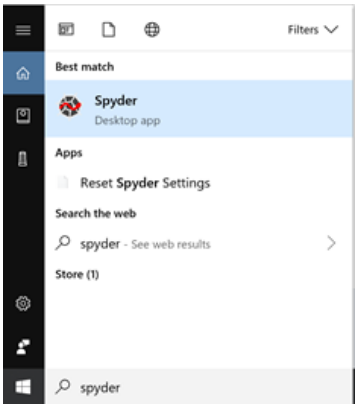
Introduction to Programming in Python through Immersion

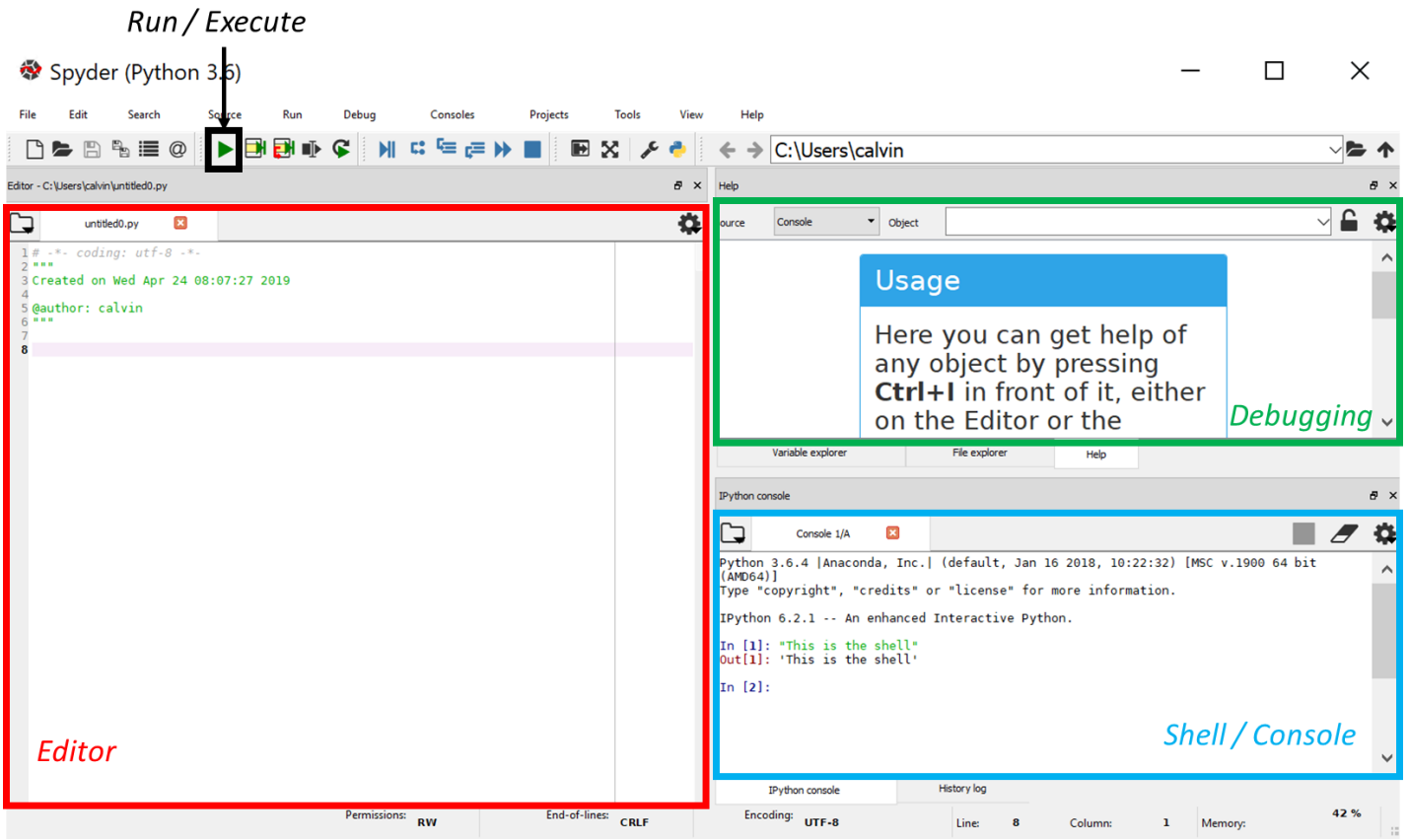
I. Background: Spyder IDE

We will use Spyder (Scientific Python Development Environment) as our Integrated Development Environment (IDE). The software is free for personal use and is distributed here. On lab machines, you can simply go to the search bar and type Spyder:



Shown below is an annotated image of the Spyder interface. There are three main sections to be aware of:

- **Editor.** The editor allows you to save code in dedicated, well-named files. This is where almost every line of code you write will be entered.
- **Debugging.** The debugging window will be useful later as we create more advanced code and need to isolate and identify bugs in our code. It offers a means of tracking variable values and other niceties.
- **Console.** When you execute code, the results will be emitted in the console window. **Shell.** *Python is an interpreted language* thus it allows direct interactivity with you, the programmer. Hence, you can enter an instruction into the shell and it will execute immediately.



II. Shell Operations

1. In the shell, type `2 ** 4` and `Enter`. (a) Report the result: _____

(b) Try a few other expressions with `**`. Based on the results, describe the function of the operator `**`.

2. Type `"Sm" + "ile!"` and `Enter`. (a) Report the result: _____

(b) Try a few other expressions with strings and `+`. Based on the results, describe the function of the operator `+` on two strings of characters in double quotes.

3. Type `'Fro' + 'wn.'` and `Enter`. Report the result: _____

4. Based on the answers to problems **2**(a) and **3**, does there seem to be a distinction between single quotes (`'`) and double quotes (`"`)? Explain.

5. Type `print("Hello")` and `Enter`. Report the output: _____

6. Type `print(3 * 4 - 5)` and `Enter`. Report the output: _____

7. Type `print("Result is: ", 3 * 4 - 5, ".")` and `Enter`. Report the output: _____

8. Write and execute a `print` statement that would output `"The total is 24."` where `24` should be computed with an expression (e.g., `2 * 12`, etc.) Report the print statement used; don't forget the period at the end.

III. Roots Code (Round I)

9. In this question, we will try (unsuccessfully) to run our first python program.

(a) From Moodle, download `roots-bad.py`.

(b) Open the file in Spyder; it should look something like this:

```
1 import math
2 def roots(fltA, fltB, fltC):
3 """Computes and prints the roots of a quadratic polynomial: ax^2 + bx + c
4 @input: fltA -- a in the expression ax^2 + bx + c
5 @input: fltB -- b in the expression ax^2 + bx + c
6 @input: fltC -- c in the expression ax^2 + bx + c
7 @output: No output; the result is printed
8 @Restrictions: None
9 """
10 # Compute the discriminant of the quadratic formula
11 fltDiscriminant = fltB * fltB - 4 * fltA * fltC
12 # Check if roots are imaginary: no roots
13 if fltDiscriminant < 0:
14 print ('None')
15 return
16 # One unique root
17 if fltDiscriminant == 0:
18 print ('x =', -fltB / (2 * fltA))
19 return
20 # Compute the two roots
21 fltX_1 = (-fltB + math.sqrt(fltDiscriminant)) / (2 * fltA)
22 fltX_2 = (-fltB - math.sqrt(fltDiscriminant)) / (2 * fltA)
23 # Report two roots
24 print ('x1 =', fltX_1)
25 print ('x2 =', fltX_2)
26 def test():
27 #
28 # Call the function with three input values: a, b, c
29 #
30 floatA = 1
31 floatB = -2
32 floatC = -3
33 print(str(floatA) + "x^2 + " + str(floatB) + "x + " + str(floatC) + ":")
34 roots(floatA, floatB, floatC)
35 print()
36 floatA = 1
37 floatB = 0
38 floatC = 0
39 print(str(floatA) + "x^2 + " + str(floatB) + "x + " + str(floatC) + ":")
40 roots(floatA, floatB, floatC)
41 # Invokes the testing function
42 if __name__ == "__main__":
43 test()
```

(c) Execute the Python file by selecting (click the play button).

(d) Report any error message(s). _____

(e) On what line did the error occur? _____

Notice how on that line, the IDE gives an indication about where an error will occur (without executing the program):

IV. Roots Code (Round II)

10. We should try that again.

(a) From Moodle, download `roots-good-v1.py`.

(b) Open the file in Spyder; it should look something like this:

```
import math

def roots(fltA, fltB, fltC):
    """Computes and prints the roots of a quadratic polynomial: ax^2 + bx + c
       @input: fltA -- a in the expression ax^2 + bx + c
       @input: fltB -- b in the expression ax^2 + bx + c
       @input: fltC -- c in the expression ax^2 + bx + c
       @output: No output; the result is printed
       @Restrictions: None
    """

    # Compute the discriminant of the quadratic formula
    fltDiscriminant = fltB * fltB - 4 * fltA * fltC

    # Check if roots are imaginary: no roots
    if fltDiscriminant < 0:
        print ('None')
        return

    # One unique root
    if fltDiscriminant == 0:
        print ('x =', -fltB / (2 * fltA))
        return

    # Compute the two roots
    fltX_1 = (-fltB + math.sqrt(fltDiscriminant)) / (2 * fltA)
    fltX_2 = (-fltB - math.sqrt(fltDiscriminant)) / (2 * fltA)

    # Report two roots
    print ('x1 =', fltX_1)
    print ('x2 =', fltX_2)


def test():
    #
    # Call the function with three input values: a, b, c
    #
    floatA = 1
    floatB = -2
    floatC = -3
    print(str(floatA) + "x^2 + " + str(floatB) + "x + " + str(floatC) + ":")
    roots(floatA, floatB, floatC)

    print()

    floatA = 1
    floatB = 0
    floatC = 0
    print(str(floatA) + "x^2 + " + str(floatB) + "x + " + str(floatC) + ":")
    roots(floatA, floatB, floatC)

# Invokes the testing function
if __name__ == "__main__":
    test()
```

(c) Execute this Python file. Record the output:

(d) State the main difference between the two sets of code from `roots-bad.py` and `roots-good-v1.py`.

We will come to realize that indentation is critical in the Python programming language.

The remaining questions in this section refer to the code in `roots-good-v1.py`. ***Follow the instructions carefully*** as some of the questions will ask you to modify the code (and possibly return the code to its original form).

11. In general, what does this code do? Describe how you know with at least three reasons.

Comments

12. Several lines (12, 15, 20, 25, etc.) begin with `#` (Spyder uses light-grayed text). Remove one of the `#` and execute the code. Report the error that results. Return the code to functioning properly.

13. Based on the results of the previous question, is the information after each `#` used or ignored when the program is executed? If the information is used, how do you know?

14. The text that appears after each `#` seems to be English-like information. We refer to these items as *comments*. Give at least two reasons for why we might add comments to code (lines beginning with `#`) to source code.

15. Lines 4 through 10 are highlighted in green in Spyder. Remove one `"` and execute the code. Report the error that results. Return the code to functioning properly.

16. Is the information between the `"""` used or ignored when the program is executed? If the information is used, how do you know?

17. The text that appears between the `"""` seems to be another set of English-like information. This is a special type of comment referred to as a ***docstring*** comment. It is used to describe various elements of a function. State four (4) types of information that a docstring comment provides. Please attempt to answer this question in general: your answer should not be specific to the `roots` function.

18. Docstring comments are, admittedly, tedious to write and contain some redundant information, but are very important to communicating the exact parameters of a function. Rerun the code to ensure that it executes properly. Then, in the shell, enter `help(roots)`. Do not copy the response to this command, but instead describe the response.

19. Enter `help(abs)`. Report the docstring comment for this function.

20. Do you feel that the docstring comment for `abs` is meaningful? That is, you are able to describe expected input and output as well as purpose of the function.

Whitespace

21. Remove the empty lines 11 and 14. Execute. Do the blank lines (vertical whitespace) matter in the Python programming language?

Yes / No

22. Give reason(s) for why we might add vertical whitespace to source code.

23. We have seen previously that indentation plays a critical role in properly formed Python code. However, does horizontal whitespace between characters matter? On line 26, remove the space around the operations. The line should now look like:

```
fltX_1=(-fltB+math.sqrt(fltDiscriminant))/(2*fltA)
```

Execute the code. Is there any change? Yes / No

24. Give reason(s) why we might add horizontal whitespace around expressions.

Modules

25. On line 26, change `math.sqrt` to `sqrt`. Execute. Record the error. Return the code to functioning properly.

26. With one small change, our Python code is now attempting to use a function (`sqrt`) for which it cannot find its definition. Speculate where the definition of `sqrt` is found.

27. There are some functions that are predefined, but do not require an import statement or special reference to a module like `math`. One is example is `abs`. Give another example of such a function that does not require a module to use. Hint: review page 2 of this lab.

Changing and Interpreting Code

28. Line 42 refers to using `roots` with 3 integer values: `(1, -2, -3)`. Based on your understanding of the code described in question 11, what is the value of `fltA`, `fltB`, and `fltC` on Line 3.

`fltA`: _____ `fltB`: _____ `fltC`: _____

29. Change the values on Lines 38 through 40 from `(1, -2, -3)` to `(1, 2, 1)`. Execute. Report the output. State the values of `fltA`, `fltB`, and `fltC` on Line 3.

Output: `fltA`: _____ `fltB`: _____ `fltC`: _____

30. Change the values on Lines 38 through 40 to guarantee the output will be `None`. Execute. Report the output as well as the values of `fltA`, `fltB`, `fltC`, and `fltDiscriminant`.

`fltA`: _____ `fltB`: _____ `fltC`: _____ `fltDiscriminant`: _____

31. On Lines 17, 22, 30, and 31 there are print statements in `roots`. We wish to add a print statement in `test` that describes the output. That is, for `(1, -2, -3)`, we wish to have the following output:

```
The solution(s) for 1x^2 + -2x + -3:
x1 = 3.0
x2 = -1.0
```

Modify the print statement on Line 41 to add this text. Report what you added.

32. Line 44 contains a blank print statement. What is its effect? That is, what does it print?

33. In the shell, enter `print("||")`. Report the output: _____

34. In the shell, enter `print("|\n|")`. Report the output:

35. Speculate on what `\n` represents?

36. In the shell, enter `print("|\t|")`. Report the output: _____

37. Speculate on what `\t` represents?

Control Flow and Selection Structures

38. The source code contains two `if`-statements on Lines 16 and 21. Each `if` is accompanied with a question (of sorts) we call a *condition*. State the conditions.

Line 16 Condition: _____ Line 21 Condition: _____

39. Change Lines 46 through 48 use `(1, 0, 1)`. State the expected output from lines 49 and 50; be honest. State the actual output.

Expected Output: Actual Output:

40. In the previous question, the code *selectively* printed `None`.

(a) Describe when `None` is printed referring to the value of the variable `fltDiscriminant`.

(b) Consider the purpose of this source code. Independent of the code, what does it say about a quadratic polynomial when `None` is printed by the `roots` function?

41. Keep Lines 46 through 48 As `(1, 0, 1)`. However, comment out Line 18. State the expected output; be honest. State the actual output or error.

Expected Output: Actual Output:

42. Take a look at the output in the Shell. What Line did the error from the previous question occur on? What caused the error?

43. The error was a result of omitting a `return` statement. Speculate on the impact of the `return` statement. What happens when a `return` is executed?

44. Give a reason for the code not having a `return` statement on Line 31.

The remaining questions refer to the code in file `roots-good-v2.py`. Consider a different version of the code that is functionally equivalent to the code; only the roots function is shown below.

```
1  import math
2
3  def roots(fltA, fltB, fltC):
4      """Computes and prints the roots of a quadratic polynomial: ax^2 + bx + c
5         @input: fltA -- a in the expression ax^2 + bx + c
6         @input: fltB -- b in the expression ax^2 + bx + c
7         @input: fltC -- c in the expression ax^2 + bx + c
8         @output: No output; the result is printed
9         @Restrictions: None
10     """
11
12     # Compute the discriminant of the quadratic formula
13     fltDiscriminant = fltB * fltB - 4 * fltA * fltC
14
15     # Check if roots are imaginary: no roots
16     if fltDiscriminant < 0:
17         print('None')
18
19     # One unique root
20     elif fltDiscriminant == 0:
21         print('x =', -fltB / (2 * fltA))
22
23     # Compute the two roots
24     else:
25         fltX_1 = (-fltB + math.sqrt(fltDiscriminant)) / (2 * fltA)
26         fltX_2 = (-fltB - math.sqrt(fltDiscriminant)) / (2 * fltA)
27
28         # Report two roots
29         print('x1 =', fltX_1)
30         print('x2 =', fltX_2)
```

45. State at least two differences between the code in `roots-good-v1.py` and the code in `roots-good-v2.py`.

(a)

(b)

46. We previously investigated the impact of the `if` keyword selecting specific lines to execute. If a condition is not satisfied we do not execute those (indented) statements. Line 20 contains the keyword `elif` (short for else if). Make a conjecture about the meaning of `elif` to the lines of code executed (we call this the flow of control, or simply, control flow). That is, under what conditions are the 'elif' statements executed?

47. Line 24 contains the keyword `else`. Make a conjecture about the meaning of `else` with respect to the flow of control. That is, under what conditions are the 'else' statements executed?

48. We previously investigated the impact of `return` and how the `if` keyword selects specific lines of code to execute. The updated code does not contain any `return` statements. Explain why.

49. In your opinion, which version of the `roots` code was easier to understand? Explain briefly.