Scripting Lesson04: Using Print Commands in Scripts

Last update: 01/12/12

These lessons were originally written in and for PSP 8, but have now been made "version-independent". All significant changes particular to versions higher than PSP 8 will be placed in green boxes like this one. Changes or new features are listed for the version in which they first appeared. Unless otherwise indicated, these changes apply to succeeding versions as well. Where a note/tip refers to a version of PSP and all higher versions, a + sign will be used to indicate this. For example, if a note/tip applies to PSP X and higher versions, I will use the convention "PSP X+". The lessons can be completed in any version of PSP that supports scripting. All scripts included with lessons were recorded in PSP 8.

Goals and Objectives

- To learn how to use print commands in scripts.

Lesson

A. Overview:

In our first three lessons, we examined the Script toolbar, became acquainted with the Script Output palette (SOP), used the PSP Script Editor to make changes to a script, and edited a script using the Text Editor.

In this lesson, we will be learning how to add print commands to a script.

B. Review of Lesson03:

Let's go through the exercises, just to be sure you understood them, and to provide a bit of review. The first exercise said to use the Text Editor to change the Execution Mode from Default to Silent on the Drop Shadow step of the **TEST01** script, and then run the changed script in both Silent and Interactive modes. Here's the line in the DropShadow step that had to be changed:

```
        'ExecutionMode': App.Constants.ExecutionMode.Default,
```

And here's what you should have changed it to:

```
        'ExecutionMode': App.Constants.ExecutionMode.Silent,
```

Did you have any problems with this one? If you did it correctly, the Drop Shadow dialog would not pop up when you ran the script, whether you ran the script in Silent or Interactive mode, because it was internally set to Silent. Good job if you got this one on your own! And you DID work on a copy of the **TEST01** script, right? That should be your mantra - "always work on a copy"!

The next exercise suggested that you experiment with changing some of the true/false parameters in the **TEST01** script, carefully thinking through what the expected changes to the results or dialogs should be, and verifying your expectations for each change. How'd you do on this one? Did you try out several different things? Some parameters that would produce verifiable differences would be the Transparent parameter in the NewFile command, or the UseForground parameter in the Fill command. Did you try these changes? If not, why don't you take a break and try them now, on a COPY of the **TEST01** script, of course - the last Lesson suggested naming this copy **Lesson03-2.PspScript**. It's really, really important that you try these exercises! We'll wait here while you go finish last week's homework! No excuses, now - the dog DID NOT eat your homework!

Here are some of the results you might have seen:

- If you change the Transparent parameter (NewFile step) to false, the image would have been created with a solid color background using the color last used in the NewFile dialog color box.
- If you change the UseForground parameter (Fill step) to false, the script will use the current Background Material; unchanged, it will use the current Foreground Material.

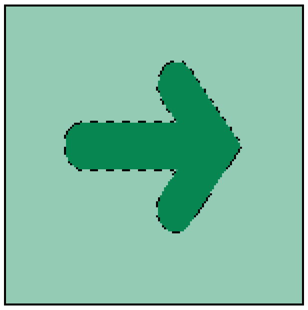
There are a few other true/false parameters in this script, but changing them may not produce any results you can see. In the case of the VectorBackground parameter in the NewFile step, changing the parameter may even cause the script to fail.

There is one other true/false parameter you may have tried, and that's the one for the NewLayer parameter in the DropShadow step. For this one, you may have seen a verifiable change - <u>or</u> you may have noticed that even though you changed the parameter from false to true, the shadow was still not created on a new layer. The explanation to this is complex, and made more so by the fact that in Silent mode, the shadow IS placed on a new layer, and in Interactive mode, it is NOT. In this particular case, what happens in Interactive mode is the correct behavior. What appears to happen in Silent mode is a script "bug" in PSP which was corrected in PSP 9.

The shadow for a selection should NEVER be placed on a new layer. Think about that for a minute. In our case, it seems like it would be okay, because our image is transparent, but the rules have to apply to all instances. First of all, notice that whenever you do place a shadow on a new layer, the shadow layer is BELOW the current layer in the Layers palette, so that it appears "behind" the image. This is an important concept, and key to understanding the rest of this discussion. The shadow layer has to be behind the current layer.

Now consider an image where there is no transparency, where the entire layer is filled, like in the following image:

In this image, we see a green arrow on a lighter green background. If you decide you want to apply a drop shadow to the green arrow, can you? Sure. With the Magic Wand click on the light green to select it, invert the selection, and now the arrow is selected:



Then just apply a drop shadow. Notice when you do, though, that the "Shadow on new layer" option is greyed out. Makes perfect sense, though, doesn't it? If you were able to make that drop shadow on a new layer, where would that layer be? Behind the current layer, right? So would you even be able to see the shadow? Not hardly. Hence, shadows for selections should not have that option even available to them. Try it yourself outside a script, and you'll see how it works. Create an image, fill with some color, make a selection and fill the selection with another color. Now apply a drop shadow to the selection - you should see that the Shadow on new layer option is greyed out. Think about this for a while. It's a perplexing concept, but valid nonetheless, and when you get it, it will be like a light just turned on! That's how it was for me!

Just one other note here - the above discussion applies to plain, ordinary selections. It does not apply to floating selections, which are separate from the layer, sort of "floating above the layer". Nor does it apply to instances where a drop shadow is applied to a layer that has no selection. In both of these instances, the Shadow on new layer option is completely valid and correctly available:

- If there is a selection in your image which is "floating" and you check the Shadow on new layer option, the shadow layer would be inserted between the floating selection (called "*Floating Selection*" in the Layers palette) and the layer it came from, and would be called "Floating Selection Shadow 1" in the Layers palette.
- If you apply a shadow to a layer called "Layer A" in the Layers palette and you check the Shadow on new layer option, the shadow would appear on a new layer below "Layer A" called "Layer A Shadow 1." In this case, because there is no selection on "Layer A", the drop shadow would be applied to all the non-transparent pixels of "Layer A" - the image or object or tube on that layer.

Therefore, for those using PSP 8, this bug is there and won't be fixed now that PSP 9 is out. Just remember it exists and be careful in your scripts that you don't "edit in" this problem. A recorded script will be okay - but when you edit scripts, you can change parameters so that scripts behave in unpredictable ways. Therefore, I'm just cautioning you not to make this (or any change) in a script without first testing to be sure it's a valid change.

PSP X+ Update:

For some unknown reason, in PSP X+ the Drop Shadow command was changed to allow the application of a Drop Shadow for a selection to a new layer. Though this makes no sense whatsoever on an opaque image, be aware that it is now allowed. Use this capability cautiously in a script, so as not to create hidden or missing drop shadows.

And now on to that first challenge - did you figure out how to use a square or a triangle selection shape in the Selection step? I'll just bet you did, and you tested it, too, didn't you. For those who are still chasing that dog (ahem), here's the parameter that needed to be changed:

```
        'SelectionShape': App.Constants.SelectionShape.Rectangle,
```

All you had to do was substitute other selection shape names for the name I've marked in red above, like this:

```
        'SelectionShape': App.Constants.SelectionShape.Triangle,
```

And there are 15 different selection shape names to choose from - did you try several? We can wait while you go practice! Come back here when you're finished. And if the name of the selection shape has a space, drop the space when you use the name - so if you're using the Star 1 shape, code App.Constants.SelectionShape.Star1. We'll learn how to look this information up later, but for now, just believe me!

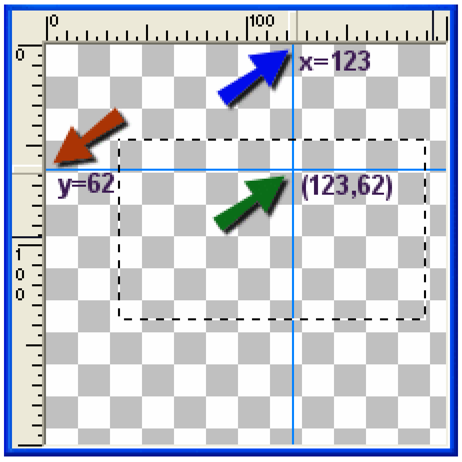
And did you try making your image larger? Did you figure out how to alter the script to do that? Good for you. To do this, you had to alter the NewFile step, changing the Width and/or the Height parameters, which can be changed to just about anything:

```
     'Width': 200,
     'Height': 200,
```

There is the possibility that the new selection shape you used in the above challenge no longer was filled with the Foreground Material - did that happen to you? If so, read the next section, and see if you can figure out why that occurred. We'll come back to it after looking at Challenge#2.

The second challenge introduced the Point parameter in the Fill step. As mentioned in that challenge, the Point parameter tells PSP exactly where to click when using the Flood Fill tool. The challenge was to think about what would happen if you changed the Point parameter in this step to something like (199, 199).

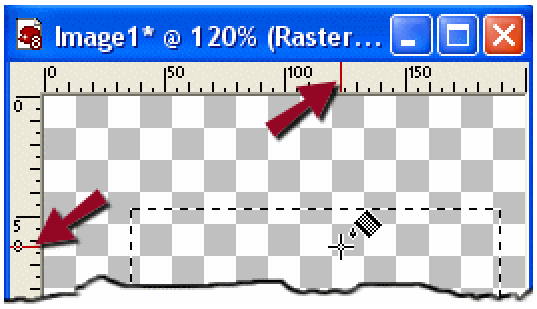
But before we can respond to this challenge, we have to understand the notation used for points. I don't know what your Point parameter is, but when I first recorded the **TEST01** script, my Point parameter was (123.5,62.5). This is the exact point where I clicked with the Flood Fill tool to fill my rectangle when I was recording this script. Take a look at the image below, which shows this point at the green arrow. The blue arrow shows the 123 on the horizontal ruler (or x-axis), and the red arrow shows the 62 on the vertical ruler (or y-axis). I've enabled the guides (blue lines) so you can see where the exact point (123,62) is:



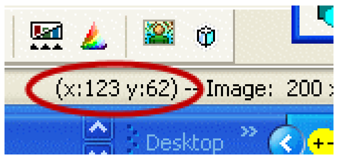
Even though PSP often records those points with a decimal, you can just drop the decimal to find the point, using (123,62). There's actually a reason PSP does that, but it's not important to this discussion.

As a review, here's how to locate any fill point:

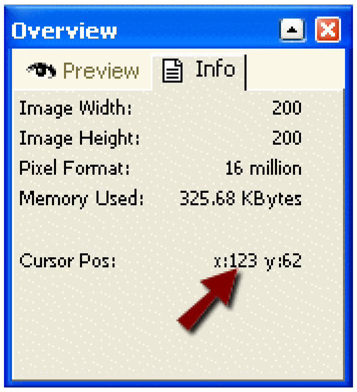
- Find the first coordinate along the horizontal ruler (x-axis).
- Staying right at that horizontal distance, move down until you find the second point on the vertical ruler (y-axis). You'll actually see small black lines moving along your ruler as you move the cursor. In the following image, I've found the exact spot (123,62), and I've colored those small black lines moving along the ruler in red so you can see them - do you see them where the red arrows are pointing?



So that's one way you can find an exact point. Another way is to watch the lower right corner of the screen on the Status toolbar (choose View...Toolbars and click on the Status toolbar). As you move the cursor around the screen, its exact position is always indicated on the Status toolbar at the bottom, right of your screen. When I made the above screen print, the Status toolbar looked exactly like this (the coordinates of the fill point have been circled in red):



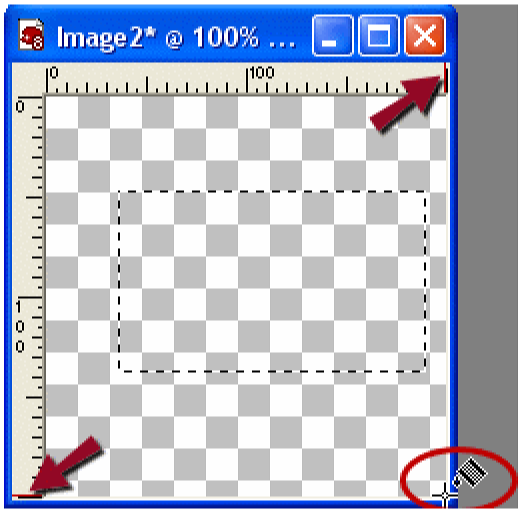
The cursor position is also reflected on the Info tab of the Overview palette (which can be activated by pressing the F9 key), as shown below:



```
Image Width:          200
Image Height:         200
Pixel Format:  16 million
Memory Used: 325.68 KBytes

Cursor Pos:    x:123 y:62
```

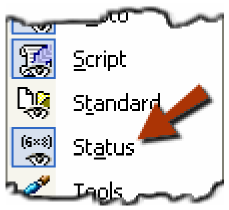
Where you click is important! Notice the point (123,62) is within the selection. Which brings us back to the original question which was what would happen if you changed the Point parameter in this step to something like (199, 199)? That point is here:



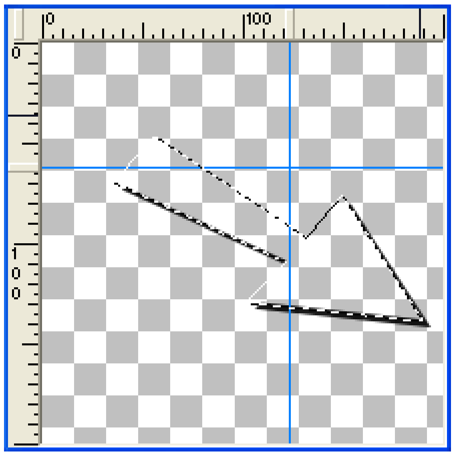
If you click there, nothing will happen! When you have a selection and you click to fill it, you must be within the selection, or no fill will occur! Try this on an image - create a new image and make a selection. Click with the Flood Fill tool outside the selection. Nothing happens, right? The same thing will happen in a script. Change the Fill Point parameter to (199,199), save your script, and run it. Same as in real life, nothing will happen!

There are several lessons we can learn from the previous discussion:

1. First of all, it's extremely important in a script to know where those fill points are. As a script author you must be aware of this, and ensure your fill points will work for any user running your script.
2. The rulers can be very handy in choosing points. If you are not seeing your rulers at the top and left side of your images, as in my images above, choose View...Rulers. I leave my rulers viewable all the time.
3. The Status toolbar contains much vital information that can make your life (and scripting) much easier. If you do not see the numbers in the Status Bar like I indicated in the image above, you probably do not have that toolbar enabled - enable it by choosing View...Toolbars, and clicking on the Status toolbar (which on my system, is always enabled):



Finally, back to why, with some selection shapes, the selection was not filled with the Foreground Material as in the original script and with the original rectangle shape. Do you now know the reason? If you're not sure, run that script again, with the shape that did not fill, and then find your fill point in your image. Where is it? Outside the selection, huh? Here's one I did, using Arrow3 as my selection shape:



The intersection of the blue Guide lines on the image shows my fill point of (123,62) - definitely outside the selection image. So now you know why the selection did not fill! But could you make it work? Sure you could - just change the fill point to something within the selection, and you're in business! Be sure to try that!

Well, that takes care of the exercises. I hope you learned something from them, and that you're beginning to feel a bit more comfortable with editing scripts! Now on to today's lesson.

C. Adding Print Statements to A Script:

The print command is used to write or print information to the SOP. It can be used to print documentation or to give instructions to users or list values calculated in a script - it can be used to cause just about any information or value to be printed to the SOP. Note that we're not talking about printing scripts here - we're talking about writing or printing information to the SOP during the execution of a script. So let's take a look at this command and see how we can use it in PSP scripts.

The general format of the print command is very simple:

```
print expr
```

where *expr* represents the expression that you want to print. If the expression is a string (or collection of letters, numbers, or other characters), it must be enclosed in double or single quotes, like this:

```
print "This is Lesson04"
print 'I am learning some Python'
```

Strings are just sequences of characters treated as a unit, like "Bill Smith" - they're what we often refer to as text, though there can be numbers and other characters in strings as well. The important thing to remember here is that strings must be enclosed in quotes (either single or double) - we'll actually learn why later.

If the expression is a number, no quotes are needed:

```
print 4000
```

Because the results of the print statement go to the Script Output palette, you want to be sure it's activated. Let's open our **TEST01-copy2** script in the Text Editor and add some print statements.

From the previous lesson, you probably have the following lines just before the "# New File" ("# FileNew in PSP 9) line near the beginning of the script:

```
### This is my very first script
### I'm inserting comment lines now
### I'm actually editing a script - WOW!
```

Place your cursor at the end of the last line in the comment, and press the ENTER key twice. This will insert a blank line, and put you on a new line. We need to be in column 5, so move your cursor back to the left margin (column 1) and insert 4 spaces - remember to use the space bar to insert spaces. Other text editors will always return the cursor to column 1, so use the space bar to move over to column 5. Now type print, leave a space, and type the following string: "This is a copy of the TEST01 script". Your screen should look like this:

```
    ### This is my very first script
    ### I'm inserting comment lines now
    ### I'm actually editing a script - WOW!

    print "This is a copy of the TEST01 script"
```

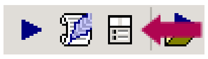
Note: Your comment lines might not have started in column 5 - that's perfectly okay - but your code will look just a bit different from mine as we go along. Your print command, however, must start in column 5.

Be sure you do NOT use an uppercase letter on the word print - it must be written with a lowercase "p". Python commands are case-sensitive, so you have to get them right or the Python interpreter will spit them out! Let's enter one more print line. Hit ENTER to get to the next line - the cursor should be right below the word print, in the 5th column. Type print and then this string: "This script will create a new image". Hit ENTER again and type print and then this string: "containing a filled, shadowed selection". Your script should now look like this:

```
    ### This is my very first script
    ### I'm inserting comment lines now
    ### I'm actually editing a script - WOW!

    print "This is a copy of the TEST01 script"
    print "This script will create a new image"
    print "containing a filled, shadowed selection"
```

Be sure your script looks exactly like mine - at least for the print lines. Be careful not to drop any quotes, now! We're printing strings, so we need those quotes. When you have it right, save your script and run it from the Script toolbar in Silent mode (be sure the Interactive Script Playback Toggle button is NOT pressed, and looks like this:



Note: In Interactive mode, most steps are NOT listed in the SOP - that's why I want you running your scripts, in this lesson at least, in Silent mode.

Your now-familiar image will be created, and if you look at the SOP, you'll see your printed statements. The SOP should look like this:

```
Executing RunScript
Executing EnableOptimizedScriptUndo
This is a copy of the TEST01 script
This script will create a new image
containing a filled, shadowed selection
Executing New File
Executing SelectDocument
Executing Selection
Executing Fill
Executing Drop Shadow
Executing SelectNone
Executing DeleteLayer
```

Note that in PSP 9+, an additional line is inserted at the end of every script, telling you whether or not the script completed successfully:
Script 'TEST01-copy2' has completed successfully.

I will not always include this final line when I show SOP results. Just remember that you will always see it in PSP 9+, even if I don't indicate its presence.

Cool or what! But the messages you created are kind of lost with all that formal stuff, aren't they? They'd stand out better with some blank lines before and after them, wouldn't they? We can do that! Just insert a print command with no text (or a print command with empty quotes) before and after those lines, like this:

```
    print
    print "This is a copy of the TEST01 script"
    print "This script will create a new image"
    print "containing a filled, shadowed selection"
    print
```

Save the script and run it - it should produce the following in the SOP:

```
Executing RunScript
Executing EnableOptimizedScriptUndo

This is a copy of the TEST01 script
This script will create a new image
containing a filled, shadowed selection

Executing New File
Executing SelectDocument
Executing Selection
Executing Fill
Executing Drop Shadow
Executing SelectNone
Executing DeleteLayer
```

Shows up a little better now, doesn't it? Would probably look even better if it was indented a bit! To do that, just insert spaces before the words, like this:

```
    print ""
    print "  This is a copy of the TEST01 script"
    print "  This script will create a new image"
    print "    containing a filled, shadowed selection"
    print ""
```

Notice I even indented the 3rd line a little further than the others, just so it looks better. And I used the empty quotes for the blanks lines. Save the script and run it again - here's what the SOP looks like now:

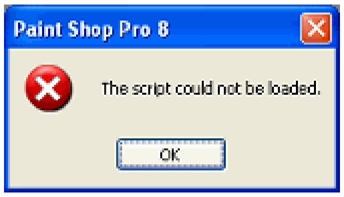
```
Executing RunScript
Executing EnableOptimizedScriptUndo

  This is a copy of the TEST01 script
  This script will create a new image
    containing a filled, shadowed selection

Executing New File
Executing SelectDocument
Executing Selection
Executing Fill
Executing Drop Shadow
Executing SelectNone
Executing DeleteLayer
```

Oops, I forgot to stop and see if everyone is keeping up! If you left off any quotes, you'll get an error. Of course, I know YOU are doing just perfectly, and didn't get any errors at all, but just so you know what OTHERS might get, drop the quote at the end of the last line and see what happens. Save the script and run again.

You should get a message box like this:



The script could not be loaded.

and the SOP will contain some dark red, ugly messages like this:

```
 File "<string>", line 21
   print "       containing a filled, shadowed selection
                                        ^
 SyntaxError: invalid token
```

Note: Errors like this one (missing quotes) which violate one of Python's grammatical rules and prevent your program from running are called "syntax errors". You'll see a lot of these in your scripting career, even after you become somewhat more experienced, believe me! There are other types of errors, but syntax errors are probably the most common, and the ones you'll "create" most often when you're editing your scripts.

The error handler attempts to indicate the place where the error occurred using a caret (the character above the number 6 on the keyboard) under that location. Though it usually finds the line with the error, it does not always do well indicating the exact location. In this case, the caret is under the word "shadowed", which is not exactly where the error occurs. But once you have the line number, you can usually find the syntax error if you look really hard.

SyntaxError in PSP 9, PSP X, PSP XI, and PSP X2:

```
 File "C:\PSP\PSP9 Files\Scripts-Restricted\TEST01-copy2.PspScript",
line 21
   print "       containing a filled, shadowed selection
                                        ^
 SyntaxError: EOL while scanning single-quoted string
```

In these versions, the path to the script is given and the message is slightly different. In PSP X3+, the path name was dropped, and the message reverted back to the format in PSP 8. Why this change, you ask? I don't know!

Well, let's go back and fix that error! Just add the quote back in, and save again. What do you think would happen if you didn't have those print commands right under each other? That's right - another syntax error, or an indentation error. If one of the print commands is only indented 3 spaces instead of four, like this:

```
    print
    print "  This is a copy of the TEST01 script"
   print "  This script will create a new image"
    print "    containing a filled, shadowed selection"
    print
```

you'll get this error:

```
IndentationError: unindent does not match any outer indentation level
```

You should try these variations just to see what error messages are generated. It's good information to file away while you're learning. Then, when something like this happens in real life, and it WILL HAPPEN, that I can assure you, you won't be so flubbergusted! Try all sorts of things - spell print wrong, or use an uppercase P, or mix double and single quotes - and see what errors are generated. We'll wait here while you run some tests.

Did you get a lot of syntax errors? Now that you've seen them, remember them when you're doing some 'real' coding. Usually it's a misspelled word, or uppercase when it should be lowercase (or vice-versa) - everything has to be 'exact' or the program just won't work!

There's another way you can break your print lines without coding separate lines. You can actually code just a single continuous line, inserting the **new line** characters, or "new line escape sequence" (\n) where you want the breaks to occur. You'll notice when you do this that the text will wrap within the script, filling several lines, but there's only one print command needed. Let's try this now, printing this text: "This is a long sentence that I want to print on more than one line, but I don't want to have to code individual print statements for each line, so I will make it happen by inserting new line characters!". At every location where you want the sentence to break to a new line, insert the "new line escape sequence" (\n) - that's backslash and the letter "n". Your print command will look something like this within your script:

```
    print
    print "This is a long sentence that I want \nto print on more than one
line, \nbut I do not want to have to code \nindividual print statements \nfor each
line, so I will make it happen \nby inserting new line characters!"
    print
```

Kind of a jumble, but the output in the SOP will look like this:

```
This is a long sentence that I want
to print on more than one line,
but I do not want to have to code
individual print statements
for each line, so I will make it happen
by inserting new line characters!
```

And we did it with just one print command!

Note: Though it's tempting to use the ENTER key to place each section of the long quote on a different line so it's easier to read, this will cause a syntax error. Just keep right on coding, letting the text wrap in your Text Editor. No carriage returns (hitting the ENTER key) allowed!

Here's another way you can control the printed output - use the **tab** characters or "tab escape sequence" (\t). This will allow you to indent lines a certain number of spaces. Say we want to have the following print in the SOP:

Before you run the script, do the following:

1. Create a new image.
2. Add the tube of your choice.
3. Set the Foreground Material to white.
4. Set the Background Material to black.

We could do it with individual print commands, inserting spaces to get the numbers lined up beneath each other, but it's much easier to use the "tab escape sequence" (\t), like this:

```
    print
    print "Before you run the script, do the following: \n\t1. Create a new
image. \n\t2. Add the tube of your choice. \n\t3. Set the Foreground Material to
white. \n\t4. Set the Background Material to black."
    print
```

Notice I used the escape sequence for a new line before each tab - if you don't do that, you won't change lines before tabbing. The script will work, but will produce one long line in the SOP with about 8 spaces before each number! You try this now - go ahead, type it in, and give it a whirl! We'll just wait here until you finish.

What if you have a quote within the text you want printed? Perhaps you want to print a string something like the following:

Be sure to set the "Foreground Material" to transparent before you run the script.

If you code the print command this way, can you guess what will happen?:

```
    print
    print "Be sure to set the "Foreground Material" to transparent before you run the
script."
    print
```

You got it - another syntax error. The caret might be anywhere underneath that line, probably quite near that quote before Foreground. The Python interpreter sees the opening quote, and the next matching quote it finds (the one before Foreground) is considered the closing quote. The interpreter has no idea what that next sequence of characters is, so it throws an error - like an interpreter hissy fit!!!

Is there a way to handle this, you ask? Surely, surely, surely! You can use double quotes for the outside quotes, and single quotes for the inside quotes, (or vice-versa) like this:

```
    print
    print 'Be sure to set the "Foreground Material" to transparent before you run the
script.'
    print
```

Same thing applies if you use an apostrophe. If you want to print this string:

It's important that you read these notes before proceeding.

code it like this:

```
    print
    print "It's important that you read these notes before proceeding."
    print
```

Summary

That just about covers the basics of using the print command. Let's summarize:

- The print command itself is written in all lower case letters - Python is case-sensitive, so this is important.
- If you want to print a string (a sequence of characters), enclose the string in either single or double quotes.
- To insert blank lines, code the print command with no quotes, or with empty quotes.
- To indent print lines, code blank spaces inside the quotes before the text.
- To include quoted material, or words with apostrophes, use external double quotes and internal single quotes.
- To break printed text into separate lines without using separate print commands, use the new line escape sequence (\n) - but do not use the ENTER key while within the quotes!
- To indent print lines, use spaces, or the tab escape sequence (\t).

In the next lesson, we'll do some more work with the print command, and see how we can use print statements to assist in debugging problem scripts.

Suggested Practice Activities

1. Make a copy of the script we used in today's lesson, calling it **Lesson04-1.PspScript**. Using the Text Editor, remove all the print lines we inserted during the lesson. Save the script and run it to be sure it works. Now insert a print statement before the Fill command that says "The selection will now be filled with the Foreground Material.". Run the script again and check the SOP for your message.
2. Using a copy of the script created in the first exercise, and calling it **Lesson04-2.PspScript**, change the SelectionShape parameter in the Selection step to something other than Rectangle. Then insert a print statement before the Selection command telling the user what shape the selection is going to be. Run the script and check the messages in the SOP - do they fit with the image you created?
3. Make a copy of any of the scripts you've created and call it **Lesson04-3.PspScript**. Insert individual print statements at the end of the script that send the following to the SOP:

   This script did the following:
   1. Created a new image.
   2. Added a selection.
   3. Filled the selection with the Foreground Material.

   Thank you for trying this script!

   Run the script and check the messages in the SOP.

4. Challenge #1: Repeat the above exercise, this time using only a single print command with the new line escape sequence to create the separate lines. For this exercise, use inserted spaces to create the indented list. Call the script for this exercise **Lesson04-Challenge#1.PspScript**. Don't forget to run the script and verify everything printed as expected.
5. Challenge #2: Repeat the above exercise, this time using a single print command, the new line escape sequence to create the separate lines, and the tab escape sequence to create the indents. Call the script for this exercise **Lesson04-Challenge#2.PspScript**. Run the script to verify the print statement did what you wanted it to.

   Note: In Exercises 3, 4, and 5, you will be printing the same thing in 3 different ways, just like we learned in the lesson. Your aim is to produce results in the SOP that are nearly identical, no matter which method of coding print statements is used.