Variable names – Which are allowed?

How do we check if a variable name is a keyword or built-in function?

How to check if a variable name is a keyword?

```
In [1]:
1 import keyword
2 'variable_name' in keyword.kwlist
```

```
Out[1]: False
```

```
In [2]:
1 'def' in keyword.kwlist
```

```
Out[2]: True
```

How to list all keywords?

```
In [5]:
1 print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async
', 'await', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for', 'from', 'gl
obal', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal
', 'not', 'or', 'pass', 'raise', 'return', 'try', 'whil
e', 'with', 'yield']
```

How to check if a variable name is a built-in function?

```
In [9]:
1 import builtins
2 'variable_name' in dir(builtins)
```

```
Out[9]: False
```

```
In [10]:
1 'max' in dir(builtins)
```

```
Out[10]: True
```

How to list all built-in functions?

```
In [6]:
1 print(dir(builtins))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'Buf
ferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedErro
r', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportE
rror', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'Key
boardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'N
otImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError',
'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'Sto
pAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'Timeou
tError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'Uni
codeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError',
'__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '_
_spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'clas
smethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate',
'eval', 'exec', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', '
help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map',
'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'r
eversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'va
rs', 'zip']
```

(Note: You do not have to memorize this list.)

IPython Debugger Variables

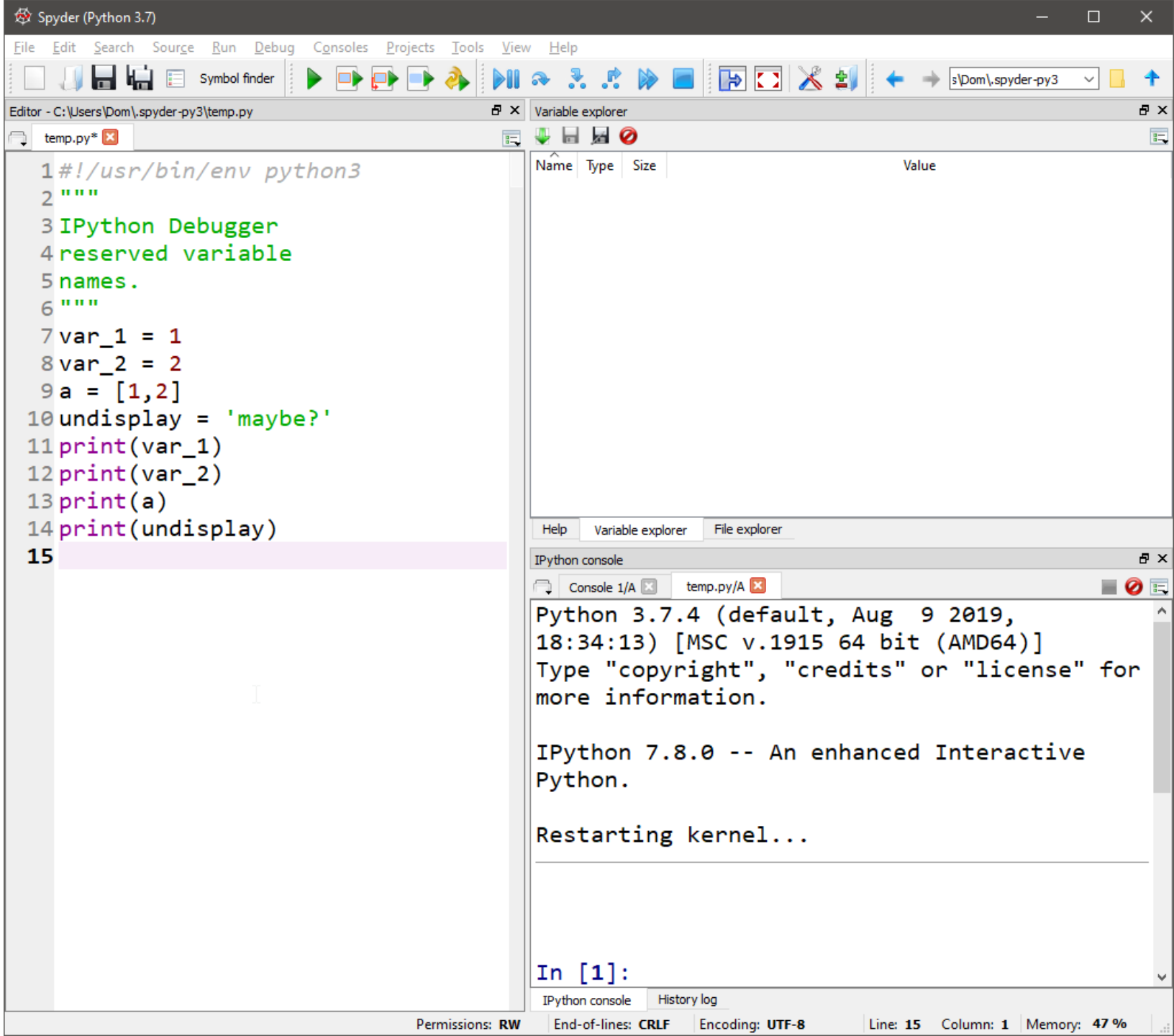
These are a few more variables which have a special meaning by default while you are in the debug mode (not otherwise).

If you use the debug mode frequently, please go through this guide once and be aware of the IPython Debugger Variables.

An Example follows hereafter.

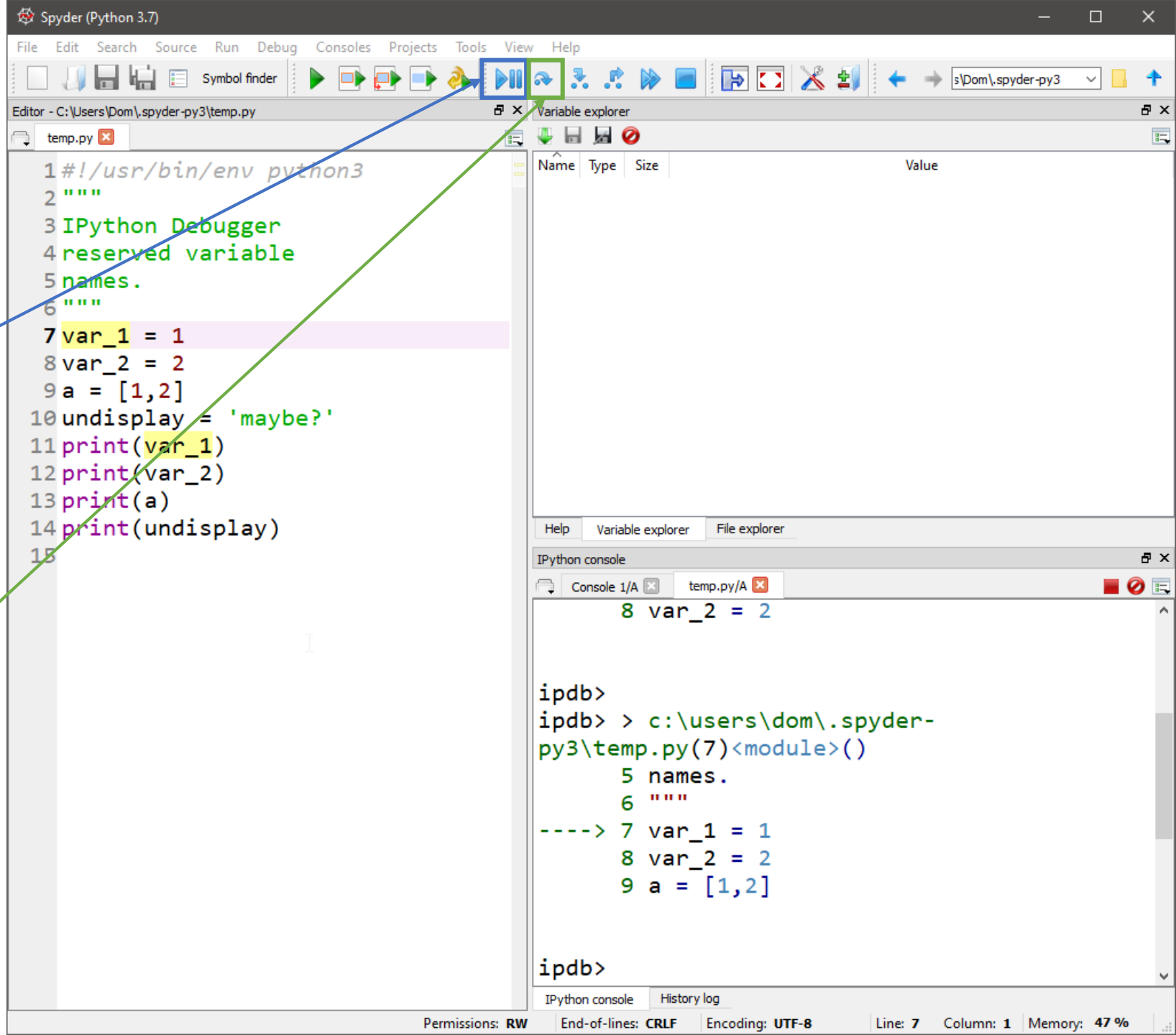
Summary: You can use any of the variable names of the IPython Debugger Variables as variable names for your own variables. You just have to be careful to not change their value by typing into the console while you are in the debug mode.

Let's work with this code:

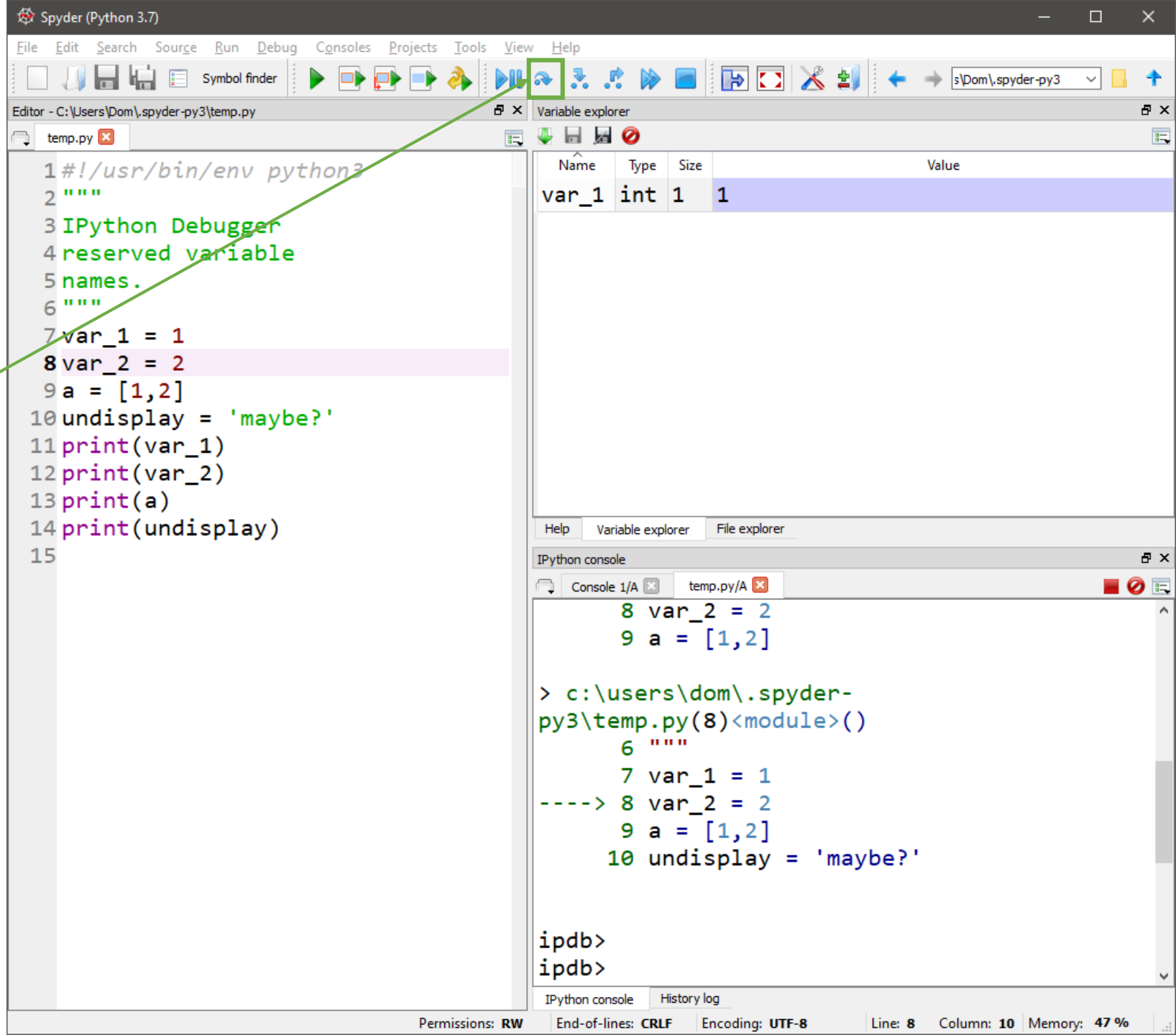


Enter the debug mode.

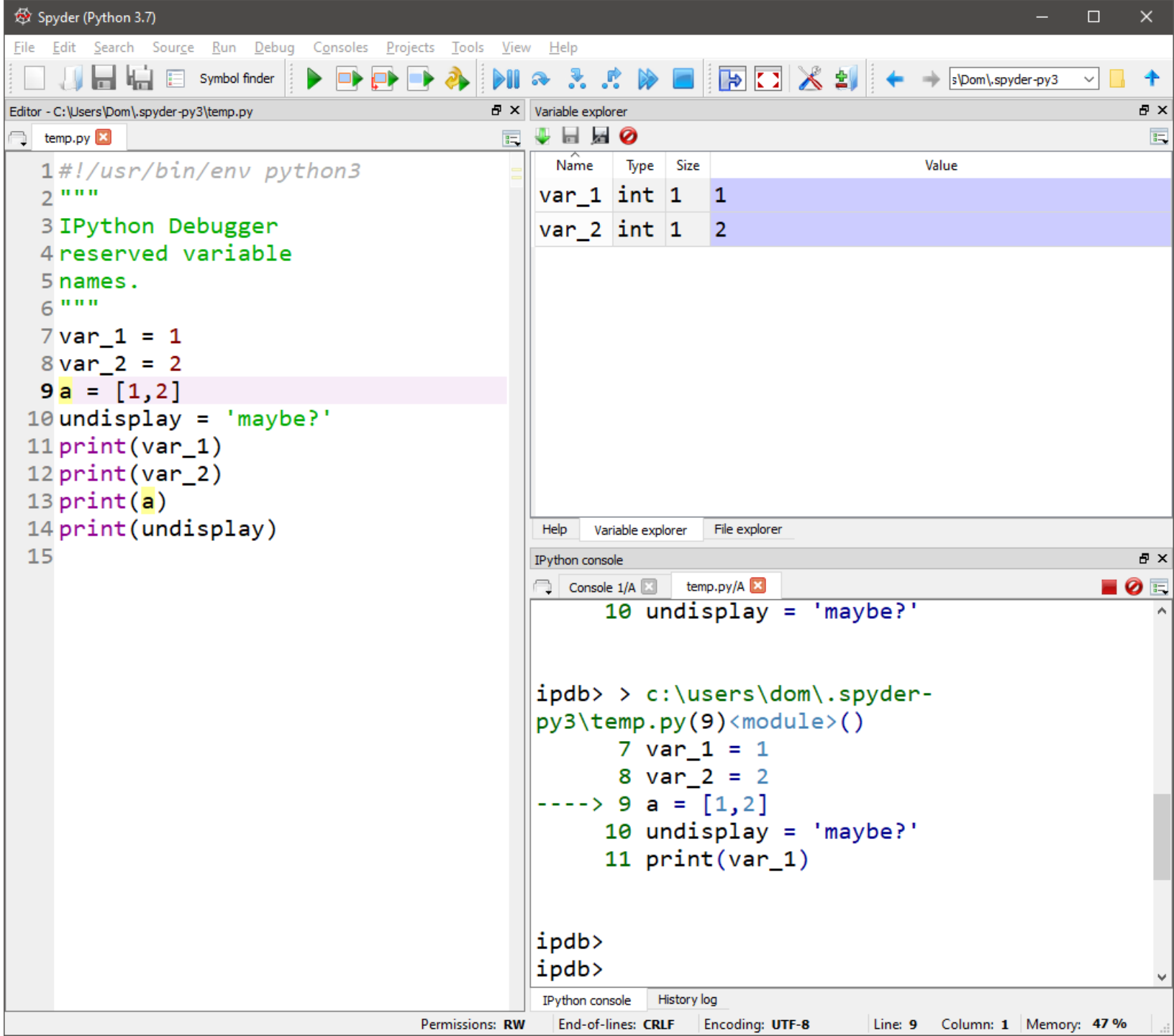
Go to Line 7.



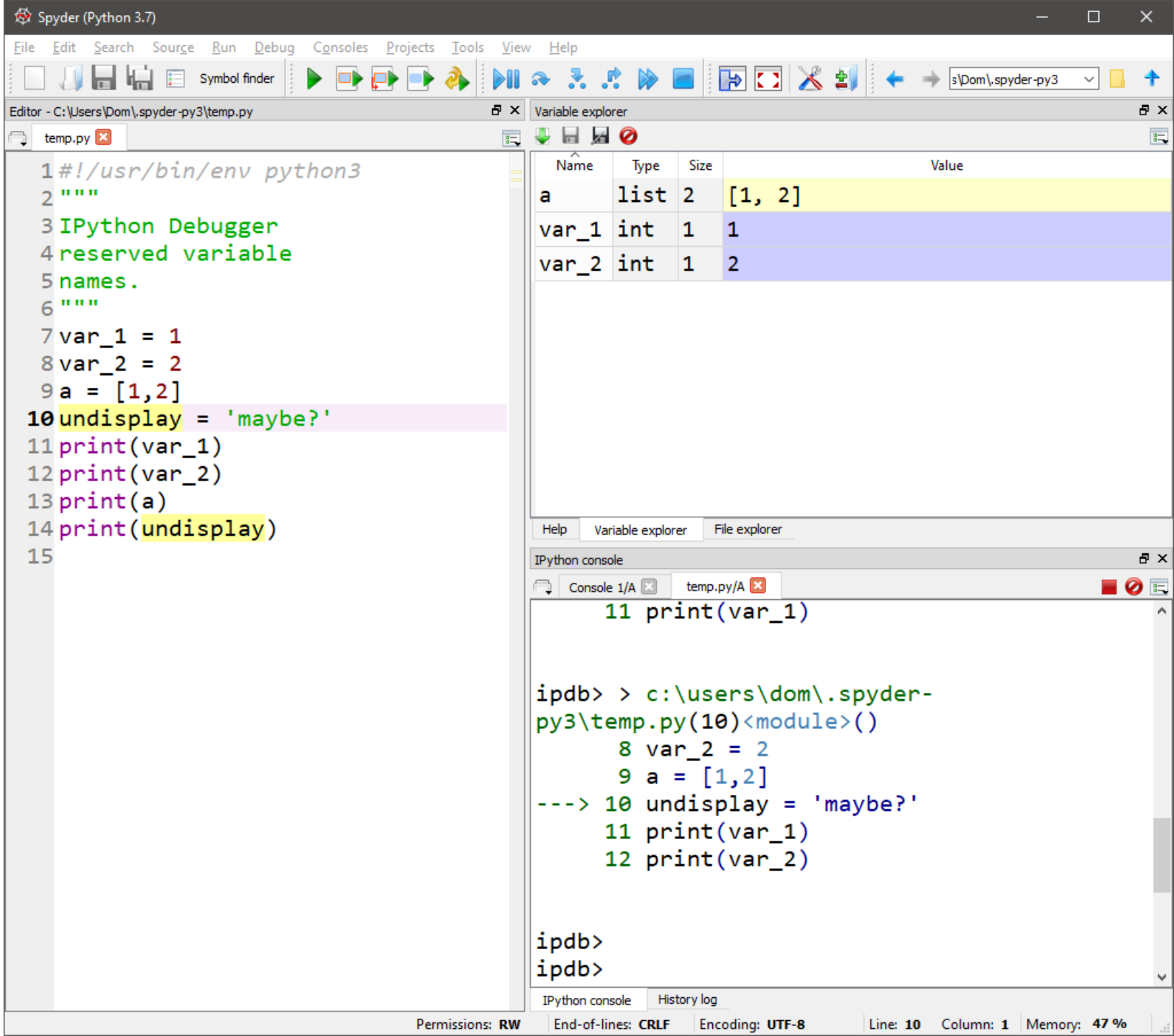
Execute Line 7.



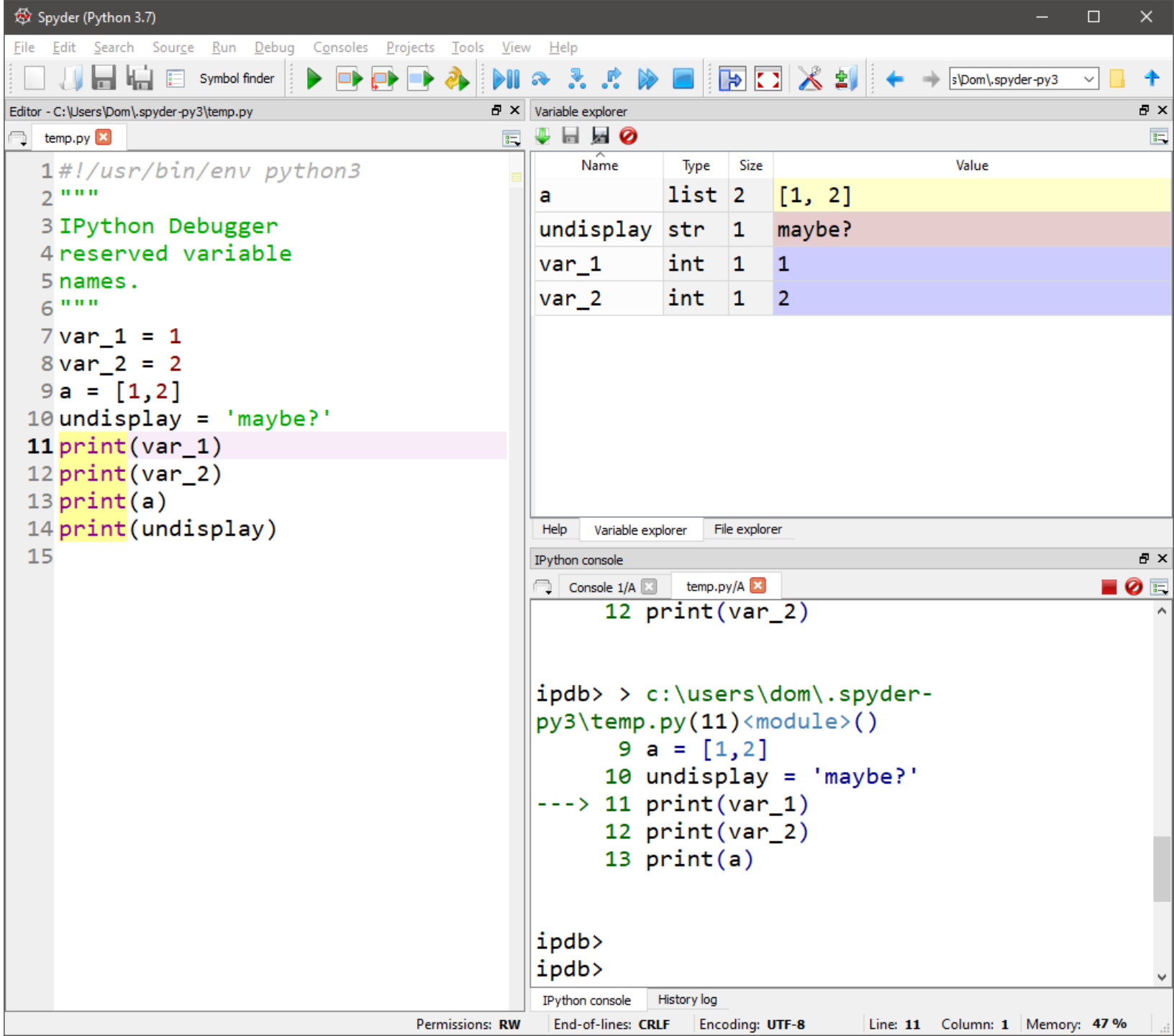
Execute Line 8.



Execute
Line 9.

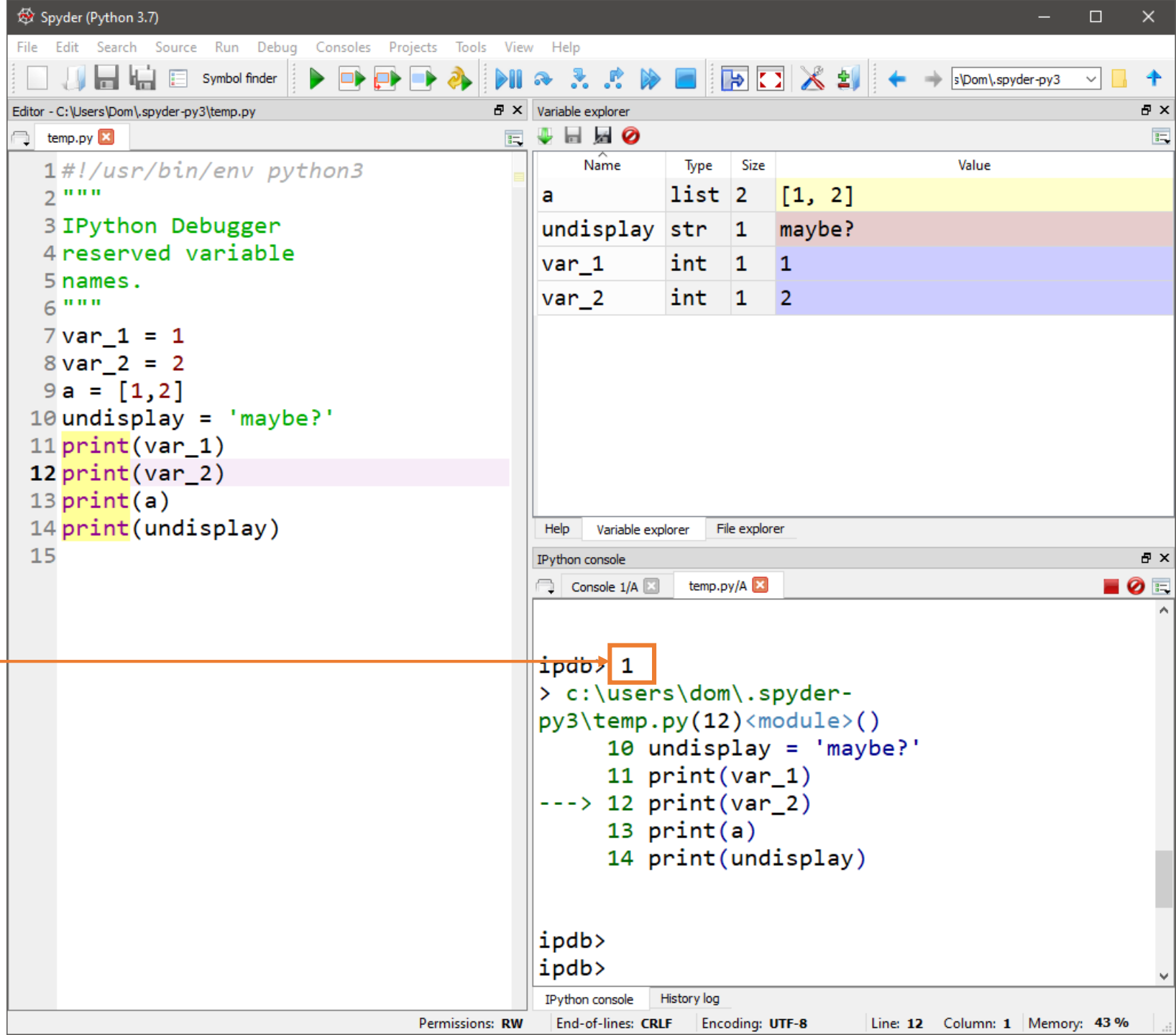


Execute
Line 10.

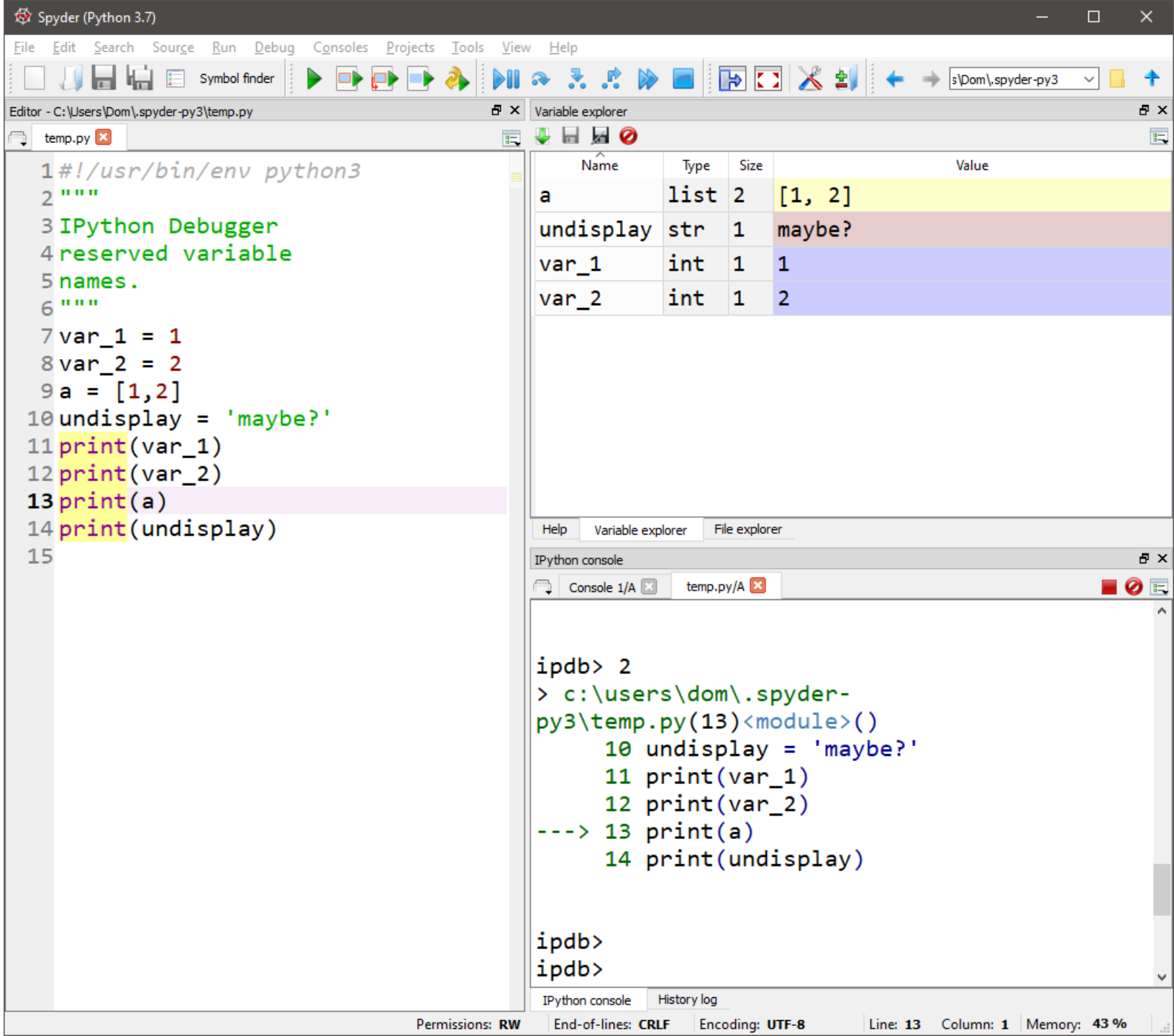


Execute Line 11.

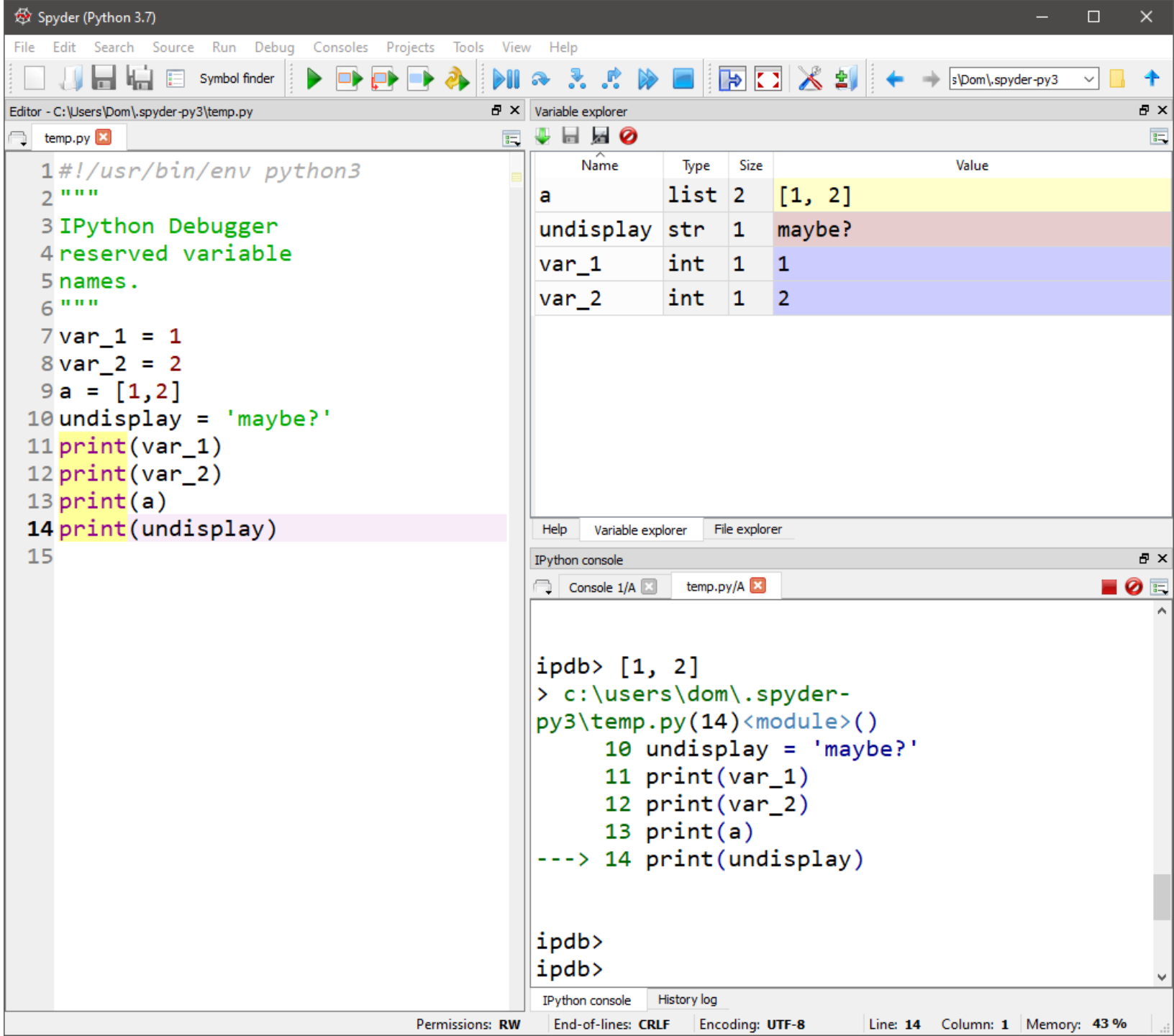
(Here is the printed output)



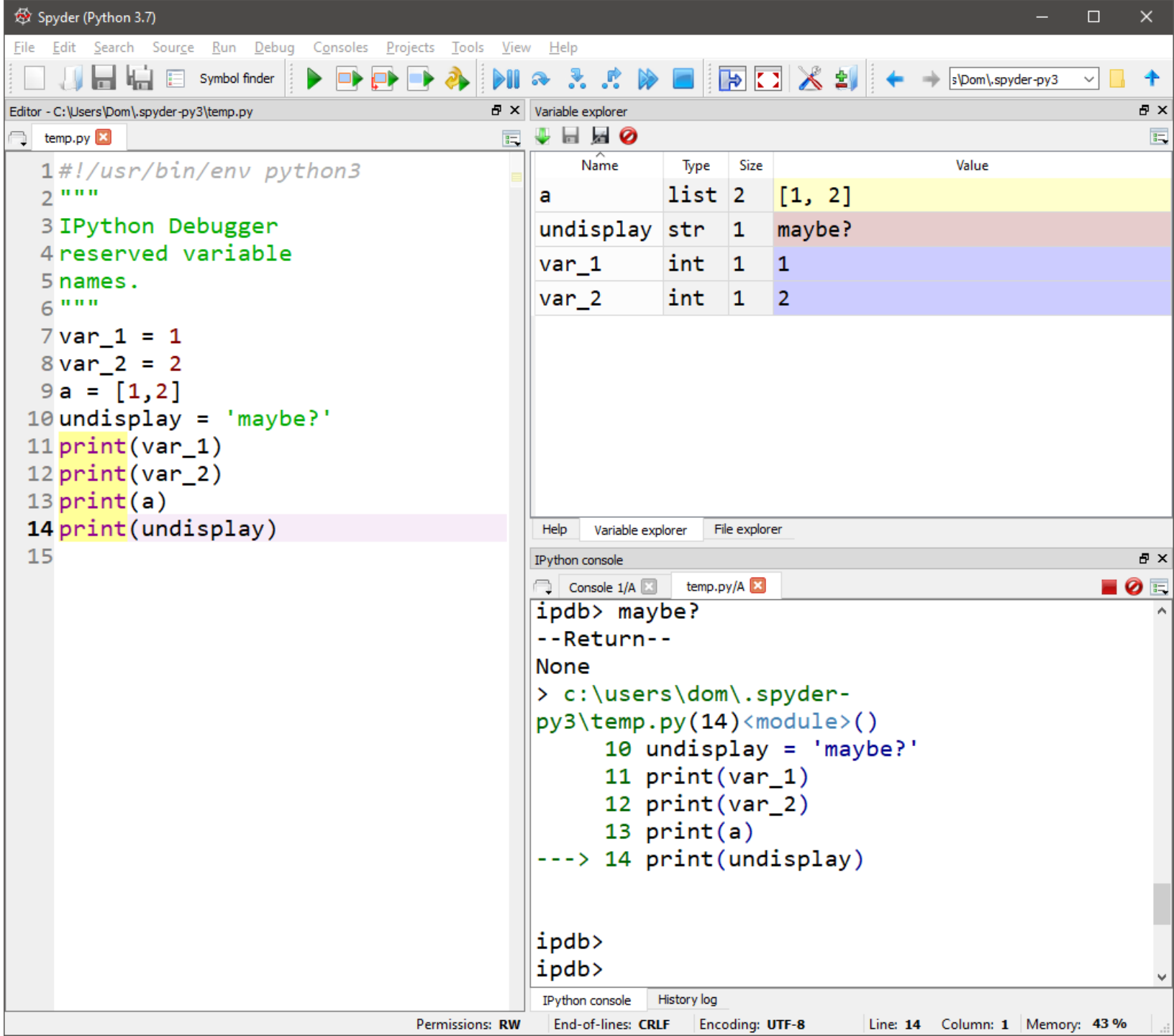
Execute
Line 12.



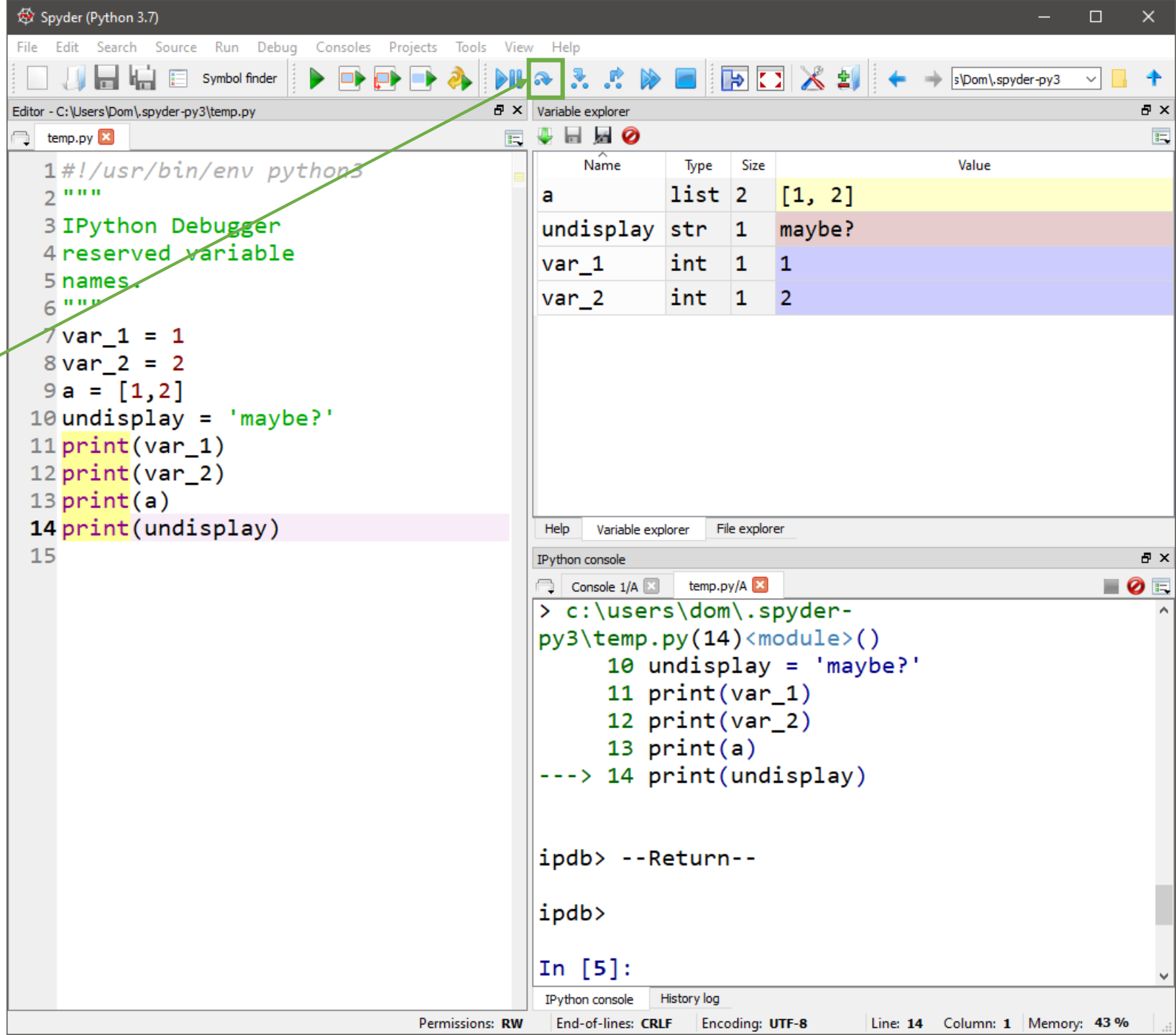
Execute
Line 13.



Execute
Line 14.



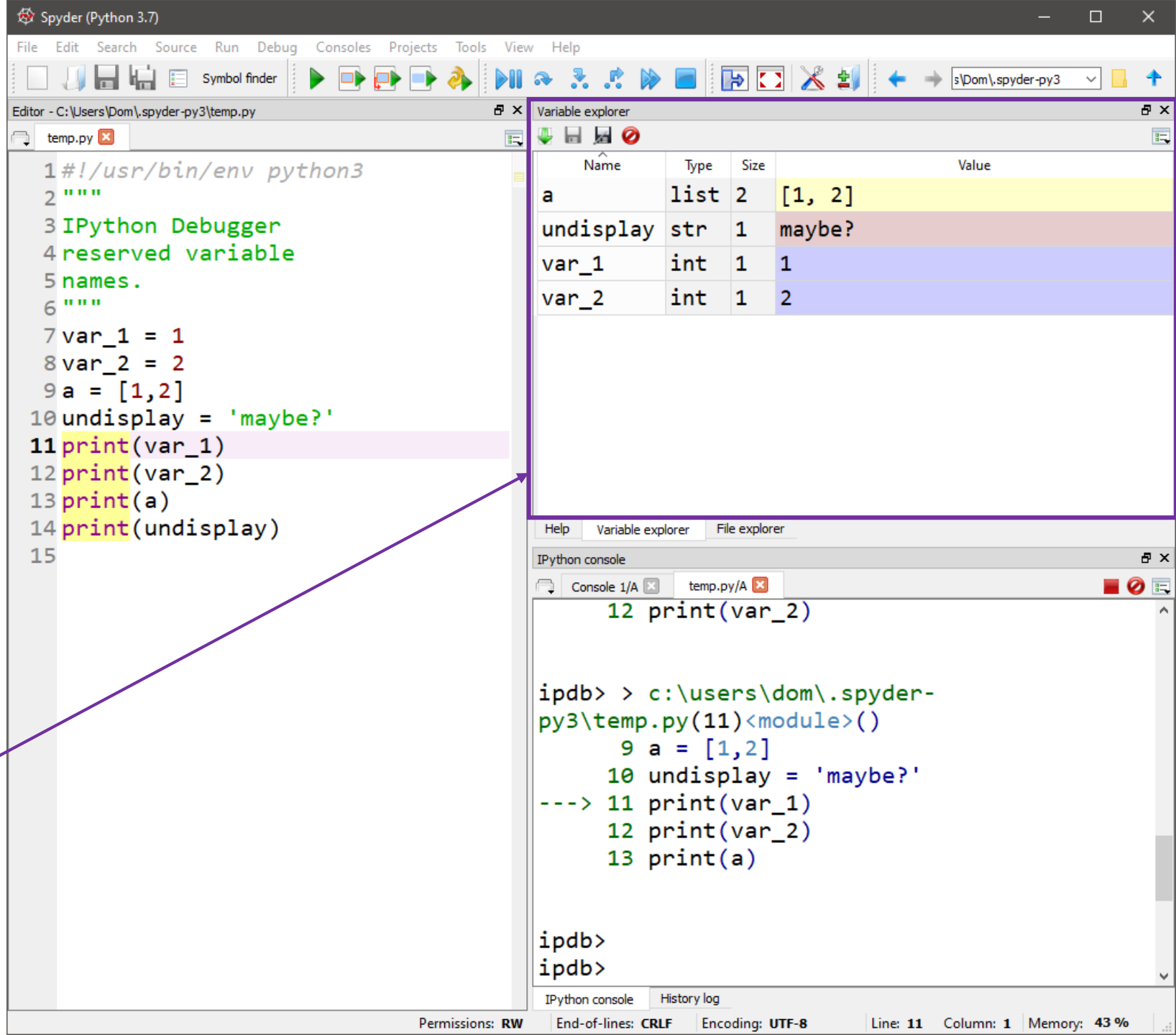
And, as usual, one more click on next line exits the debugger and brings us back to normal.



So far, everything is normal.

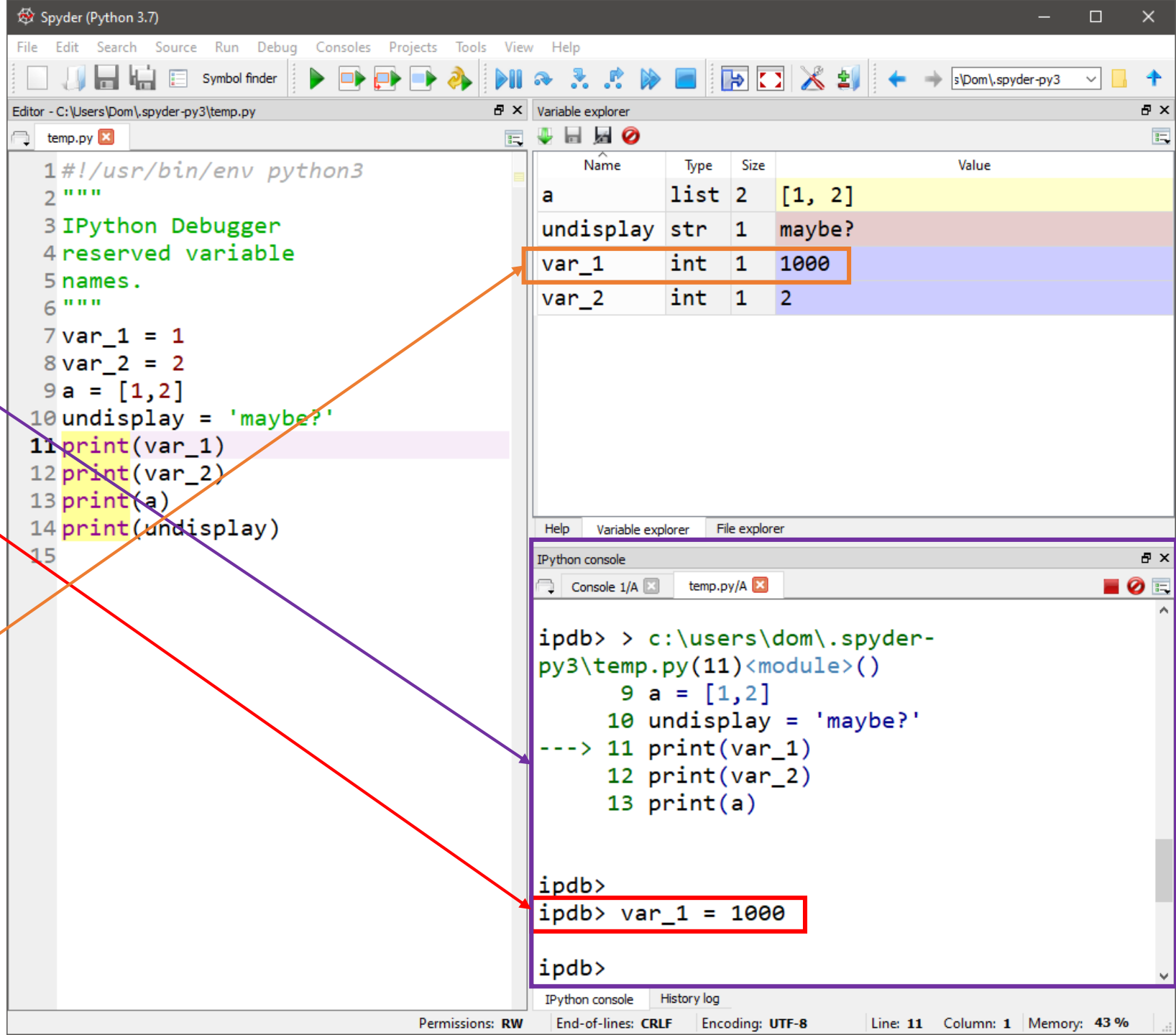
Now, let’s change some variable values in the debug mode by typing into the console while we execute the program.

Enter the debug mode and execute all lines until including line 10. You see the values of all variables in the Variable Explorer.



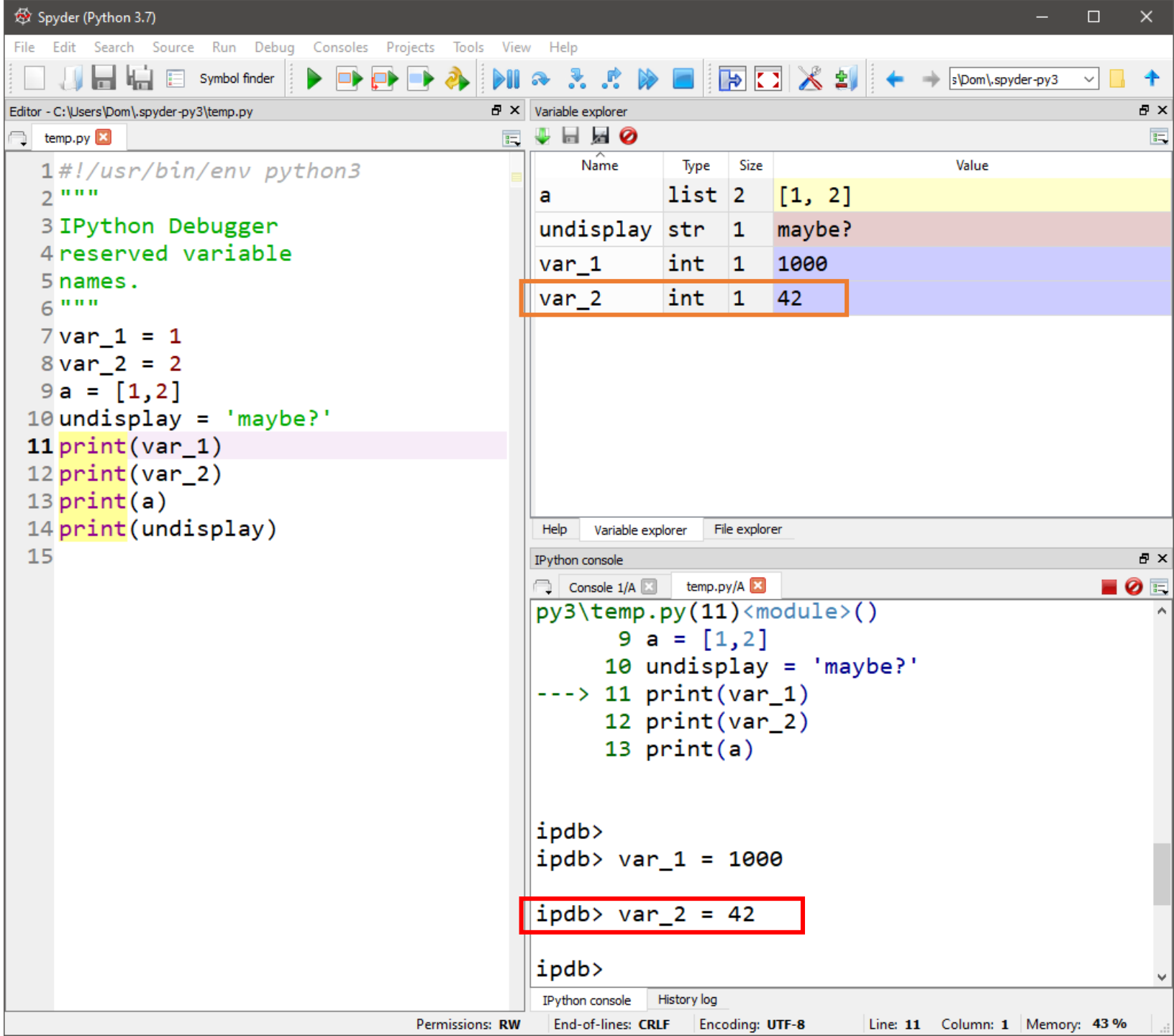
Type the following into the Console: “`var_1 = 1000`”.

You see, that the variable value changes to 1000 as expected. So far, so good.



Let's try the same with "`var_2`".

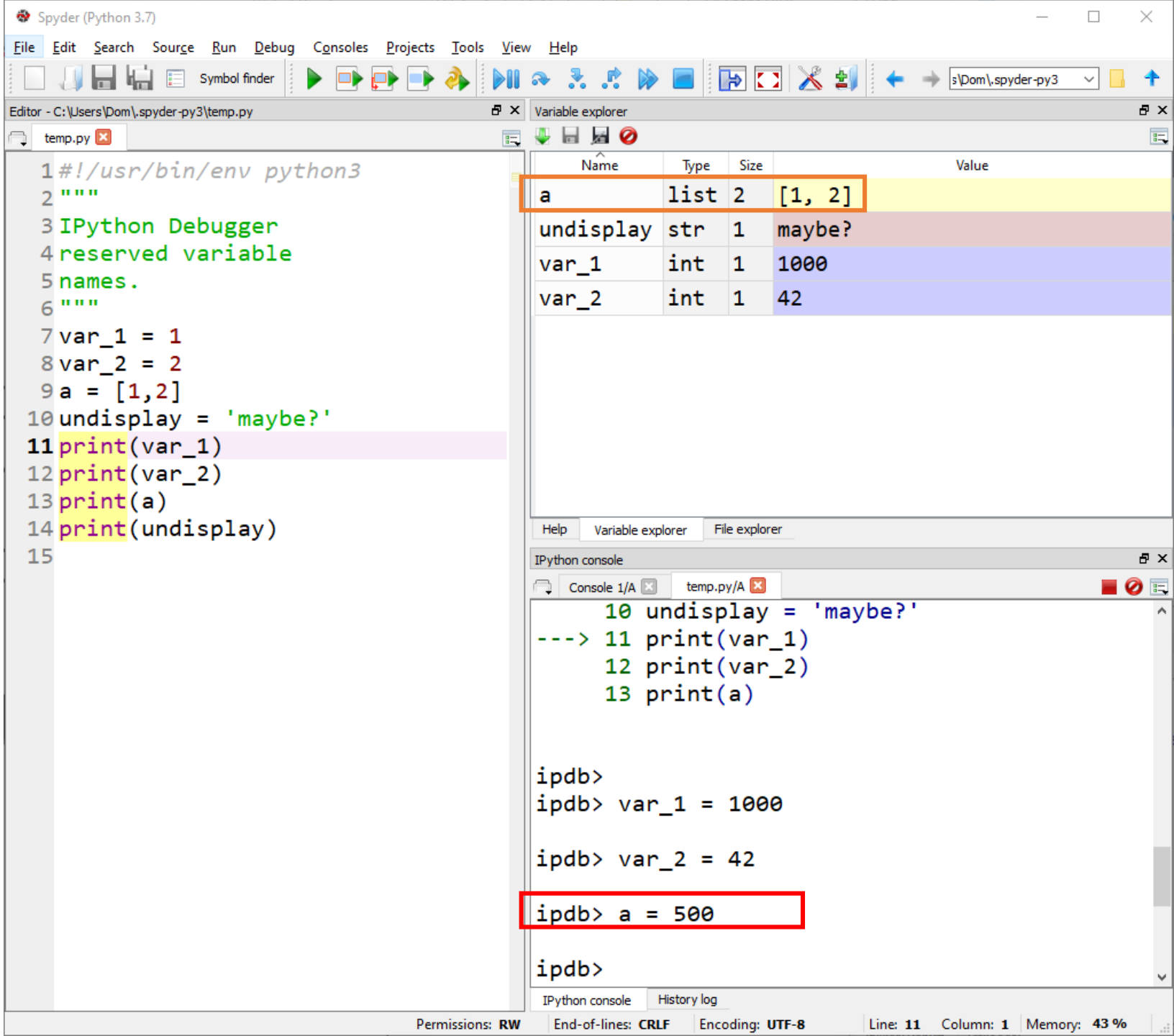
It works as well.



Let’s try the same with “a”.

It does not work.

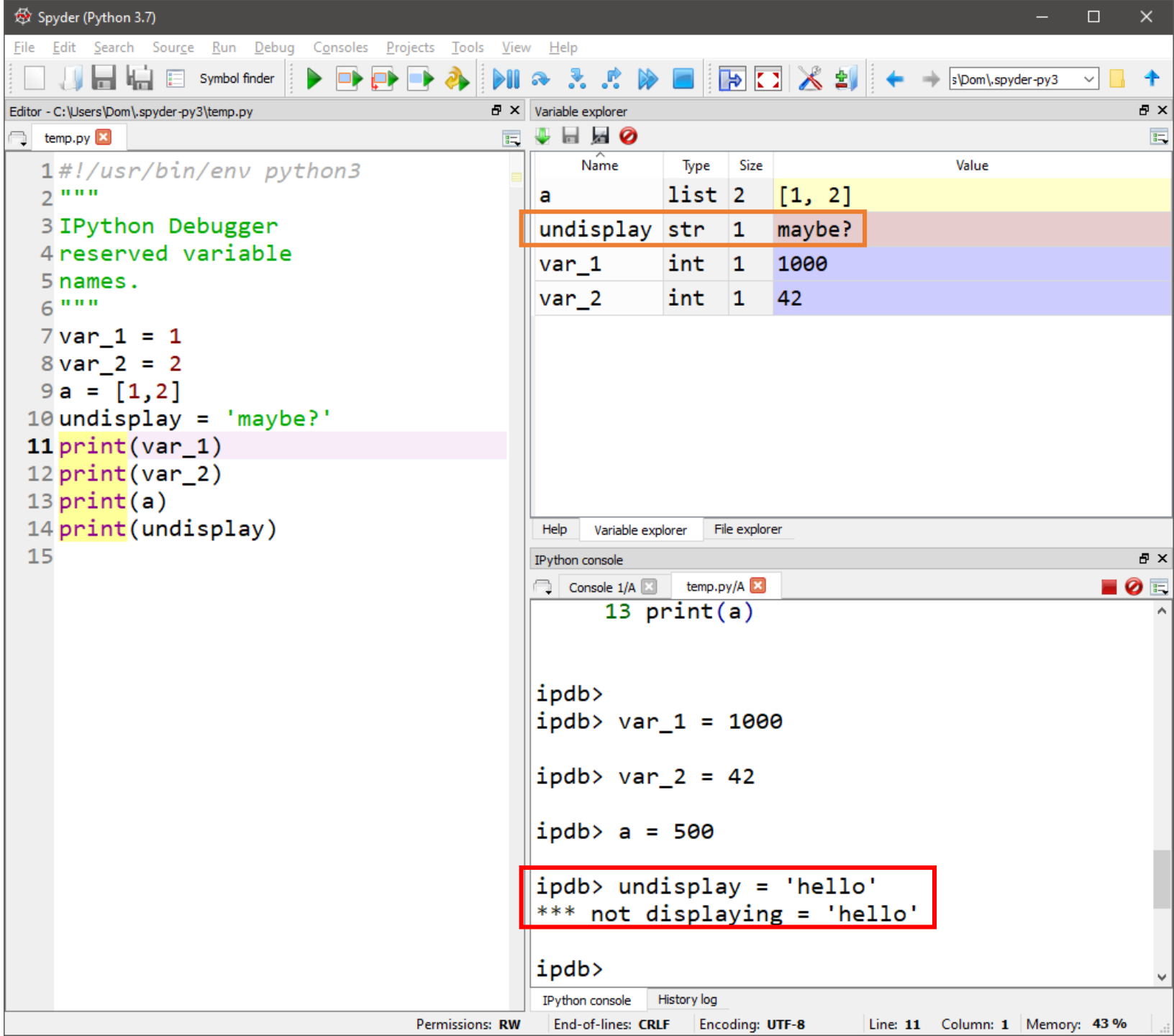
This is NOT because of the data type of “a”.



Let's try the same with "`undisplay`".

It does not work either.

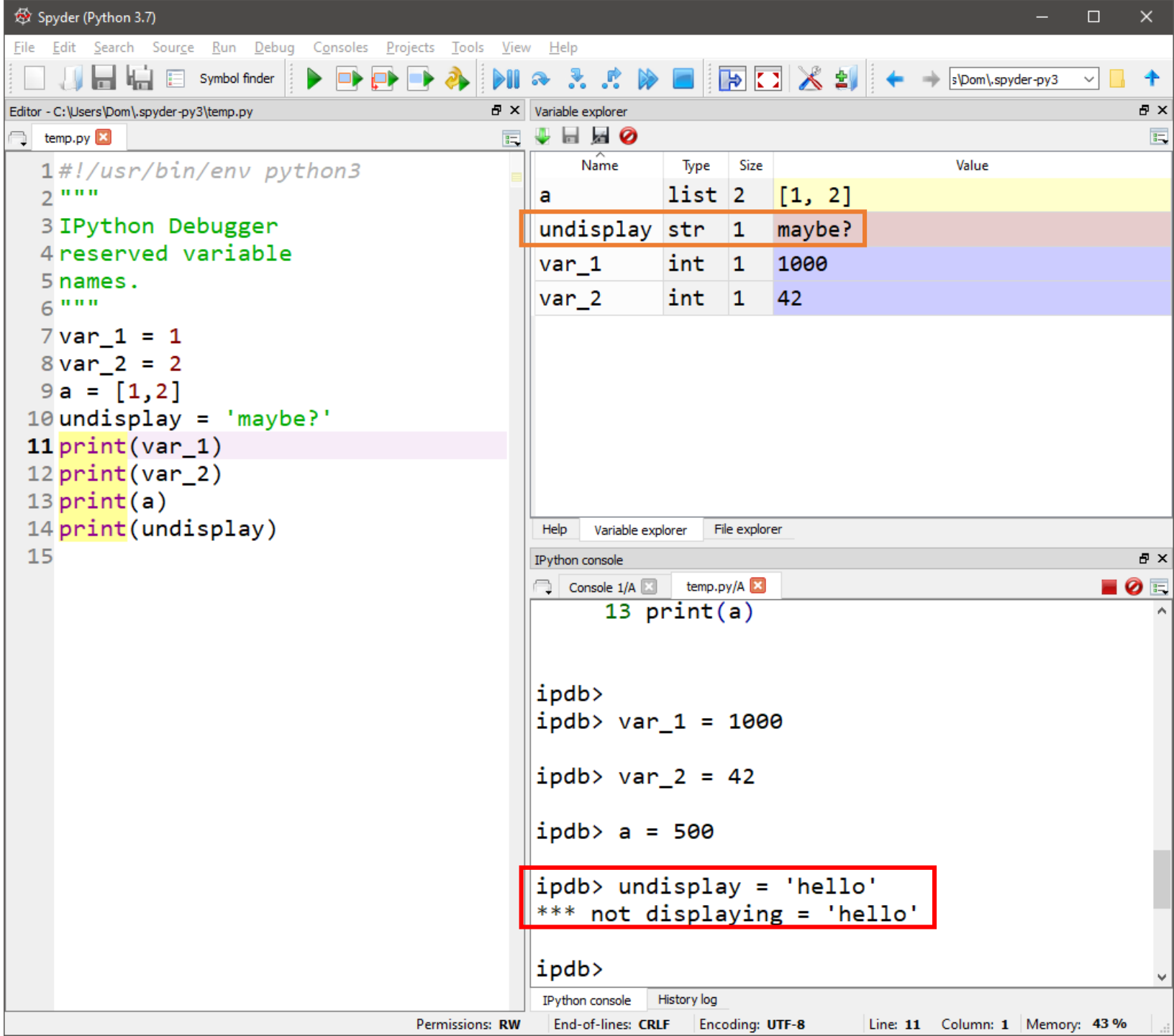
This is NOT because of the data type of "`undisplay`".



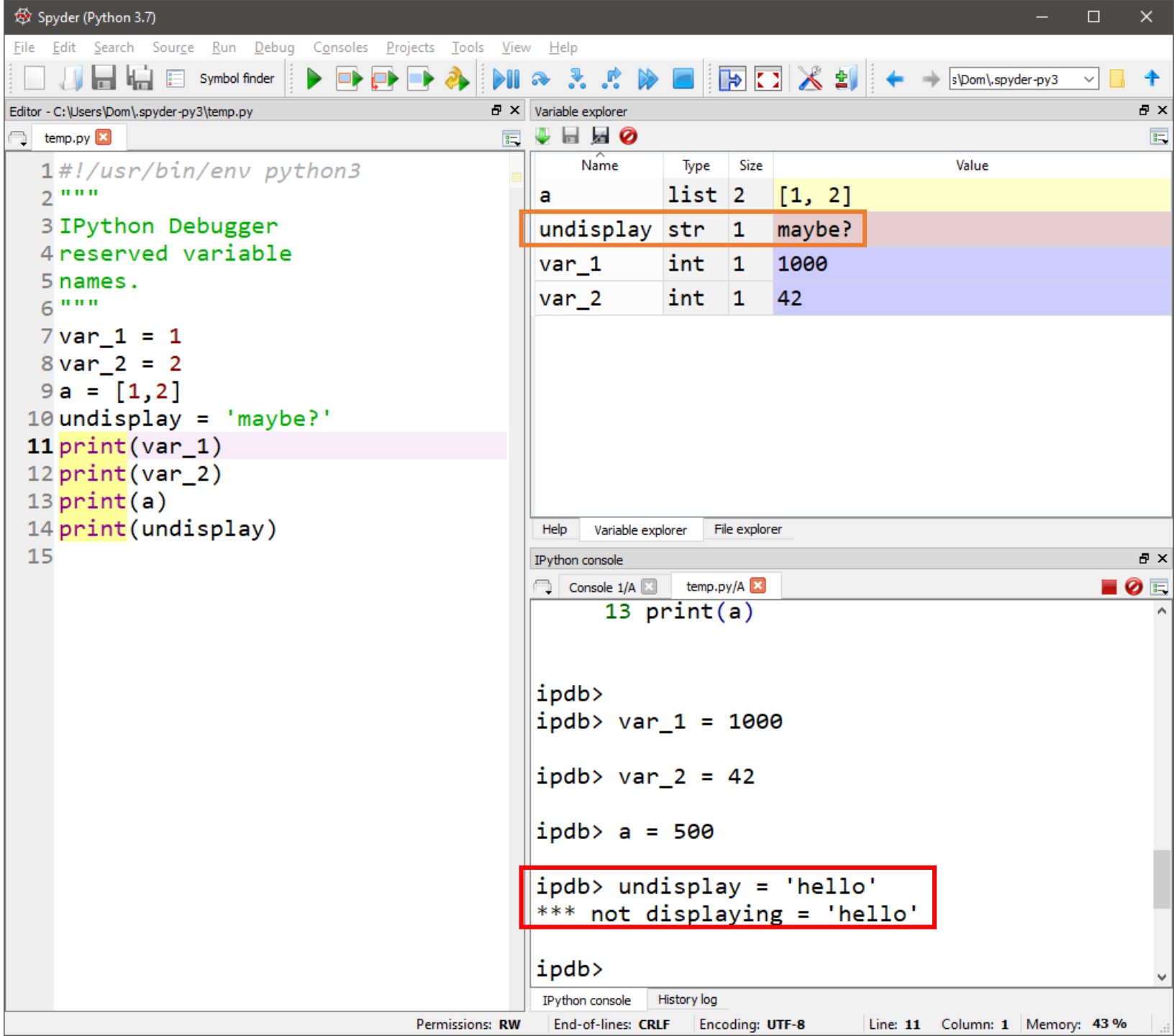
Note how we also got more output than we thought? What does

```
*** not displaying = 'hello'
```

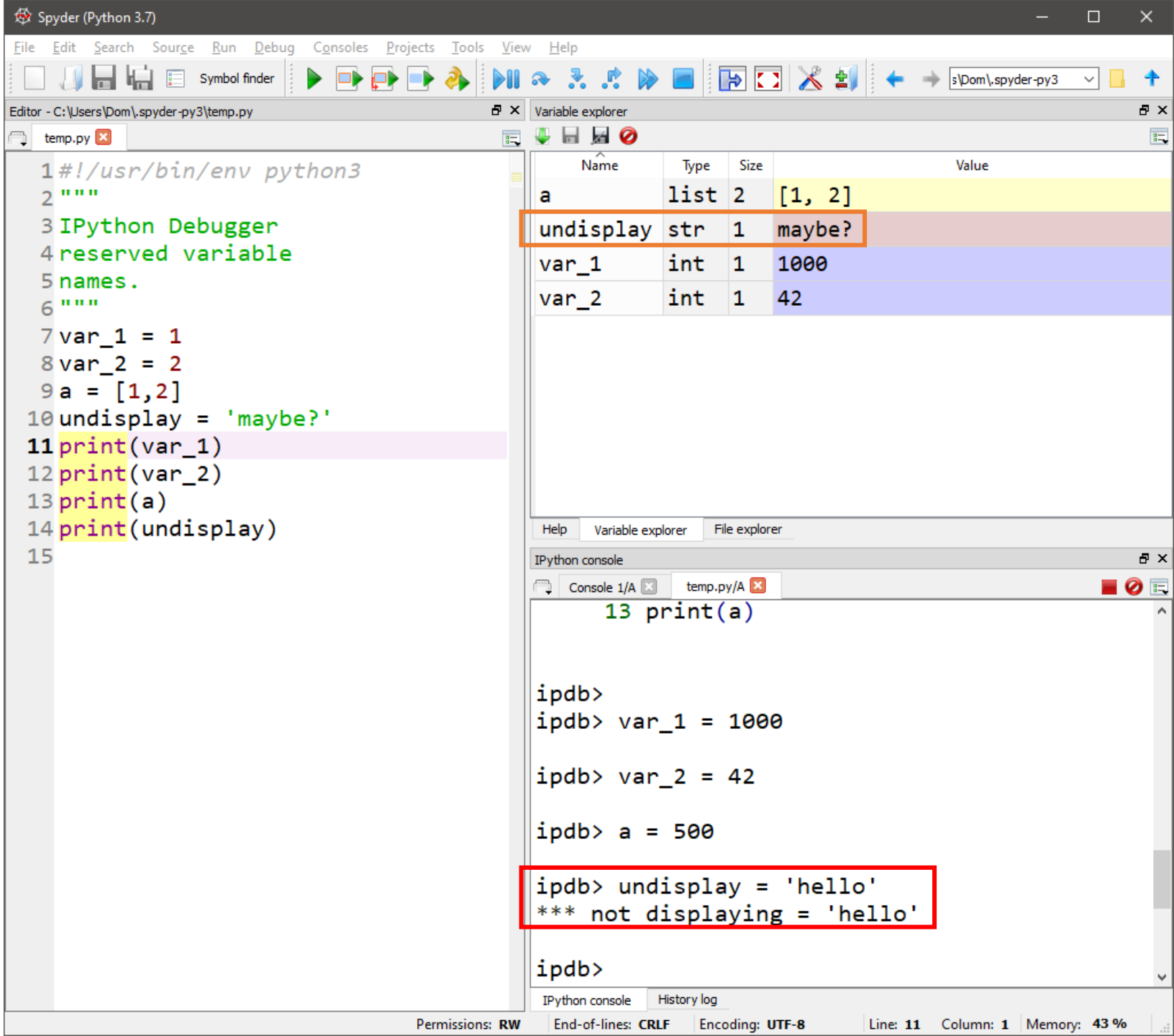
mean, and why is it there?



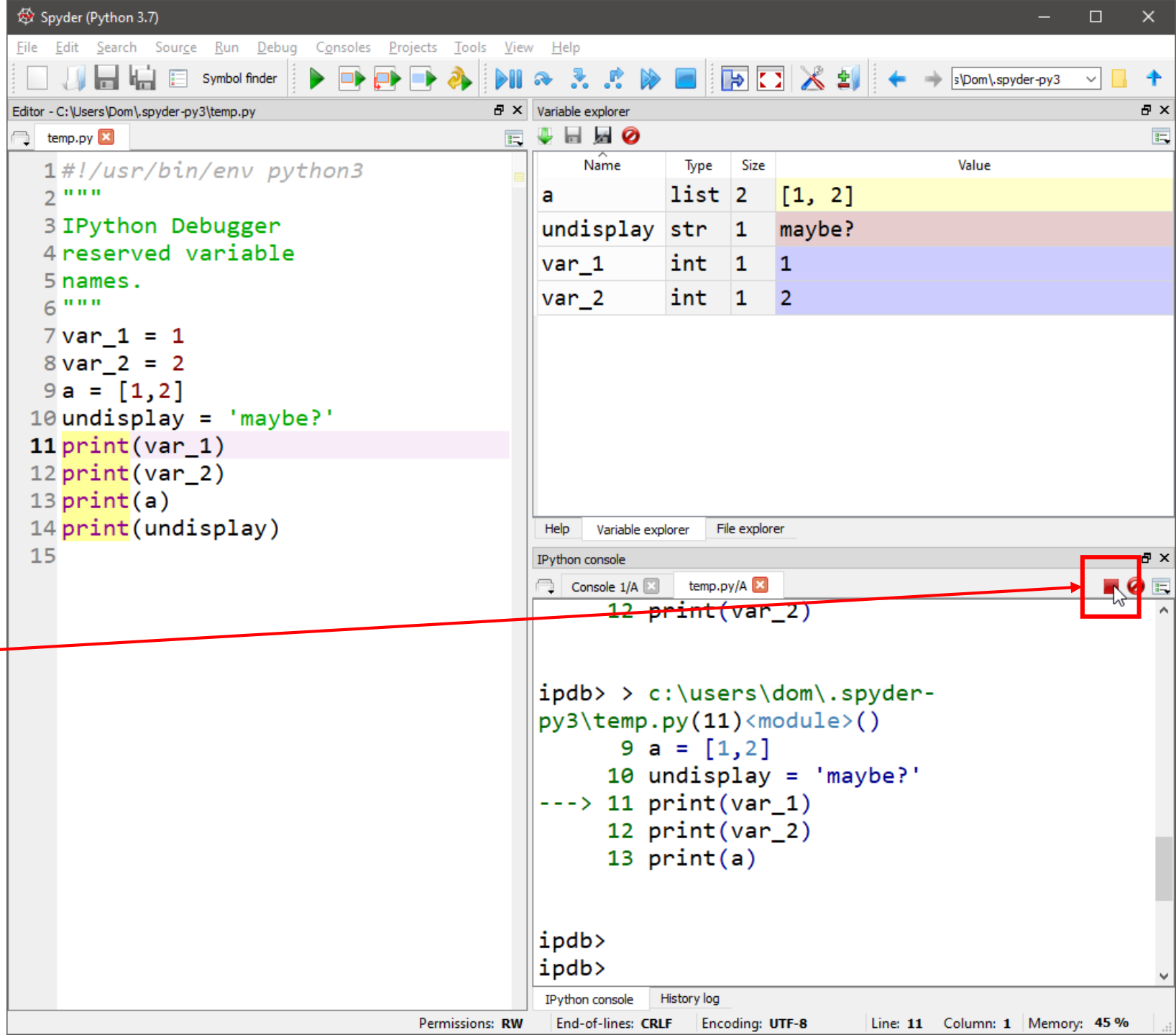
The debugger of Spyder, the IPython Debugger, has some reserved variable names of its own. "`undisplay`" has a special meaning, so does "`a`".



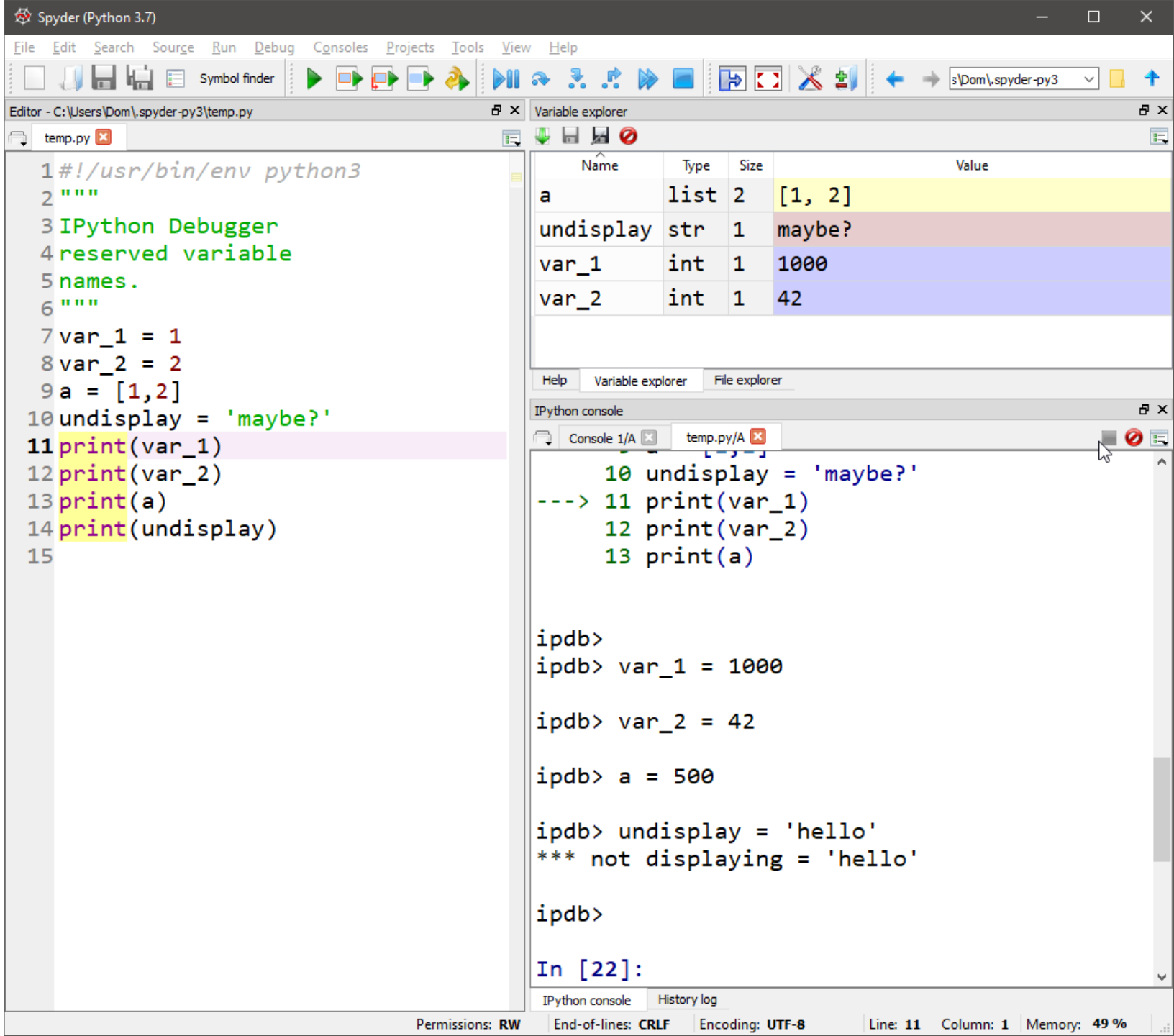
These special variables allow us to do certain things. If you are interested what you can do with these variables, search the internet.



By assigning new values to these variables, we can “break” the IPython Debugger accidentally. Click the red square to stop the current execution.

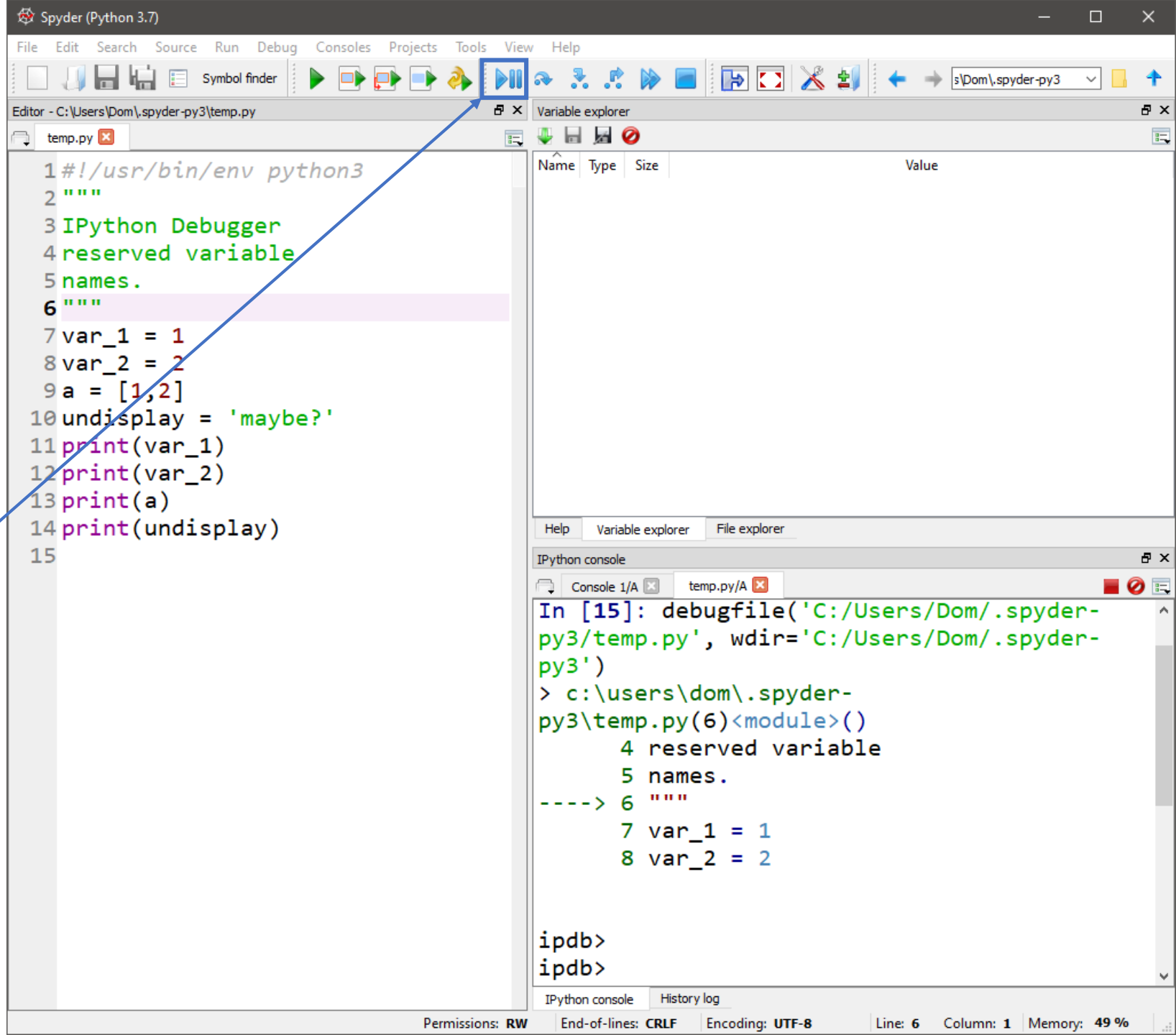


Then, you are back to normal (not in the debug mode anymore).



How do we know which variables have a special meaning?

Enter the debug mode again.



Then type "help" into the Console and the list gets displayed.

Note that you are good to use any of these variable names for your own variables in your code!

<u>The only thing</u> to be careful with is to not change their value by typing into the console WHILE you are in the debug mode. (It won't change the value and you might "break" the debugger").