**INTRODUCTION TO PYTHON PROGRAMMING**

**1. WHAT IS PYTHON?**

Python is a **high-level, interpreted, and general-purpose programming language**.

Python is one of the most popular programming languages. It’s simple to use, packed with features and supported by a wide range of libraries and frameworks. Its clean syntax makes it beginner-friendly.

* A high-level language, used in web development, data science, automation, AI and more.
* Known for its readability, which means code is easier to write, understand and maintain.
* Backed by library support, so we don’t have to build everything from scratch, there’s probably a library that already does what we need.

**It is widely used for:**

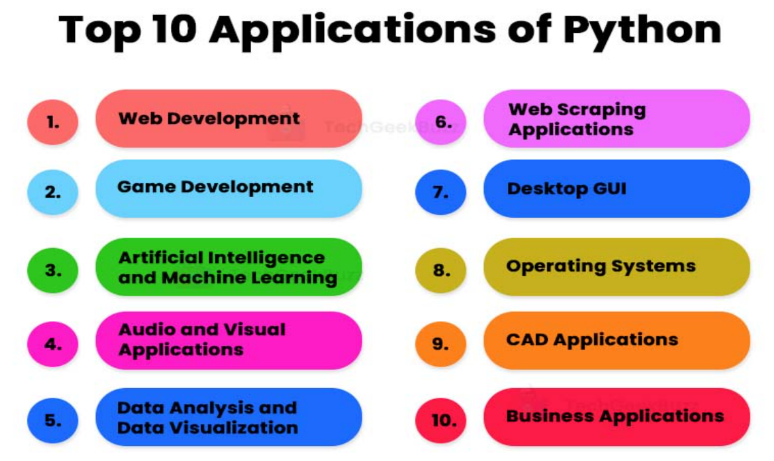
* Web development
* Data science
* Artificial intelligence
* Machine learning
* Automation

**Key Features:**

* Simple and easy to learn
* Open-source and free
* Large standard library
* Cross-platform support
* Object-oriented and functional

**why to learn python?**

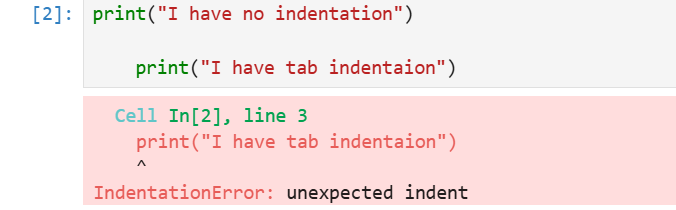
* Requires fewer lines of code compared to other programming languages.
* Provides Libraries / Frameworks like Django, Flask, Pandas, TensorFlow, Scikit-learn and many more for Web Development, AI/ML, Data Science and Data Analysis
* Cross-platform, works on Windows, Mac and Linux without major changes.
* Used by top tech companies like Google, Netflix and NASA.
* Many Python coding job opportunities in Software Development, Data Science and AI/ML.



**INDENTATION IN PYTHON**

In Python, Indentation is used to define blocks of code. It tells the Python interpreter that a group of statements belongs to a specific block. All statements with the same level of indentation are considered part of the same block. Indentation is achieved using whitespace (spaces or tabs) at the beginning of each line. The most common convention is to use 4 spaces or a tab, per level of indentation.

**EXAMPLE:**



* first **print**statement has no indentation, so it is correctly executed.
* second **print**statement has **tab indentation**, but it doesn't belong to a new block of code. Python expects the indentation level to be consistent within the same block. This inconsistency causes an **Indentation Error**.

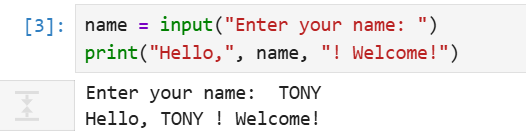
**INPUT & OUTPUT IN PYTHON**

Understanding input and output operations is fundamental to Python programming. With the print() function, we can display output in various formats, while the input() function enables interaction with users by gathering input during program execution.

TAKING INPUT IN PYTHON

Python's **input() function** is used to take user input. By default, it returns the user input in form of a string.

**EXAMPLE:**

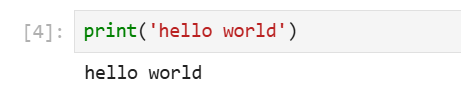


The code prompts the user to input their name, stores it in the variable "name" and then prints a greeting message addressing the user by their entered name.

**Printing Output using print() in Python**

At its core, printing output in Python is straightforward, thanks to the print() function. This function allows us to display text, variables and expressions on the console. Let's begin with the basic usage of the print() function:

**EXAMPLE:**

****

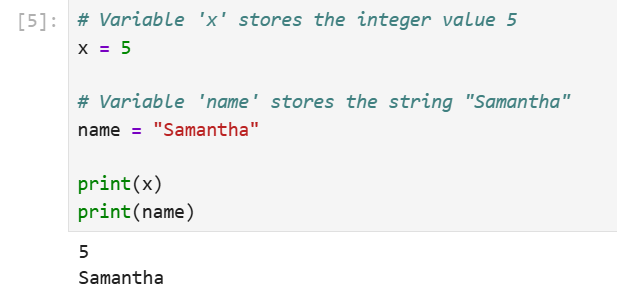
In this example, "Hello, World!" is a string literal enclosed within double quotes. When executed, this statement will output the text to the console.

**PYTHON VARIABLES**

In Python, variables are used to store data that can be referenced and manipulated during program execution. A variable is essentially a name that is assigned to a value. Unlike many other programming languages, Python variables do not require explicit declaration of type. The type of the variable is inferred based on the value assigned.

Variables act as placeholders for data. They allow us to store and reuse values in our program.

**EXAMPLE:**

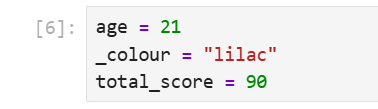
****

**Rules for Naming Variables**

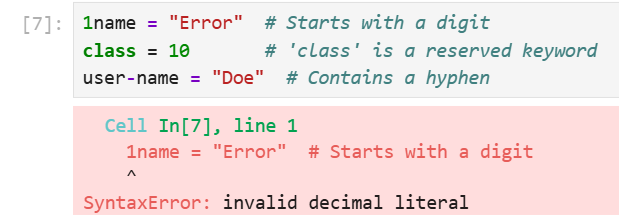
To use variables effectively, we must follow Python’s naming rules:

* Variable names can only contain letters, digits and underscores (\_).
* A variable name cannot start with a digit.
* Variable names are case-sensitive (myVar and myvar are different).
* Avoid using Python keywords (e.g., if, else, for) as variable names.

**VALID EXAMPLE:**

****

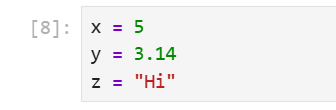
**INVALID EXAMPLE:**

****

**ASSIGNING VALUES TO VARIABLES**

Variables in Python are assigned values using the = operator.

**EXAMPLE:**

****

**DYNAMIC TYPING**

Python variables are dynamically typed, meaning the same variable can hold different types of values during execution.

**EXAMPLE:**

****

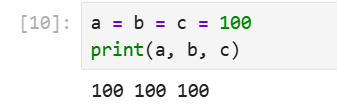
**Multiple Assignments**

Python allows multiple variables to be assigned values in a single line.

**Assigning the Same Value**

Python allows assigning the same value to multiple variables in a single line, which can be useful for initializing variables with the same value.

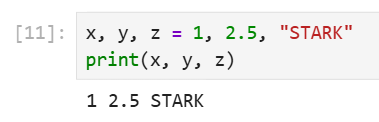
**EXAMPLE:**

****

**Assigning Different Values**

We can assign different values to multiple variables simultaneously, making the code concise and easier to read.

**EXAMPLE:**

****

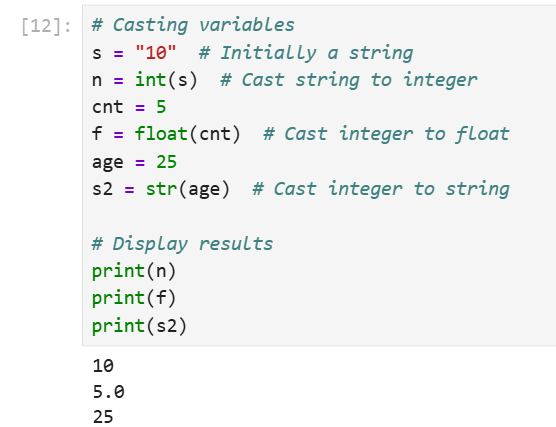
**Type Casting a Variable**

Type casting refers to the process of converting the value of one data type into another. Python provides several built-in functions to facilitate casting, including int(), float() and str() among others.

**Basic Casting Functions**

* **int() -** Converts compatible values to an integer.
* **float() -** Transforms values into floating-point numbers.
* **str() -** Converts any data type into a string.

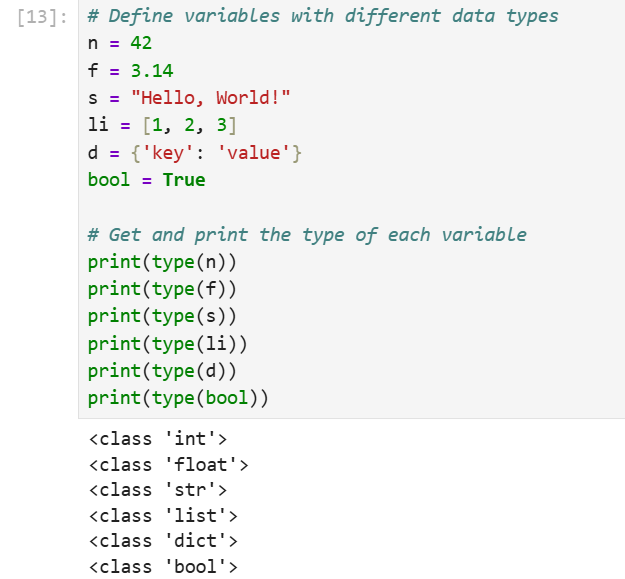
**Examples of Casting:**

****

**Getting the Type of Variable**

In Python, we can determine the type of a variable using the type() function. This built-in function returns the type of the object passed to it.

**EXAMPLE:**



**Scope of a Variable**

In Python, the scope of a variable defines where it can be accessed in the program. There are two main types of scope: local and global.

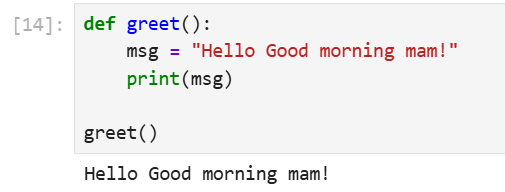
**Local Variables:**

* Defined within a function or block, accessible only inside that scope.
* Destroyed once the function/block ends.
* Temporary, used for short-term data.

**Global Variables:**

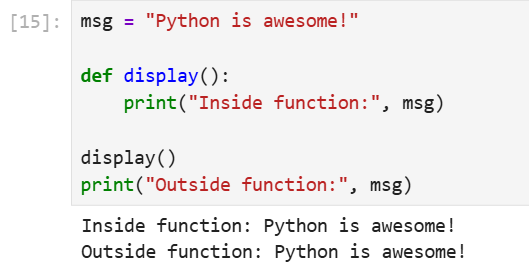
* Defined outside functions, accessible throughout the program.
* To modify within a function, use the global keyword.
* Persist in memory for the program’s duration, useful for shared data.

**EXAMPLE FOR LOCAL VARIABLE:**

****

We define greet() with a local variable msg and print it. Since msg exists only during the function's execution, it's accessed within the function. Calling greet() displays the message.

**EXAMPLE FOR GLOBAL VARIABLE:**

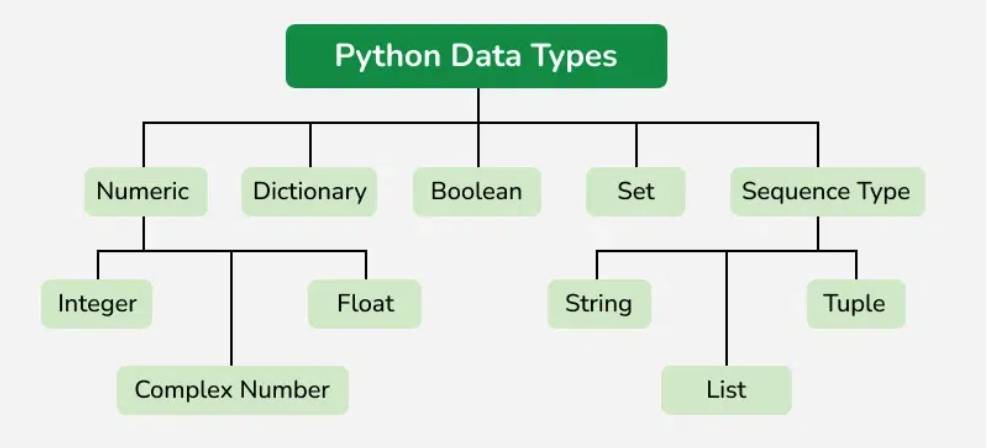
****

msg is a global variable accessible both inside and outside the display() function. Calling display() prints the global msg and printing msg outside the function works as expected.

**PYTHON DATA TYPES**

Python Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Since everything is an object in Python programming, Python data types are classes and variables are instances (objects) of these classes. The following are the standard or built-in data types in Python:

* **Numeric** - int, float, complex
* **Sequence** Type - string, list, tuple
* **Mapping** **Type** - dict
* **Boolean** - bool
* **Set** **Type** - set, frozenset
* **Binary** **Types** - bytes, bytearray, memoryview

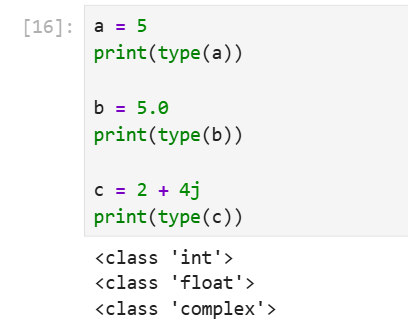


**Numeric Data Types in Python**

The numeric data type in Python represents the data that has a numeric value. A numeric value can be an integer, a floating number, or even a complex number. These values are defined as Python int, Python float and Python complex classes in Python.

* **Integers**- This value is represented by int class. It contains positive or negative whole numbers (without fractions or decimals). In Python, there is no limit to how long an integer value can be.
* **Float** - This value is represented by the float class. It is a real number with a floating-point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
* **Complex Numbers**- A complex number is represented by a complex class. It is specified as *(real part) + (imaginary part) j.* **For example** - 2+3j

**EXAMPLE:**



**Sequence Data Types in Python**

The sequence Data Type in Python is the ordered collection of similar or different Python data types. Sequences allow storing of multiple values in an organized and efficient fashion. There are several sequence data types of Python:

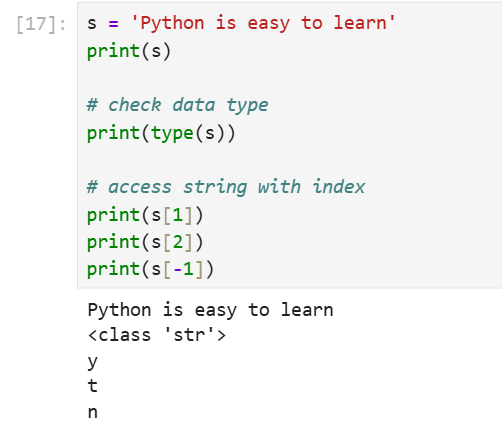
* Python String
* Python List
* Python Tuple

**String Data Type**

Python Strings are arrays of bytes representing Unicode characters. In Python, there is no character data type Python, a character is a string of length one. It is represented by str class.

Strings in Python can be created using single quotes, double quotes or even triple quotes. We can access individual characters of a String using index.

**EXAMPLE:**



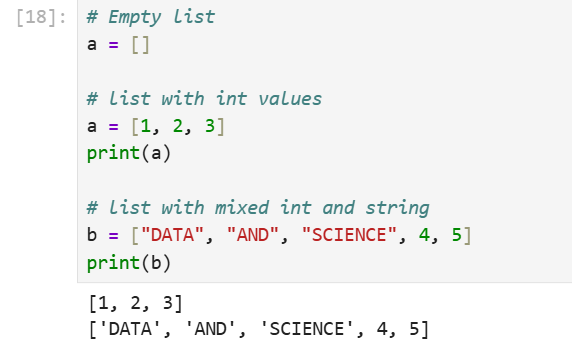
**List Data Type**

Lists are just like arrays, declared in other languages which is an ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

**Creating a List in Python**

Lists in Python can be created by just placing the sequence inside the square brackets[].

**EXAMPLE:**

****

**Tuple Data Type**

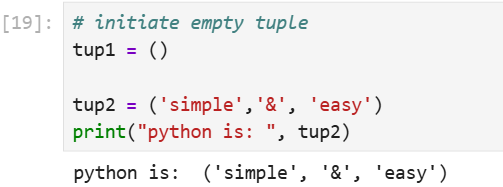
Just like a list, a tuple is also an ordered collection of Python objects. The only difference between a tuple and a list is that tuples are immutable. Tuples cannot be modified after it is created.

**Creating a Tuple in Python**

In Python Data Types, tuples are created by placing a sequence of values separated by a ‘comma’ with or without the use of parentheses for grouping the data sequence. Tuples can contain any number of elements and of any datatype (like strings, integers, lists, etc.).

**Note:** Tuples can also be created with a single element, but it is a bit tricky. Having one element in the parentheses is not sufficient, there must be a trailing ‘comma’ to make it a tuple.

**EXMAPLE:**

****

**Boolean Data Type in Python**

Python Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are falsy (false). However non-Boolean objects can be evaluated in a Boolean context as well and determined to be true or false. It is denoted by the class bool.

**Example:**The first two lines will print the type of the boolean values True and False, which is **<class 'bool'>.**The third line will cause an error, because true is not a valid keyword in Python. Python is case-sensitive, which means it distinguishes between uppercase and lowercase letters.

**Set Data Type in Python**

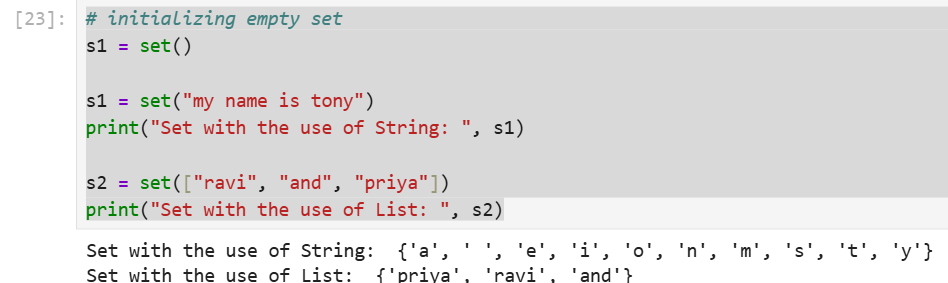
In Python Data Types, Set is an unordered collection of data types that is iterable, mutable, and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

**Create a Set in Python**

Sets can be created by using the built-in set() function with an iterable object or a sequence by placing the sequence inside curly braces, separated by a **‘comma’.**The type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

**Example:**The code is an example of how to create sets using different types of values, such as **strings,** **lists,** and mixed values





**Dictionary Data Type**

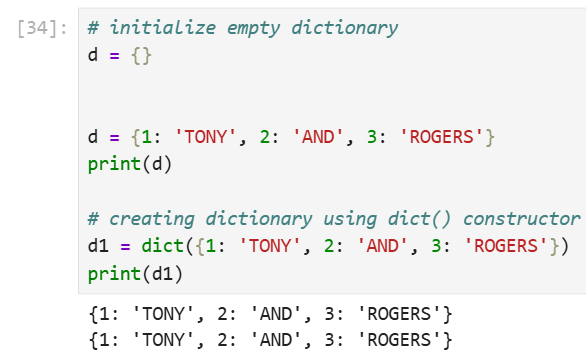
A dictionary in Python is a collection of data values, used to store data values like a map, unlike other Python Data Types that hold only a single value as an element, a Dictionary holds a key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon : , whereas each key is separated by a ‘comma’.

**Create a Dictionary in Python**

Values in a dictionary can be of any datatype and can be duplicated, whereas keys can’t be repeated and must be immutable. The dictionary can also be created by the built-in function **dict().**

**Note** - Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

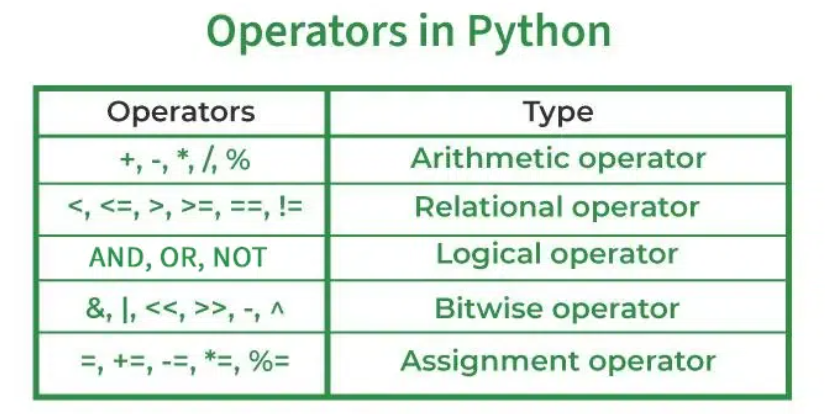
**EXAMPLE:**

****

**PYTHON OPERATORS**

In Python programming, Operators in general are used to perform operations on values and variables. These are standard symbols used for logical and arithmetic operations. In this article, we will look into different types of Python operators.

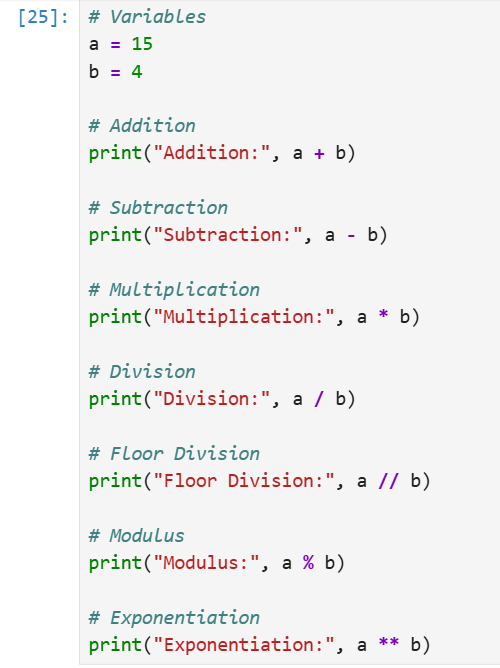
* OPERATORS: These are the special symbols. Eg- + , \* , /, etc.
* OPERAND: It is the value on which the operator is applied.

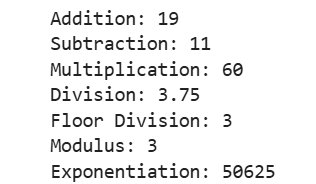


**Arithmetic Operators in Python**

Python Arithmetic operators are used to perform basic mathematical operations like**addition, subtraction, multiplication** and **division**.

**EXAMPLE:**

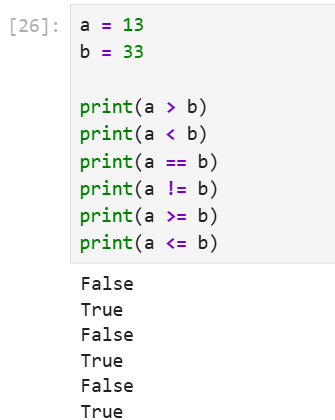




**Comparison of Python Operators**

In Python Comparisonof Relational operators compares the values. It either returns **True** or **False** according to the condition.

**Example of Comparison Operators in Python:**

****

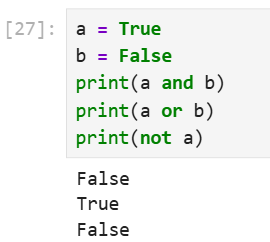
**Logical Operators in Python**

Python Logical operators perform Logical AND, Logical OR and Logical NOT operations. It is used to combine conditional statements.

**The precedence of Logical Operators in Python is as follows:**

1. Logical not
2. logical and
3. logical or

**EXAMPLE:**



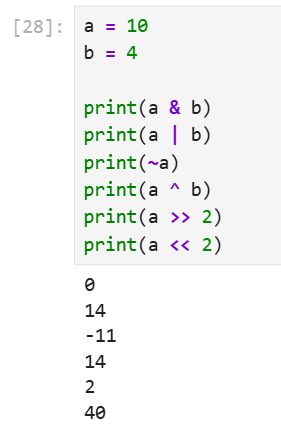
**Bitwise Operators in Python**

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

Bitwise Operators in Python are as follows:

1. Bitwise NOT
2. Bitwise Shift
3. Bitwise AND
4. Bitwise XOR
5. Bitwise OR

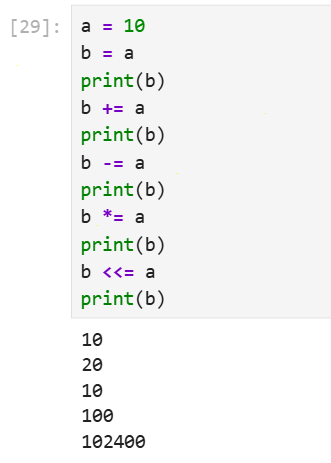
**EXAMPLE:**

****

**Assignment Operators in Python**

Python Assignment operators are used to assign values to the variables. This operator is used to assign the value of the right side of the expression to the left side operand.

**EXAMPLE:**

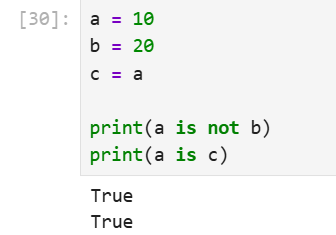
****

**Identity Operators in Python**

In Python, is and is not are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal do not imply that they are identical.

1. is True if the operands are identical
2. is not True if the operands are not identical

**EXAMPLE:**

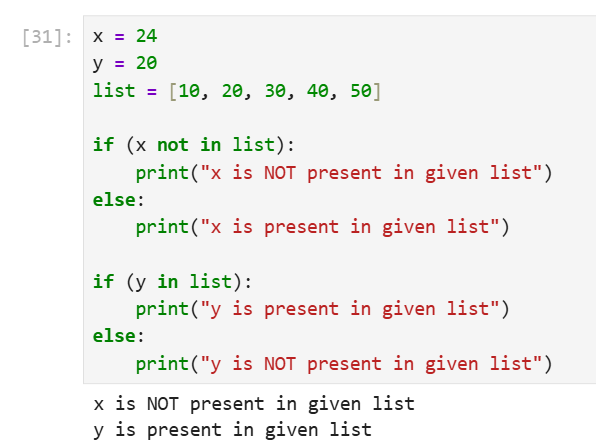
****

**Membership Operators in Python**

In Python, in and not in are the membership operators that are used to test whether a value or variable is in a sequence.

1. in True if value is found in the sequence
2. not in True if value is not found in the sequence

**EXAMPLE:**

****

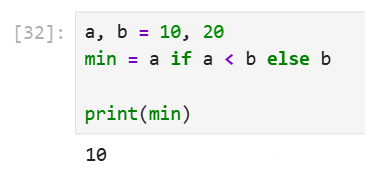
**Ternary Operator in Python**

in Python, Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false. It was added to Python in version 2.5.

It simply allows testing a condition in a single line replacing the multiline if-else making the code compact.

**Syntax:**[on true] if [expression] else [on false]

**EXAMPLE:**

****

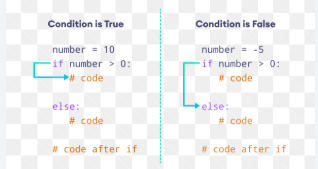
**CONDITIONAL STATEMENTS**

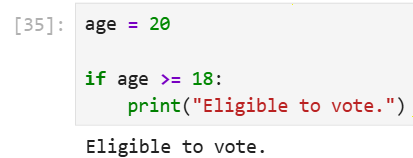
Conditional statements in Python are used to execute certain blocks of code based on specific conditions. These statements help control the flow of a program, making it behave differently in different situations.

**If Conditional Statement in Python**

If statement is the simplest form of a conditional statement. It executes a block of code if the given condition is true.

**EXAMPLE:**

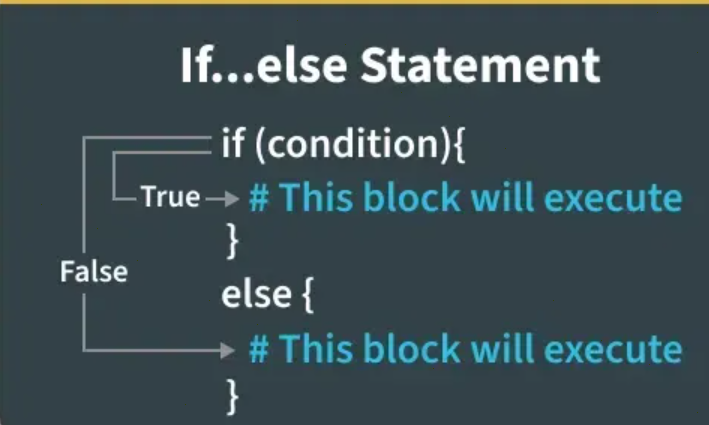
****

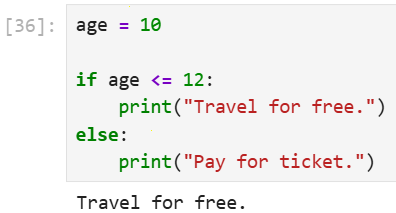
****

**If else Conditional Statements in Python**

Else allows us to specify a block of code that will execute if the condition(s) associated with an if or elif statement evaluates to False. Else block provides a way to handle all other cases that don't meet the specified conditions.

**EXAMPLE:**

****

****

**LOOPS IN PYTHON (FOR, WHILE, NESTED)**

Loops in Python are used to repeat actions efficiently. The main types are For loops (counting through items) and While loops (based on conditions). In this article, we will look at Python loops and understand their working with the help of examples.

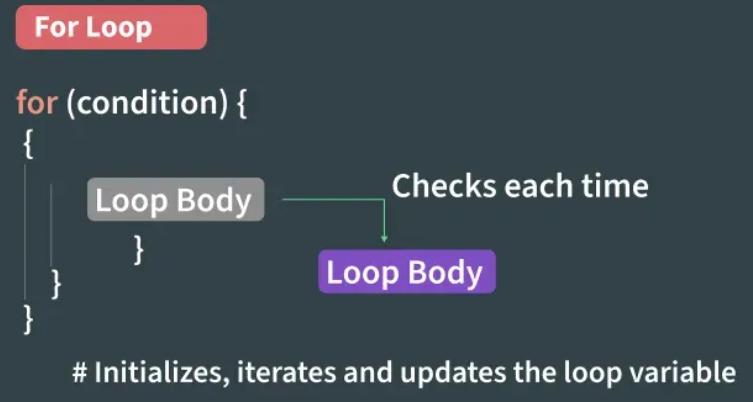
**For Loop in Python**

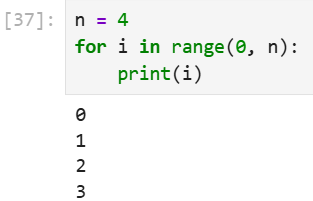
For loops is used to iterate over a sequence such as a list, tuple, string or range. It allow to execute a block of code repeatedly, once for each item in the sequence.

**Syntax:**

for iterator\_var in sequence:  
statements(s)

**EXAMPLE:**

****

****

This code prints the numbers from 0 to 3 (inclusive) using a for loop that iterates over a range from 0 to n-1 (where n = 4).

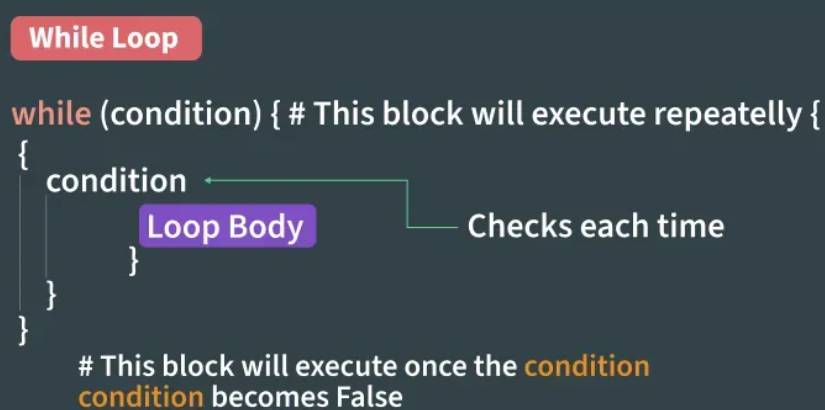
**While Loop in Python**

In Python, a while loop is used to execute a block of statements repeatedly until a given condition is satisfied. When the condition becomes false, the line immediately after the loop in the program is executed.

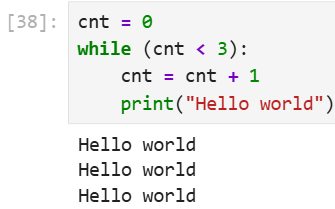
**Syntax:**

while expression:  
statement(s)

**EXAMPLE:**

****

All the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements.



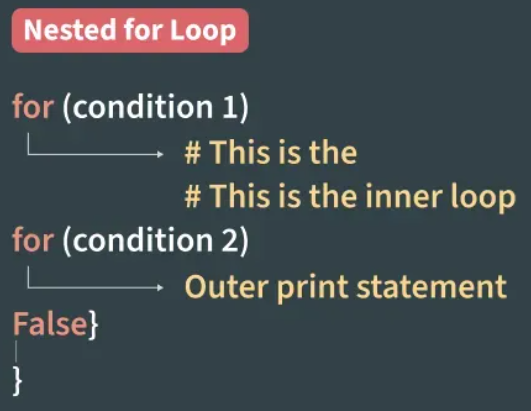
**Nested Loops in Python**

Python programming language allows to use one loop inside another loop which is called nested loop. Following section shows few examples to illustrate the concept.

**Syntax:**

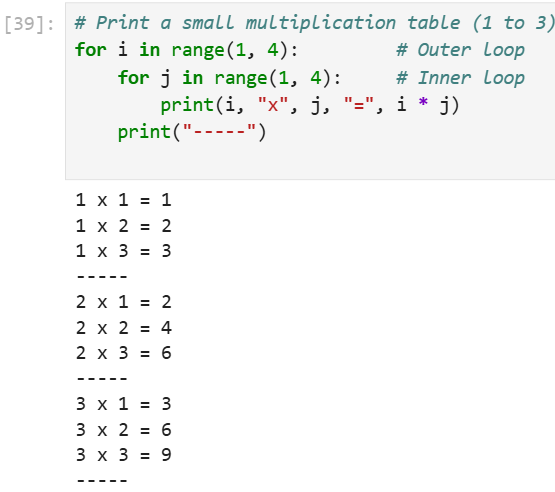
for iterator\_var in sequence:  
for iterator\_var in sequence:  
statements(s)  
statements(s)

**EXAMPLE:**

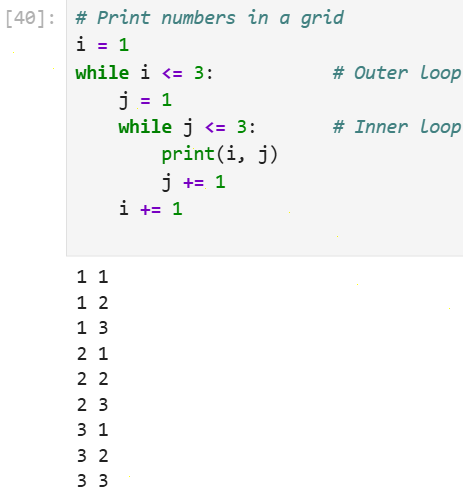
****



**EXAMPLE FOR NESTED FOR LOOP:**

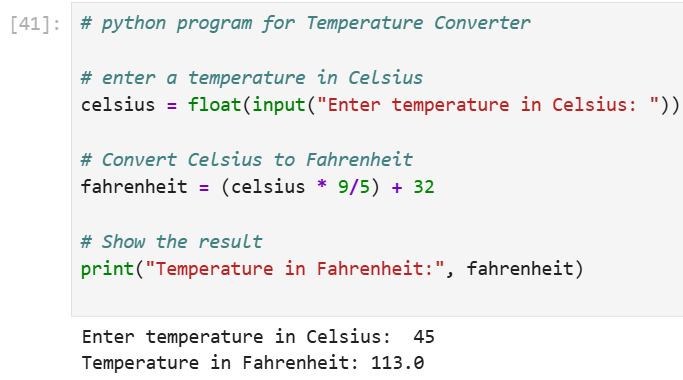


**EXAMPLE FOR NESTED WHILE LOOP:**



**BASIC PYTHON PROGRAMS:**

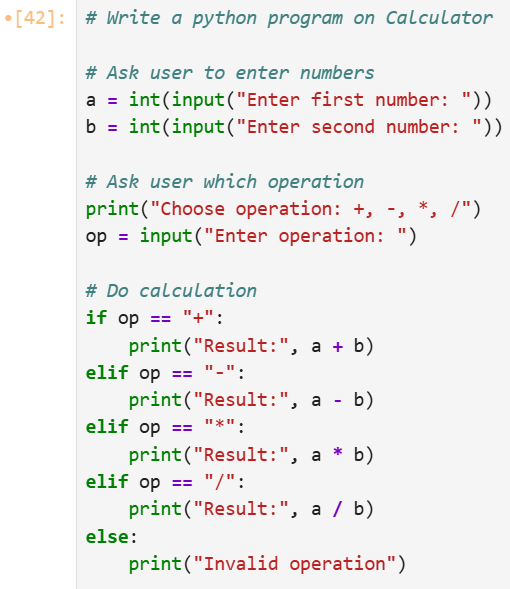
1. **Python program for temperature converter?**

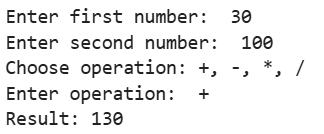
****

**EXPLANATION:**

* + input() → takes input from the user.
  + float() → converts the input into a number (decimal allowed).
  + Formula → Fahrenheit = (Celsius \* 9/5) + 32
  + Multiply Celsius by 1.8 (or 9/5).
  + Add 32 to shift to Fahrenheit scale.
  + print() → shows the result on the screen.

1. **Python program for calculator?**





**EXPPLANATION:**

* **Input numbers** – input() takes values from the user. int() changes them into numbers.
* **Choose operation** – user selects one of +, -, \*, /.
* **If/elif/else** – program checks which operation was chosen and performs it.
* **Operators**
* + → addition
* - → subtraction
* \* → multiplication
* / → division (always gives a decimal result in Python 3)
* **Else** – if user types something else, program shows “Invalid operation”.