Python Basics

Python Syntax - It is like grammar for this programming language. Syntax refers to the set of rules that defines how to write and organize code so that the Python Interpreter can understand and run it correctly.

Python Indentation - It refers to the use of whitespace (spaces or tabs) at the beginning of code line. It is used to define the code blocks. Indentation is crucial in Python because, unlike many other programming languages that use braces "{}" to define blocks, Python uses indentation. It improves the readability of Python code, but on other hand it became difficult to rectify indentation errors. Even one extra or less space can leads to indentation error.

Variables - These are essentially named references pointing to objects in memory. Unlike some other languages, you don't need to declare a variable's type explicitly in Python. Based on the value assigned, Python will dynamically determine the type.

```
# Variable -
# Different steps of variable- Declaration, defination and initialization in
single step
```

Ex:- a = “door”
print(type(a))

b = 'Goal'
print(type(b))

```
d=2 # 2 is value and assigning to variable d
# type() function is used to identify the datatype of variable
print(d, type(d))
# id is used to identify the memory location of the variable.
```

[Note: variable 'a' is initialize with integer value and variable 'b' with a string. Because of dynamic-types behavior, data type will be decide during runtime.]

Python Identifiers – They are unique names that are assigned to variables, functions, classes, and other entities. They are used to uniquely identify the entity within the program. They should start with a letter (a-z, A-Z) or an underscore "_" and can be followed by letters, numbers, or underscores.

In the below example "first_name" is an identifier that store string value.
first_name = "Roll"

Rules for declaring a variables

- Identifiers can be composed of alphabets (either uppercase or lowercase), numbers (0-9), and the underscore character (_). They shouldn't include any special characters or spaces.
- The starting character of an identifier must be an alphabet or an underscore.
- Within a specific scope or namespace, each identifier should have a distinct name to avoid conflicts. However, different scopes can have identifiers with the same name without interference.
- Keywords can’t be used.

[Note: 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.]

Assigning Values to Variables

Basic Assignment
Variables in Python are assigned values using the = (assignment operator)

```
x = 5
y = 3.14
z = "Hi"
```

Dynamic Typing

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

```
x = 10
x = "Now a string"
```

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.

```
a = b = c = 100
print(a, b, c)
```

Assigning Different Values

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

```
x, y, z = 1, 2.5, "Python"
print(x, y, z)
```

Casting a Variable

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.

- **Implicit Casting**:

```
num = 5     # int
result = num + 2.5  # num is implicitly cast to float
```

- **Explicit Casting**:

```
str_num = "123"
int_num = int(str_num)  # Explicitly cast string to integer
```

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.

s = "10" # Initially a string
n = int(s) # Cast string to integer
cnt = 5
f = float(cnt) # Cast integer to float
age = 25
s2 = str(age) # Cast integer to string

[Note: If the variable str is already declared in the program, you will get the error TypeError: 'str' object is not callable.
To resolve this, first delete str using del str."]

```
str = '''
This is all about python.
python var... .
'''
b=20
c=str(b)
print(c)
```

TypeError: 'str' object is not callable.

```

To resolve this use del keyword before using str function.
 del str
 c=str(b)
 Print(c)
```

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.

Id= id() is used get the memory location of variable.

```
str = '''
This is all about python.
```

```
python var... .
'''
print(str)
Print(type(str)) # print the data type of the variable
Print(id(str)) # print the memory location of the variable
```

Python keywords

Keywords are reserved words that have special meanings and serve specific purposes in the language syntax. They cannot be used as identifiers (names for variables, functions, classes, etc.). For instance, "for", "while", "if", and "else" are keywords and cannot be used as identifiers.

Below is the list of keywords in Python:

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

import keyword
# printing all keywords at once using "kwlist()"
print("The list of keywords is : ")
print(keyword.kwlist)

Comments – These are statements written within the code. They are meant to explain, clarify, or give context about specific parts of the code. The purpose of comments is to explain the working of a code, they have no impact on the execution or outcome of a program.

```
Reading purpose and better understanding about programs
```

```
# denotes Single line comment
```

Single line comments are preceded by the "#" symbol. Everything after this symbol on the same line is considered a comment.

```
'''
Multiple line comment
'''
```

```
# [Note : If we assigning multiline ''' ''' to var it will be treated as string
otherwise comments.]

str = '''
This is all about python.
python var... .
'''
print(str)
```

Multiple Line Statements-

Writing a long statement in a code is not feasible or readable. Breaking a long line of code into multiple lines makes is more readable.

Using Backslashes (\)

In Python, you can break a statement into multiple lines using the backslash (\). This method is useful, especially when we are working with strings or mathematical operations.

```
sentence = "This is a very long sentence that we want to " \
"split over multiple lines for better readability."
print(sentence)

    # For mathematical operations

    total = 1 + 2 + 3 + \
            4 + 5 + 6 + \
            7 + 8 + 9
    print(total)
```

Input and 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.

Printing Variables
We can use the print() function to print single and multiple variables. We can print multiple variables by separating them with commas.

Example:

Single variable
```
s = "Bob"
print(s)
```

Multiple Variables
```
s = "Alice" age = 25 city = "New York"
print(s, age, city)
```

Output Formatting

Output Formatting in python with various techniques including the format() method, manipulation of the sep and end parameters, f-strings and the versatile % operator. These methods enable precise control over how data is displayed, enhancing the readability and effectiveness of your Python programs.

Example 1: Using format()

```
amount = 150.75
print("Amount: ${:.2f}".format(amount))
```
OutputAmount: $150.75

Example 2: Using sep and end parameter

```
# end Parameter with '@'
print("Python", end='@')
print("Programming”)

# Seprating with Comma
print('G', 'F', 'G', sep='')

# for formatting a date
print('09', '12', '2016', sep='-')
```

Output: Python@Programming
GFG
09-12-2016

```
print("Hii", "Hello", "Bye")
# by default seperate means space
# we want to give explicityly seperator so we use sep keyword
print("Hii", "Hello", "Bye", sep="-")
print("Hii", "Hello", "Bye", sep=":")
print("Hii", "Hello", "Bye", sep="=")
# end keyword is used to print at the end
print("Hii", "Hello", "Bye", end="-")
print("Hii", "Hello", "Bye") # by default end means it will start from next line
\n
# combination of end and sep
print("Hii", "Hello", "Bye", sep="-", end='|')
```

Example 3: Using f-string

```
name = 'Nutan'
age = 23
print(f"Hello, My name is {name} and I'm {age} years old.")
```

Output: Hello, My name is Nutan and I'm 23 years old.

Example 4: Using % operator

We can use **'%'** operator. % values are replaced with zero or more value of elements. The formatting using % is similar to that of 'printf' in the C programming language.

- %d –integer
- %f – float
- %s – string
- %x –hexadecimal
- %o – octal

Taking input from the user

```
num = int(input("Enter a value: "))
add = num + 5
```

Output

```
print("The sum is %d" %add)
```

Taking input in Python

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

Syntax: *input(prompt)*

How to Take Input in Python

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

```
name = input("Enter your name: ")
print("Hello,", name, "! Welcome!")
```

Output

Enter your name: shiv
Hello shiv ! Welcome !

How to Change the Type of Input in Python

By default input() function helps in taking user input as string. If any user wants to take input as int or float, we just need to typecast it.

How to input Names in Python

The code prompts the user to input a string (the color of a rose), assigns it to the variable color, and then prints the inputted color.

```
# Taking input as string
color = input("What color is rose?: ")
print(color)
```

[Note: If the user enters 22, it will not raise any error because 22 will be treated as a string, not an integer.]

Output

What color is rose?: Red
Red

Take Multiple Input in Python

We are taking multiple input from the user in a single line, splitting the values entered by the user into separate variables for each value using the split() method. Then, it prints the values with corresponding labels, either two or three, based on the number of inputs provided by the user.

```
# taking two inputs at a time


x, y = input("Enter two values: ").split()
print("Number of boys: ", x)
print("Number of girls: ", y)

# taking three inputs at a time

x, y, z = input("Enter three values: ").split()
print("Total number of students: ", x)
print("Number of boys is : ", y)
print("Number of girls is : ", z)
```

How to input Numbers in Python

The code prompts the user to input an integer representing the number of roses, converts the input to an integer using typecasting, and then prints the integer value.

```
# Taking input as int
# Typecasting to int
n = int(input("How many balls? "))
print(n)
```

[Note: If you want to use the variable for further operations, you must store it into a variable. Here if user enter “Hello” it will raise a **ValueError**. This is because the int() function expects a valid numeric input, and "hello” is a non-numeric string that cannot be converted to an integer.]

```
print(int(input("How many balls? ")))
```

Elaborate actions in detail below:

1. **input("How many balls? ")**: This prompts the user with the message "How many balls? " and waits for the user to type something in. The input will be read as a string.
2. **int(...)**: The int() function attempts to convert the user’s input (which is a string) into an integer.
3. **print(...)**: This then prints the integer result to the console.

Output

```
How many balls?: 8
8
```

How to take input Float/Decimal Number in Python

The code prompts the user to input the price of each rose as a floating-point number, converts the input to a float using typecasting, and then prints the price.


```
# Taking input as float
```

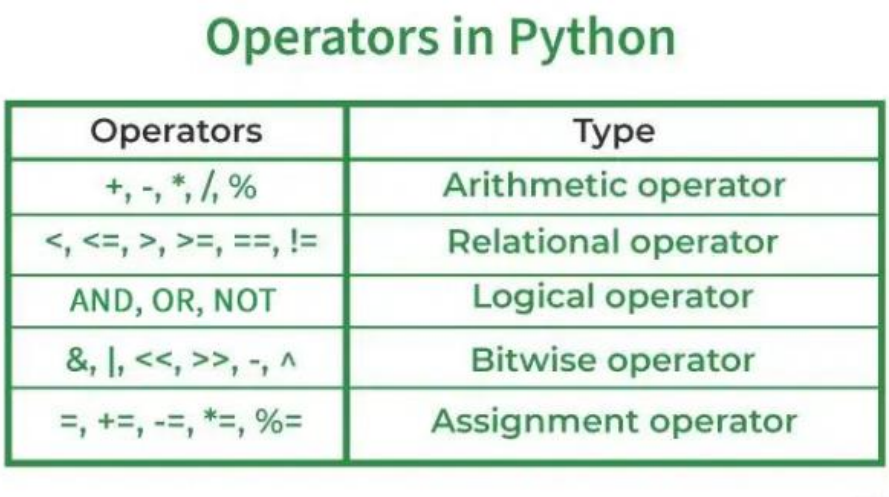
Taking input as float
Typecasting to float

```
price = float(input("Price of each  ball?: "))
print(price)
```

input () function first takes the input from the user and converts it into a string. The type of the returned object always will be <class 'str'>. It does not evaluate the expression it just returns the complete statement as String. It executes, program flow will be stopped until the user has given input.

Operators- Operators are used to perform operations on values and variables. These are standard symbols used for logical and arithmetic operations.

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



1. Arithmetic Operators are used to perform basic mathematical operations like **addition, subtraction, multiplication** and **division**.
2. Comparision of relational operators compares the values. It either returns **True** or **False(Boolean Value)** according to the condition.

```
a = 13
b = 33

print(a > b) // False 13>33
```

```
print(a < b) // True  33>13
print(a == b) // False 13 == 33
print(a != b) // True 13 is not equals to 33
print(a >= b) // False 13 is equals to or greater than 33
print(a <= b) // True 13 is equals to or less than 33
```

3. Logical opearator 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

```
a = True ->True value means 1
b = False -> False value means 0
print(a and b) //  False 1 and 0
print(a or b) // True 1 or 0
print(not a) // False opposite of 1(True)
```

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

Bitwise Operators :

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

```
a = 10
b = 4

print(a & b) // 0 -> 1010 (a) & 0100 (b) -------- 0000 (result) 0 in decimal
print(a | b) // 14 -> 1010(a) | 0100(b) --------- 1110 (result) 14 in decimal
print(~a)   // The binary representation of 10 is 1010.
              Flipping all the bits gives: 0101 becomes 1010 (but since we are
               using two's complement representation, it results in -11).
```

```
print(a ^ b) // XOR 1010 (a) ^ 0100 (b) -------- 1110 (result)
print(a >> 2) // Right Shift (>>) 1010 (binary representation of 10) >> 2
                (shift by 2)        ------- 0010 (result is 2 in decimal)
print(a << 2) // Left shift The << (left shift) operator shifts the bits of the
    number to the left by a specified number of positions. This operation
    effectively multiplies the number by 2 for each shift.
```

5. 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.

```
a = 10
b = a
print(b)
b += a // b=b+a
print(b)
b -= a // b=b-a
print(b)
b *= a // b=b*a
print(b)
b <<= a // b=b<<a
 print(b)
```

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.

is *True if the operands are identical*
is not *True if the operands are not identical*

a = 10

b = 20

c = a

print(a is not b) //True

print(a is c) // True

Python IS Operator

The **is** operator evaluates to True if the variables on either side of the operator point to the same object in the memory and false otherwise.

```
num1 = 5
num2 = 5

a = [1, 2, 3]
b = [1, 2, 3]
c = a

s1 = "hello world"
s2 = "hello world"

# using 'is' identity operator on different datatypes
print(num1 is num2) // True
print(a is b) // False
print(a is c) // True
print(s1 is s2) // True
```

[Note: We can see here that even though both the lists, i.e., 'lst1' and 'lst2' have same data, the output is still False. This is because both the lists refers to different objects in the memory. Where as when we assign 'lst3' the value of 'lst1', it returns True. This is because we are directly giving the reference of 'lst1' to 'lst3'.*]*

Python IS NOT Operator

The **is not** operator evaluates True if both variables on the either side of the operator are not the same object in the memory location otherwise it evaluates False.

using 'is not' identity operator on different datatypes

```
print(num1 is not num2) // False
print(a is not b) //True
print(a is not c) // False
 print(s1 is not s2) // False
print(s1 is not s1) // False
```

Python Membership Operators

The Python membership operators test for the membership of an object in a sequence, such as strings, lists, or tuples. Python offers two membership operators to check or validate the membership of a value.

Python IN Operator

The **in** operator is used to check if a character/substring/element exists in a sequence or not. Evaluate to True if it finds the specified element in a sequence otherwise False.

```
str=”Road”
print(‘R’ in str) //True
print(‘r’ in str) // False coz it is case sensitive small means small, caps letter means caps
```

Python NOT IN Operator

The ‘not in’ Python operator evaluates to true if it does not find the variable in the specified sequence and false otherwise.

```
Print(‘r’ not in ‘Road’) // False
```

operators.contains() Method

An alternative to Membership ‘in’ operator is the contains() function. This function is part of the Operator module in Python. The function take two arguments, the first is the sequence and the second is the value that is to be checked.

Syntax:`operator.contains(sequence,value)`

```
ex: operator.contains(“Road”, ‘R’)
```

Difference between ‘==’ and ‘is’ Operator

While comparing objects in Python, the users often gets confused between the Equality operator and Identity ‘is’ operator. The equality operator is used to compare the value of two variables, whereas the identities operator is used to compare the memory location of two variables.

of 'is' and '==' operators

```
a=[1,2,3]
b=[1,2,3]
```

using 'is' and '==' operators

```
print(a is b) // False
print(a==b)//True
```

!= is defined as **not equal to** operator. It returns **True** if operands on either side are not equal to each other, and returns **False** if they are equal.

```
a = 10
b = 10
print(a != b) // False
print(id(a), id(b))

c = "Python"
d = "Python"
print(c != d) // False
print(id(c), id(d))
```

Operator Precedence and Associativity

In Python, operators have different levels of precedence, which determine the order in which they are evaluated. When multiple operators are present in an expression, the ones with higher precedence are evaluated first. In the case of operators with the same precedence, their

associativity comes into play, determining the order of evaluation.

Operator Precedence: Higher precedence operators are evaluated before lower precedence ones.
Associativity: Determines how operators with the same precedence level are evaluated (left-to-right or right-to-left).

Operator	Precedence	Associativity	Description
`()`	1	Left to Right	Parentheses (used for grouping and function calls)
`**`	2	Right to Left	Exponentiation
`+x`, `-x`, `~x`	3	Right to Left	Unary plus, Unary minus, Bitwise NOT
`*`, `/`, `//`, `%`	4	Left to Right	Multiplication, Division, Floor division, Modulo
`+`, `-`	5	Left to Right	Addition, Subtraction
`<<`, `>>`	6	Left to Right	Bitwise shift left, Bitwise shift right
`<`, `<=`, `>`, `>=`, `==`, `!=`	7	Left to Right	Comparison operators
`not`	8	Left to Right	Logical NOT
`and`	9	Left to Right	Logical AND
`or`	10	Left to Right	Logical OR
`if` `else`	11	Right to Left	Ternary conditional (inline if)
`lambda`	12	Right to Left	Lambda function definition
`:=`	13	Right to Left	Walrus operator (assignment expression)
`=` `+=` `-=` `*=` `/=` `//=` `%=`	14	Right to Left	Assignment operators and augmented assignments
`is`, `is not`, `in`, `not in`	15	Left to Right	Identity and membership tests
`[]`, `()`, `{}`	16	Left to Right	Subscript, Slicing, Calls, Dictionary creation
`await`	17	Right to Left	Await expression (used in asynchronous code)

Example : result = (5 + 3 * 2 ** 3 - (10 // 3) % 4) * 2 + 1 and 5 > 4 or not (6 == 6) is not 7 + 3

Parentheses ( ): Parentheses have the highest precedence, so we start by evaluating everything inside them. We have two parenthesis but associativity left to right ->

First, evaluate the inner expression `(10 // 3)` of `(5+3*2**3-(10//3)%4)`

```
10 // 3 = 3
Then ** exponential has higher precedence
2**3=8

Now this parenthesis expression becomes
(5+3*8-3%4)
% and * has precedence but associativity left to right ->
First, evaluate * then %
So 3*8=24
5+24-3%4
Then 3%4=3(Remainder of 3 divided by 4)

Now, this parenthesis expression becomes
(5+24-3) => 29-3=> 26 # + and – both has same precedence but associativity
left to right

After evaluating first parenthesis final expression becomes
```

Expression - (26) * 2 + 1 and 5 > 4 or not (6 == 6) is not 7 + 3

Now evaluate 2nd parenthesis
(6==6) =>True

Expression - (26) * 2 + 1 and 5 > 4 or not True is not 7 + 3

Now * then + , associativity left to right
26*2=>52+1=53 , 7+3=10

Expression - 53 and 5 > 4 or not True is not 10

Now > has higher precedence then other operators
5>4=>True

Expression - 53 and True or not True is not 10

Now not has higher precedence then and then or then is not
not True=> False # inversion of True
53 and True=> True # 53 is non zero numeric
[Note; if any 0 and True then always False]

Expression - True or False is not 10

Expression – False is not 10 => o is not 10 => True

[Note: False =0 and True=1 or greater than 1

Expression - True or True => 1 or 1

result=True