

Introduction to the SPL Interpreter

Stephen Monk
November 2011



Background and System Requirements



Background

- This is an introduction to the SPL interpreter – the application that executes programs written in SPL.
- It is not an introduction to SPL itself.
- However, even if you don't know SPL, you will probably understand most of this presentation.

System Requirements

- The SPL interpreter requires a Java virtual machine (version 6 or later) and any modern browser (IE, Firefox, Chrome, Safari, etc.)
- The essential features of the SPL interpreter are available in all computing environments that meet those conditions. These features are:
 - Creating and running programs
 - Tracing programs with breakpoints
 - Formatting code
 - Creating snapshots of a program's state
 - Drag-and-Drop text between the interpreter and native applications.

Continued ...

System Requirements

- Two additional features of the interpreter – File Operations and Copy/Paste – require permission to access your local machine. Permission is granted via the Java Network Launch Protocol (JNLP).
- It is very likely that JNLP is already installed on your machine – it is part of all recent Java releases.
- However, if JNLP is not available on your computer, and if you are not allowed to install it (perhaps because you are in a restricted environment, such as a computer lab), then:
 - You will not see the buttons related to these features. (See next slide.)
 - You will not be able to use these features (but note that Drag-and-Drop is still available.)
 - In this tutorial, simply skip the sections dealing with these features.

System Requirements

The image shows a screenshot of a Java IDE interface. At the top, there is a toolbar with various icons. Two groups of icons are highlighted with red boxes and labeled: "File Operations" (containing icons for New, Open, Save, and Print) and "Copy/Paste" (containing icons for Copy and Paste). Below the toolbar, the main workspace is divided into several panels: a code editor on the left with a yellow highlight on line 1, an "Output" panel at the bottom left, a "Memory" panel on the right, and a "User Input" panel at the bottom right. Two red arrows point from the "File Operations" and "Copy/Paste" boxes to a central text box. A second text box is located below the first one. The status bar at the bottom left shows "File: Untitled" and the bottom right shows "1 | 1".

File Operations

Copy/Paste

1

Memory

User Input

Output

File: Untitled

1 | 1

These features require access to your machine via JNLP.

If you don't see these buttons, **don't worry** – you can still use the essential features of the interpreter.

Proceed with tutorial ...

Overview of Application Components



Sections

Tool Bar

The image shows a programming IDE interface with several sections labeled in red text:

- Code Editor:** A large central area with a yellow highlight on line 1. A red box contains the text "Code Editor" and "Your program goes here."
- Memory:** A panel on the right side. A red box contains the text: "Shows values of all variables stored in memory during program execution."
- User Input:** A panel on the right side, below Memory. A red box contains the text: "Shows values entered by the user during program execution – i.e., as a result of 'input' commands."
- Output:** A panel at the bottom left. A red box contains the text: "Shows all output produced by your program during execution."
- Tool Bar:** A horizontal bar at the top containing various icons for file operations, execution, and editing. A red line with arrows points to it from the "Tool Bar" label above.
- Status Bar:** A horizontal bar at the bottom. It contains the text "File: Untitled" on the left, "Status Bar" in the center, and "1 | 1" on the right.

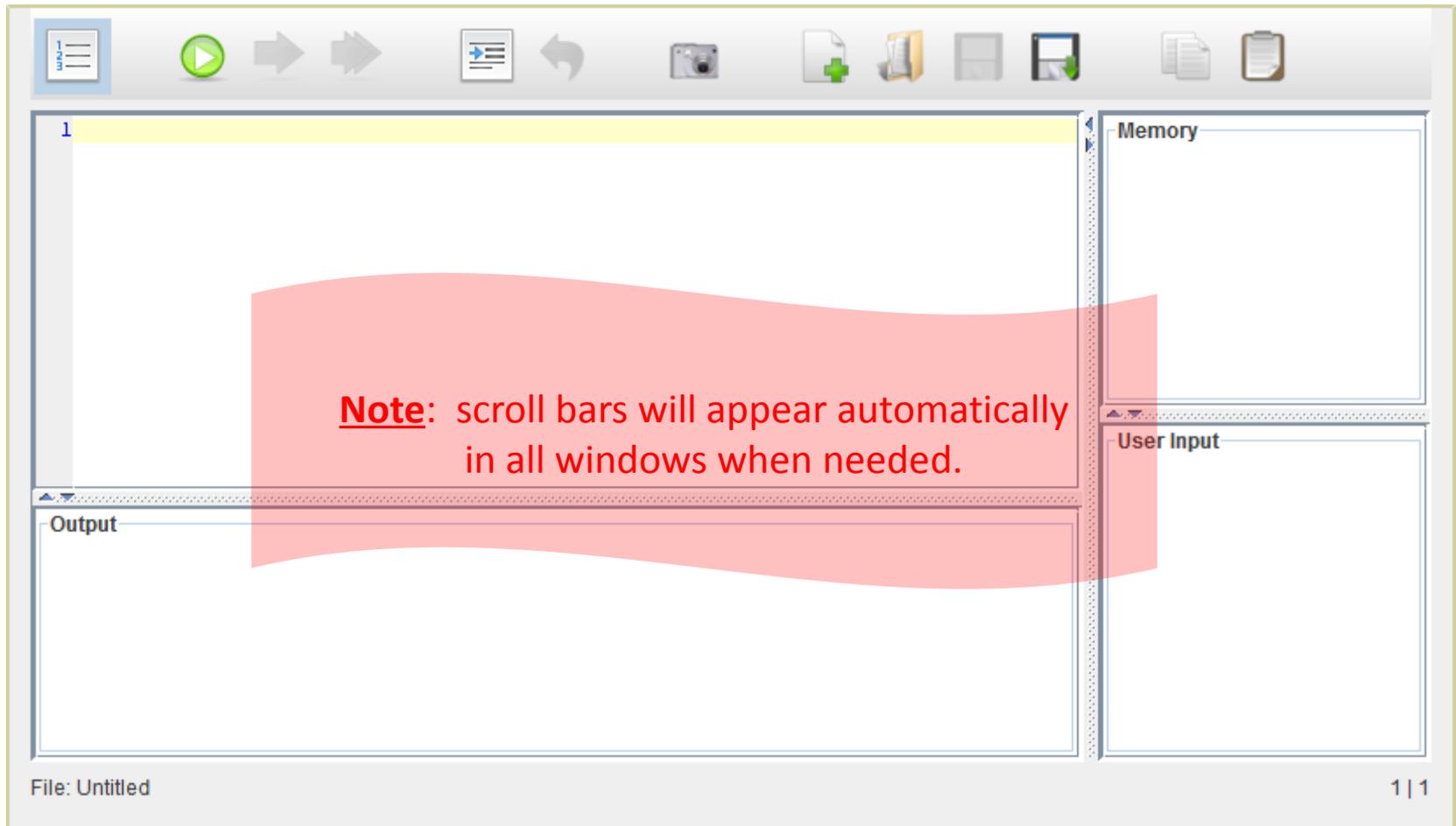
Windows

Adjust the sizes of the windows by dragging the separator bars.

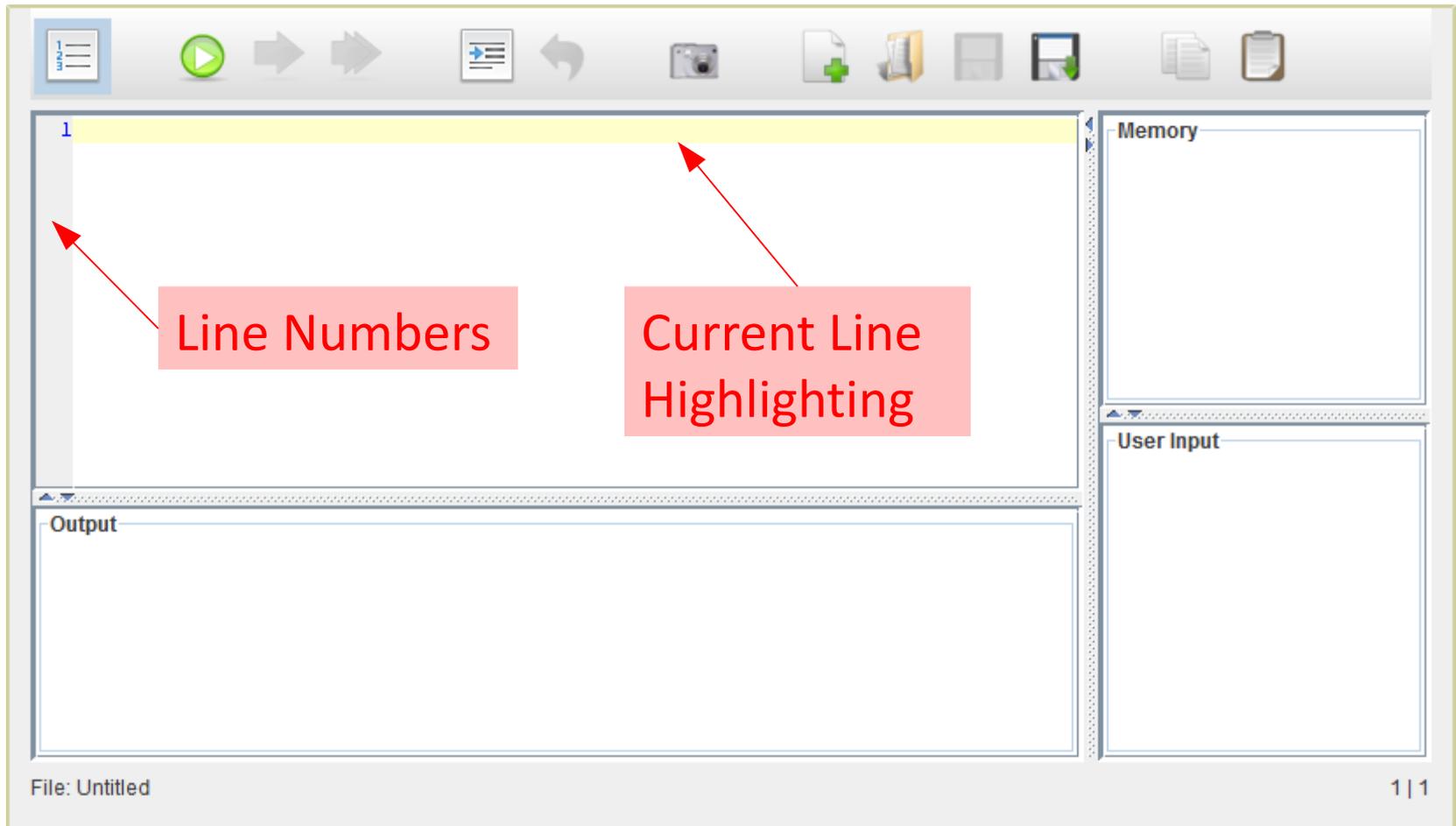
Or use the blue arrows to maximize or minimize.

File: Untitled 1 | 1

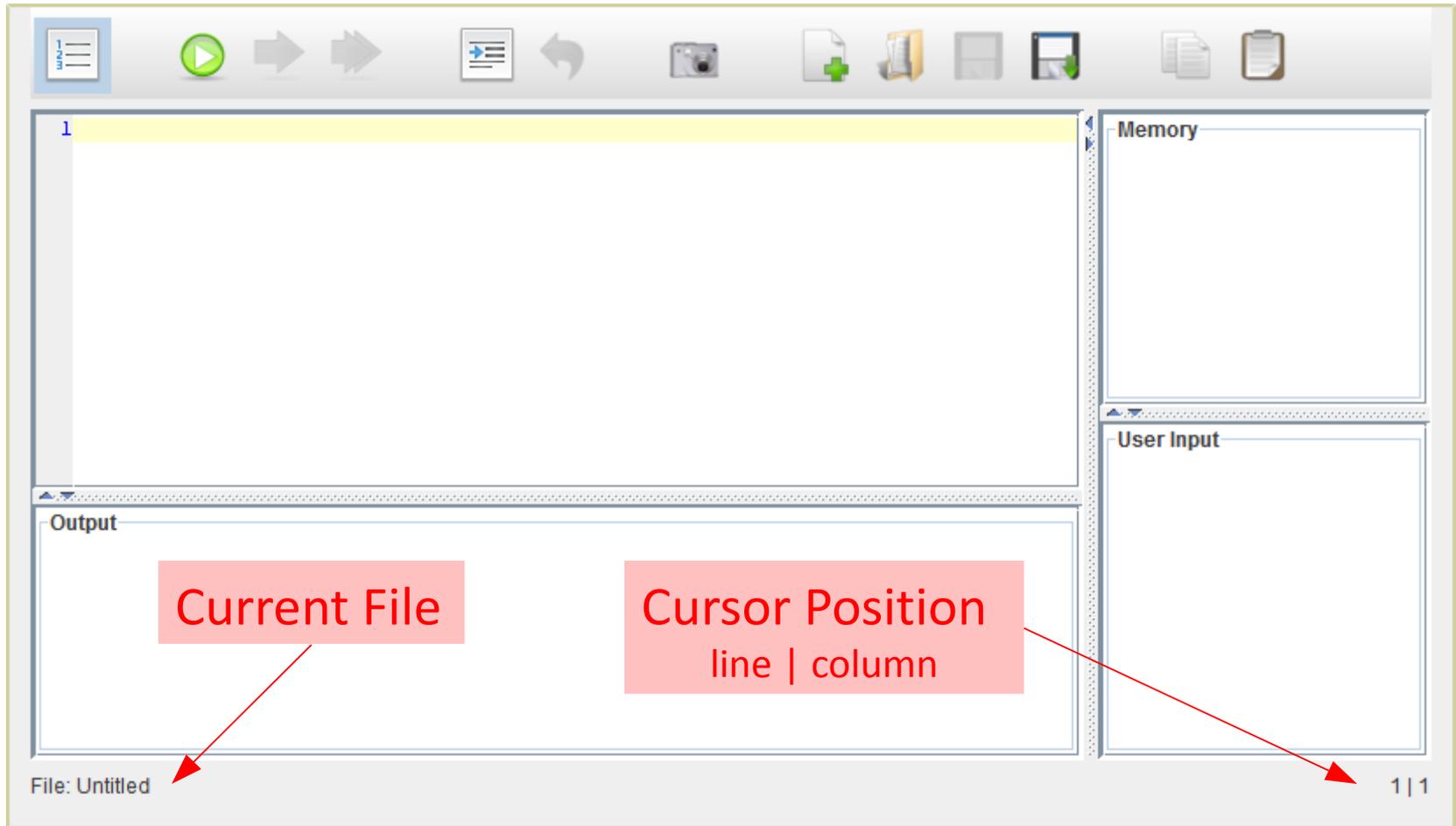
Windows



Windows



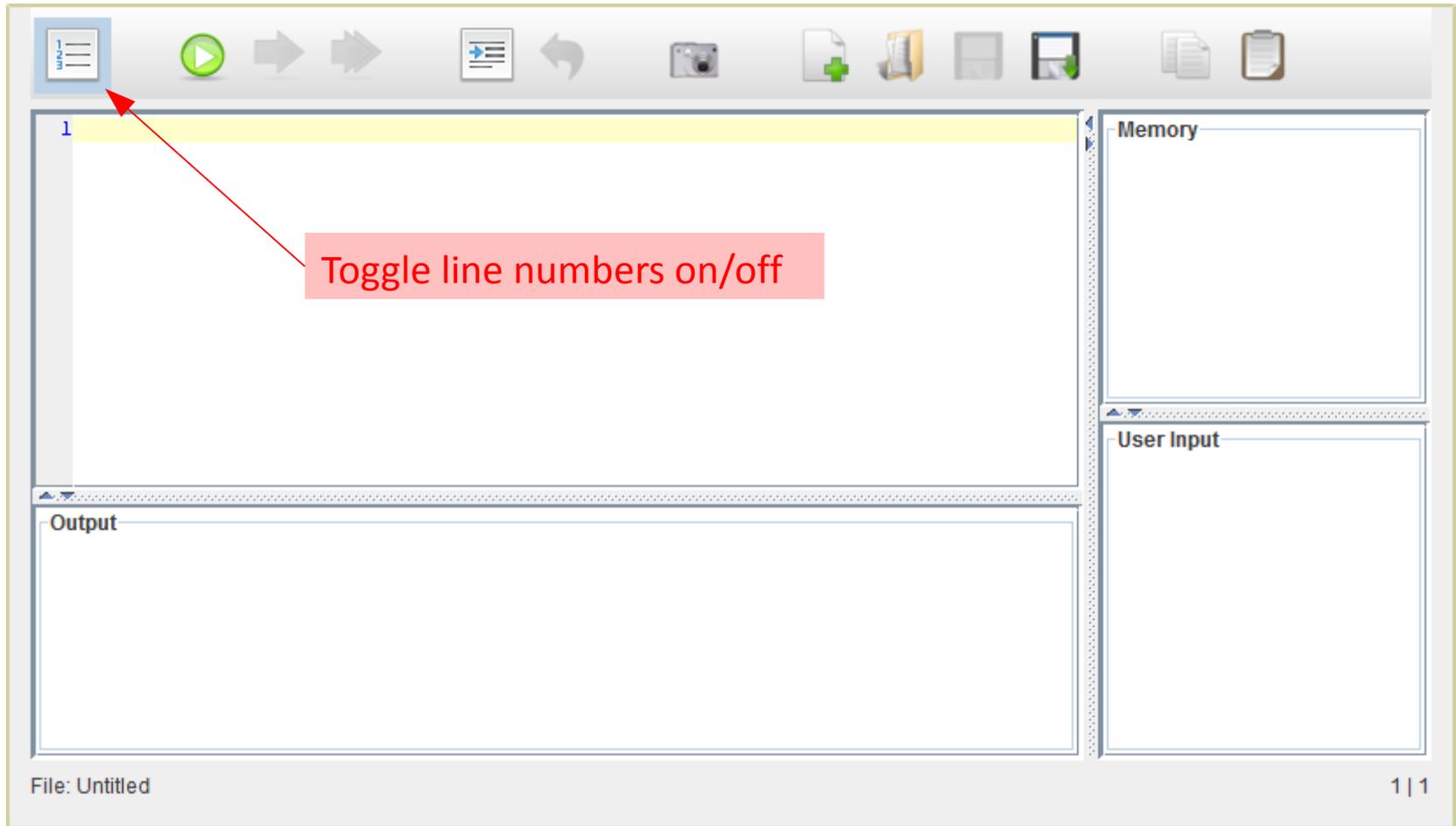
Status Bar



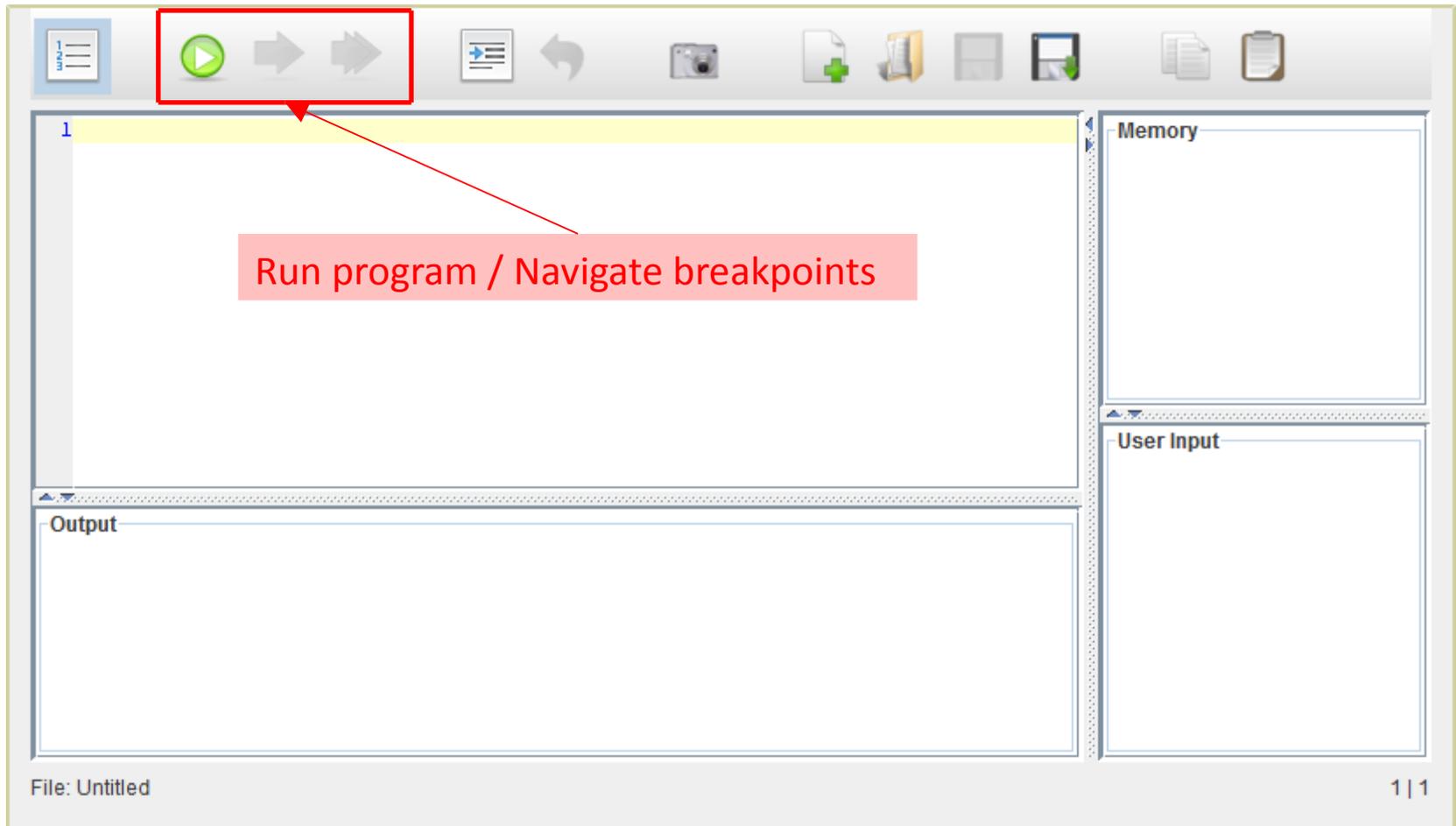
The Tool Bar – Details



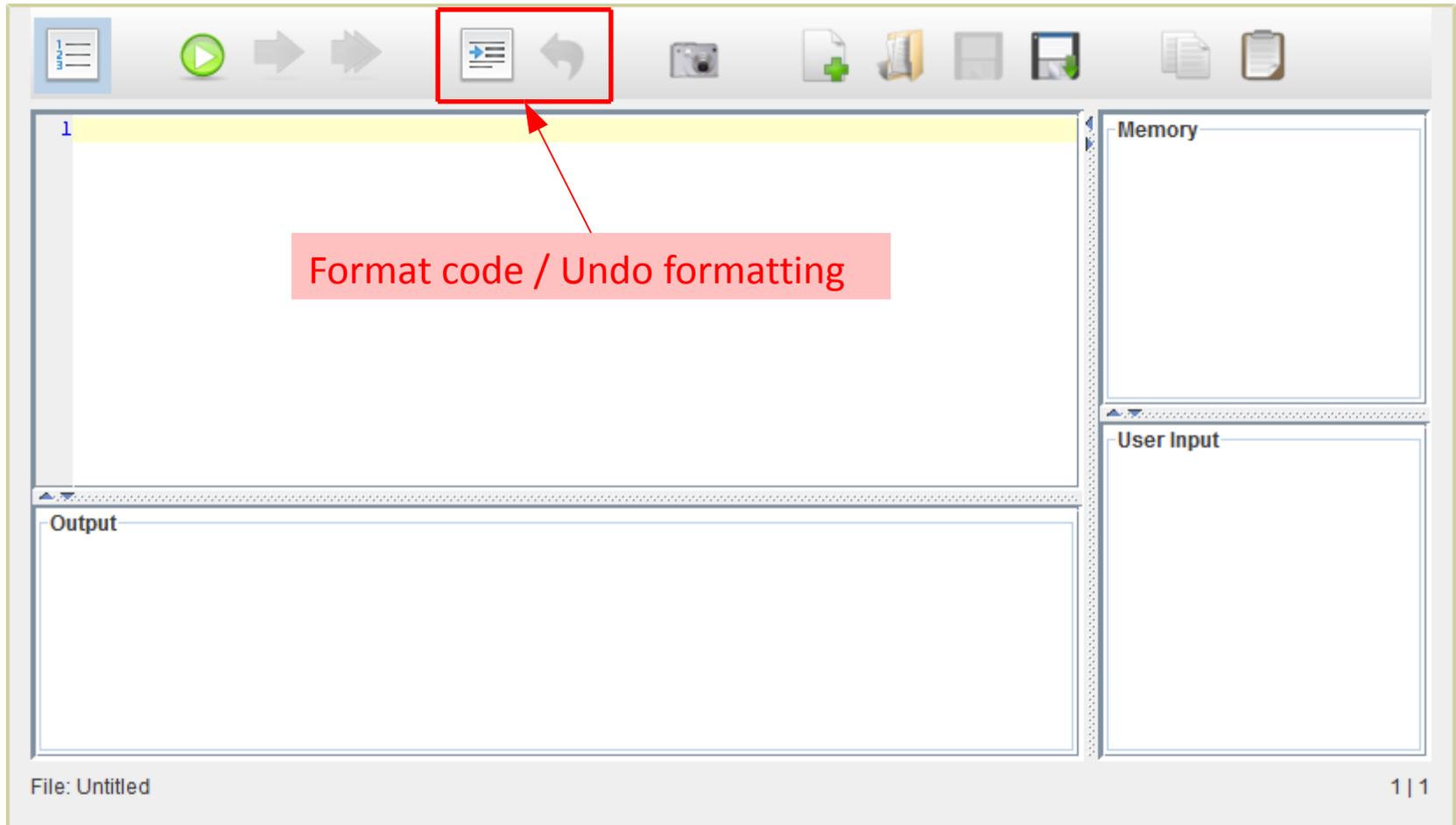
Tool Bar



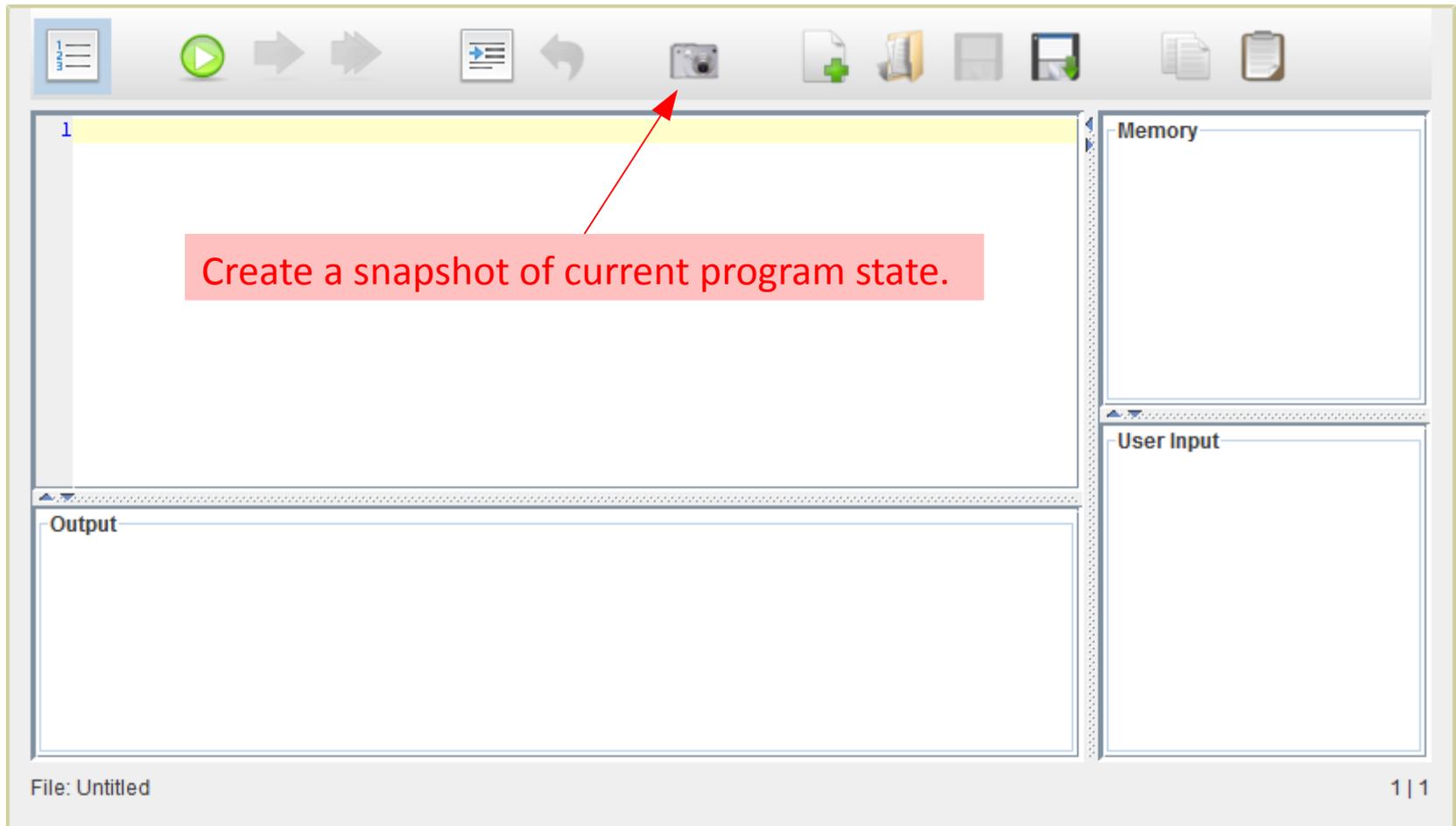
Tool Bar



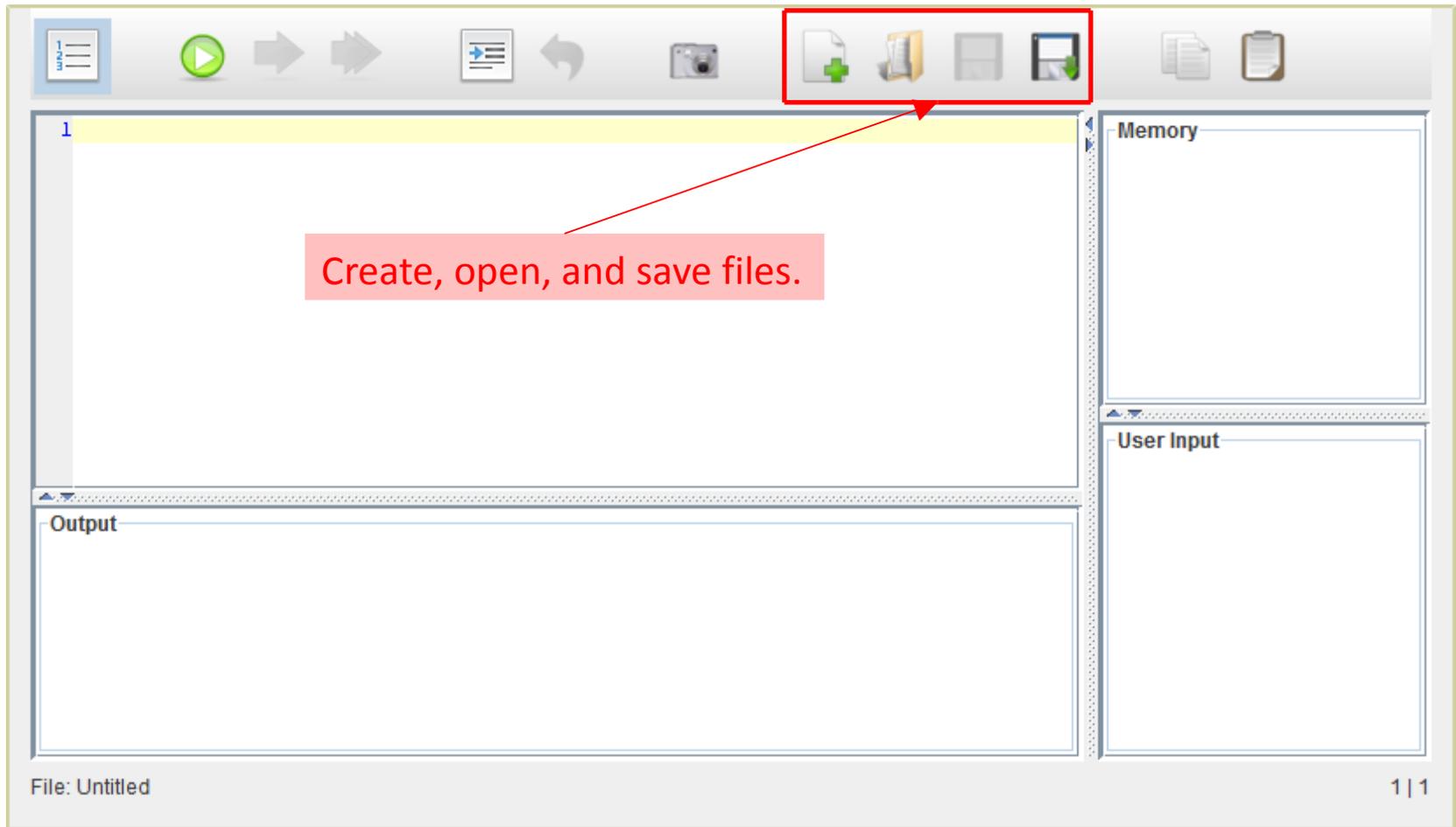
Tool Bar



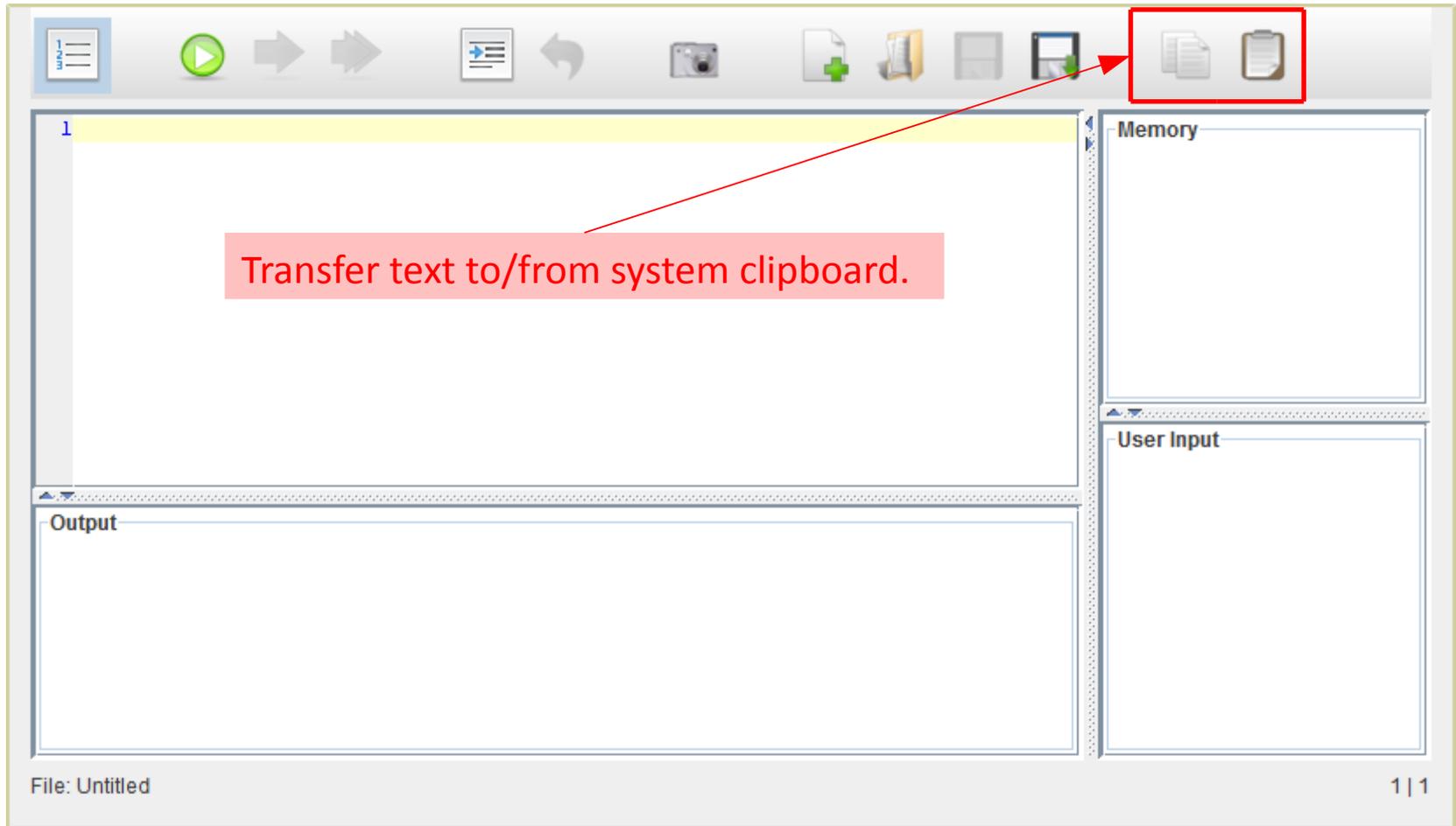
Tool Bar



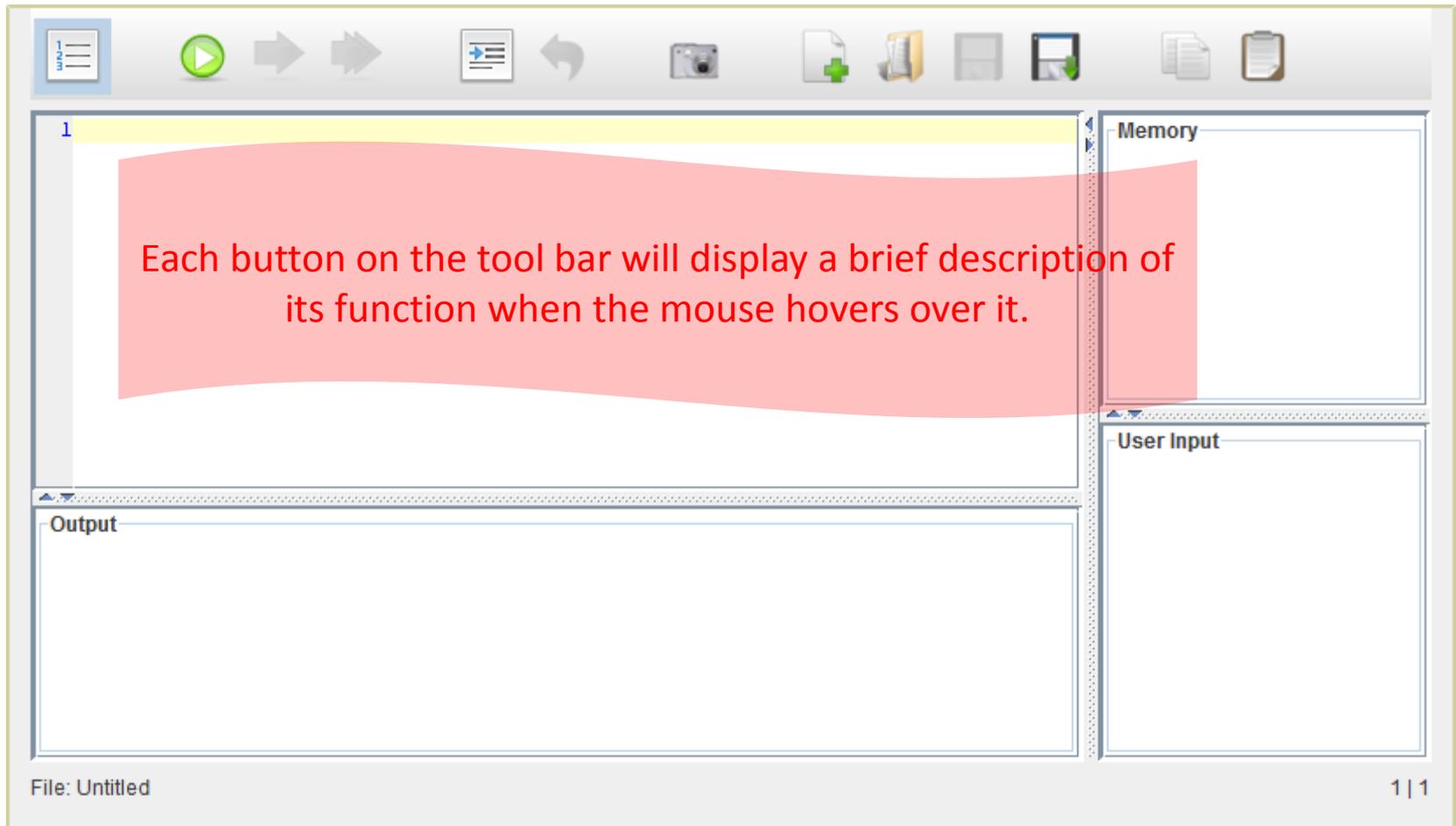
Tool Bar



Tool Bar



Tool Bar



Running a Program



Running a Program

The image shows a programming IDE interface. At the top is a toolbar with icons for running (a green play button), stepping through code (right arrows), undo (a curved arrow), and other development tools. Below the toolbar is a code editor window with a yellow background for the first line. Two red-bordered text boxes are overlaid on the editor. To the right of the editor are three monitoring windows: 'Memory', 'User Input', and 'Output', all of which are currently empty. The status bar at the bottom left shows 'File: Untitled' and the bottom right shows '1 | 1'.

1

After you have entered a program here, in the editor window, you are ready to run the program.

The following slides demonstrate how to do this with an example program.

Memory

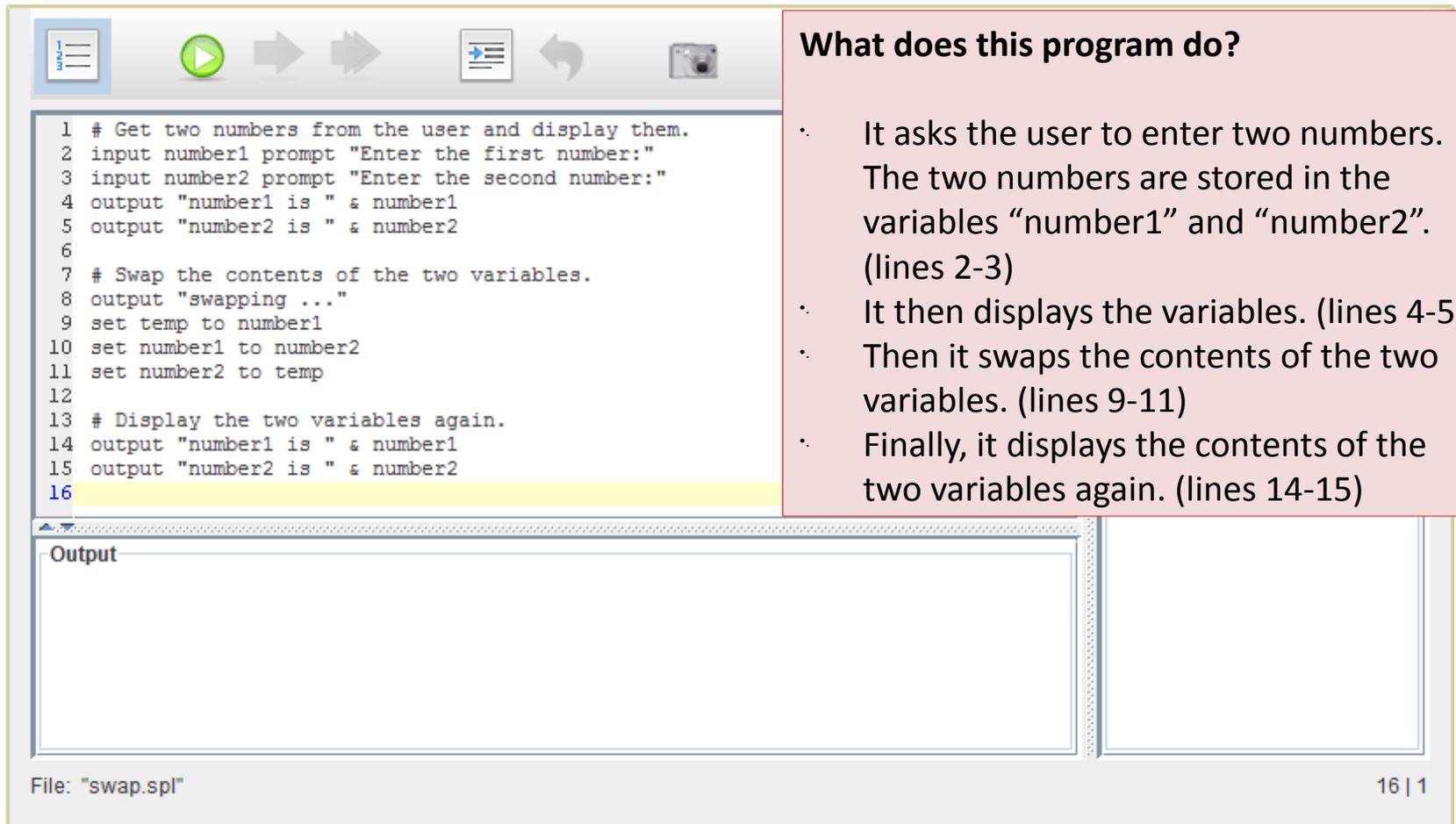
User Input

Output

File: Untitled

1 | 1

Running a Program



The screenshot shows a program editor window with a toolbar at the top containing icons for a list, play, step forward, search, step back, and a camera. The code in the editor is as follows:

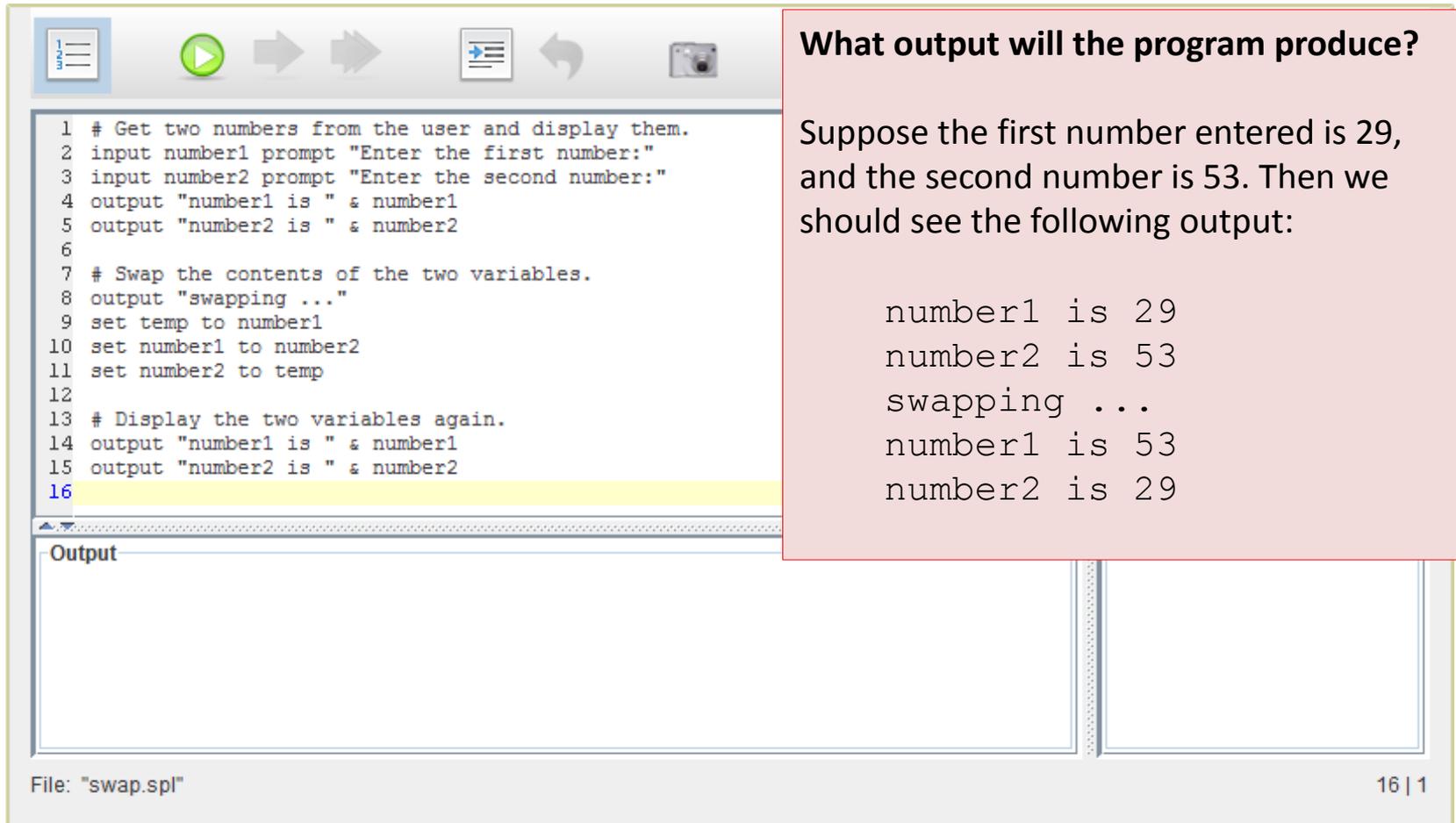
```
1 # Get two numbers from the user and display them.
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two variables.
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again.
14 output "number1 is " & number1
15 output "number2 is " & number2
16
```

Below the code is an empty output window with the title "Output". At the bottom left of the editor, it says "File: 'swap.spl'". At the bottom right, it says "16 | 1".

What does this program do?

- It asks the user to enter two numbers. The two numbers are stored in the variables "number1" and "number2". (lines 2-3)
- It then displays the variables. (lines 4-5)
- Then it swaps the contents of the two variables. (lines 9-11)
- Finally, it displays the contents of the two variables again. (lines 14-15)

Running a Program



The image shows a screenshot of a program editor window. The editor contains the following code:

```
1 # Get two numbers from the user and display them.  
2 input number1 prompt "Enter the first number:"  
3 input number2 prompt "Enter the second number:"  
4 output "number1 is " & number1  
5 output "number2 is " & number2  
6  
7 # Swap the contents of the two variables.  
8 output "swapping ..."  
9 set temp to number1  
10 set number1 to number2  
11 set number2 to temp  
12  
13 # Display the two variables again.  
14 output "number1 is " & number1  
15 output "number2 is " & number2  
16
```

The code is intended to take two numbers as input, swap them, and display the results. The output window is currently empty.

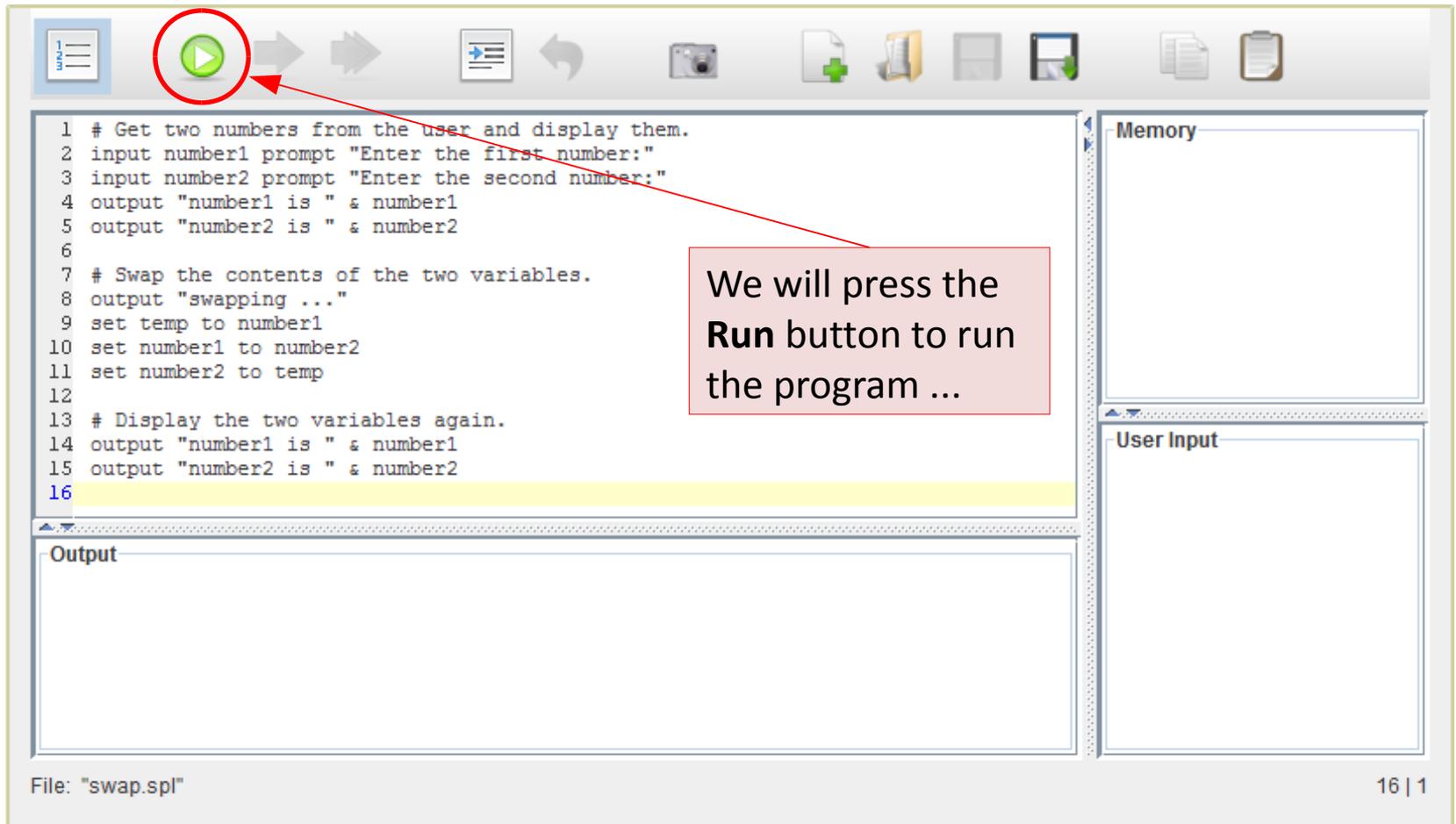
What output will the program produce?

Suppose the first number entered is 29, and the second number is 53. Then we should see the following output:

```
number1 is 29  
number2 is 53  
swapping ...  
number1 is 53  
number2 is 29
```

File: "swap.spl" 16 | 1

Running a Program



The screenshot shows a programming IDE interface. At the top is a toolbar with various icons. The Run button, represented by a green play icon, is circled in red. A red arrow points from this button to a callout box. The main area is a code editor with the following code:

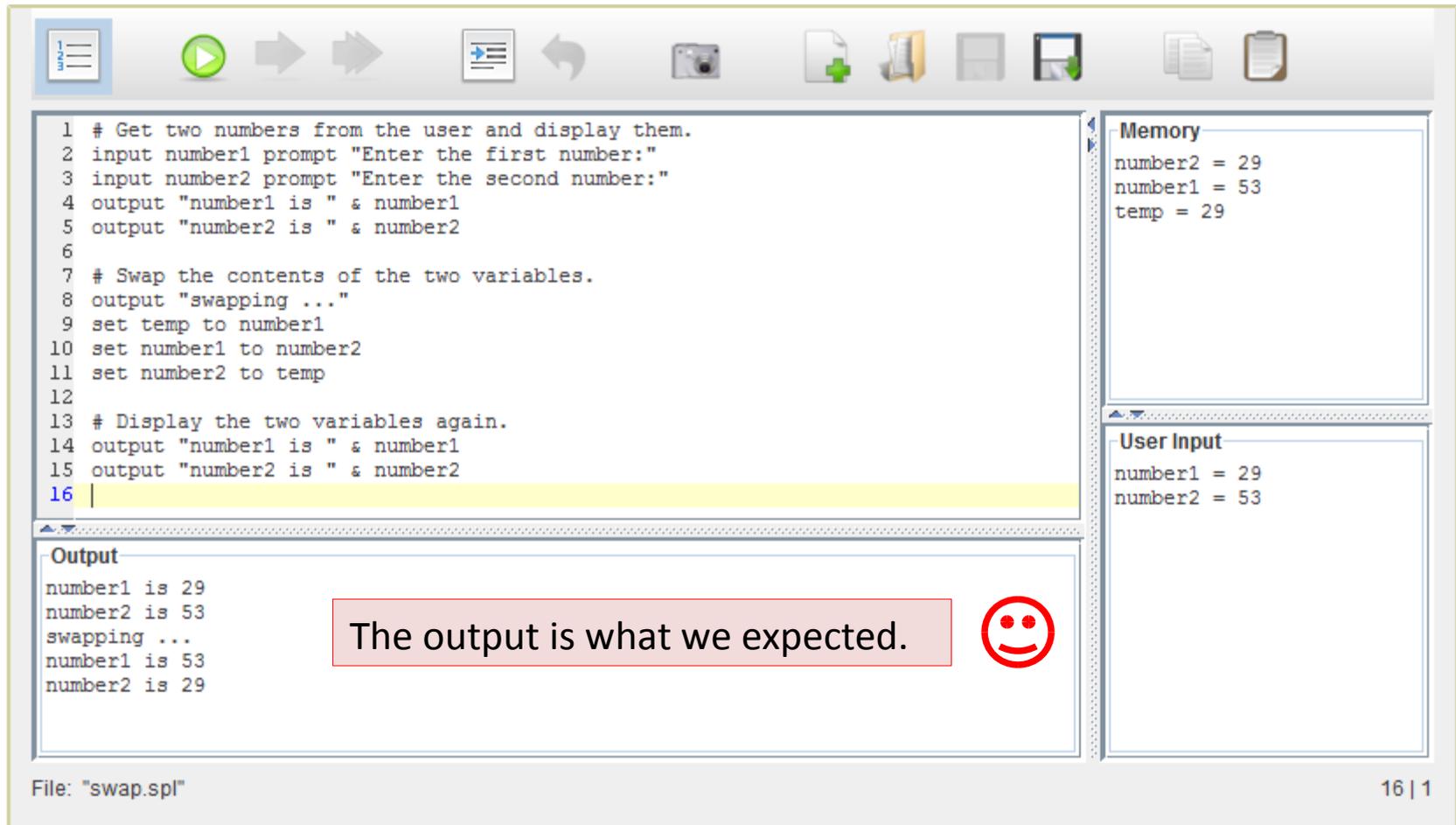
```
1 # Get two numbers from the user and display them.  
2 input number1 prompt "Enter the first number:"  
3 input number2 prompt "Enter the second number:"  
4 output "number1 is " & number1  
5 output "number2 is " & number2  
6  
7 # Swap the contents of the two variables.  
8 output "swapping ..."  
9 set temp to number1  
10 set number1 to number2  
11 set number2 to temp  
12  
13 # Display the two variables again.  
14 output "number1 is " & number1  
15 output "number2 is " & number2  
16
```

The line number 16 is highlighted in yellow. To the right of the code editor are three panels: Memory, User Input, and Output, all of which are currently empty. At the bottom left, the file name is "swap.spl". At the bottom right, the page number is "16 | 1".

We will press the **Run** button to run the program ...

... and ...

Running a Program



The screenshot shows a program execution environment with a toolbar at the top containing icons for running, stepping through, and other controls. The main area is divided into several panels:

- Code Panel:** Contains 16 lines of code for a swap program. Line 16 is highlighted in yellow.
- Memory Panel:** Shows the current state of memory: `number2 = 29`, `number1 = 53`, and `temp = 29`.
- User Input Panel:** Shows the input provided by the user: `number1 = 29` and `number2 = 53`.
- Output Panel:** Shows the program's output: `number1 is 29`, `number2 is 53`, `swapping ...`, `number1 is 53`, and `number2 is 29`.

A red-bordered box with the text "The output is what we expected." and a red smiley face icon is overlaid on the output panel.

File: "swap.spl" 16 | 1

but there's more ...

Running a Program

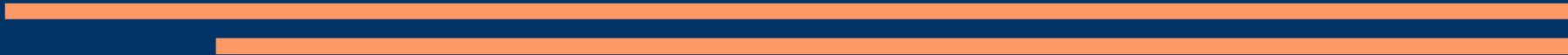
- The Memory window shows the **current** values of all variables.
- So, after the program has finished, the final values of the variables are displayed.
- Notice that it also shows the variable “temp” – this variable was not set by the user, but came from the line 9 of the program.

```
7 # Swap the contents of the two variables.  
8 output "swapping ..."  
9 set temp to number1  
10 set number1 to number2  
11 set number2 to temp  
12
```

- The User Input window shows the **original** values of any variables that came from the user (i.e., from an “input” command).
- For example, notice that the value of “number1” shown here is 29, which is the value the user originally entered.

The screenshot shows a window with a title bar containing icons for home, refresh, and clipboard. The main content area is divided into two panes. The top pane, titled "Memory", displays the current state of variables: number2 = 29, number1 = 53, and temp = 29. The bottom pane, titled "User Input", displays the original values entered by the user: number1 = 29 and number2 = 53. A red arrow points from the text box above to the "temp" variable in the Memory pane. Another red arrow points from the text box below to the "number1" variable in the User Input pane. A third red arrow points from the left side of the code editor to line 9.

Debugging/Tracing a Program



Debugging/Tracing a Program

- Most programs execute very quickly. There are many things happening in the computer's memory during that short time.
- It is sometimes useful to watch what is happening at a slower pace.
 - Lets us understand why the program behaves the way it does.
 - Helps us figure out problems with the program.

Debugging/Tracing a Program

- The process of examining the state of a program at intermediate points in its life-cycle is called “debugging” or “tracing”.
- This is accomplished with the use of **breakpoints**.
 - A breakpoint is an instruction that tells the program to pause at a particular line.

Debugging/Tracing a Program

- You can add a breakpoint to your program by clicking on the line number to the left of the code. The line of code will be highlighted in pink.
- You can remove a breakpoint by clicking the line number again. The highlighting will disappear.
- You can add as many breakpoints as you want, but only on lines that contain executable code.
 - You can't add a breakpoint to a blank line, a line containing a comment, or a line starting with "else", "endif", or "endwhile".

Debugging/Tracing a Program

- When you run the program, it will pause when it reaches a breakpoint and wait for you to tell it to continue. This allows you to see what is happening at that point.
- To illustrate the debugging features, let's return to the "swap" program that we used earlier ...

Debugging/Tracing a Program

Run

Next

Finish

What will happen when we run the program?

- All of the code before line 4 will execute, and then the program will pause at line 4.
- The highlighting on line 4 will change from pink to green.
- The tool bar will change to indicate that the program is now in “debug” mode:
 - the **Run** button will be disabled;
 - the **Next** button and the **Finish** button will be enabled.

Three breakpoints have been set – at lines 4, 10, and 11.

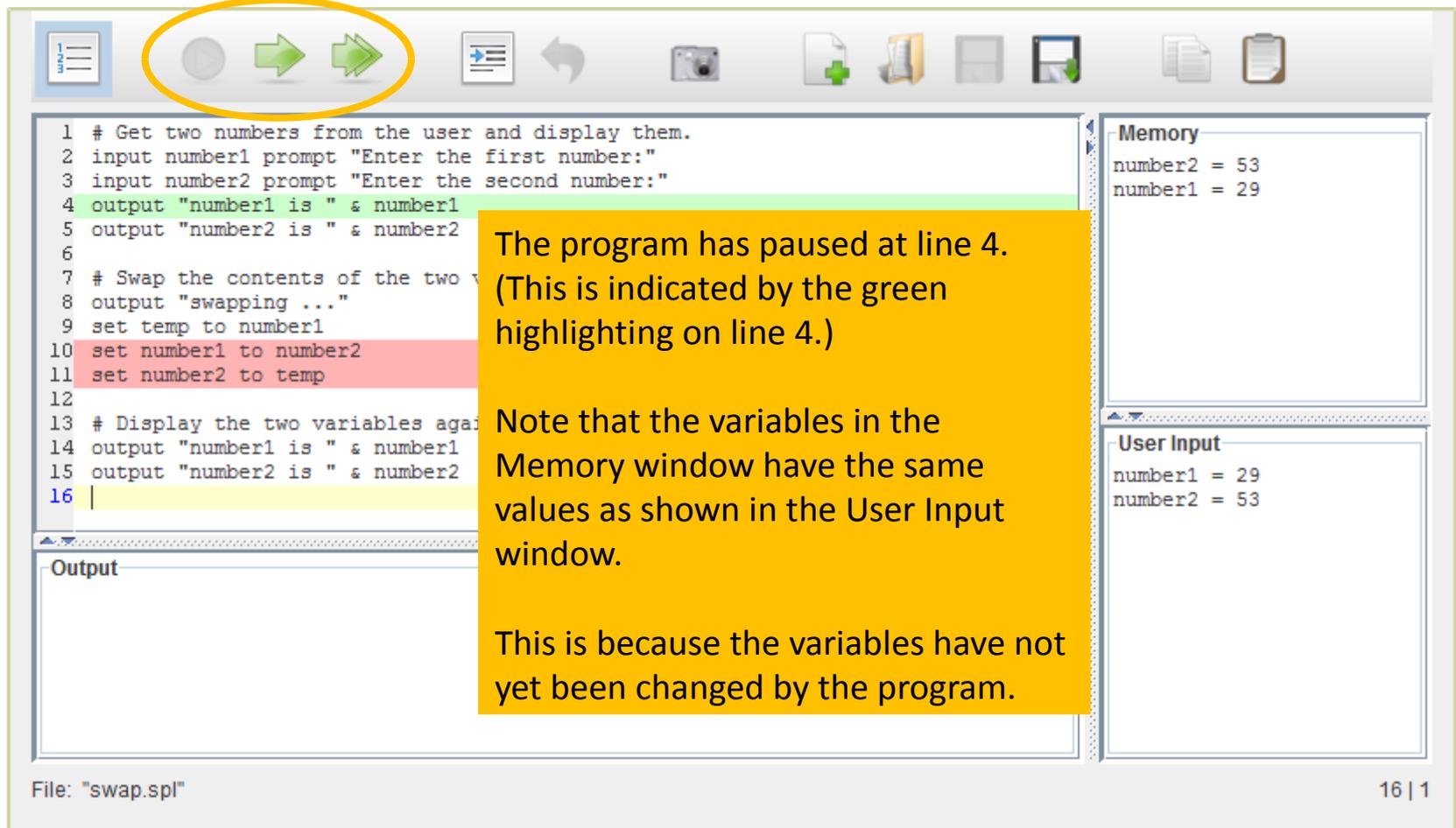
File: "swap.spl"

16 | 1

Continued ...

Debugging/Tracing a Program

Also note that the **Run** button is now disabled, and the **Next** and **Finish** buttons are enabled.



The program has paused at line 4. (This is indicated by the green highlighting on line 4.)

Note that the variables in the Memory window have the same values as shown in the User Input window.

This is because the variables have not yet been changed by the program.

```
1 # Get two numbers from the user and display them.
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two variables
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again
14 output "number1 is " & number1
15 output "number2 is " & number2
16 |
```

Memory

```
number2 = 53
number1 = 29
```

User Input

```
number1 = 29
number2 = 53
```

Output

File: "swap.spl" 16 | 1

Continued ...

Debugging/Tracing a Program

To proceed, we have a choice:

- Pressing the **Next** button will advance to the next breakpoint (line 10 in this case); or
- Pressing the **Finish** button will ignore the remaining breakpoints and complete the program without pausing again.

Let's choose the first option ...

File: "swap.spl" 16 | 1

Continued ...

Debugging/Tracing a Program

1 # Get two numbers from the user and display them.
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two vari
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again.
14 output "number1 is " & number1
15 output "number2 is " & number2
16

Output
number1 is 29
number2 is 53
swapping ...

Memory
number2 = 53
number1 = 29
temp = 29

User Input
number1 = 29
number2 = 53

File: "swap.spl" 16 | 1

Lines 4, 5, 8, and 9 have been executed, and the program has paused at line 10.

In the Output window, we see the results of lines 4, 5, and 8.

In the Memory window, we see the result of line 9: the variable "temp" has been created with the same value as "number1".

Now let's press the **Next** button again to advance to the next breakpoint (line 11).

Continued ...

Debugging/Tracing a Program

1 # Get two numbers from the user and display them.
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two variables.
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again.
14 output "number1 is " & number1
15 output "number2 is " & number2
16

Memory
number2 = 53
number1 = 53
temp = 29

User Input
number1 = 29
number2 = 53

Output
number1 is 29
number2 is 53
swapping ...

File: "swap.spl" 16 | 1

Line 10 has been executed, and the program has paused at line 11.

In the Memory window, we see the result of line 10: the variable "number1" has been changed so that it contains the same value as "number2".

Now let's press the **Next** button again. Since there are no more breakpoints, this will complete the program's execution.

Continued ...

Debugging/Tracing a Program

The screenshot displays a program execution environment with several components:

- Code Editor:** Contains 16 lines of code. Lines 4, 5, 10, and 11 are highlighted in pink. Line 16 is highlighted in yellow. The code is as follows:

```
1 # Get two numbers from the user
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two variables.
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again
14 output "number1 is " & number1
15 output "number2 is " & number2
16 |
```
- Control Buttons:** A yellow oval highlights the Run button (a green play icon), which is enabled. The Next and Finish buttons (grey arrows) are disabled.
- Memory Window:** Shows the current state of memory:

```
Memory
number2 = 29
number1 = 53
temp = 29
```

An arrow points from the value 29 in the Memory window to the corresponding value in the User Input window.
- User Input Window:** Shows the input values:

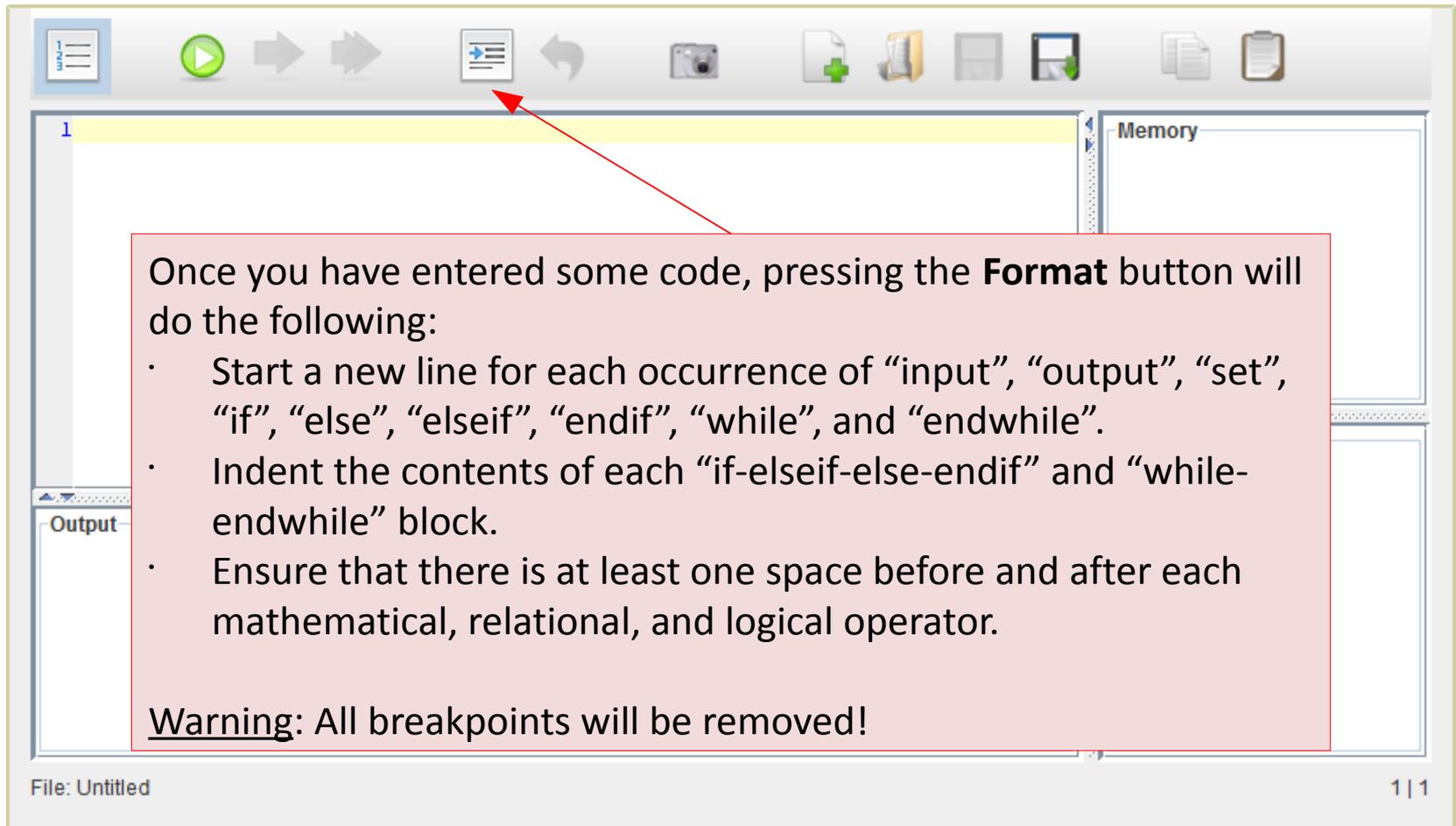
```
User Input
number1 = 29
number2 = 53
```
- Output Window:** Shows the program's output:

```
Output
number1 is 29
number2 is 53
swapping ...
number1 is 53
number2 is 29
```
- Annotations:** Three yellow callout boxes provide context:
 - Top: "Note that the Run button is now enabled again, and the Next and Finish buttons are disabled."
 - Middle: "Lines 11-15 have been executed, and the program is finished."
 - Bottom-left: "In the Output window, the results of lines 14 and 15 have been appended."
 - Bottom-right: "In the Memory window, we see the result of line 11: the variable 'number2' has been changed so that it contains the same value as 'temp'."
- Footer:** The file name "File: 'swap.spl'" is shown at the bottom left, and the page number "16 | 1" is at the bottom right.

Formatting Code



Formatting Code



The screenshot shows a code editor window with a toolbar at the top. The toolbar contains several icons: a play button, a right arrow, a double right arrow, a document with a plus sign (the Format button), a left arrow, a camera, a document with a plus sign, a folder, a document with a plus sign, a document with a plus sign, and a clipboard. A red arrow points from the Format button to a red-bordered text box. The text box contains the following text:

Once you have entered some code, pressing the **Format** button will do the following:

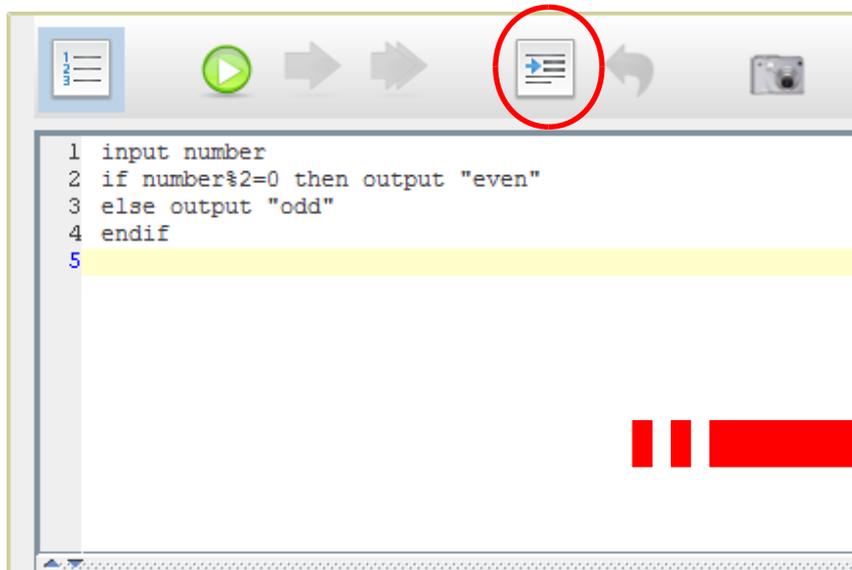
- Start a new line for each occurrence of “input”, “output”, “set”, “if”, “else”, “elseif”, “endif”, “while”, and “endwhile”.
- Indent the contents of each “if-elseif-else-endif” and “while-endwhile” block.
- Ensure that there is at least one space before and after each mathematical, relational, and logical operator.

Warning: All breakpoints will be removed!

File: Untitled 1 | 1

see next slide for example ...

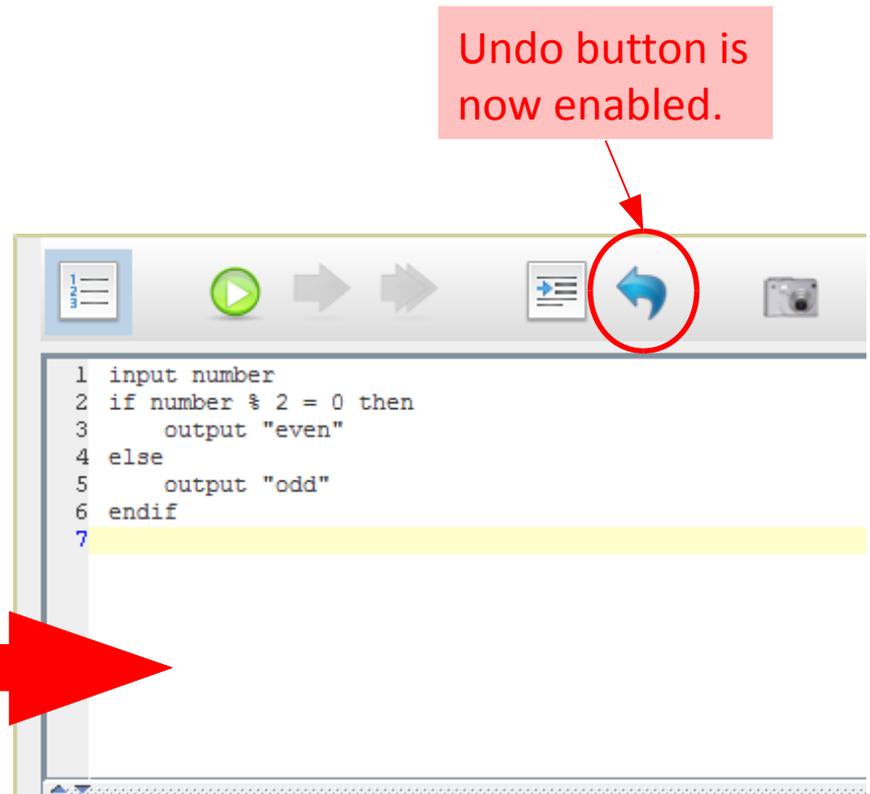
Formatting Code



A screenshot of a code editor interface. The top toolbar contains several icons: a list icon, a play button, two right-pointing arrows, a 'Format' icon (a document with a blue arrow), a left-pointing arrow, and a camera icon. The 'Format' icon is circled in red. Below the toolbar, the code is displayed in a text area with line numbers 1 through 5. Line 5 is highlighted in yellow.

```
1 input number
2 if number%2=0 then output "even"
3 else output "odd"
4 endif
5
```

Undo button is now enabled.



A screenshot of a code editor interface, similar to the one on the left. The top toolbar contains the same icons as the left screenshot, but the 'Undo' icon (a blue curved arrow) is circled in red. A red arrow points from a text box above to this icon. Below the toolbar, the code is displayed in a text area with line numbers 1 through 7. Line 7 is highlighted in yellow.

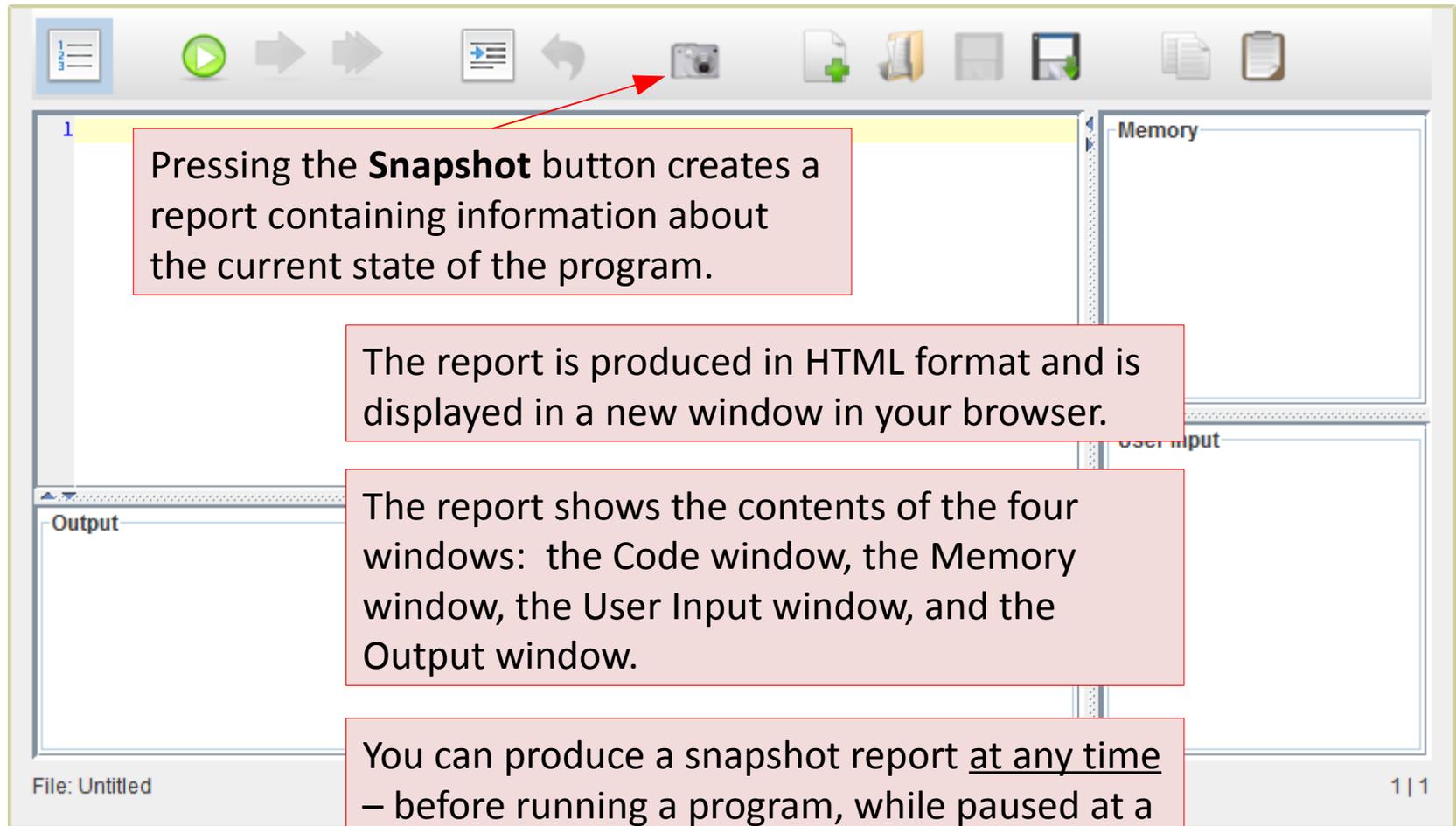
```
1 input number
2 if number % 2 = 0 then
3     output "even"
4 else
5     output "odd"
6 endif
7
```



Snapshots



Snapshots



Example ...

Snapshots

The “swap” program is running and is paused at a breakpoint at line 11.

A snapshot report should accurately display the contents of the four windows. (It won't show the line highlighting, though.)

```
1 # Get two numbers from the user and display them.
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two variables
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again.
14 output "number1 is " & number1
15 output "number2 is " & number2
16
```

Memory

```
number2 = 53
number1 = 53
temp = 29
```

User Input

```
number1 = 29
number2 = 53
```

Output

```
number1 is 29
number2 is 53
swapping ...
```

File: "swap.spl" 16 | 1

Let's press the **Snapshot** button to see the result ...

Continued ...

Snapshots

The report shown below is one document, but it's too big to show in one screen.

The image shows two Firefox browser windows side-by-side, both displaying the same file: `file:///E:/jdev/splWeb/dist/interpreter.html`.

The left window displays the **Source Code** of a program. The code is as follows:

```
1 # Get two numbers from the user and display them.
2 input number1 prompt "Enter the first number:"
3 input number2 prompt "Enter the second number:"
4 output "number1 is " & number1
5 output "number2 is " & number2
6
7 # Swap the contents of the two variables.
8 output "swapping ..."
9 set temp to number1
10 set number1 to number2
11 set number2 to temp
12
13 # Display the two variables again.
14 output "number1 is " & number1
15 output "number2 is " & number2
```

Below the source code, the label **User Input** is visible. A yellow arrow points from this label to the right window.

The right window displays the execution report, which is a standard HTML document. It contains the following sections:

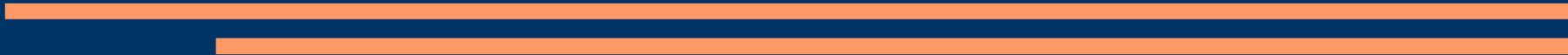
- User Input**
 - number1 = 29
 - number2 = 53
- Memory**
 - number2 = 53
 - number1 = 53
 - temp = 29
- Output**
 - 1 number1 is 29
 - 2 number2 is 53
 - 3 swapping ...

A green box on the right window contains the text: "The report is independent of the program. It is standard HTML, so you can now do whatever you want with it – save it, print it, email it, etc."

In fact, the reason that the Snapshot feature exists is because a screenshot doesn't have enough room to show everything at once.

File Operations

(Skip this section if these features are not available in your environment.)



File Operations

The **New File** button closes the current file (if any) and starts a new, blank file.

File: Untitled 1 | 1

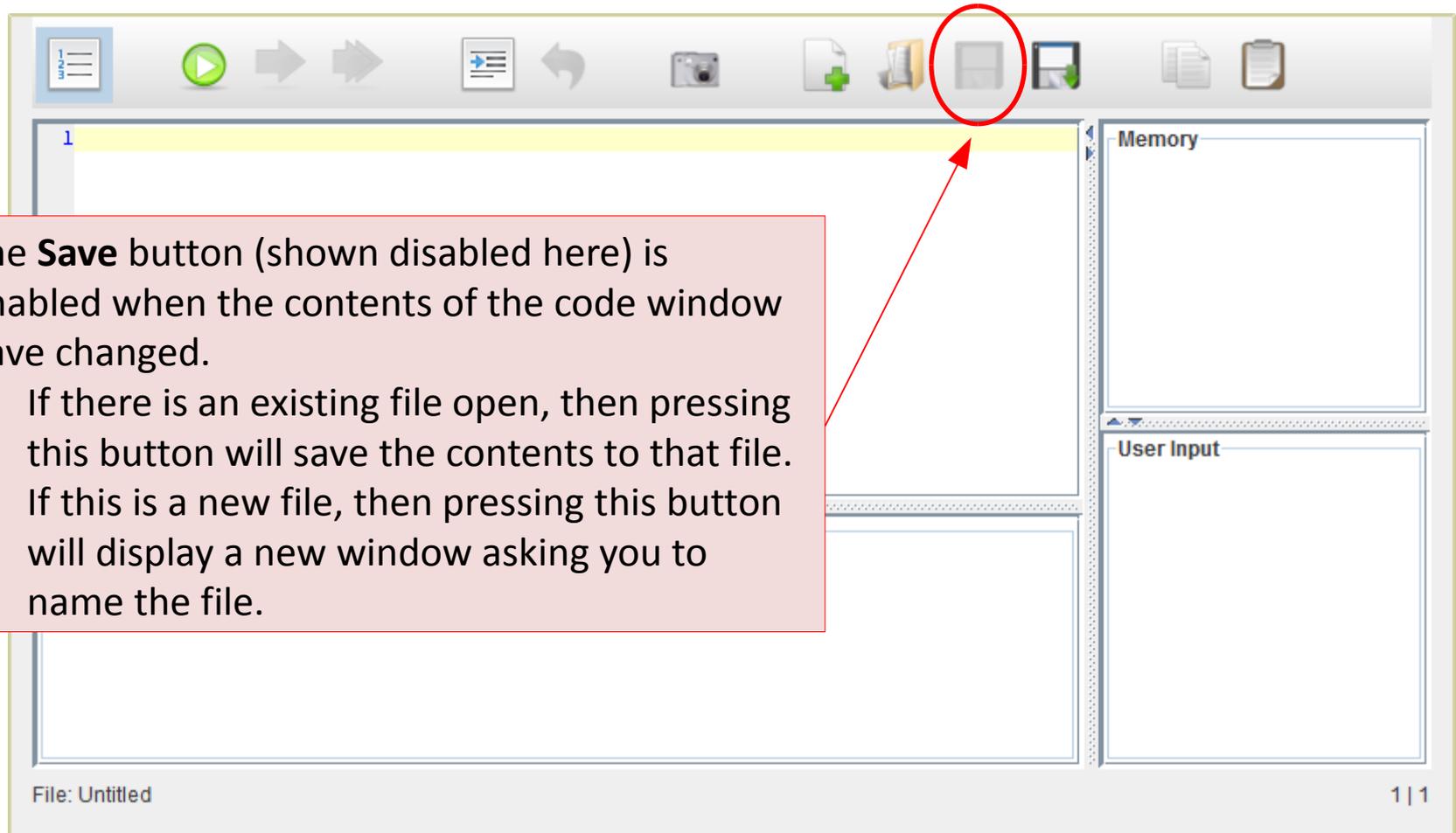
File Operations

The **Open File** button displays a new window that allows you to choose an existing file on your local machine.

File: Untitled

1 | 1

File Operations



See next slide for example ...

File Operations – Save Example

The screenshot shows a code editor window with a toolbar at the top. The toolbar contains icons for running, stepping through, undo, redo, opening a file, saving a file, and other functions. The save icon is disabled. The main editor area contains the following code:

```
1 # Get an input from the user and display it.  
2  
3 input value prompt "Enter something:"  
4
```

Line 4 is highlighted in yellow. A red arrow points from a text box to the save icon in the toolbar. Another red arrow points from a text box to the file name in the status bar.

The **Save** button is disabled because the user has not yet entered any new code.

The user has opened a file called "userInput.spl"

File: "userInput.spl" 4 | 1

Continued ...

File Operations – Save Example

The screenshot shows a programming IDE with a toolbar at the top. The toolbar includes icons for running, stepping through, undo, redo, and saving. The main editor area contains the following code:

```
1 # Get an input from the user and display it.  
2  
3 input value prompt "Enter something:"  
4 output "You entered: " & value  
5
```

Line 4 is highlighted in yellow. A red arrow points from a text box to this line. Another red arrow points from a text box to the save icon in the toolbar.

The IDE also features a "Memory" panel on the right and an "Output" panel at the bottom. The status bar at the bottom left shows "File: 'userInput.spl'" and the bottom right shows "5 | 1".

The user has entered new code on line 4.

The **Save** button is now enabled.
Let's see what happens when the button is pressed ...

Continued ...

File Operations – Save Example

```
1 # Get an input from the user and display it.  
2  
3 input value prompt "Enter something:"  
4 output "You entered: " & value  
5
```

The user has pressed the **Save** button.

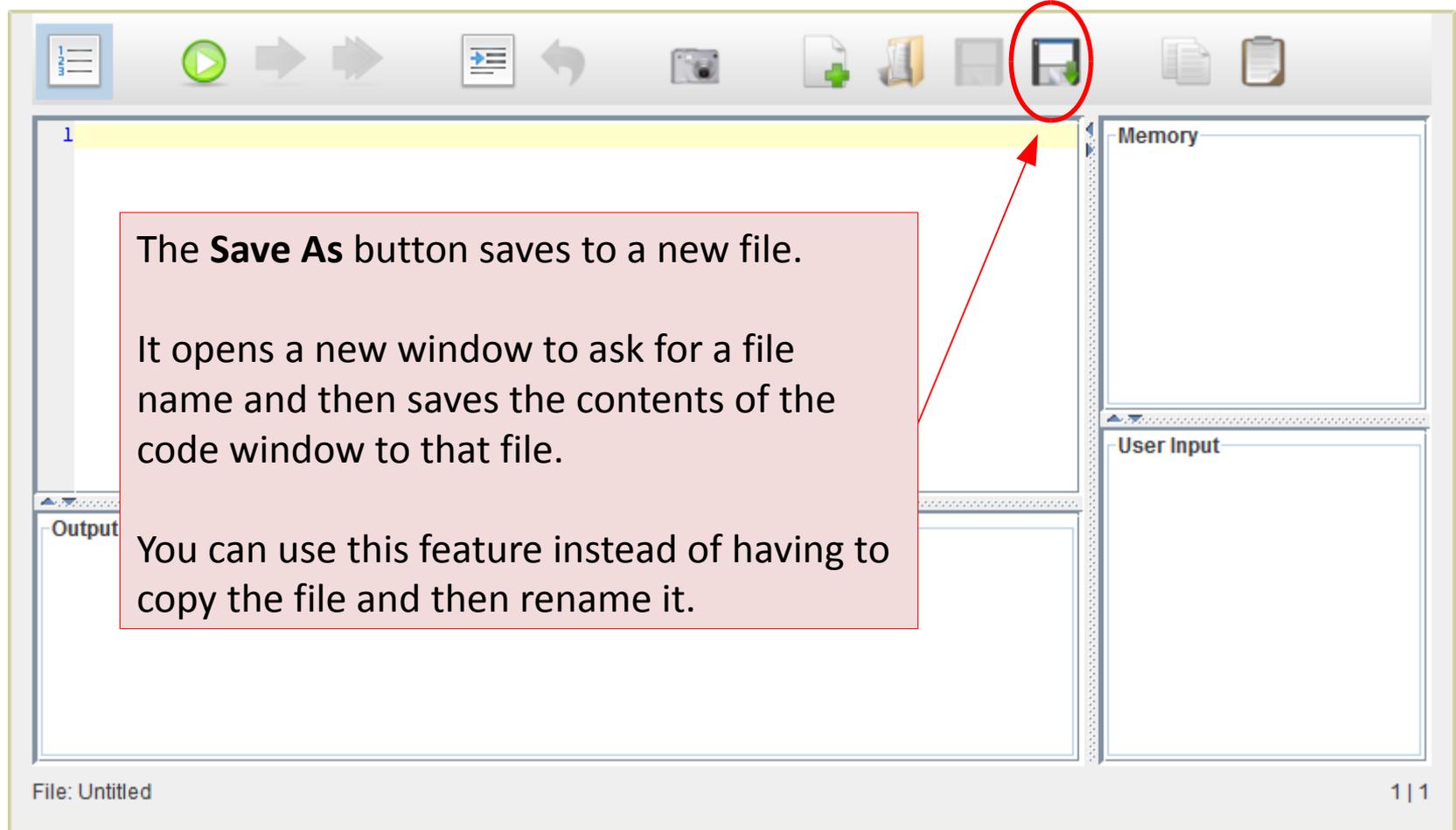
The status bar shows the time and date of the save operation.

The **Save** button is now disabled again.

File: "userInput.spl" saved at 08 November 2011 12:13:13 AST

5 | 1

File Operations



The **Save As** button saves to a new file.

It opens a new window to ask for a file name and then saves the contents of the code window to that file.

You can use this feature instead of having to copy the file and then rename it.

File: Untitled

1 | 1

Transferring Text

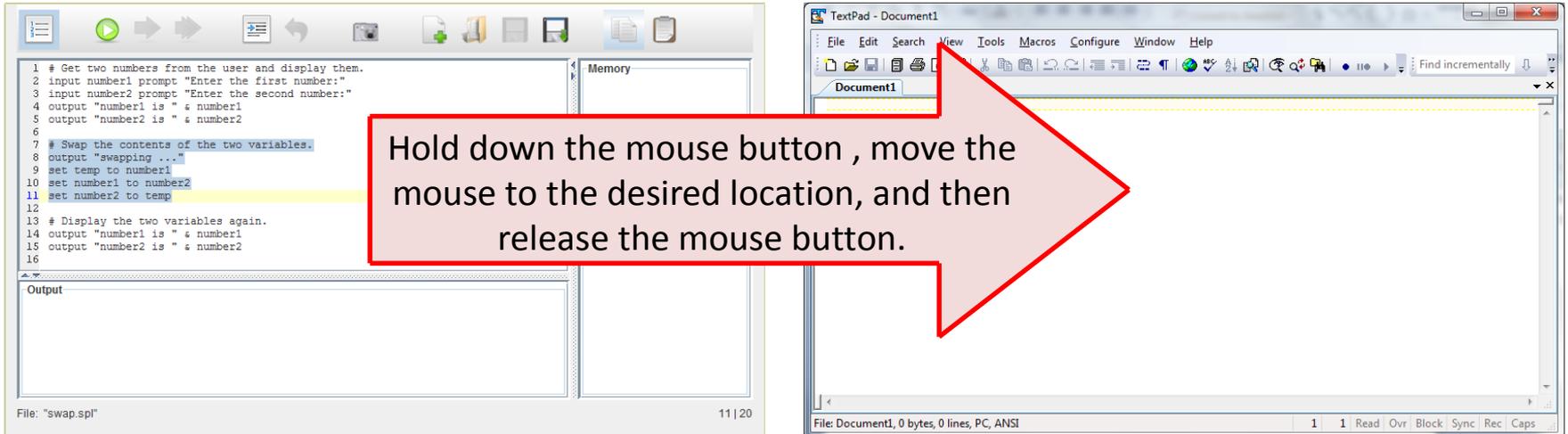


Drag-and-Drop

- The quickest and easiest way to transfer text between the interpreter and a native application is to “drag-and-drop” the text.
- These actions do not access your local machine and therefore do not require special permission.

Examples ...

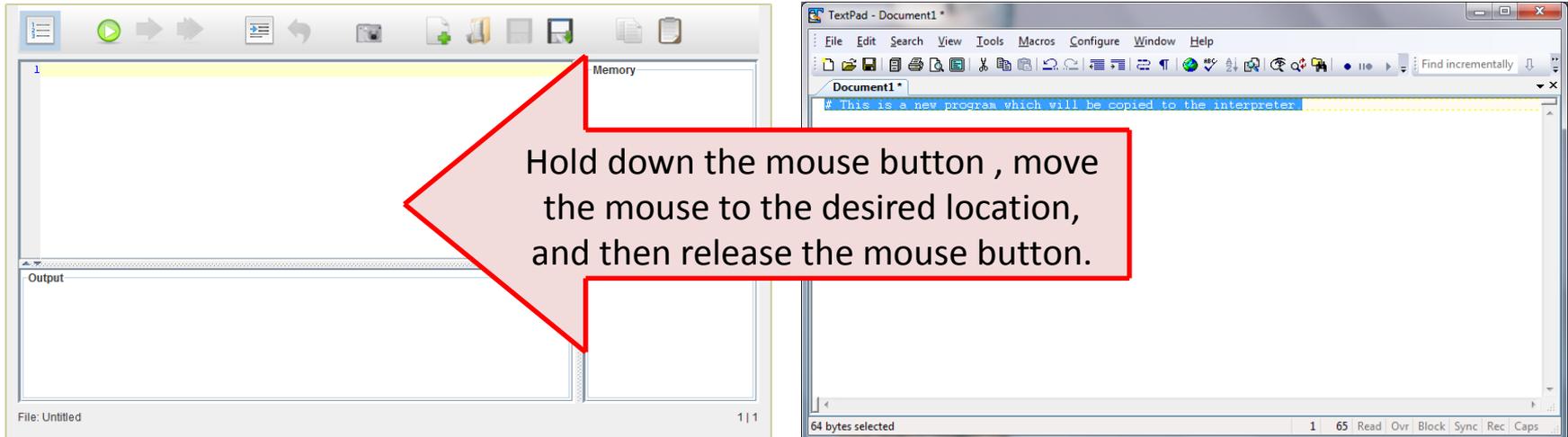
Drag-and-Drop 1



To drag-and-drop text from the interpreter to a native application, select the text you want to copy, and then:

1. Drag it to the native application window; and
2. Drop the text in the desired location.

Drag-and-Drop 2



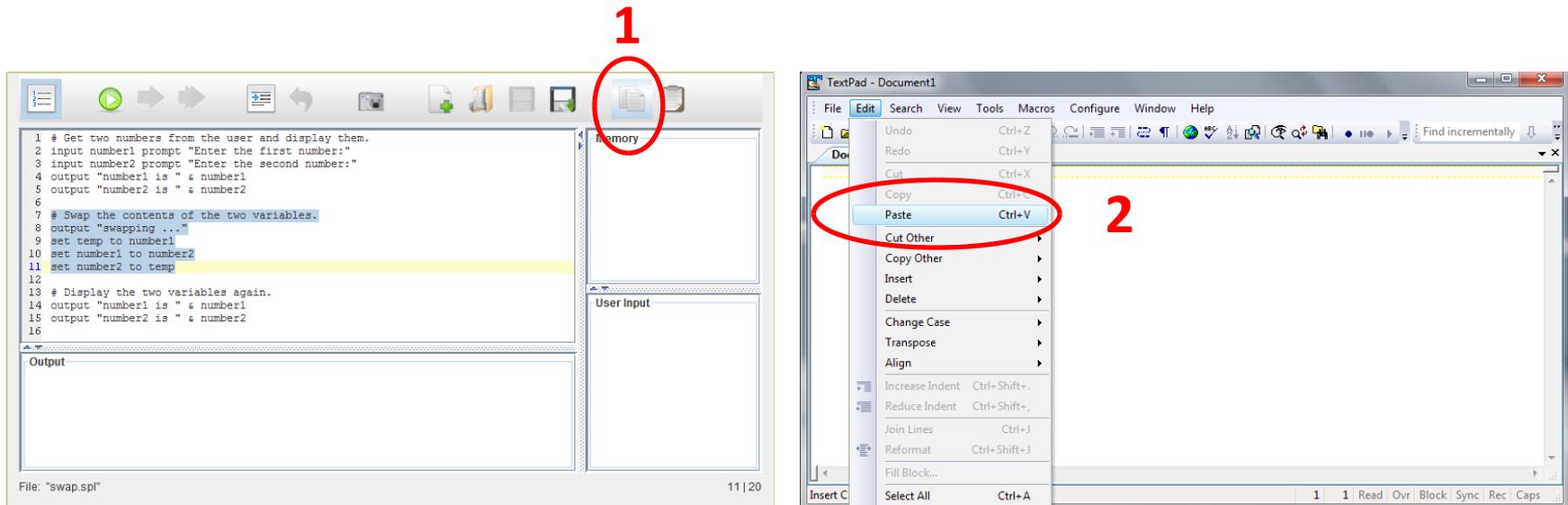
To drag-and-drop text from a native application to the interpreter, select the text you want to copy, and then:

1. Drag it to the interpreter's code window; and
2. Drop the text in the desired location.

Using the Copy and Paste Buttons

- As an alternative to “drag-and-drop”, you can use the **Copy** and **Paste** buttons to transfer text between the interpreter and a native application.
- These actions require access to your local machine and therefore require special permission.

Using the Copy Button



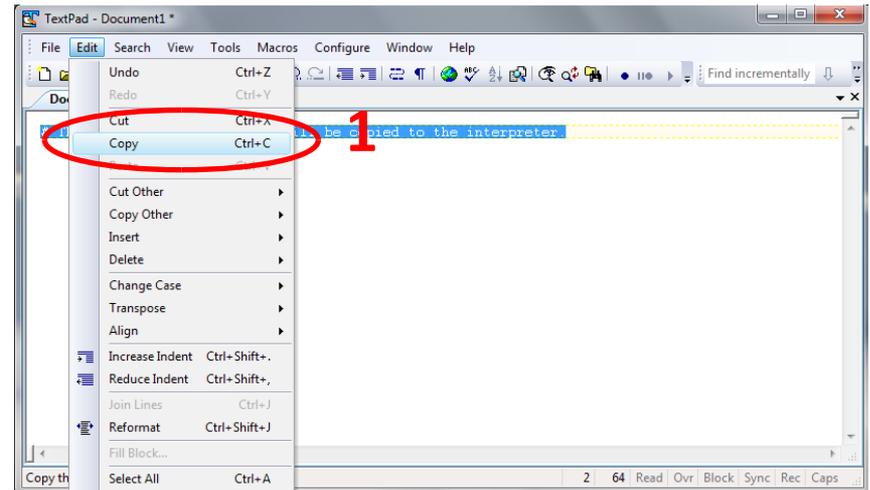
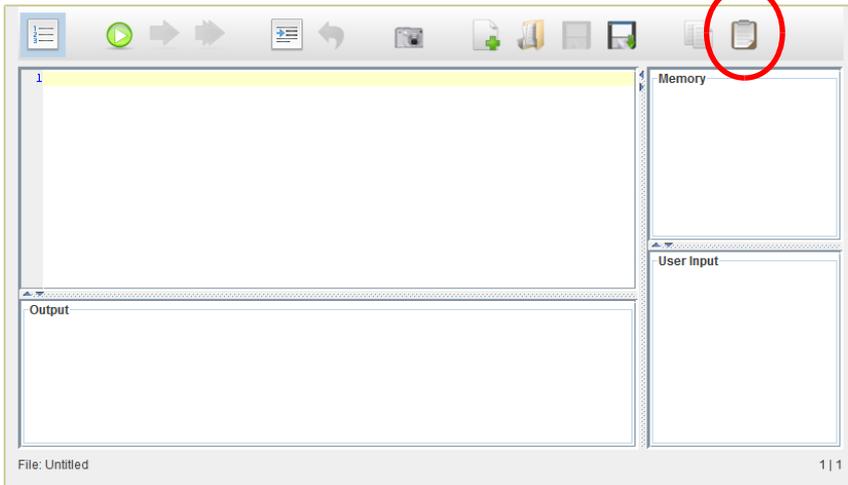
To copy text from the interpreter to a native application, select the text you want to copy and then:

1. Press the **Copy** button in the interpreter; and
2. Use the native application's "paste" function.

This requires local machine permissions.

Using the Paste Button

2



To paste text from a native application to the interpreter, select the text you want to copy and then:

1. Use the native application's "copy" function; and
2. Press the **Paste** button in the interpreter.

This requires local machine permissions.

End of Presentation

