**PYTHON**

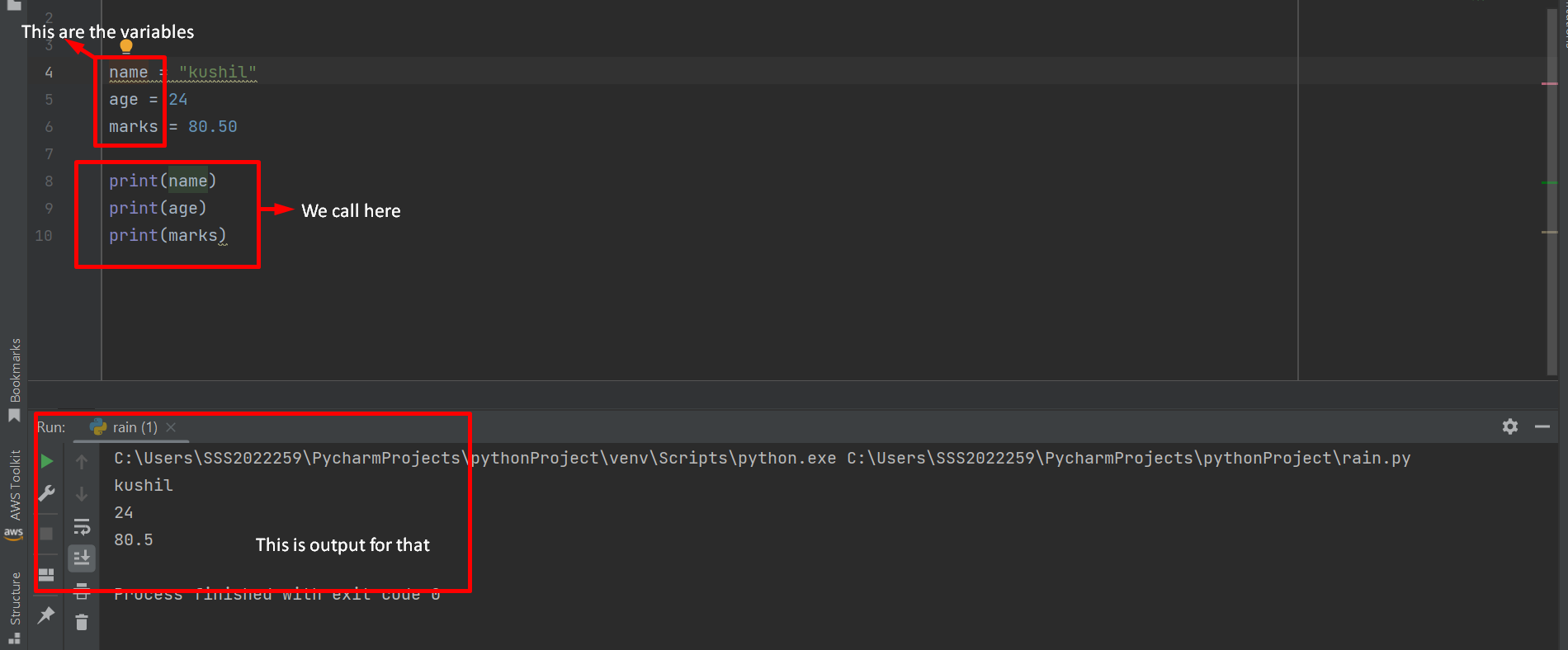
**What Is python :-**

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum in 1991.

Python is commonly used for developing websites and software, task automation, data analysis, and data visualization.

**Python Variables :-**

Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.



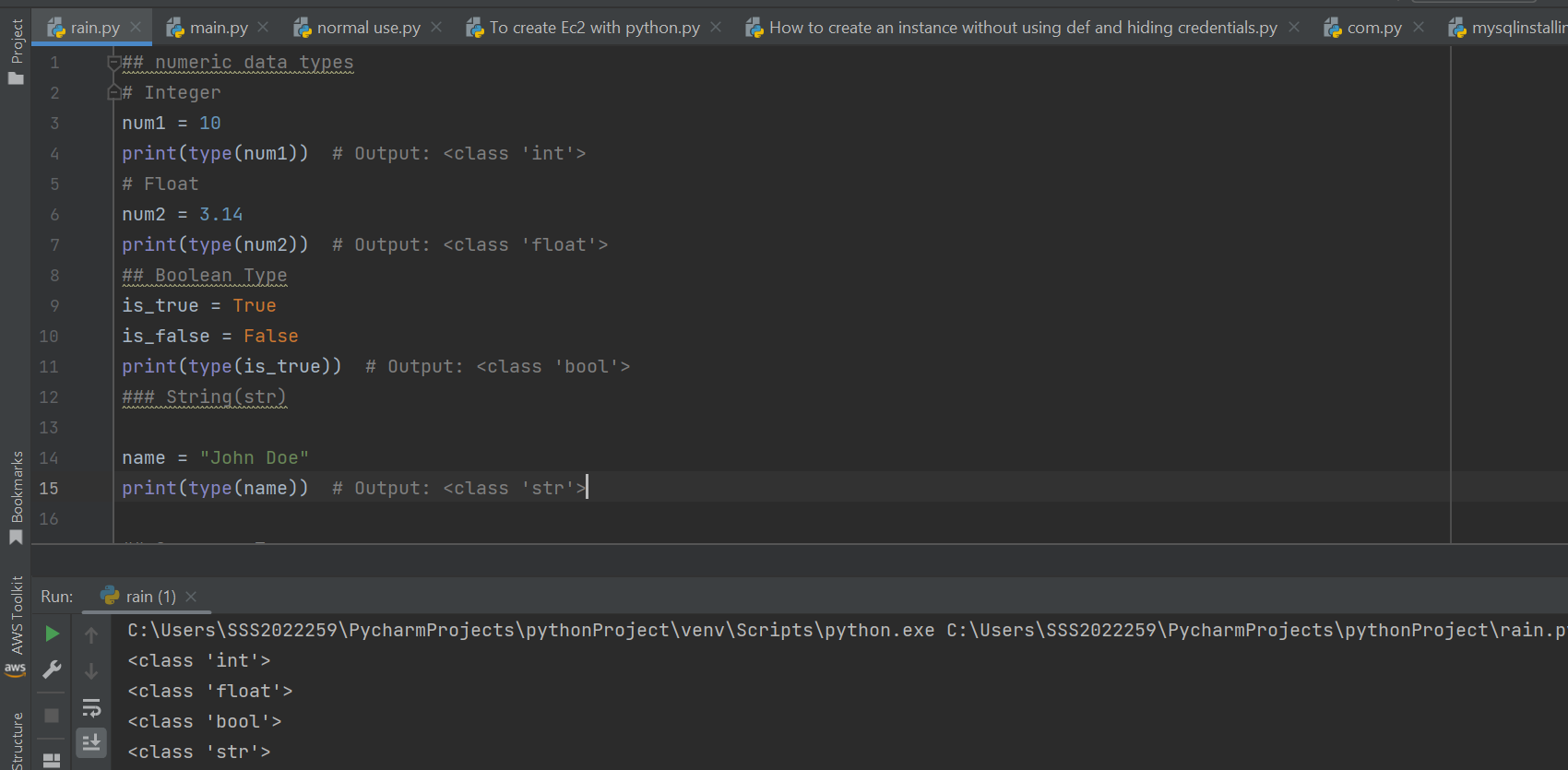
Here name is variable that store some data called ‘kushil’.

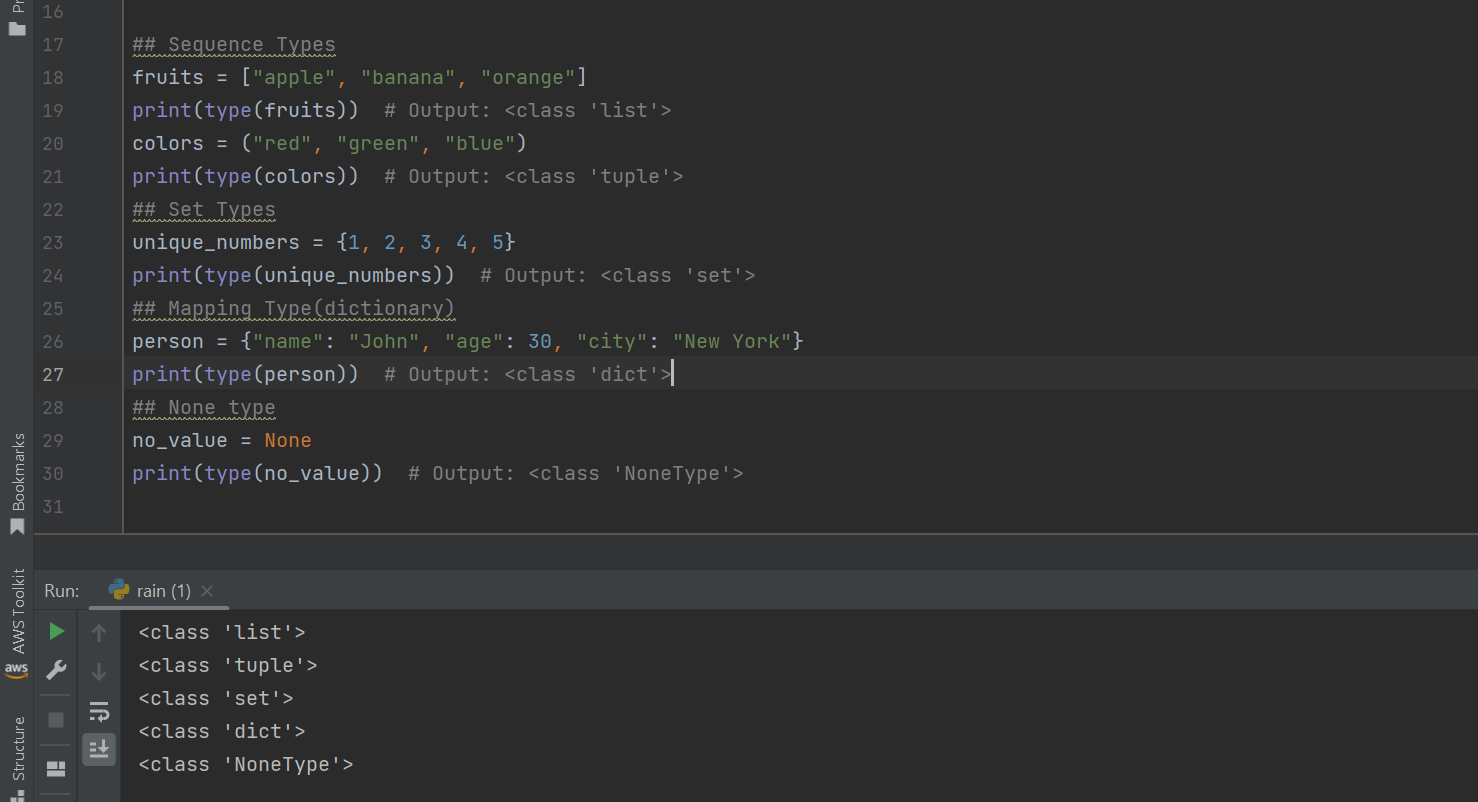
The variable stores all types of data types. we stored specific data In that variables.

**Python Data** **Types** :-

In python programming, The data type is an important concept.Variables can store data of different types, and different types can do different things.Python has the following data types built-in by default, in these categories.

|  |  |  |
| --- | --- | --- |
| **Data Types** | **Names** | **Data Store Sturacture** |
| Text type | String(str) | ‘’ or ” “ or ’’’ ‘’’ |
| Numeric Types | Int,Float,Complex | 10,12.67,3+2j |
| Sequence Types | List,Tuple | [ ] , ( ) |
| Mapping Type | Dict | {key:values} |
| Set Types | Set | {} |
| Boolean Type | Bool | True or False |
| None Type | Nonetype | none |

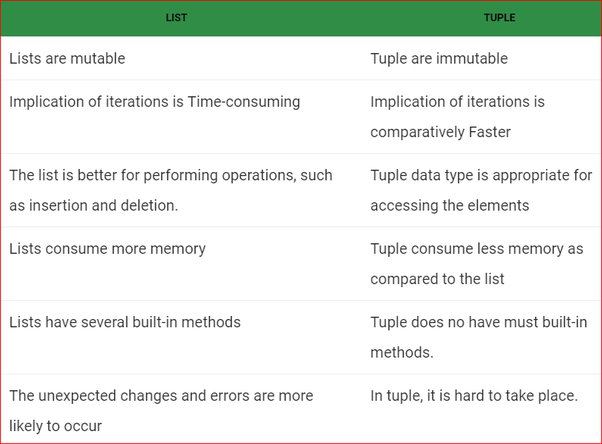




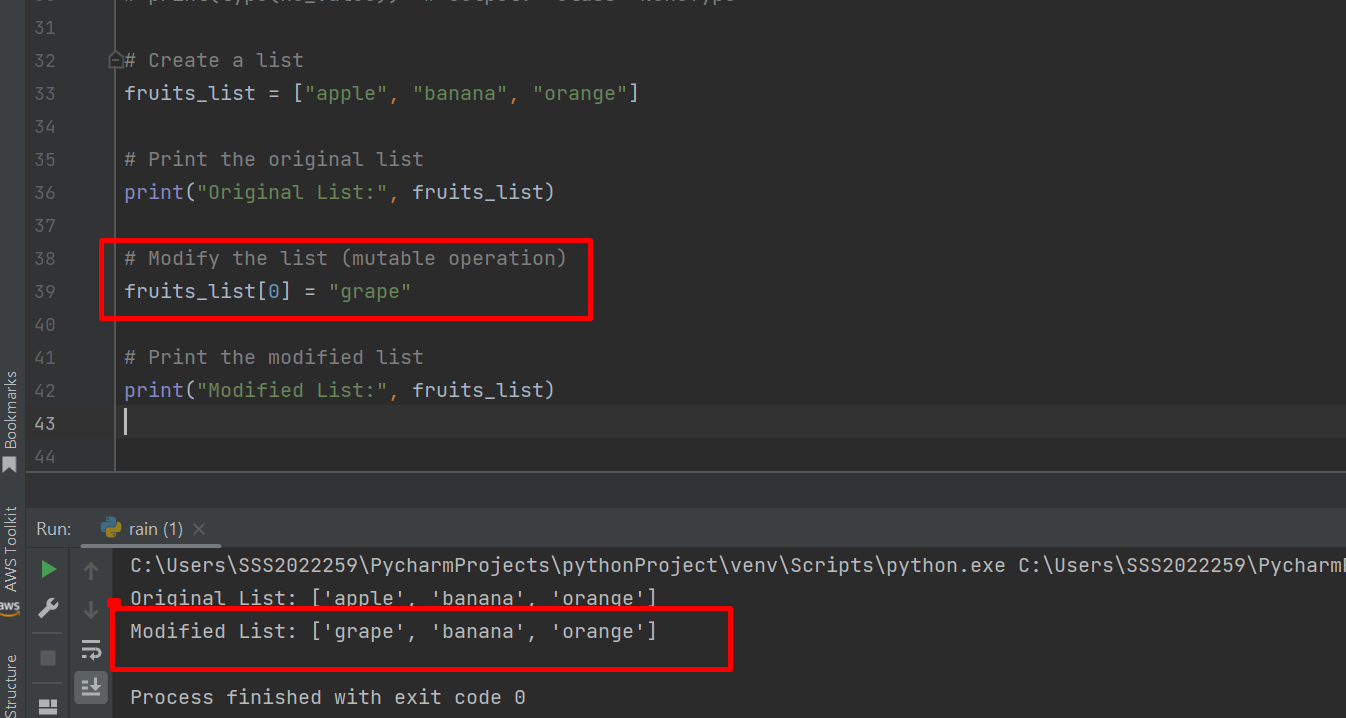
The main difference between **list** and **tuple** is list is **mutable** and tuple is **immutable** .

**Mutable** means we **change** data.

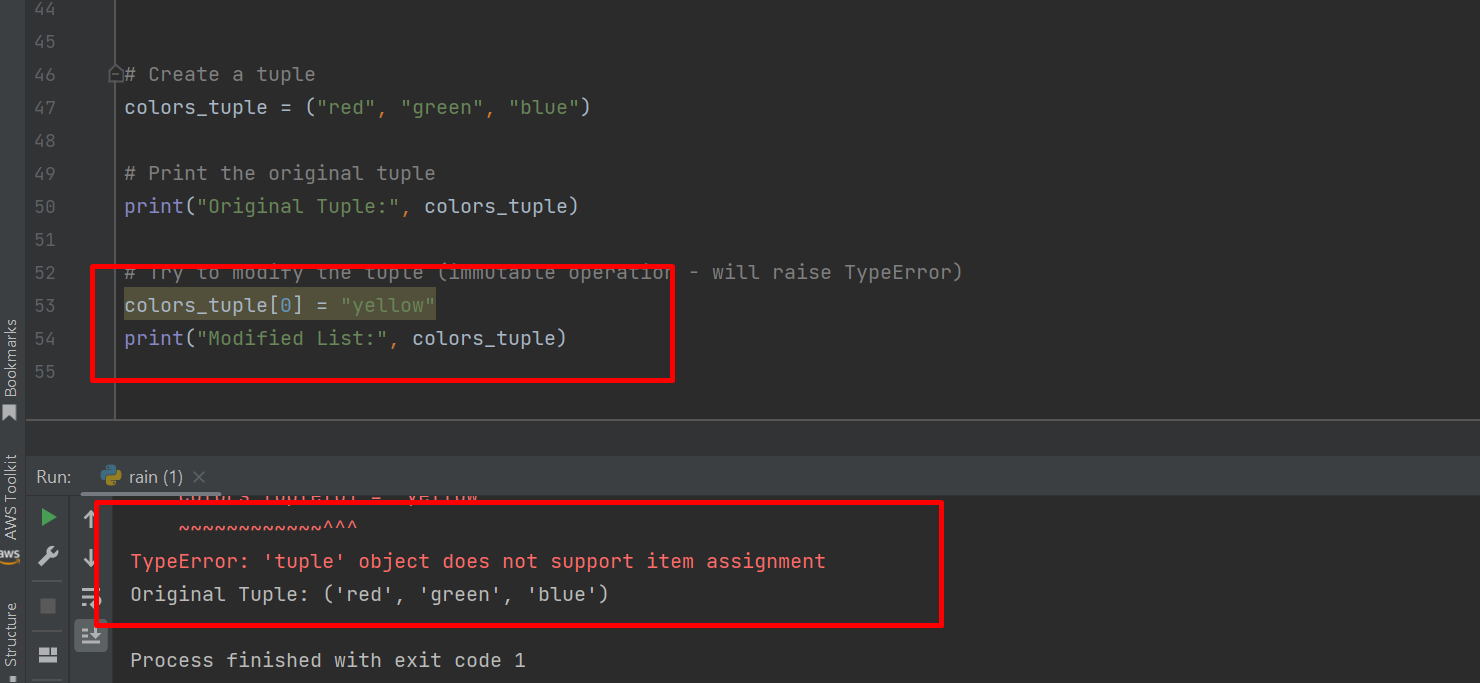
**Immutable** means we **can not change** data.



**MUTABLE EXAMPLE :-**



**IMMUTABLE EXAMPLE :-**



**Python Keywords :-**

Python keywords are unique words reserved with defined meanings and functions that we can only apply for those functions. You'll never need to import any keyword into your program because they're permanently present.

Assigning a particular meaning to Python keywords means you can't use them for other purposes in our code. You'll get a message of Syntax Error if you attempt to do the same.

So we don’t use keywords in variables and some data in our code. We have some keywords are in python.They are

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| False | await | else | import | pass |
| None | break | except | in | raise |
| True | class | finally | is | return |
| and | continue | for | lambda | try |
| as | def | from | nonlocal | while |
| assert | del | global | not | with |
| async | elif | if | or | yield |

**Python Literals :-**

Python Literals can be defined as data that is given in a variable or constant.

The different types of literals are :

### String literals:

### Numeric literals:

### Boolean literals:

### Special literals:

### Literal Collections:

**Python Operators :-**

In Python, operators are symbols or special characters that perform various operations on variables, values, or expressions.

They allow you to manipulate data and perform calculations in your Python programs.

Here's an overview of the different types of operators in Python:

**1.Arithmetic Operators:**

These operators perform basic arithmetic operations.

Addition: +

Subtraction: -

Multiplication: \*

Division: /

Floor Division: // (returns the quotient without the remainder)

Modulus: % (returns the remainder of division)

Exponentiation: \*\* (raise to the power)

**2.Assignment Operators:**

These operators are used to assign values to variables.

Assignment: =

Addition Assignment: +=

Subtraction Assignment: -=

Multiplication Assignment: \*=

Division Assignment: /=

Modulus Assignment: %=

Exponentiation Assignment: \*\*=

Floor Division Assignment: //=

**3.Comparison Operators:**

These operators compare values and return Boolean results.

Equal to: ==

Not equal to: !=

Greater than: >

Less than: <

Greater than or equal to: >=

Less than or equal to: <=

**4.Logical Operators:**

These operators perform logical operations on Boolean values.

Logical AND: and

Logical OR: or

Logical NOT: not

**5.Bitwise Operators:**

These operators perform bitwise operations on integers.

Bitwise AND: &

Bitwise OR: |

Bitwise XOR: ^

Bitwise NOT: ~

Left Shift: <<

Right Shift: >>

**6.Membership Operators:**

These operators test for membership in a sequence (e.g., lists, strings, tuples, etc.).

in: Returns True if the value is found in the sequence.

not in: Returns True if the value is not found in the sequence.

**7.Identity Operators:**

These operators compare the memory addresses of two objects.

is: Returns True if both variables point to the same object.

is not: Returns True if both variables point to different objects.

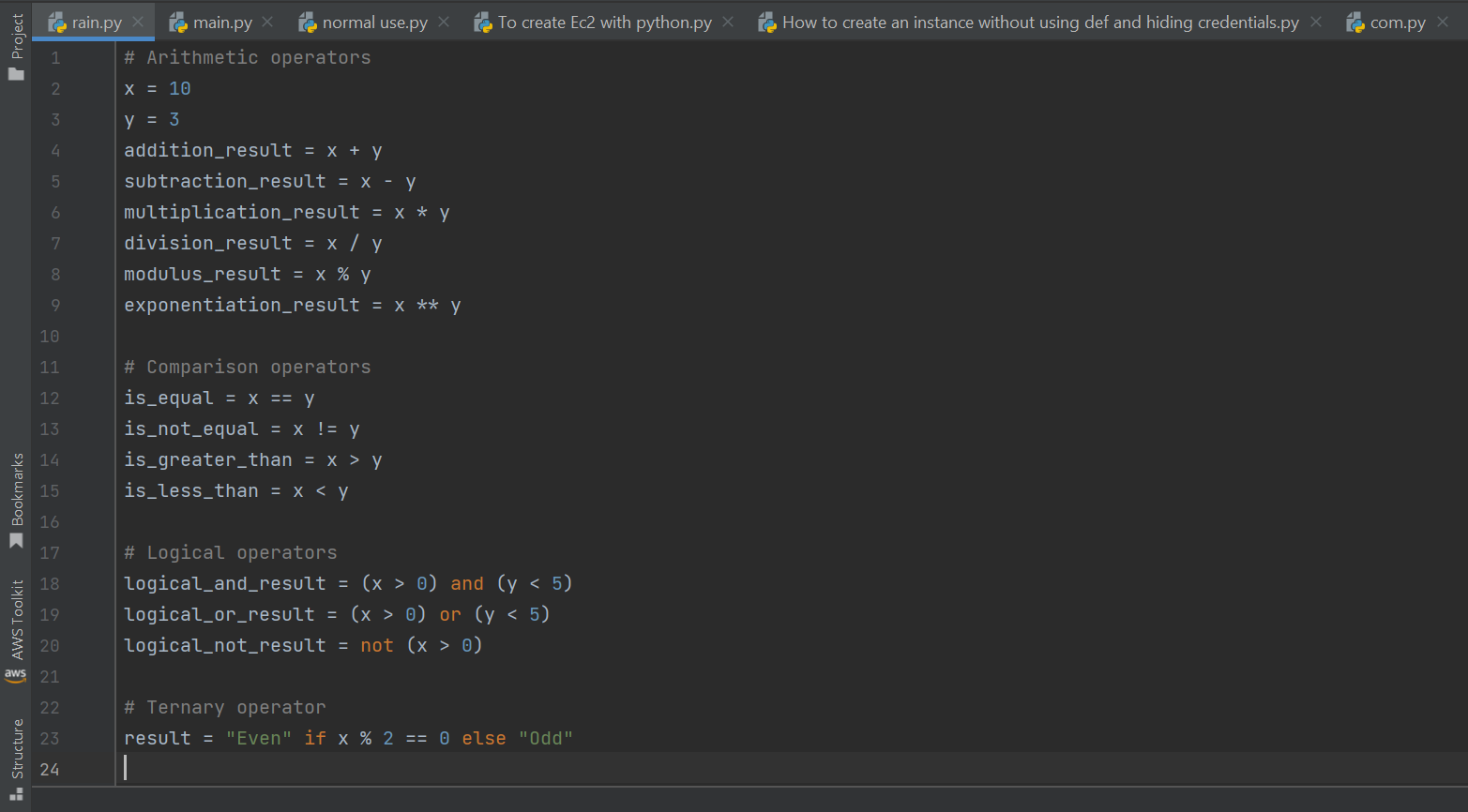
**8.Ternary Operator (Conditional Expression):**

The ternary operator is a shorthand way of writing conditional expressions.

x if condition else y: Returns x if condition is True, else returns y.

These operators can be used in combination to perform complex operations. For example, you can use arithmetic, comparison, and logical operators together to create conditional statements or complex calculations.

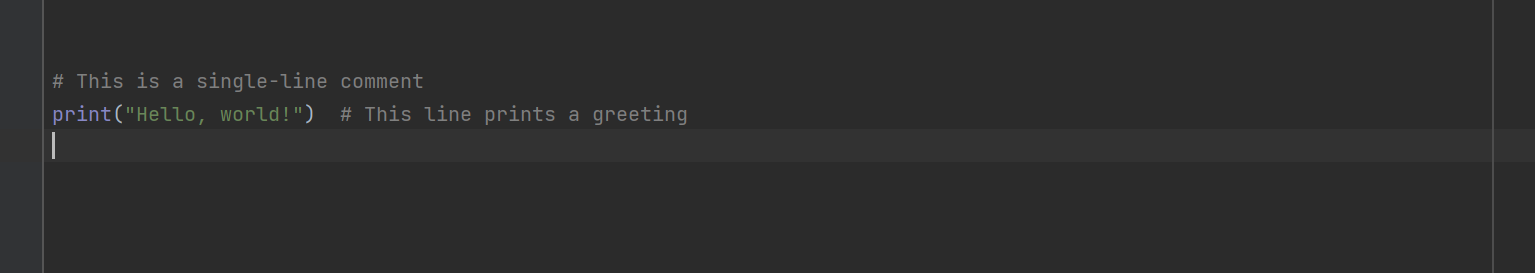
Here's an example of using some operators in Python:



**Pythons comments :-**

In Python, comments are lines in the code that are ignored by the interpreter and are used to provide explanations, documentation, or temporary disablement of code.

Comments help make the code more readable and understandable for developers and others who might review the code.



**Python If-Else :-**

In Python, the 'if-else' statement is used for conditional execution.

It allows you to execute certain blocks of code based on whether a given condition is true or false.

The basic syntax of the if-else statement is as follows:

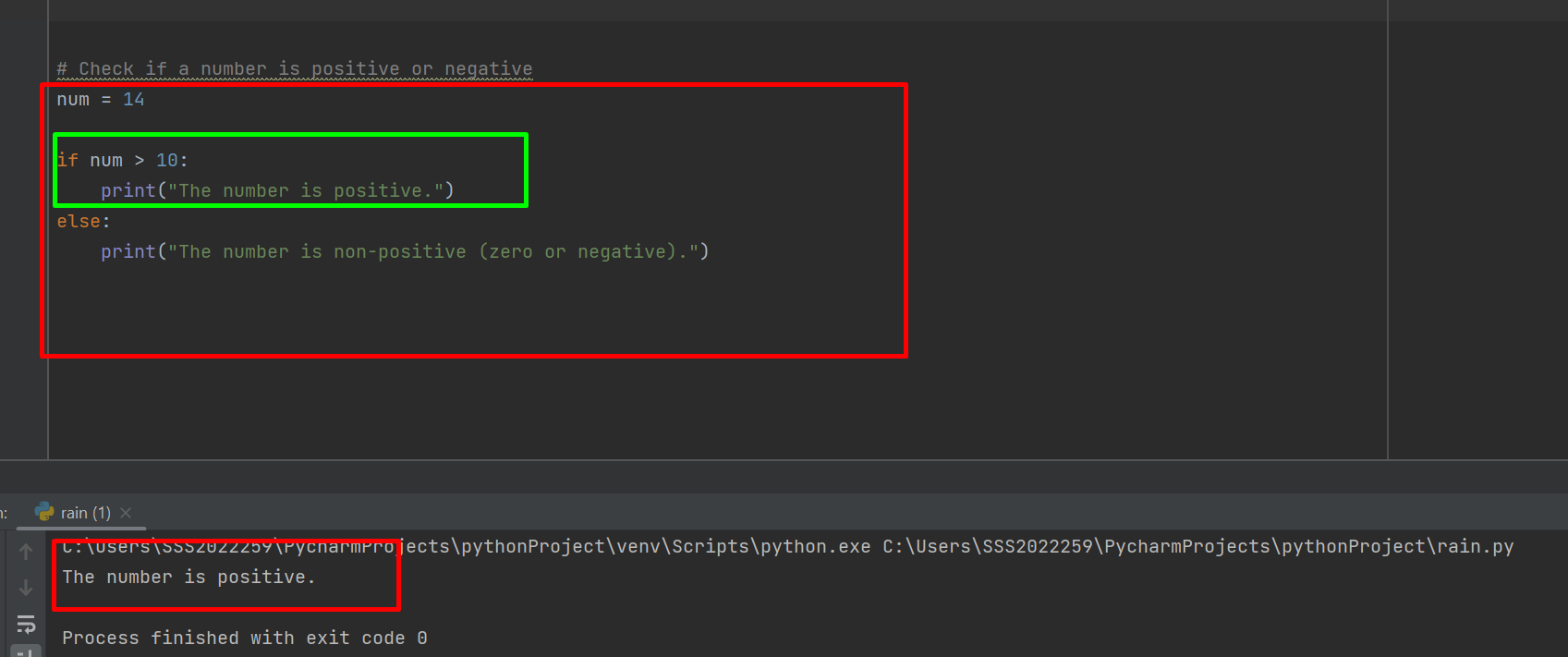
**if condition:**

\* Code block to be executed if the condition is True.

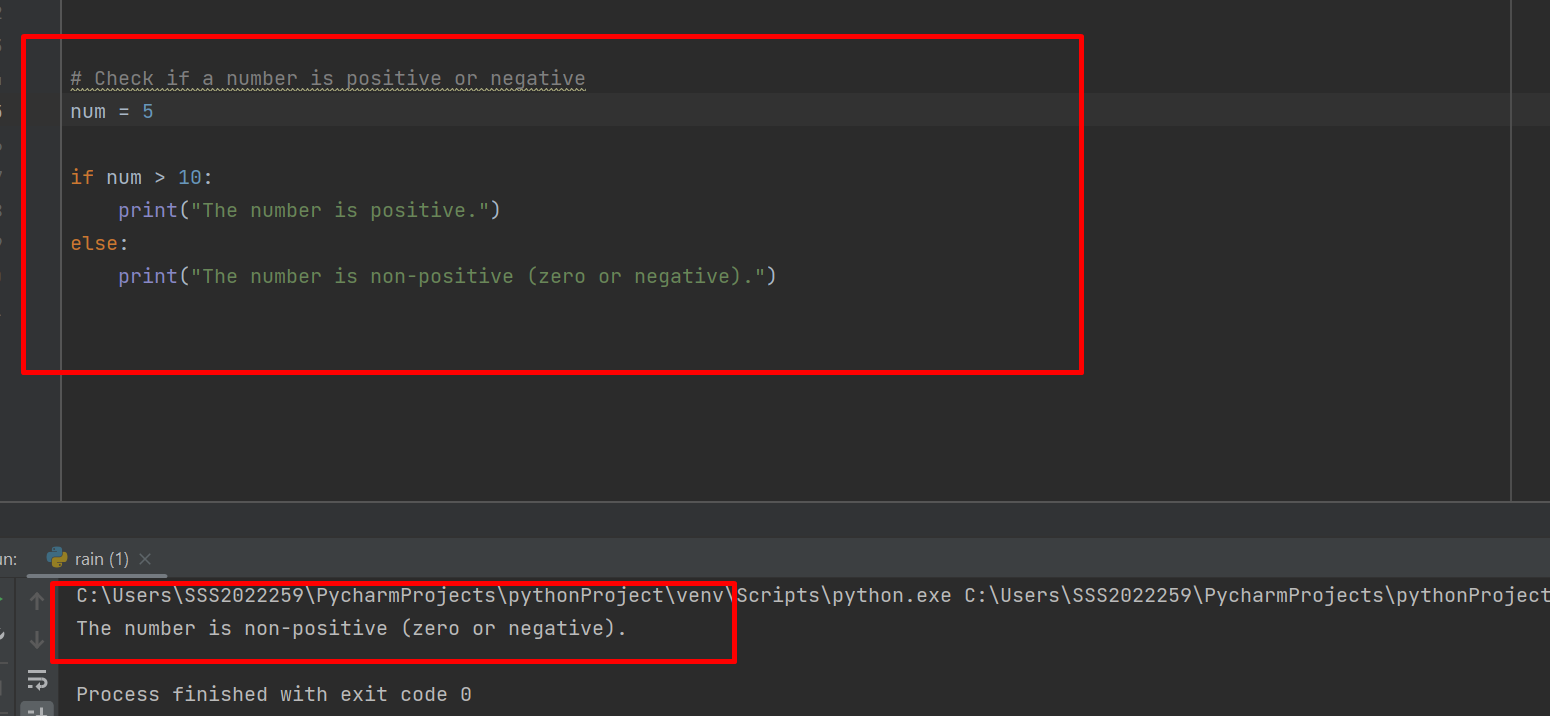
**else:**

\* Code block to be executed if the condition is False.

**If condition example :**



**Else condition example :**



**Python Loops :-**

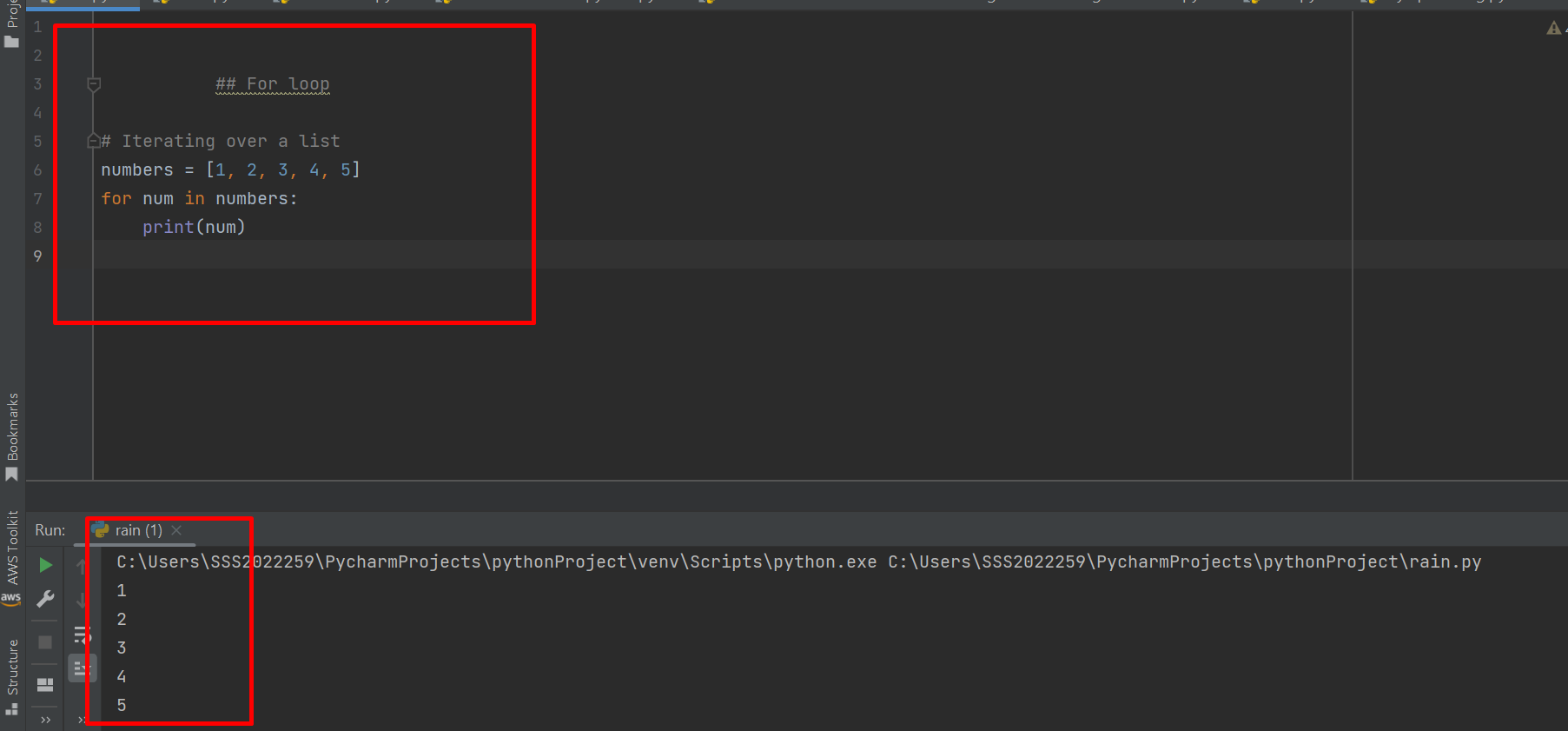
In Python,The loops are used to repeatedly execute a block of code based on a specified condition.

Python supports two main types of loops 1.**for loops** and 2.**while loops**.

**For loop :-**

The for loop is used to iterate over a sequence (e.g., a list, tuple, string, etc.) and execute a block of code for each item in the sequence.

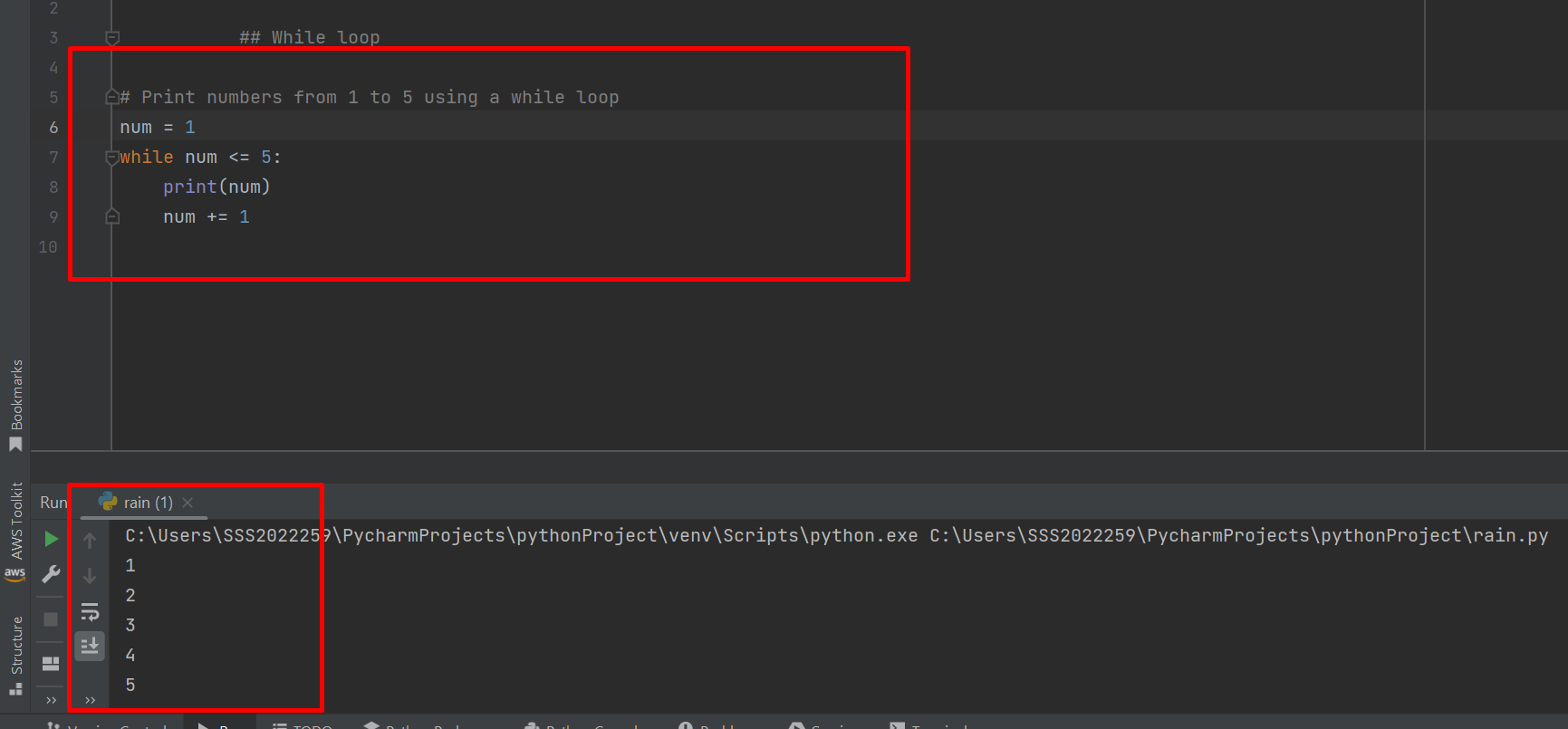
Example:-



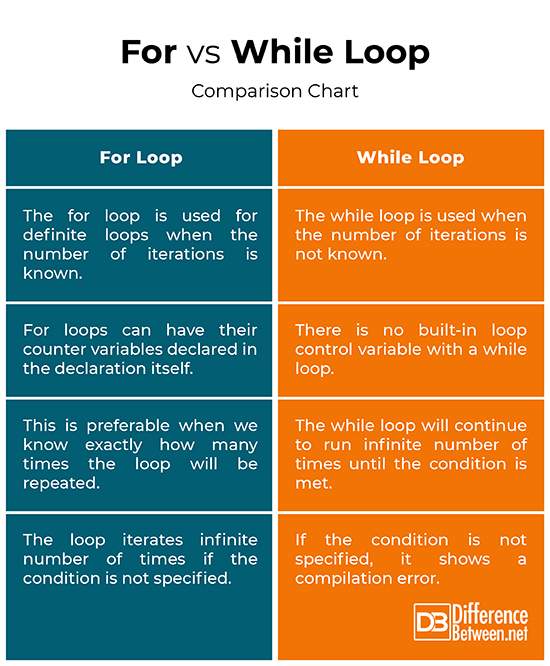
**While loop:**

The while loop is used to repeatedly execute a block of code as long as a specified condition is True.

**Example :-**



The main differences between the **for loop** and **while loop** are :



**Python break , continue and pass :-**

In Python, **break**, **continue**, and **pass** are control flow statements used within loops and conditional statements to modify the flow of execution.

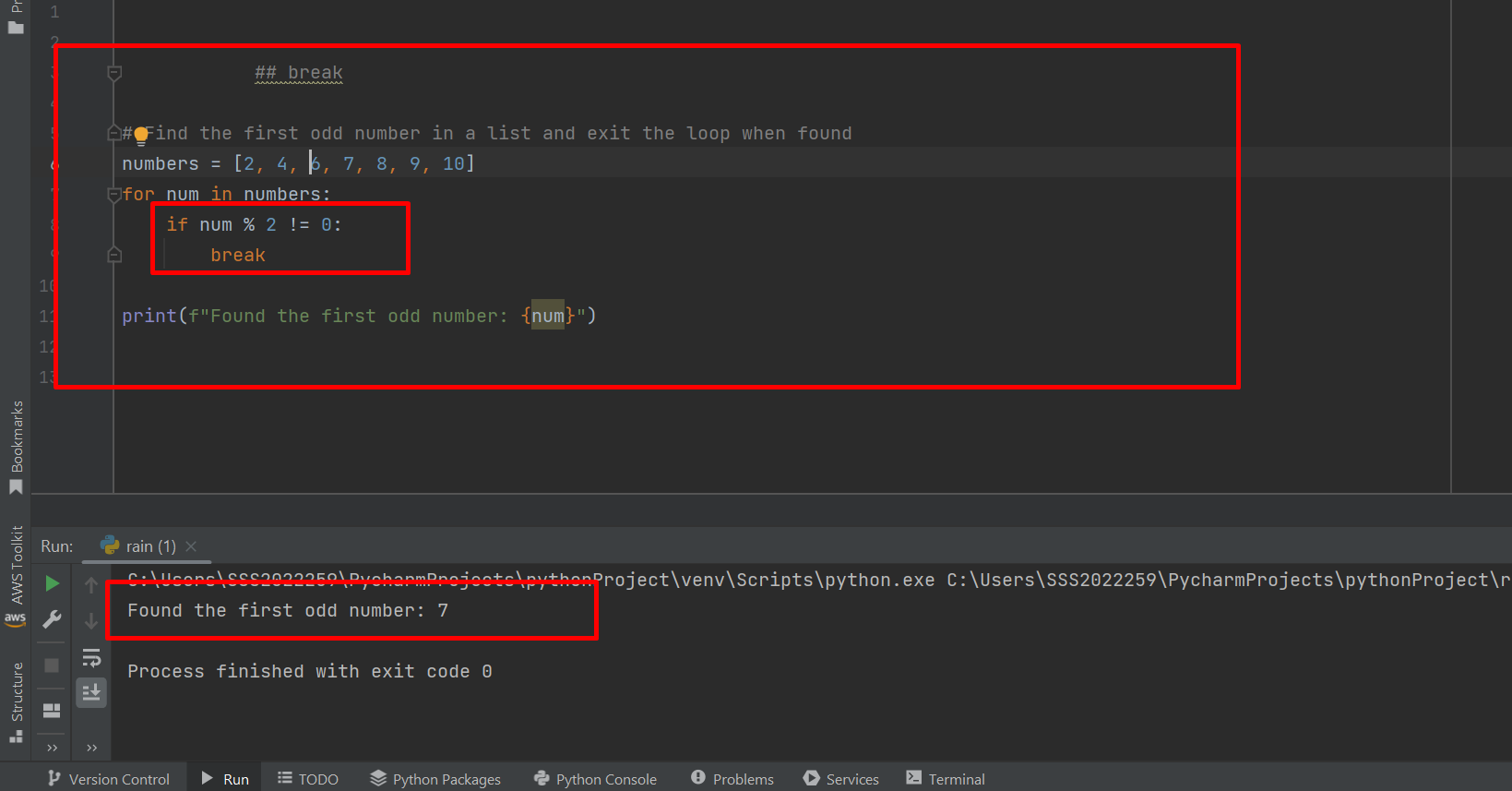
Let's take a closer look at each of these statements with examples:

1. **Break:**

The break statement is used to exit a loop prematurely when a certain condition is met.

When the break statement is encountered, the loop immediately terminates, and the program continues executing from the statement after the loop.

**Example:-**

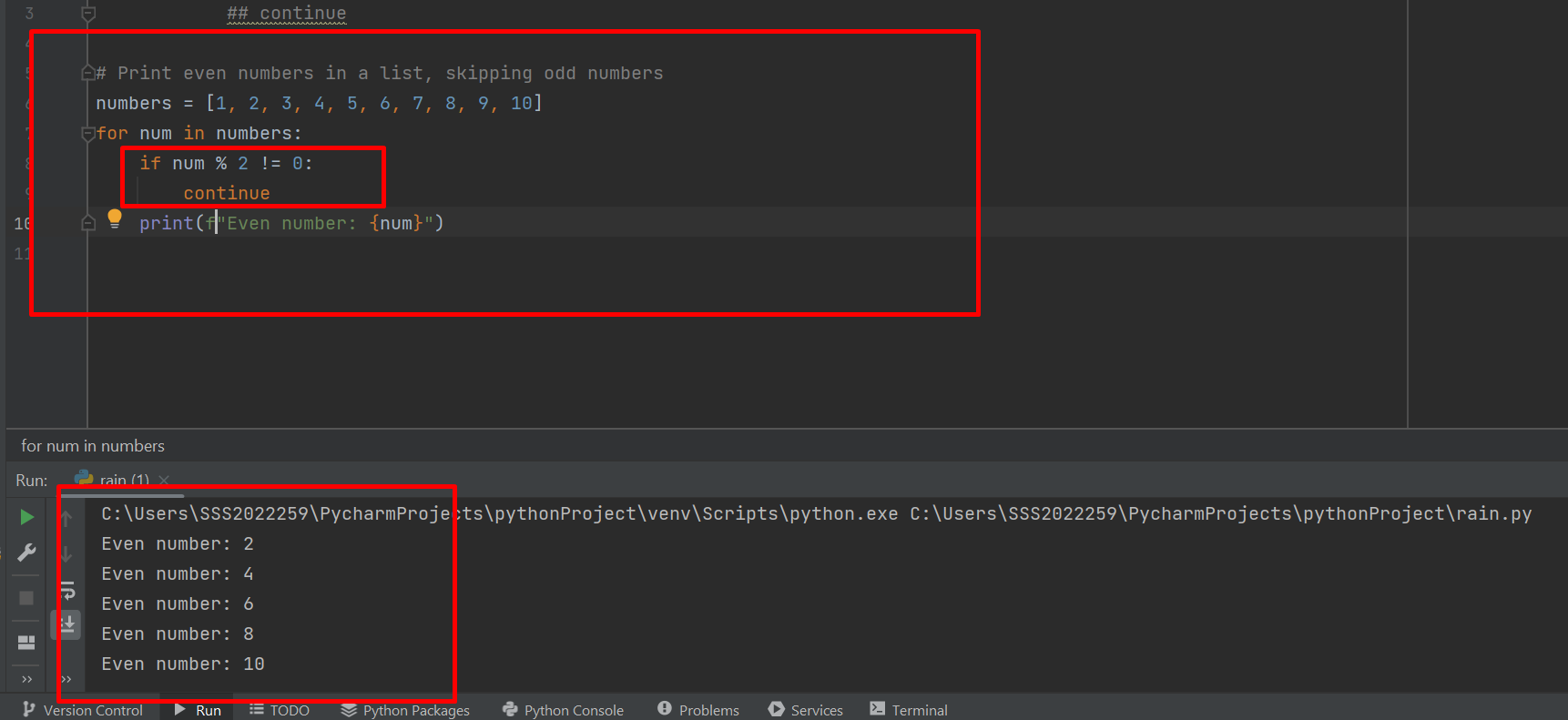


**2.continue:**

The continue statement is used to skip the rest of the loop iteration and continue with the next iteration.

When the continue statement is encountered, the loop jumps to the next iteration, skipping any code that comes after the continue statement within the loop.

**Example:**

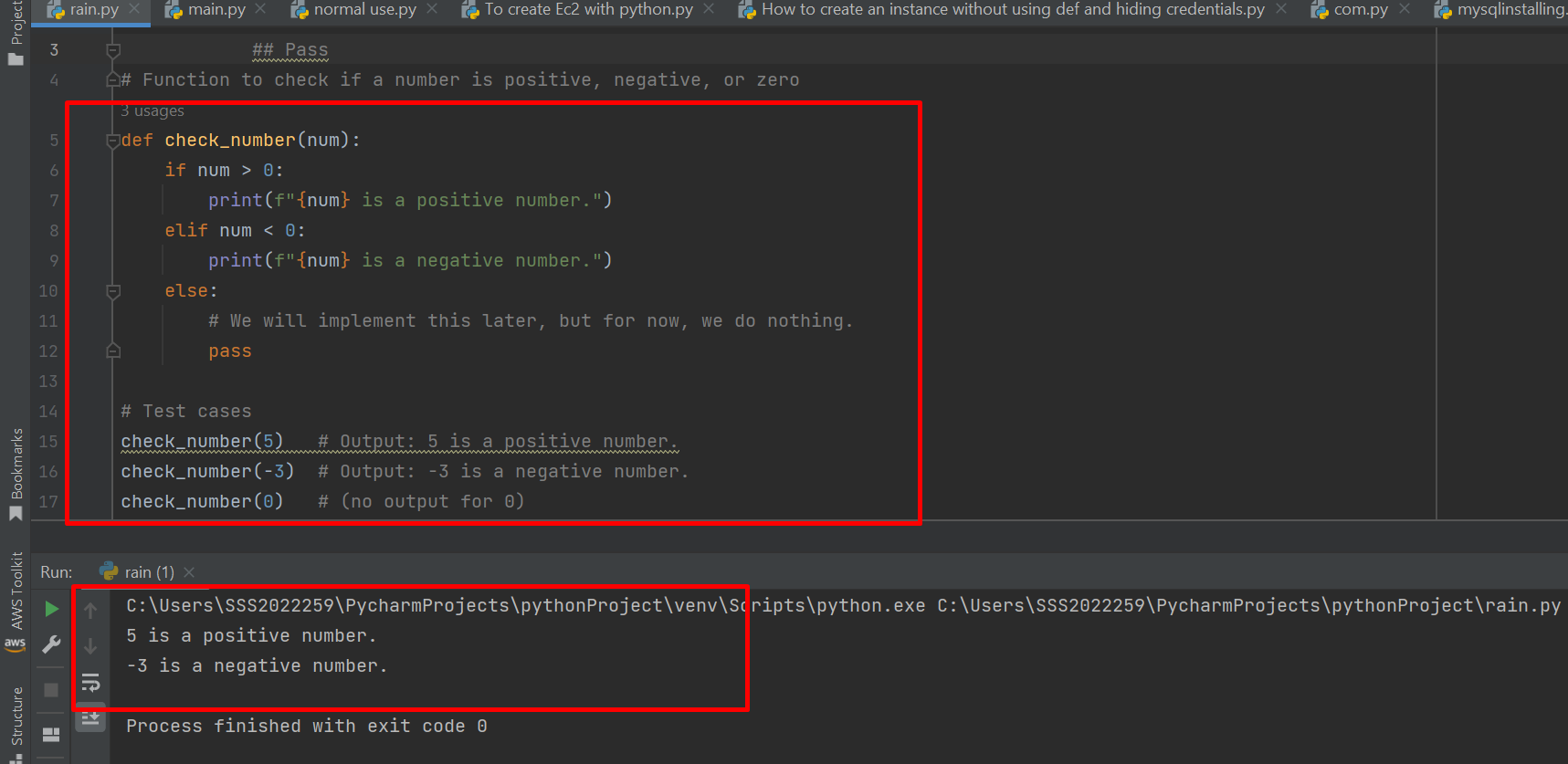


**3. pass :-**

The **pass** statement is a null operation; it does nothing when executed.

It is used as a placeholder where syntactically a statement is required, but no action is needed. It is often used in situations where you plan to implement code later but want to avoid syntax errors in the meantime.

**Example:**



**Python Functions :-**

In Python, functions are blocks of reusable code that perform specific tasks. Functions help break down complex problems into smaller, manageable parts and promote code re-usability and maintainability.

A function is defined using the “**def ”** keyword, followed by the function name, parameters (if any), and a block of code inside the function.

**1.Function without Parameters and Return Value:**

Functions can be defined without any parameters and without returning any value.

**Example:**



**2.Function with Parameters:**

Functions can take parameters (inputs) that can be used inside the function.

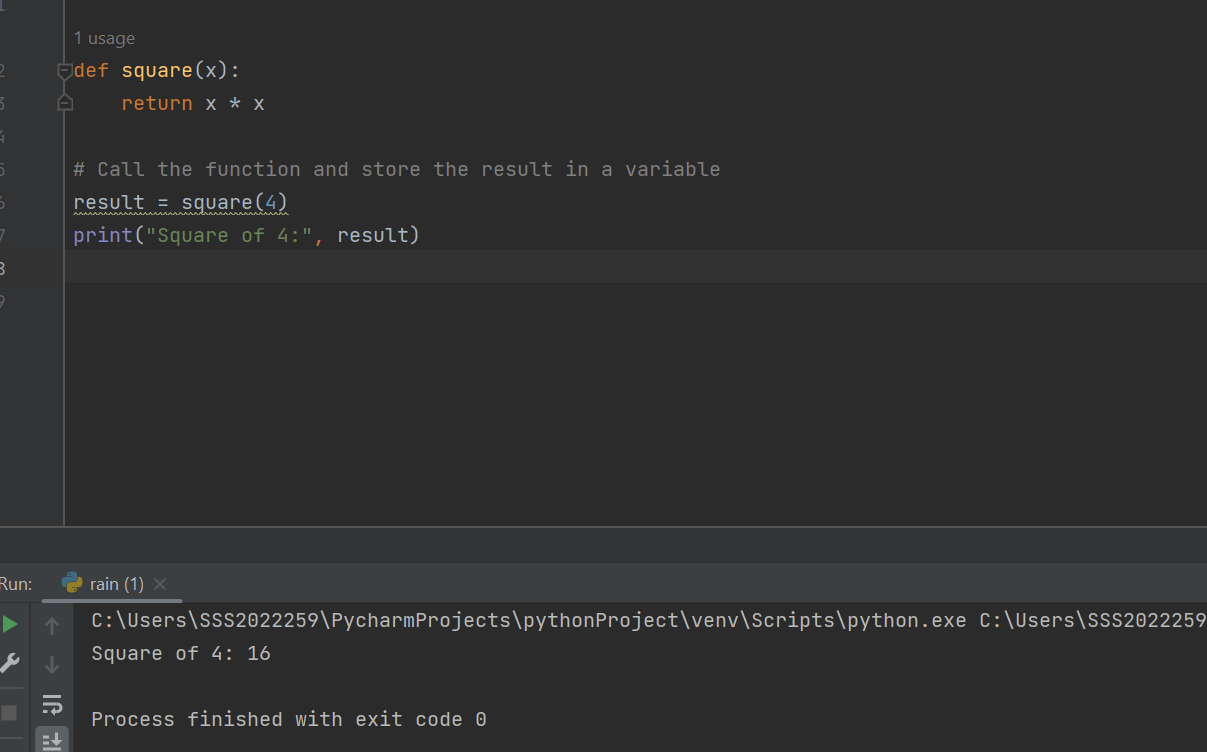
**Example:**



**3.Function with Return Value:**

Functions can return a value using the return statement.

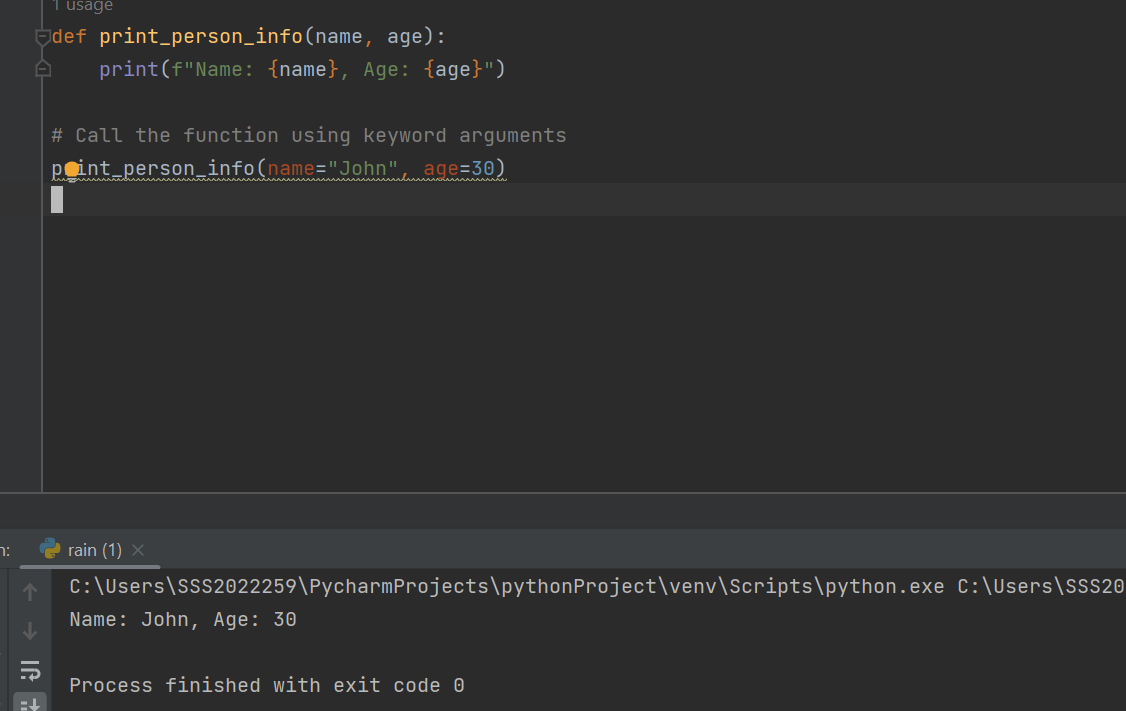
**Example:**



**4.Function with Keyword Arguments:**

You can use keyword arguments to pass values to function parameters explicitly.

**Example:**



**Python Built-In Functions:-**

In Python, **built**-in functions are pre-defined functions that are included in the Python standard library and can be used directly without requiring any additional imports.

These functions cover a wide range of tasks and operations, making Python a powerful and user-friendly programming language.

Some examples for built-in functions are 1.print (), 2.len(), 3.max(), 4.min(), 5.range(), 6.input()…..etcs.

**Python Lambda Functions :-**

In Python, a **lambda** function is a small anonymous function that can have any number of arguments but can only have one expression.

Lambda functions are often used when you need a simple function for a short period and don't want to define a full-fledged function using the "def" keyword.

Lambda functions are also known as "anonymous functions" because they don't have a name.

The Example for **lambda** functions is :

