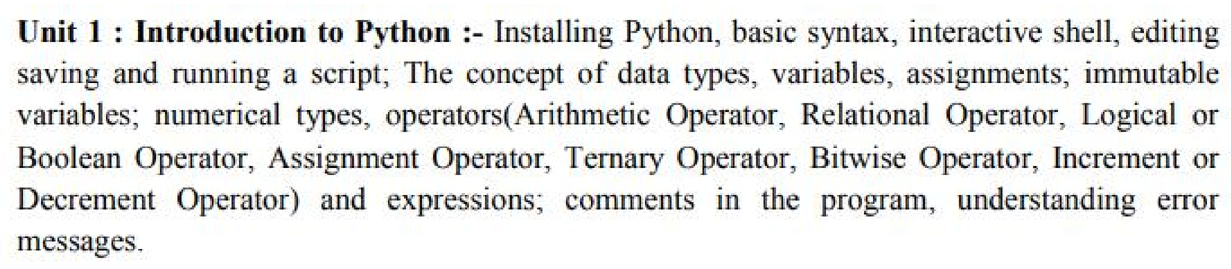

Unit 1 : Introduction to Python :- Installing Python, basic syntax, interactive shell, editing saving and running a script; The concept of data types, variables, assignments; immutable variables; numerical types, operators(Arithmetic Operator, Relational Operator, Logical or Boolean Operator, Assignment Operator, Ternary Operator, Bitwise Operator, Increment or Decrement Operator) and expressions; comments in the program, understanding error messages.

Installing Python

1. Download the Python Installer:

   - Open a web browser and navigate to the official Python website: python.org/downloads.
   - Locate the latest stable release of Python 3 (e.g., Python 3.13.0).
   - Click on the appropriate Windows installer (e.g., "Windows installer (64-bit)") to download the executable file (.exe).

2. Run the Installer:

   - Once the download is complete, locate the downloaded .exe file and double-click it to launch the installer.
   - **Crucially, ensure you check the box that says "Add Python 3.x to PATH":** (where 3.x represents the Python version you are installing). This step is vital as it allows you to run Python commands directly from the command prompt.
   - You can choose "Install Now" for a standard installation or "Customize installation" if you want to select specific features or change the installation location. For most users, "Install Now" is sufficient.

3. Complete the Installation:

   - Follow the on-screen prompts, accepting any license agreements and allowing the installer to proceed.
   - The installation process may take a few minutes.
   - Once the installation is complete, you may see an option to "Disable path length limit." While not strictly necessary for basic use, it is recommended for Python development to prevent issues with long file paths.

4. Verify the Installation:

- Open a new Command Prompt or PowerShell window.
- Type python --version and press Enter.
- If Python is installed correctly, the command prompt will display the installed Python version (e.g., Python 3.13.0). This confirms that Python is successfully installed and accessible from your system's PATH

Basic Python Syntax:

Learn all the basic Python syntaxes you need to start coding. This guide covers comments, variables, functions, loops, and more — explained simply for beginners.

```
# This is a comment
print("Hello, World!")
def greet(name):
return f"Hello, {name}"
for i in range(5):
```

Every programming language has a unique syntax. Some languages borrow syntax from others, while others create something wholly different. No matter the language you intend to use, you have to understand its syntax; otherwise, you’ll struggle to get anything done.

Syntax is a set of rules that define how code is written in a particular language. Some of the key elements of a language’s syntax include:

- **Keywords** are reserved words in the programming language, such as “if,” “else,” and “for.”
- **Variables** are used to store data values and have names, types, and scopes.
- **Control Flow Statements** determine the sequence of execution in a program.
- **Indentation** determines code blocks.
- **Comments** make it possible to document your code inline.

Basic Python Syntax

Let’s take a look at basic Python syntax, which is comprised of the same items above with the addition of:

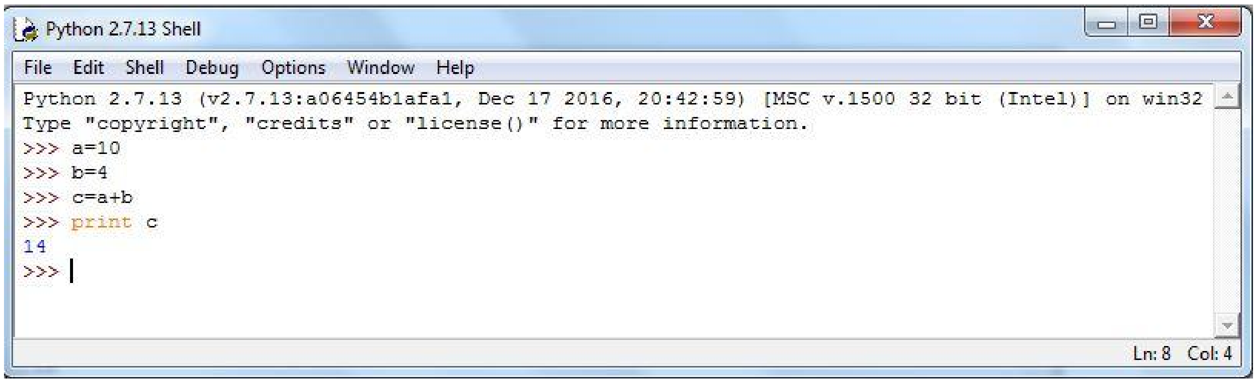
- Functions
- Data types
- Classes and Objects
- Lists, Tuples, and Dictionaries

Here’s a basic Python script that shows most of the elements that make up the basic syntax:

# Variable Declaration and Assignment
x = 5 # Integer variable assignment with ‘=’ symbol. The value of x is set to 5.
y = “Hello!” # String variable assignment – Strings are defined using the quotes “” around text content!

Interactive shell

The Python interactive shell, also known as the Python interpreter or IDLE (Integrated Development and Learning Environment), provides a command-line interface for interacting with Python. It allows users to execute Python code line by line and receive immediate feedback.



```
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:42:59) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> a=10
>>> b=4
>>> c=a+b
>>> print c
14
>>> 
```

Editing, Saving, and Running a Script

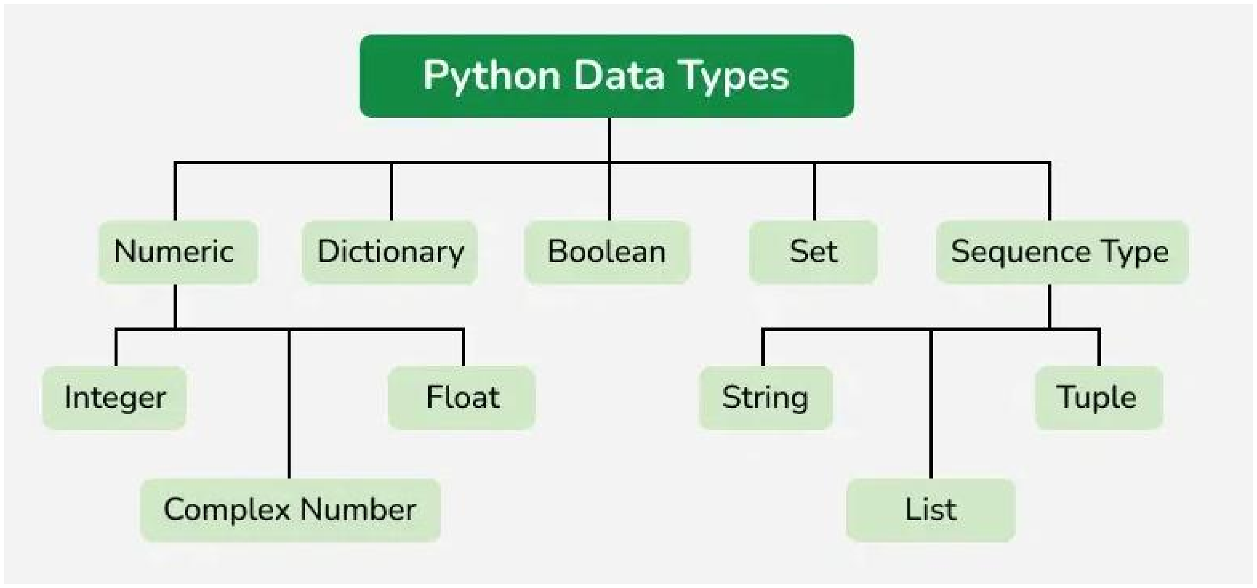
- A script is a Python program saved with a .py extension.
- Steps:

1. Open any editor (Notepad, VS Code, PyCharm).
2. Write Python code.
3. Save as filename.py.
4. Run with python filename.py

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



DataTypes

This code assigns variable **'x'** different values of few Python data types – **int, float, list, tuple and string.** Each assignment replaces the previous value, making **'x'** take on the data type and value of the most recent assignment.

```
#  int, float, string, list and set
x = 50
x = 60.5
x = "Hello World"
x = ["geeks", "for", "geeks"]
x = ("geeks", "for", "geeks")
```

1. 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+3ja = 5

```
print(type(a))

b = 5.0
print(type(b))

c = 2 + 4j
print(type(c))
```

Output

```
<class 'int'>

<class 'float'>

<class 'complex'>
```

2. 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.

```
s = 'Welcome to the Geeks World'
print(s)

# check data type
print(type(s))

# access string with index
print(s[1])
print(s[2])
print(s[-1])
```

Output

```
Welcome to the Geeks World
<class 'str'>
e
l
d
```

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[].

```
# Empty list
a = []

# list with int values
a = [1, 2, 3]
print(a)

# list with mixed int and string
b = ["Geeks", "For", "Geeks", 4, 5]
print(b)
```

Output

```
[1, 2, 3]
['Geeks', 'For', 'Geeks', 4, 5]
```

Access List Items

In order to access the list items refer to the index number. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

```
a = ["Geeks", "For", "Geeks"]
```

```python
print("Accessing element from the list")
print(a[0])
print(a[2])

print("Accessing element using negative indexing")
print(a[-1])
print(a[-3])
```

Output

```
Accessing element from the list

Geeks

Geeks

Accessing element using negative indexing

Geeks

Geeks
```

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*

```python
# initiate empty tuple
tup1 = ()

tup2 = ('Geeks', 'For')
print("\nTuple with the use of String: ", tup2)
```

Output

```
Tuple with the use of String:  ('Geeks', 'For')
```

Note *- The creation of a Python tuple without the use of parentheses is known as Tuple Packing.*

Access Tuple Items

In order to access the tuple items refer to the index number. Use the index operator [ ] to access an item in a tuple.

```
tup1 = tuple([1, 2, 3, 4, 5])

# access tuple items
print(tup1[0])
print(tup1[-1])
print(tup1[-3])
```

Output

```
1

5

3
```

3. 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.

```
print(type(True))
print(type(False))
print(type(true))
```

Output:

```
<class 'bool'>
<class 'bool'>
Traceback (most recent call last):
  File "/home/7e8862763fb66153d70824099d4f5fb7.py", line 8, in
    print(type(true))
NameError: name 'true' is not defined
```

4. 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

```
# initializing empty set
s1 = set()

s1 = set("GeeksForGeeks")
print("Set with the use of String: ", s1)

s2 = set(["Geeks", "For", "Geeks"])
print("Set with the use of List: ", s2)
```

Output

```
Set with the use of String:  {'s', 'o', 'F', 'G', 'e', 'k', 'r'}

Set with the use of List:  {'Geeks', 'For'}
```

Access Set Items

Set items cannot be accessed by referring to an index, since sets are unordered the items have no index. But we can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the in the keyword.

```
set1 = set(["Geeks", "For", "Geeks"])
print(set1)

# loop through set
for i in set1:
    print(i, end=" ")

# check if item exist in set
print("Geeks" in set1)
```

Output

```
{'Geeks', 'For'}

Geeks For True
```

5. 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.*

```
# initialize empty dictionary
d = {}


d = {1: 'Geeks', 2: 'For', 3: 'Geeks'}
print(d)

# creating dictionary using dict() constructor
d1 = dict({1: 'Geeks', 2: 'For', 3: 'Geeks'})
print(d1)
```

Output

```
{1: 'Geeks', 2: 'For', 3: 'Geeks'}

{1: 'Geeks', 2: 'For', 3: 'Geeks'}
```

Accessing Key-value in Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. Using **get() method** we can access the dictionary elements.

```
d = {1: 'Geeks', 'name': 'For', 3: 'Geeks'}

# Accessing an element using key
```

```
print(d['name'])

# Accessing a element using get
print(d.get(3))
```

Output

```
For

Geeks
```

Python Data Type Exercise Questions

Below are two exercise questions on Python Data Types. We have covered list operation and tuple operation in these exercise questions. For more exercises on Python data types visit the page mentioned below.

Q1. Code to implement basic list operations

```
fruits = ["apple", "banana", "orange"]
print(fruits)
fruits.append("grape")
print(fruits)
fruits.remove("orange")
print(fruits)
```

Output

```
['apple', 'banana', 'orange']

['apple', 'banana', 'orange', 'grape']

['apple', 'banana', 'grape']
```

Q2. Code to implement basic tuple operation

```
coordinates = (3, 5)
print(coordinates)
print("X-coordinate:", coordinates[0])
print("Y-coordinate:", coordinates[1])
```

Output

```
(3, 5)

X-coordinate: 3

Y-coordinate: 5
```

Introduction Variables

Variables are used in programming to store and manipulate data. In Python, variables are crucial for shaping code structures. Developers use variables in Python to interact with data and make programs robust and adaptable.

This tutorial will explore Python variables, their rules, types, and more with the help of examples.

What Are Variables in Python?
In Python, we use variables as containers to store data values like numbers, text, or complex structures. We don’t need to declare a variable’s type, just assign a value, and Python handles the rest. This makes programming flexible and beginner-friendly.

A Python variable points to a memory location where the assigned data is stored. We can change its value anytime during the program. Variables improve code readability and are essential for building efficient, dynamic applications.

Rules for Python Variables
We follow rules for naming Python variables so that our code is easy to read and doesn’t break. These rules help Python understand what we’re trying to do.

Python variables must start with a letter or an underscore (e.g., _name, age).

We can not start variable names with a number.

Use only letters, numbers, and underscores (no spaces or special characters).

Don’t use Python keywords like if, class, or while as variable names.

Variable names are case-sensitive (Age and age are different).

Choose names that make sense (like total_price, not tp).

If we follow these simple rules, we can avoid errors and write clean, bug-free code with ease.

Examples Showing Python Variable Rules
Invalid Example:
```
2name = "Varun"
student age = 14
if = True
```

In this code:

2name starts with a number (invalid)

student age contains a space (invalid)

if is a Python keyword (invalid)

Valid Example:

```
name2 = "Varun"
student_age = 14
is_enrolled = True

print(name2, student_age, is_enrolled)
```

Now, all variable names follow Python rules and will run without errors.

Variables Assignment

Variable assignment means storing a value inside a variable using the = operator. Python variables can hold numbers, text, or any type of data.

Declaration and Initialization of Variables

It’s easy for us to create a variable in Python, and it is a dynamically typed language, so we don’t need to declare the type. A variable is created as soon as we assign it a value. Here’s how we can declare and define a variable in Python. We have also shown how to print Python variables.

Example:

```
name = "Varun"
age = 25
print(name)
print(age)
```

Run Code

Output:

```
Varun
25
```

Redeclaring Variables in Python

We can redeclare a variable in Python even after it’s already defined. Python is a dynamically typed language, so we’re free to give a variable a new value or even change its type.

Example:

```
item_count = 10     # initial value
print(item_count)

item_count = 5      # updated value
print(item_count)
```

Run Code

```
Output:

10
5
```

Assigning Values to Multiple Variables in Python
We can assign a single value to multiple variables in one line using (=), which makes the code concise and readable.

Example:

```
a = b = c = 100 # assign multiples variable
print(a)
print(b)
print(c)
```
Run Code

Output:

```
100
100
100
```

Assigning Different Values to Multiple Variables
We can assign different values to multiple variables in a single line using commas. This keeps our code short, clean, and easy to read and manage.

Example:

```
name, age, city = "Amit", 28, "Delhi"  # assign values to all three
variables at once

print(name)
print(age)
print(city)
```
Run Code

Output:

```
Amit
28
Delhi
```

Immutable Variables

- Variables that **cannot be changed** once created.
- Example: int, float, str, tuple are immutable.

```
x = "hello"
x[0] = "H" # Error (string is immutable)
```

Operators in Python

In Python, operators are **special symbols or keywords** that carry out operations on values and python variables.They serve as a basis for expressions, which are used to modify data and execute computations.Python contains several operators, each with its unique purpose.

Types of Python Operators

Python language supports various types of operators, which are:

1. **Arithmetic Operators**
2. **Comparison (Relational) Operators**
3. **Assignment Operators**
4. **Logical Operators**
5. **Bitwise Operators**
6. **Membership Operators**
7. **Identity Operators**

1. Python Arithmetic Operators

- Mathematical operations including addition, subtraction, multiplication, and division are commonly carried out using Python arithmetic operators.
- They are compatible with integers, variables, and expressions.
- In addition to the standard arithmetic operators, there are operators for modulus, exponentiation, and floor division.

Operator	**Name**	**Example**
+	Addition	10 + 20 = 30
-	Subtraction	20 – 10 = 10
*	Multiplication	10 * 20 = 200
/	Division	20 / 10 = 2
%	Modulus	22 % 10 = 2

**	Exponent	4**2 = 16
//	Floor Division	9//2 = 4

Example of Python Arithmetic Operators in Python Compiler

```
    a = 21
b = 10
# Addition
print ("a + b : ", a + b)
# Subtraction
print ("a - b : ", a - b)
# Multiplication
print ("a * b : ", a * b)
# Division
print ("a / b : ", a / b)
# Modulus
print ("a % b : ", a % b)
# Exponent
print ("a ** b : ", a ** b)
# Floor Division
print ("a // b : ", a // b)
Run Code >>
```

This code defines the two variables "a" and "b." It then applies several arithmetic operations to them (including addition, subtraction, multiplication, division, modulus, exponentiation, and floor division) and outputs the results.

Output

```
a + b : 31
a - b : 11
a * b : 210
a / b : 2.1
a % b : 1
a ** b : 16679880978201
a // b : 2
```

2. Python Comparison Operators

- To compare two values, Python comparison operators are needed.
- Based on the comparison, they produce a Boolean value (True or False).

Operator	Name	Example
==	Equal	4 == 5 is not true.
!=	Not Equal	4 != 5 is true.
>	Greater Than	4 > 5 is not true
<	Less Than	4 < 5 is true
>=	Greater than or Equal to	4 >= 5 is not true.
<=	Less than or Equal to	4 <= 5 is true.

Example of Python Comparison Operators

```
        a = 4
b = 5
# Equal
print ("a == b : ", a == b)
# Not Equal
print ("a != b : ", a != b)
# Greater Than
print ("a > b : ", a > b)
# Less Than
print ("a < b : ", a < b)
# Greater Than or Equal to
print ("a >= b : ", a >= b)
# Less Than or Equal to
print ("a <= b : ", a <= b)
Run Code >>
```

This code compares the values of python variables 'a' and 'b' and prints if they are equal, not equal, greater than, less than, more than or equal to, and less than or equal to each other.

Output

```
a == b : False
a != b : True
a > b : False
a < b : True
a >= b : False
```

```
a <= b : True
```

3. Python Assignment Operators

- Python assignment operators are used to assign values to variables in Python.
- The single equal symbol (=) is the most fundamental assignment operator.
- It assigns the value on the operator's right side to the variable on the operator's left side.

Operator	**Name**	**Example**
=	Assignment Operator	a = 10
+=	Addition Assignment	a += 5 (Same as a = a + 5)
-=	Subtraction Assignment	a -= 5 (Same as a = a - 5)
*=	Multiplication Assignment	a *= 5 (Same as a = a * 5)
/=	Division Assignment	a /= 5 (Same as a = a / 5)
%=	Remainder Assignment	a %= 5 (Same as a = a % 5)
**=	Exponent Assignment	a **= 2 (Same as a = a ** 2)
//=	Floor Division Assignment	a //= 3 (Same as a = a // 3)

Example of Python Assignment Operators

```
        # Assignment Operator
a = 10
# Addition Assignment
a += 5
print ("a += 5 : ", a)
# Subtraction Assignment
a -= 5
print ("a -= 5 : ", a)
```

```
# Multiplication Assignment
a *= 5
print ("a *= 5 : ", a)
# Division Assignment
a /= 5
print ("a /= 5 : ",a)
# Remainder Assignment
a %= 3
print ("a %= 3 : ", a)
# Exponent Assignment
a **= 2
print ("a **= 2 : ", a)
# Floor Division Assignment
a //= 3
print ("a //= 3 : ", a)
Run Code >>
```

The Python assignment operators are shown in this code in the **Python Editor**. It begins with the value of 'a' equal to 10, and then goes through the steps of addition, subtraction, multiplication, division, remainder, exponentiation, and floor division, updating 'a' as necessary and outputting the results.

Output

```
a += 5 : 105
a -= 5 : 100
a *= 5 : 500
a /= 5 : 100.0
a %= 3 : 1.0
a **= 2 : 1.0
a //= 3 : 0.0
```

4. Python Bitwise Operators

- Python bitwise operators execute operations on individual bits of binary integers.
- They work with integer binary representations, performing logical operations on each bit location.
- Python includes various bitwise operators, such as AND (&), OR (|), NOT (), XOR (), left shift (), and right shift (>>).

Operator	**Name**	**Example**
&	Binary AND	Sets each bit to 1 if both bits are 1
|	Binary OR	Sets each bit to 1 if one of the two bits is

		1
^	Binary XOR	Sets each bit to 1 if only one of two bits is 1
~	Binary Ones Complement	Inverts all the bits
~	Binary Ones Complement	Inverts all the bits
<<	Binary Left Shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Binary Right Shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Example of Python Bitwise Operators

```
        a = 60 # 60 = 0011 1100
b = 13 # 13 = 0000 1101
# Binary AND
c = a & b # 12 = 0000 1100
print ("a & b : ", c)
# Binary OR
c = a | b # 61 = 0011 1101
print ("a | b : ", c)
# Binary XOR
c = a ^ b # 49 = 0011 0001
print ("a ^ b : ", c)
# Binary Ones Complement
c = ~a; # -61 = 1100 0011
print ("~a : ", c)
# Binary Left Shift
c = a << 2; # 240 = 1111 0000
print ("a << 2 : ", c)
# Binary Right Shift
c = a >> 2; # 15 = 0000 1111
print ("a >> 2 : ", c)
Run Code >>
```

The binary representations of the numbers 'a and b' are subjected to bitwise operations in this code. It displays the results of binary AND, OR, XOR, Ones Complement, Left Shift, and Right Shift operations.

Output

```
a & b : 12
a | b : 61
a ^ b : 49
~a : -61
a >> 2 : 240
a >> 2 : 15
```

5. Python Logical Operators

- Python logical operators are used to compose Boolean expressions and evaluate their truth values.
- They are required for the creation of conditional statements as well as for managing the flow of execution in programs.
- Python has three basic logical operators: AND, OR, and NOT.

Operator	**Description**	**Example**
and Logical AND	If both of the operands are true then the condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands is non-zero then the condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand	Not(a and b) is false.

Example of Python Logical Operators in Python Online Editor

```
        x = 5
y = 10
if x > 3 and y < 15:
  print("Both x and y are within the specified range")
Run Code >>
```

The code assigns the values 5 and 10 to variables x and y. It determines whether x is larger than 3 and y is less than 15. If both conditions are met, it writes "Both x and y are within the specified range."

Output

```
Both x and y are within the specified range
```

6. Python Membership Operators

- Python membership operators are used to determine whether or not a certain value occurs within a sequence.
- They make it simple to determine the membership of elements in various Python data structures such as lists, tuples, sets, and strings.
- Python has two primary membership operators: the in and not in operators.

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Example of Python Membership Operators

```
        fruits = ["apple", "banana", "cherry"]
if "banana" in fruits:
    print("Yes, banana is a fruit!")
else:
    print("No, banana is not a fruit!")
Run Code >>
```

The code defines a list of fruits and tests to see if the word "banana" appears in the list. If it is, the message "Yes, banana is a fruit!" is displayed; otherwise, the message "No, banana is not a fruit!" is displayed.

Output

```
Yes, banana is a fruit!
```

7. Python Identity Operators

- Python identity operators are used to compare two objects' memory addresses rather than their values.
- If the two objects refer to the same memory address, they evaluate to True; otherwise, they evaluate to False.
- Python includes two identity operators: the is and is not operators.

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise	x is y, here are results in 1 if id(x) equals id(y)
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise	x is not y, there are no results in 1 if id(x) is not equal to id(y).

Example of Python Identity Operators

```
        x = 10
y = 5
if x is y:
    print("x and y are the same object")
else:
    print("x and y are not the same object")
Run Code >>
```

The code sets the variables x and y to 10 and 5, respectively. It then uses the is keyword to determine whether x and y relate to the same item in memory. If they are, it displays "x and y are the same object"; otherwise, it displays "x and y are not the same object."

Output

```
x and y are not the same object
```

Python Increment Operator (+=)

In Python, we can achieve incrementing by using Python '+=' operator. This operator adds the value on the right to the variable on the left and assigns the result to the variable. In this section, we will see how use Increment Operator in Python. We don't write things like:

```
for (int i = 0; i < 5; ++i)
```

For normal usage, instead of i++, if you are increasing the count, you can use

```
i+=1 or i=i+1
```

In this example, a variable x is initialized with the value 5. The += operator is then used to increment the variable by 1, and the result is displayed, showcasing a concise way to perform the increment operation in Python.

```python
# Initializing a variable
x = 5

# Incrementing the variable by 1
# Equivalent to x = x + 1
x += 1

# Displaying the result
print("Incremented value:", x)
```

Output

```
Incremented value: 6
```

Python Decrement Operator (-=)

We do not have a specific decrement operator in Python (like -- in some other programming languages). However, you can achieve decrementing a variable using the -= operator. This operator subtracts the value on the right from the variable on the left and assigns the result to the variable.
For normal usage, instead of i--, if you are increasing the count, you can use

```
i-=1 or i=i-1
```

```python
# Initializing a variable
x = 10

# Decrementing the variable by 1
# Equivalent to x = x - 1
x -= 1

# Displaying the result
print("Decremented value:", x)
```

Output

```
Decremented value: 9
```

Comments in the Program

- Comments are notes written for humans, not executed by Python.
- Single-line comment → `#`
- Multi-line comment → `''' ... '''`
- `# This is a single-line comment`
- `'''`
- `This is`
- `a multi-line comment`
  `'''`

Understanding Error Messages

- Errors tell us what went wrong in the code.
- Example:
- `print(Hello)`

❌ Error: NameError: name 'Hello' is not defined

✔ Fix: print("Hello")