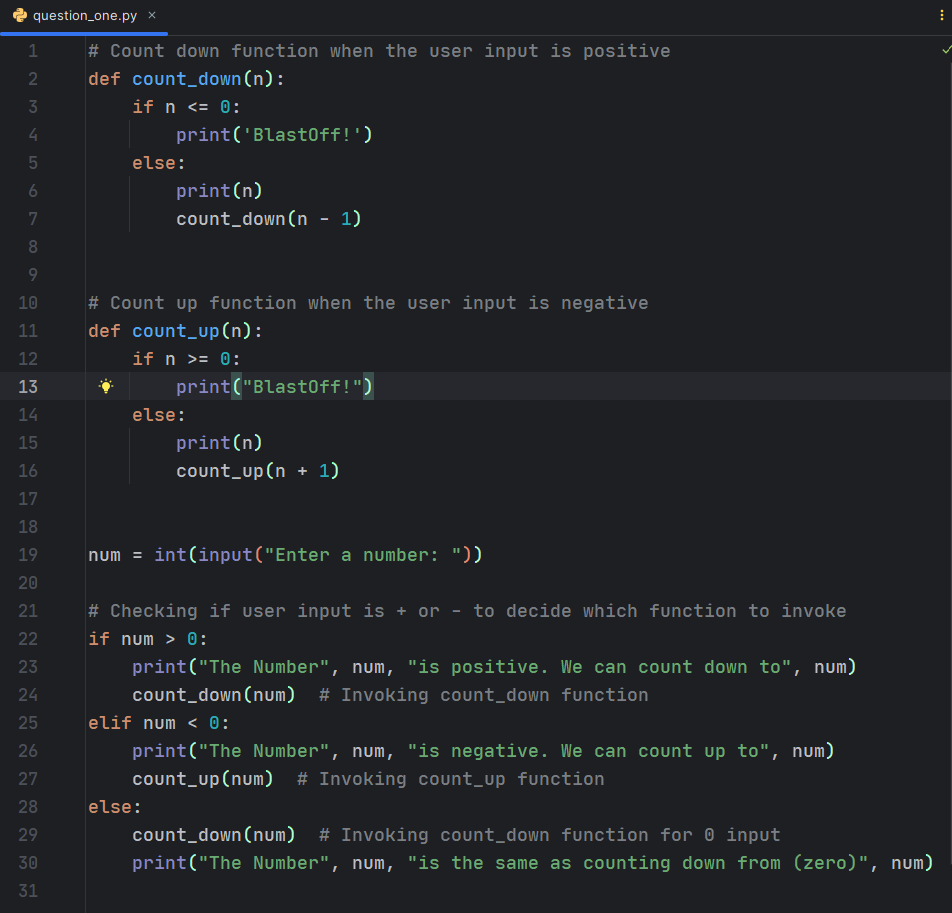
Q1

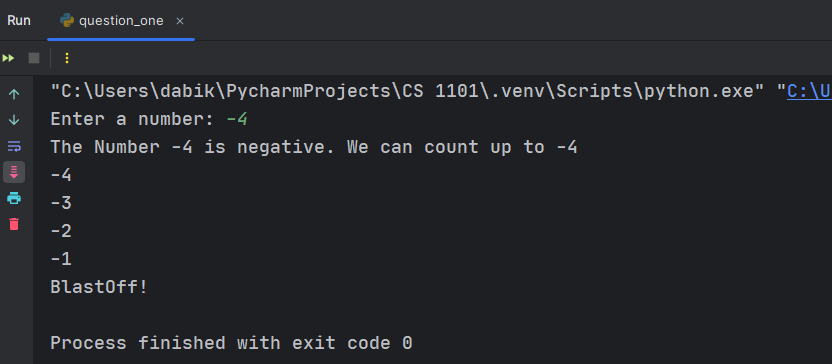
**Code Screenshots**



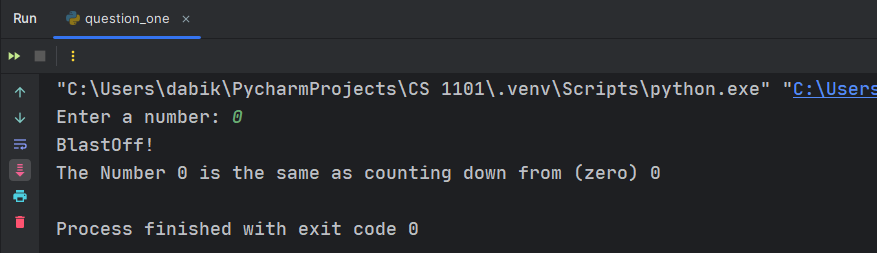
**Results when user input is positive**



**Results when user input is negative**



**Results when user input is 0 (zero)**

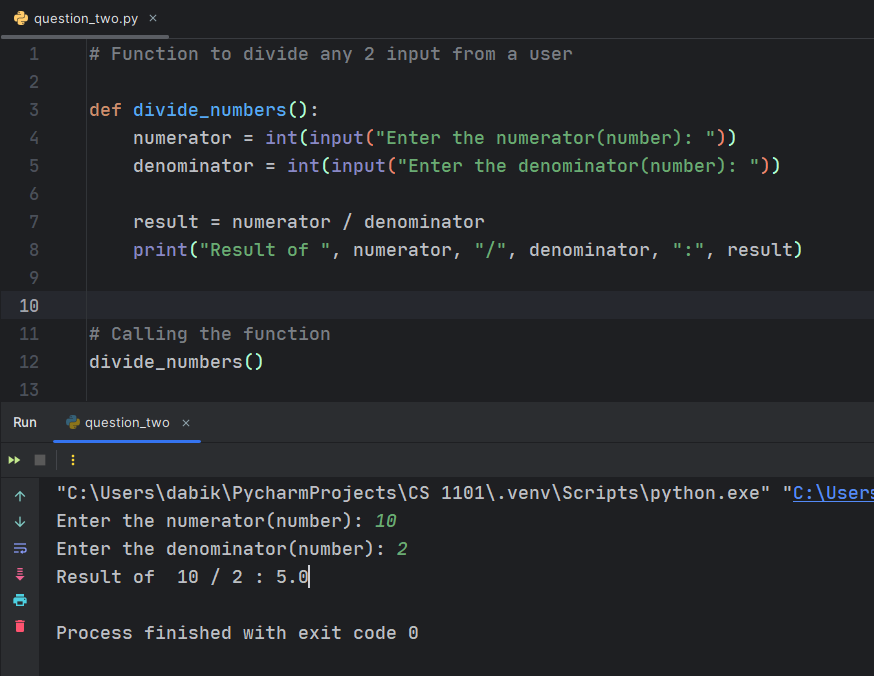


**Explanation**

* When the user input is negative, the program calls the count\_up function. This counts up to zero from the negative number and prints “BlastOff!”
* When the user input is positive, the program calls the count\_down function. This counts down to zero and prints “BlastOff!”
* For an input of zero, the program directly calls count\_down(num) to print "BlastOff! The Number 0 is the same as counting down from (zero) 0" as it is eventually the same as counting down from zero. This decision maintains consistency in my program behaviour for all input values provided by the user.

Q2

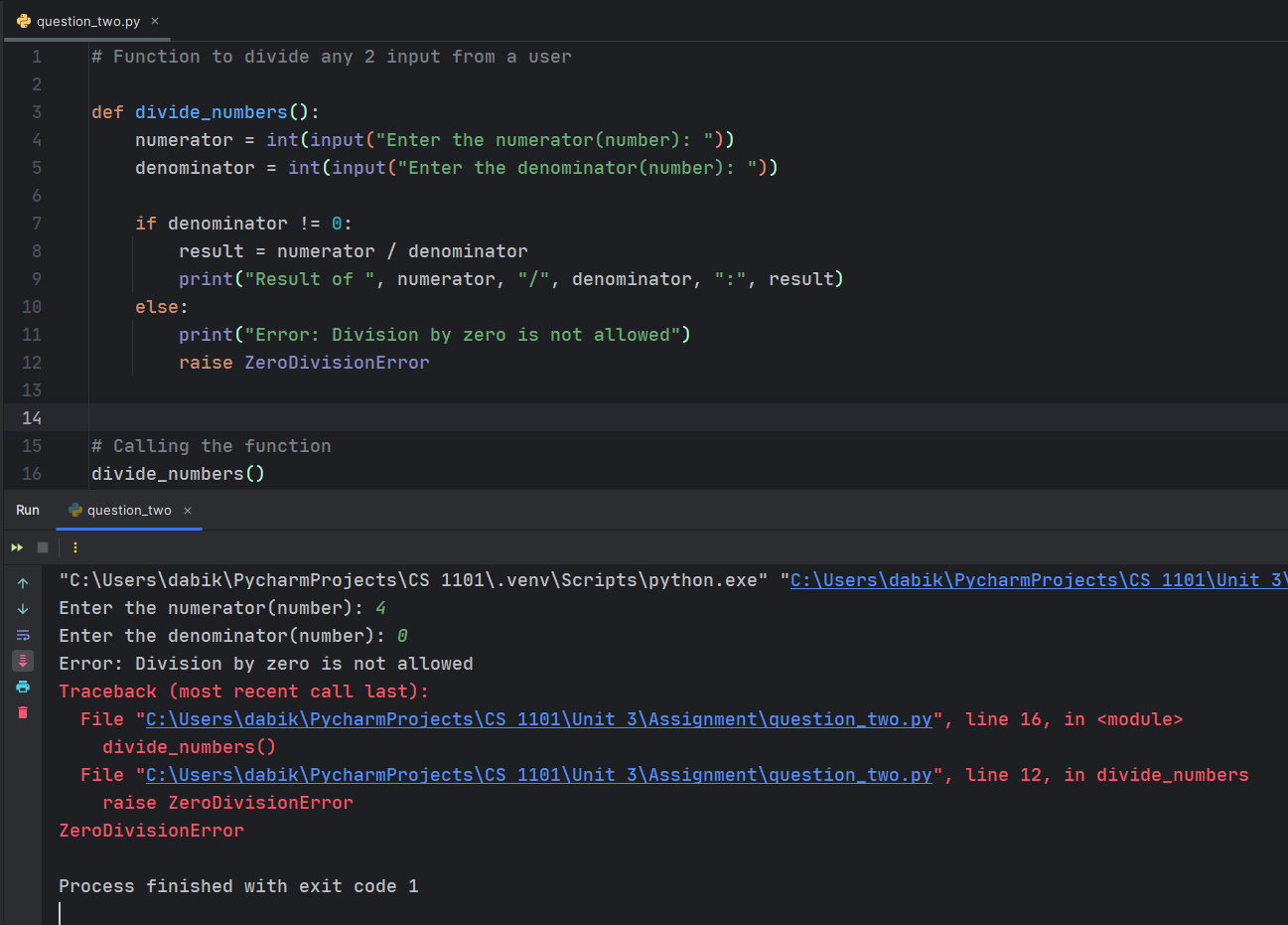
1. A Python program that prompts the user to enter two numbers and performs a division operation on the entered numbers



**Explanation**

* When the user provides a positive numerator and denominator, the user gets a response without any error. The results above show **10/2 = 5.0;** providing the results in a float-type

1. A condition that raises a runtime error if the second number is zero and an error message that indicates the cause of the error.



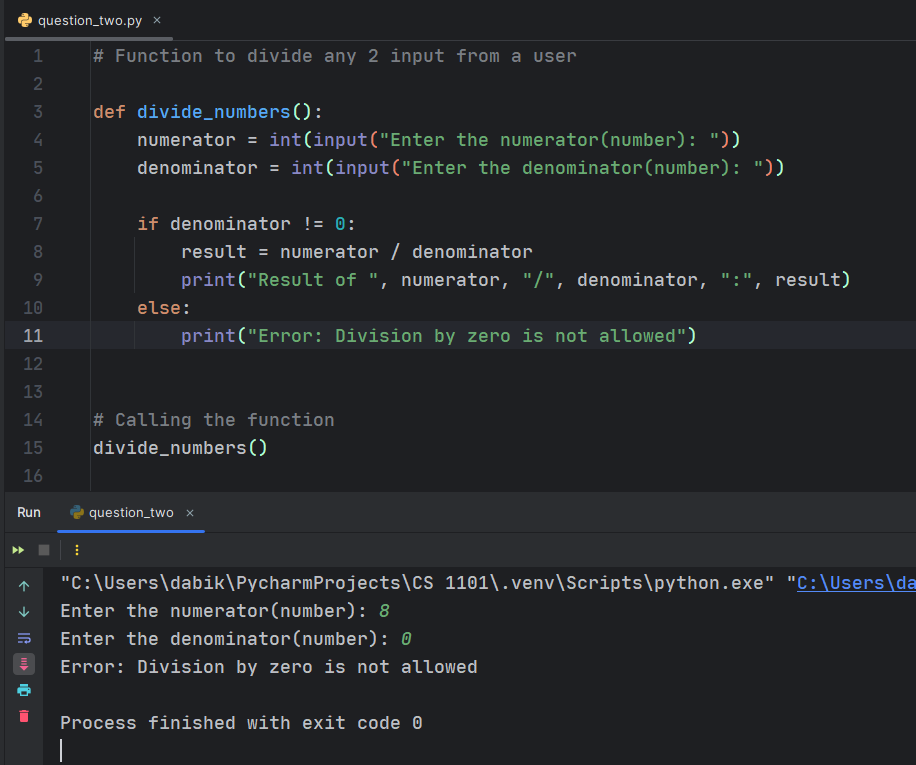
**Explanation**

* This program will raise a ***ZeroDivisionError*** if the user enters zero as the denominator because you can't divide by zero (Geeksforgeeks, 2024). This provides a learning opportunity for junior developers to understand how to handle division by zero errors in their programs.

1. Guiding junior developers in identifying the error message and implementing error handling techniques to handle the division by zero scenario (Allen Downey, 2015).

* **Identifying the error:** When running the program and providing zero as the denominator, it will raise a ***ZeroDivisionError***. This error message indicates that the program attempted to divide a number by zero, which is not allowed in programming/mathematics.
* **Locate the problematic line (where the problem occurred):** The error occurs on the line (line 12 as seen in the image above) where the division operation is performed, which is result = numerator/denominator.
* **Handle the error:** To prevent the program from crashing when dividing by zero, we need to handle the ***ZeroDivisionError.*** This can be done by adding a conditional statement to check if the denominator is zero before performing the division.

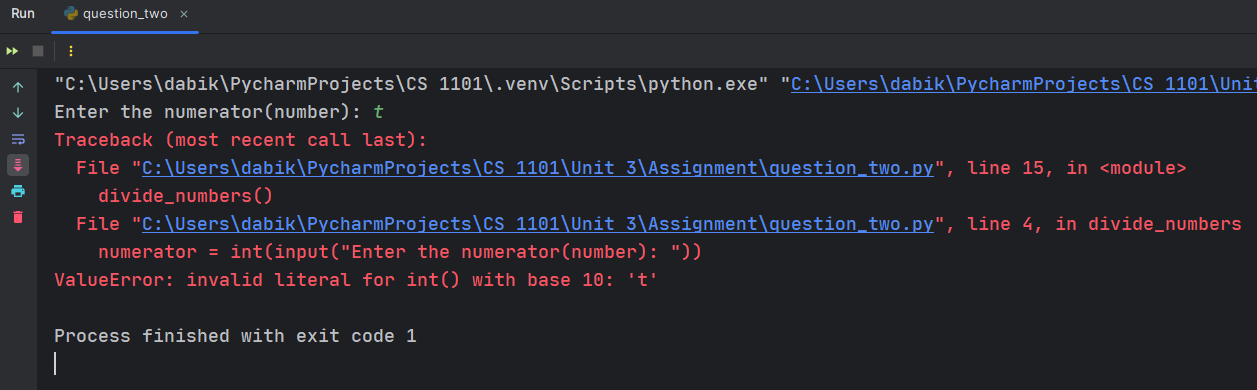
Code error handling division by 0.



* 8/0 gives the user a clear and clean error “*Error: Division by zero is not allowed*”

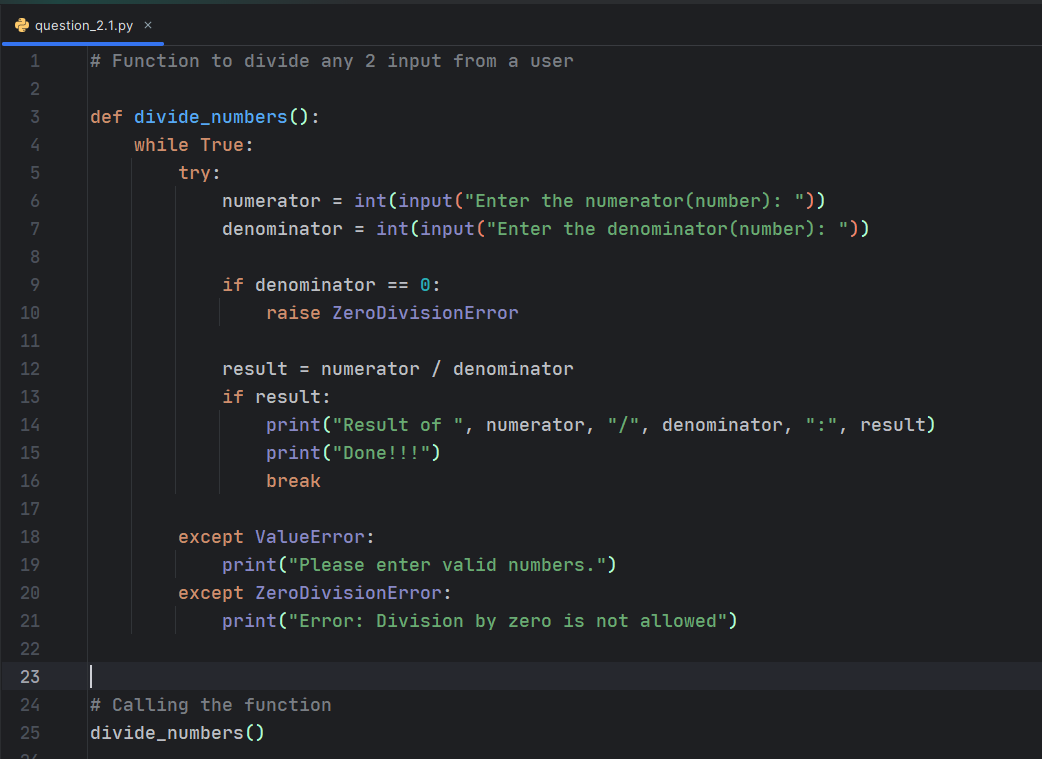
As a programmer, also ask questions as you code. An open mind is always encouraged when programming as it is a necessary skill (Diaz, 2023).

So, my question is, yes, the user is required to enter just numbers (integers) for the program to run without any error. But what if the user tries alphabetical letters like a, f, g, h or k? This will eventually create another error as shown in the image below.

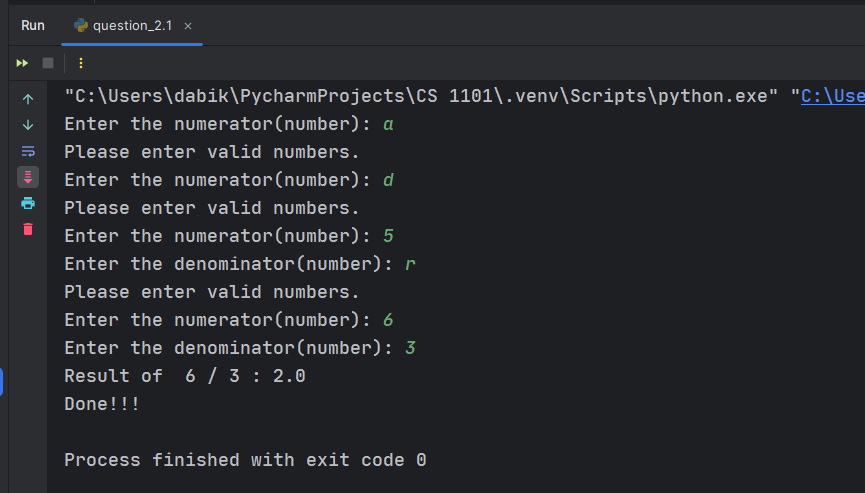


This might be intentional or an error by the user. Either way, we have to think about how to handle such exceptional cases.

Below is a full code that handles this case.



Results when invalid inputs are provided



**Explanation**

* The same function name is given but with a modification on the function body
* A While Loop is used in this case to make sure the user must provide a valid input before any further calculations are made (Allen Downey, Chapter 7: Iteration - 7.3 The while statement, 2015).
* This program will raise a ***ZeroDivisionError*** if the user enters zero as the denominator because you can't divide by zero.
* A conditional statement is used to check if the results are provided without any error. A ***break keyword*** is used to break out of the loop otherwise, the user will keep providing input for calculations.
* Inside the try block, int (input ("...")) is used to get user input and convert it to an integer. If the conversion is successful, the integer is returned.
* If the user enters something that cannot be converted to an integer (such as letters or symbols), a ***ValueError*** will be raised. This is caught in the except ***ValueError*** block, which prints an error message and prompts the user to enter a valid integer again. (Pankaj, 2022)
* This ensures that the user input is an integer and provides a way to handle cases where the input is not valid.

# References

Allen Downey, G. T. (2015). Chapter 7: Iteration - 7.3 The while statement. In G. T. Allen Downey, *Think Python: How to Think Like a Computer Scientist* (p. 64). Needham, Massachusetts: Green Tea Press.

Allen Downey, G. T. (2015). *Think Python: How to Think Like a Computer Scientist.* Needham, Massachusetts: Green Tea Press.

Diaz, O. (2023, 07 04). *5 Soft Skills Every Software Developer Should Have*. Retrieved from IEEE Computer Society: https://www.computer.org/publications/tech-news/build-your-career/5-soft-skills-for-software-developers

Geeksforgeeks. ( 2024, Feb 06 ). *Zerodivisionerror Integer by Zero in Python*. Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/zerodivisionerror-integer-by-zero-in-python/

Pankaj. (2022, August 3). *Python ValueError Exception Handling Examples*. Retrieved from Digitalocean: https://www.digitalocean.com/community/tutorials/python-valueerror-exception-handling-examples