**MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE REPUBLIC OF KAZAKHSTAN**

****

**International Information Technology University**

**Department of Cybersecurity**

**PROGRAMMING LANGUAGE PYTHON**

|  |  |
| --- | --- |
| **Student** | Nurlan Inkar |
|  | *(given name, surname) (фамилия, имя, отчество)* |
| **Group** | SIS-2214 |
| **Lecturer** | Nessibeli Askarbekova |
| **Аssessment** |  |

Almaty, 2024

**Week 2**

**Module 2. Python Data Types, Variables, Operators, and Basic I/O Operations**

* 2.1.5 LAB Working with the print() function

The print() command, which is one of the easiest directives in Python, simply prints out a line to the screen.

In your first lab:

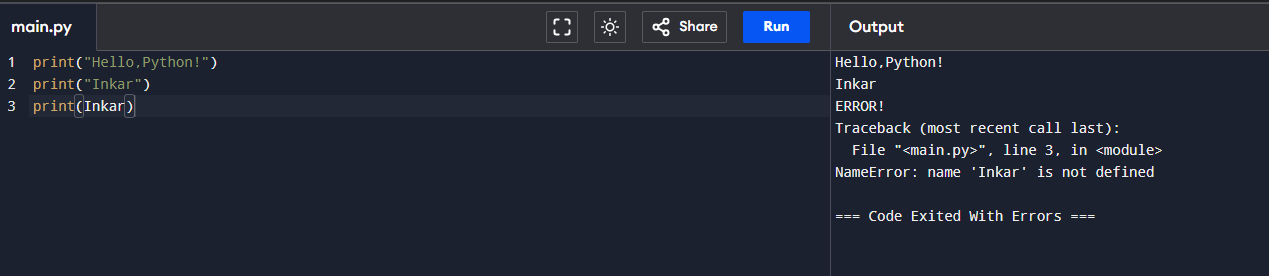
Use the print() function to print the line Hello, Python! to the screen. Use double quotes around the string.

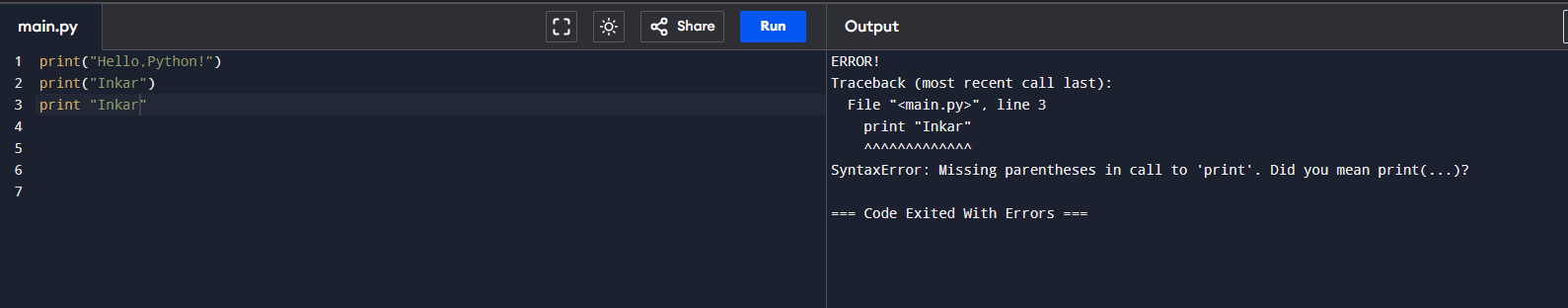
Having done that, use the print() function again, but this time print your first name.

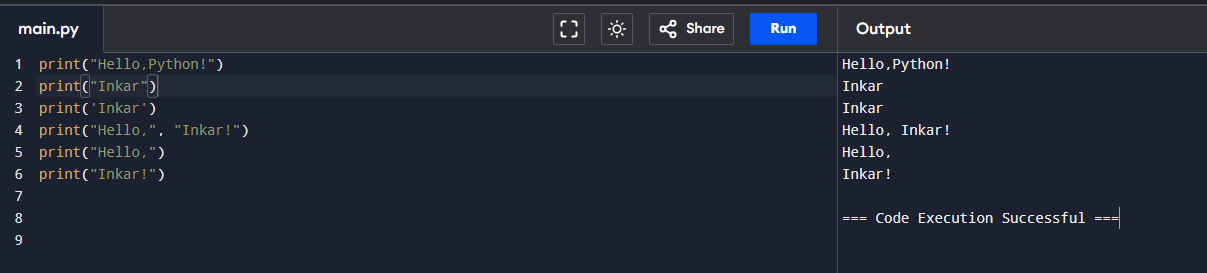
Remove the double quotes and run your code. Watch Python's reaction. What kind of error is thrown?

Then, remove the parentheses, put back the double quotes, and run your code again. What kind of error is thrown this time?

Experiment as much as you can. Change double quotes to single quotes, use multiple print() functions on the same line, and then on different lines. See what happens.







* 2.1.12 LAB The print() function and its arguments

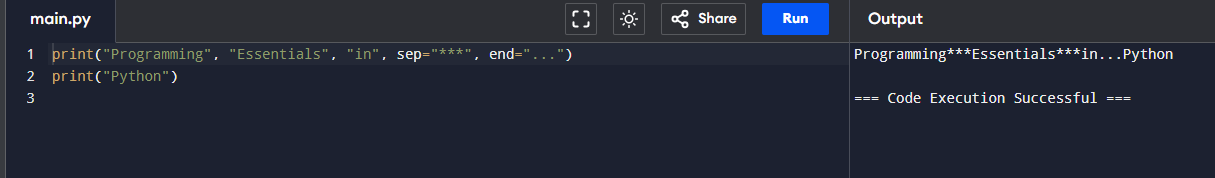
Scenario

Modify the first line of code in the editor, using the sep and end keywords, to match the expected output. Use the two print() functions in the editor.

Don't change anything in the second print() invocation.

Expected output

Programming\*\*\*Essentials\*\*\*in...Python



* 2.1.13 LAB Formatting the output

Scenario

We strongly encourage you to play with the code we've written for you, and make some (maybe even destructive) amendments. Feel free to modify any part of the code, but there is one condition ‒ learn from your mistakes and draw your own conclusions.

Try to:

minimize the number of print() function invocations by inserting the \n sequence into the strings;

make the arrow twice as large (but keep the proportions)

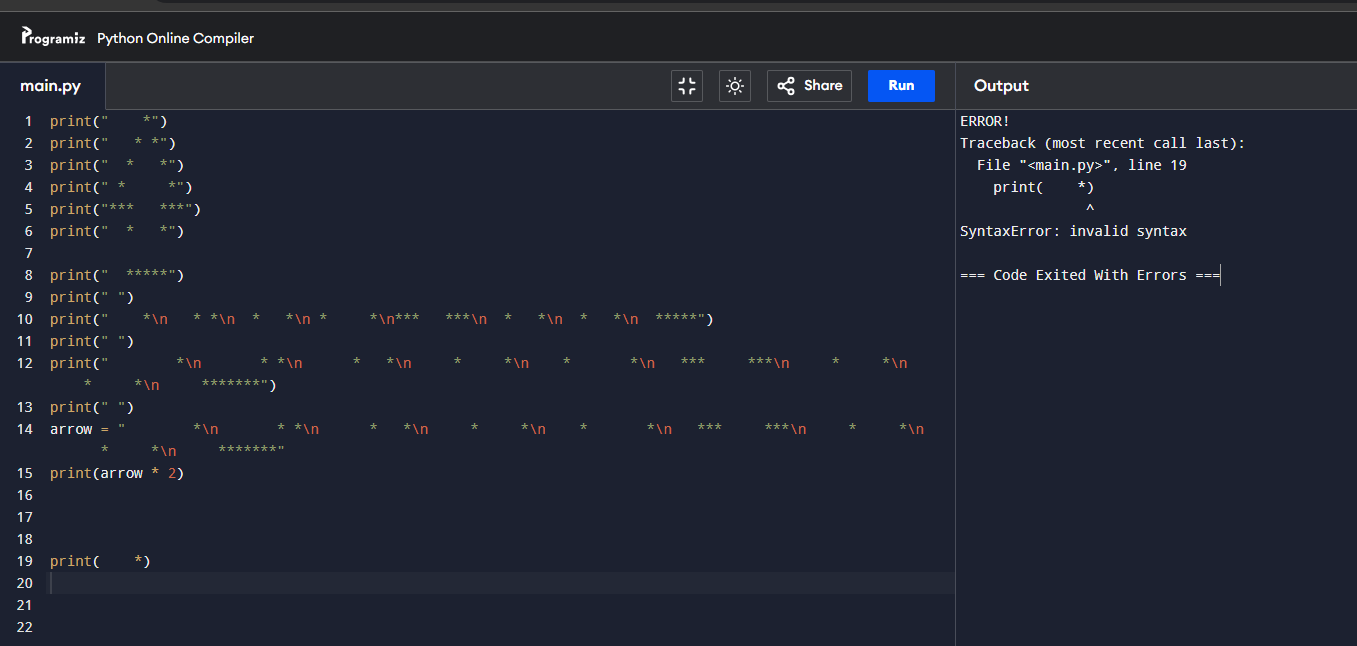
duplicate the arrow, placing both arrows side by side; note: a string may be multiplied by using the following trick: "string" \* 2 will produce "stringstring" (we'll tell you more about it soon)

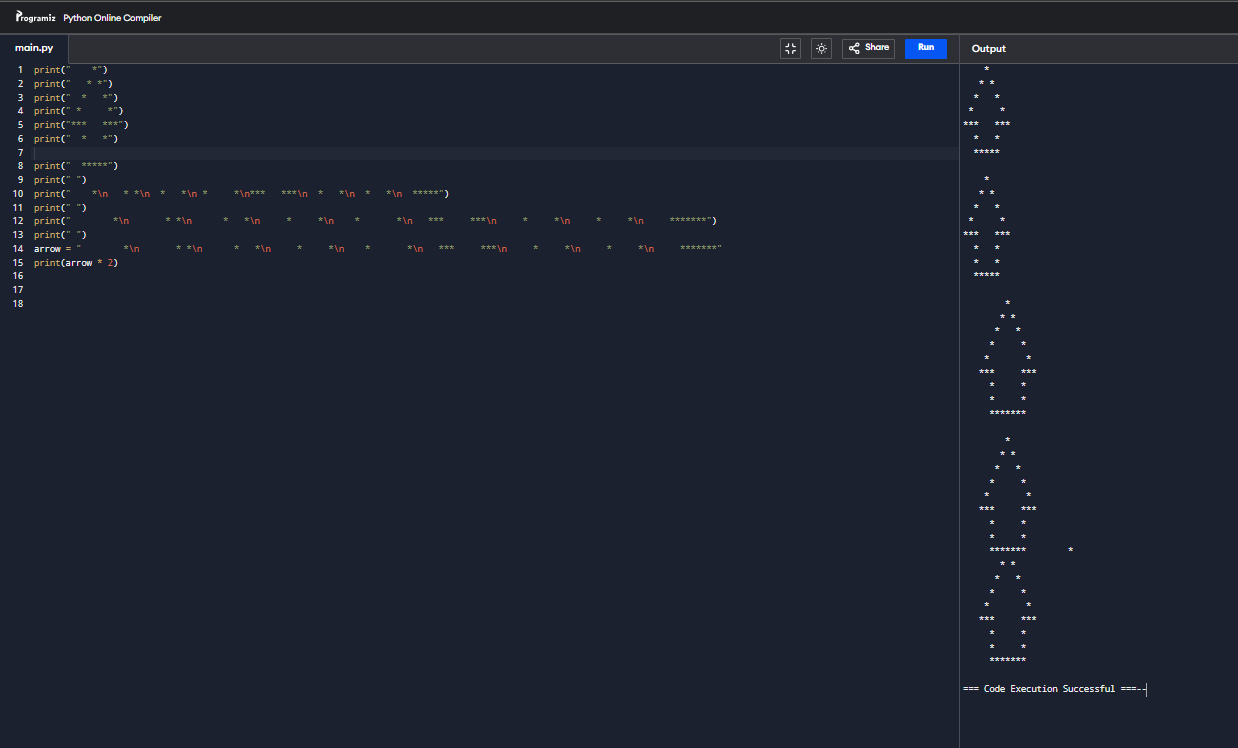
remove any of the quotes, and look carefully at Python's response; pay attention to where Python sees an error ‒ is this the place where the error really exists?

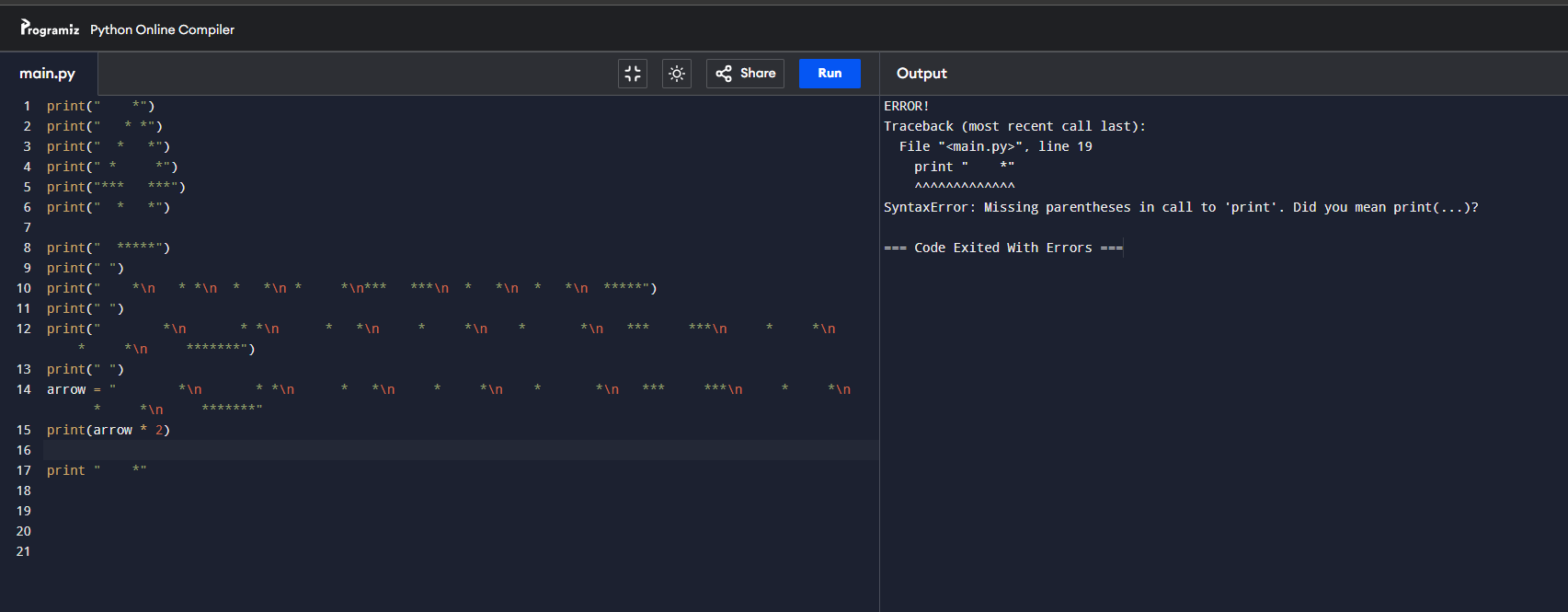
do the same with some of the parentheses;

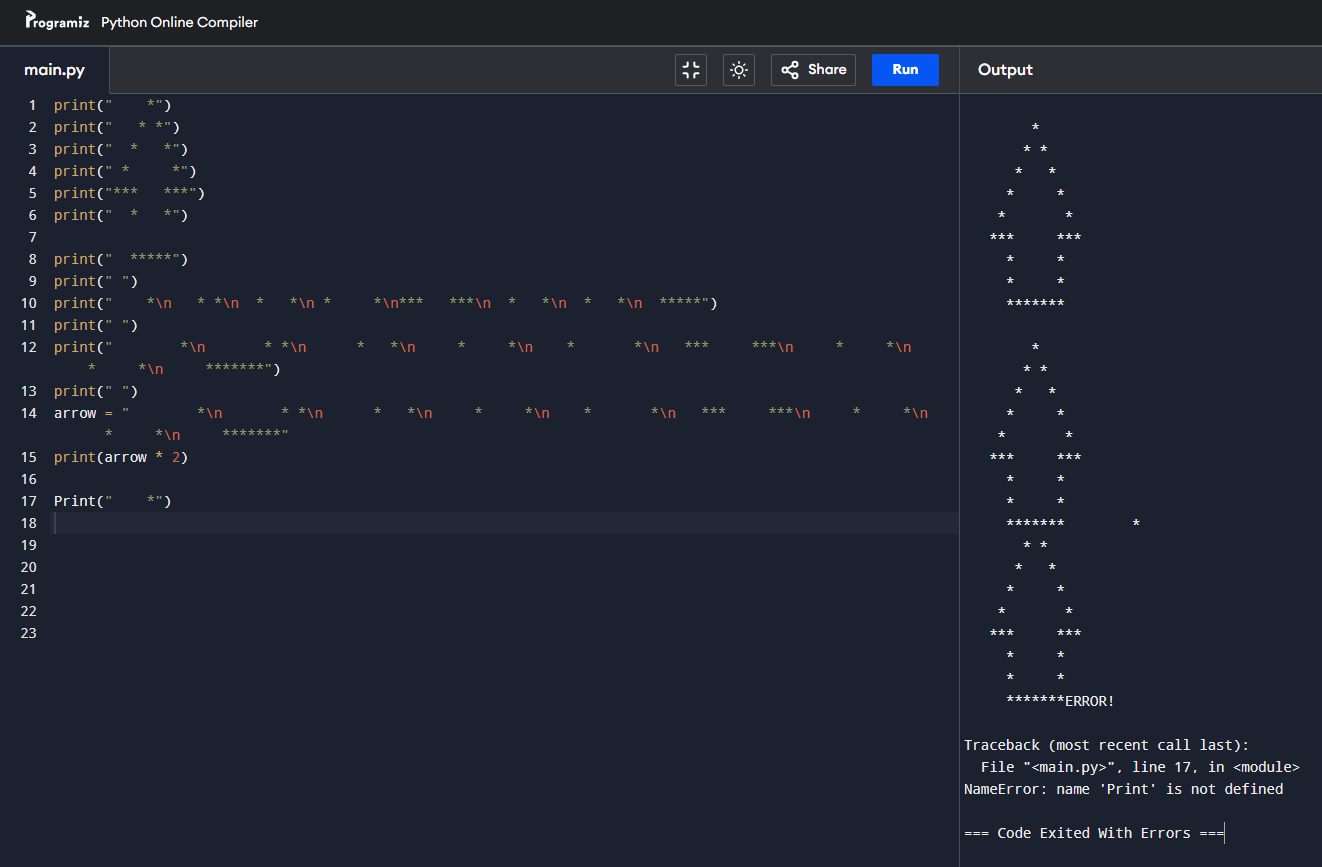
change any of the print words into something else, differing only in case (e.g., Print) ‒ what happens now?

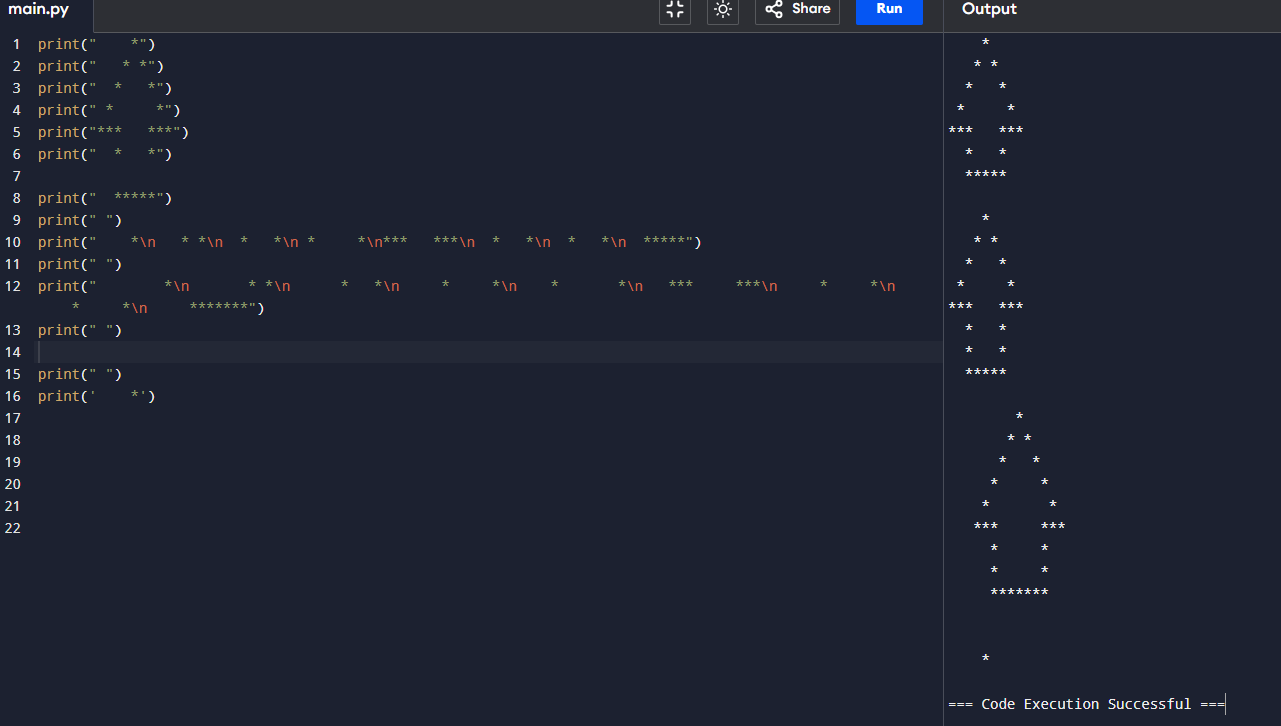
replace some of the quotes with apostrophes; watch what happens carefully.











* 2.2.6 LAB Python literals – strings

Scenario

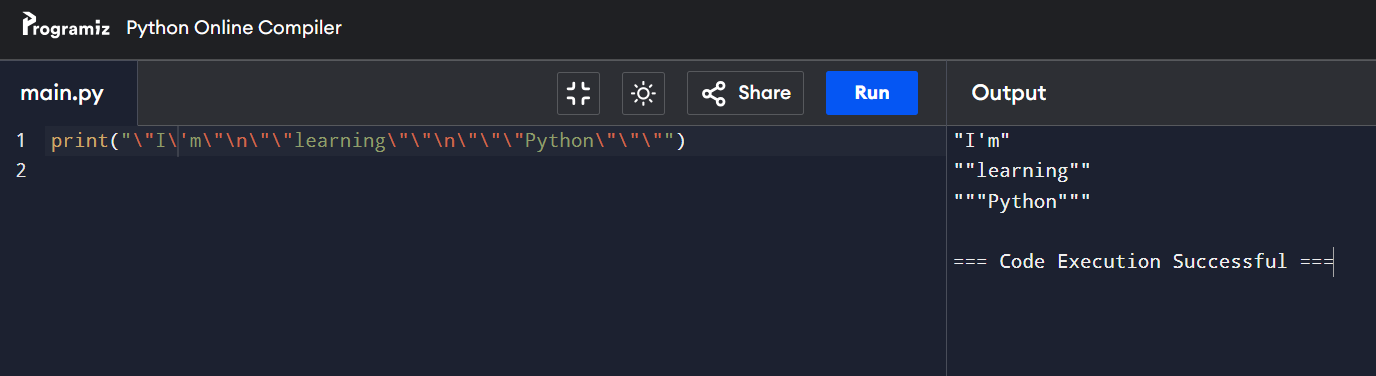
Write a one-line piece of code, using the print() function, as well as the newline and escape characters, to match the expected result outputted on three lines.

Expected output

"I'm"

""learning""

"""Python"""



* 2.4.7 LAB Variables

Scenario

Here is a short story:

Once upon a time in Appleland, John had three apples, Mary had five apples, and Adam had six apples. They were all very happy and lived for a long time. End of story.

Your task is to:

create the variables: john, mary, and adam;

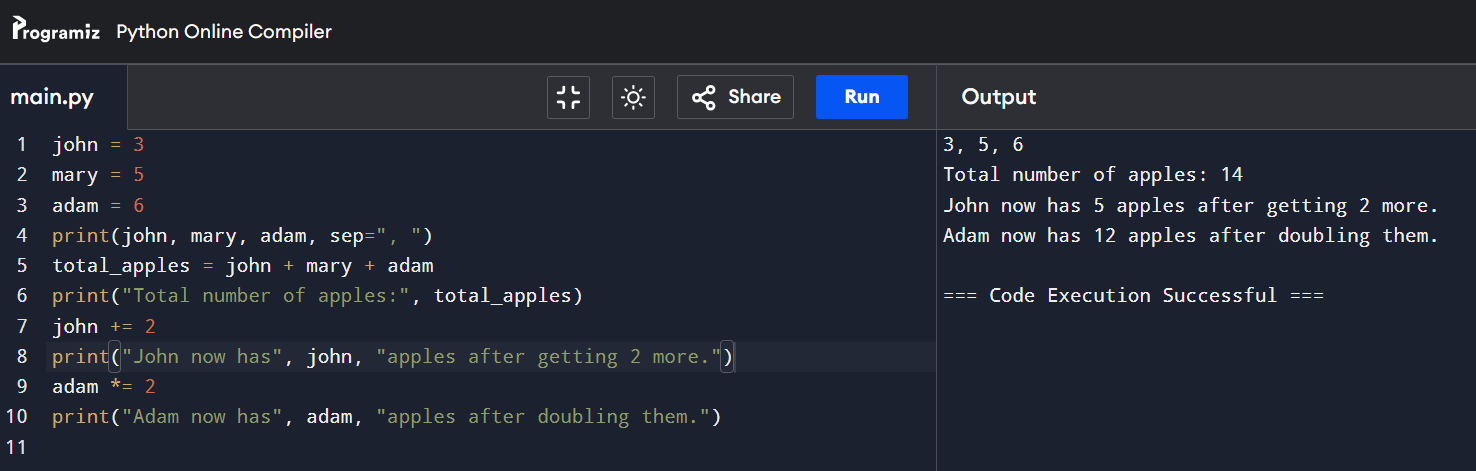
assign values to the variables. The values must be equal to the numbers of fruit possessed by John, Mary, and Adam respectively;

having stored the numbers in the variables, print the variables on one line, and separate each of them with a comma;

now create a new variable named total\_apples equal to the addition of the three previous variables.

print the value stored in total\_apples to the console;

experiment with your code: create new variables, assign different values to them, and perform various arithmetic operations on them (e.g., +, -, \*, /, //, etc.). Try to print a string and an integer together on one line, e.g., "Total number of apples:" and total\_apples.



* 2.4.9 LAB Variables ‒ a simple converter

Scenario

Miles and kilometers are units of length or distance.

Bearing in mind that 1 mile is equal to approximately 1.61 kilometers, complete the program in the editor so that it converts:

miles to kilometers;

kilometers to miles.

Do not change anything in the existing code. Write your code in the places indicated by ###. Test your program with the data we've provided in the source code.

Pay particular attention to what is going on inside the print() function. Analyze how we provide multiple arguments to the function, and how we output the expected data.

Note that some of the arguments inside the print() function are strings (e.g., "miles is", whereas some other are variables (e.g., miles).

Tip

There's one more interesting thing happening there. Can you see another function inside the print() function? It's the round() function. Its job is to round the outputted result to the number of decimal places specified in the parentheses, and return a float (inside the round() function you can find the variable name, a comma, and the number of decimal places we're aiming for). We're going to talk about functions very soon, so don't worry that everything may not be fully clear yet. We just want to spark your curiosity.

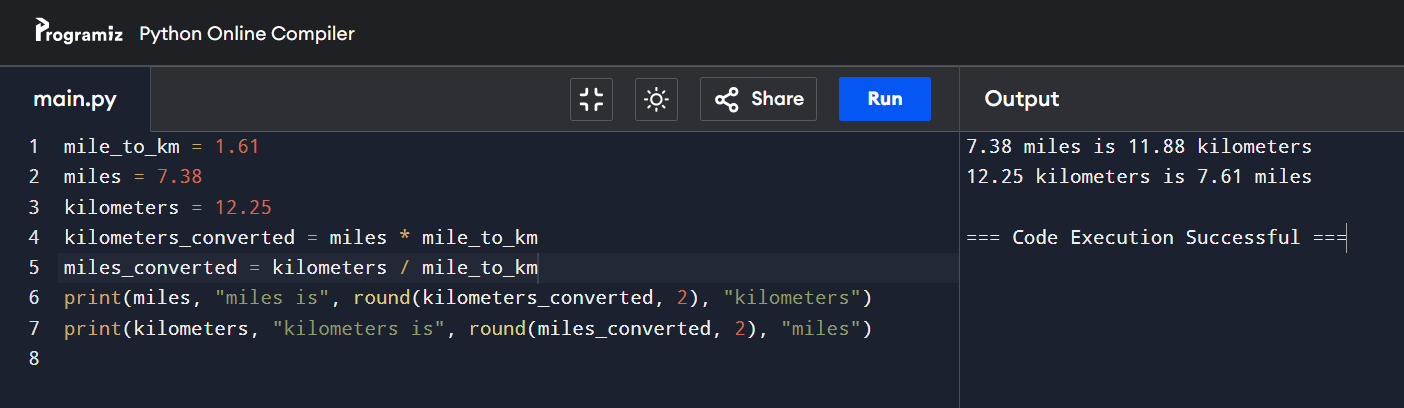
After completing the lab, open the Sandbox, and experiment some more. Try to write different converters, e.g., a USD to EUR converter, a temperature converter, etc. ‒ let your imagination fly! Try to output the results by combining strings and variables. Try to use and experiment with the round() function to round your results to one, two, or three decimal places. Check out what happens if you don't provide any number of digits. Remember to test your programs.

Experiment, draw conclusions, and learn. Be curious.

Expected output

7.38 miles is 11.88 kilometers

12.25 kilometers is 7.61 miles



* 2.4.10 LAB Operators and expressions

Scenario

Take a look at the code in the editor: it reads a float value, puts it into a variable named x, and prints the value of a variable named y. Your task is to complete the code in order to evaluate the following expression:

3x3 - 2x2 + 3x - 1

The result should be assigned to y.

Remember that classical algebraic notation likes to omit the multiplication operator ‒ you need to use it explicitly. Note how we change data type to make sure that x is of type float.

Keep your code clean and readable, and test it using the data we've provided, each time assigning it to the x variable (by hardcoding it). Don't be discouraged by any initial failures. Be persistent and inquisitive.

Sample input

x = 0

x = 1

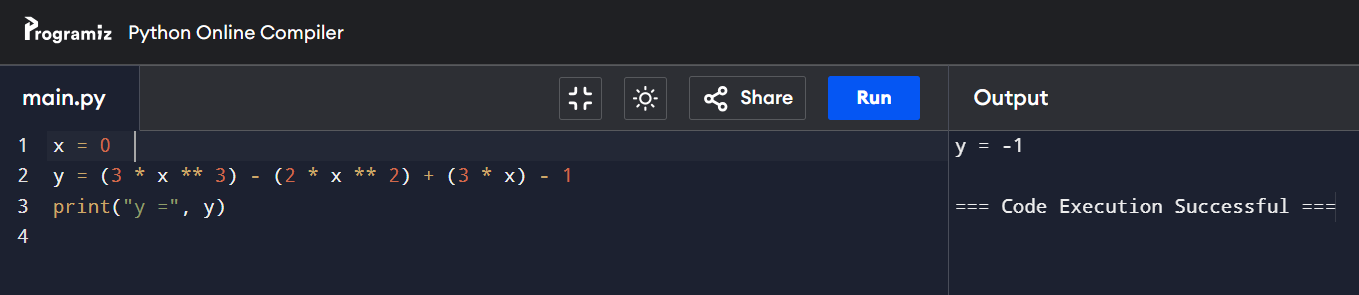
x = -1

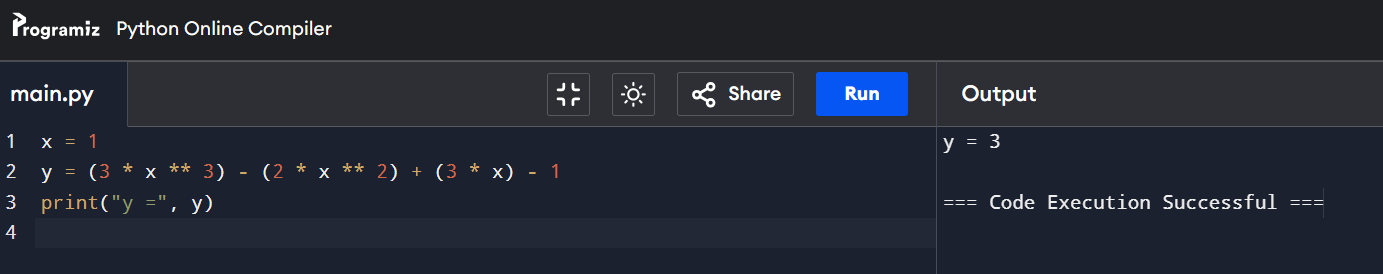
Sample output

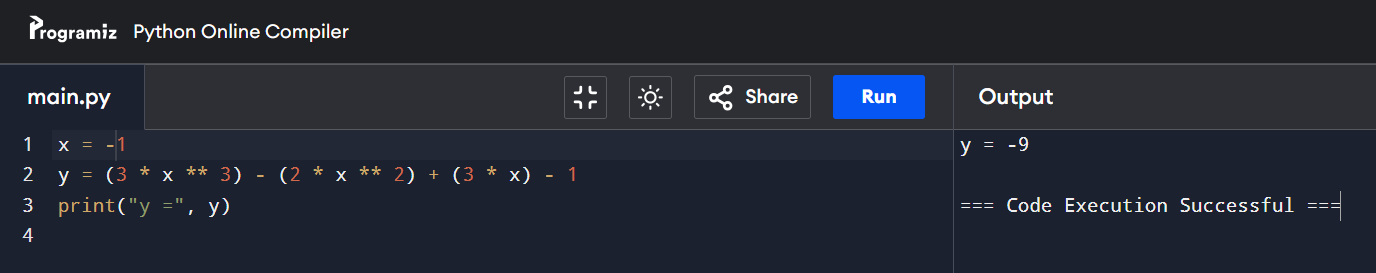
y = -1.0

y = 3.0

y = -9.0







* 2.5.3 LAB Comments

Scenario

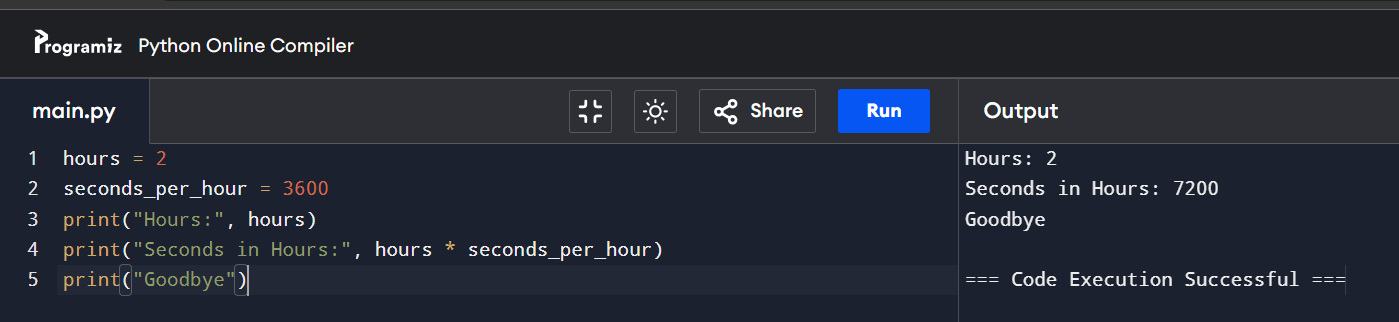
The code in the editor contains comments. Try to improve it: add or remove comments where you find it appropriate (yes, sometimes removing a comment can make the code more readable), and change variable names where you think this will improve code comprehension.

Note

Comments are very important. They are used not only to make your programs easier to understand, but also to disable those pieces of code that are currently not needed (e.g., when you need to test some parts of your code only, and ignore others). Good programmers describe each important piece of code, and give self-commenting names to variables, as sometimes it is simply much better to leave information in the code.

It's good to use readable variable names, and sometimes it's better to divide your code into named pieces (e.g., functions). In some situations, it's a good idea to write the steps of computations in a clearer way.

One more thing: it may happen that a comment contains a wrong or incorrect piece of information ‒ you should never do that on purpose!



* 2.6.9 LAB Simple input and output

Scenario

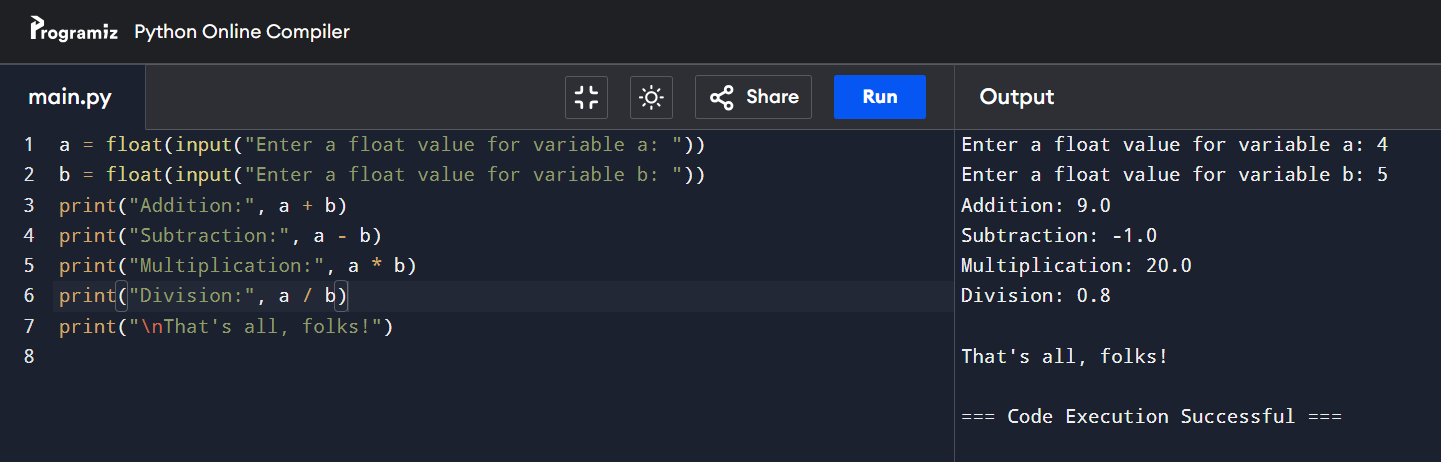
Your task is to complete the code in order to evaluate the results of four basic arithmetic operations.

The results have to be printed to the console.

You may not be able to protect the code from a user who wants to divide by zero. That's okay, don't worry about it for now.

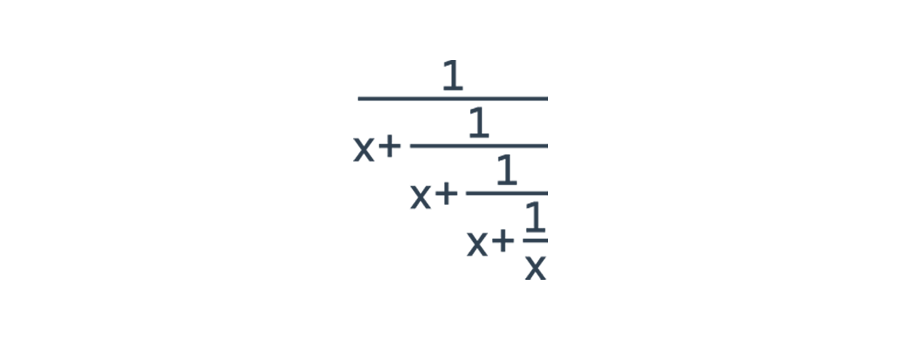
Test your code ‒ does it produce the results you expect?

We won't show you any test data ‒ that would be too simple.



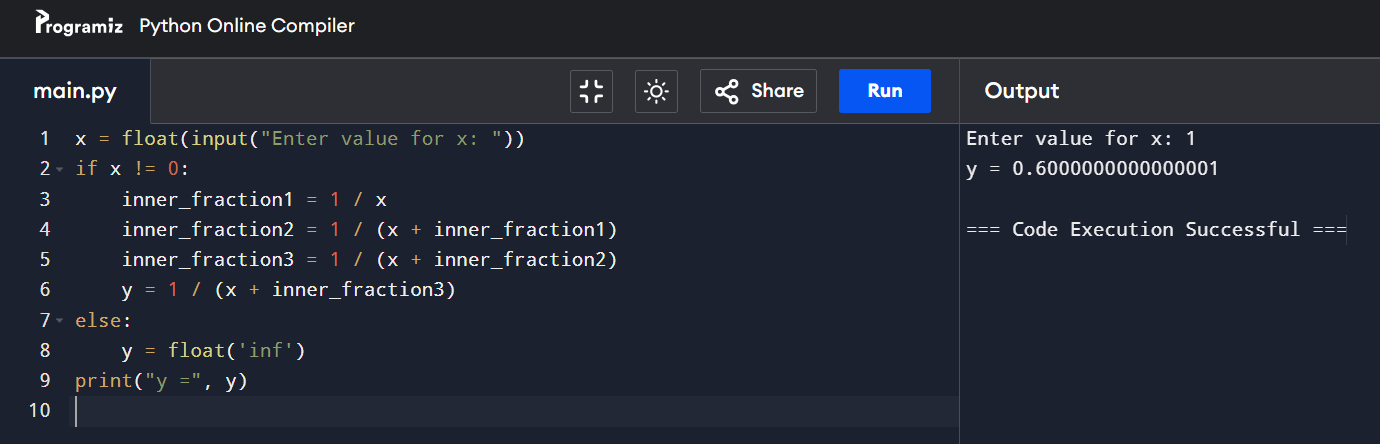
* 2.6.10 LAB Operators and expressions

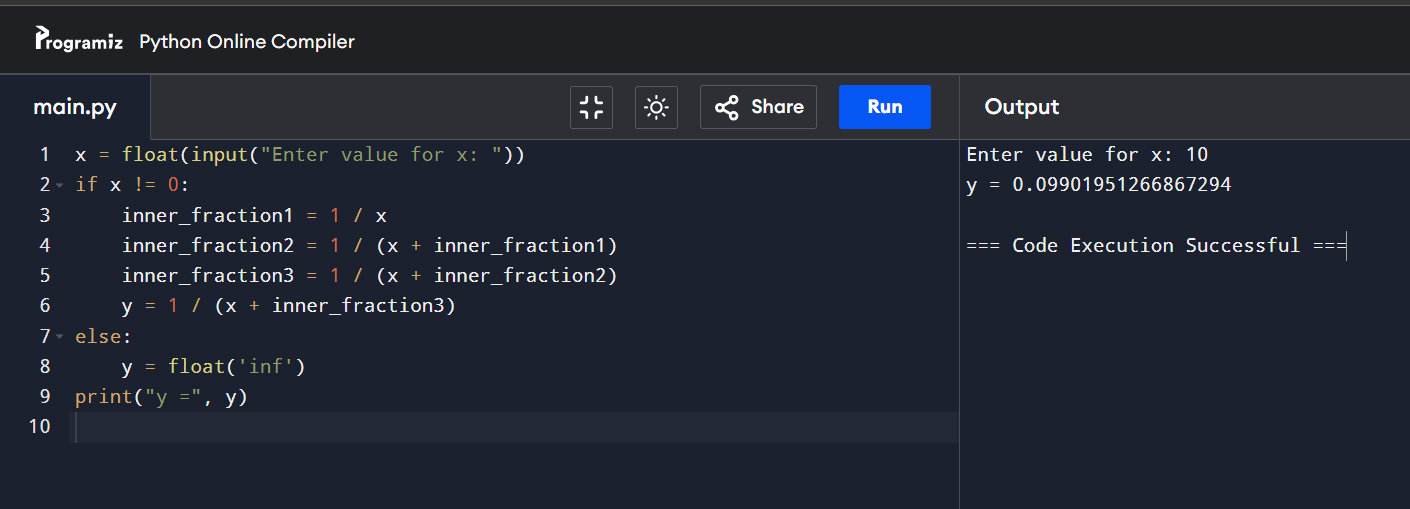
Your task is to complete the code in order to evaluate the following expression:

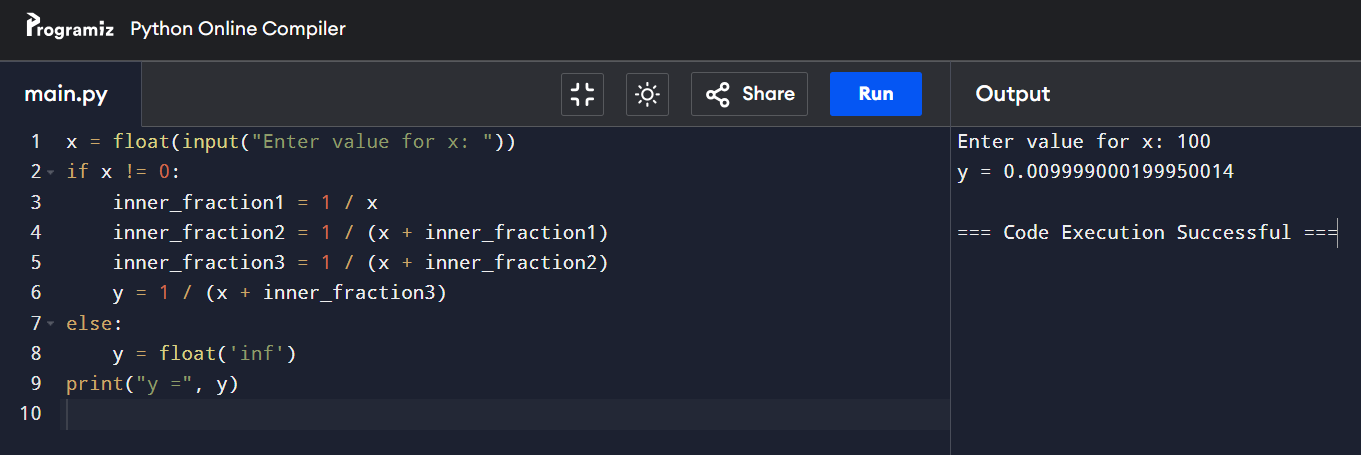


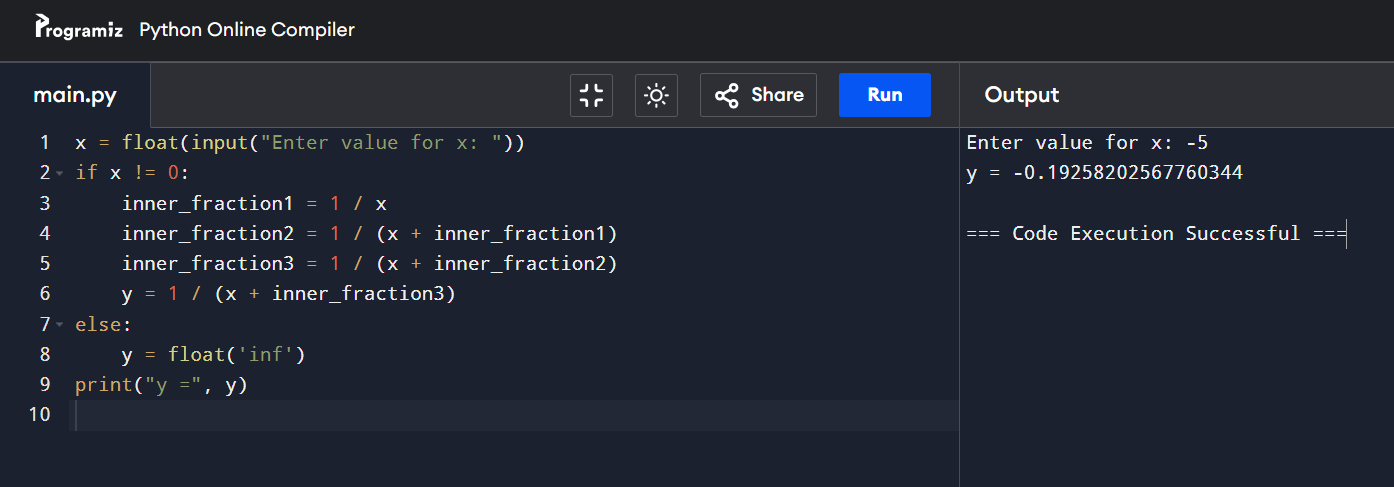
The result should be assigned to y. Be careful ‒ watch the operators and keep their priorities in mind. Don't hesitate to use as many parentheses as you need.

You can use additional variables to shorten the expression (but it's not necessary). Test your code carefully.









* 2.6.11 LAB Operators and expressions – 2

Scenario

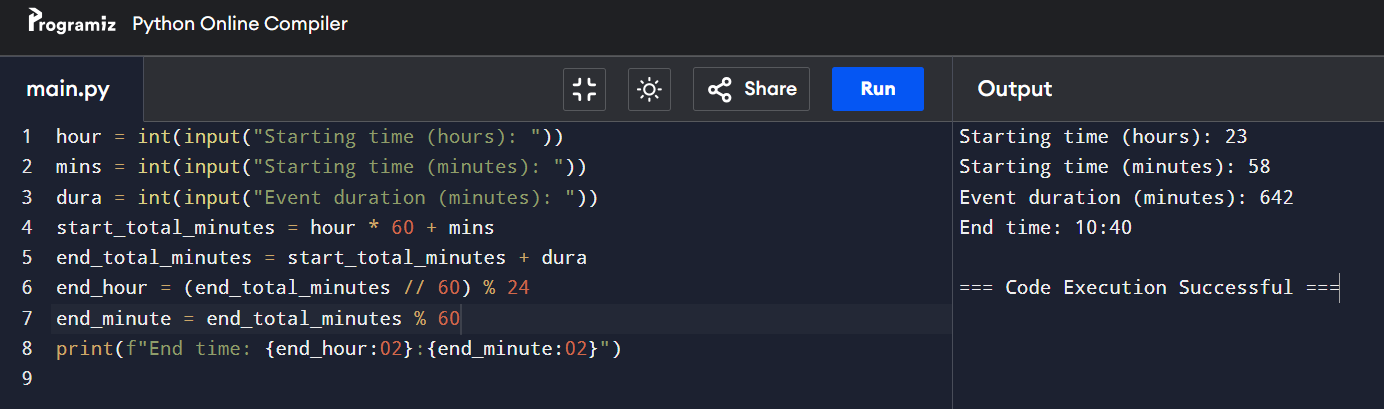
Your task is to prepare a simple code able to evaluate the end time of a period of time, given as a number of minutes (it could be arbitrarily large). The start time is given as a pair of hours (0..23) and minutes (0..59). The result has to be printed to the console.

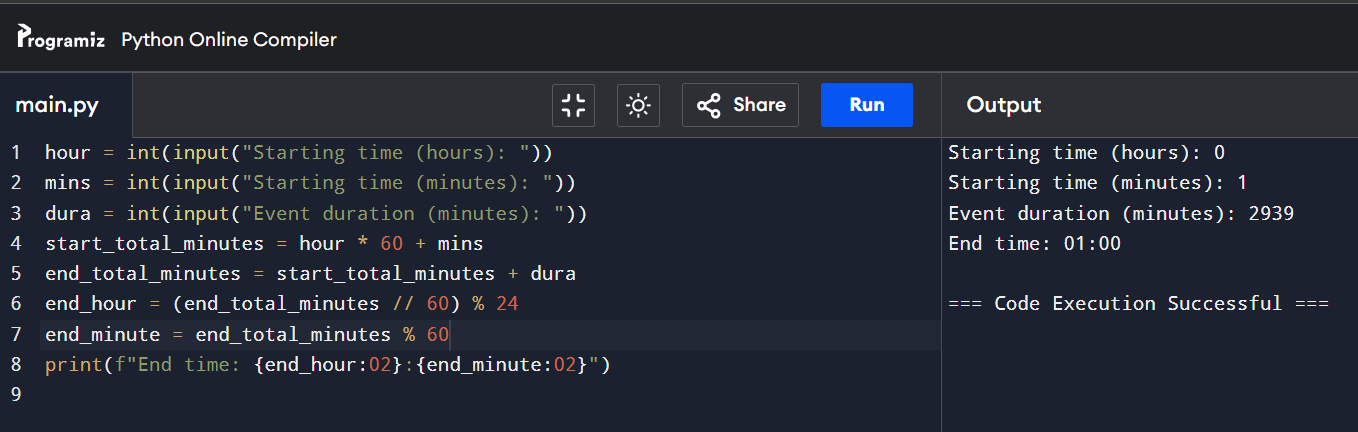
For example, if an event starts at 12:17 and lasts 59 minutes, it will end at 13:16.

Don't worry about any imperfections in your code ‒ it's okay if it accepts an invalid time ‒ the most important thing is that the code produces valid results for valid input data.

Test your code carefully. Hint: using the % operator may be the key to success.







# **CONCLUSION**

The tasks helped me practice writing and improving Python code for basic operations and time calculations while focusing on clear comments and variable names. I learned how to handle errors and keep my code well-organized. By doing these exercises, I improved my ability to write clean and accurate code that works in different situations.