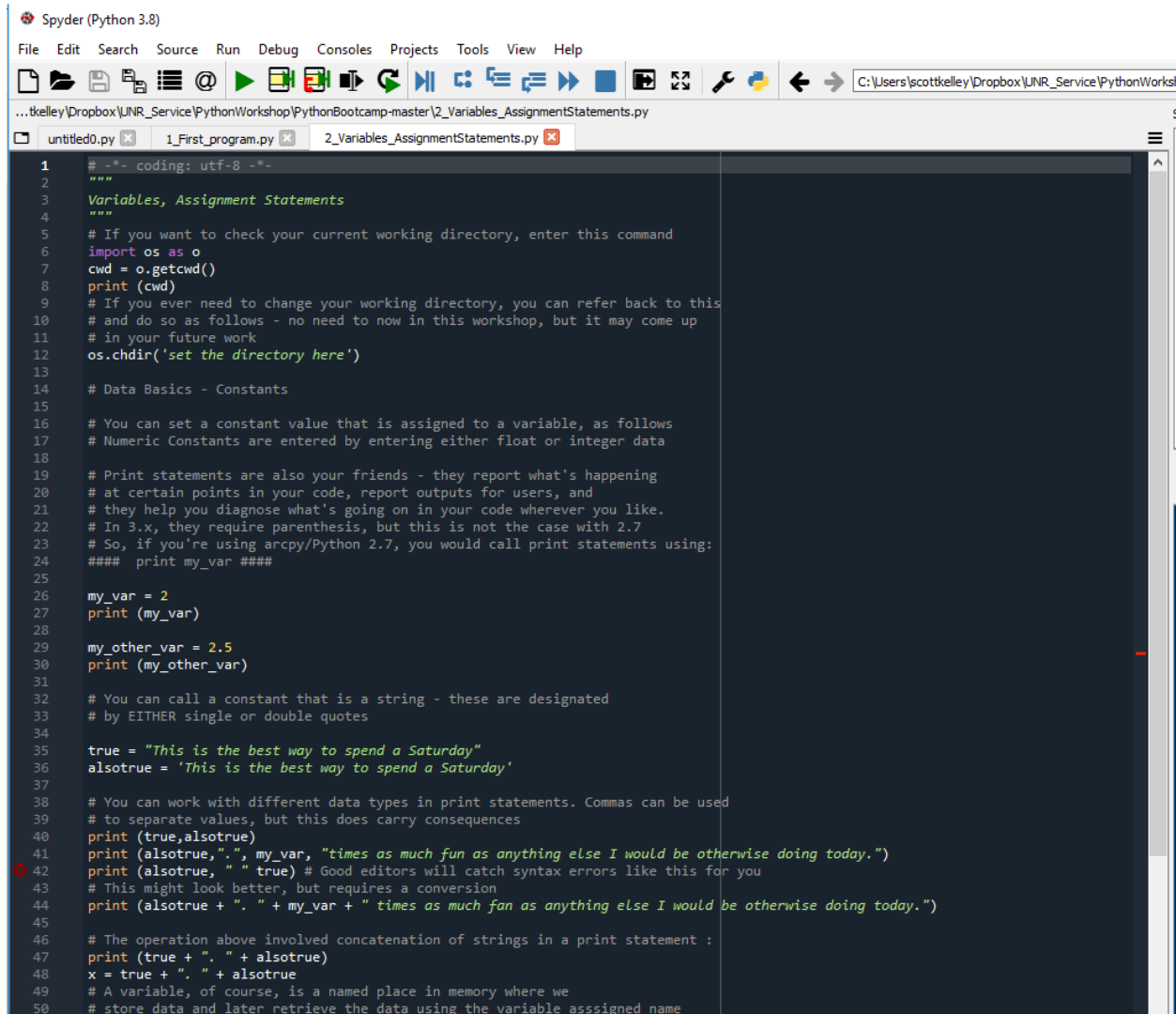


Module 2 – Variables and Assignment Statements

Instructions

- 1) From the Anaconda Navigator Home screen, launch Spyder
- 2) In Spyder, go to “File”, then “Open”, then navigate to the location where you saved: “2_Variables_AssignmentStatements.py”, and select it to open the script file. When you do, you should see something like this:



```
1  # -*- coding: utf-8 -*-
2  """
3  Variables, Assignment Statements
4  """
5  # If you want to check your current working directory, enter this command
6  import os as o
7  cwd = o.getcwd()
8  print (cwd)
9  # If you ever need to change your working directory, you can refer back to this
10 # and do so as follows - no need to now in this workshop, but it may come up
11 # in your future work
12 os.chdir('set the directory here')
13
14 # Data Basics - Constants
15
16 # You can set a constant value that is assigned to a variable, as follows
17 # Numeric Constants are entered by entering either float or integer data
18
19 # Print statements are also your friends - they report what's happening
20 # at certain points in your code, report outputs for users, and
21 # they help you diagnose what's going on in your code wherever you like.
22 # In 3.x, they require parenthesis, but this is not the case with 2.7
23 # So, if you're using arcpy/Python 2.7, you would call print statements using:
24 ##### print my_var #####
25
26 my_var = 2
27 print (my_var)
28
29 my_other_var = 2.5
30 print (my_other_var)
31
32 # You can call a constant that is a string - these are designated
33 # by EITHER single or double quotes
34
35 true = "This is the best way to spend a Saturday"
36 alsotru = 'This is the best way to spend a Saturday'
37
38 # You can work with different data types in print statements. Commas can be used
39 # to separate values, but this does carry consequences
40 print (true,alsotru)
41 print (alsotru,".", my_var, "times as much fun as anything else I would be otherwise doing today.")
42 print (alsotru, " " true) # Good editors will catch syntax errors like this for you
43 # This might look better, but requires a conversion
44 print (alsotru + ". " + my_var + " times as much fan as anything else I would be otherwise doing today.")
45
46 # The operation above involved concatenation of strings in a print statement :
47 print (true + ". " + alsotru)
48 x = true + ". " + alsotru
49 # A variable, of course, is a named place in memory where we
50 # store data and later retrieve the data using the variable assigned name
```

Of course, the pathnames will differ for you, and as mentioned in the Introduction, the style may look different (and is adjustable), but the content of lines 1-50 should look like the screenshot above. We'll walk through these documented segments next.

3) Variables, Print Statements, and a first look at Functions.

First, review lines 5-8.

```
# If you want to check your current working directory, enter this command
import os as o
cwd = o.getcwd()
print (cwd)
```

Line 5 uses the symbol (#) at the far left side of the line to denote a code comment, which is a handy way to provide code documentation. Like other languages you have seen in the workshops so far, comments can be placed in the code to provide instructions or greater detail to other users without compromising the code's ability to run. Line 5, in this case, provides instructions on what running lines 6-8 will do. Python will ignore everything after that # symbol for that line of code.

You don't always have to place a comment at the beginning of a line. For example, you could move everything on Line 5 to the end of Line 8 (like in the screenshot below), and there would be no difference in the output compared to running Lines 5-8 as in the screenshot above.

```
import os as o
cwd = o.getcwd()
print (cwd) # If you want to check your current working directory, enter this command
```

Next, **run Lines 5-8**, which, as advertised in Line 5, will help you identify your current working directory. This becomes more useful when working with input-output operations. To do so, first highlight Lines 5-8 in the code using your cursor, as below. Review the video to the first module if you need.

```
1  # -*- coding: utf-8 -*-
2  """
3  Variables, Assignment Statements
4  """
5  # If you want to check your current working directory, enter this command
6  import os as o
7  cwd = o.getcwd()
8  print (cwd)
```

If you need a refresher, though, once selected, either press F9 (on Windows), or simply click on the "Run Selection or Current Line icon" near the top of the screen:



Since the four lines of code that we want to run are already selected, this block – and only this block – will run.

When you do so, now view the output in the console at right. You should see something like this, and of course, the output location for you will look different than mine:

```
In [6]:
...: import os as o
...: cwd = o.getcwd()
...: print (cwd)
C:\Users\scottkelley\Dropbox\UNR_Service\PythonWorkshop\Online2020
```

A few things to notice here. First, each time a block of code is run, you'll see a unique increasing index value (e.g., 'In [6]: '). That means, prior to me running this block in this session of Spyder, I've run 5 operations where an output has been produced. If this is the first time you've run any code in your session, your "In" value will be 1.

Next, you'll note a number of things happening in the output. First is the 'import' command in Line 6, which allows access to code from another module that is not automatically accessible from a base working environment of Python (the 'os' module in this case), and then we define a name in the local namespace ('o', in this case). This helps to provide a shorthand way of accessing any of the properties and methods from the 'os' module for this script. In other words, anytime I want to access anything from this 'os' module, typing in "o." will do the trick, as evidenced in line 7.

Line 7 is also the first place where I set a variable, denoted by the '=' sign. **cwd**, which is a handy shorthand for "current working directory" is now initiated as a **variable**. As we'll see though, you can call this variable anything you want. 'c', 'workingdirectory', or 'ghhrwst' would all be valid variable names, for example. But as is good practice, choose variable names that make sense to you. 'cwd' makes sense to me in this case.

This variable **cwd** now stores the output of accessing the os module (shorthand of 'o' using the name in the local namespace grants that access), and applying the .getcwd() command from that module (which reports the current working directory). To get the report of that directory, I simply now initiate a print statement on Line 8. Accessing the 'print' **function**, which is a built-in function to any base Python installation and session, allows me to print out a message on the screen in the output at right. Note that I don't need an 'import' statement to get access to the print function, like I needed to do with os to get access to the getcwd() function.

In this case, my current working directory is reported at the end of the output block of text in the console at right: "C:\Users\scottkelley\Dropbox\UNR_Service\PythonWorkshop\Online2020"

*Question: if you were to run Line 12 as-is, what might go wrong? What two things would need to be corrected in order for this to work? *Hint, 'chdir()' will change the working directory*

4) Constants and Print Statements

Lines 26/27 and Lines 29/30 are each examples of 1) setting a variable as a **constant** and then 2) providing a **print** statement that reports the value of the variable in each case.

Review the code documentation (Lines 14-24) for more information. When done, **run Lines 26-30**. Either select and run the full group, or select and run Lines 26/27 and Lines 29/30 separately. Either approach is fine, just confirm that **my_var** is now set to the value '2', and **my_other_var** is set to the value 2.5.

Then, review the documentation on Lines 32-33 and **run Lines 35-36**. Notice that again the '=' sign designates a variable, and can be used to set constants that are either numeric data or string data.

At this point, you should have 5 variable values set: cwd, my_var, my_other_var, true, also true

To test that is the case, you can always enter variable names in the console input, like this, and get an immediate output return.

```
In [10]: my_var
Out[10]: 2
```

Next, review lines and **run 40 and 41** independently, which provide examples of combining data types in print statements, and examples of how to integrate proper punctuation and spacing.

5) Syntax errors, Part 1 (of many) in diagnosing and fixing them

Run line 42 as-is. You may have already noticed the red 'x' next to the line number in Spyder. That's actually a good thing: Spyder is warning us in advance that something is amiss. Still, let's pretend like we didn't notice it. When you run it, you'll notice in the console output a multi-colored message indicating we have an issue. There is some indication of where the issue is, too, noted by the '^' icon.

```
In [15]: print (alsotru, " " true) # Good editors will catch syntax errors like this for you
File "<ipython-input-15-b9a8e5640ae1>", line 1
    print (alsotru, " " true) # Good editors will catch syntax errors like this for you
              ^
SyntaxError: invalid syntax
```

Activity - Look at the outputs to Lines 40 and 41, and use what you see there to fix Line 42. Experiment with fixes to get the print statement to 1) work without error, and 2) also have proper punctuation and spacing.

6) Concatenation

Among other things, print statements are useful tools for helping diagnosing the state of our program at various points in the process. They are especially helpful for reporting a variable's value before or after that variable is used by some other piece of code.

Line 45 shows an example of doing so in a print statement. Notice the '+' operator, as opposed to the ",", operators on Lines 40-42. **Run Line 45** to see the output

Line 46 stores the output of a string concatenation into a variable. Run Line 46, then enter the variable x in the console (at right) to see the output, as below:

```
In [11]: x = true + ". " + alsotru
In [12]: x
Out[12]: 'This is the best way to spend a Saturday. This is the best way to spend a Saturday'
```

7) More on Setting Variables

Read the documentation on Lines 47-52 about variables. Then, **run lines 54 and 55**, noting that both the lower- and upper-case values of 'x' store different constants. **Run Line 57** to confirm, though we see it doesn't look clean without a comma separating those values. Let's add a comma to take care of that.

Run Line 59 (you'll see the warning about what's to come in Line 58). Since each variable is a numeric constant, the string notation in the statement causes some issues here, evidenced by the nature of the error message in the output.

Line 61 offers one solution to address the issue by converting both numeric variable values to string data. **Run Line 61**. Then, the concatenation operator in the statement works smoothly. **Run Line 62**, which – in contrast to 61 – stores this into a variable for potential later use instead of in a print statement.

One important thing about variable that becomes super useful with more complex looping operations: you can overwrite existing variable values. Line 65 sets 'x', then Lines 66 uses the stored value of x from Line 65 in order to update x in Line 66. **Run Line 65**, then enter x in the console to see its value. Repeat after **running Line 66**. **Run Line 67** and view the output to confirm.

Review Line 69 and **run Line 70**. Then enter separately a, b, then c in the console to see the output of each

This completes the structured part of Module 2. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.