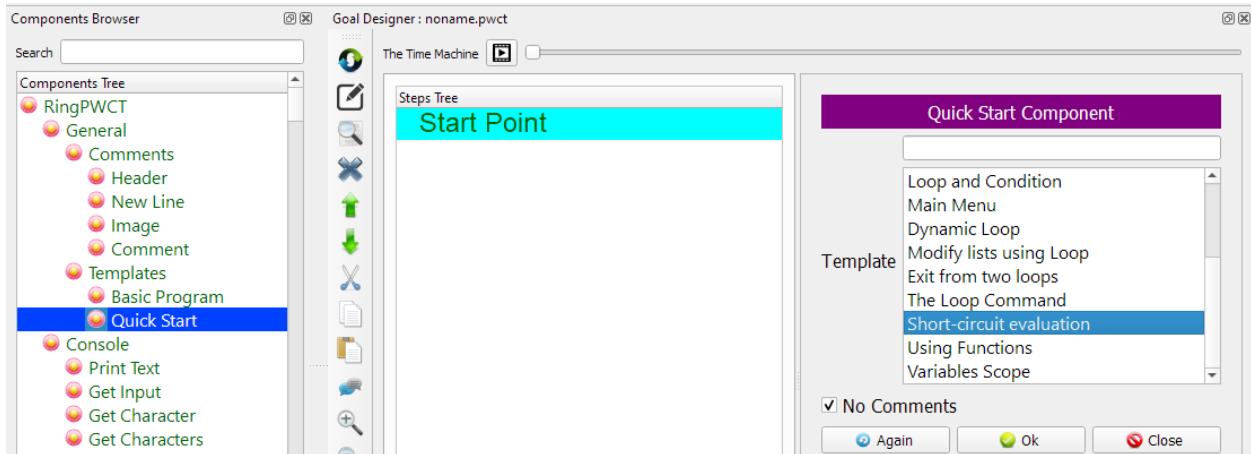
CHAPTER NINETEEN

SHORT CIRCUIT EVALUATION

In this chapter we are going to learn about the Short Circuit Evaluation

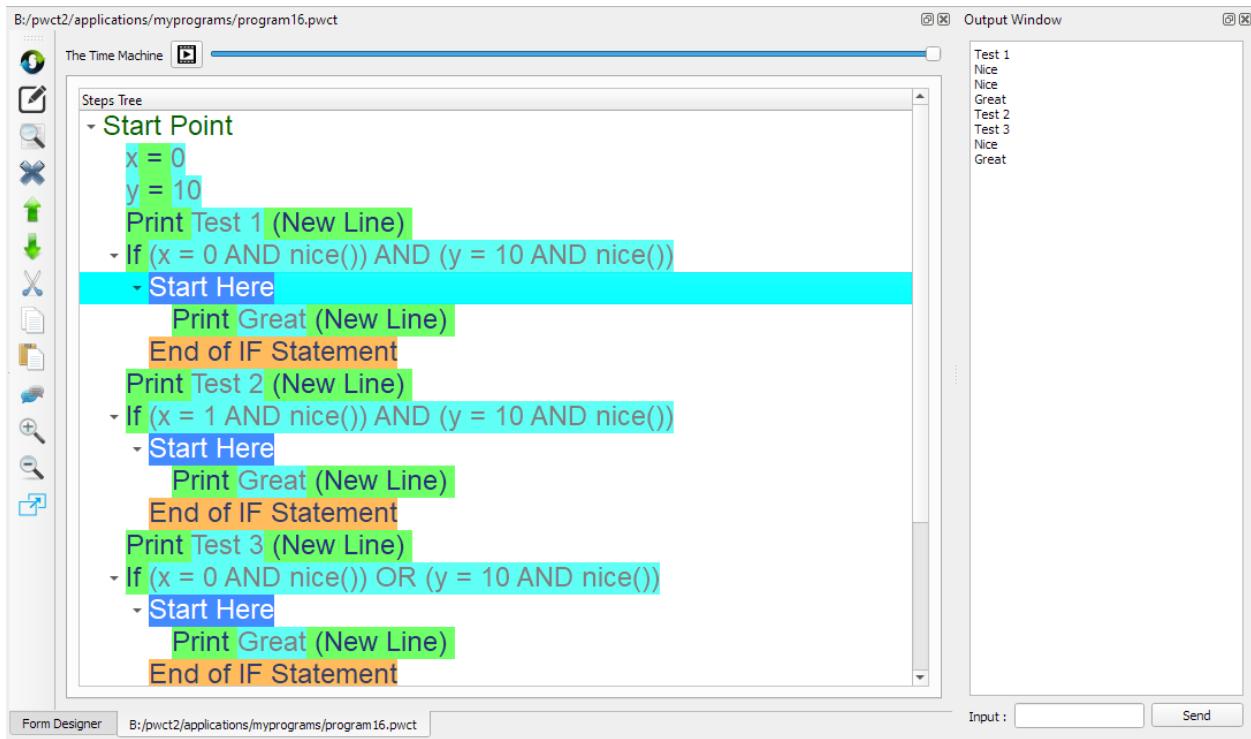
19.1 Introduction

We can create this program quickly using the Quick Start component



19.2 Program Steps

After selecting the (Short Circuit Evaluation) template, we will get the next steps in the Goal Designer



The Steps Tree:

```
x = 0
y = 10
Print Test 1 (New Line)
If (x = 0 AND nice()) AND (y = 10 AND nice())
    Print Great (New Line)
End of IF Statement
Print Test 2 (New Line)
If (x = 1 AND nice()) AND (y = 10 AND nice())
    Print Great (New Line)
End of IF Statement
Print Test 3 (New Line)
If (x = 0 AND nice()) OR (y = 10 AND nice())
    Print Great (New Line)
End of IF Statement
function nice
    Print Nice (New Line)
    Return 1
End of Function
```

19.3 Creating the Program

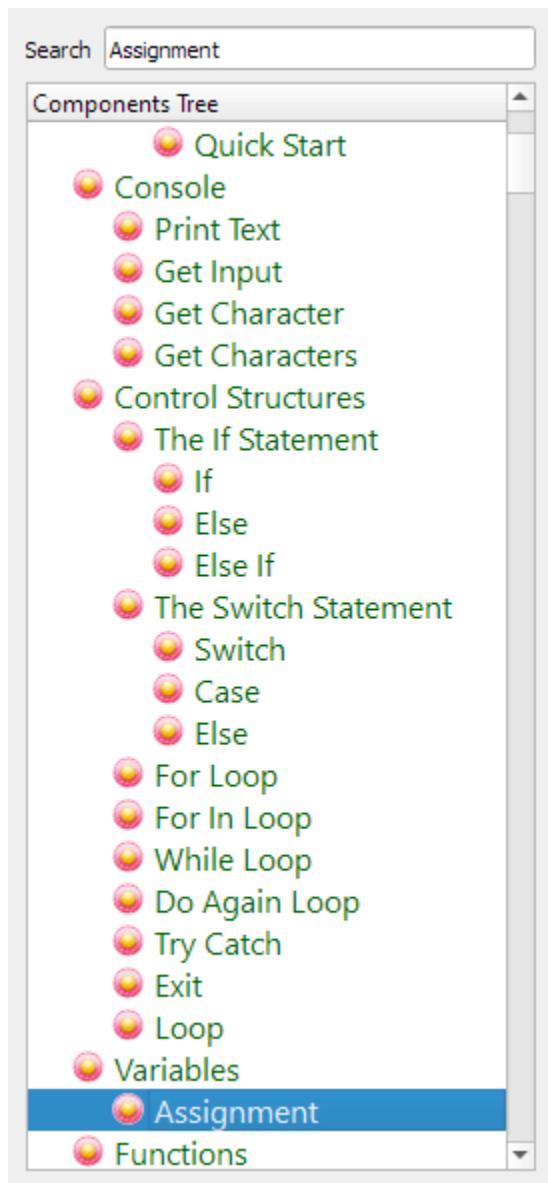
To create this program we will use the next components

- Assignment
- Print Text
- If Statement
- Define Function
- Return

In the begining the Steps Tree is empty



Set $x = 0$ using the Assignment component



The screenshot shows two windows related to the Assignment component.

Assignment Component Dialog:

- Left Side :** A text input field containing "x".
- Right Side :** A text input field containing "0".
- Buttons:** "Again" (with a question mark icon), "Ok" (with a checkmark icon), and "Close" (with a circular error icon).

Steps Tree:

- Start Point:** A node in the tree labeled "x = 0".

Set $y = 10$ using the Assignment component



Assignment Component

Left Side :

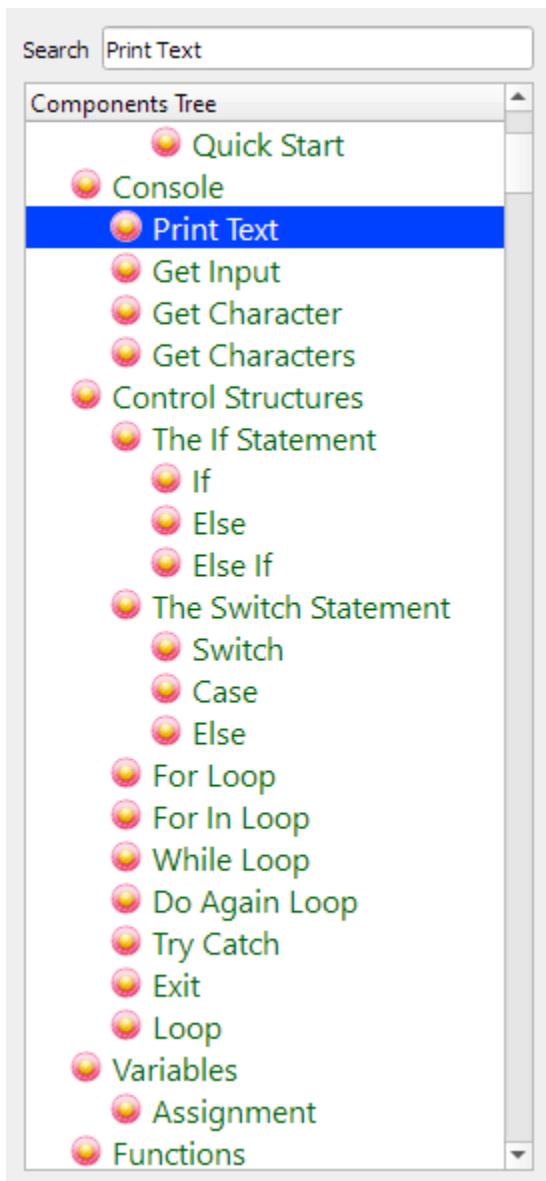
Right Side :

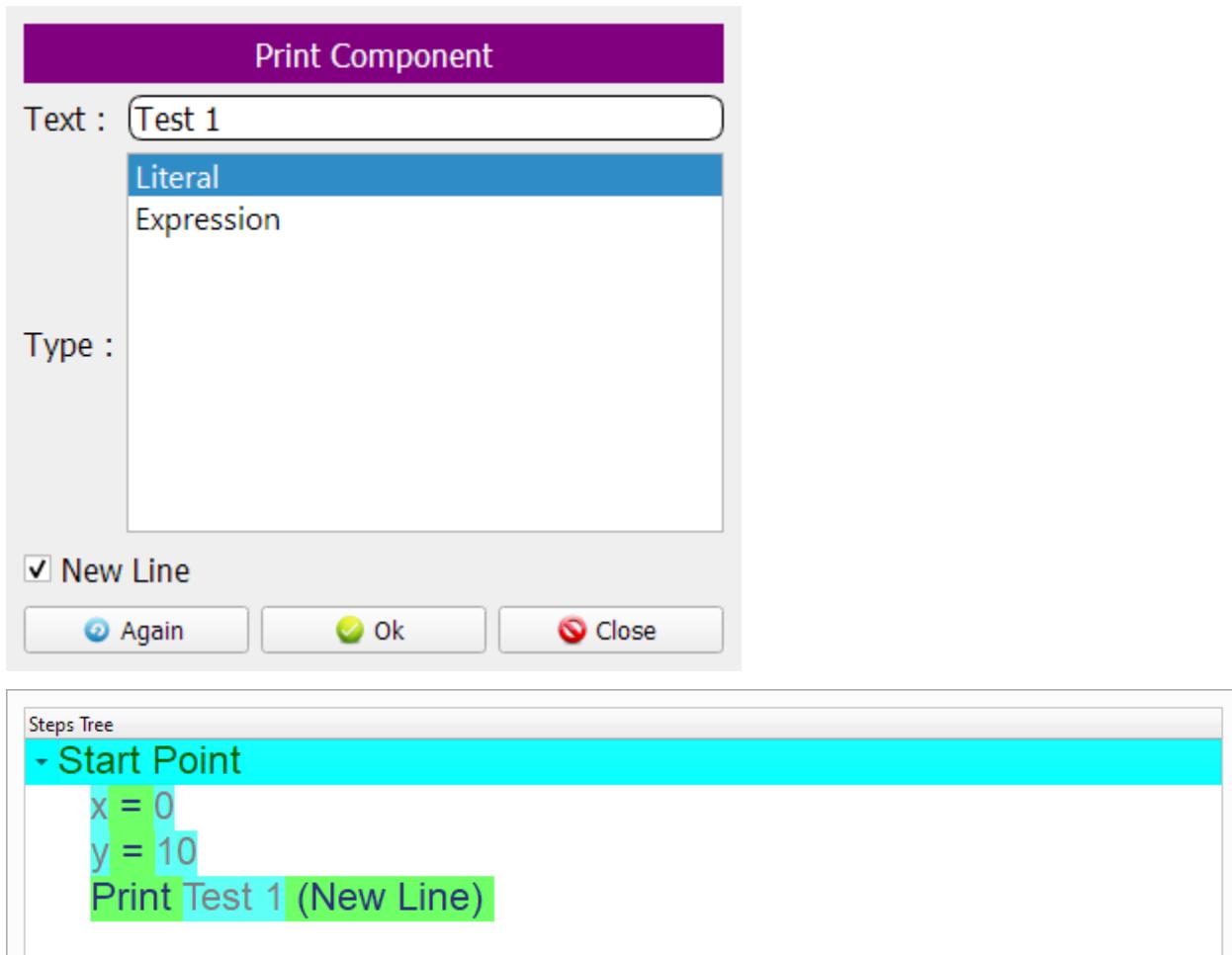
Steps Tree

- Start Point

x = 0
y = 10

Print (Test 1)

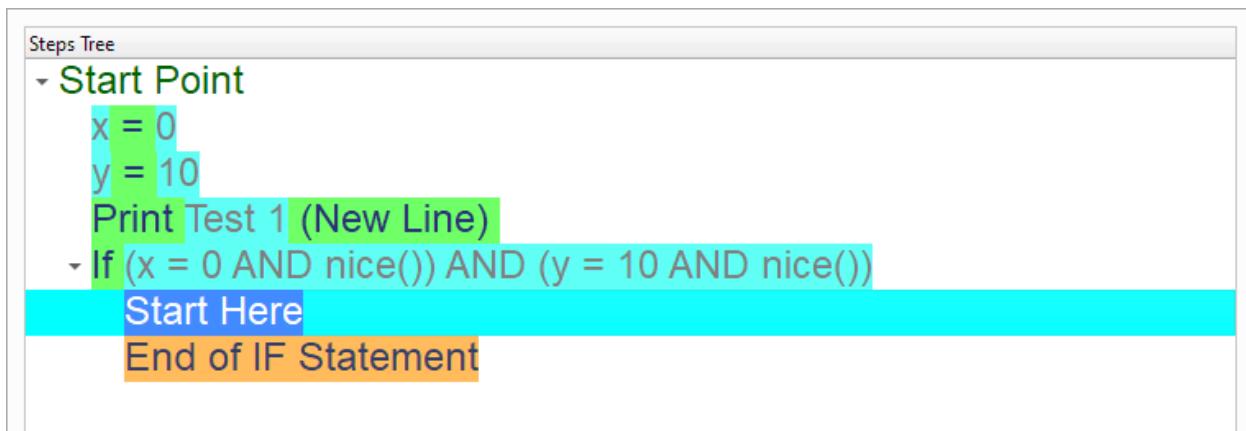




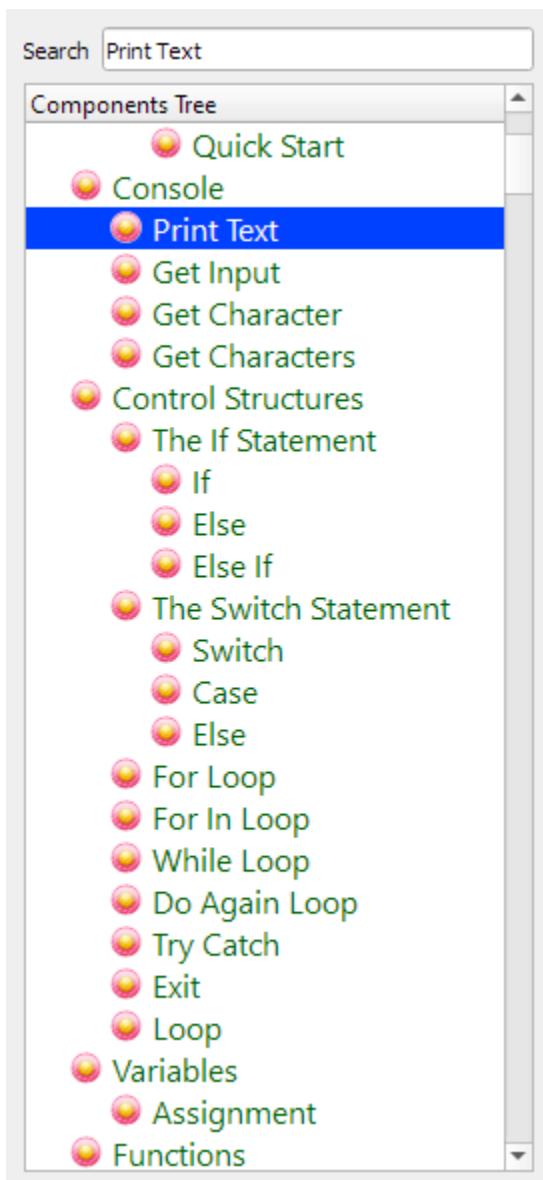
The screenshot shows the PWCT Components Tree interface. At the top left is a search bar with the text "Search If". Below it is a tree view with the following structure:

- Components Tree
 - Quick Start
 - Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
 - Control Structures
 - The If Statement
 - If (selected)
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
- Assignment
- Functions

A blue horizontal bar highlights the "If" node. Below the tree is a modal dialog titled "If Statement Component". It contains a condition field with the expression `(x = 0 AND nice()) AND (y = 10 AND nice())`. At the bottom are three buttons: "Again" (with a circular arrow icon), "Ok" (with a checkmark icon), and "Close" (with a red circle and cross icon).



Print (Great)



Print Component

Text : Great

Type :

Literal
Expression

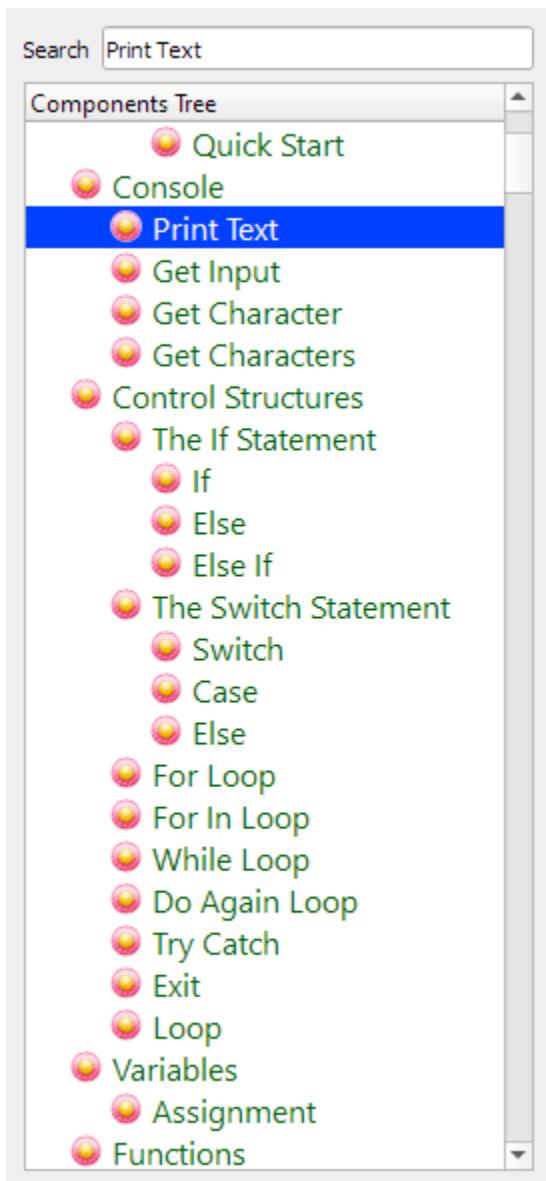
New Line

 Again  Ok  Close

Steps Tree

```
- Start Point
  x = 0
  y = 10
  Print Test 1 (New Line)
- If (x = 0 AND nice()) AND (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
  End of IF Statement
```

Print (Test 2)



Print Component

Text : Test 2

Type :

Literal
Expression

New Line

Steps Tree

- Start Point
 - x = 0
 - y = 10
 - Print Test 1 (New Line)
- If (x = 0 AND nice()) AND (y = 10 AND nice())
 - Start Here
 - Print Great (New Line)
 - End of IF Statement
 - Print Test 2 (New Line)

Check the condition: (x = 1 AND nice()) AND (y = 10 AND nice())

The screenshot shows the PWCT Components Tree interface. At the top left is a search bar with the text "Search If". Below it is a tree view with the following structure:

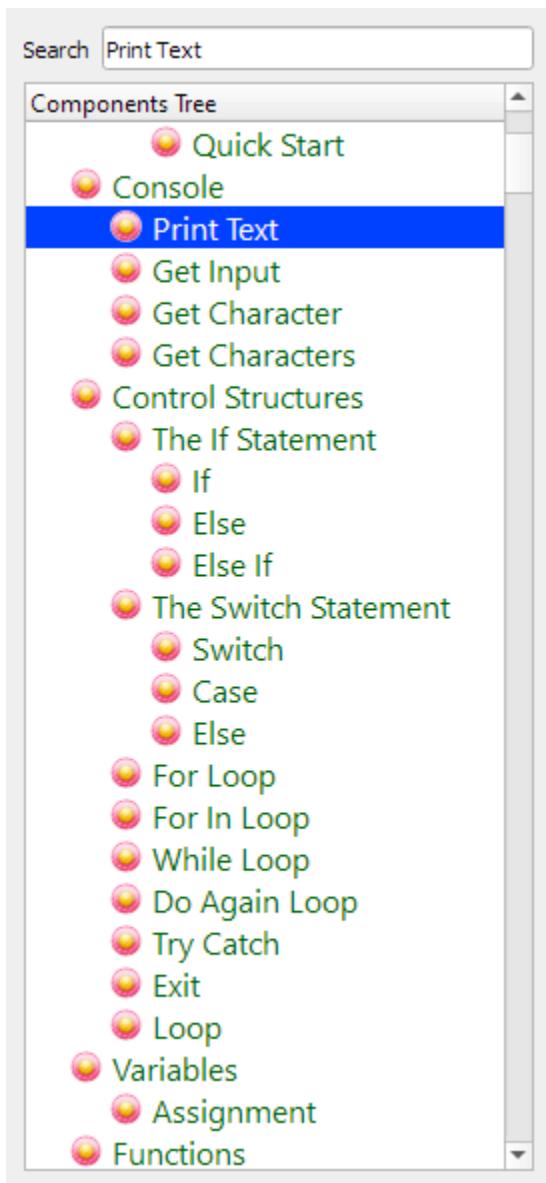
- Components Tree
 - Quick Start
 - Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
 - Control Structures
 - The If Statement
 - If (selected)
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
 - Variables
 - Assignment
 - Functions

Below the tree is a modal dialog titled "If Statement Component". It contains the following text:
Condition : `(x = 1 AND nice()) AND (y = 10 AND nice())`

At the bottom of the dialog are three buttons: "Again" (with a circular arrow icon), "Ok" (with a checkmark icon), and "Close" (with a red circle and cross icon).



Print (Great)



Print Component

Text : Great

Type :

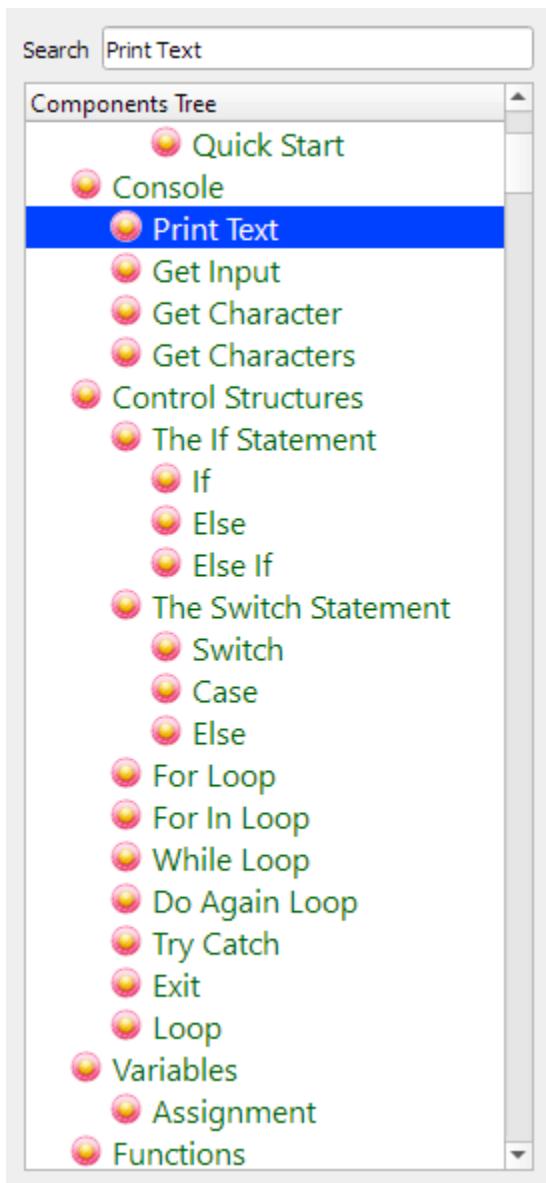
Literal
Expression

New Line

Steps Tree

- Start Point
 - x = 0
 - y = 10
 - Print Test 1 (New Line)
- If (x = 0 AND nice()) AND (y = 10 AND nice())
 - Start Here
 - Print Great (New Line)
 - End of IF Statement
 - Print Test 2 (New Line)
- If (x = 1 AND nice()) AND (y = 10 AND nice())
 - Start Here
 - Print Great (New Line)
 - End of IF Statement

Print (Test 3)



Print Component

Text :

Type :

Literal
Expression

New Line

Steps Tree

```

- Start Point
  - x = 0
  - y = 10
  - Print Test 1 (New Line)
- If (x = 0 AND nice()) AND (y = 10 AND nice())
  - Start Here
    - Print Great (New Line)
    - End of IF Statement
    - Print Test 2 (New Line)
- If (x = 1 AND nice()) AND (y = 10 AND nice())
  - Start Here
    - Print Great (New Line)
    - End of IF Statement
  - Print Test 3 (New Line)

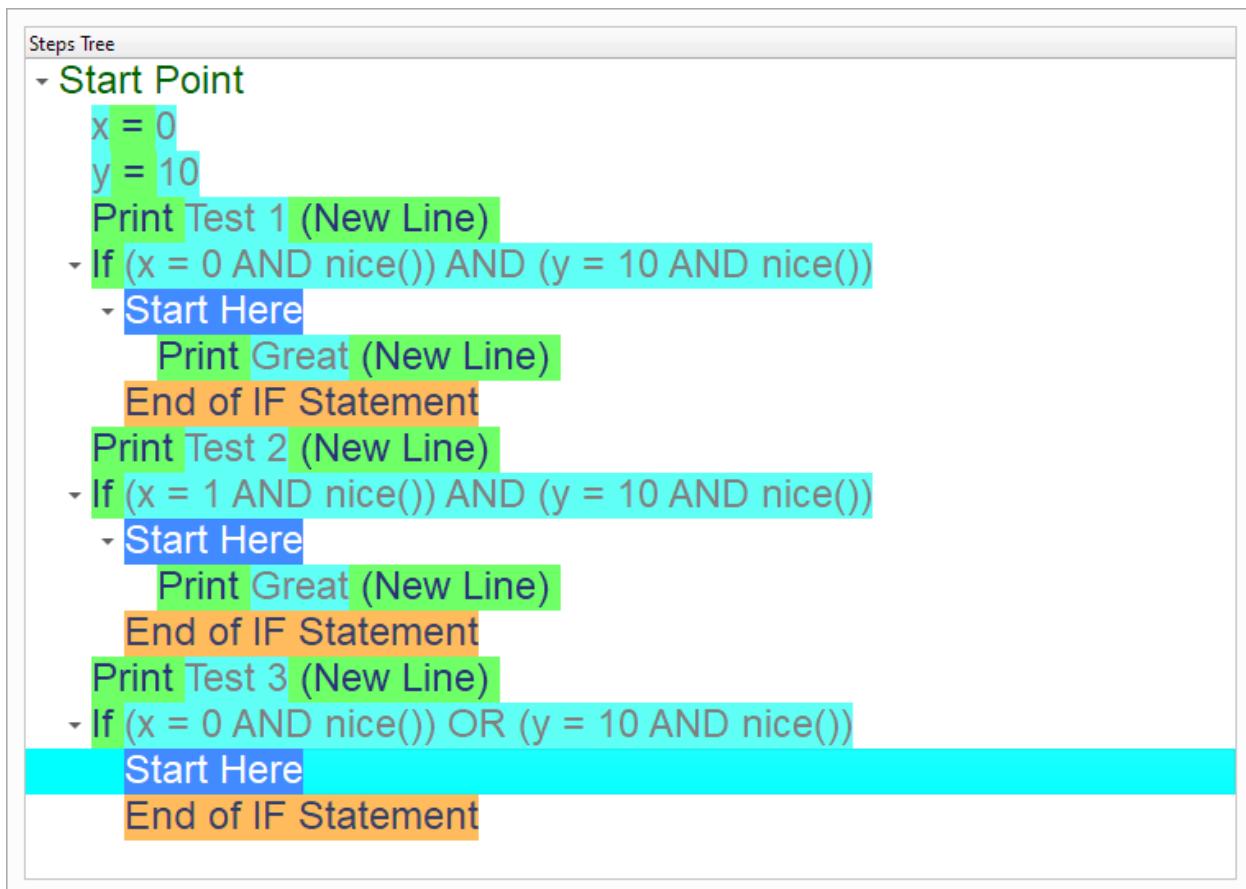
```

Check the condition: $(x = 0 \text{ AND } \text{nice}()) \text{ OR } (y = 10 \text{ AND } \text{nice}())$

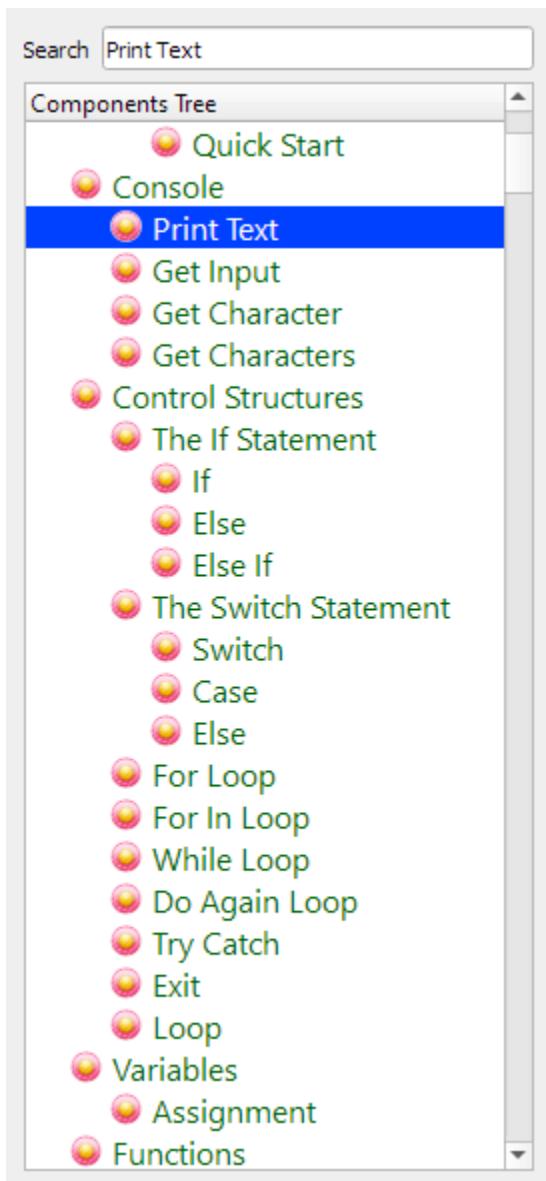
The screenshot shows the PWCT Components Tree interface. At the top left is a search bar with the text "Search If". Below it is a tree view with the following structure:

- Components Tree
 - Quick Start
 - Console
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
 - Control Structures
 - The If Statement
 - If (selected)
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
 - Variables
 - Assignment
 - Functions

Below the tree, a modal dialog is open with the title "If Statement Component". It contains the following text:
Condition : `(x = 0 AND nice()) OR (y = 10 AND nice())`
Buttons at the bottom: "Again" (disabled), "Ok" (highlighted with a green checkmark), and "Close".



Print (Great)



Print Component

Text : Great

Type :

Literal
Expression

New Line

 Again  Ok  Close

Steps Tree

```

x = 0
y = 10
Print Test 1 (New Line)
- If (x = 0 AND nice()) AND (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
    End of IF Statement
  Print Test 2 (New Line)
- If (x = 1 AND nice()) AND (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
    End of IF Statement
  Print Test 3 (New Line)
- If (x = 0 AND nice()) OR (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
    End of IF Statement

```

Define the (Nice) function

Search Define Function

Components Tree

- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
- Assignment
- Functions
 - Define Function
 - Call Function

Define Function Component

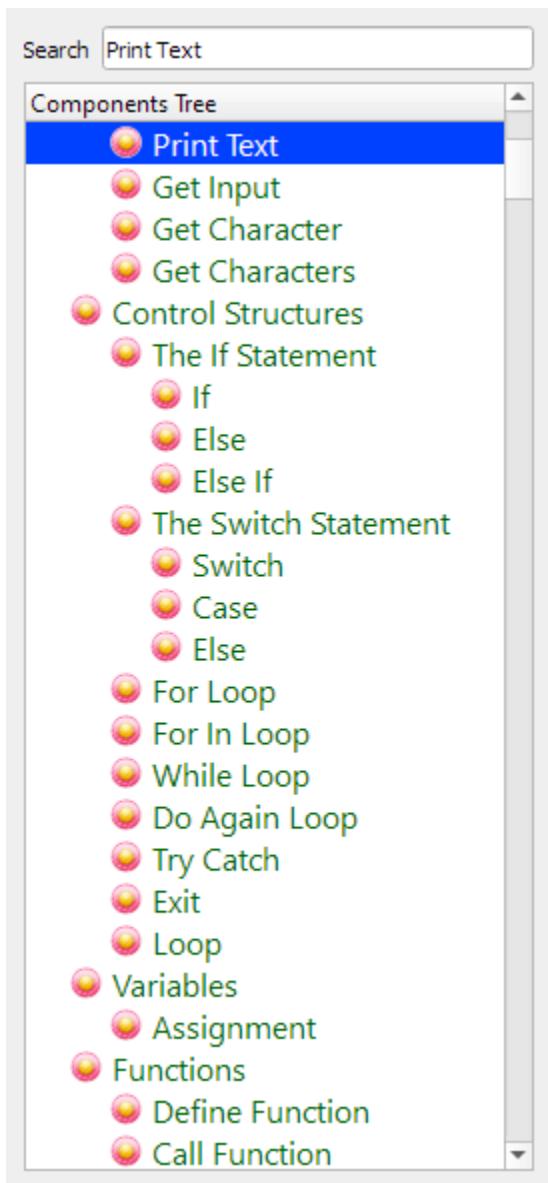
Name : nice

Parameters :

Output :



Print (Nice)



Print Component

Text :

Literal
Expression

Type :

New Line

Steps Tree

```

- If (x = 0 AND nice()) AND (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
    End of IF Statement
  Print Test 2 (New Line)
- If (x = 1 AND nice()) AND (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
    End of IF Statement
  Print Test 3 (New Line)
- If (x = 0 AND nice()) OR (y = 10 AND nice())
  - Start Here
    Print Great (New Line)
    End of IF Statement
- function nice
  - Start Here
    Print Nice (New Line)
    End of Function

```

Return (True) from the (Nice) function

The screenshot shows the PWCT Components Tree on the left and a modal dialog titled "Return Component" on the right.

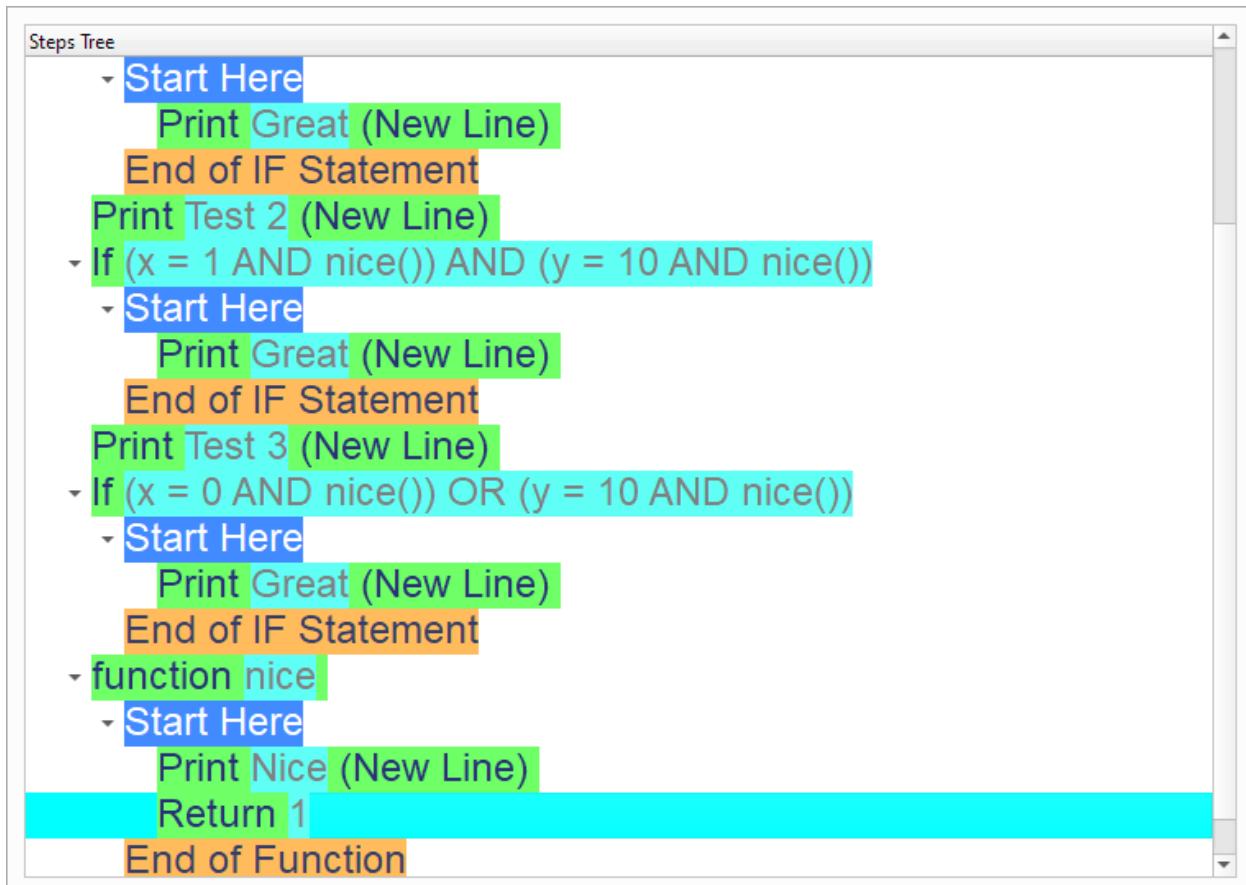
Components Tree:

- Search Return
- Components Tree
 - Get Character
 - Get Characters
 - Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
 - Variables
 - Assignment
 - Functions
 - Define Function
 - Call Function
 - Return
 - Program Structure

Return Component Dialog:

- Value :
- Buttons:
 - Again
 - Ok
 - Close

Now we have the final Steps Tree in our program



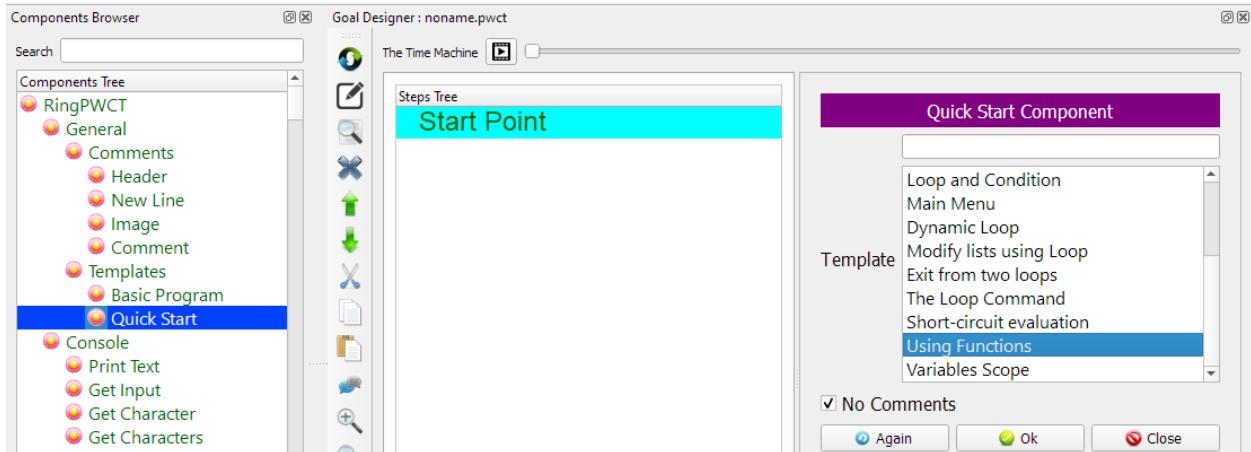
CHAPTER TWENTY

USING FUNCTIONS

In this chapter we are going to learn about the Using Functions

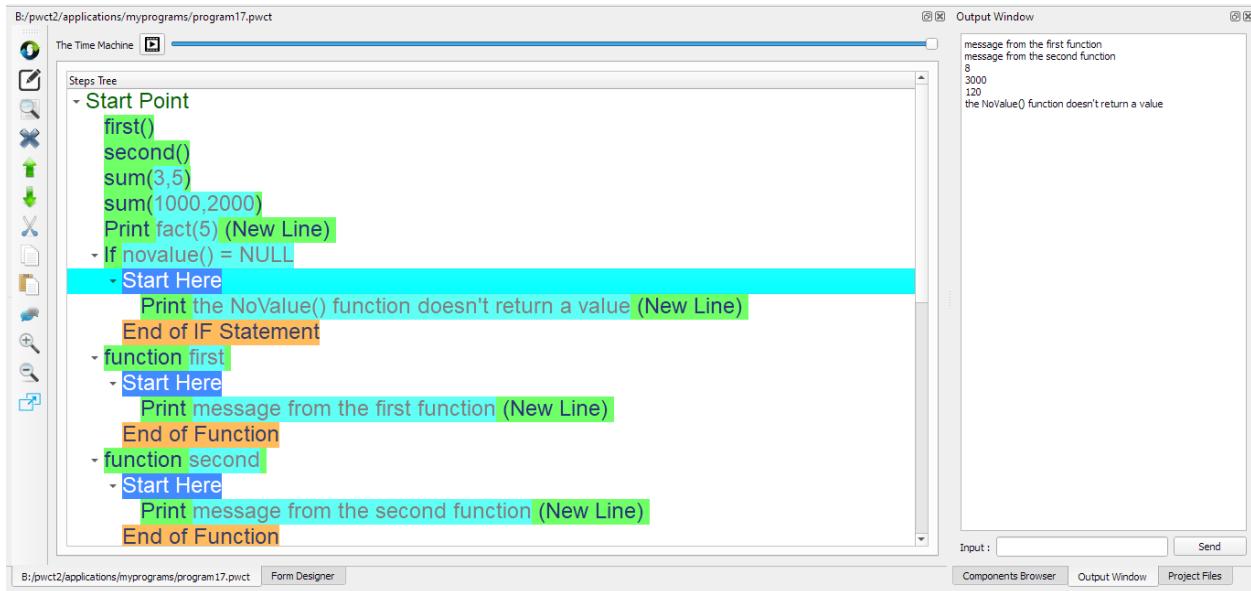
20.1 Introduction

We can create this program quickly using the Quick Start component



20.2 Program Steps

After selecting the (Using Functions) template, we will get the next steps in the Goal Designer



The Steps Tree:

```

first()
second()

sum(3,5)
sum(1000,2000)

Print fact(5) (New Line)

If novalue() = NULL
    Print the NoValue() function doesn't return a value (New Line)
End of IF Statement

function first
    Print message from the first function (New Line)
End of Function

function second
    Print message from the second function (New Line)
End of Function

function sum x,y
    Print x+y (New Line)
End of Function

function fact x
    If x = 0
        Return 1
    Else

```

(continues on next page)

(continued from previous page)

```
    Return x*fact(x-1)
End of IF Statement
End of Function
```

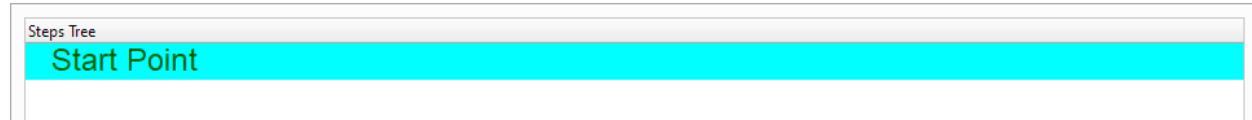
```
function novalue
End of Function
```

20.3 Creating the Program

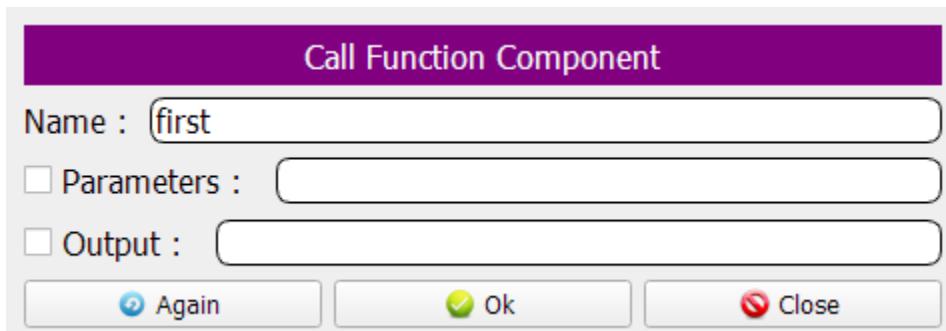
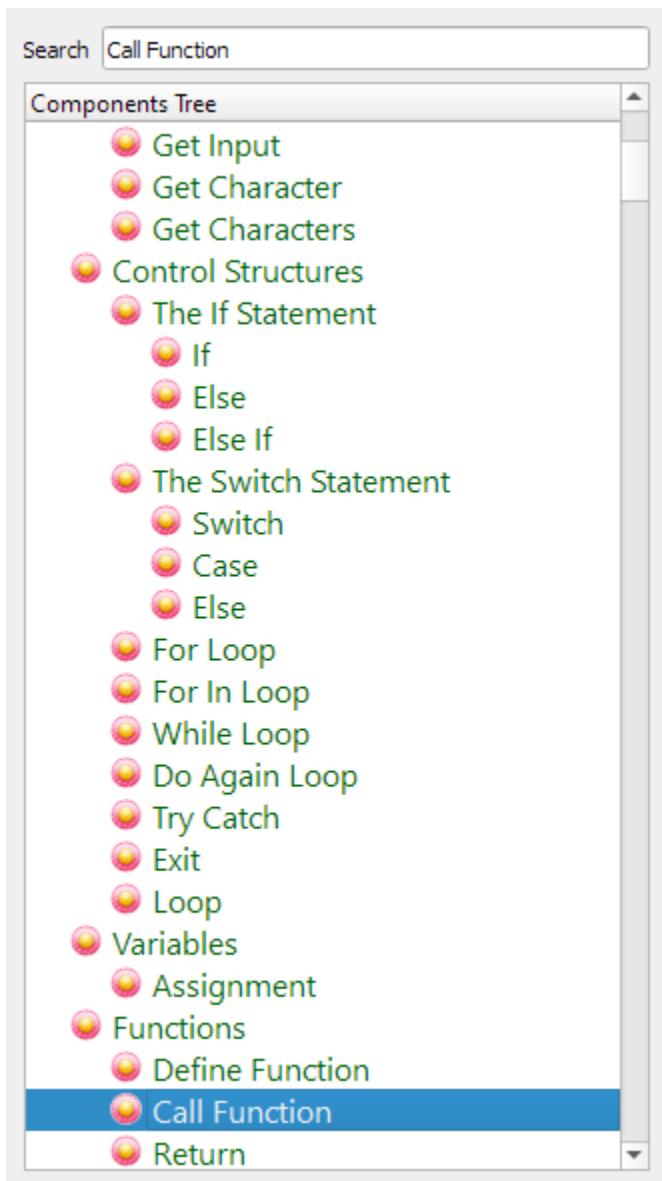
To create this program we will use the next components

- Call Function
- If Statement
- Else
- Define Function
- Print Text
- Return

In the begining the Steps Tree is empty



Call the first() function



Call the Second() function

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar labeled "Search" and a button labeled "Call Function". Below the search bar is a tree view titled "Components Tree" containing the following items:

- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
- Assignment
- Functions
 - Define Function
 - Call Function
- Return

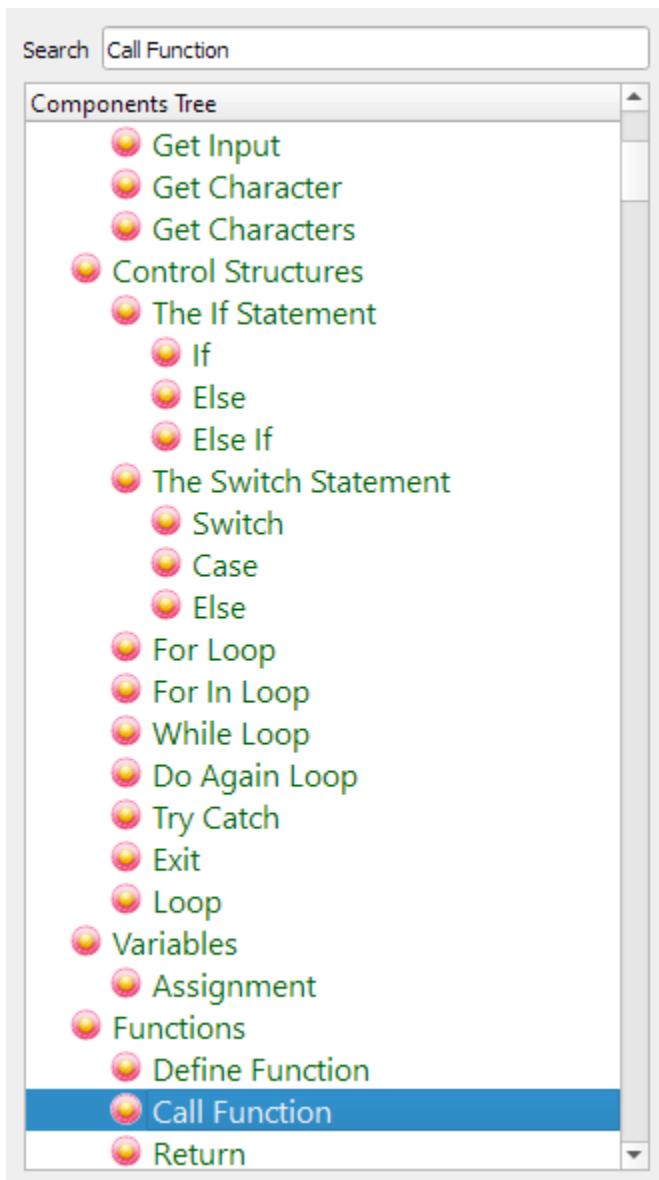
The "Call Function" item is highlighted with a blue selection bar at the bottom of the tree.

Below the tree, a modal dialog box is displayed with the title "Call Function Component". The dialog contains the following fields:

Name :	second
<input type="checkbox"/> Parameters :	
<input type="checkbox"/> Output :	
<input type="button" value="Again"/> <input type="button" value="Ok"/> <input type="button" value="Close"/>	



Call the Sum() function - Send 3,5 as parameters



Call Function Component

Name : sum

Parameters : 3,5

Output :

 Again  Ok  Close

Steps Tree

- Start Point
 - first()
 - second()
 - sum(3,5)

Call the Sum() function - Send 1000,2000 as parameters

Search Call Function

Components Tree

- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function
 - Return

Call Function Component

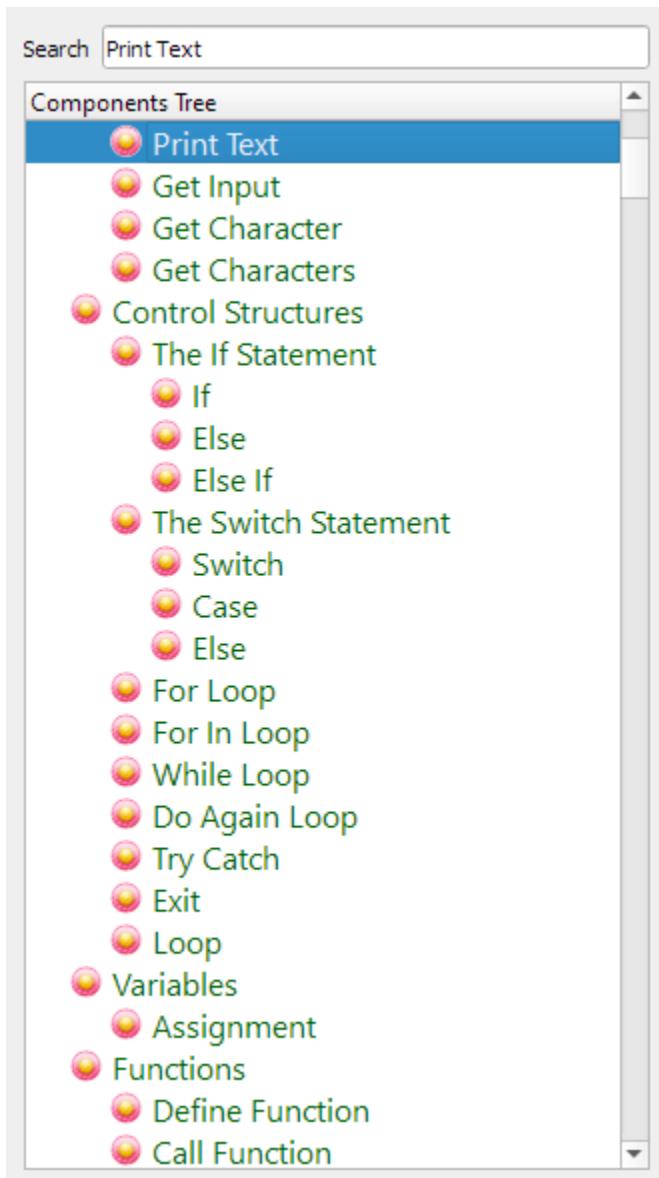
Name : sum

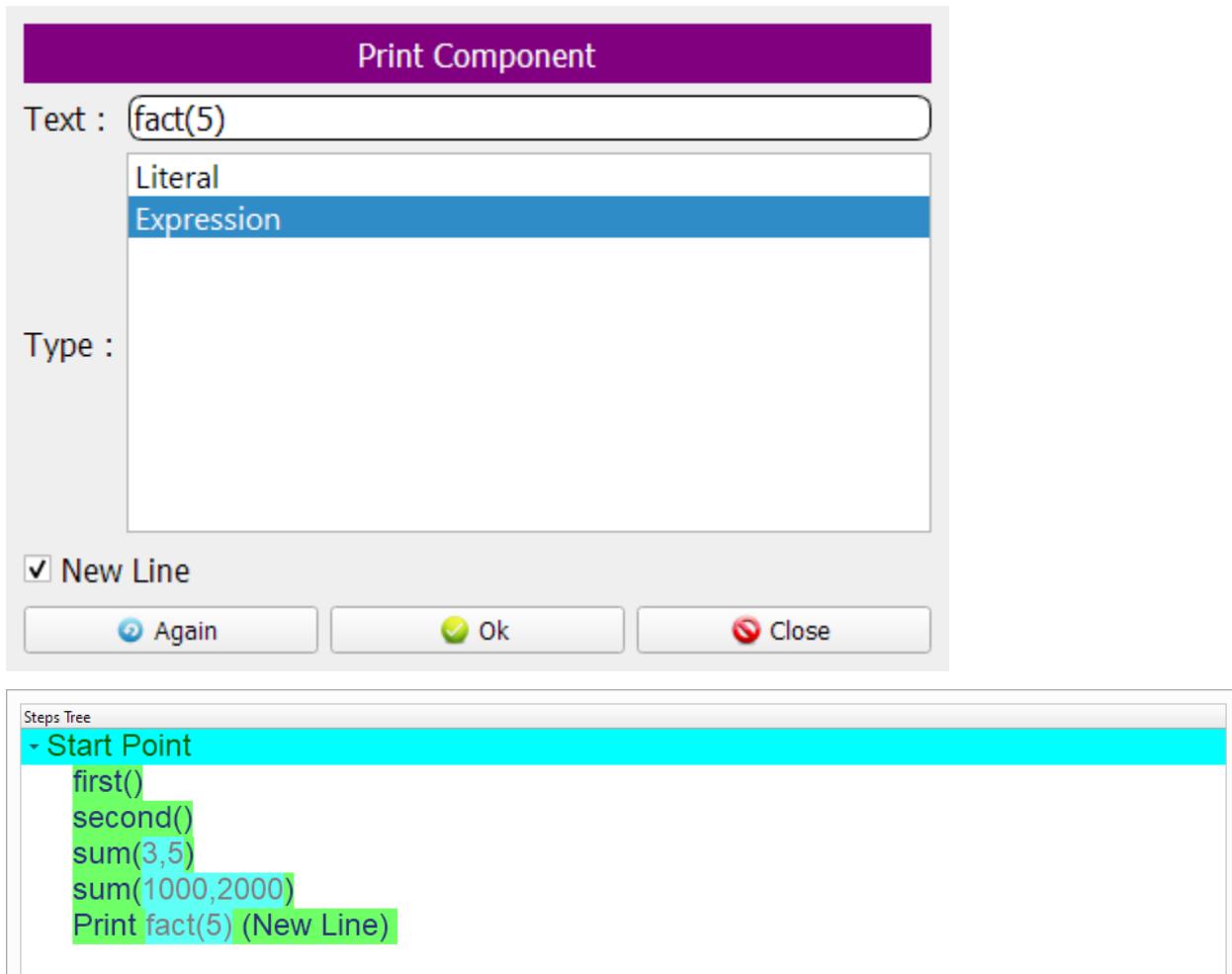
Parameters : 1000,2000

Output :



Call the Fact() function - Send 5 as parameter





Check the output of the NoValue() function

Search If

Components Tree

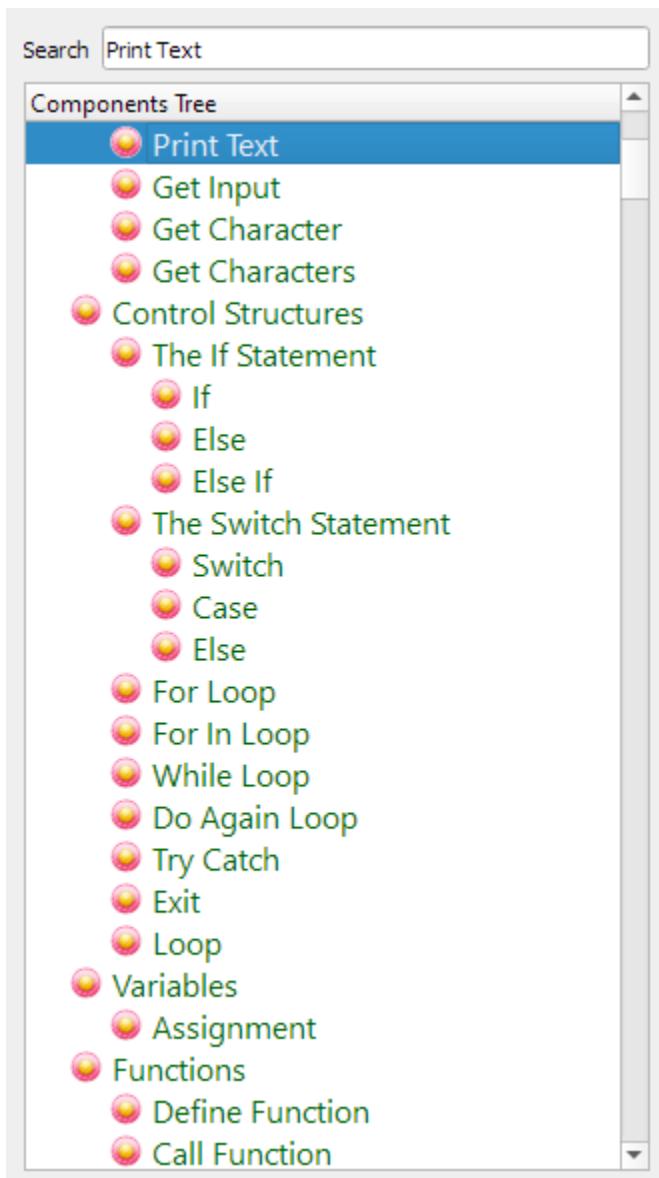
- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
- The Switch Statement
 - Switch
 - Case
 - Else
- For Loop
- For In Loop
- While Loop
- Do Again Loop
- Try Catch
- Exit
- Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function

If Statement Component

Condition : novalue() = NULL



Print a message using the (Print Text) component



Print Component

Text : the NoValue() function doesn't return a value

Type :

Literal
Expression

New Line

 Again  Ok  Close

Steps Tree

- Start Point
 - first()
 - second()
 - sum(3,5)
 - sum(1000,2000)
 - Print fact(5) (New Line)
- If novalue() = NULL
 - Start Here
 - Print the NoValue() function doesn't return a value (New Line)
 - End of IF Statement

Define the First() function

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar labeled "Search" with the text "Define Function". Below the search bar is a tree view titled "Components Tree". The tree structure includes the following categories and components:

- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function

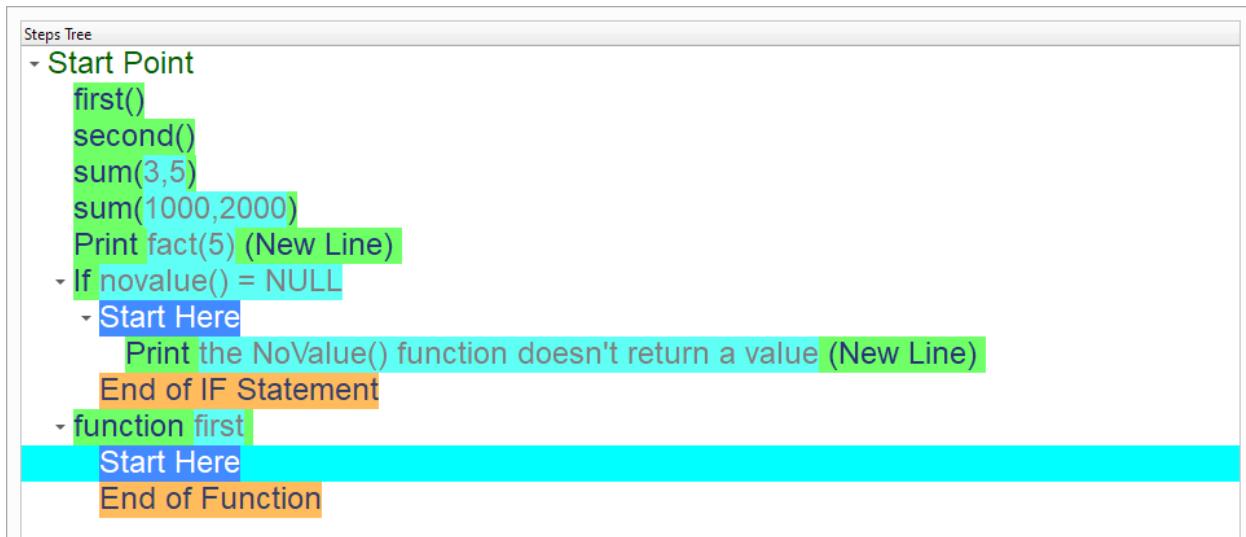
The "Define Function" component is highlighted with a blue selection bar at the bottom of the tree.

Define Function Component

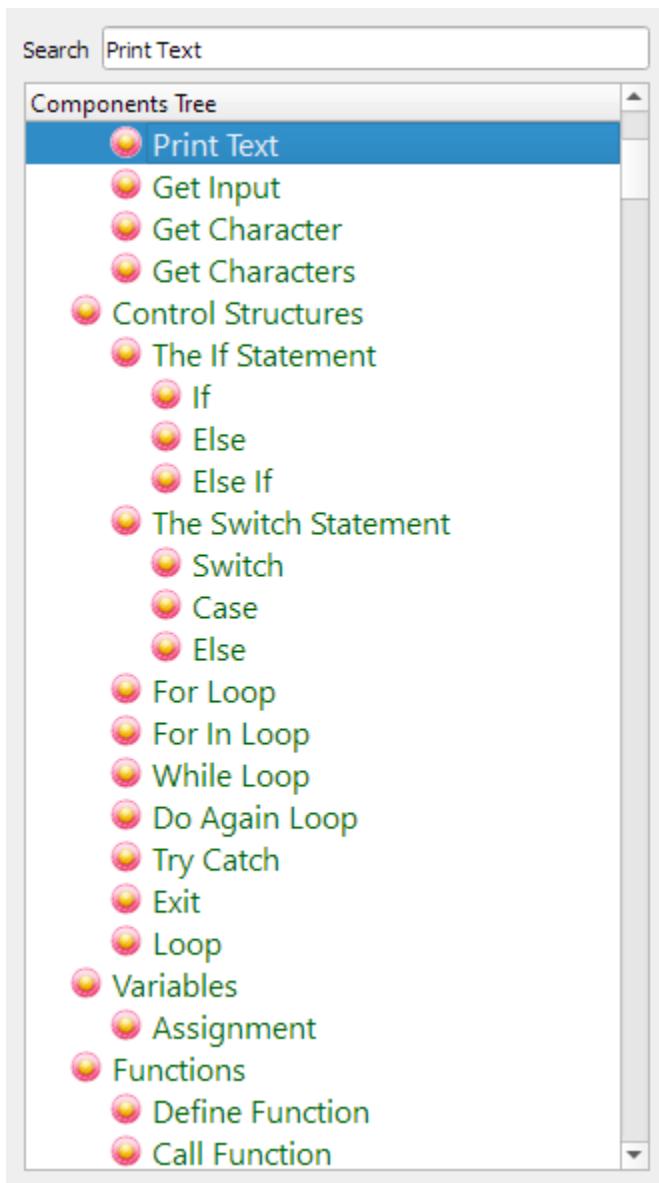
Name :

Parameters :

Output :



Print a message from the First() function



Print Component

Type :

Literal
Expression

New Line

Steps Tree

```

Start Point
  first()
  second()
  sum(3,5)
  sum(1000,2000)
  Print fact(5) (New Line)
  - If novalue() = NULL
    - Start Here
      Print the NoValue() function doesn't return a value (New Line)
      End of IF Statement
  - function first
    - Start Here
      Print message from the first function (New Line)
      End of Function
  
```

Define the Second() function

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar labeled "Search" with the text "Define Function". Below the search bar is a tree view titled "Components Tree". The tree structure includes the following categories and components:

- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function

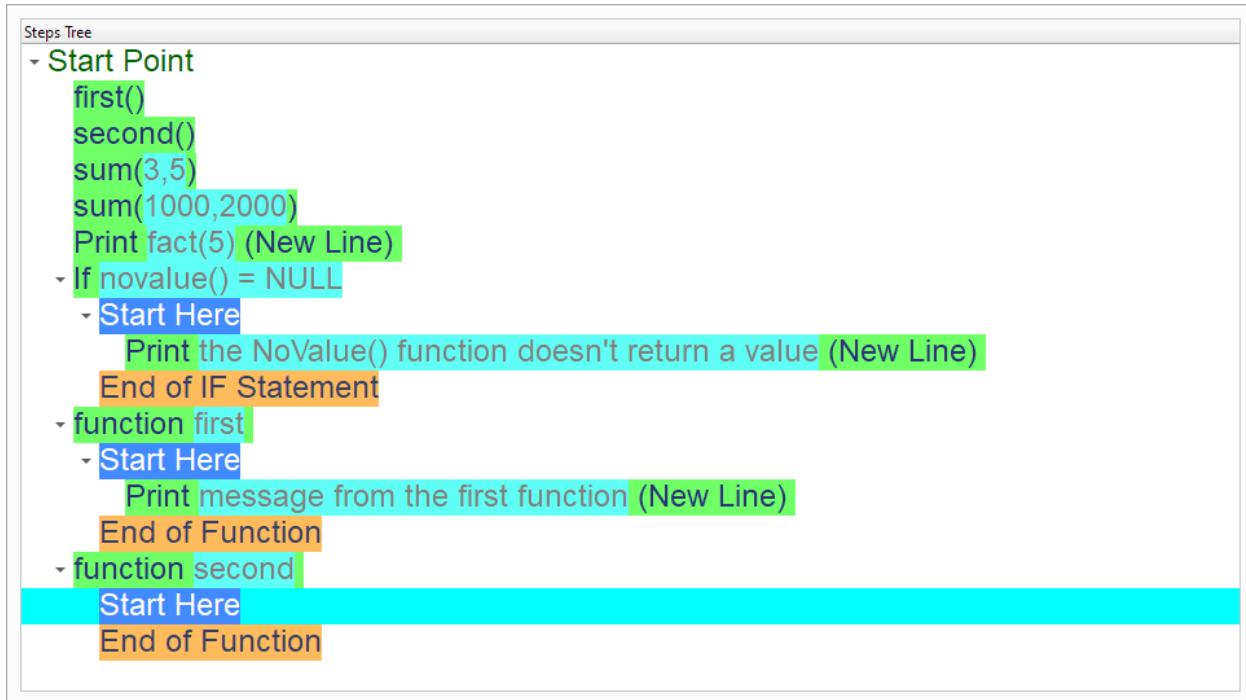
The "Define Function" component is highlighted with a blue selection bar at the bottom of the tree.

Define Function Component

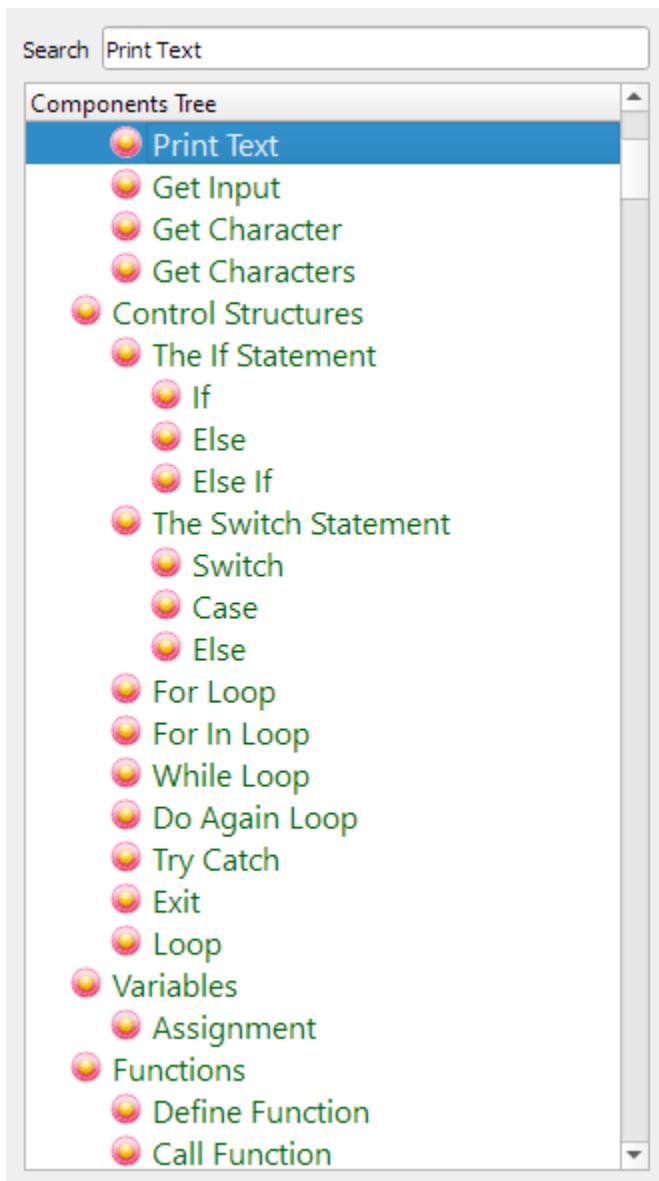
Name :

Parameters :

Output :



Print a message from the Second() function



Print Component

Text : message from the second function

Type :

Literal
Expression

New Line

 Again  Ok  Close

Steps Tree

```

first()
second()
sum(3,5)
sum(1000,2000)
Print fact(5) (New Line)
- If novalue() = NULL
  - Start Here
    Print the NoValue() function doesn't return a value (New Line)
    End of IF Statement
- function first
  - Start Here
    Print message from the first function (New Line)
    End of Function
- function second
  - Start Here
    Print message from the second function (New Line)
    End of Function

```

Define the Sum() function

This function takes x,y as parameters

The function will print the sum of x and y

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar labeled "Search" with the text "Define Function". Below the search bar is a tree view titled "Components Tree". The tree structure includes the following categories and components:

- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function

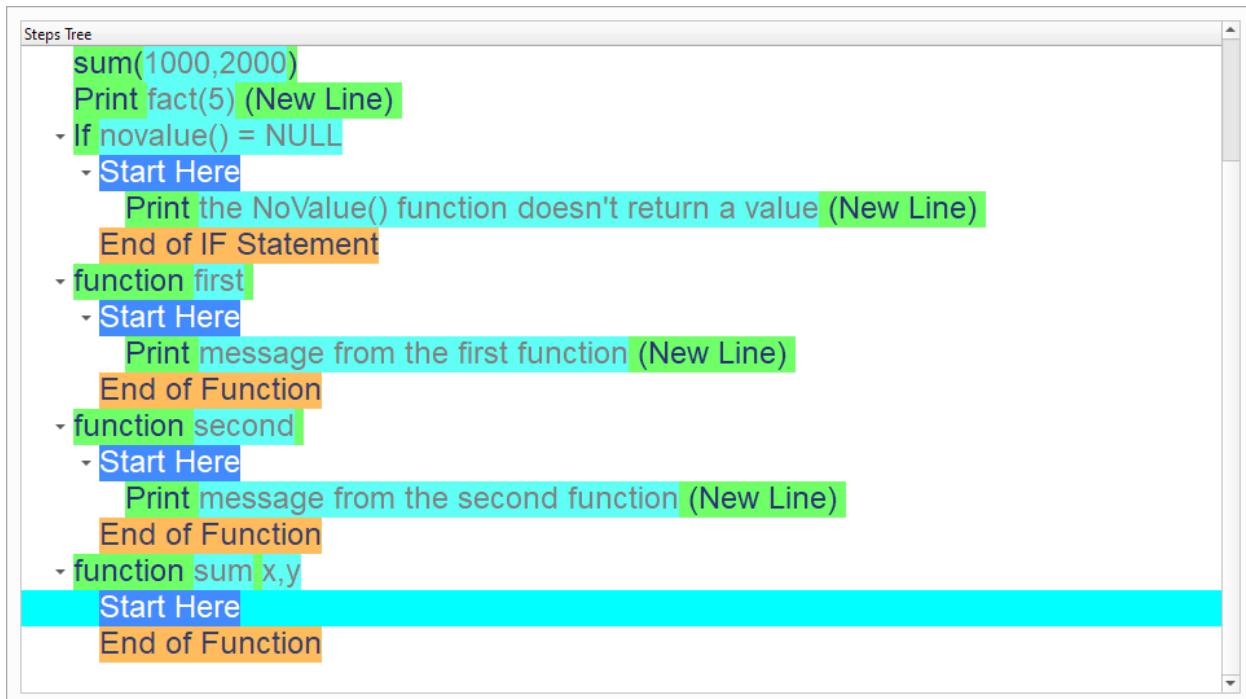
The "Define Function" component is highlighted with a blue selection bar at the bottom of the tree.

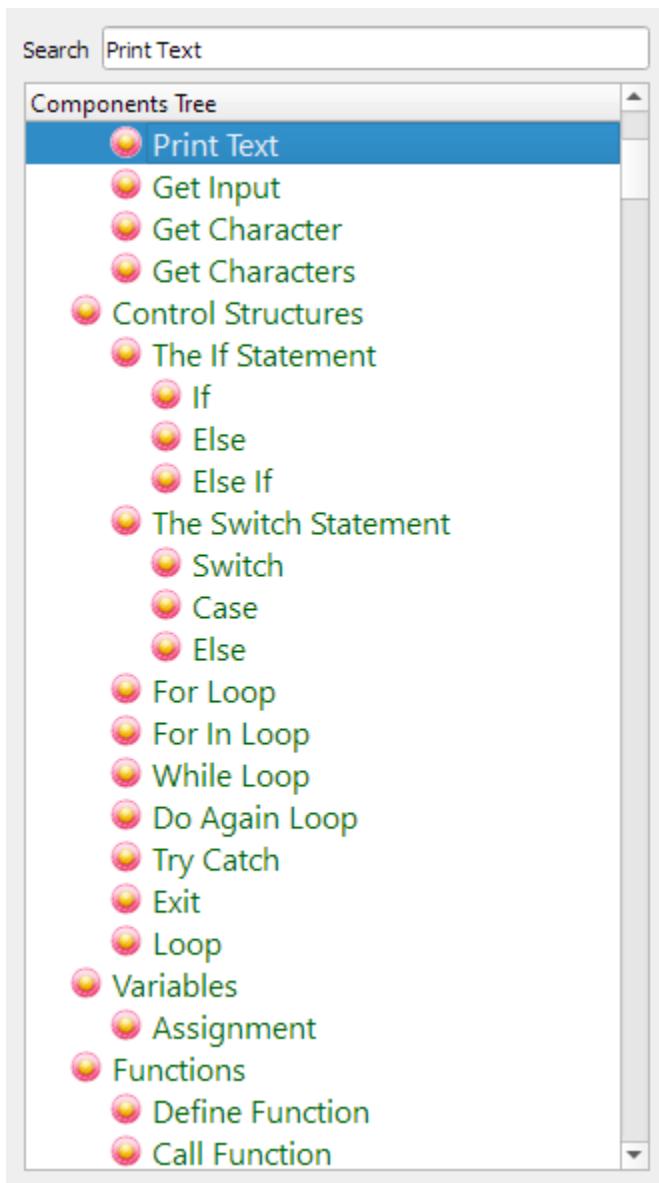
Define Function Component

Name : sum

Parameters : X,y

Output :





Print Component

Text :

Type :

Literal
Expression

New Line

Steps Tree

```

Print fact(5) (New Line)
- If novalue() = NULL
  - Start Here
    Print the NoValue() function doesn't return a value (New Line)
    End of IF Statement
- function first
  - Start Here
    Print message from the first function (New Line)
    End of Function
- function second
  - Start Here
    Print message from the second function (New Line)
    End of Function
- function sum x,y
  - Start Here
    Print x+y (New Line)
    End of Function

```

Define the Fact() function

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar labeled "Search" with the text "Define Function". Below the search bar is a tree view with the following structure:

- Components Tree
 - Print Text
 - Get Input
 - Get Character
 - Get Characters
 - Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
 - Variables
 - Assignment
 - Functions
 - Define Function
 - Call Function

Define Function Component

Name : fact

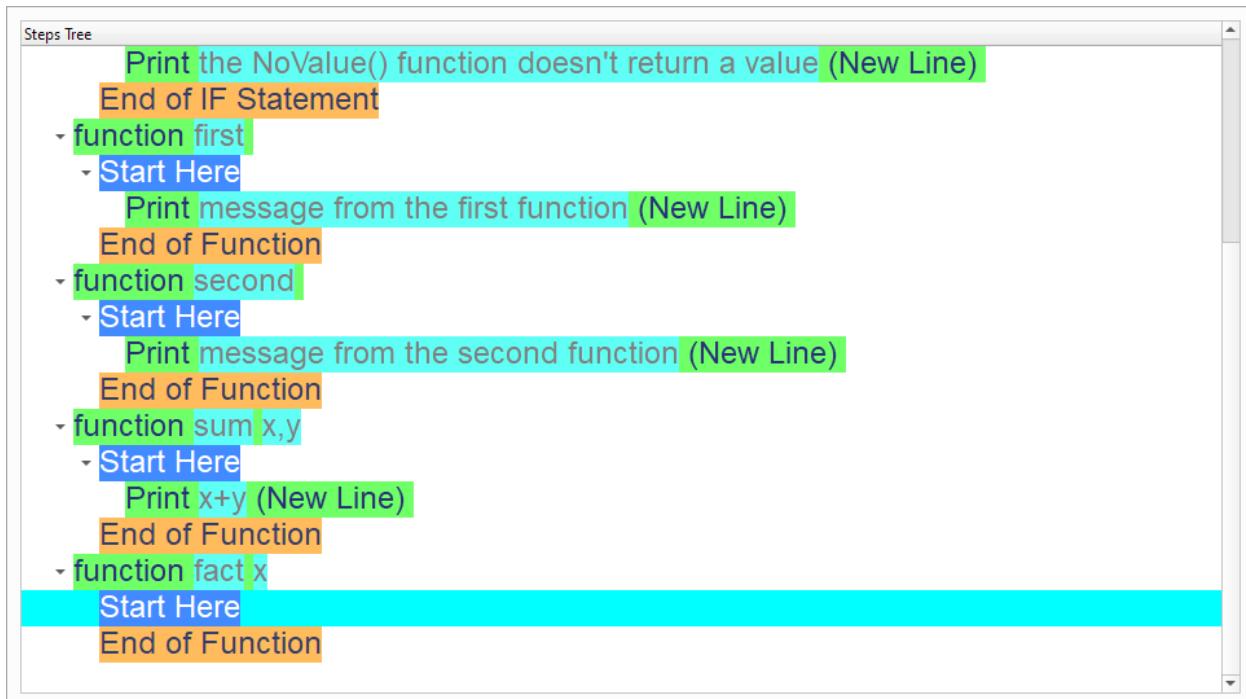
Parameters : x

Output :

Again

Ok

Close



Search If

Components Tree

- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
- The Switch Statement
 - Switch
 - Case
 - Else
- For Loop
- For In Loop
- While Loop
- Do Again Loop
- Try Catch
- Exit
- Loop
- Variables
- Assignment
- Functions
 - Define Function
 - Call Function

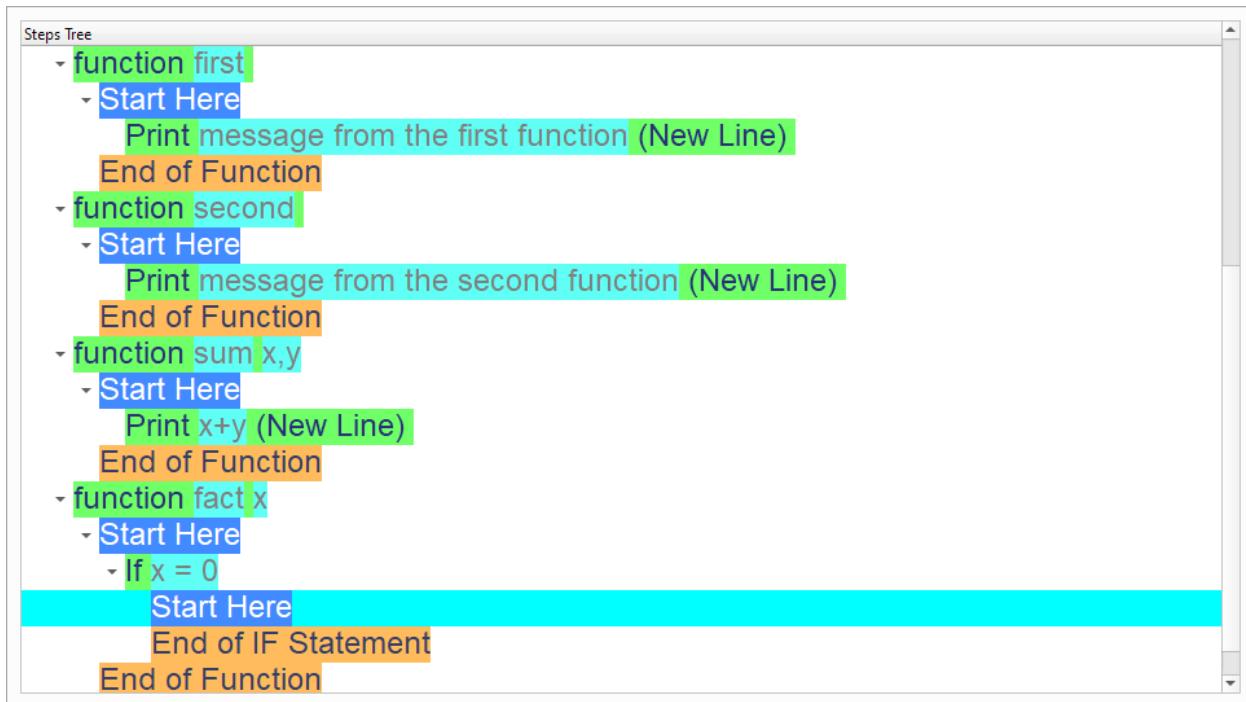
If Statement Component

Condition :

 Again

 Ok

 Close



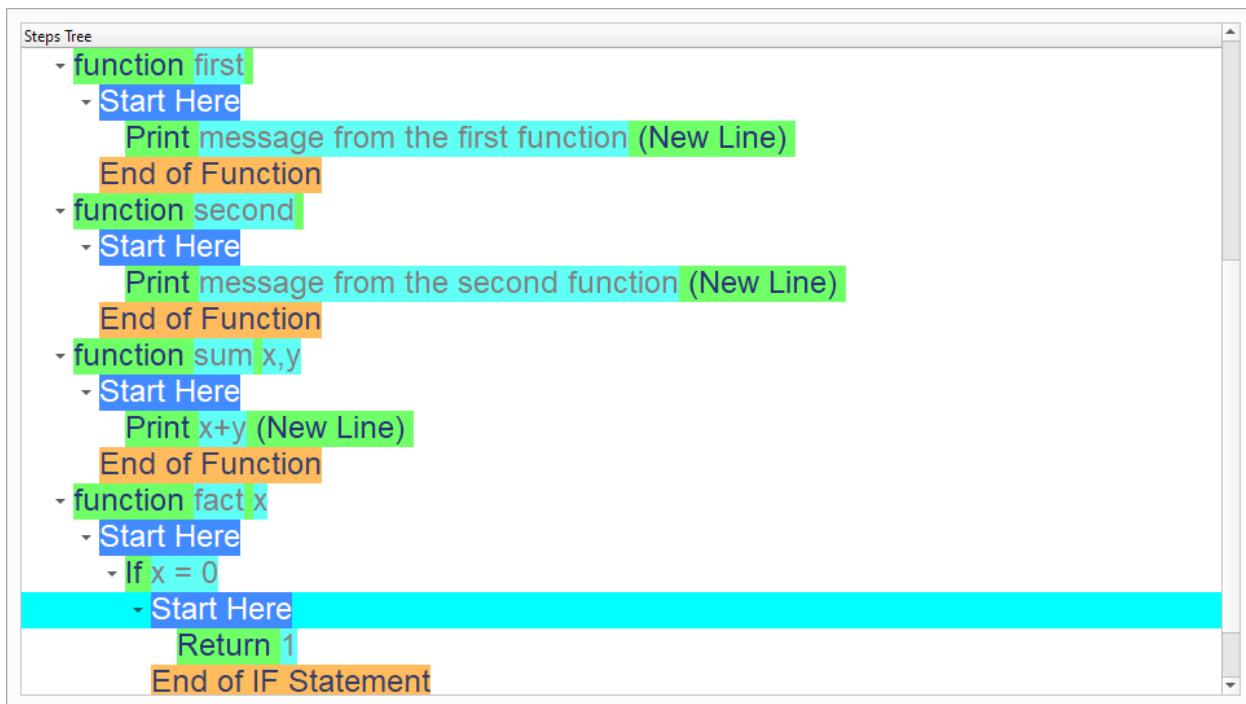
Search Return

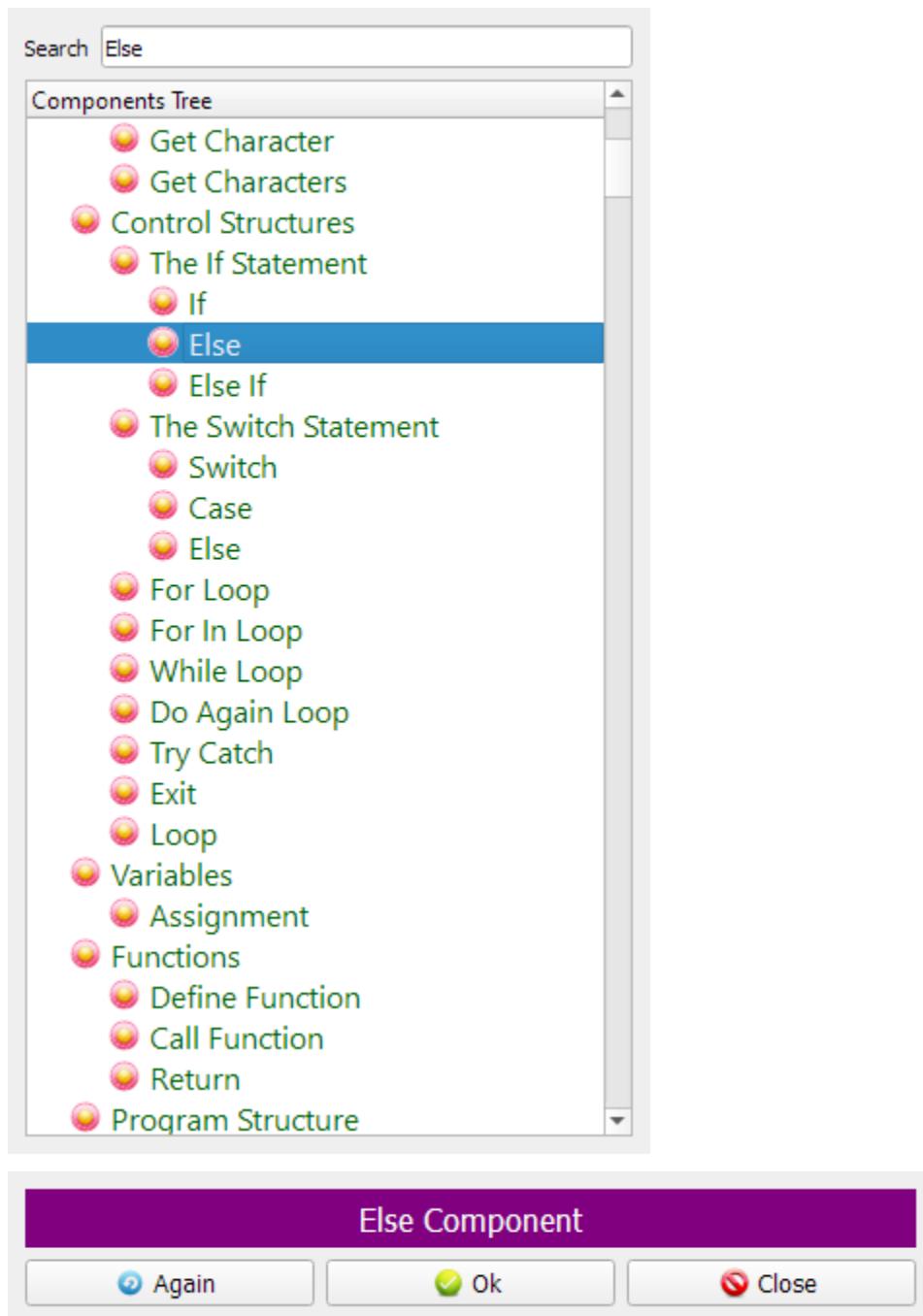
Components Tree

- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function
- Return
- Program Structure

Return Component

Value :







Search

Components Tree

- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function
-
- Program Structure

Return Component

Value :



Define the NoValue() function

The screenshot shows the PWCT Components Tree interface. At the top, there is a search bar labeled "Search" with the text "Define Function". Below the search bar is a tree view titled "Components Tree". The tree structure is as follows:

- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function
 - Return
- Program Structure

The "Define Function" node is highlighted with a blue selection bar at the bottom of the tree.

Below the tree, there is a dialog box titled "Define Function Component". The dialog contains the following fields:

Name :	novalue
<input type="checkbox"/> Parameters :	
<input type="checkbox"/> Output :	
<input type="button" value="Again"/> <input type="button" value="Ok"/> <input type="button" value="Close"/>	

Now we have the final Steps Tree in our program



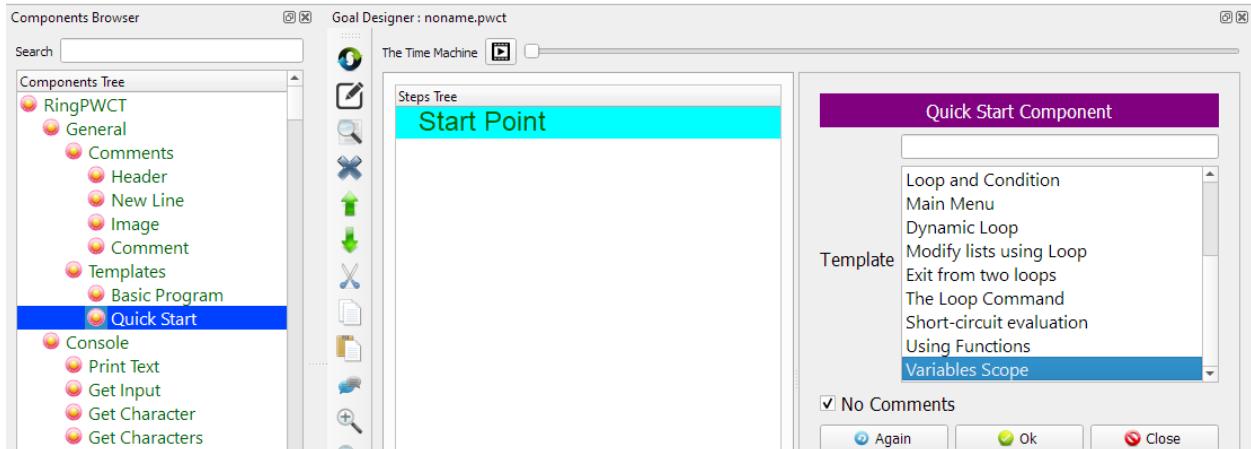
CHAPTER TWENTYONE

VARIABLES SCOPE

In this chapter we are going to learn about the Variables Scope

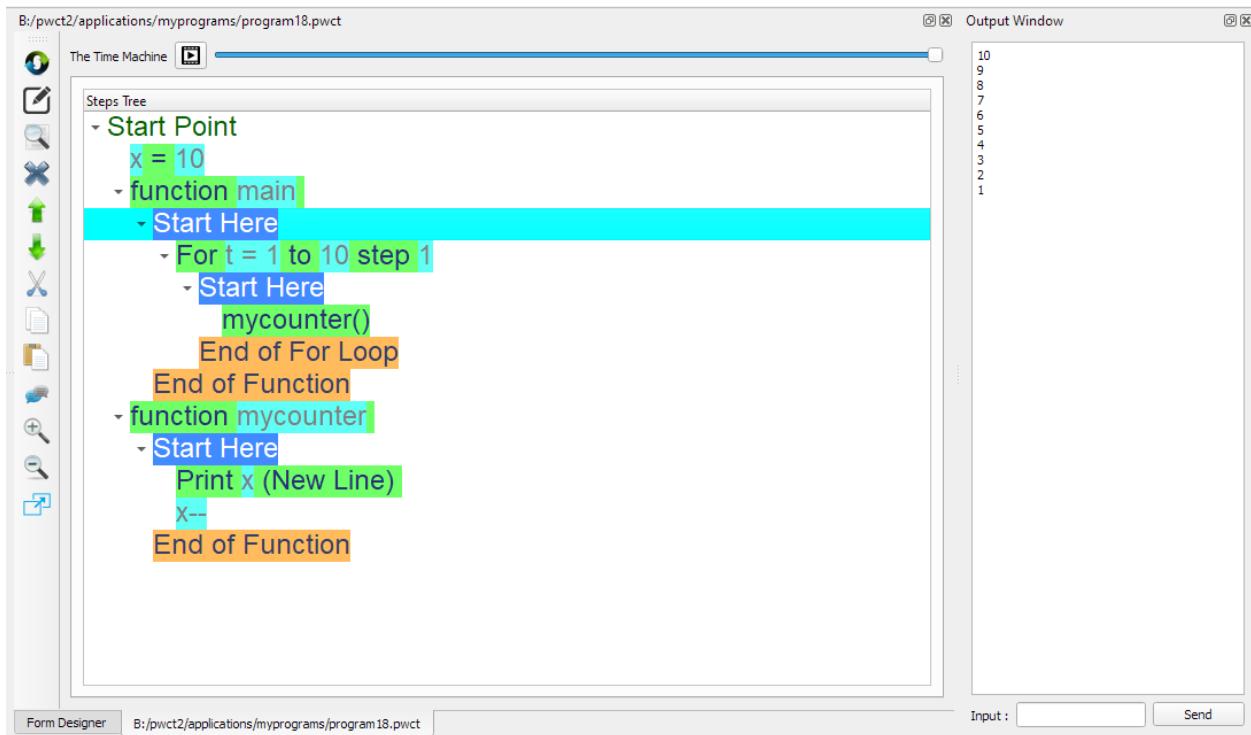
21.1 Introduction

We can create this program quickly using the Quick Start component



21.2 Program Steps

After selecting the (Variables Scope) template, we will get the next steps in the Goal Designer



21.3 Creating the Program

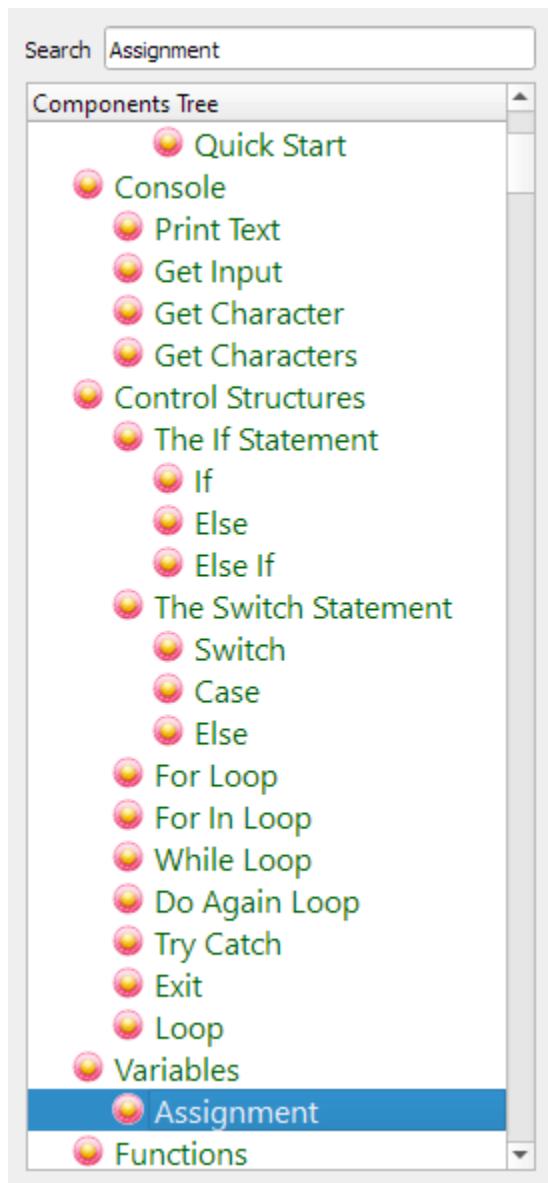
To create this program we will use the next components

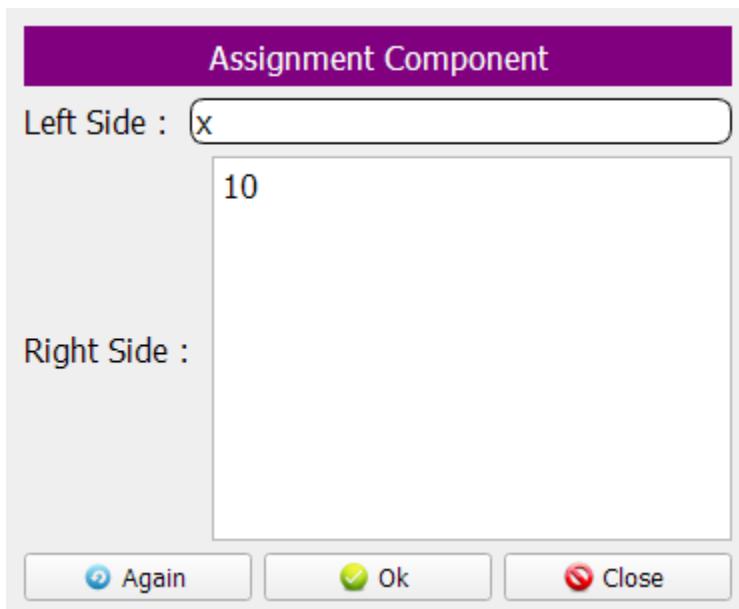
- Assignment
- Define Function
- Call Function
- For Loop
- Print Text

In the begining the Steps Tree is empty



Set $x=10$ using the Assignment component





Define the Main() function

Search Define Function

Components Tree

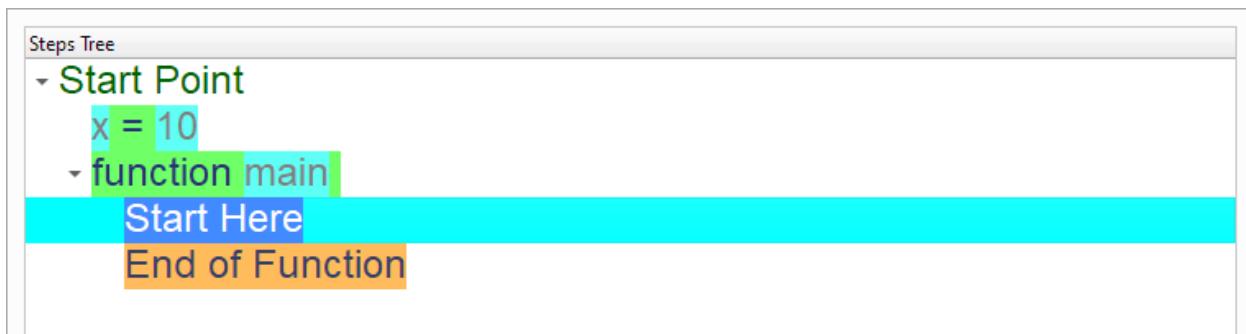
- Print Text
- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
- Assignment
- Functions
 - Define Function
 - Call Function

Define Function Component

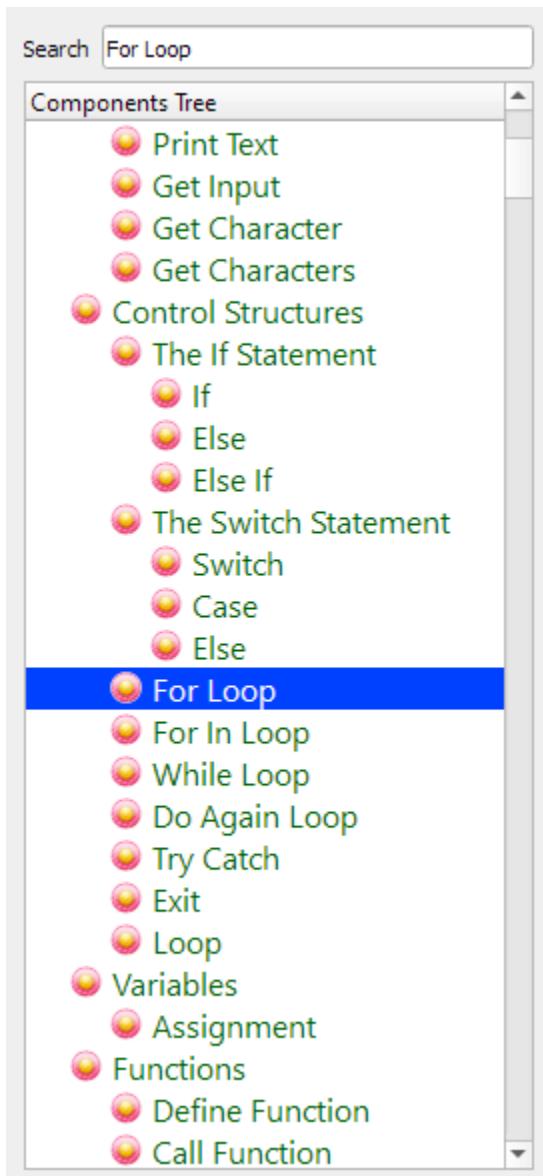
Name :

Parameters :

Output :



Count from 1 to 10 using a For Loop



For Loop Component

Start :

To :

Step :

Steps Tree

- Start Point
 - x = 10
- function main
 - Start Here
 - For t = 1 to 10 step 1
 - Start Here
 - End of For Loop
 - End of Function

Call the mycounter() function

Search Call Function

Components Tree

- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
 - Assignment
- Functions
 - Define Function
 - Call Function
 - Return

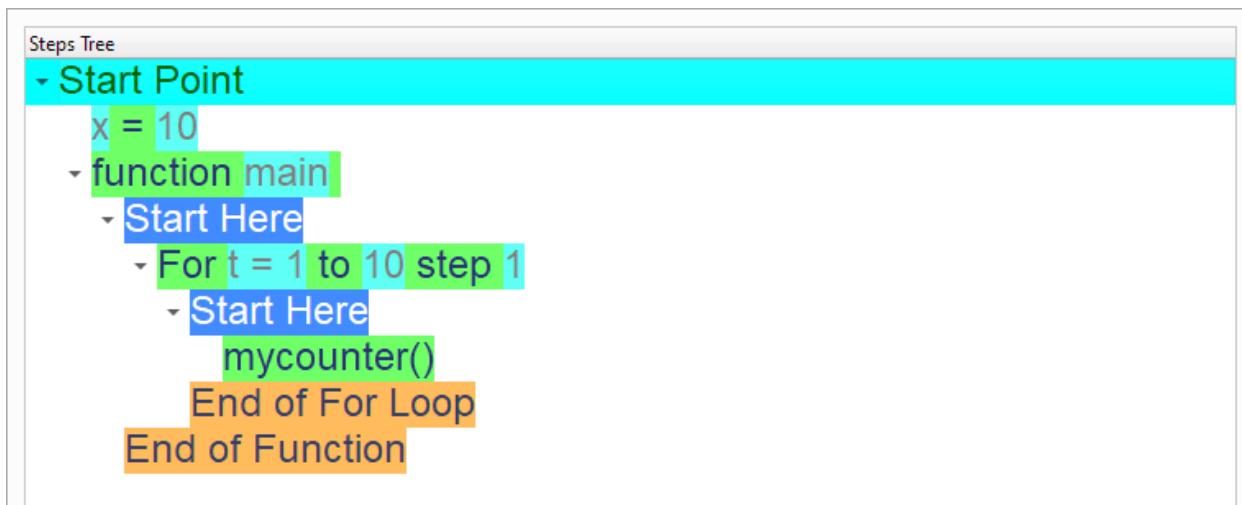
Call Function Component

Name : mycounter

Parameters :

Output :

Again Ok Close



Define the mycounter() function

Search Define Function

Components Tree

- Get Input
- Get Character
- Get Characters
- Control Structures
 - The If Statement
 - If
 - Else
 - Else If
 - The Switch Statement
 - Switch
 - Case
 - Else
 - For Loop
 - For In Loop
 - While Loop
 - Do Again Loop
 - Try Catch
 - Exit
 - Loop
- Variables
- Assignment
- Functions
 - Define Function
 - Call Function
 - Return

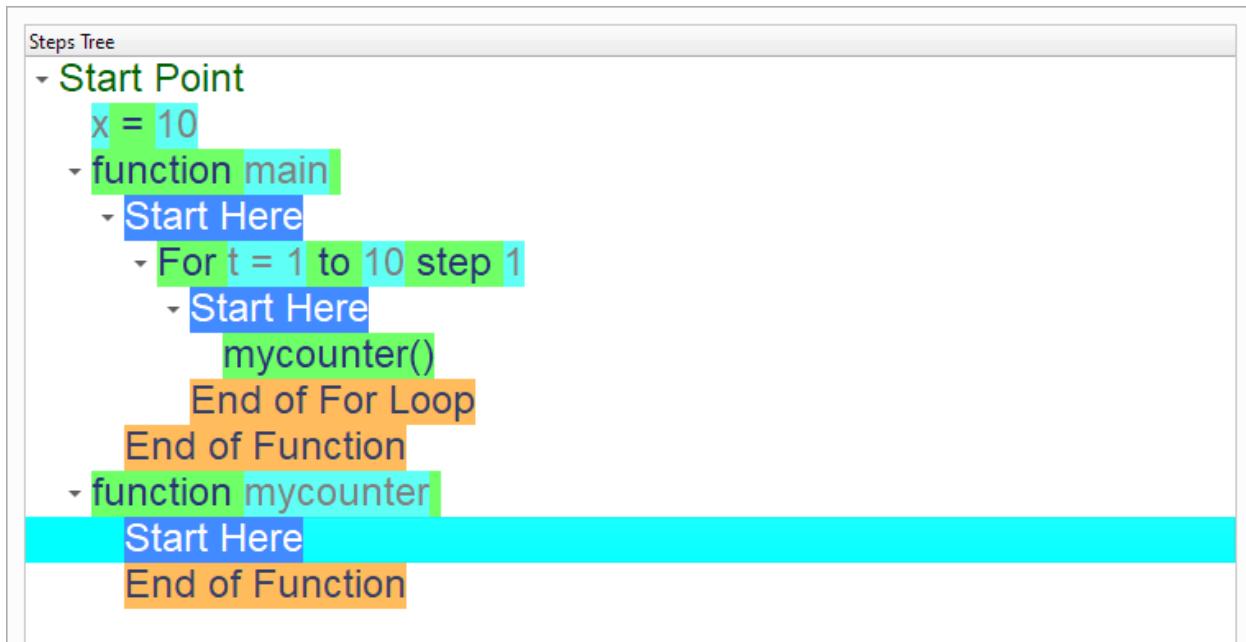
Define Function Component

Name : mycounter

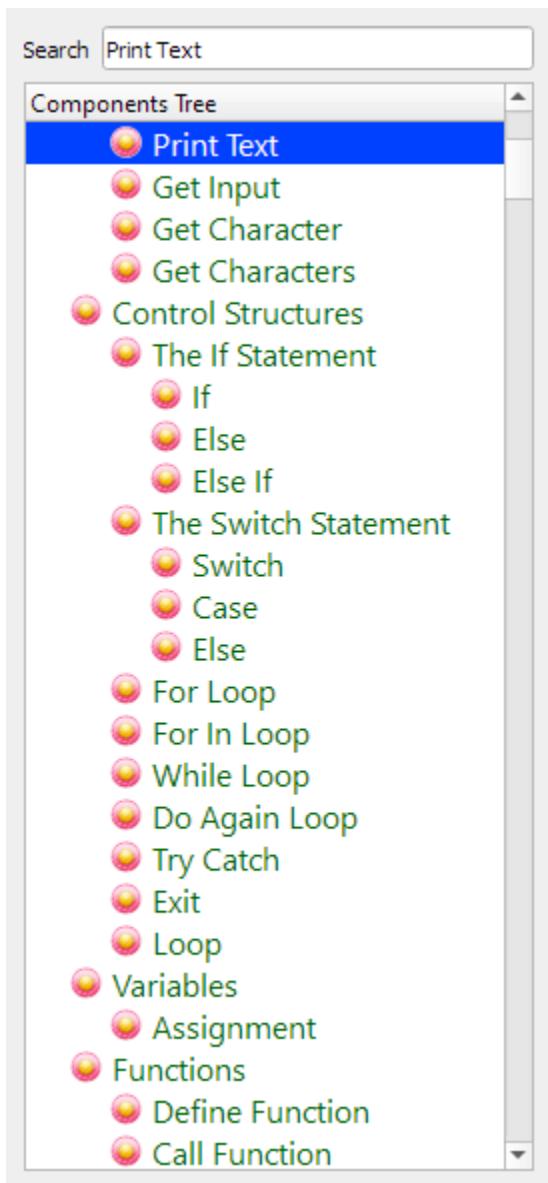
Parameters :

Output :

Again Ok Close



Print the X variable



Print Component

Text :

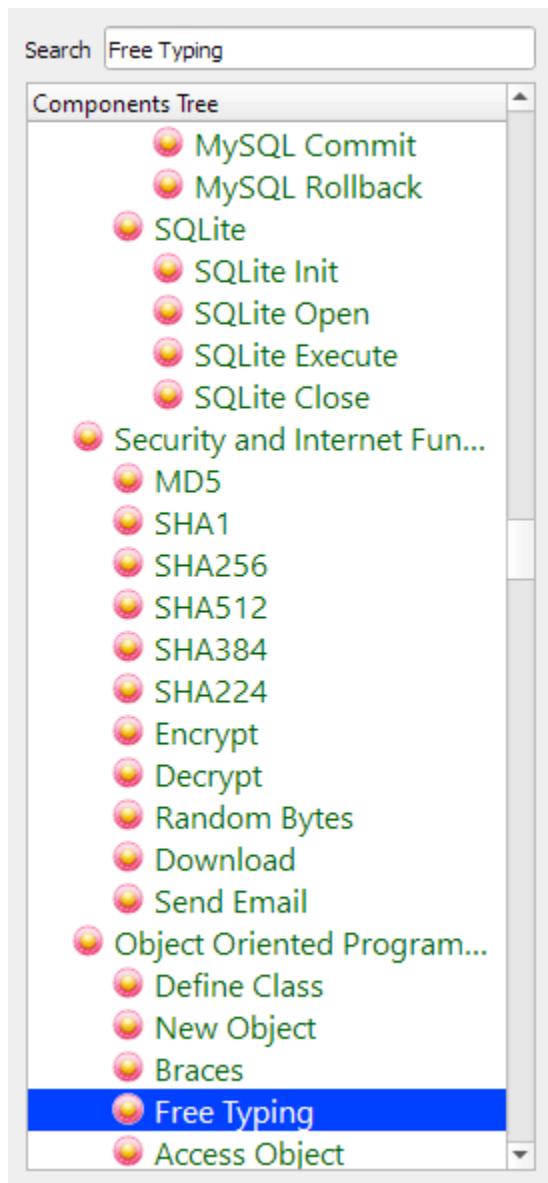
Type :

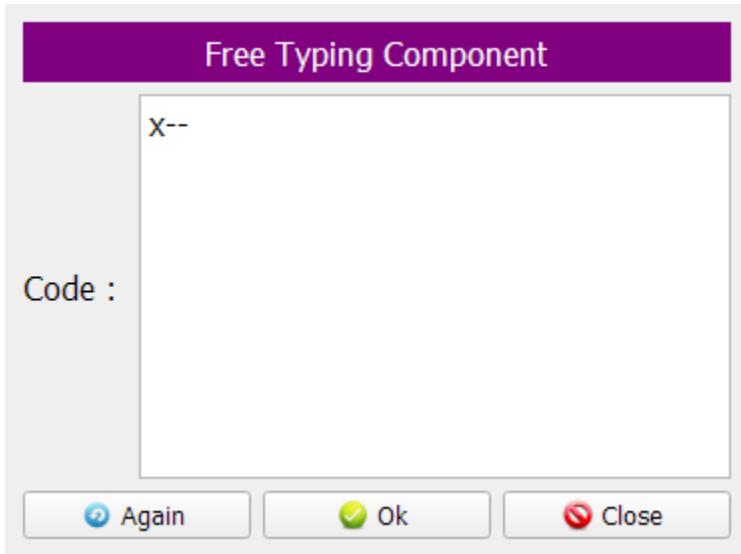
New Line

Steps Tree

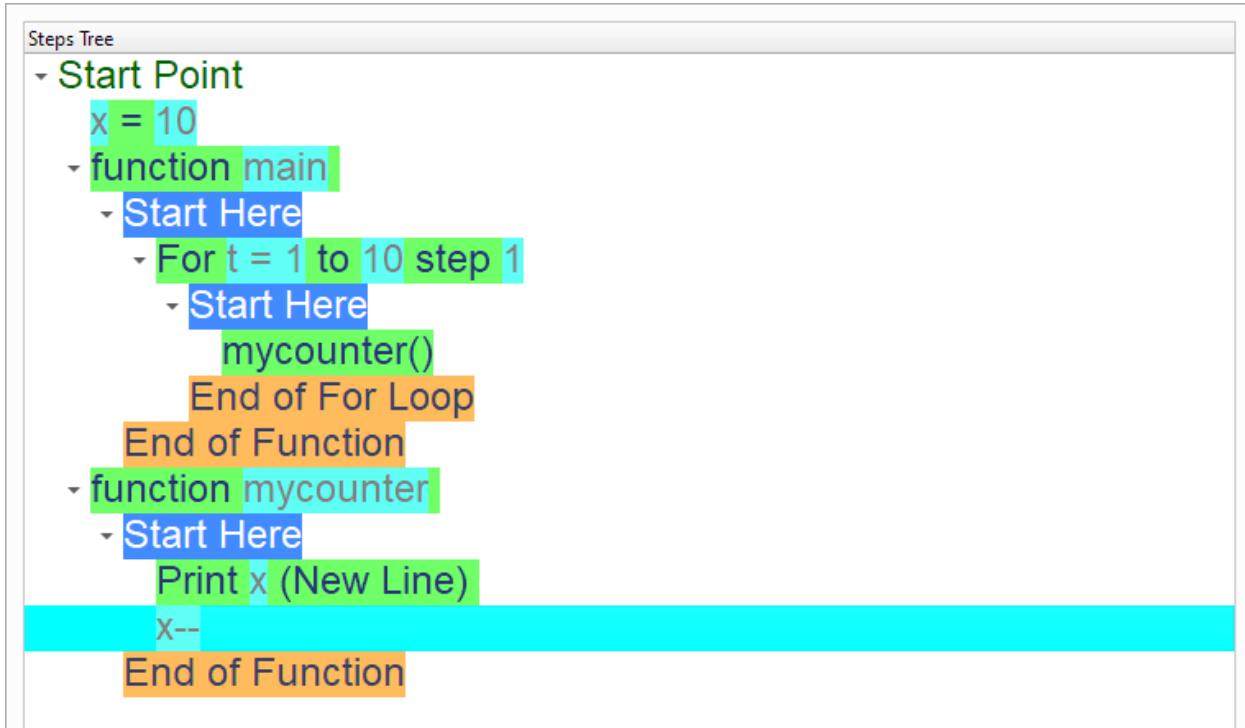
- Start Point
 - x = 10
- function main
 - Start Here
 - For t = 1 to 10 step 1
 - Start Here
 - mycounter()
 - End of For Loop
 - End of Function
 - function mycounter
 - Start Here
 - Print x (New Line)
 - End of Function

Decrement the X value





Now we have the final Steps Tree in our program



CHAPTER
TWENTYTWO

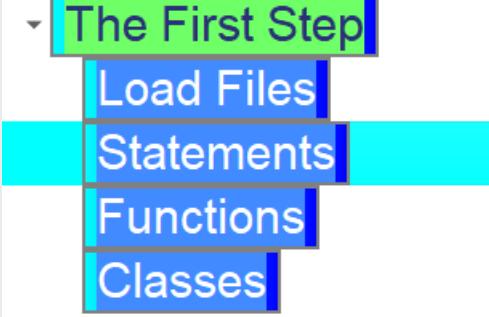
RINGPWCT VISUAL COMPONENTS

In this chapter, we present the visual components available in RingPWCT.

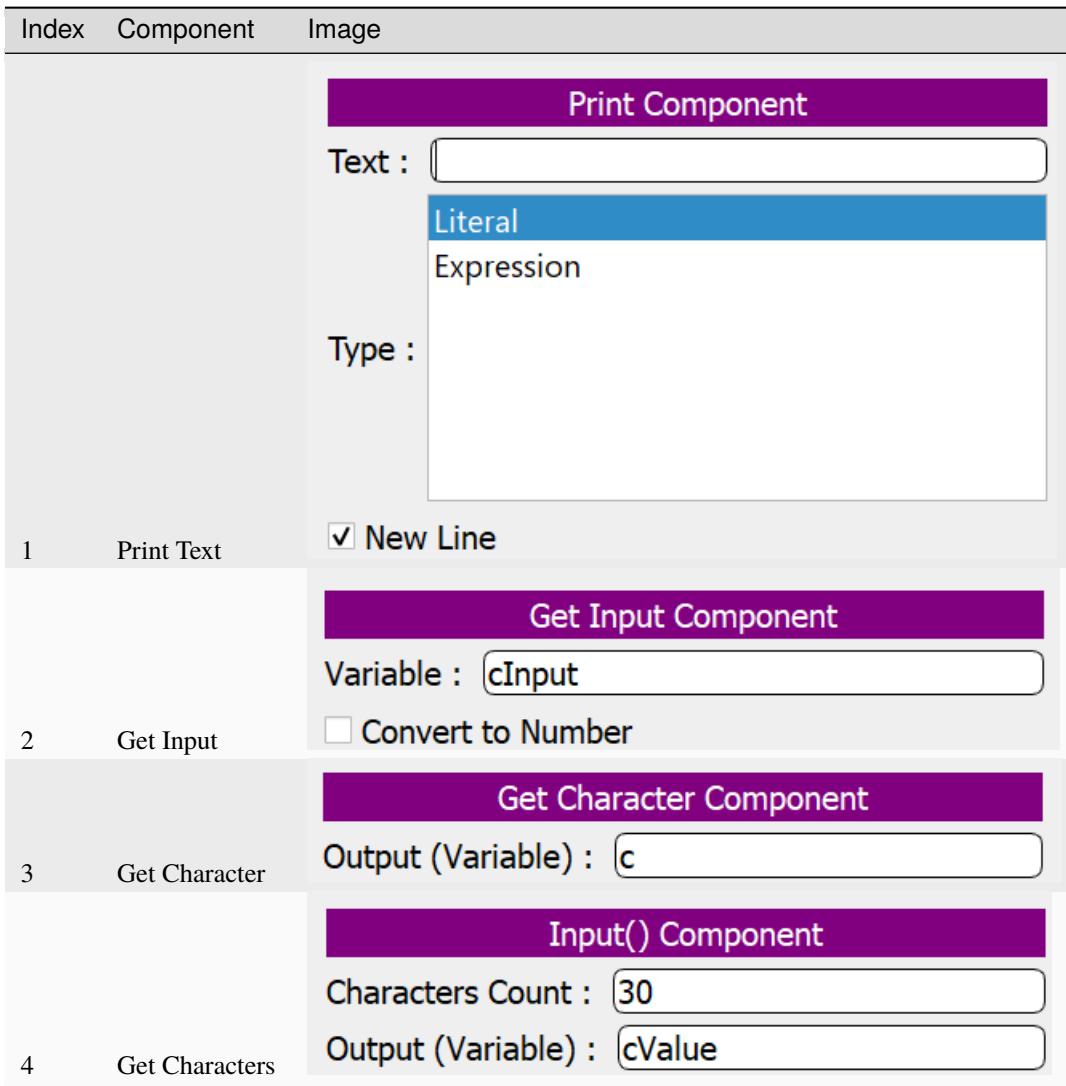
22.1 General/Comments

	Index Component Image
1	<p>Comment (Header) Component</p> <p>Text : <input type="text"/></p> <p>Size : <input type="text" value="3"/></p> <p>Color : <input type="text" value="purple"/></p> <p>Back Color : <input type="text"/></p> <p>Align: Left Center Right</p>
2	<p>Comment - New Line</p> <p>Type: Draw line Empty Line</p>
3	<p>Comment (Image) Component</p> <p>Image File: <input type="text"/></p> <p>Width: <input type="text"/></p> <p>Height: <input type="text"/></p> <p>Align: Left Center Right</p>

22.2 General/Templates

	Index	Component	Image
			
1	Basic Program		<p>The First Step</p> <p>Load Files</p> <p>Statements</p> <p>Functions</p> <p>Classes</p>
			<p>Quick Start Component</p> <p>Hello World</p> <p>Say Hello</p> <p>Variables</p> <p>Deep Copy</p> <p>Implicit Conversion</p> <p>Operators Precedence</p> <p>Loop and Condition</p>
2	Quick Start	Template	<p><input checked="" type="checkbox"/> No Comments</p>

22.3 Console



22.4 Control Structures

Index	Component	Image
1	If	<p>If Statement Component</p> <p>Condition : <input type="text"/></p> <p>- Else</p> <p>Start Here</p>
2	Else	<p>Else If Component</p> <p>Condition : <input type="text"/></p>
3	Else If	<p>Switch Component</p> <p>Variable : <input type="text"/></p>
4	Switch	<p>Case Component</p> <p>Value : <input type="text"/></p>
5	Case	<p>- Else</p> <p>Start Here</p>
6	Else (Other)	<p>For Loop Component</p> <p>Start : <input type="text"/> t = 1</p> <p>To : <input type="text"/> 10</p> <p>Step : <input type="text"/> 1</p>
7	For Loop	<p>For In Component</p> <p>Variable : <input type="text"/> t</p> <p>In : <input type="text"/> aList</p> <p>Step : <input type="text"/> 1</p>
8	For In Loop	<p>While Loop Component</p> <p>Condition : <input type="text"/> True</p>
9	While Loop	<p>Do Again Component</p>
10	Do Again Loop	<p>Condition : <input type="text"/> True</p> <p>- Try</p> <p>Start Here</p>

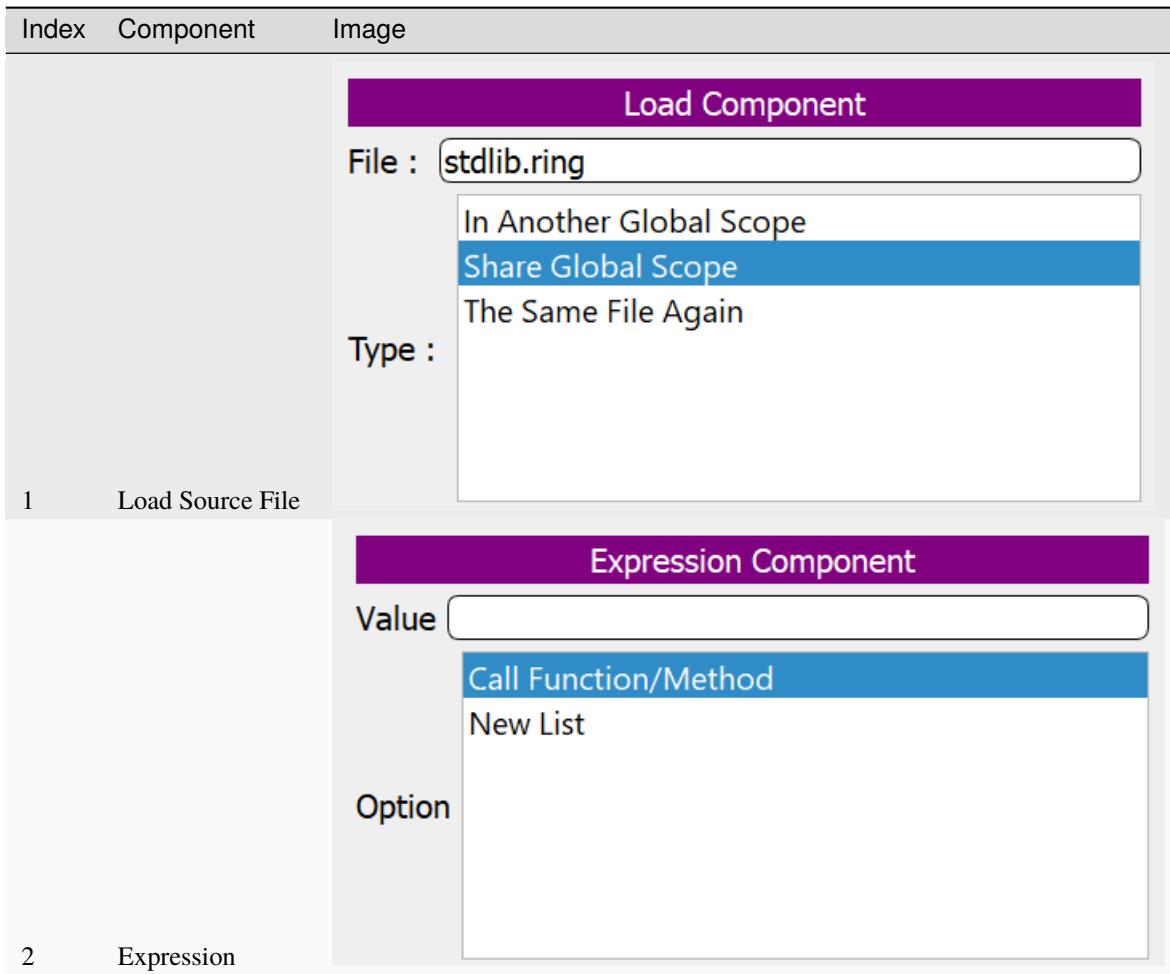
22.5 Variables and Operators

Index	Component	Image
1	Assignment	<p style="text-align: center;">Assignment Component</p> <p>Left Side : <input type="text"/></p> <p>Right Side : <input type="text"/></p>
2	Sum	<p style="text-align: center;">Sum Component</p> <p>Input (1) : <input type="text"/></p> <p>Input (2) : <input type="text"/></p> <p>Output : <input type="text"/></p>
3	Subtract	<p style="text-align: center;">Subtract Component</p> <p>Input (1) : <input type="text"/></p> <p>Input (2) : <input type="text"/></p> <p>Output : <input type="text"/></p>
4	Multiplication	<p style="text-align: center;">Multiplication Component</p> <p>Input (1) : <input type="text"/></p> <p>Input (2) : <input type="text"/></p> <p>Output : <input type="text"/></p>
5	Division	<p style="text-align: center;">Division Component</p> <p>Input (1) : <input type="text"/></p> <p>Input (2) : <input type="text"/></p> <p>Output : <input type="text"/></p>
6	Modulus	<p style="text-align: center;">Modulus Component</p> <p>Input (1) : <input type="text"/></p> <p>Input (2) : <input type="text"/></p> <p>Output : <input type="text"/></p>
22.5. Variables and Operators		366

22.6 Functions

Index	Component	Image
		<p style="text-align: center;">Define Function Component</p> <p>Name : <input type="text"/></p> <p><input type="checkbox"/> Parameters : <input type="text"/></p> <p><input type="checkbox"/> Output : <input type="text"/></p>
1	Define Function	
		<p style="text-align: center;">Call Function Component</p> <p>Name : <input type="text"/></p> <p><input type="checkbox"/> Parameters : <input type="text"/></p> <p><input type="checkbox"/> Output : <input type="text"/></p>
2	Call Function	
		<p style="text-align: center;">Return Component</p> <p>Value : <input type="text"/></p>
3	Return	

22.7 Program Structure



22.8 Lists

Index	Component	Image
1	New Empty List	<p style="text-align: center;">New Empty List Component</p> <p>Variable : <input type="text"/></p>
2	New List By Range	<p style="text-align: center;">New List by Range Component</p> <p>Variable : <input type="text"/></p> <p>From <input type="text"/></p> <p>To <input type="text"/></p>
3	New List By Size	<p style="text-align: center;">New List By Size Component</p> <p>Variable : <input type="text"/></p> <p>Size : <input type="text"/></p>
4	Add Item	<p style="text-align: center;">Add Item Component</p> <p>List : <input type="text"/></p> <p>Value : <input type="text"/></p>
5	Get List Size	<p style="text-align: center;">Get List Size Component</p> <p>List : <input type="text"/></p> <p>Output : <input type="text"/></p>
6	Delete Item	<p style="text-align: center;">Delete Item Component</p> <p>List : <input type="text"/></p> <p>Index : <input type="text"/></p>
7	Get Item From List	<p style="text-align: center;">Get List Item Component</p> <p>List : <input type="text"/></p> <p>Index : <input type="text"/></p> <p>Output : <input type="text"/></p>
8	Set List Item	<p style="text-align: center;">Set List Item Component</p> <p>List : <input type="text"/></p> <p>Item : <input type="text"/></p> <p>Value : <input type="text"/></p>
22.8. Lists		370
<p style="text-align: center;">Reverse List Component</p> <p>Input (List) : <input type="text"/></p>		

22.9 Strings

Index	Component	Image
1	Get String Length	<p style="text-align: center;">Get String Length Component</p> <p>String : <input type="text"/></p> <p>Output : <input type="text"/></p>
2	Lower Case	<p style="text-align: center;">Lower Component</p> <p>String : <input type="text"/></p> <p>Output : <input type="text"/></p>
3	Upper Case	<p style="text-align: center;">Upper Component</p> <p>String : <input type="text"/></p> <p>Output : <input type="text"/></p>
4	Set String Index	<p style="text-align: center;">Set String Index Component</p> <p>String : <input type="text"/></p> <p>Index : <input type="text"/></p> <p>Value : <input type="text"/></p>
5	Get String Index	<p style="text-align: center;">Get String Index Component</p> <p>String : <input type="text"/></p> <p>Index : <input type="text"/></p> <p>Output : <input type="text"/></p>
6	Get Letters From Left	<p style="text-align: center;">Get Characters from Left Component</p> <p>String : <input type="text"/></p> <p>Count : <input type="text"/></p> <p>Output : <input type="text"/></p>
7	Get Letters from Right	<p style="text-align: center;">Get Characters from Right Component</p> <p>String : <input type="text"/></p> <p>Count : <input type="text"/></p> <p>Output : <input type="text"/></p>
8	Trim	<p style="text-align: center;">Trim Component</p> <p>String : <input type="text"/> 372</p> <p>Output : <input type="text"/></p>

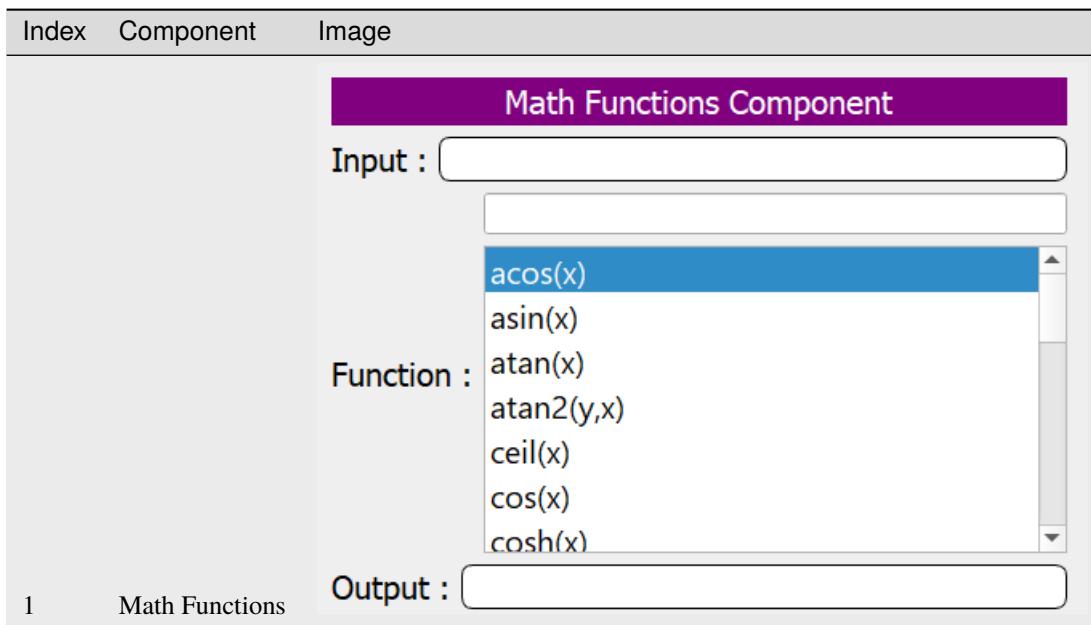
22.10 Date and Time

Index	Component	Image
1	Clock	<p style="text-align: center;">Clock Component</p> <p>Output : <input type="text"/></p>
2	Clocks per second	<p style="text-align: center;">Clocks per second Component</p> <p>Output : <input type="text"/></p>
3	Time	<p style="text-align: center;">Time Component</p> <p>Output : <input type="text"/> cTime</p>
4	Date	<p style="text-align: center;">Date Component</p> <p>Output : <input type="text"/> cDate</p>
5	Time List	<p style="text-align: center;">Time List Component</p> <p>Output : <input type="text"/></p>
6	Add Days	<p style="text-align: center;">Add Days Component</p> <p>Date : <input type="text"/></p> <p>Days : <input type="text"/></p> <p>Output : <input type="text"/></p>
7	Diffdays	<p style="text-align: center;">Diff Days Component</p> <p>Date 1 : <input type="text"/></p> <p>Date 2 : <input type="text"/></p> <p>Output : <input type="text"/></p>

22.11 Check Data Type and Conversion

	Index	Component	Image
	Check Data Type Component		
1	Check Data Type	Value : <input type="text"/>	Function : Get type Is list Is null Is number Is string
	Output : <input type="text"/>		
	Check Character Component		
2	Check Character	Input : <input type="text"/>	Function : Is Alpha Is Alpha Number Is Cntrl Is Digit Is Graph Is Lower Is Print
	Output : <input type="text"/>		
	Conversion Component		
3	Conversion	Input <input type="text"/>	Function Hex to String String to Hex To ASCII To Character To Dec To Hex To Number
	Output <input type="text"/>		

22.12 Math



22.13 Files

Index	Component	Image
1	Read File to String	<p style="text-align: center;">Read File Component</p> <p>File : <input type="text"/></p> <p>Output : <input type="text"/></p>
2	Write File from String	<p style="text-align: center;">Write File From String Component</p> <p>File : <input type="text"/></p> <p>String : <input type="text"/></p>
3	Directory Contents	<p style="text-align: center;">Dir Component</p> <p>Folder : <input type="text"/></p> <p>Output : <input type="text"/></p>
4	Rename File	<p style="text-align: center;">Rename File Component</p> <p>Rename File : <input type="text"/></p> <p>To : <input type="text"/></p>
5	Remove File	<p style="text-align: center;">Remove File Component</p> <p>File Name : <input type="text"/></p>
6	Open File	<p style="text-align: center;">Open File Component</p> <p>File Name : <input type="text"/></p> <p style="margin-left: 150px;">Appends</p> <p style="margin-left: 150px;">Create for Read/Write</p> <p style="margin-left: 150px;">Reading</p> <p style="margin-left: 150px;">Reading & Appending</p> <p style="margin-left: 150px;">Update</p> <p style="margin-left: 150px;">Writing</p> <p>Output (Handle) : <input type="text"/></p>
7	Close File	<p style="text-align: center;">Close File Component</p> <p>File Handle <input type="text"/></p>
8	File Flush	<p style="text-align: center;">File Flush Component</p> <p>File Handle : <input type="text"/></p>

22.14 System

Index	Component	Image
1	Run System Command	<p style="text-align: center;">Run system command Component</p> <p>Command : <input type="text"/></p>
2	Get System Variable	<p style="text-align: center;">Get System Variable Component</p> <p>Variable Name : <input type="text"/></p> <p>Output : <input type="text"/></p>
3	Check Operating System	<p style="text-align: center;">Check Operating System Component</p> <p>Output <input type="text"/></p> <p style="margin-left: 150px;">Operating System</p> <ul style="list-style-type: none"> IsAndroid IsFreeBSD IsLinux IsMSDOS IsMacOSX IsUnix IsWindows
4	Windows New Line	<p style="text-align: center;">Windows New Line Component</p> <p>Output : <input type="text"/></p>
5	Get Active File Name	<p style="text-align: center;">File Name Component</p> <p>Output : <input type="text"/></p>
6	Previous File Name	<p style="text-align: center;">Previous File Name Component</p> <p>Output : <input type="text"/></p>
7	Current Directory	<p style="text-align: center;">Current Directory Component</p> <p>Output : <input type="text"/></p>
8	Get Executable File Name	<p style="text-align: center;">Get Executable File Name Component</p> <p>Output : <input type="text"/></p>
9	Change Directory	<p style="text-align: center;">Change Directory Component</p> <p>Directory : <input type="text"/></p> <p>Output : <input type="text"/></p>

22.15 Dynamic Code and Debugging

Index	Component	Image
		A screenshot of a software interface showing three buttons labeled "Eval Component", "Raise Component", and "Assert Component".
1	Eval	A screenshot of a software interface showing three buttons labeled "Eval Component", "Raise Component", and "Assert Component".
2	Raise	A screenshot of a software interface showing three buttons labeled "Eval Component", "Raise Component", and "Assert Component".
3	Assert	A screenshot of a software interface showing three buttons labeled "Eval Component", "Raise Component", and "Assert Component".

22.16 Database/ODBC

Index	Component	Image
1	ODBC Init	<p style="text-align: center;">ODBC Init Component</p> <p>Handle : <input type="text"/></p>
2	ODBC Drivers	<p style="text-align: center;">ODBC Drivers Component</p> <p>ODBC Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
3	ODBC Data Sources	<p style="text-align: center;">ODBC Data Sources Component</p> <p>ODBC Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
4	ODBC Close	<p style="text-align: center;">ODBC Close Component</p> <p>ODBC Handle : <input type="text"/></p>
5	ODBC Connect	<p style="text-align: center;">ODBC Connect Component</p> <p>ODBC Handle : <input type="text"/></p> <p>Connection String : <input type="text"/></p> <p>Output : <input type="text"/></p>
6	ODBC Disconnect	<p style="text-align: center;">ODBC Disconnect Component</p> <p>ODBC Handle : <input type="text"/></p>
7	ODBC Execute	<p style="text-align: center;">ODBC Execute Component</p> <p>ODBC Handle : <input type="text"/></p> <p>SQL Command : <input type="text"/></p> <p>Output : <input type="text"/></p>
8	ODBC Column Count	<p style="text-align: center;">ODBC Column Count Component</p> <p>ODBC Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
9	ODBC Fetch	<p style="text-align: center;">ODBC Fetch Component</p> <p>ODBC Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
		<p style="text-align: center;">ODBC Get Data Component</p>

22.17 Database/MySQL

Index	Component	Image
1	MySQL Info	<p style="text-align: center;">MySQL Info Component</p> <p>Output : <input type="text"/></p>
2	MySQL Init	<p style="text-align: center;">MySQL Init Component</p> <p>Output : <input type="text"/></p>
3	MySQL Error	<p style="text-align: center;">MySQL Error Component</p> <p>MySQL Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
4	MySQL Connect	<p style="text-align: center;">MySQL Connect Component</p> <p>MySQL Handle : <input type="text"/></p> <p>MySQL Server : <input type="text"/></p> <p>MySQL User Name : <input type="text"/></p> <p>MySQL Password : <input type="text"/></p> <p>Output : <input type="text"/></p>
5	MySQL Close	<p style="text-align: center;">MySQL Close Component</p> <p>MySQL Handle : <input type="text"/></p>
6	MySQL Query	<p style="text-align: center;">MySQL Query Component</p> <p>MySQL Handle : <input type="text"/></p> <p>SQL Query : <input type="text"/></p>
7	MySQL Insert ID	<p style="text-align: center;">MySQL Insert ID Component</p> <p>Output : <input type="text"/></p>
8	MySQL Result	<p style="text-align: center;">MySQL Result Component</p> <p>MySQL Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
22.17. Database/MySQL		<p style="text-align: center;">MySQL Next Result Component</p> <p>MySQL Handle : <input type="text"/></p> <p>Output : <input type="text"/></p>
		<p style="text-align: center;">MySQL Columns Component</p> <p>MySQL Handle : <input type="text"/></p>

22.18 Database/SQLite

Index	Component	Image
1	SQLite Init	<p style="text-align: center;">SQLite Init Component</p> <p>Output : <input type="text"/></p>
2	SQLite Open	<p style="text-align: center;">SQLite Open Component</p> <p>SQLite Handle : <input type="text"/></p> <p>File : <input type="text"/></p>
3	SQLite Execute	<p style="text-align: center;">SQLite Execute Component</p> <p>SQLite Handle : <input type="text"/></p> <p>SQL : <input type="text"/></p>
4	SQLite Close	<p style="text-align: center;">SQLite Close Component</p> <p>SQLite Handle : <input type="text"/></p>

22.19 Security and Internet Functions

Index	Component	Image
1	MD5	<p style="text-align: center;">MD5 Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
2	SHA1	<p style="text-align: center;">SHA1 Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
3	SHA256	<p style="text-align: center;">SHA 256 Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
4	SHA512	<p style="text-align: center;">SHA 512 Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
5	SHA384	<p style="text-align: center;">SHA 384 Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
6	SHA224	<p style="text-align: center;">SHA 224 Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
7	Encrypt	<p style="text-align: center;">Encrypt Component</p> <p>String : <input type="text"/></p> <p>Key : <input type="text"/></p> <p>IV : <input type="text"/></p> <p>Output : <input type="text"/></p>
8	Decrypt	<p style="text-align: center;">Decrypt Component</p> <p>String : <input type="text"/></p> <p>Key : <input type="text"/></p> <p>IV : <input type="text"/></p> <p>Output : <input type="text"/></p>

22.20 Object Oriented Programming

Index	Component	Image
1	Define Class	<p>Class Component</p> <p>Name : <input type="text"/></p> <p>Parent : <input type="text"/></p> <p><input type="checkbox"/> Private Attributes/Methods</p>
2	New Object	<p>New Object Component</p> <p>Object Name : <input type="text"/></p> <p>Class : <input type="text"/></p> <p>Parameters : <input type="text"/></p> <p><input checked="" type="checkbox"/> Access Object using Braces</p> <p><input type="checkbox"/> Call Init() Method</p>
3	Braces	
1	Free Typing	<p>Free Typing Component</p> <p>Code : <input type="text"/></p>
5	Access Object	<p>Access Object Component</p> <p>Object Name : <input type="text"/></p>
6	Define Package	<p>Define Package Component</p> <p>Name : <input type="text"/></p>
7	Import Package	<p>Import Package Component</p> <p>Name : <input type="text"/></p>
		<p>Set Object Attribute Component</p> <p>Object : <input type="text"/></p>

22.21 Functional Programming

Index	Component	Image
1	Call Function by Variable	<p style="text-align: center;">Call Function by Variable Component</p> <p>Variable Name : <input type="text"/></p> <p><input type="checkbox"/> Parameters : <input type="text"/></p> <p><input type="checkbox"/> Output : <input type="text"/></p>
2	Anonymous Function	<p style="text-align: center;">Anonymous Function Component</p> <p>Parameters <input type="text"/></p>
3	Nested Function	<p style="text-align: center;">Nested Function Component</p> <p>Variable : <input type="text"/></p> <p>Parameters : <input type="text"/></p>

22.22 Reflection and Meta-programming

Index	Component	Image
1	Locals Info	<p style="text-align: center;">Locals Component</p> <p>Output : <input type="text"/></p>
2	Globals Info	<p style="text-align: center;">globals Component</p> <p>Output : <input type="text"/></p>
3	Functions Info	<p style="text-align: center;">Functions Component</p> <p>Output : <input type="text"/></p>
4	C Functions Info	<p style="text-align: center;">C Functions Component</p> <p>Output : <input type="text"/></p>
5	Is Local	<p style="text-align: center;">Is Local Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
6	Is Global	<p style="text-align: center;">Is Global Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
7	Is Function	<p style="text-align: center;">Is Function Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
8	Is C Function	<p style="text-align: center;">Is C Function Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
9	Packages Info	<p style="text-align: center;">Packages Component</p> <p>Output : <input type="text"/></p>
10	Is Package	<p style="text-align: center;">Is Package Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
	22.22. Reflection and Meta-programming	<p style="text-align: center;">Classes Component</p> <p>Output : <input type="text"/></p>

22.23 StdLib/Functions

Index	Component	Image
1	SL Puts	<p>Puts Component</p> <p>Expression : <input "="" style="width: 150px; height: 20px; border: 1px solid black;" type="text" value="\"/></p>
2	SL Print	<p>Print Component</p> <p>String : <input "="" style="width: 150px; height: 20px; border: 1px solid black;" type="text" value="\"/></p>
3	SL Print to String	<p>Print to String Component</p> <p>String : <input "="" style="width: 150px; height: 20px; border: 1px solid black;" type="text" value="\"/></p> <p>Output : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>
4	SL Get String	<p>Get String Component</p> <p>Output : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>
5	SL Get Number	<p>Get Number Component</p> <p>Output : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>
6	SL Application Path	<p>Get Application Path Component</p> <p>Output : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>
7	SL Just File Path	<p>Just File Path Component</p> <p>File : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p> <p>Output : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>
8	SL Just File Name	<p>Just File Name Component</p> <p>File : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p> <p>Output : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>
9	SL Times	<p>Times Component</p> <p>Count : <input style="width: 150px; height: 20px; border: 1px solid black;" type="text"/></p>

continues on next page

Table 1 – continued from previous page

Index	Component	Image
10	SL Map	<p style="text-align: center;">Map Component</p> <p>List : <input type="text"/></p> <p>Variable : <input type="text"/></p> <p>Output : <input type="text"/></p>
11	SL Filter	<p style="text-align: center;">Filter Component</p> <p>List : <input type="text"/></p> <p>Variable : <input type="text"/></p> <p>Output : <input type="text"/></p>
12	SL Split	<p style="text-align: center;">Split Component</p> <p>String : <input type="text"/></p> <p>Delimiter : <input type="text"/></p> <p>Output : <input type="text"/></p>
13	SL Split Many	<p style="text-align: center;">Split Many Component</p> <p>String : <input type="text"/></p> <p>Delimiters : <input type="text"/></p> <p>Output : <input type="text"/></p>
14	SL New List	<p style="text-align: center;">New List Component</p> <p>Rows : <input type="text"/></p> <p>Columns : <input type="text"/></p> <p>Output : <input type="text"/></p>
15	SL Capitalized	<p style="text-align: center;">Capitalized Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
16	SL Is Special	<p style="text-align: center;">Is Special Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>

continues on next page

Table 1 – continued from previous page

Index	Component	Image
17	SL Is Vowel	<p style="text-align: center;">Is Vowel Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
18	SL Line Count	<p style="text-align: center;">Line Count Component</p> <p>File : <input type="text"/></p> <p>Output : <input type="text"/></p>
19	SL Factorial	<p style="text-align: center;">Factorial Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
20	SL Fibonacci	<p style="text-align: center;">Fibonacci Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
21	SL Is Prime	<p style="text-align: center;">Is Prime Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
22	SL Sign	<p style="text-align: center;">Sign Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
23	SL List to File	<p style="text-align: center;">List to File Component</p> <p>List : <input type="text"/></p> <p>File : <input type="text"/></p>
24	SL File to List	<p style="text-align: center;">File To List Component</p> <p>File : <input type="text"/></p> <p>Output : <input type="text"/></p>

continues on next page

Table 1 – continued from previous page

Index	Component	Image
25	SL Starts With	<p style="text-align: center;">Starts With Component</p> <p>String : <input type="text"/></p> <p>Sub String : <input type="text"/></p> <p>Output : <input type="text"/></p>
26	SL Ends With	<p style="text-align: center;">Ends With Component</p> <p>String : <input type="text"/></p> <p>Sub String : <input type="text"/></p> <p>Output : <input type="text"/></p>
27	SL Greatest Common Divisor	<p style="text-align: center;">Greatest Common Divisor Component</p> <p>Number 1 : <input type="text"/></p> <p>Number 2 : <input type="text"/></p> <p>Output : <input type="text"/></p>
28	SL Least Common Multiple	<p style="text-align: center;">Least Common Multiple Component</p> <p>Number 1 : <input type="text"/></p> <p>Number 2 : <input type="text"/></p> <p>Output : <input type="text"/></p>
29	SL Sum List	<p style="text-align: center;">Sum List Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
30	SL Product of a List	<p style="text-align: center;">Product of a List Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>
31	SL Even or Odd	<p style="text-align: center;">Even or Odd Component</p> <p>Input : <input type="text"/></p> <p>Output : <input type="text"/></p>

continues on next page

Table 1 – continued from previous page

Index	Component	Image
32	SL Factors	<p style="text-align: center;">Factors Component</p> <p>Number : <input type="text"/></p> <p>Output : <input type="text"/></p>
33	SL Is Palindrome	<p style="text-align: center;">Is Palindrome Component</p> <p>String : <input type="text"/></p> <p>Output : <input type="text"/></p>
34	SL Is Leap Year	<p style="text-align: center;">Is Leap Year Component</p> <p>Year : <input type="text"/></p> <p>Output : <input type="text"/></p>
35	SL Binary Digits	<p style="text-align: center;">Binary Digits Component</p> <p>Number : <input type="text"/></p> <p>Output : <input type="text"/></p>
36	SL Matrix Multiply	<p style="text-align: center;">Matrix Multiply Component</p> <p>List 1 : <input type="text"/></p> <p>List 2 : <input type="text"/></p> <p>Output : <input type="text"/></p>
37	SL Matrix Transpose	<p style="text-align: center;">Matrix Transpose Component</p> <p>List : <input type="text"/></p> <p>Output : <input type="text"/></p>
38	SL Day of Week	<p style="text-align: center;">Day of Week Component</p> <p>Date : <input type="text"/></p> <p>Output : <input type="text"/></p>
39	SL Permutation	<p style="text-align: center;">Permutation Component</p> <p>List : <input type="text"/></p>

continues on next page

Table 1 – continued from previous page

Index	Component	Image
40	SL Read Line	<p style="text-align: center;">Read Line Component</p> <p>File Object : <input type="text"/></p> <p>Output : <input type="text"/></p>
41	SL Sub String Position	<p style="text-align: center;">Substring Position Component</p> <p>String : <input type="text"/></p> <p>Sub String : <input type="text"/></p> <p>Start from Position : <input type="text"/></p> <p>Output : <input type="text"/></p>
42	SL Change Sub String	<p style="text-align: center;">Change Sub String Component</p> <p>String : <input type="text"/></p> <p>Pos 1 : <input type="text"/></p> <p>Pos 2 : <input type="text"/></p> <p>New Sub String : <input type="text"/></p> <p>Output : <input type="text"/></p>
43	SL Sleep	<p style="text-align: center;">Sleep Component</p> <p>Seconds : <input type="text"/></p>
44	SL Is Main Source File	<p style="text-align: center;">Is Main Source File Component</p> <p>Output : <input type="text"/></p>
45	SL Directory Exists	<p style="text-align: center;">Directory Exists Component</p> <p>Path : <input type="text"/></p> <p>Output : <input type="text"/></p>
46	SL Make Directory	<p style="text-align: center;">Make Directory Component</p> <p>Name : <input type="text"/></p>

continues on next page

Table 1 – continued from previous page

Index	Component	Image
47	SL File Size	<p style="text-align: center;">File Size Component</p> <p>File Object : <input type="text"/></p> <p>Output : <input type="text"/></p>
48	SL Trim	<p style="text-align: center;">Trim Component</p> <p>Input : <input type="text"/></p> <p style="text-align: center;">Trim All Trim Left Trim Right</p> <p>Function : <input type="text"/></p> <p>Output : <input type="text"/></p>
49	SL Epoch Time	<p style="text-align: center;">Epoch Time Component</p> <p>Date : <input type="text"/></p> <p>Time : <input type="text"/></p> <p>Output : <input type="text"/></p>
50	SL System Command	<p style="text-align: center;">System Command Component</p> <p>Command : <input type="text"/></p> <p>Output : <input type="text"/></p>
51	SL List All Files	<p style="text-align: center;">List All Files Component</p> <p>Folder : <input type="text"/></p> <p>Extension : <input type="text"/></p> <p>Output : <input type="text"/></p>

22.24 StdLib/Classes

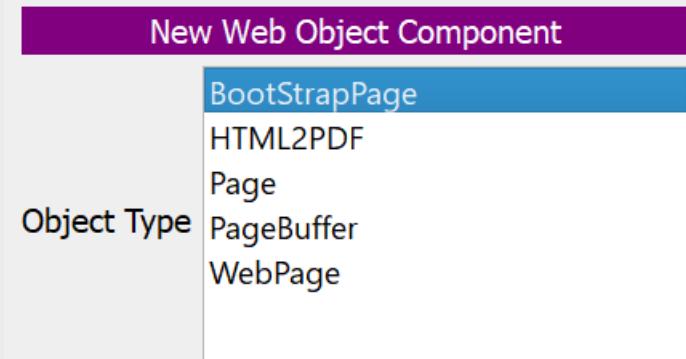
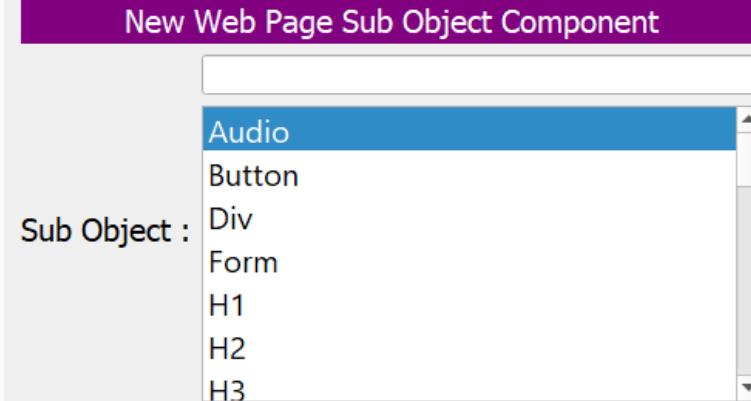
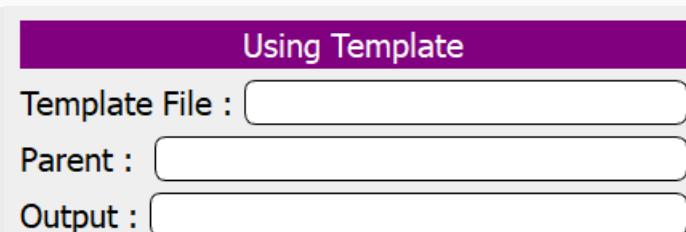
In-	Component	Image
Index		

StdBase Class Component	
Object	
Method	<ul style="list-style-type: none"> init(x) print() println() set(vValue) size() --> Number value() --> vValue
Parameters	
Output	
1 StdBase Class	

String Class Component	
Object	
Method	<ul style="list-style-type: none"> copy(nCount) --> String (Object) endswith(cSubString) --> Boolean getfrom(nPos1) init(string number list) left(nCount) --> String (Object) lines() --> Number lower() --> String (Object)
Parameters	
Output	
2 String Class	

List Class Component	
Object	
Method	<ul style="list-style-type: none"> add(vValue) delete(nIndex) find(vValue) --> Number findincolumn(nColumn,vValue) --> Number first() --> vItem init(String List)

22.25 WebLib/General

In- dex	Component	Image
1	WL Start Web Application	
2	WL New Web Page	
3	WL New Web Object	
4	WL New Web Page Sub Ob- ject	
5	WL Using Template	

22.26 WebLib/Classes

In-	Component	Image
Index		

Application Class Component		
Object	<input type="text"/>	
Method	<input type="text"/> cookie(cName,cValue) decode(cInput) --> List decodestring(cString) --> List getcookies() --> List getfilename(aArray,cVar) --> String gethtmlstart() --> String	
Parameters	<input type="text"/>	
Output	<input type="text"/>	
Page Class Component		
Object	<input type="text"/>	
Method	<input type="text"/> addattributes(aPara) audio(aPara) boxend() boxstart() braceend() button(aPara)	
Parameters	<input type="text"/>	
Output	<input type="text"/>	

WebPage Class Component		
Object	<input type="text"/>	
Method	<input type="text"/> braceend() cookie(cName,cValue) decode(cInput) --> List decodestring(cString) --> List getaudio() --> Object getbutton() --> Object	

22.27 LibCurl

In- dex	Component	Image
1	LibCurl Easy Init	<p style="text-align: center;">LibCurl - Easy Init Component</p> <p>Object : <input type="text"/></p>
2	LibCurl Easy Perform	<p style="text-align: center;">LibCurl - Easy Perform Component</p> <p>Object : <input type="text"/></p>
3	LibCurl Easy Cleanup	<p style="text-align: center;">LibCurl - Easy Cleanup Component</p> <p>Object : <input type="text"/></p>
4	LibCurl Easy Perform Silent	<p style="text-align: center;">LibCurl - Easy Perform Silent Component</p> <p>Object : <input type="text"/></p> <p>Output : <input type="text"/></p>
5	LibCurl Easy Set Option	<p style="text-align: center;">LibCurl Set Option Class Component</p> <p>Object : <input type="text"/></p> <p>Option : <input type="text"/></p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;"> CURLOPT_ACCEPT_ENCODING CURLOPT_ADDRESS_SCOPE CURLOPT_APPEND CURLOPT_AUTOREFERER CURLOPT_BUFFERSIZE CURLOPT_CAINFO </div> <p>Value : <input type="text"/></p>

22.28 GUI/Add Components

Index	Component	Image
1	new Application	<p style="text-align: center;">New GUI Application Component</p> <p>Object Name : oApp</p>
2	new Window	<p style="text-align: center;">Window Component</p> <p>Object Name : win</p> <p>Title : ""</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 400</p> <p>Height : 400</p> <p>Style Sheet : ""</p>
3	new Label	<p style="text-align: center;">Label Component</p> <p>Object Name : lbl</p> <p>Text : ""</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 100</p> <p>Height : 30</p> <p>Style Sheet: ""</p> <p>Parent : win</p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
4	new Button	<p style="text-align: center;">Button Component</p> <p>Object Name : <code>btn</code></p> <p>Text : <code>" "</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p> <p>Click Event : <code>" "</code></p>
5	new LineEdit	<p style="text-align: center;">LineEdit Component</p> <p>Object Name : <code>LineEdit</code></p> <p>Text : <code>" "</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image																
6	newTextEdit	<p style="text-align: center;">TextEdit Component</p> <table border="1"> <tr><td>Object Name :</td><td>TextEdit</td></tr> <tr><td>Text :</td><td>" "</td></tr> <tr><td>Top :</td><td>10</td></tr> <tr><td>Left :</td><td>10</td></tr> <tr><td>Width :</td><td>100</td></tr> <tr><td>Height :</td><td>100</td></tr> <tr><td>Style Sheet:</td><td>" "</td></tr> <tr><td>Parent :</td><td>win</td></tr> </table>	Object Name :	TextEdit	Text :	" "	Top :	10	Left :	10	Width :	100	Height :	100	Style Sheet:	" "	Parent :	win
Object Name :	TextEdit																	
Text :	" "																	
Top :	10																	
Left :	10																	
Width :	100																	
Height :	100																	
Style Sheet:	" "																	
Parent :	win																	
7	newListWidget	<p style="text-align: center;">ListWidget Component</p> <table border="1"> <tr><td>Object Name :</td><td>list</td></tr> <tr><td>Items :</td><td>[]</td></tr> <tr><td>Top :</td><td>10</td></tr> <tr><td>Left :</td><td>10</td></tr> <tr><td>Width :</td><td>100</td></tr> <tr><td>Height :</td><td>100</td></tr> <tr><td>Style Sheet:</td><td>" "</td></tr> <tr><td>Parent :</td><td>win</td></tr> </table>	Object Name :	list	Items :	[]	Top :	10	Left :	10	Width :	100	Height :	100	Style Sheet:	" "	Parent :	win
Object Name :	list																	
Items :	[]																	
Top :	10																	
Left :	10																	
Width :	100																	
Height :	100																	
Style Sheet:	" "																	
Parent :	win																	
8	newTreeview	<p style="text-align: center;">TreeView Component</p> <table border="1"> <tr><td>Object Name :</td><td>tree</td></tr> <tr><td>Top :</td><td>10</td></tr> <tr><td>Left :</td><td>10</td></tr> <tr><td>Width :</td><td>100</td></tr> <tr><td>Height :</td><td>100</td></tr> <tr><td>Style Sheet:</td><td>" "</td></tr> <tr><td>Parent :</td><td>win</td></tr> </table>	Object Name :	tree	Top :	10	Left :	10	Width :	100	Height :	100	Style Sheet:	" "	Parent :	win		
Object Name :	tree																	
Top :	10																	
Left :	10																	
Width :	100																	
Height :	100																	
Style Sheet:	" "																	
Parent :	win																	

continues on next page

Table 2 – continued from previous page

Index	Component	Image
9	new Treewidget	<p style="text-align: center;">TreeWidget Component</p> <p>Object Name : <code>tree</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>100</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>
10	new Combobox	<p style="text-align: center;">Combobox Component</p> <p>Object Name : <code>combo</code></p> <p>Items : <code>[]</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>
11	new Tab	<p style="text-align: center;">Tab Component</p> <p>Object Name : <code>tab</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>100</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
12	new TableWidget	<p style="text-align: center;">Table Component</p> <p>Object Name : <code>table</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>100</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>
13	new Progressbar	<p style="text-align: center;">Progress Bar Component</p> <p>Object Name : <code>progressbar</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>
14	new Spinbox	<p style="text-align: center;">Spinbox Component</p> <p>Object Name : <code>spinbox</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
15	new Slider	<p style="text-align: center;">Slider Component</p> <p>Object Name : slider</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 30</p> <p>Height : 100</p> <p>Style Sheet: " "</p> <p>Parent : win</p>
16	new DateEdit	<p style="text-align: center;">DateEdit Component</p> <p>Object Name : dateedit</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 100</p> <p>Height : 30</p> <p>Style Sheet: " "</p> <p>Parent : win</p>
17	new Dial	<p style="text-align: center;">Dial Component</p> <p>Object Name : dial</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 100</p> <p>Height : 100</p> <p>Style Sheet: " "</p> <p>Parent : win</p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
18	new Webview	<p style="text-align: center;">Web View Component</p> <p>Object Name : <code>webview</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>100</code></p> <p>Height : <code>100</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>
19	new Checkbox	<p style="text-align: center;">Checkbox Component</p> <p>Object Name : <code>checkbox</code></p> <p>Text : <code>" "</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>150</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>
20	new Radiobutton	<p style="text-align: center;">Radiobutton Component</p> <p>Object Name : <code>radiobutton</code></p> <p>Text : <code>" "</code></p> <p>Top : <code>10</code></p> <p>Left : <code>10</code></p> <p>Width : <code>150</code></p> <p>Height : <code>30</code></p> <p>Style Sheet: <code>" "</code></p> <p>Parent : <code>win</code></p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
21	new Buttongroup	<p style="text-align: center;">Button Group Component</p> <p>Object Name : <input type="text" value="buttongroup"/></p> <p>Parent : <input type="text" value="win"/></p>
22	new Hyperlink	<p style="text-align: center;">Hyperlink Component</p> <p>Object Name : <input type="text" value="hyperlink"/></p> <p>Top : <input type="text" value="10"/></p> <p>Left : <input type="text" value="10"/></p> <p>Width : <input type="text" value="100"/></p> <p>Height : <input type="text" value="30"/></p> <p>Style Sheet: <input "="" type="text" value="\"/></p> <p>Parent : <input type="text" value="win"/></p> <p>Text : <input "="" type="text" value="\"/></p> <p>URL : <input "="" type="text" value="\"/></p>
23	new Video	<p style="text-align: center;">Video Widget Component</p> <p>Object Name : <input type="text" value="video"/></p> <p>Top : <input type="text" value="10"/></p> <p>Left : <input type="text" value="10"/></p> <p>Width : <input type="text" value="100"/></p> <p>Height : <input type="text" value="100"/></p> <p>Style Sheet: <input "="" type="text" value="\"/></p> <p>Parent : <input type="text" value="win"/></p>
24	new Media Player	<p style="text-align: center;">Media Player Component</p> <p>Name : <input type="text"/></p> <p>Media : <input type="text"/></p> <p>Video Widget : <input type="text"/></p> <p>Position : <input type="text"/></p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
25	new Frame	<p style="text-align: center;">Frame Component</p> <p>Object Name : frame</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 100</p> <p>Height : 100</p> <p>Style Sheet: " "</p> <p>Parent : win</p>
26	new Image	<p style="text-align: center;">Image Component</p> <p>Object Name : image</p> <p>Top : 10</p> <p>Left : 10</p> <p>Width : 100</p> <p>Height : 100</p> <p>Style Sheet: " "</p> <p>Parent : win</p> <p>Image File : " "</p>
27	new Timer	<p style="text-align: center;">Timer Component</p> <p>Name : timer</p> <p>Parent : win</p> <p>Interval : 1000</p> <p>Function : " "</p> <p><input type="checkbox"/> Start</p>
28	new Menubar	<p style="text-align: center;">New Menubar Component</p> <p>Name : menu</p> <p>Parent : win</p>

continues on next page

Table 2 – continued from previous page

Index	Component	Image
29	new Menu	<p style="text-align: center;">New Menu Component</p> <p>Name : <input type="text" value="sub"/></p> <p>Text : <input "="" style="width: 200px; height: 20px;" type="text" value=" "/></p>
30	new Menu Item	<p style="text-align: center;">New Menu Item Component</p> <p>Name : <input type="text" value="item"/></p> <p>parent : <input type="text" value="win"/></p> <p>Text : <input "="" style="width: 200px; height: 20px;" type="text" value=" "/></p> <p>event : <input "="" style="width: 200px; height: 20px;" type="text" value=" "/></p>
31	new Menu Separator	<p style="text-align: center;">New Menu Separator</p>
32	new Toolbar	<p style="text-align: center;">Toolbar Component</p> <p>Object Name : <input type="text" value="toolbar"/></p> <p>Top : <input type="text" value="10"/></p> <p>Left : <input type="text" value="10"/></p> <p>Width : <input type="text" value="100"/></p> <p>Height : <input type="text" value="30"/></p> <p>Style Sheet: <input "="" style="width: 200px; height: 20px;" type="text" value=" "/></p> <p>Parent : <input type="text" value="win"/></p>
33	new Statusbar	<p style="text-align: center;">Statusbar Component</p> <p>Name: <input type="text" value="status"/></p> <p>Parent: <input type="text" value="win"/></p>

22.29 GUI/Classes

Index	Component	Image
1	Application Class	<p style="text-align: center;">Application Class Component</p> <p>Object <input type="text" value="Object"/></p> <p>Method <input type="text" value="Method"/></p> <pre>addLibraryPath(QString path) --> void allWindows(void) --> QWindowList applicationDirPath(void) --> QString applicationDisplayName(void) --> QString applicationFilePath(void) --> QString applicationName(void) --> QString</pre> <p>Parameters <input type="text" value="Parameters"/></p> <p>Output <input type="text" value="Output"/></p>
2	Window Class	<p style="text-align: center;">Window Class Component</p> <p>Object <input type="text" value="win"/></p> <p>Method <input type="text" value="Method"/></p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters <input type="text" value="Parameters"/></p> <p>Output <input type="text" value="Output"/></p>

cont

Table 3 – continued from previous page

Index	Component	Image
3	Label Class	<p style="text-align: center;">Label Class Component</p> <p>Object Label1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>
4	Button Class	<p style="text-align: center;">Button Class Component</p> <p>Object Button1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
5	Vertical Layout Class	<p style="text-align: center;">Vertical Layout Class Component</p> <p>Object <code>Layout1</code></p> <p>Method</p> <pre>activate(void) --> bool addLayout(QLayout *) --> void addSpacerItem(QSpacerItem *spacerItem) --> void addSpacing(int size) --> void addStretch(int stretch) --> void addStrut(int size) --> void</pre> <p>Parameters</p> <p>Output</p>
6	Horizontal Layout Class	<p style="text-align: center;">Horizontal Layout Class Component</p> <p>Object <code>Layout1</code></p> <p>Method</p> <pre>activate(void) --> bool addLayout(QLayout *) --> void addSpacerItem(QSpacerItem *spacerItem) --> addSpacing(int size) --> void addStretch(int stretch) --> void addStrut(int size) --> void</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
7	LineEdit Class	<p style="text-align: center;">LineEdit Class Component</p> <p>Object LineEdit1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>
8	TextEdit Class	<p style="text-align: center;">TextEdit Class Component</p> <p>Object TextEdit1</p> <p>Method</p> <pre>acceptDrops(void) --> bool acceptRichText(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
9	ListView Class	<p style="text-align: center;">ListView Class Component</p> <p>Object ListView1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void addItem(QString) --> void</pre> <p>Parameters</p> <p>Output</p>
10	Checkbox Class	<p style="text-align: center;">Checkbox Class Component</p> <p>Object Checkbox1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
11	Slider Class	<p style="text-align: center;">Slider Class Component</p> <p>Object <code>Slider1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>
12	Progressbar Class	<p style="text-align: center;">Progressbar Class Component</p> <p>Object <code>Progressbar1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
13	Spinbox Class	<p style="text-align: center;">Spinbox Class Component</p> <p>Object Spinbox1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>
14	Combobox Class	<p style="text-align: center;">Combobox Class Component</p> <p>Object Combobox1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void addItem(QString,int) --> void</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
15	DateTimeEdit Class	<p style="text-align: center;">DateTimeEdit Class Component</p> <p>Object <code>DateTimeEdit1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>
16	TableWidget Class	<p style="text-align: center;">TableWidget Class Component</p> <p>Object <code>TableWidget1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void addScrollBarWidget(QWidget *widget, Qt::AlignmentFlag alignment)</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
17	TreeWidget Class	<p>TreeWidget Class Component</p> <p>Object <code>TreeWidget1</code></p> <p>Method</p> <ul style="list-style-type: none">acceptDrops(void) --> boolaccessibleDescription(void) --> QStringaccessibleName(void) --> QStringactivateWindow(void) --> voidaddAction(QAction *action) --> voidaddScrollBarWidget(QWidget *widget, Qt::AlignmentFlag alignment) <p>Parameters</p> <p>Output</p>
18	RadioButton Class	<p>RadioButton Class Component</p> <p>Object <code>RadioButton1</code></p> <p>Method</p> <ul style="list-style-type: none">acceptDrops(void) --> boolaccessibleDescription(void) --> QStringaccessibleName(void) --> QStringactivateWindow(void) --> voidaddAction(QAction *action) --> voidadjustSize(void) --> void <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
19	WebView Class	<p style="text-align: center;">WebView Class Component</p> <p>Object <code>WebView1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>
20	DialSlider Class	<p style="text-align: center;">DialSlider Class Component</p> <p>Object <code>Dial1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p>

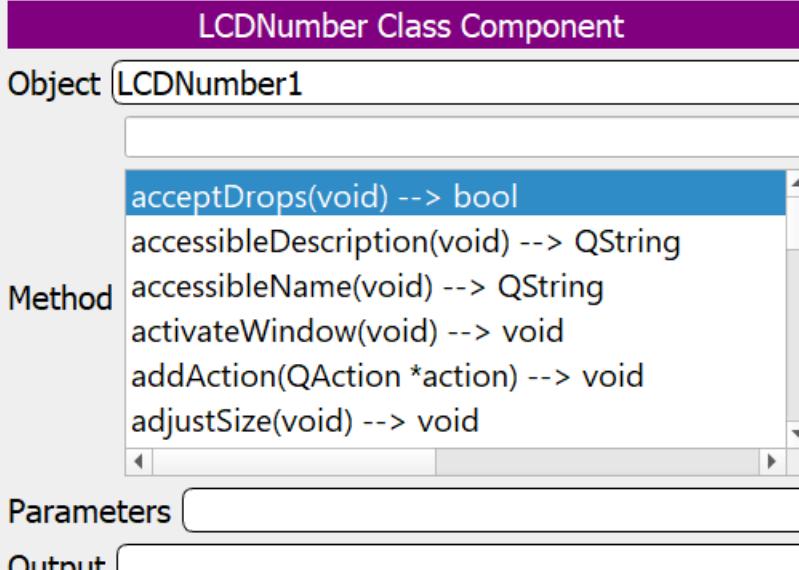
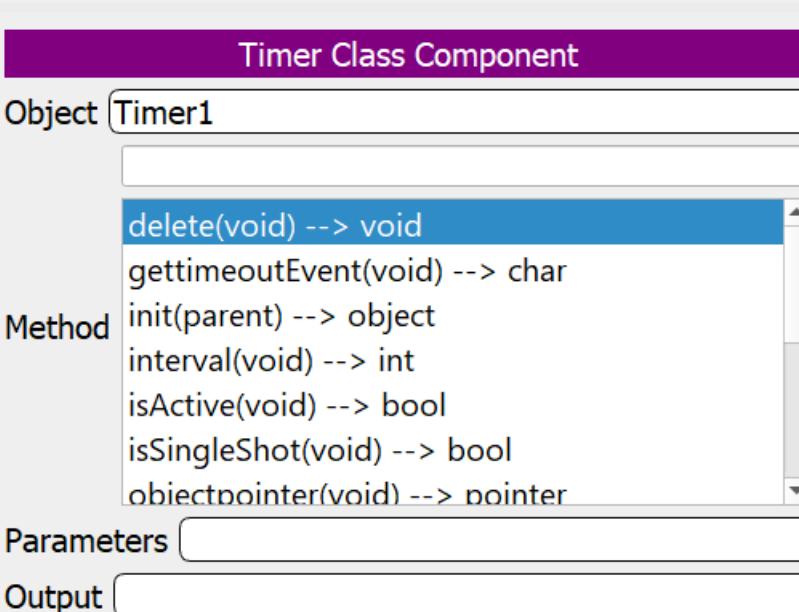
cont

Table 3 – continued from previous page

Index	Component	Image
21	VideoWidget Class	<p style="text-align: center;">VideoWidget Class Component</p> <p>Object <code>VideoWidget1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>
22	Frame Class	<p style="text-align: center;">Frame Class Component</p> <p>Object <code>Frame1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

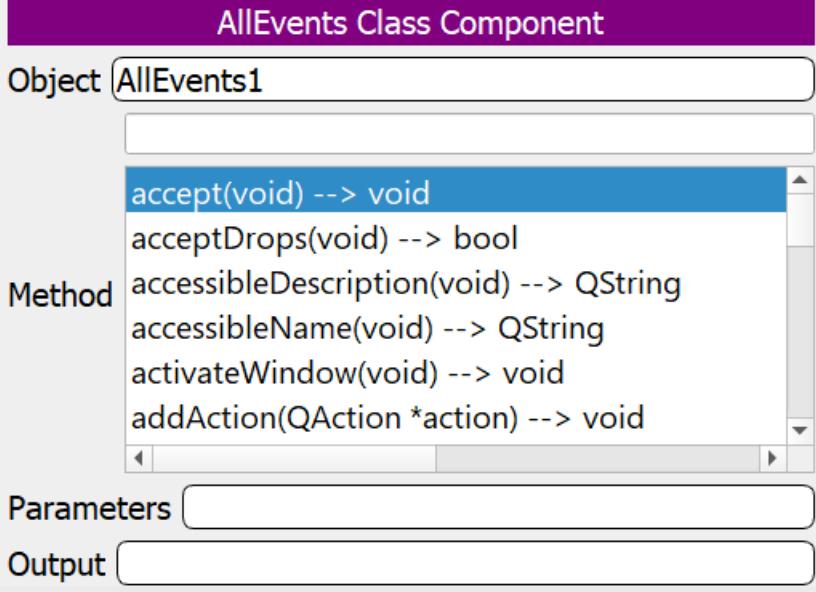
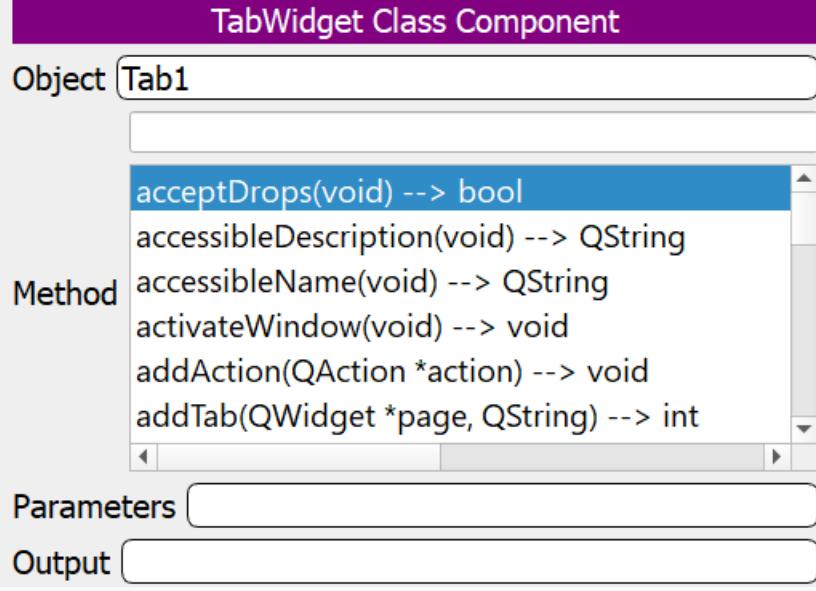
cont

Table 3 – continued from previous page

Index	Component	Image
23	LCDNumber Class	<p style="text-align: center;">LCDNumber Class Component</p> <p>Object <code>LCDNumber1</code></p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void</pre> <p>Parameters</p> <p>Output</p> 
24	Timer Class	<p style="text-align: center;">Timer Class Component</p> <p>Object <code>Timer1</code></p> <p>Method</p> <pre>delete(void) --> void gettimeoutEvent(void) --> char init(parent) --> object interval(void) --> int isActive(void) --> bool isSingleShot(void) --> bool objectpointer(void) --> pointer</pre> <p>Parameters</p> <p>Output</p> 

cont

Table 3 – continued from previous page

Index	Component	Image
25	AllEvents Class	<p style="text-align: center;">AllEvents Class Component</p> <p>Object AllEvents1</p> <p>Method</p> <pre>accept(void) --> void acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void</pre> <p>Parameters</p> <p>Output</p> 
26	TabWidget Class	<p style="text-align: center;">TabWidget Class Component</p> <p>Object Tab1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void addTab(QWidget *page, QString) --> int</pre> <p>Parameters</p> <p>Output</p> 

cont

Table 3 – continued from previous page

Index	Component	Image
27	Statusbar Class	<p style="text-align: center;">Statusbar Class Component</p> <p>Object Statusbar1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void addPermanentWidget(QWidget *widget, int stretch) --> void</pre> <p>Parameters</p> <p>Output</p>
28	Toolbar Class	<p style="text-align: center;">Toolbar Class Component</p> <p>Object Toolbar1</p> <p>Method</p> <pre>acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString actionAt(int x, int y) --> QAction activateWindow(void) --> void addAction(QString) --> QAction</pre> <p>Parameters</p> <p>Output</p>

cont

Table 3 – continued from previous page

Index	Component	Image
29	DockWidget Class	<p style="text-align: center;">DockWidget Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>
30	DesktopWidget Class	<p style="text-align: center;">DesktopWidget Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void adjustSize(void) --> void <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

cont

Table 3 – continued from previous page

Index	Component	Image
31	PixMap Class	<p style="text-align: center;">PixMap Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <pre>cacheKey(void) --> qint64 convertFromImage(QImage image, Qt::ImageConversionFlags flags) copy(int x, int y, int width, int height) --> QPixmap copy_2.QRect rectangle) --> QPixmap createHeuristicMask(bool clipTight) --> QBitmap createMaskFromColor(QColor , Qt::MaskMode) --> QBitmap</pre> <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>
32	Painter Class	<p style="text-align: center;">QPainter Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <pre>background(void) --> QBrush backgroundMode(void) --> int begin(QPaintDevice *device) --> bool beginNativePainting(void) --> void boundingRect(int x, int y, int w, int h, int flags, QString text) --> QRect brush(void) --> QBrush</pre> <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

cont

Table 3 – continued from previous page

Index	Component	Image
33	Picture Class	<p style="text-align: center;">QPicture Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> boundingRect(void) --> QRect data(void) --> char delete(void) --> void init(parent) --> object isNull(void) --> bool loadfile(QString, char *format) --> bool objectpointer(void) --> pointer <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>
34	Pen Class	<p style="text-align: center;">QPen Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> brush(void) --> QBrush capStyle(void) --> int color(void) --> QColor dashOffset(void) --> double delete(void) --> void init(parent) --> object isCosmetic(void) --> bool <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

cont

Table 3 – continued from previous page

Index	Component	Image
35	Printer Class	<p style="text-align: center;">QPrinter Class Component</p> <p>Object []</p> <p>Method</p> <ul style="list-style-type: none"> abort(void) --> bool collateCopies(void) --> bool colorMode(void) --> int copyCount(void) --> int creator(void) --> QString delete(void) --> void <p>Parameters []</p> <p>Output []</p>
36	PrintPreviewDialog Class	<p style="text-align: center;">QPrintPreviewDialog Class Component</p> <p>Object []</p> <p>Method</p> <ul style="list-style-type: none"> accept(void) --> void acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void <p>Parameters []</p> <p>Output []</p>

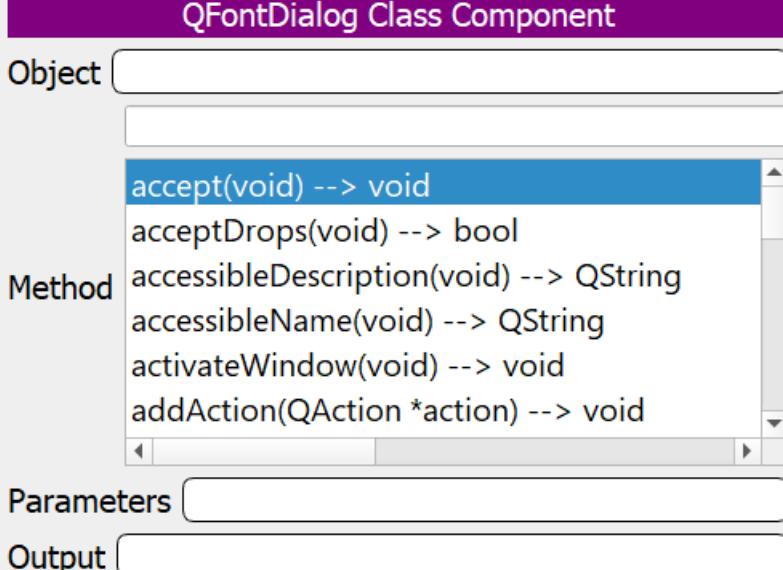
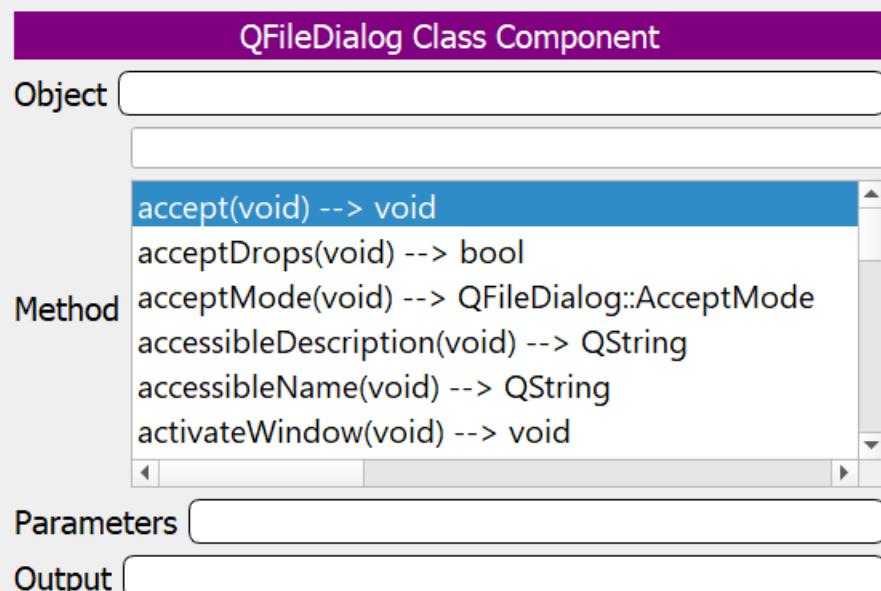
cont

Table 3 – continued from previous page

Index	Component	Image
37	MediaPlayer Class	<p style="text-align: center;">QMediaPlayer Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> <code>bufferStatus(void) --> int</code> <code>currentMedia(void) --> QMediaContent</code> <code>currentNetworkConfiguration(void) --> QNetworkConfiguration</code> <code>delete(void) --> void</code> <code>duration(void) --> int</code> <code>error(void) --> int</code> <code>errorString(void) --> QString</code> <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>
38	ColorDialog Class	<p style="text-align: center;">QColorDialog Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> <code>accept(void) --> void</code> <code>acceptDrops(void) --> bool</code> <code>accessibleDescription(void) --> QString</code> <code>accessibleName(void) --> QString</code> <code>activateWindow(void) --> void</code> <code>addAction(QAction *action) --> void</code> <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

cont

Table 3 – continued from previous page

Index	Component	Image
39	FontDialog Class	<p style="text-align: center;">QFontDialog Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> accept(void) --> void acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p> 
40	FileDialog Class	<p style="text-align: center;">QFileDialog Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> accept(void) --> void acceptDrops(void) --> bool acceptMode(void) --> QFileDialog::AcceptMode accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p> 

cont

Table 3 – continued from previous page

Index	Component	Image
41	InputDialog Class	<p style="text-align: center;">QInputDialog Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> accept(void) --> void acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString activateWindow(void) --> void addAction(QAction *action) --> void <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>
42	MessageBox Class	<p style="text-align: center;">QMessageBox Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> about(QWidget *parent, QString,QString) --> void aboutQt(QWidget *parent, QString) --> void accept(void) --> void acceptDrops(void) --> bool accessibleDescription(void) --> QString accessibleName(void) --> QString <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

cont

Table 3 – continued from previous page

Index	Component	Image
43	RegularExpression Class	<p style="text-align: center;">QRegularExpression Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> captureCount(void) --> int delete(void) --> void errorString(void) --> QString globalMatch(QString subject, int offset, init(parent) --> object isValid(void) --> bool <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p> <p style="text-align: right;"> Again Ok Close</p>
44	TcpSocket Class	<p style="text-align: center;">QTcpSocket Class Component</p> <p>Object <input type="text"/></p> <p>Method</p> <ul style="list-style-type: none"> abort(void) --> void atEnd(void) --> bool bind(QHostAddress address, int port, QAbstractSocket::BindFlag mode) --> bool blockSignals(bool block) --> bool bytesAvailable(void) --> int bytesToWrite(void) --> int <p>Parameters <input type="text"/></p> <p>Output <input type="text"/></p>

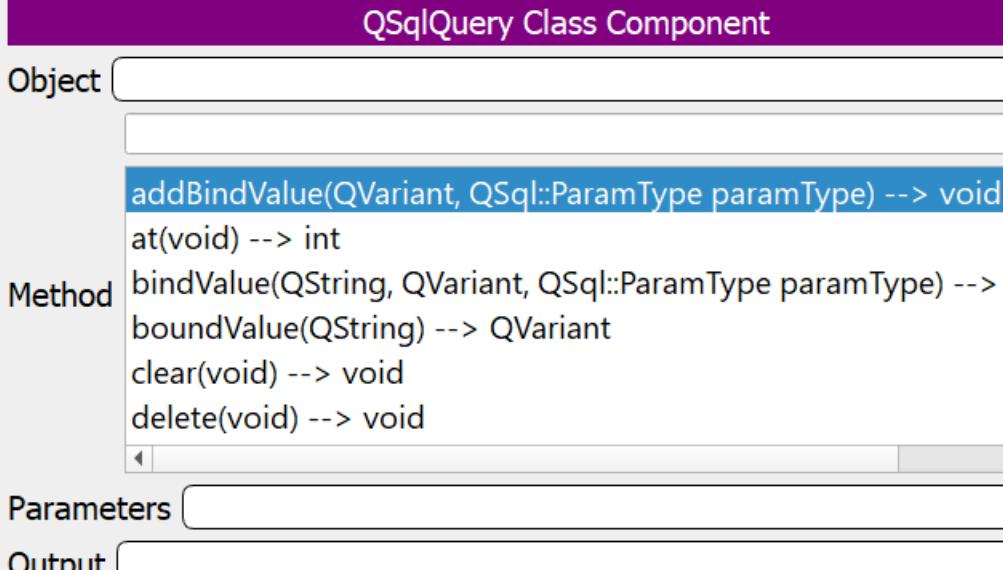
cont

Table 3 – continued from previous page

Index	Component	Image												
45	TCPServer Class	<p style="text-align: center;">QTcpServer Class Component</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">Object</td> <td style="height: 40px;"></td> </tr> <tr> <td></td> <td style="height: 40px;"></td> </tr> <tr> <td style="vertical-align: top;">Method</td> <td> <pre>close(void) --> void delete(void) --> void errorString(void) --> QString getacceptErrorEvent(void) --> char getnewConnectionEvent(void) --> char hasPendingConnections(void) --> bool</pre> </td> </tr> <tr> <td></td> <td style="height: 40px;"></td> </tr> <tr> <td>Parameters</td> <td style="height: 40px;"></td> </tr> <tr> <td>Output</td> <td style="height: 40px;"></td> </tr> </table>	Object				Method	<pre>close(void) --> void delete(void) --> void errorString(void) --> QString getacceptErrorEvent(void) --> char getnewConnectionEvent(void) --> char hasPendingConnections(void) --> bool</pre>			Parameters		Output	
Object														
Method	<pre>close(void) --> void delete(void) --> void errorString(void) --> QString getacceptErrorEvent(void) --> char getnewConnectionEvent(void) --> char hasPendingConnections(void) --> bool</pre>													
Parameters														
Output														
46	SQLDatabase Class	<p style="text-align: center;">QSqlDatabase Class Component</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 15%;">Object</td> <td style="height: 40px;"></td> </tr> <tr> <td></td> <td style="height: 40px;"></td> </tr> <tr> <td style="vertical-align: top;">Method</td> <td> <pre>addDatabase(QString) --> QSqlDatabase cloneDatabase(QSqlDatabase, QString) --> QSqlDatabase close(void) --> void commit(void) --> bool connectOptions(void) --> QString connectionName(void) --> QString</pre> </td> </tr> <tr> <td></td> <td style="height: 40px;"></td> </tr> <tr> <td>Parameters</td> <td style="height: 40px;"></td> </tr> <tr> <td>Output</td> <td style="height: 40px;"></td> </tr> </table>	Object				Method	<pre>addDatabase(QString) --> QSqlDatabase cloneDatabase(QSqlDatabase, QString) --> QSqlDatabase close(void) --> void commit(void) --> bool connectOptions(void) --> QString connectionName(void) --> QString</pre>			Parameters		Output	
Object														
Method	<pre>addDatabase(QString) --> QSqlDatabase cloneDatabase(QSqlDatabase, QString) --> QSqlDatabase close(void) --> void commit(void) --> bool connectOptions(void) --> QString connectionName(void) --> QString</pre>													
Parameters														
Output														

cont

Table 3 – continued from previous page

Index	Component	Image
47	SQLQuery Class	 <p>SQLQuery Class Component</p> <p>Object</p> <p>Method</p> <p>Parameters</p> <p>Output</p> <pre>addBindValue(QVariant, QSql::ParamType paramType) --> void at(void) --> int bindValue(QString, QVariant, QSql::ParamType paramType) --> void boundValue(QString) --> QVariant clear(void) --> void delete(void) --> void</pre>

22.30 GUI/Dialogs

Index	Component	Image
1	Show Message	<p style="text-align: center;">Show Message Component</p> <p>Title : <input type="text" value=""/></p> <p>Message : <input type="text" value=""/></p>
2	Confirm Message	<p style="text-align: center;">Confirm Message Component</p> <p>Title : <input type="text" value=""/></p> <p>Message : <input type="text" value=""/></p> <p>Output : <input type="text" value=""/></p>
3	Input Box	<p style="text-align: center;">Input Box Component</p> <p>Title : <input type="text" value=""/></p> <p>Type : Integer Number Password Text</p> <p>Message : <input type="text" value=""/></p> <p>Output : <input type="text" value=""/></p>
4	Open File Dialog	<p style="text-align: center;">Open File Dialog Component</p> <p>Parent Window: <input type="text" value=""/></p> <p>Title: <input type="text" value=""/></p> <p>Startup Folder: <input type="text" value=""/></p> <p>Files extension: <input type="text" value=""/></p> <p>Output: <input type="text" value=""/></p>
5	Save File Dialog	<p style="text-align: center;">Save File Dialog Component</p> <p>Parent Window: <input type="text" value=""/></p> <p>Title: <input type="text" value=""/></p> <p>Startup Folder: <input type="text" value=""/></p> <p>Files extension: <input type="text" value=""/></p> <p>Output: <input type="text" value=""/></p>

22.31 GUI/More

In-	Component	Image
		<p style="text-align: center;">ObjectsLib Component</p> <p>Parameters :Controller</p> <p>Function :</p> <ul style="list-style-type: none">openWindow(cController)openWindowAndLink(cController,oParentController)openWindowNoShow(cController)openWindowInPackages(cController,aPackagesList)
1	Objects Library	

CHAPTER
TWENTYTHREE

RESOURCES

In this section you will find resources about the PWCT language and related projects
PWCT is developed using the Ring programming language.

23.1 Ring Language Website

For news about the language check the website

<http://ring-lang.net>

INDEX

A

Adding Comments
 Quick Start, 78
Arithmetic Operators
 Operators, 153
Assignment Operators
 Operators, 154

B

Basic Program
 Basic Program component, 74
 Introduction, 75
 Selecting the Component, 75
 Steps Tree, 75
Bitwise Operators
 Operators, 154

C

Comment/Uncomment Steps
 Getting Started, 35
Copy & Paste the Steps
 Getting Started, 27
Creating the Program
 Deep Copy, 128
 Dynamic Loop, 209
 Exit from two loops, 255
 Implicit Conversion, 142
 Loop and Condition, 160
 Main Menu, 169
 Modify Lists, 225
 Operators Precedence, 157
 Say Hello Program, 81
 Short Circuit Evaluation, 277
 The Loop Command, 266
 Using Functions, 308
 Variables, 93
 Variables Scope, 344
Customization
 Getting Started, 49
Customization Window
 Getting Started, 51
Cut & Paste Steps

Getting Started, 33

D

Deep Copy
 Creating the Program, 128
 Deep Copy, 126
 Introduction, 127
 Program Steps, 127
Deleting Steps
 Getting Started, 37
Distribute Menu
 Getting Started, 60
Dynamic Loop
 Creating the Program, 209
 Dynamic Loop, 207
 Introduction, 208
 Program Steps, 208

E

Exit from two loops
 Creating the Program, 255
 Exit from two loops, 253
 Introduction, 254
 Program Steps, 254

F

Features
 Introduction, 4

G

Getting Started
 Comment/Uncomment Steps, 35
 Copy & Paste the Steps, 27
 Customization, 49
 Customization Window, 51
 Cut & Paste Steps, 33
 Deleting Steps, 37
 Distribute Menu, 60
 Go to line, 48
 Hello World Program, 24
 Help Menu, 61
 Inserting Steps, 34

Introduction, 15
 Modify the Steps, 29
 More Options, 56
 Moving Steps Up & Down, 32
 Opening the file, 44
 Printing the file, 46
 Program Menu, 53
 Run Programs, 53
 Save As, 46
 Saving the file, 43
 Search and Replace, 38
 Starting new file, 46
 The Main Window, 16
 Tools Menu, 57
 Undo, 42
 Using the Again Button, 40
 Using the Goal Designer, 26
 Using the Time Machine, 30
 View Menu, 49
 Visual Source Files, 43
 Go to line
 Getting Started, 48

H

Hello World Program
 Getting Started, 24
 Help Menu
 Getting Started, 61
 History
 Introduction, 4

I

Implicit Conversion
 Creating the Program, 142
 Implicit Conversion, 140
 Introduction, 141
 Program Steps, 141
 Inserting Steps
 Getting Started, 34
 Introduction
 Basic Program, 75
 Deep Copy, 127
 Dynamic Loop, 208
 Exit from two loops, 254
 Features, 4
 History, 4
 Implicit Conversion, 141
 Introduction, 1
 Loop and Condition, 160
 Main Menu, 168
 Modify Lists, 224
 Operators Precedence, 157
 Quick Start, 77
 Quotes about PWCT, 3

Say Hello Program, 80
 Short Circuit Evaluation, 276
 The Loop Command, 265
 Using Functions, 306
 Variables, 92
 Variables Scope, 343
 Welcome, 2

L

Logical Operators
 Operators, 153
 Loop and Condition
 Creating the Program, 160
 Introduction, 160
 Loop and Condition, 159
 Program Steps, 160

M

Main Menu
 Creating the Program, 169
 Introduction, 168
 Main Menu, 167
 Program Steps, 168

Misc Operators
 Operators, 155

Modify Lists
 Creating the Program, 225
 Introduction, 224
 Modify Lists, 223
 Program Steps, 224

Modify the Steps
 Getting Started, 29

More Options
 Getting Started, 56
 Moving Steps Up & Down
 Getting Started, 32

O

Opening the file
 Getting Started, 44
 Operators
 Arithmetic Operators, 153
 Assignment Operators, 154
 Bitwise Operators, 154
 Introduction, 152
 Logical Operators, 153
 Misc Operators, 155
 Operators Precedence, 155
 Relational Operators, 153
 Operators Precedence
 Creating the Program, 157
 Introduction, 157
 Operators, 155
 Operators Precedence, 156

Program Steps, 157

P

Printing some text
 Rich Comments, 71
Printing the file
 Getting Started, 46
Program Menu
 Getting Started, 53
Program Steps
 Deep Copy, 127
 Dynamic Loop, 208
 Exit from two loops, 254
 Implicit Conversion, 141
 Loop and Condition, 160
 Main Menu, 168
 Modify Lists, 224
 Operators Precedence, 157
 Say Hello Program, 80
 Short Circuit Evaluation, 276
 The Loop Command, 265
 Using Functions, 306
 Variables, 93
 Variables Scope, 343

Q

Quick Start
 Adding Comments, 78
 Introduction, 77
 Quick Start component, 76
 Selecting the Component, 77
 Steps Tree, 78
 The Interaction Page, 77
Quotes about PWCT
 Introduction, 3

R

Relational Operators
 Operators, 153
Resources
 Introduction, 443
 Ring Language Website, 444
Rich Comments
 Introduction, 62
 Printing some text, 71
 Using the Comment Component, 67
 Using the Header Component, 63
 Using the Image Component, 69
 Using the New Line Component, 65
Ring Language Website
 Resources, 444
Run Programs
 Getting Started, 53

S

Save As
 Getting Started, 46
Saving the file
 Getting Started, 43
Say Hello Program
 Creating the Program, 81
 Introduction, 80
 Program Steps, 80
 Say Hello program, 79
Search and Replace
 Getting Started, 38
Selecting the Component
 Basic Program, 75
 Quick Start, 77
Short Circuit Evaluation
 Creating the Program, 277
 Introduction, 276
 Program Steps, 276
 Short Circuit Evaluation, 275
Starting new file
 Getting Started, 46
Steps Tree
 Basic Program, 75
 Quick Start, 78

T

The Concept
 Introduction, 5
The Interaction Page
 Quick Start, 77
The Loop Command
 Creating the Program, 266
 Introduction, 265
 Program Steps, 265
 The Loop Command, 264
The Main Window
 Getting Started, 16
Tools Menu
 Getting Started, 57

U

Undo
 Getting Started, 42
Using Functions
 Creating the Program, 308
 Introduction, 306
 Program Steps, 306
 Using Functions, 305
Using the Again Button
 Getting Started, 40
Using the Comment Component
 Rich Comments, 67

Using the Goal Designer
 Getting Started, [26](#)
Using the Header Component
 Rich Comments, [63](#)
Using the Image Component
 Rich Comments, [69](#)
Using the New Line Component
 Rich Comments, [65](#)
Using the Time Machine
 Getting Started, [30](#)

V

Variables
 Creating the Program, [93](#)
 Introduction, [92](#)
 Program Steps, [93](#)
 Using Variables, [91](#)
Variables Scope
 Creating the Program, [344](#)
 Introduction, [343](#)
 Program Steps, [343](#)
 Variables Scope, [342](#)
View Menu
 Getting Started, [49](#)
Visual Source Files
 Getting Started, [43](#)

W

Welcome
 Introduction, [2](#)