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

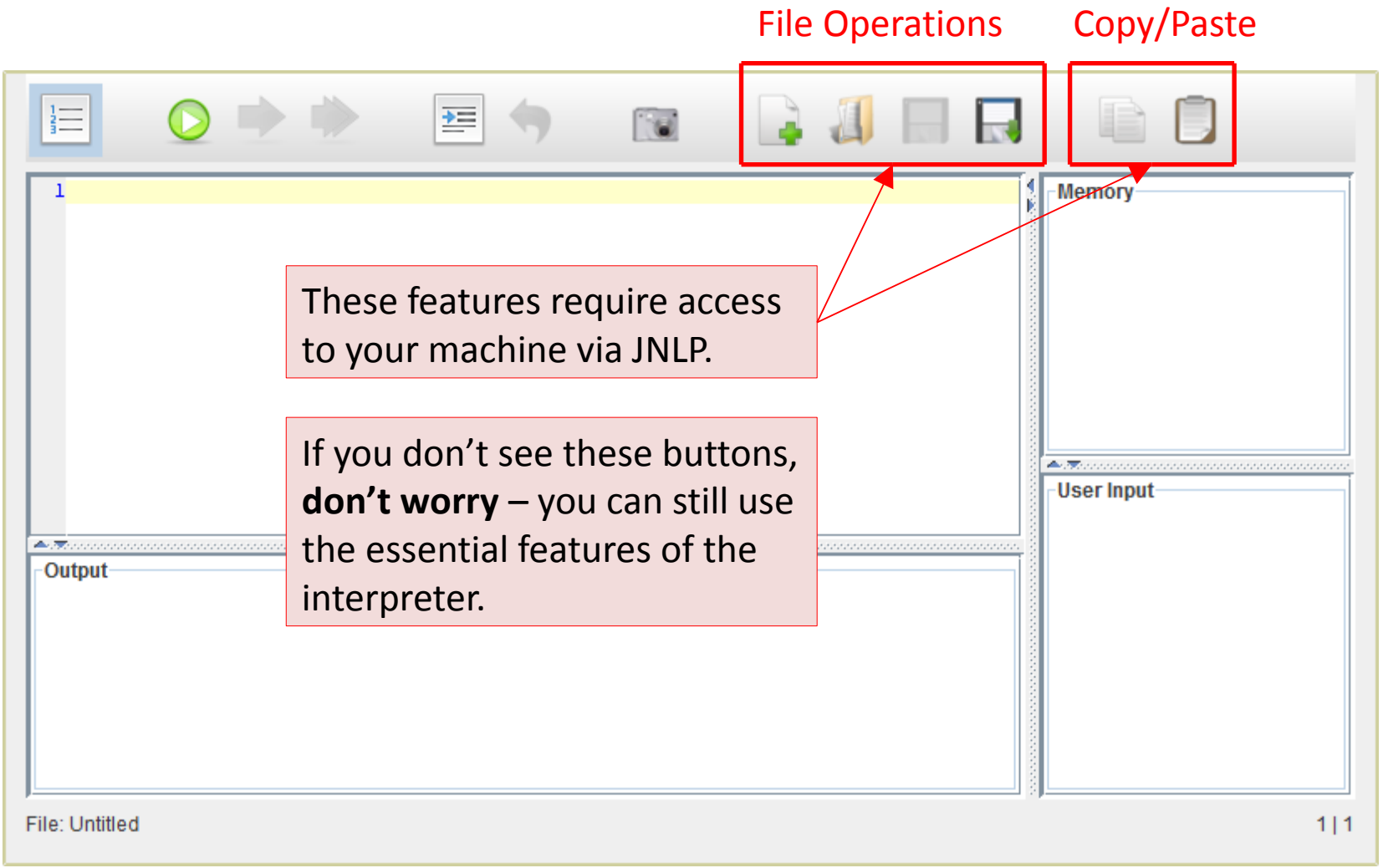
- The SPL interpreter requires a Java virtual machine (version 6 or later) and any modern browser (IE, Firefox, Chrome, Safari, etc.)
- The essential features of the SPL interpreter are available in all computing environments that meet those conditions. These features are:
  - Creating and running programs
  - 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



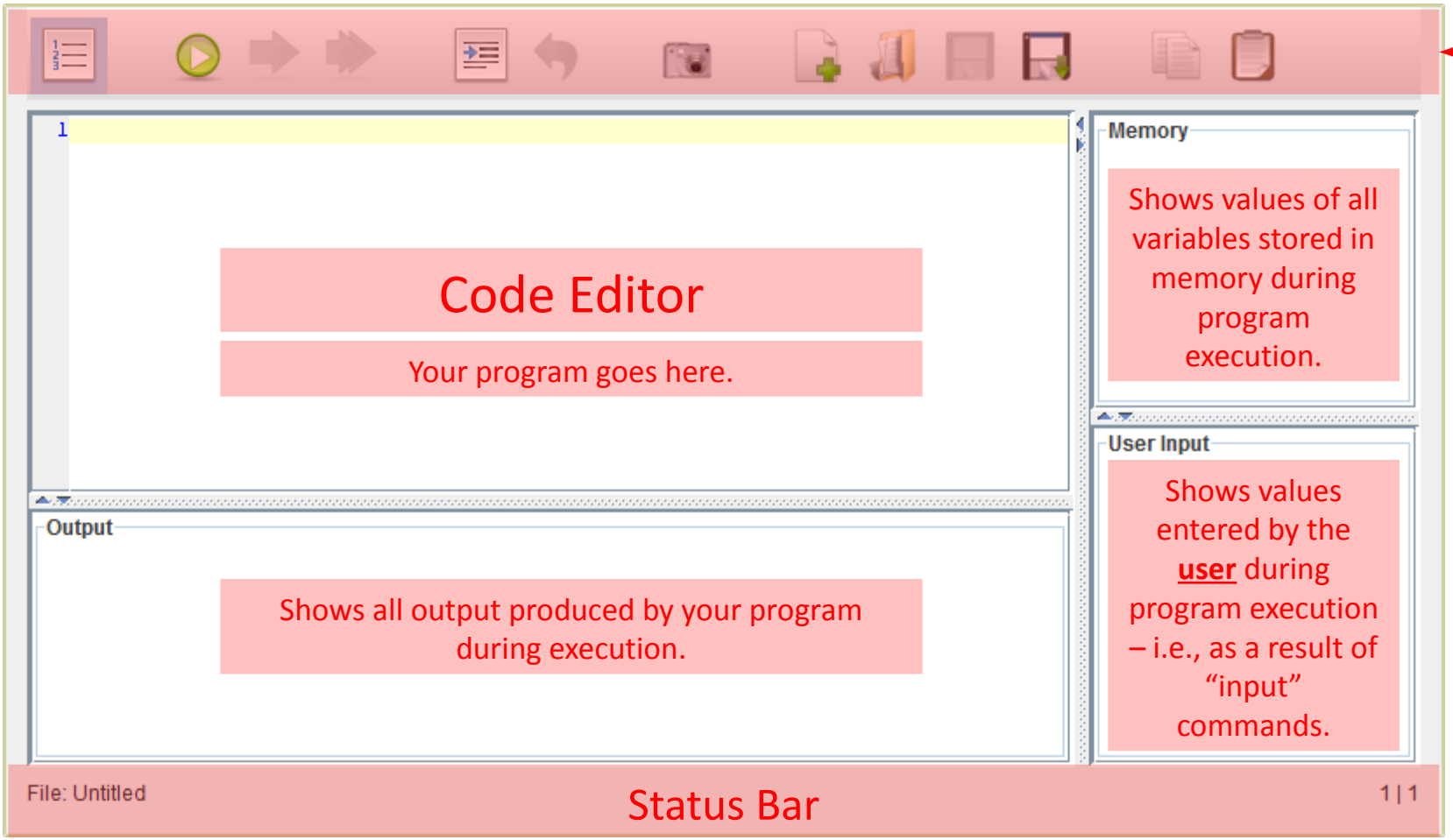
These features require access to your machine via JNLP.

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

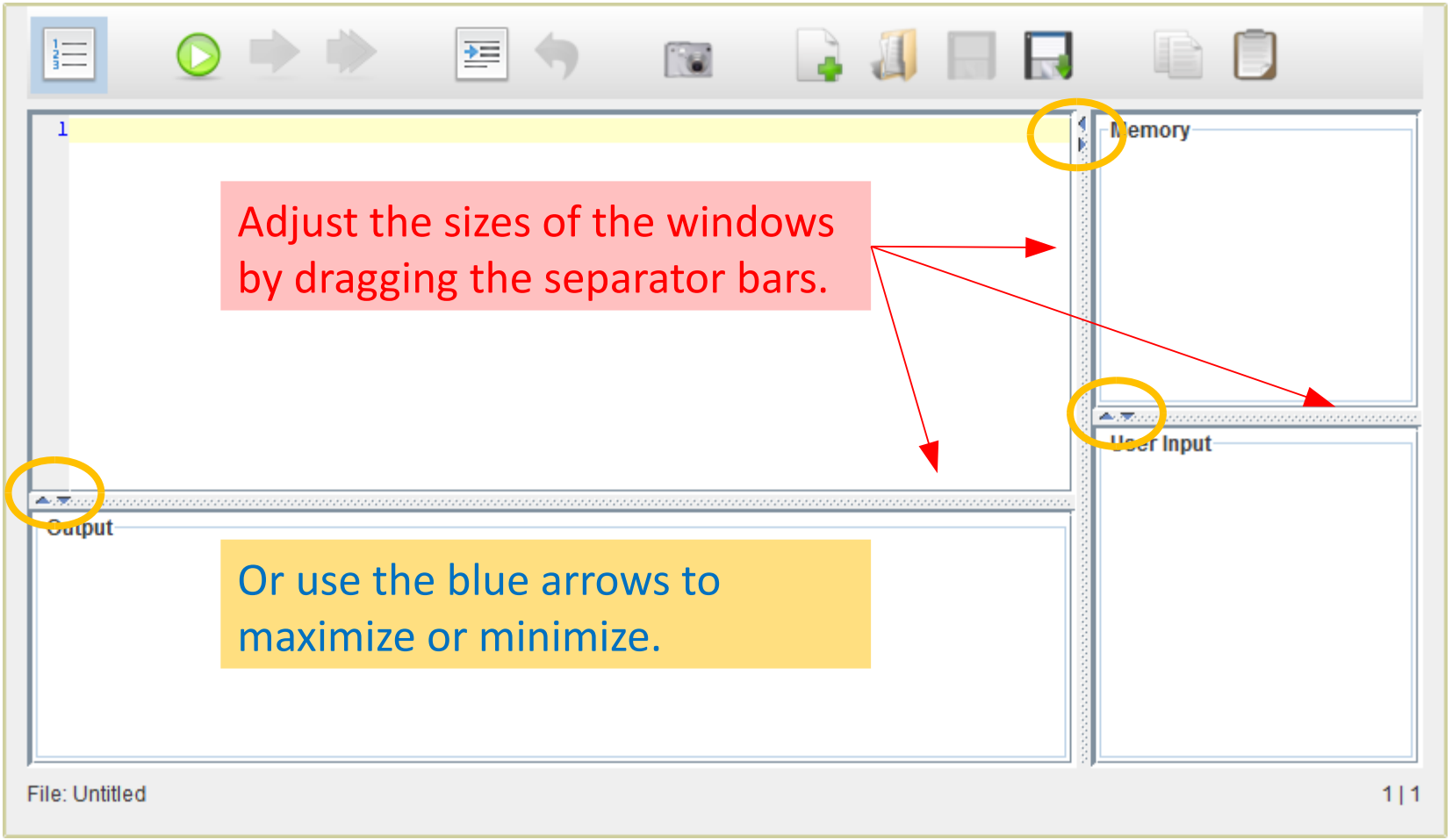
Proceed with tutorial ...

Overview of Application Components

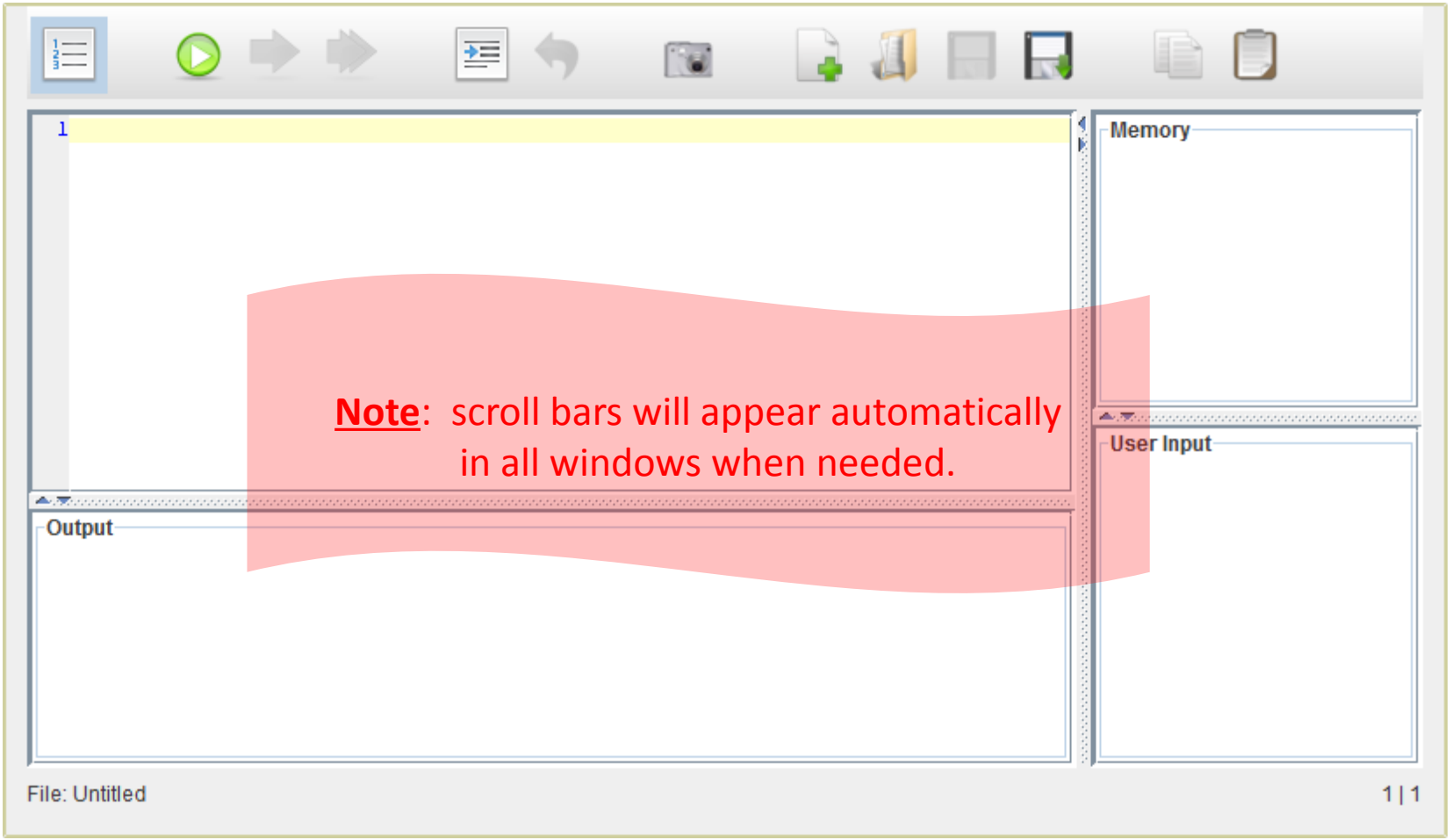
Sections


Tool Bar
1
Code Editor
Your program goes here.
Memory
Shows values of all variables stored in memory during program execution.
User Input
Shows values entered by the user during program execution – i.e., as a result of "input" commands.
Output
Shows all output produced by your program during execution.
File: Untitled
Status Bar
1 | 1

Windows

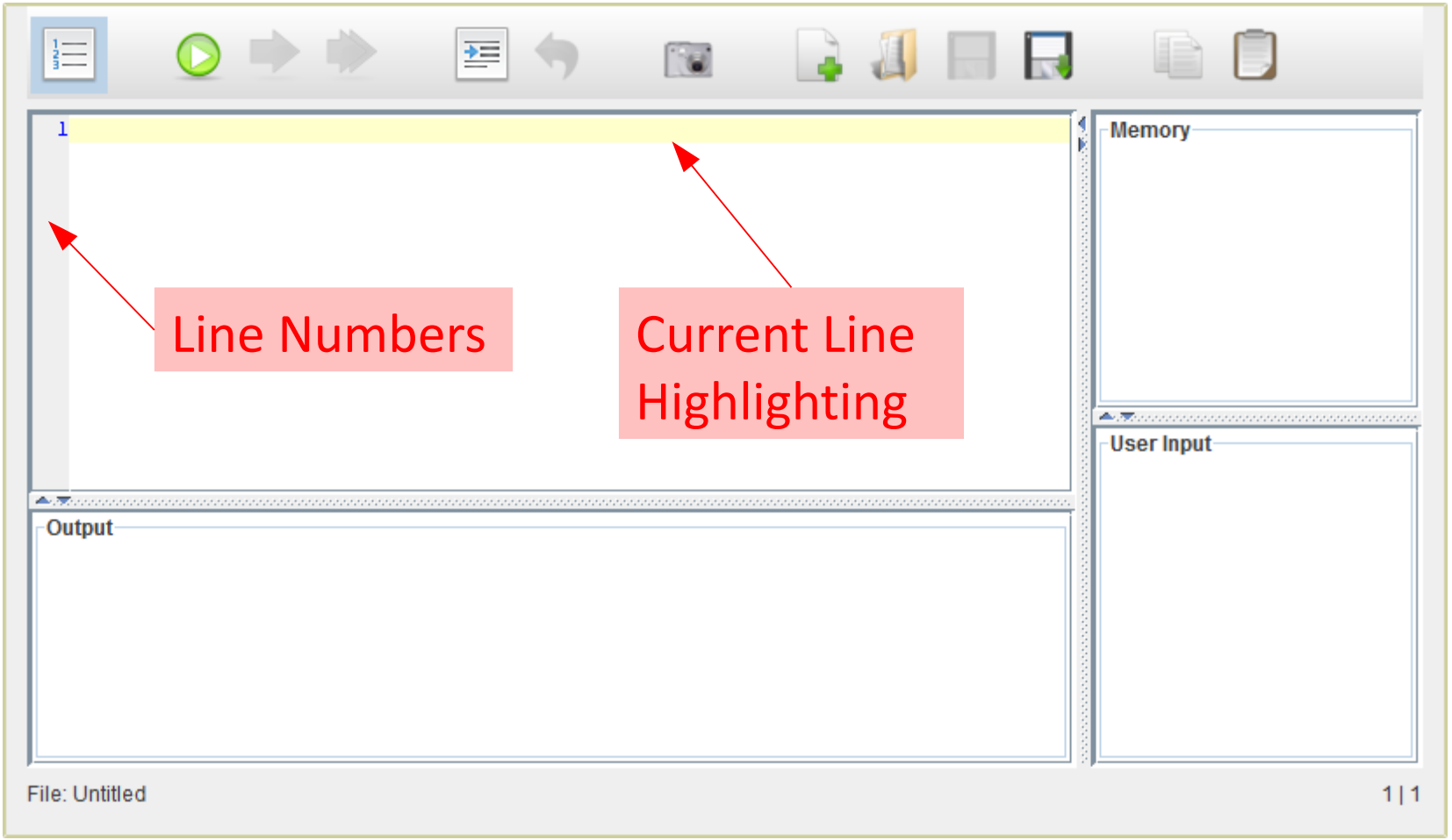

1
Memory
Adjust the sizes of the windows by dragging the separator bars.
User Input
Output
Or use the blue arrows to maximize or minimize.
File: Untitled
1 | 1

Windows

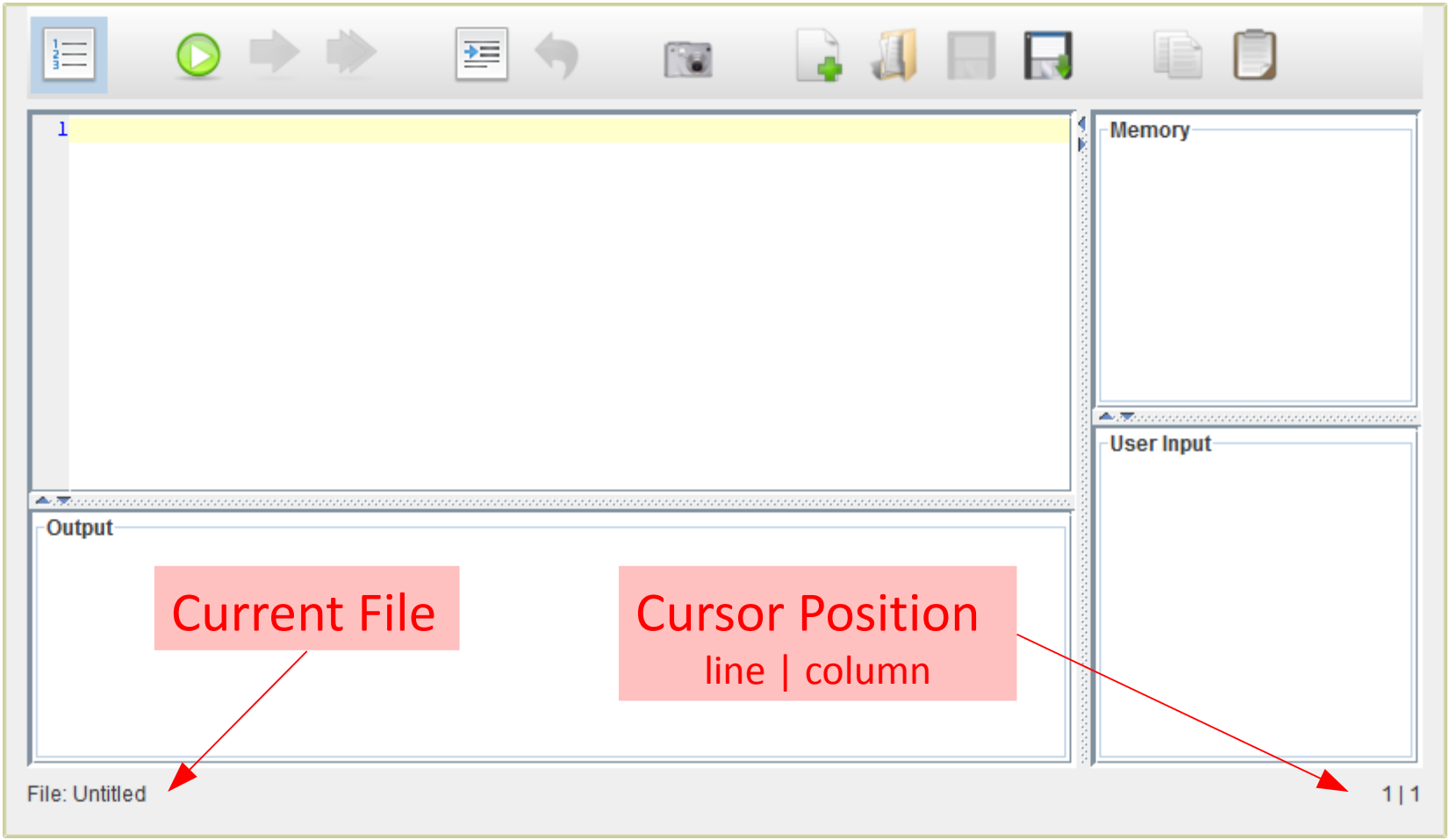


Note: scroll bars will appear automatically in all windows when needed.

Windows

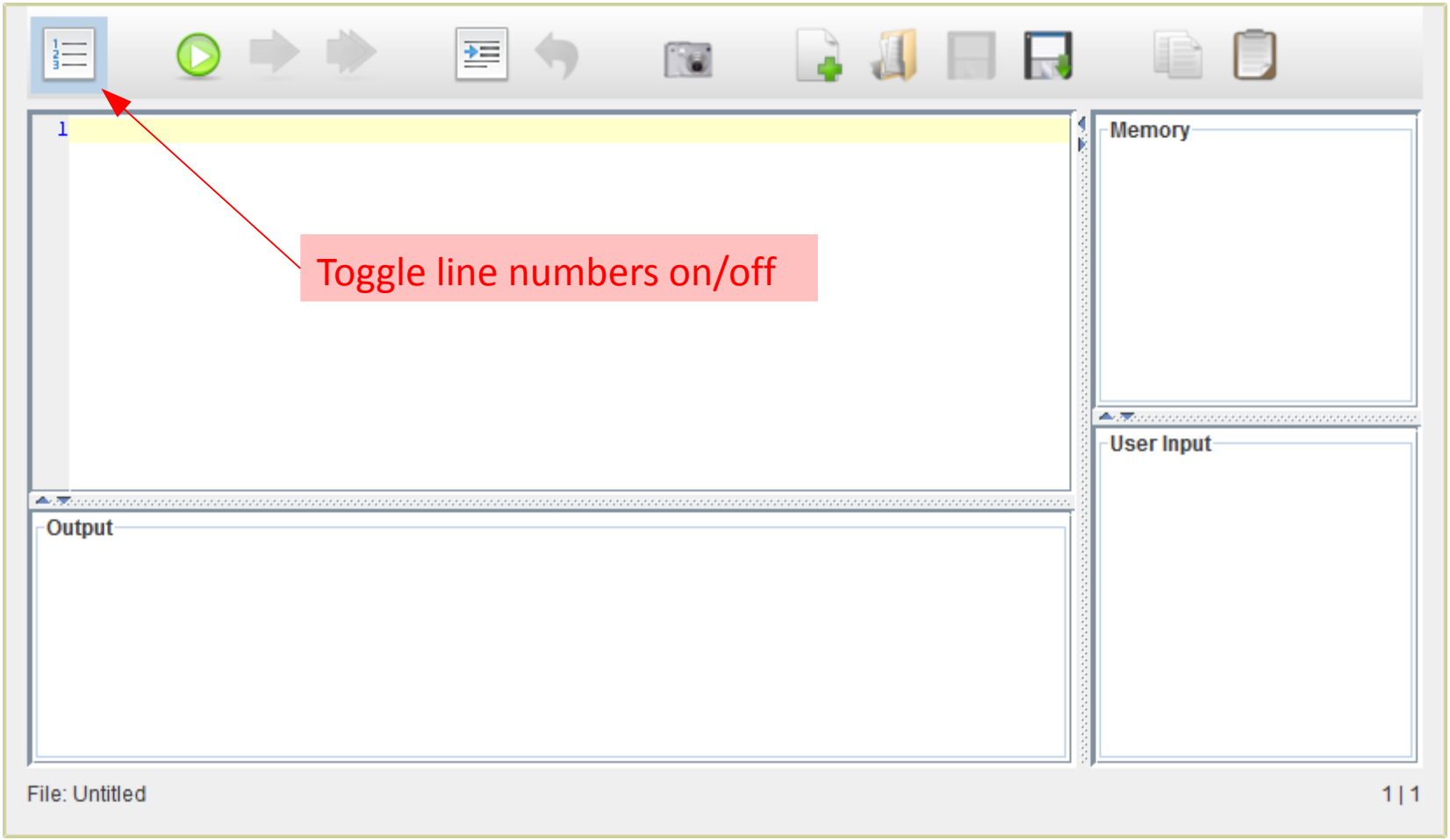

1
Line Numbers
Current Line Highlighting
Memory
User Input
Output
File: Untitled
1 | 1

Status Bar

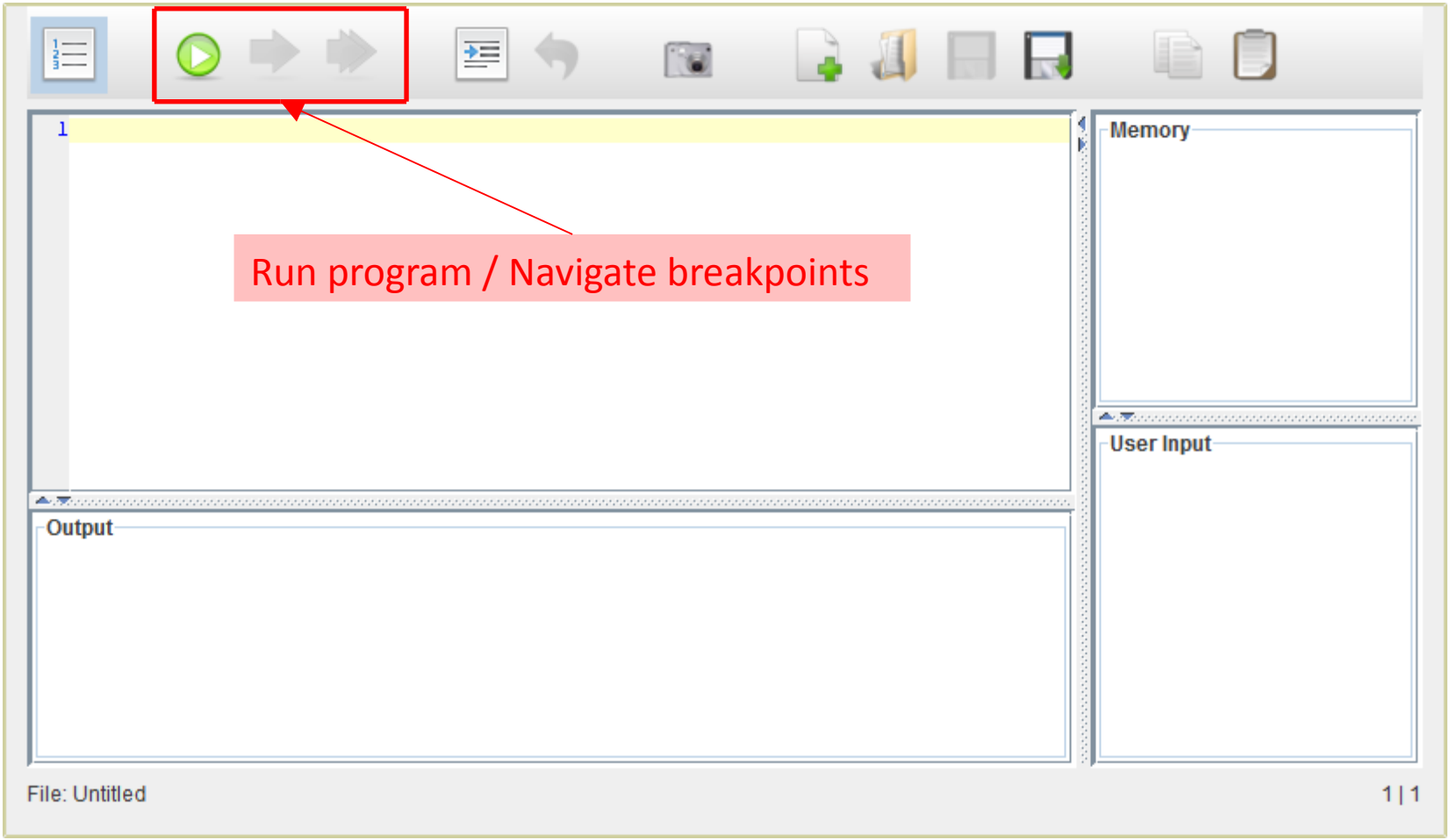

1
Memory
User Input
Output
Current File
Cursor Position
line | column
File: Untitled
1 | 1

The Tool Bar – Details

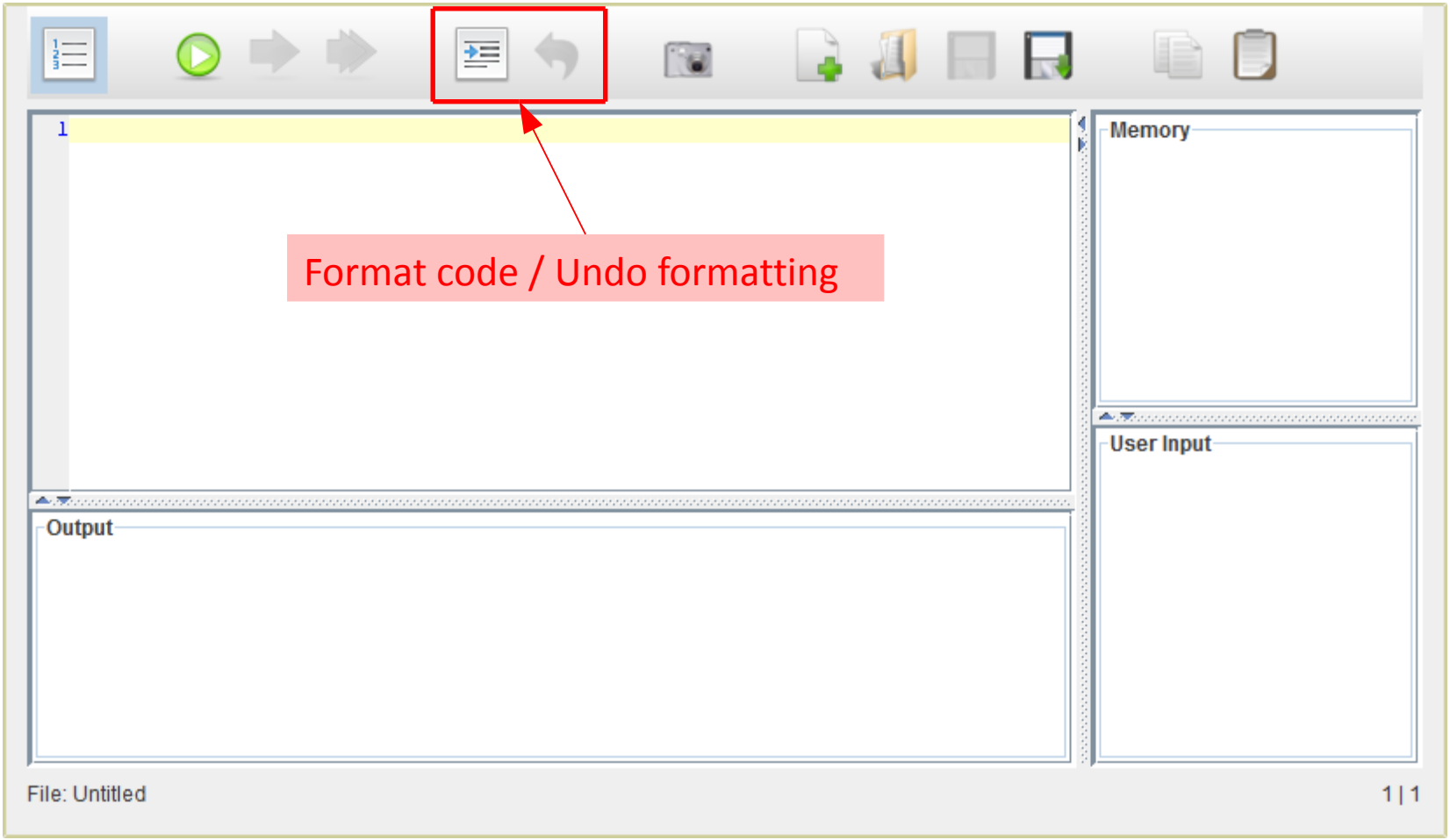
Tool Bar


Toggle line numbers on/off
Memory
User Input
Output
File: Untitled
1 | 1

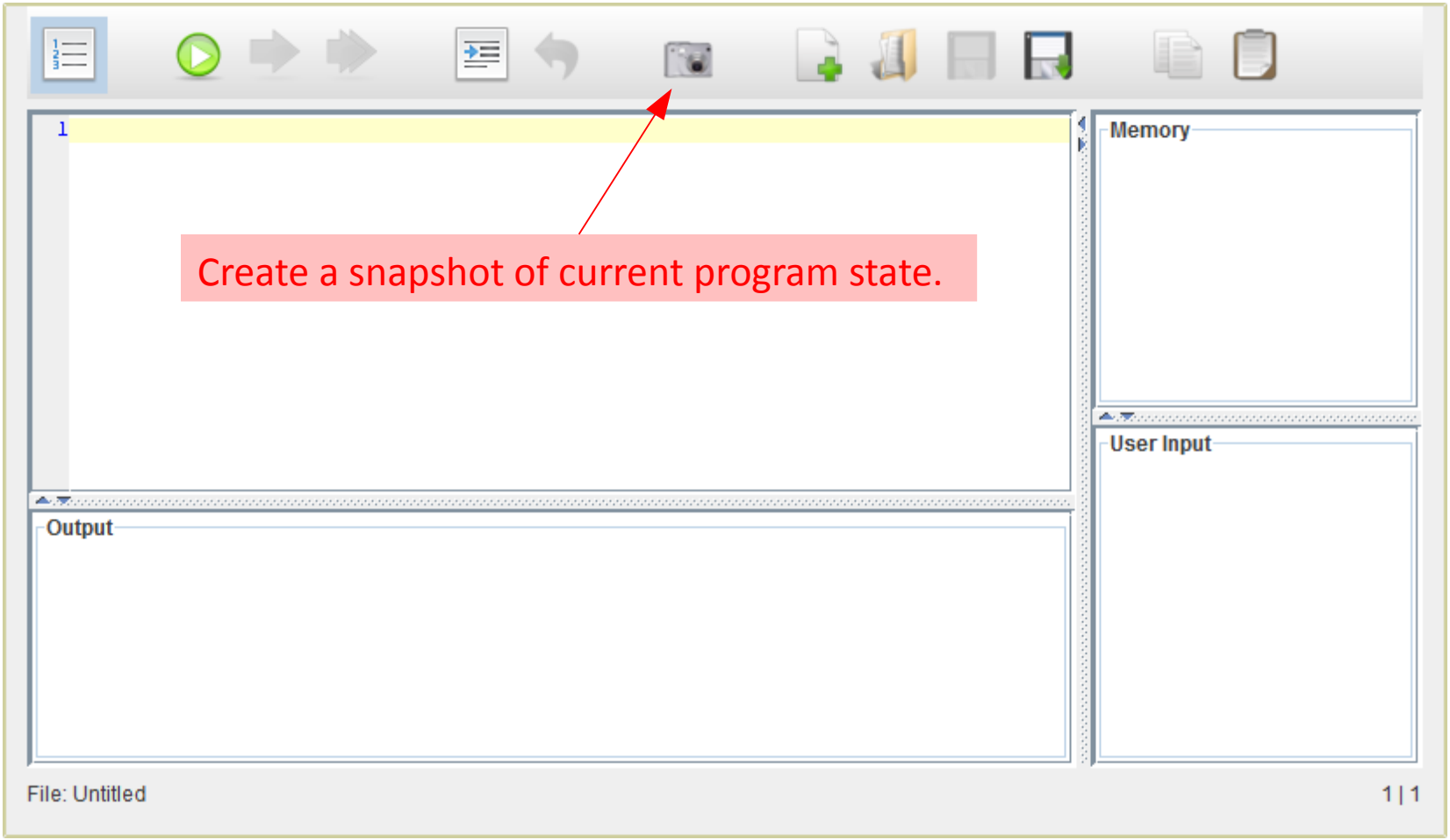
Tool Bar


Run program / Navigate breakpoints
Memory
User Input
Output
File: Untitled
1 | 1

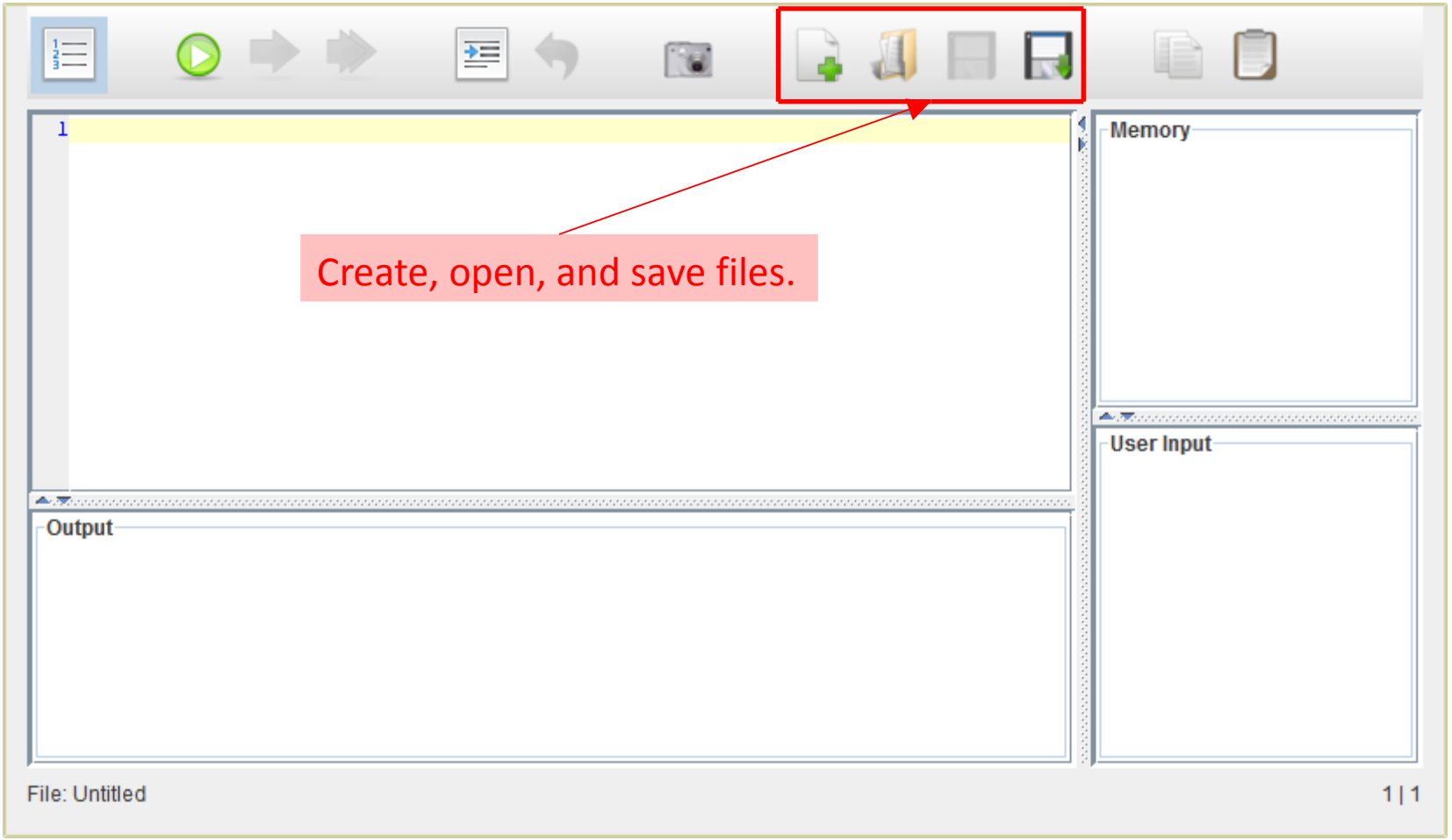
Tool Bar


Format code / Undo formatting
1
Memory
User Input
Output
File: Untitled
1 | 1

Tool Bar

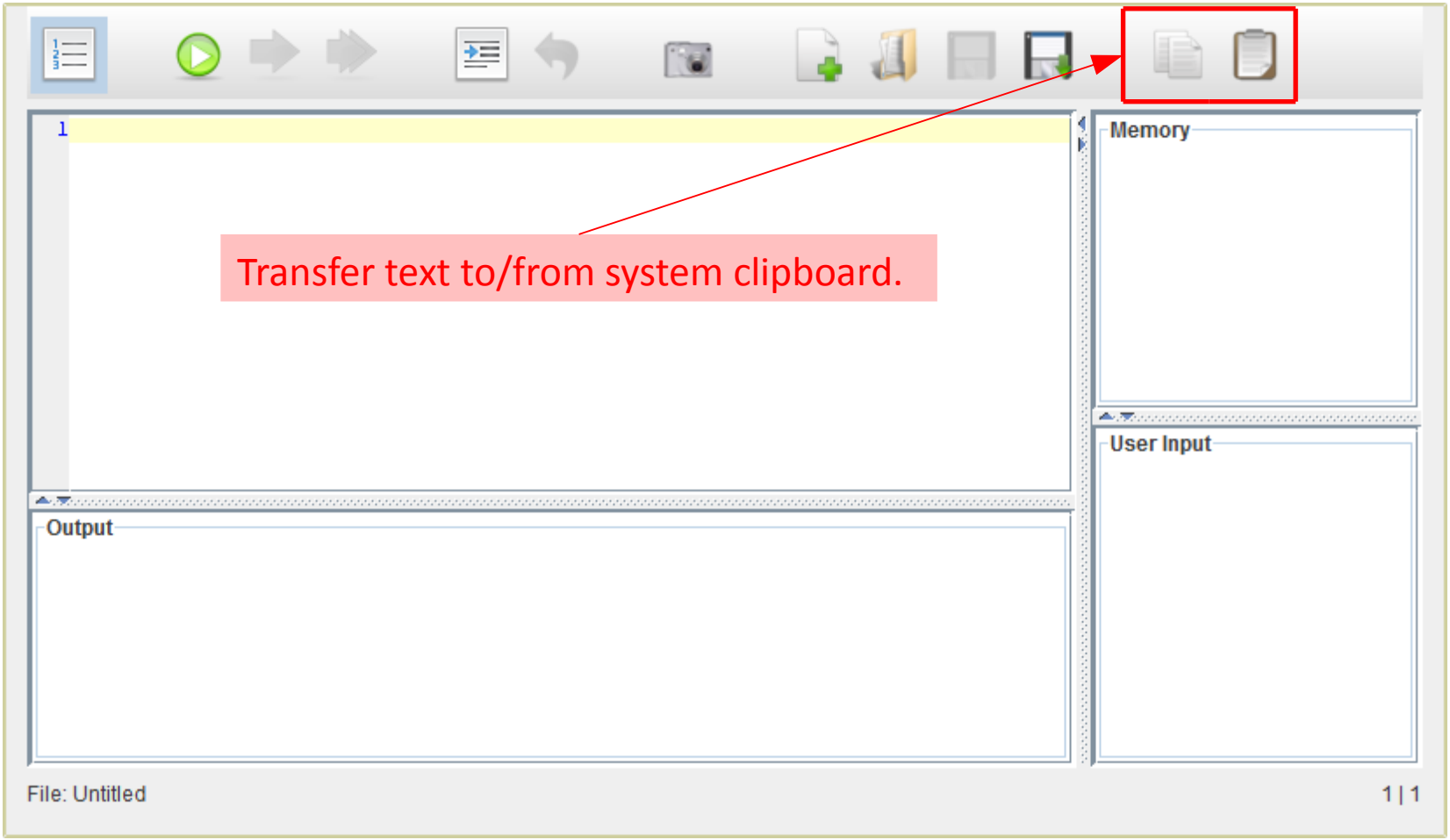

Create a snapshot of current program state.
Memory
User Input
Output
File: Untitled
1 | 1

Tool Bar

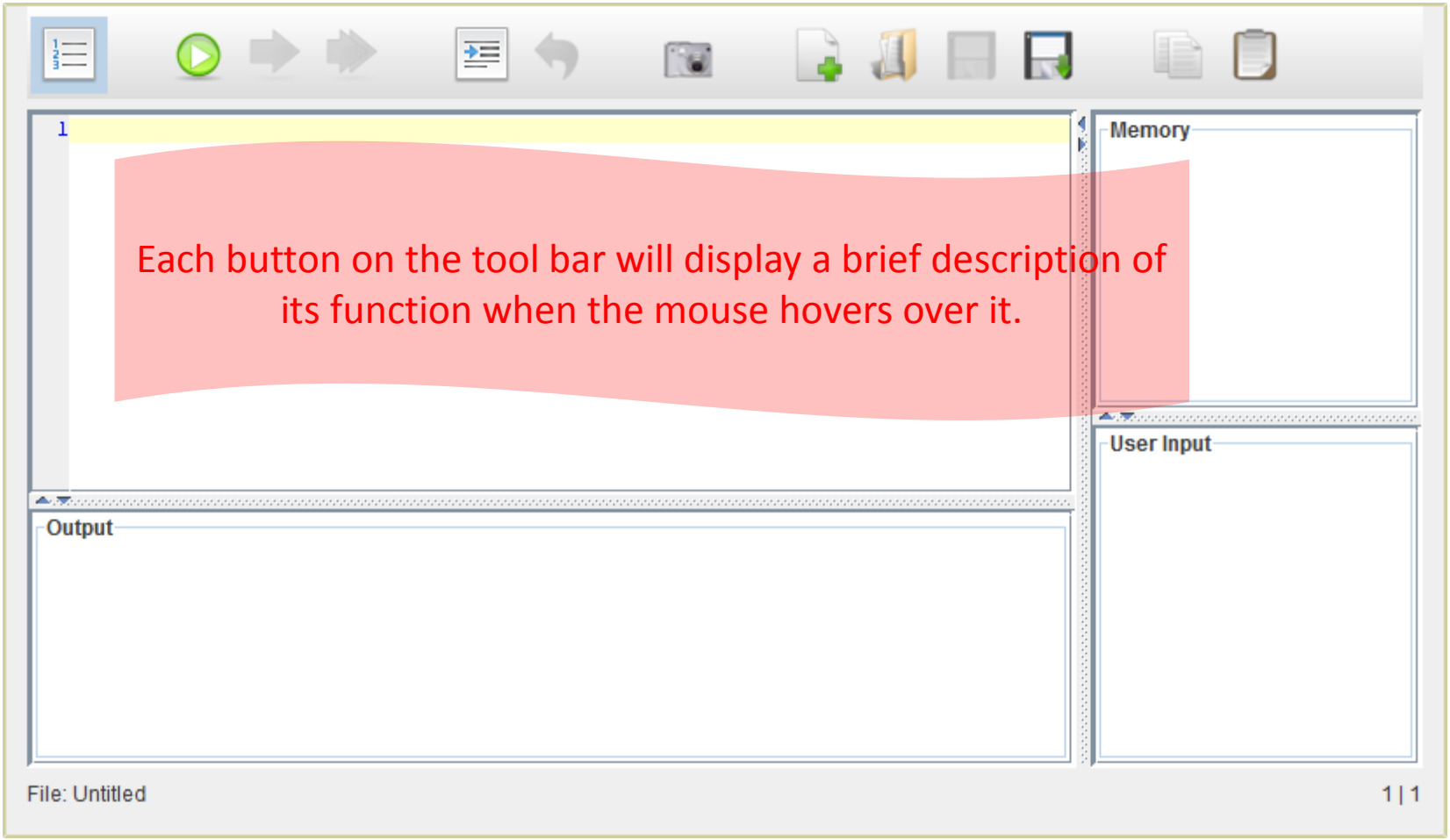


Create, open, and save files.

Tool Bar

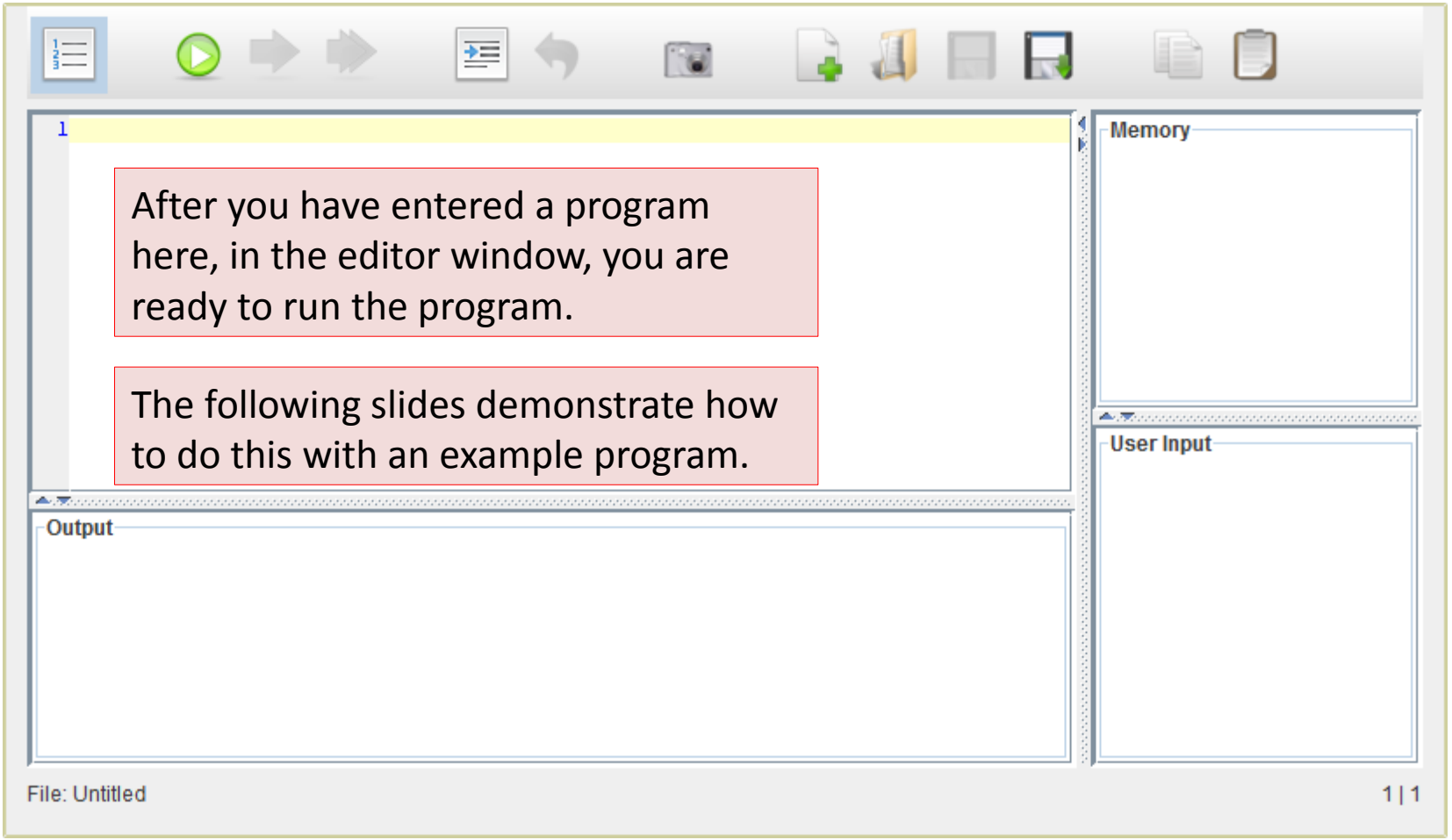

Transfer text to/from system clipboard.
Memory
User Input
Output
File: Untitled
1 | 1

Tool Bar

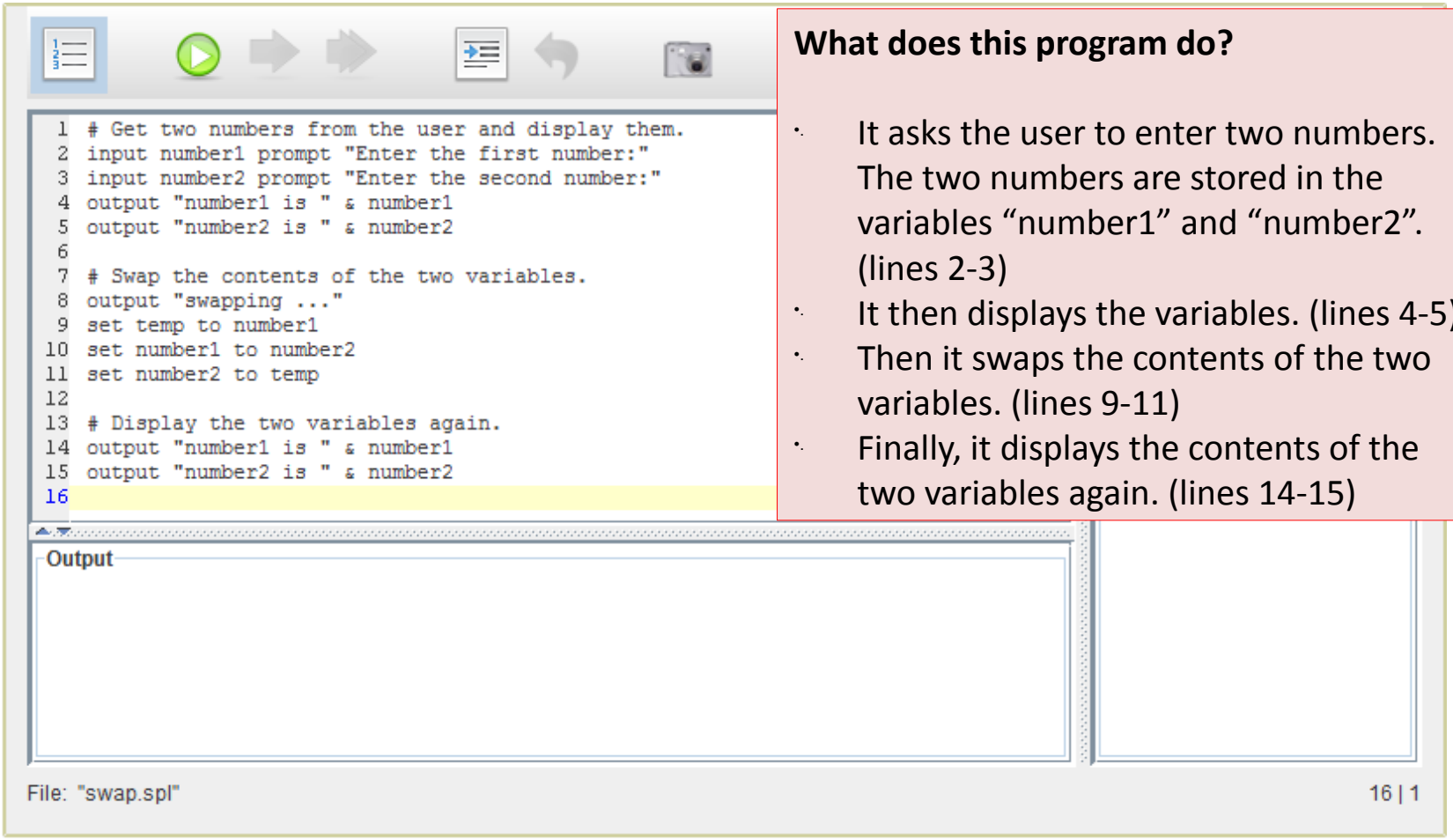

Each button on the tool bar will display a brief description of its function when the mouse hovers over it.
Memory
User Input
Output
File: Untitled
1 | 1

Running a Program

Running a Program



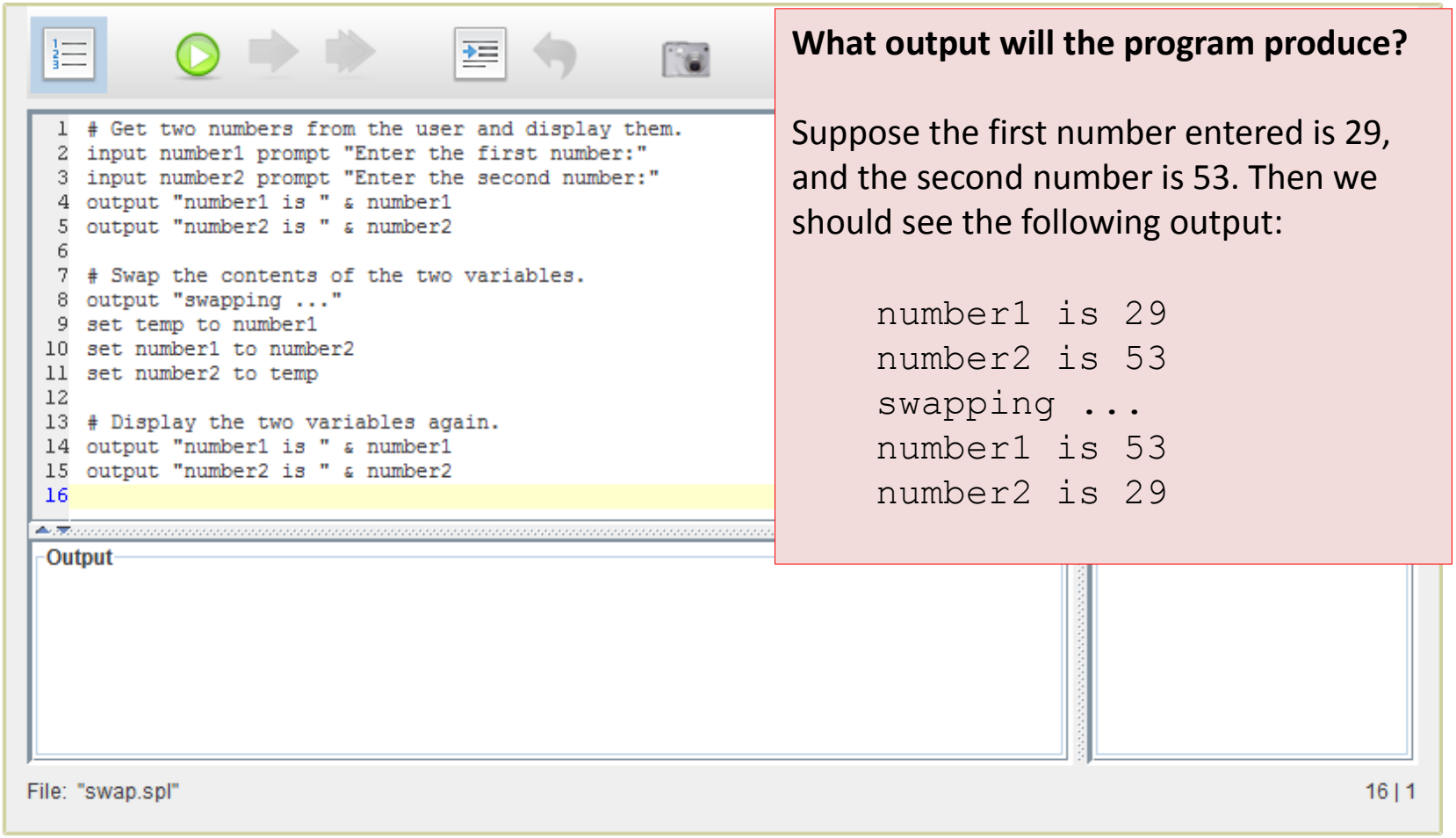
Running a Program



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

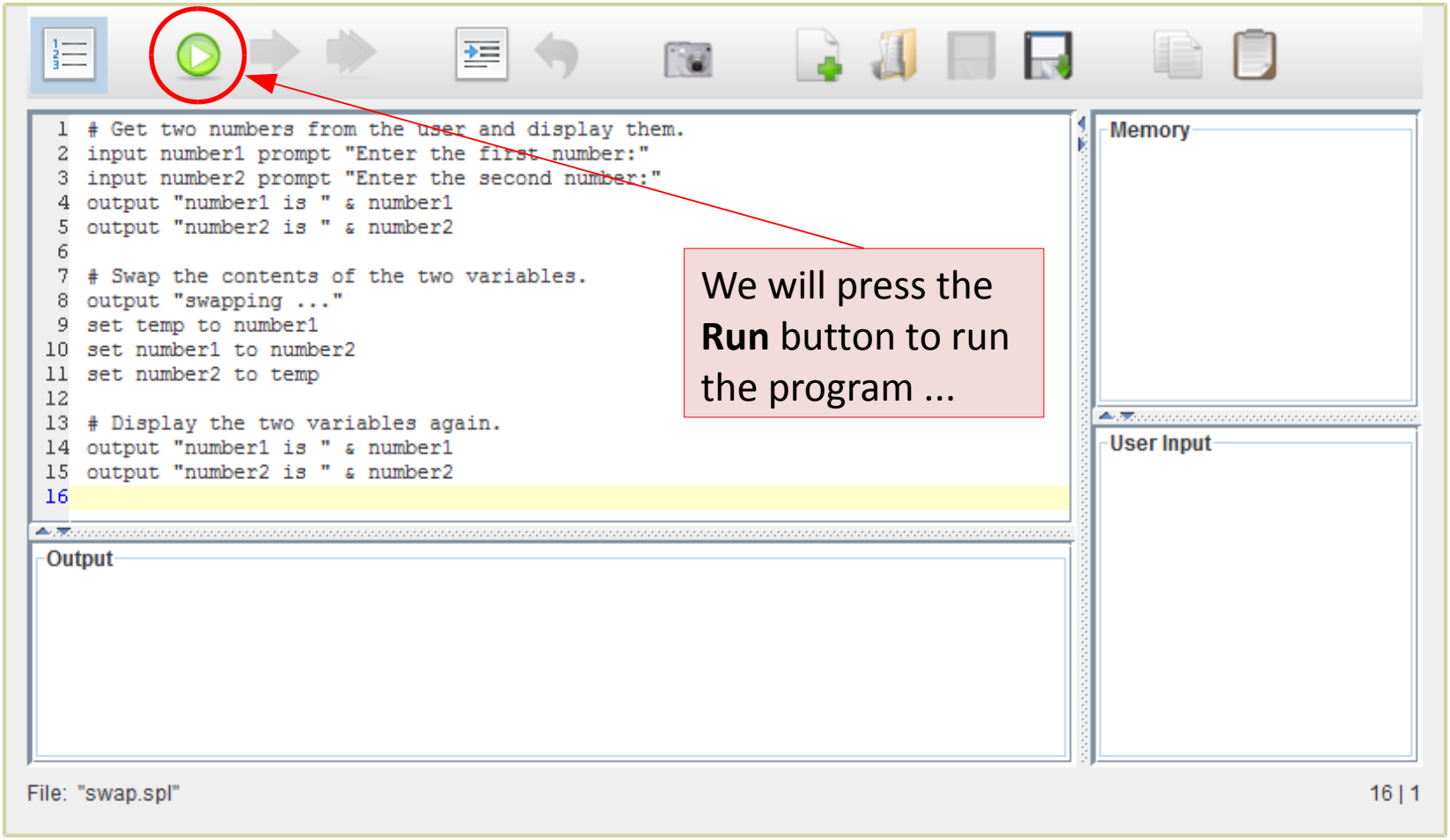


What output will the program produce?

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

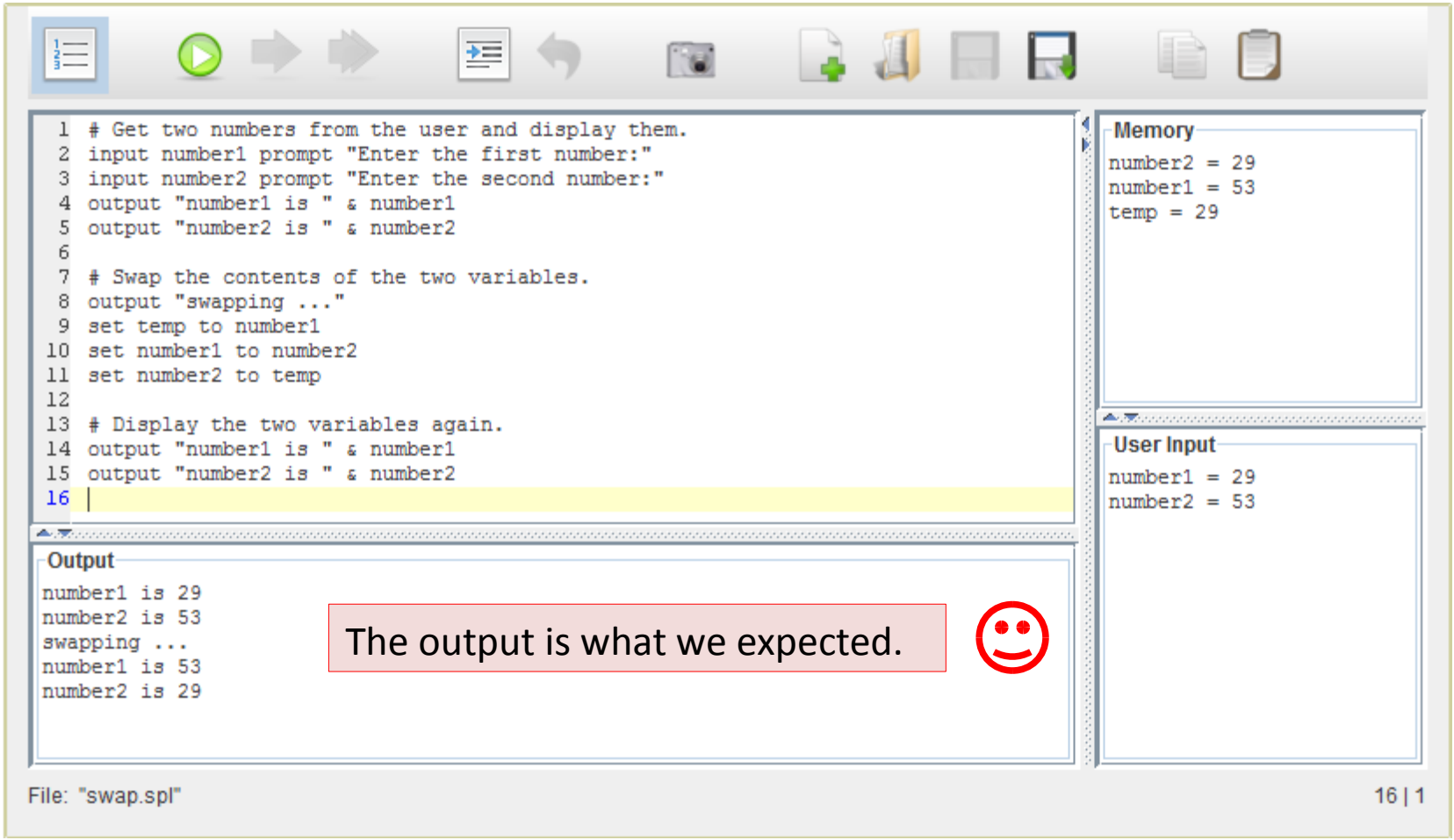
```
number1 is 29
number2 is 53
swapping ...
number1 is 53
number2 is 29
```

Running a Program



... and ...

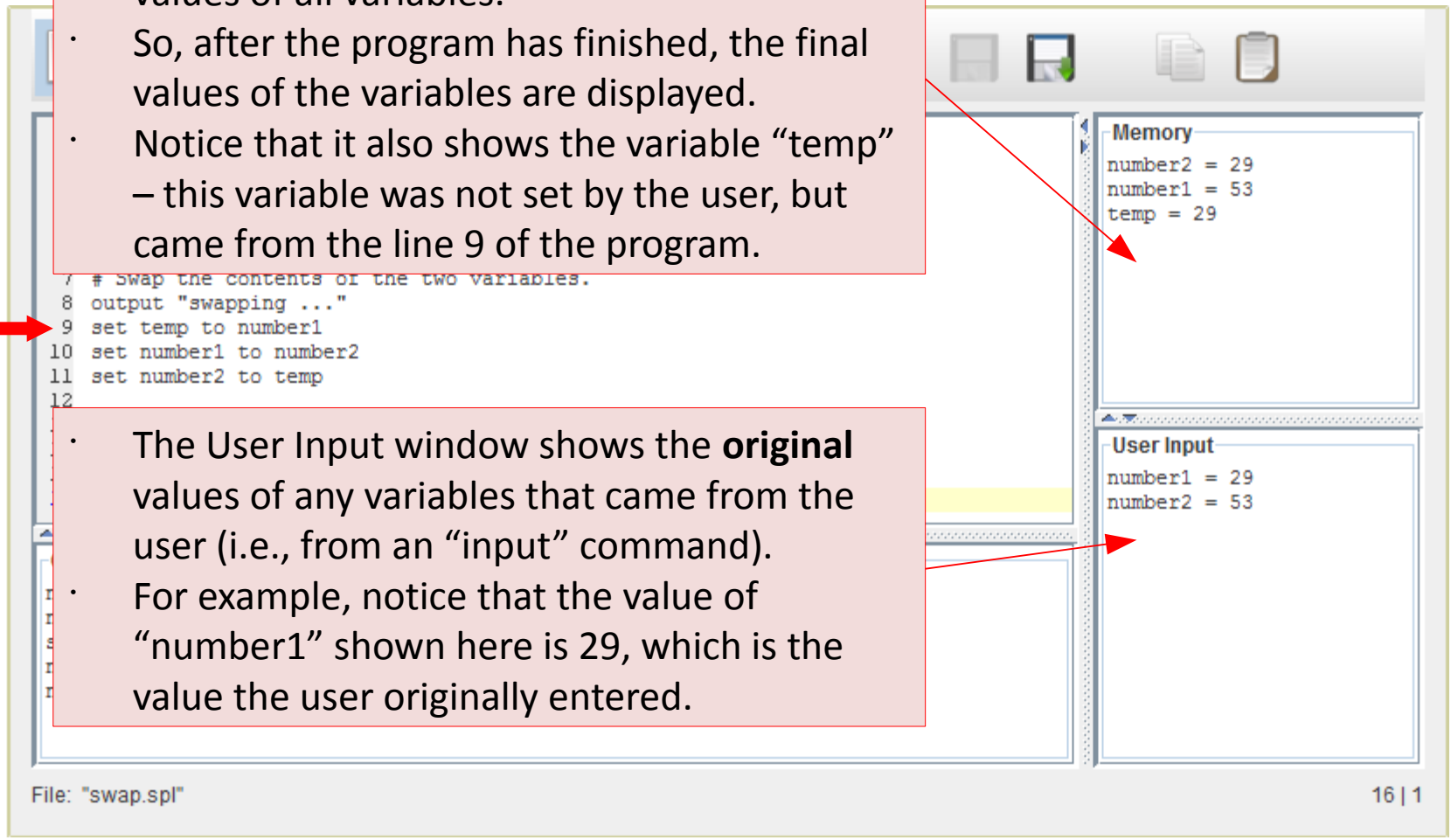
Running a Program



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.



- 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.

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.

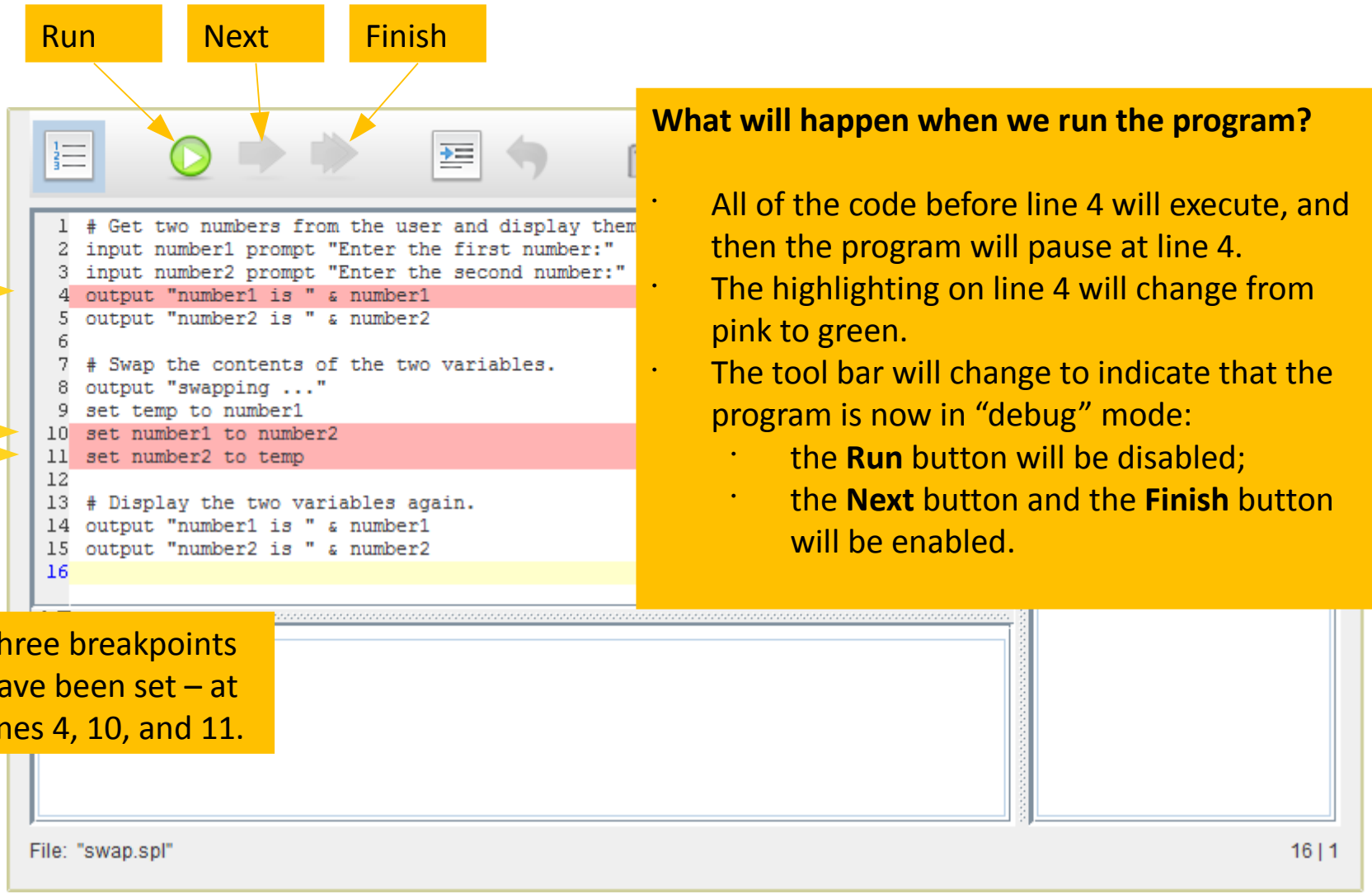
Debugging/Tracing a Program

- 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



What will happen when we run the program?

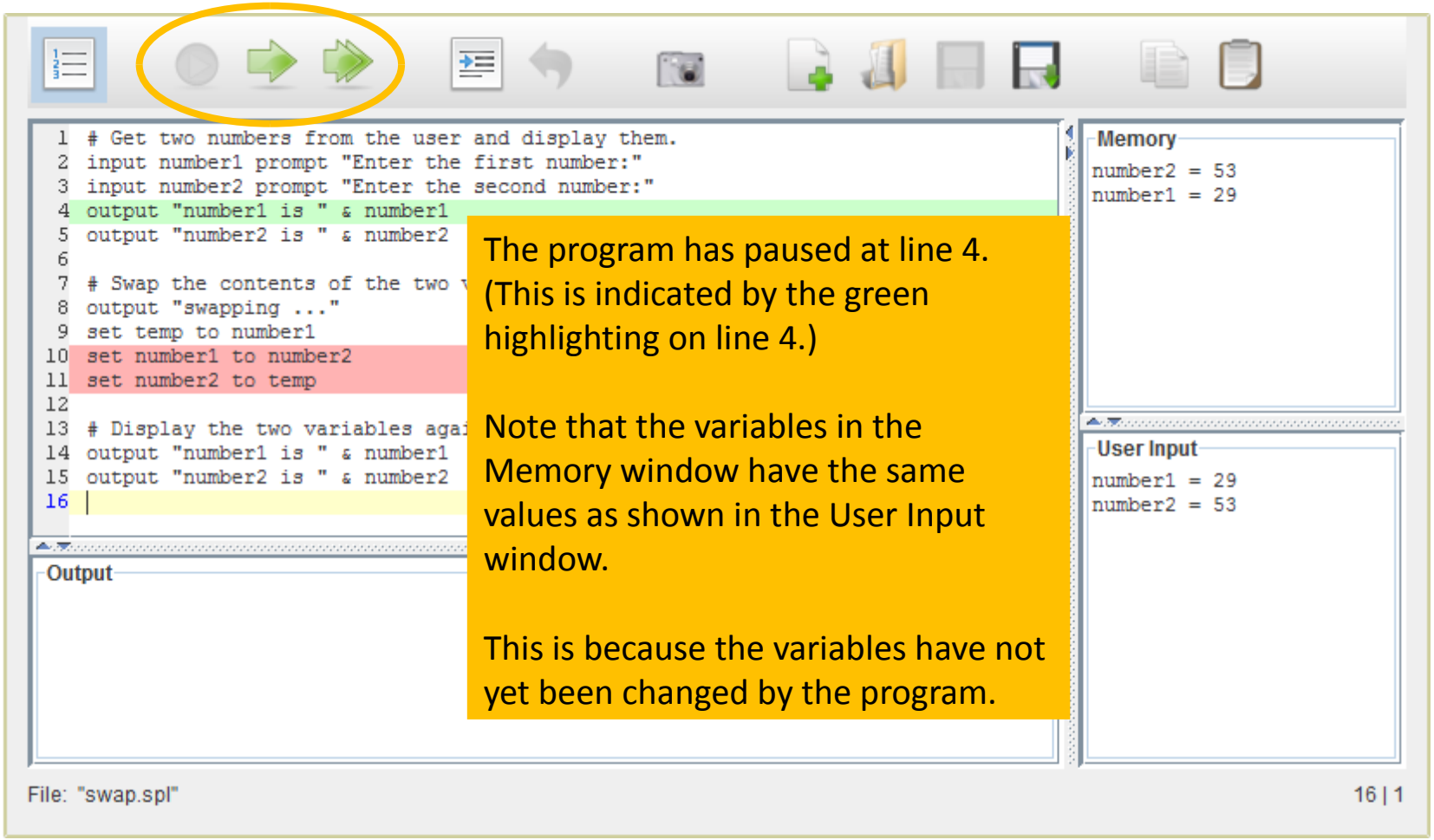
- All of the code before line 4 will execute, and then the program will pause at line 4.
- The highlighting on line 4 will change from pink to green.
- The tool bar will change to indicate that the program is now in "debug" mode:
  - the **Run** button will be disabled;
  - the **Next** button and the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

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

Continued ...

Debugging/Tracing a Program

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



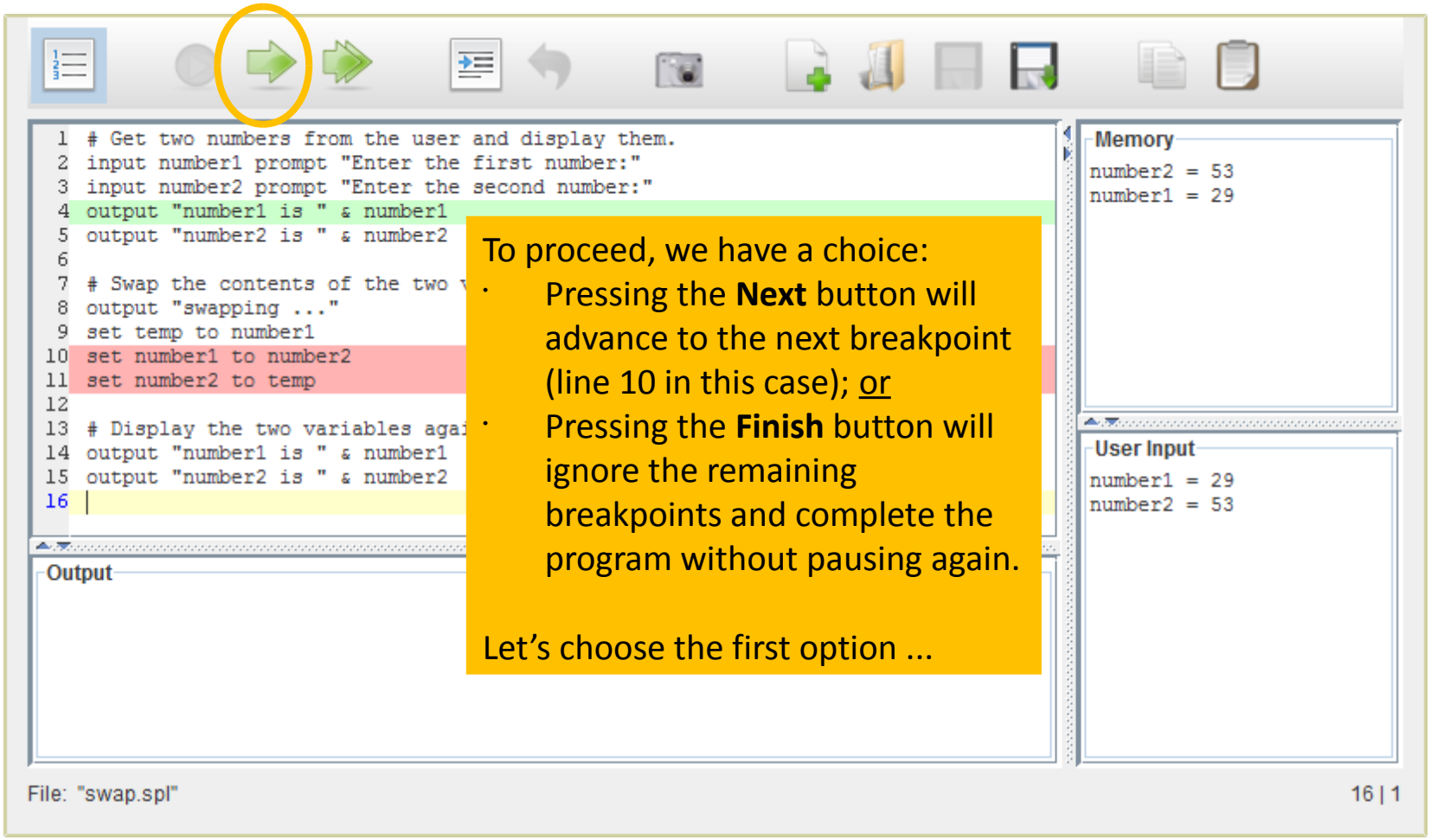
The program has paused at line 4. (This is indicated by the green highlighting on line 4.)

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.

Continued ...

Debugging/Tracing a Program



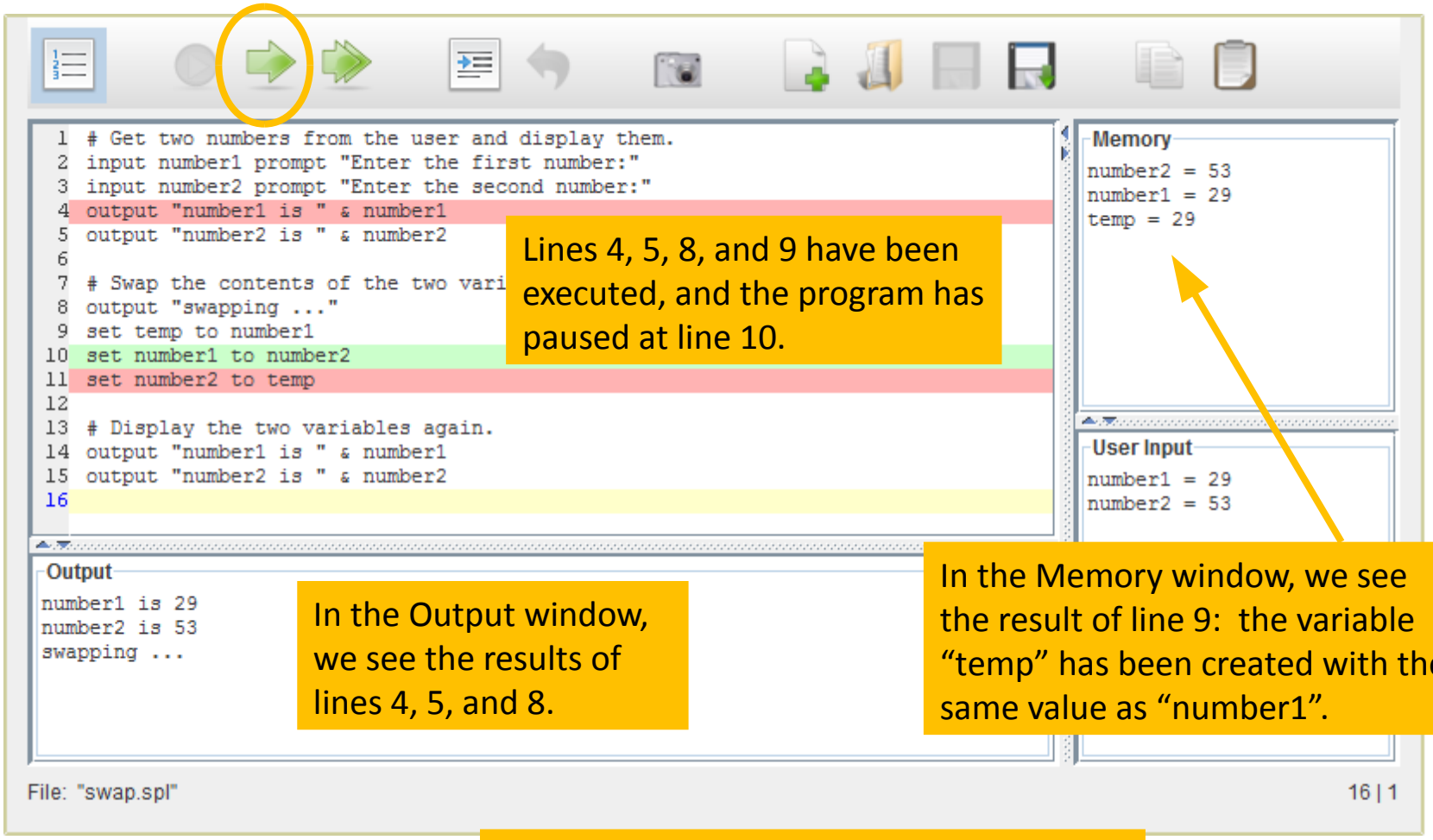
To proceed, we have a choice:

- 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 ...

Continued ...

Debugging/Tracing a Program



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

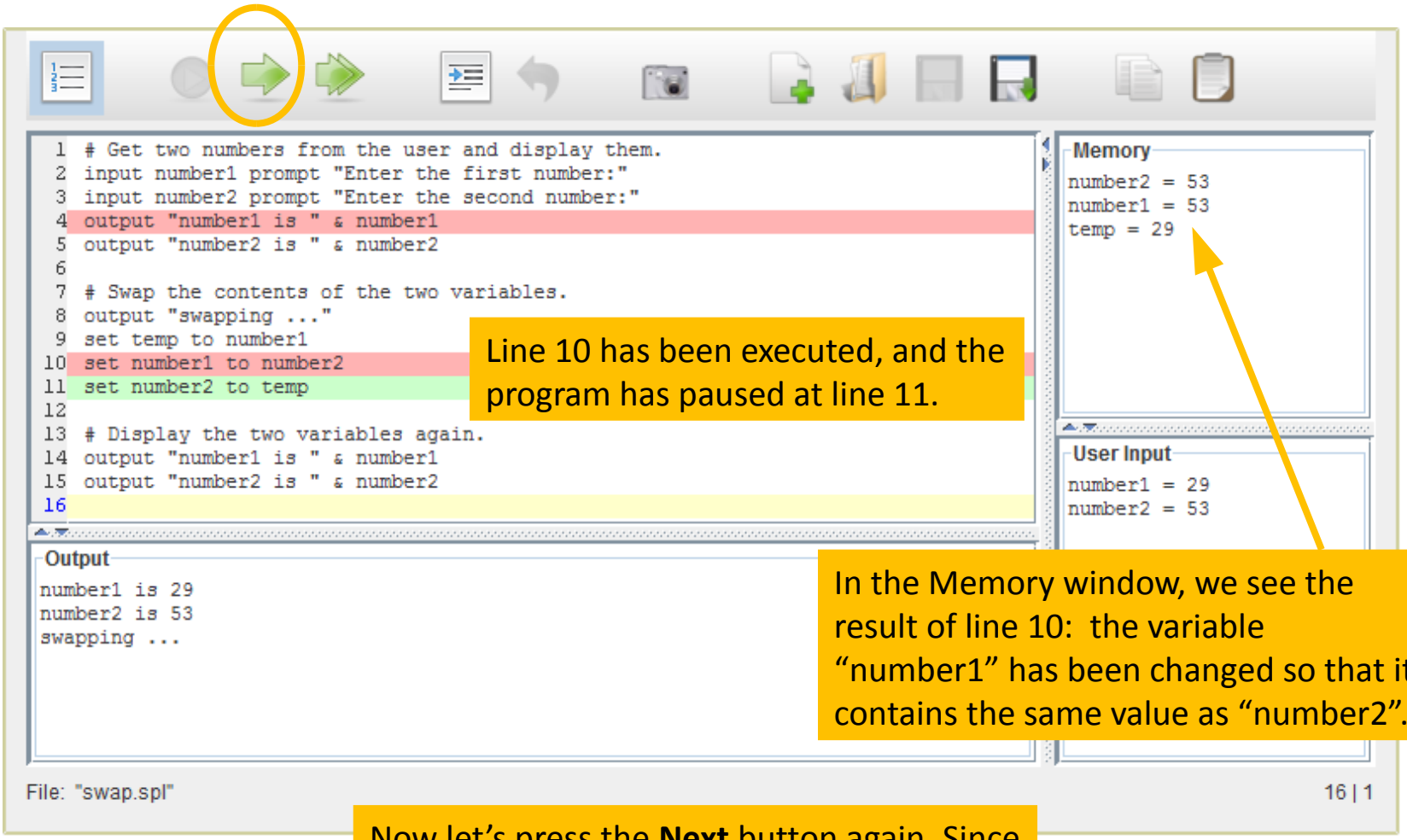
In the Output window, we see the results of lines 4, 5, and 8.

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



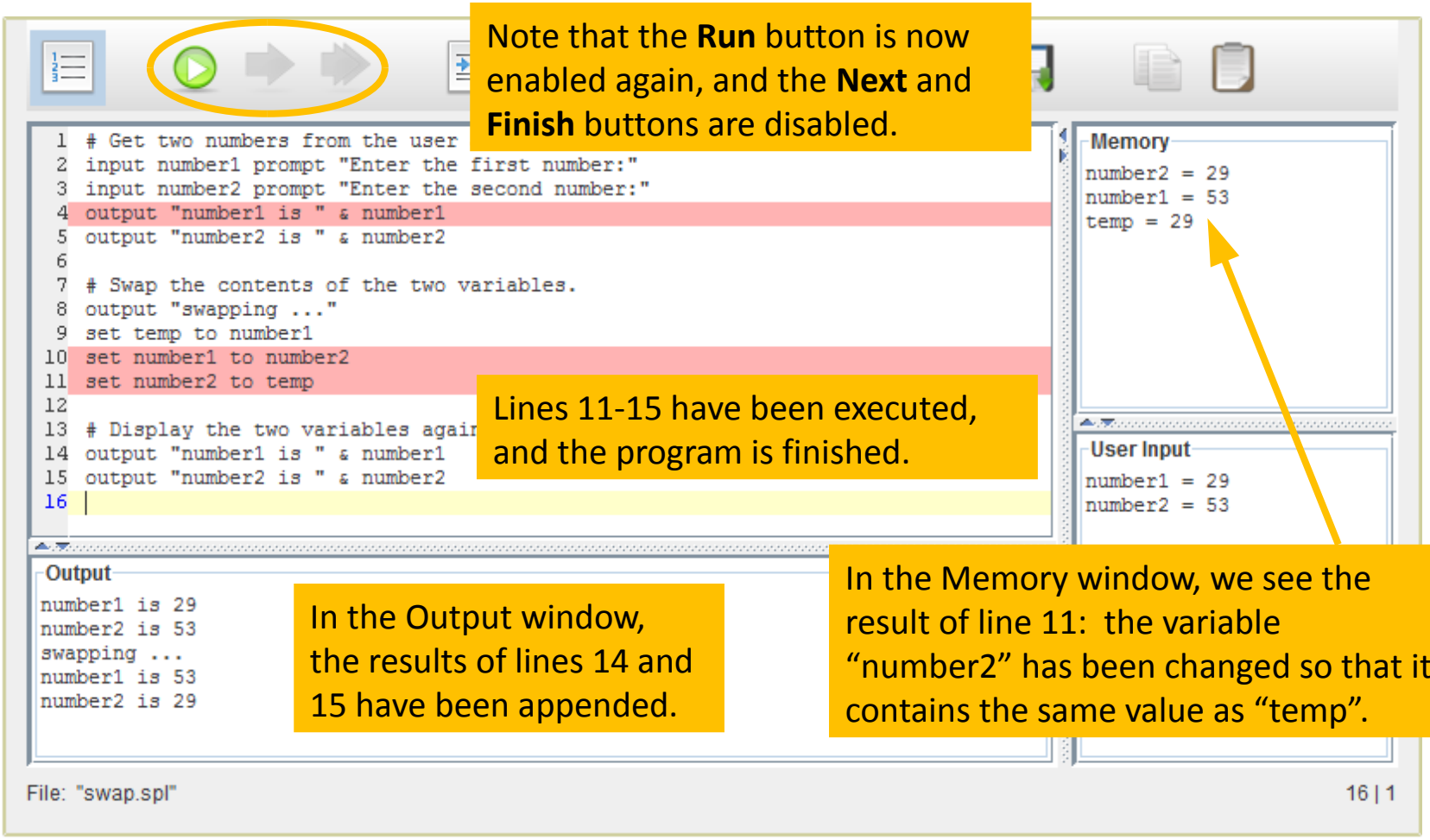
Line 10 has been executed, and the program has paused at line 11.

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.

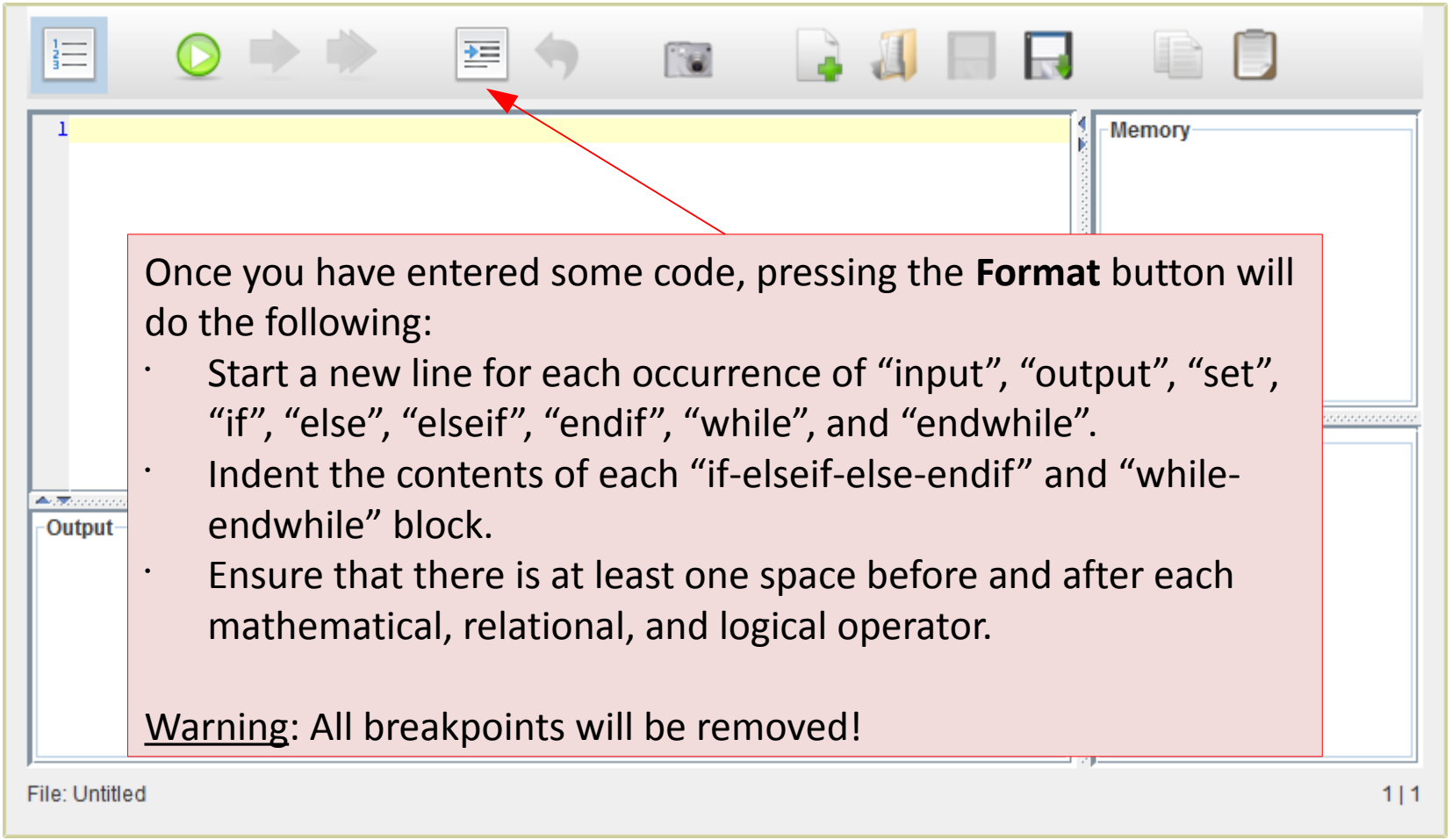
Continued ...

Debugging/Tracing a Program



Formatting Code

Formatting Code



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

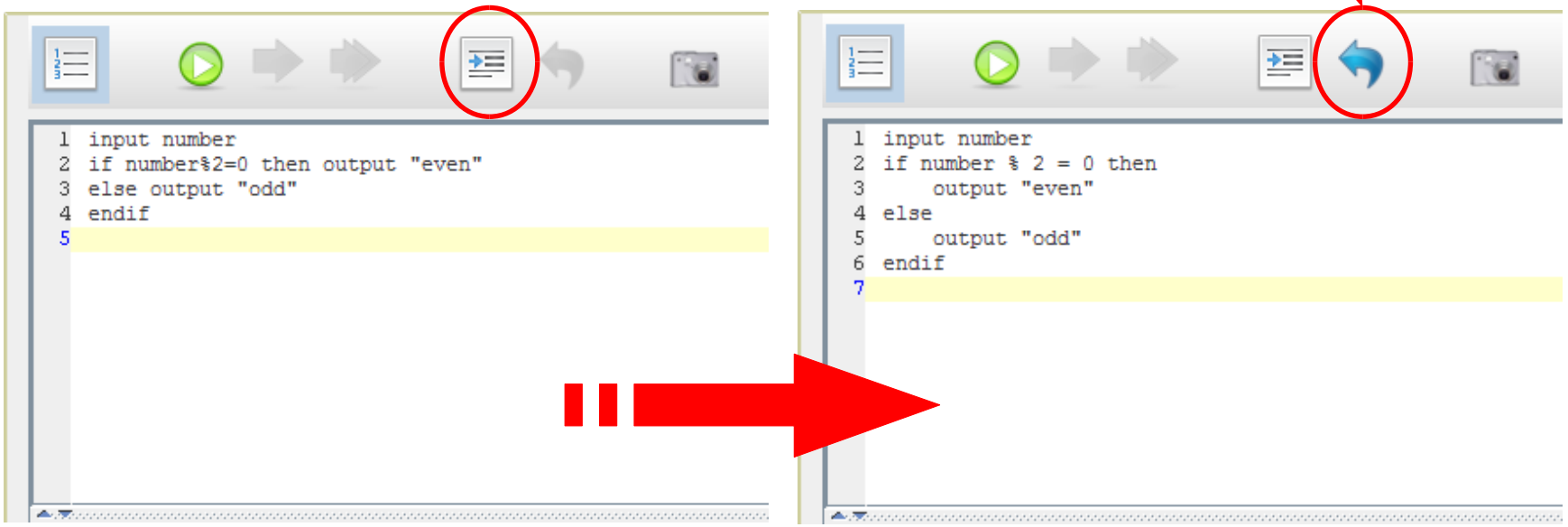
- Start a new line for each occurrence of “input”, “output”, “set”, “if”, “else”, “elseif”, “endif”, “while”, and “endwhile”.
- Indent the contents of each “if-elseif-else-endif” and “while-endwhile” block.
- Ensure that there is at least one space before and after each mathematical, relational, and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

Warning: All breakpoints will be removed!

see next slide for example ...

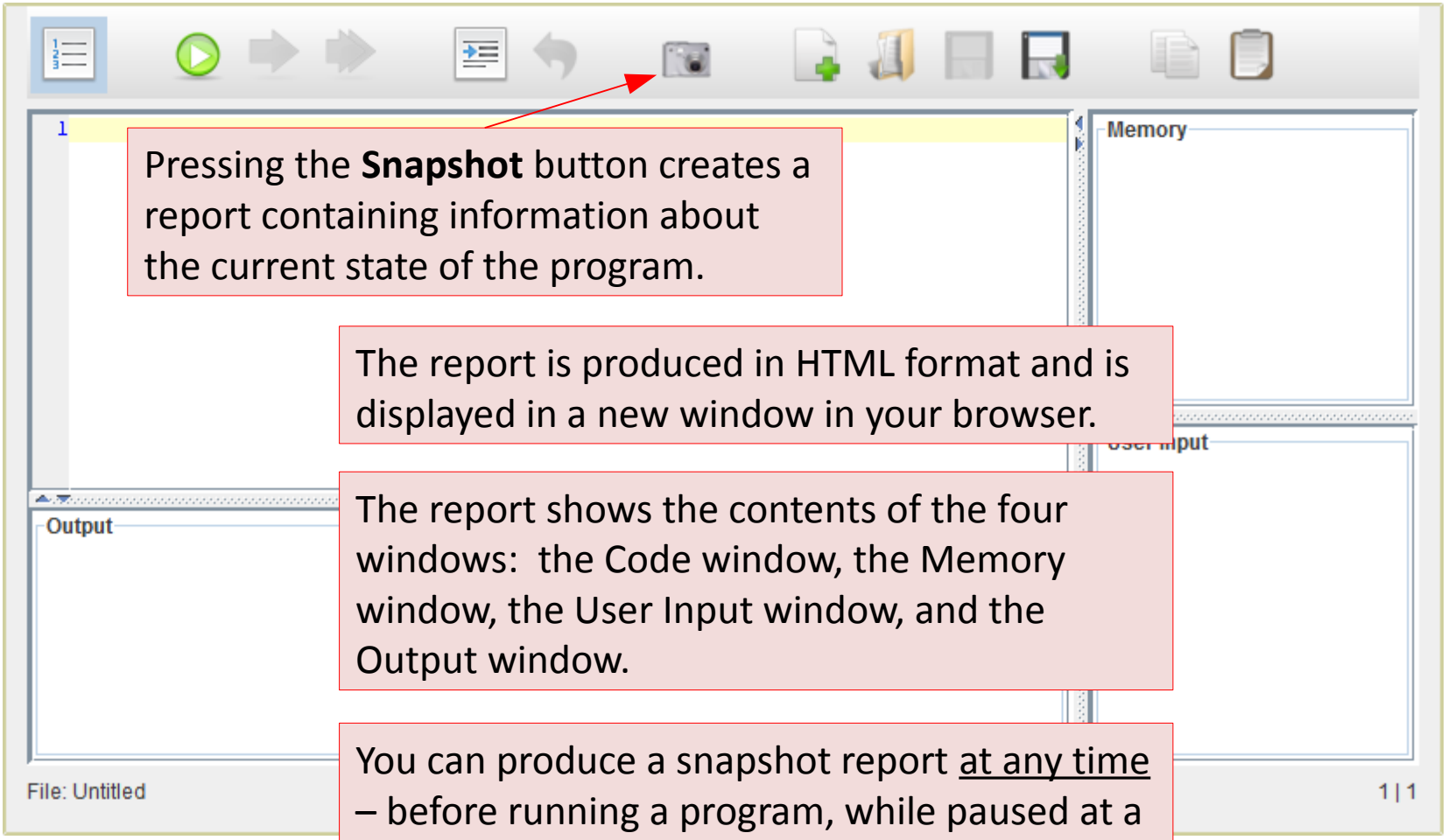
Formatting Code

Undo button is now enabled.



Snapshots

Snapshots



Pressing the **Snapshot** button creates a report containing information about the current state of the program.

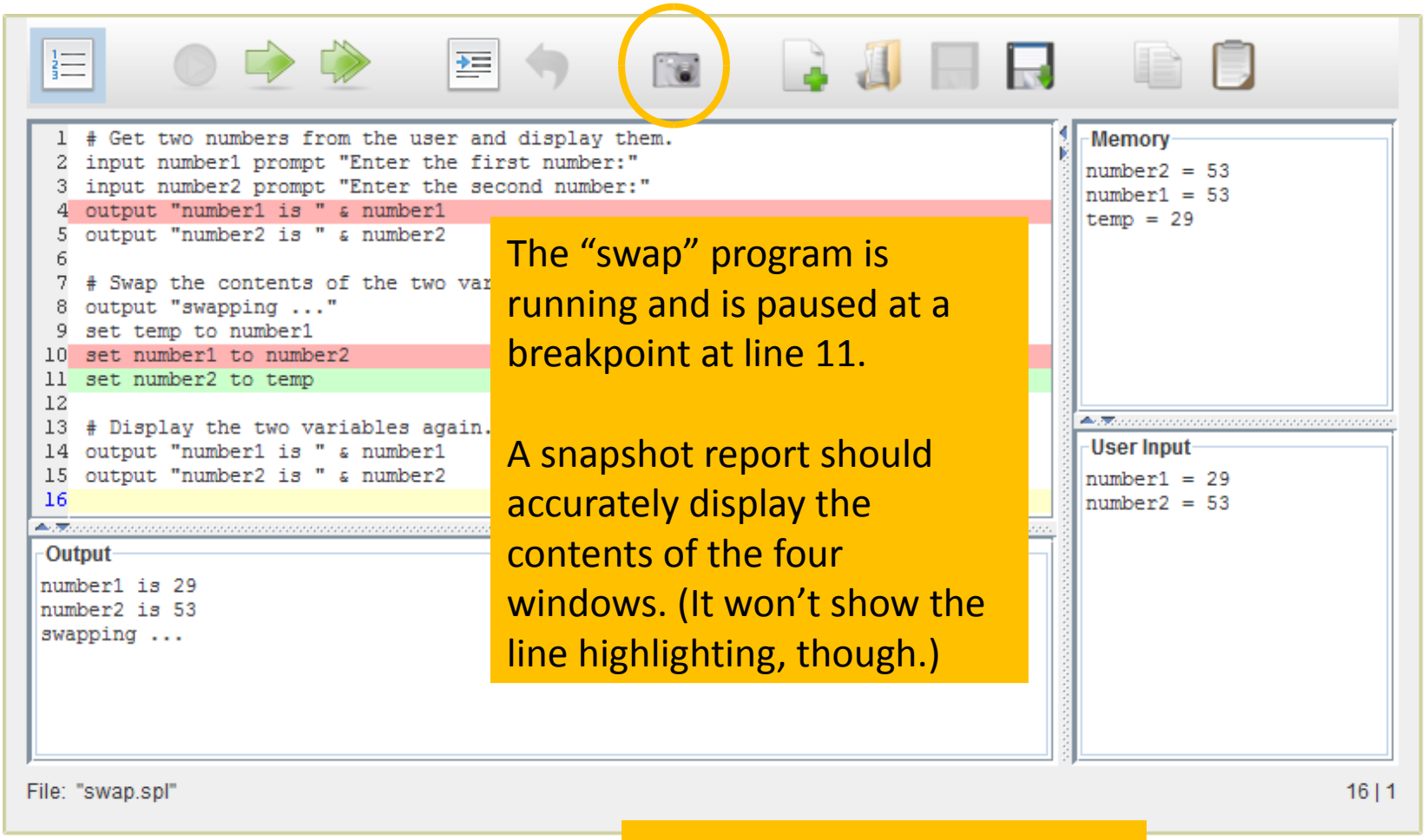
The report is produced in HTML format and is displayed in a new window in your browser.

The report shows the contents of the four windows: the Code window, the Memory window, the User Input window, and the Output window.

You can produce a snapshot report <u>at any time</u> – before running a program, while paused at a breakpoint, or after the program has finished.

Example ...

Snapshots



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

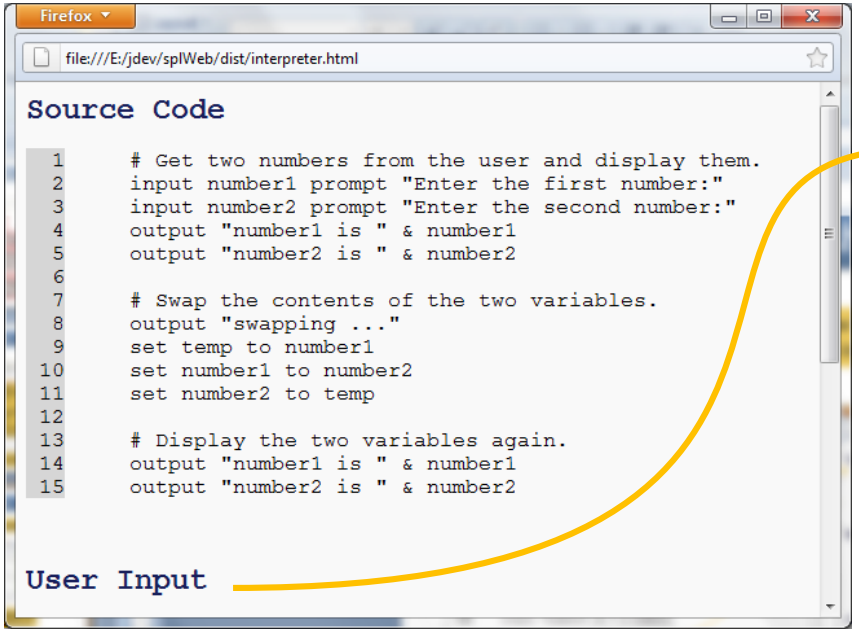
A snapshot report should accurately display the contents of the four windows. (It won't show the line highlighting, though.)

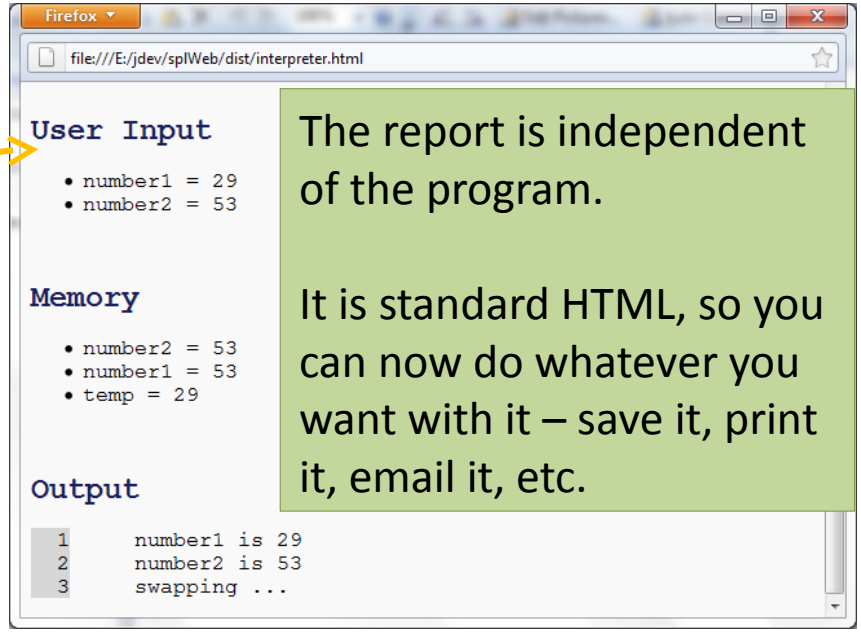
Let's press the **Snapshot** button to see the result ...

Continued ...

Snapshots

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





The report is independent of the program.

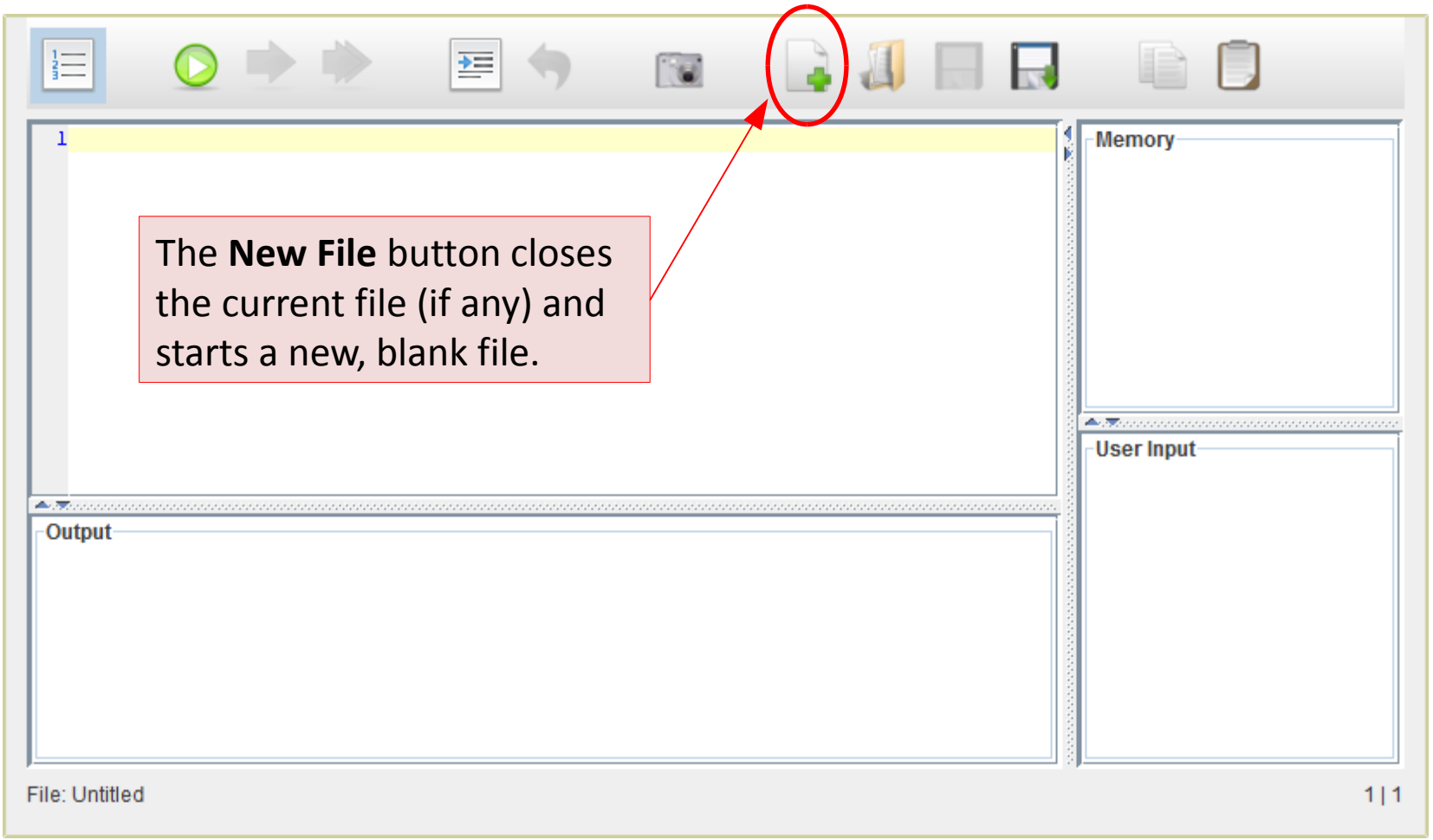
It is standard HTML, so you can now do whatever you want with it – save it, print it, email it, etc.

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

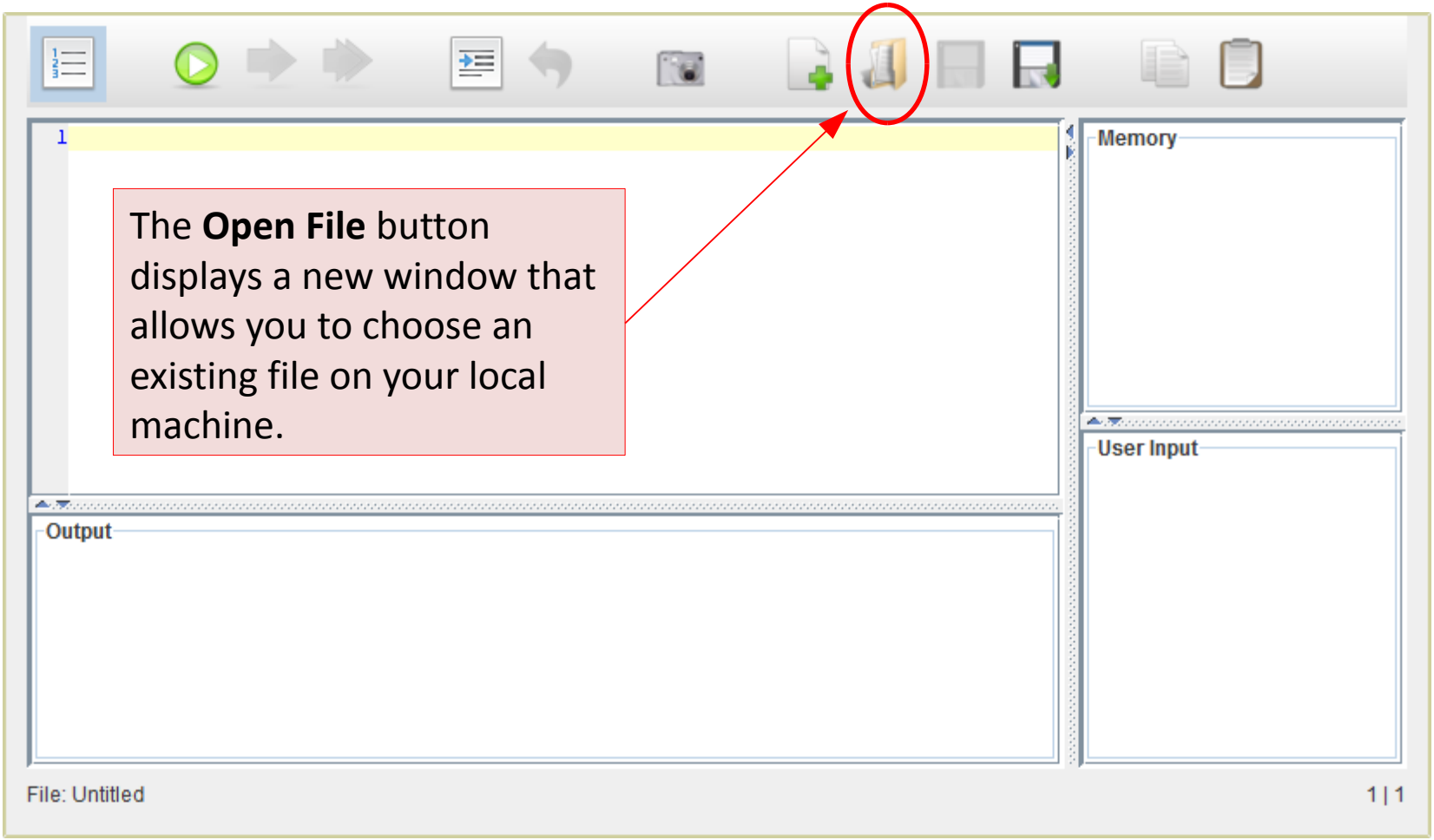
(Skip this section if these features are not available in your environment.)

File Operations



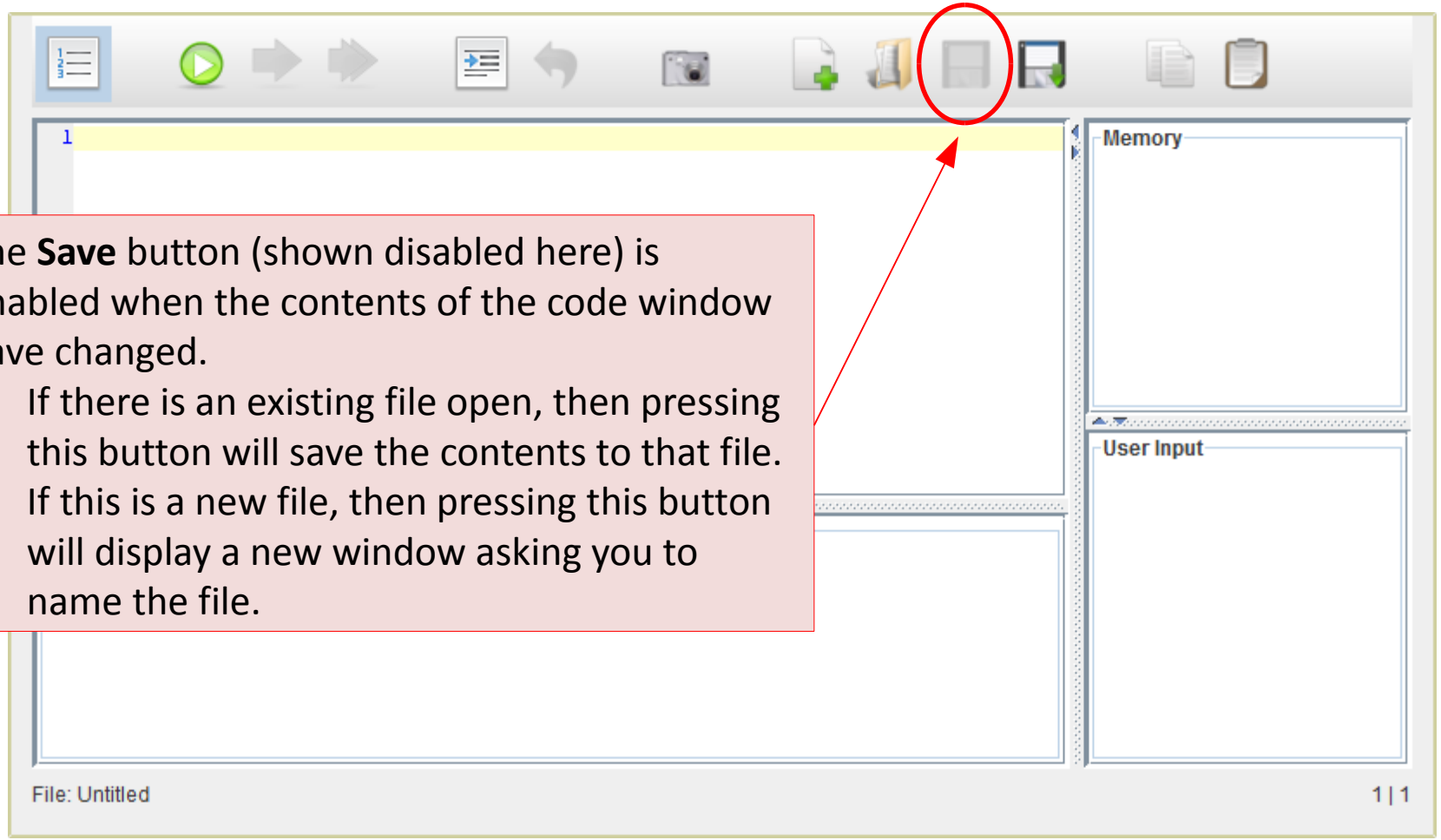
The **New File** button closes the current file (if any) and starts a new, blank file.

File Operations



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

File Operations

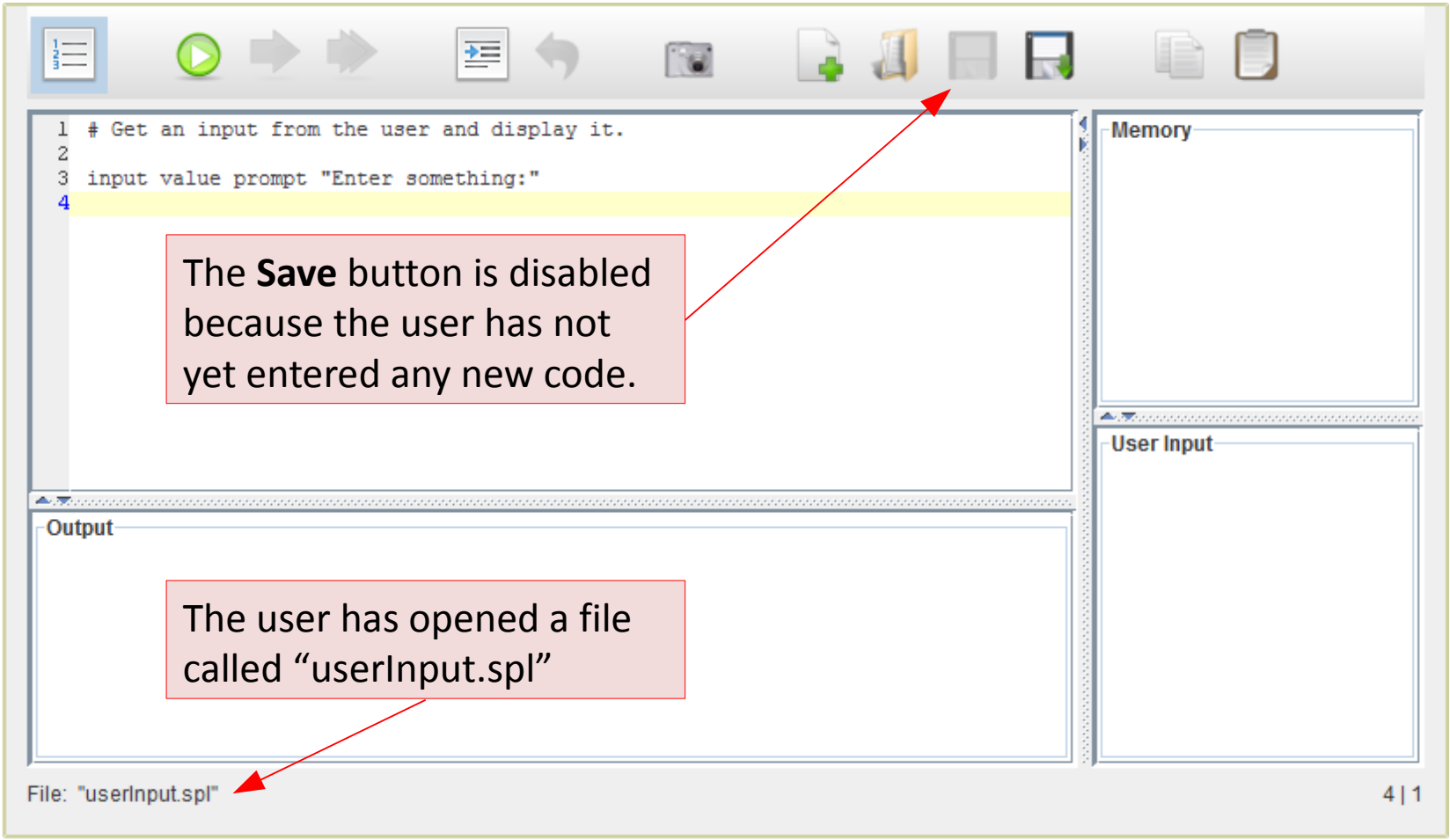


The **Save** button (shown disabled here) is enabled when the contents of the code window have changed.

- If there is an existing file open, then pressing this button will save the contents to that file.
- If this is a new file, then pressing this button will display a new window asking you to name the file.

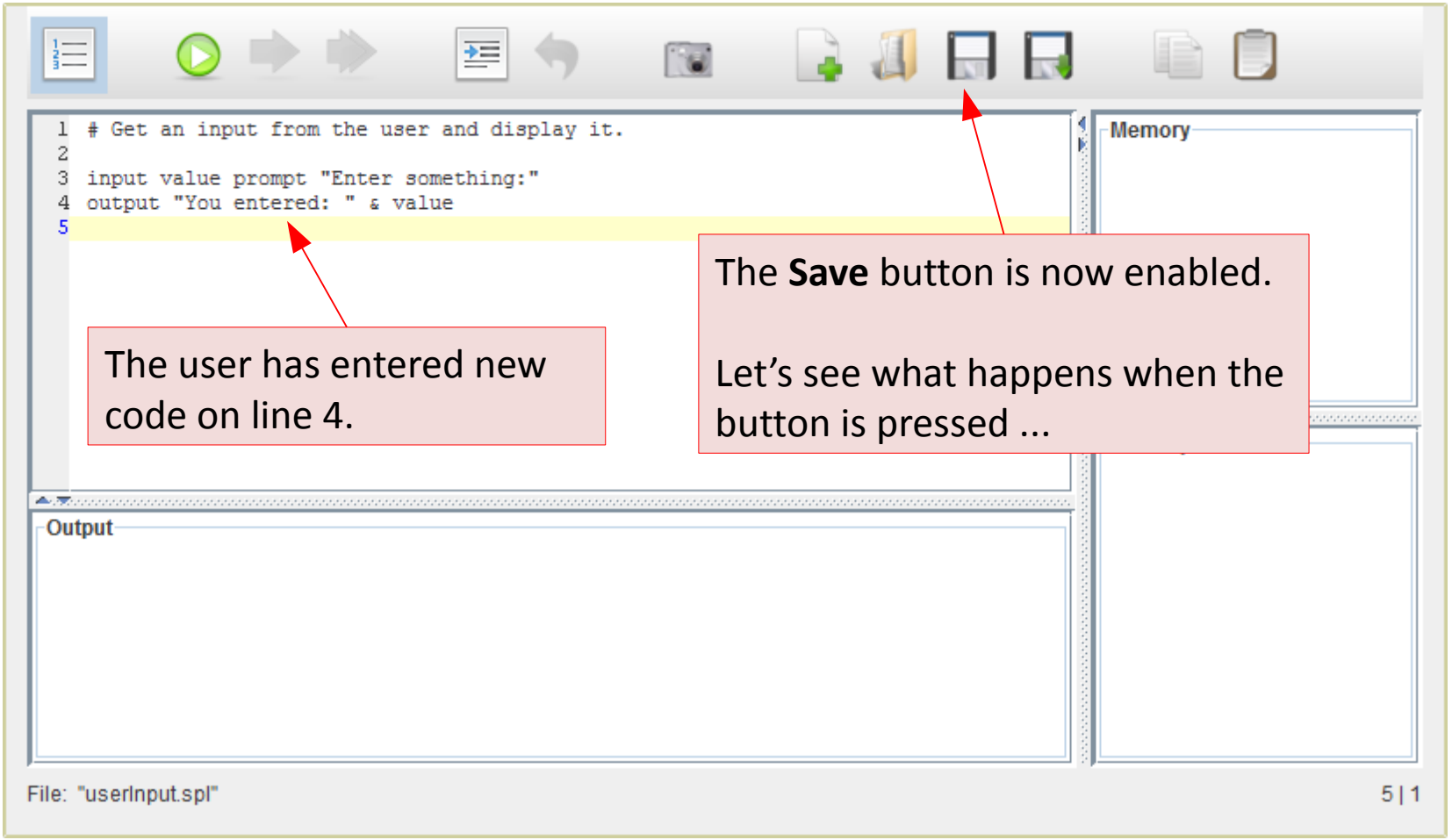
See next slide for example ...

File Operations – Save Example



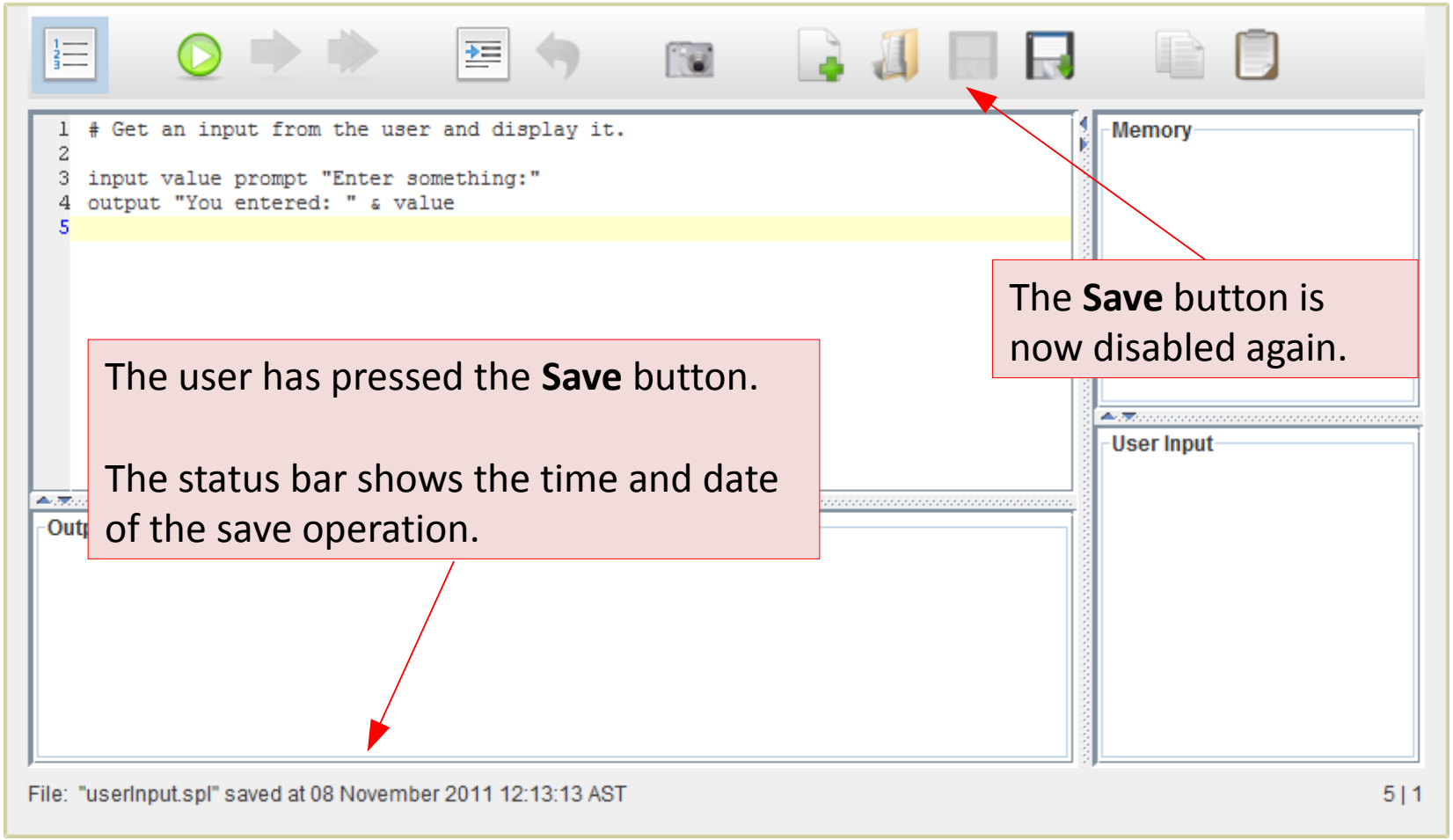
Continued ...

File Operations – Save Example

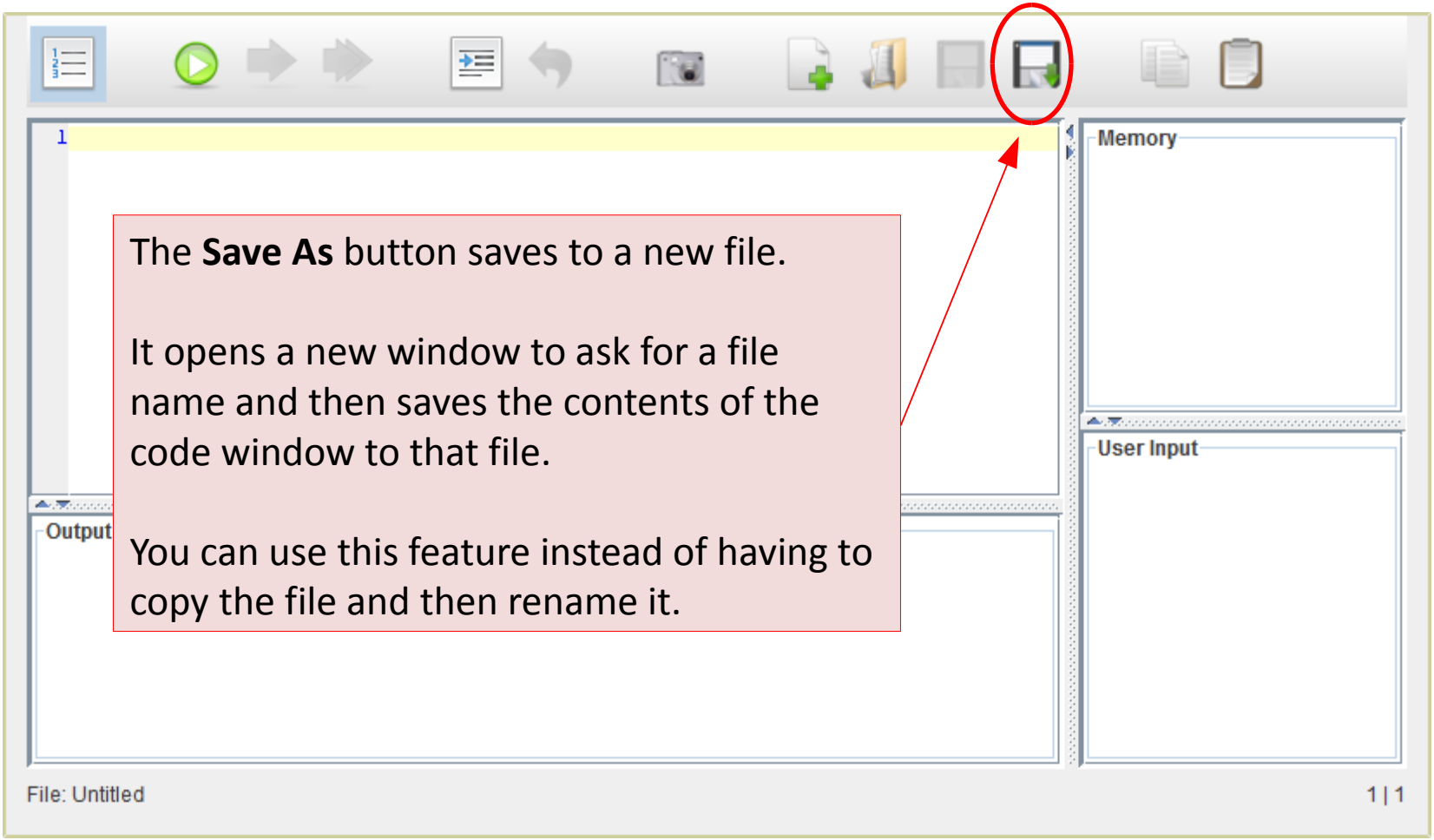


Continued ...

File Operations – Save Example



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.

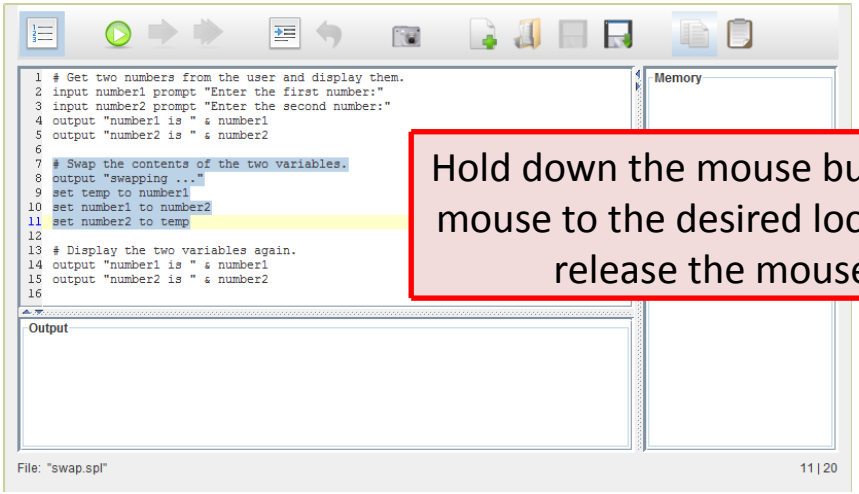
Transferring Text

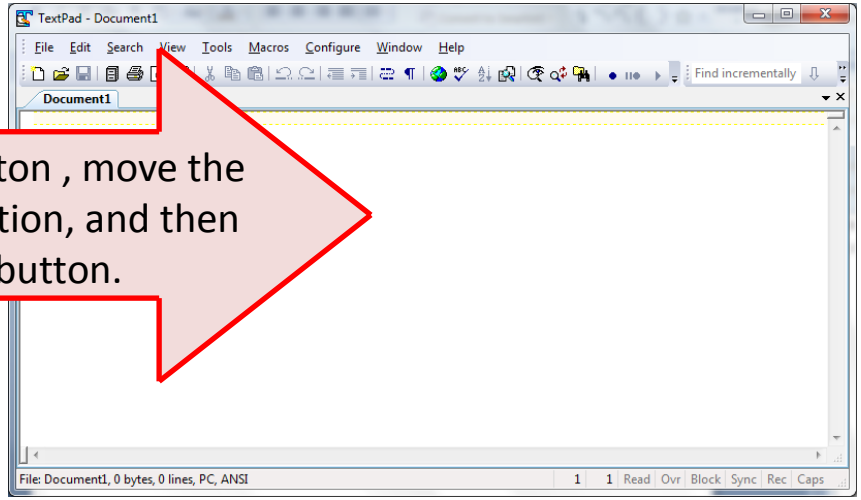
Drag-and-Drop

- 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



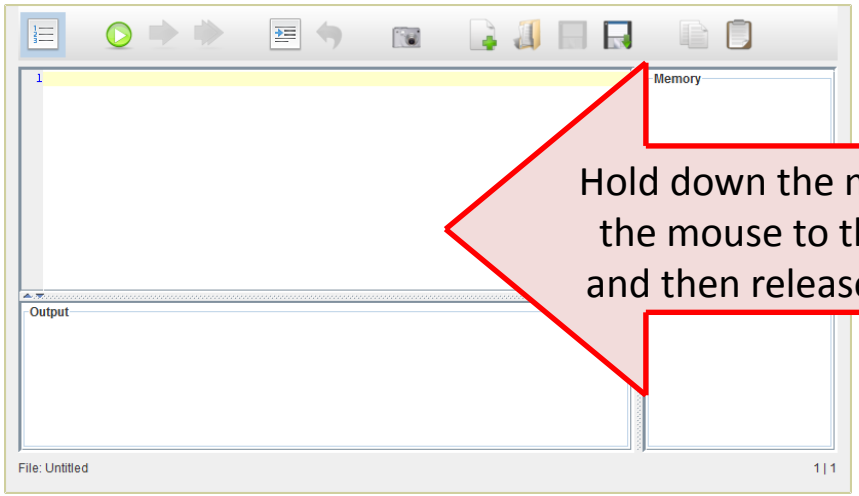


Hold down the mouse button , move the mouse to the desired location, and then release the mouse button.

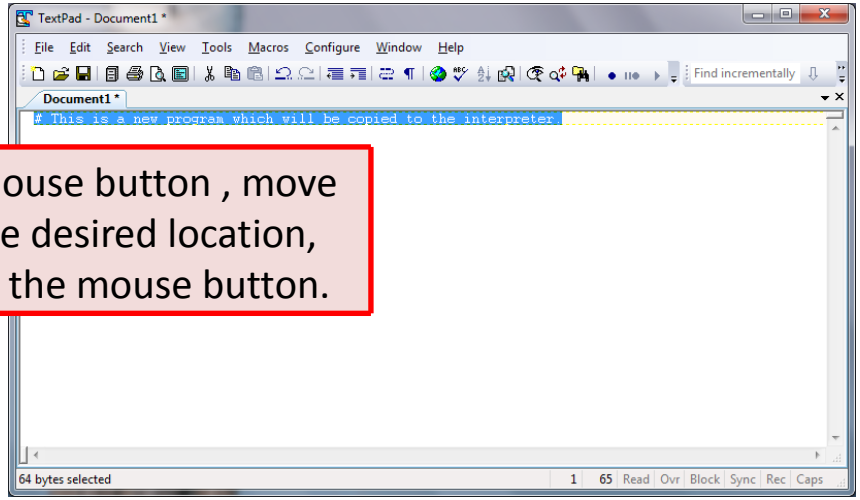
To drag-and-drop text from the interpreter to a native application, select the text you want to copy, and then:

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

Drag-and-Drop 2



Hold down the mouse button , move the mouse to the desired location, and then release the mouse button.



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

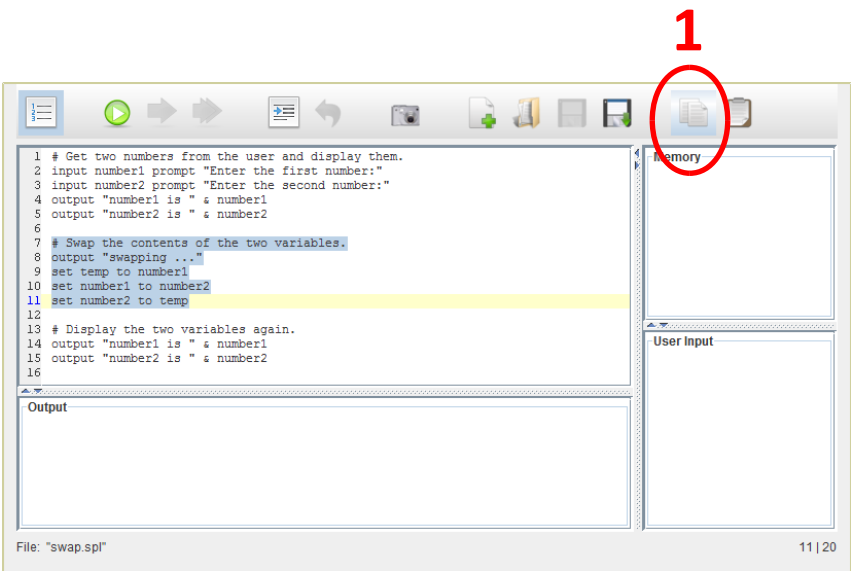
1. Drag it to the interpreter's code window; and
2. Drop the text in the desired location.

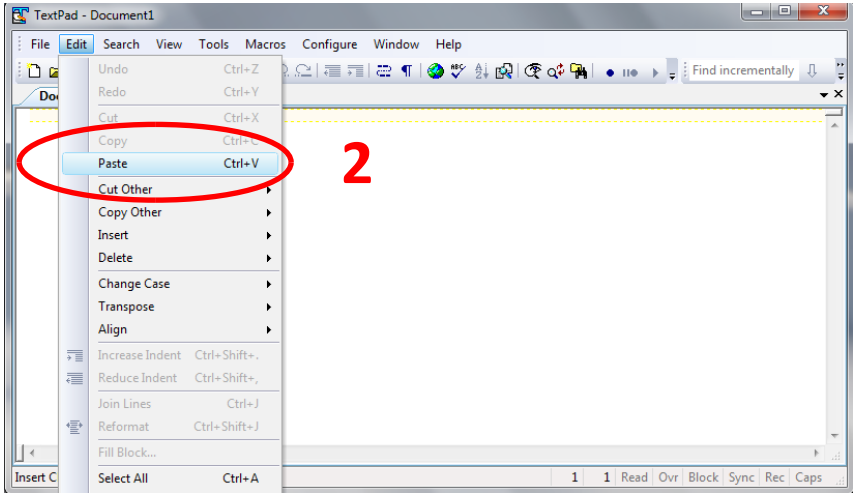
Using the Copy and Paste Buttons

- 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.

Examples ...

Using the Copy Button



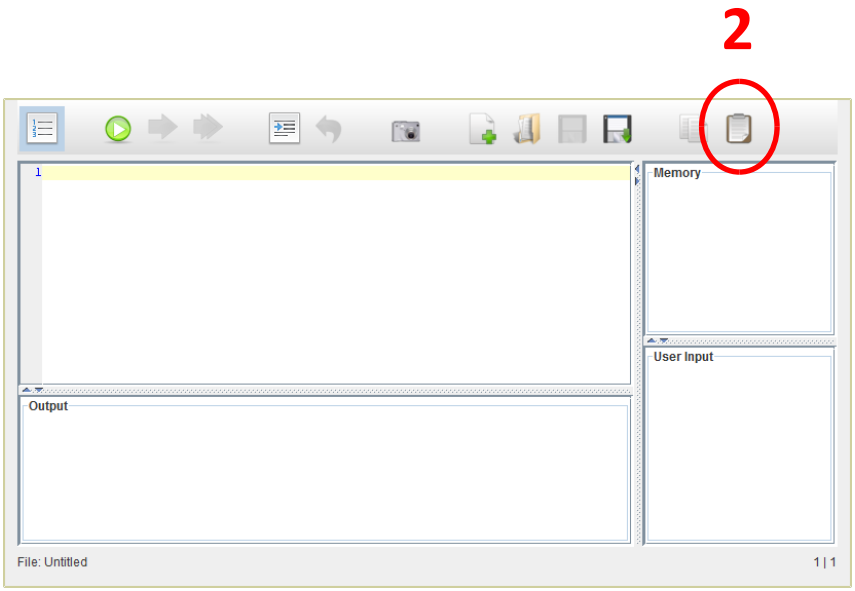


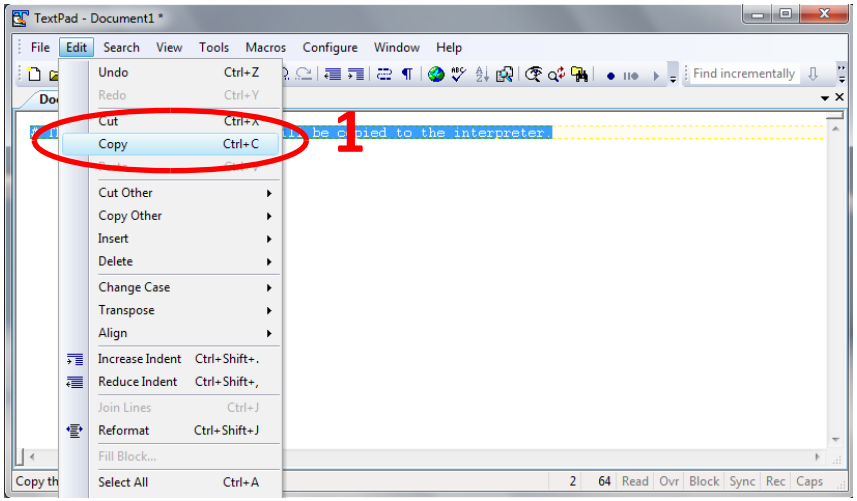
To copy text from the interpreter to a native application, select the text you want to copy and then:

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





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