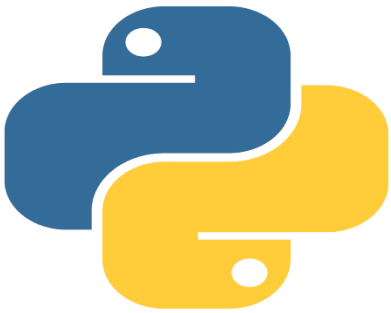

Python

Part 1

What is python

Python is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language. language. It was created by Guido van Rossum during 1985- 1990. , Python source code is also available under the GNU General Public License (GPL).

Application of python language

- Web page development
- Data science and data analysis
- Ai and machine learning
- Software testing

Python features

- General purpose language
- It is interpreter language
- Simpler to learn
- OOPS language
- Open source
- More number of platform available
- It can mix up with another language(python + html,python + c etc)
- User friendly

Interpreter language:

Python code is executed line-by-line by an interpreter at runtime. The interpreter translates the high-level Python code into machine-executable instructions as it encounters them. This approach offers benefits such as easier debugging and quicker testing

Variable in python

Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.

```
var = 10
```

Data types in python

In general, the data types are used to define the type of a variable. It represents the type of data we are going to store in a variable and determines what operations can be done on it.

Types of data types

- Numeric data type
  - int
  - float
  - complex
- String data type
- Sequential data type
  - List
  - Tuple
  - Range
- Binary data type
  - Byte
  - Bytearray
  - Memoryview
- Dictionary data type
- Set data type
  - Set
  - Frozen set
- Boolean data type
- None type

✓ **Python numeric data type**

```
Var1=10
Var2=20.5
Var3=True
Var4=2+3j
```

✓ **Python string data type**

```
str=”hello world”
```

✓ **Python sequential data type**

- **List data type**

```
List=[1,2,4,6,9]
```

- **Tuple data type**

```
Tuple=(1,7,8,5,4)
```

- **Range data type**

```
for i in range(5):
    print(i)
```

✓ **Python binary data type**

- **Byte data type**

The byte data type in Python represents a sequence of bytes. Each byte is an integer value between 0 and 255. It is commonly used to store binary data, such as images, files, or network packets.

In the following example, we are using the built-in bytes() function to explicitly specify a sequence of numbers representing ASCII values –

```
b1 = bytes([65, 66, 67, 68, 69])
print(b1)

Output:
   b'ABCDE'
```

we are using the "b" prefix before a string to automatically create a bytes object

```
b2 = b'Hello'
print(b2)

Output:
     b'Hello'
```

- **Bytearray data type**

The bytearray data type in Python is quite similar to the bytes data type, but with one key difference: it is mutable, meaning you can modify the values stored in it after it is created.

In the example below, we are creating a bytearray by passing an iterable of integers representing byte values –

```
value = bytearray([72, 101, 108, 108, 111])
print(value)

Output:
bytearray(b'Hello')
```

- **Memoryview data type**

A memoryview is a built-in object that provides a view into the memory of the original object, generally objects that support the buffer protocol, such as byte arrays (bytearray) and bytes (bytes).

We are creating a memoryview object directly by passing a supported object to the memoryview() constructor.

```
data = bytearray(b'Hello, world!')
view = memoryview(data)
print(view)

Output:
<memory at 0x00000186FFAA3580>
```

✓ **Python dictionary bit**

Python dictionary are kind of hash table type. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

```
dict = {1:'one', 2:'two', 3:'three'}
```

✓ **Python set data type**

Items in the set collection may not follow the same order in which they are entered. The position of items is optimized by Python to perform operations over set as defined in mathematics.

```
set = {2023, "Python", 3.11, 5+6j, 1.23E-4}
```

➢ **Primitive and non-primitive data types**

✓ **Primitive**

The primitive data types are the fundamental data types that are used to create complex data types (sometimes called complex data structures). There are mainly four primitive data types, which are –

- Integer
- Float
- String
- Boolean

✓ **Non primitive**

The non-primitive data types store values or collections of values. There are mainly four types of non-primitive types, which are –

- List
- Tuple
- Set
- Dictionary

print in python

Python print() function prints the message to the screen or any other standard output device. In this article, we will cover about print() function in Python as well as it's various operations.

Example:

```
print("Hello world")
```

Output:

```
Hello world
```

Example:

```
a=10
print(a)
```

Operators

Python operators are special symbols used to perform specific operations on one or more operands. The variables, values, or expressions can be used as operands. For example, Python's addition operator (+) is used to perform addition operations on two variables, values, or expressions.

The following are some of the terms related to python operators

Unary operator: Python operators that require one operand to perform a specific operation are known as unary operators.
Binary operator: Python operators that require two operands to perform a specific operation are known as binary operators.
Operand: Variables, values, or expressions that are used with the operator to perform a specific operation.

- ✓ **Arithmetic operator**

Python arithmetic operators are used to perform mathematical operations such as addition, subtraction, multiplication, division, and more on numbers.

Operator	Name	Example
+	Addition	a + b = 30
-	Subtraction	a b = -10
*	Multiplication	a * b = 200
/	Division	b / a = 2
%	Modulus	b % a = 0
**	Exponent	a**b =10**20
//	Floor Division	9//2 = 4

Example:

```
a=10
b=5
print(“sum of two numbers” ,a+b)
print(“subtraction of two numbers” ,a-b)
print(“multiplication of two numbers” ,a*b)
print(“Division of two numbers” ,a/b)
print(“modula operation of two numbers” ,a%b)
print(“Floor division of two numbers” ,a//b)
print(“Exponential of a number” ,a**b)
```

Output:

```
Sum of two numbers 22
Subtraction of two numbers 5
Multiplication of two numbers 59
Division of two numbers 5
Modula of two numbers 0
Floor division of two numbers 2
Exponential of a number 10000
```

- ✓ **Comparison operator**

Comparison operators in Python are very important in Python's conditional statements (if, else and elif) and looping statements (while and for loops). The comparison operators

also called relational operators. Some of the well known operators are "<" stands for less than, and ">" stands for greater than operator.

<	Less than	a<b
>	Greater than	a>b
<=	Less than or equal to	a<=b
>=	Greater than or equal to	a>=b
==	Is equal to	a==b
!=	Is not equal to	a!=b

Example:

```
a=10
b=20
print(a>b)
print(a<b)
‘print(a==b)
print(a!=0)
```

Output:

```
False
True
False
False
```

- ✓ **Assignment operators**

The = (equal to) symbol is defined as assignment operator in Python. The value of Python expression on its right is assigned to a single variable on its left. The = symbol as in programming in general (and Python in particular) should not be confused with its usage in Mathematics, where it states that the expressions on the either side of the symbol are equal.

Operator	Example	Same As
=	a = 10	a = 10
+=	a += 30	a = a + 30
-=	a -= 15	a = a - 15
*=	a *= 10	a = a * 10
/=	a /= 5	a = a / 5
%=	a %= 5	a = a % 5
**=	a **= 4	a = a ** 4
//=	a //= 5	a = a // 5
&=	a &= 5	a = a & 5
|=	a |= 5	a = a | 5
^=	a ^= 5	a = a ^ 5
>>=	a >>= 5	a = a >> 5
<<=	a <<= 5	a = a << 5

Example:

```
a =6
b = 2
c = 0
print (“a: {} b: {} c : {}”.format(a,b,c))
c = a + b
print (“a: {}  c = a + b: {}”.format(a,c))
c += a
print (“a: {} c += a: {}”.format(a,c))
c *= a
print (“a: {} c *= a: {}”.format(a,c))
c /= a
print (“a: {} c /= a : {}”.format(a,c))
c  = 2
print (“a: {} b: {} c : {}”.format(a,b,c))
c %= a
print (“a: {} c %= a: {}”.format(a,c))
c **= a
print (“a: {} c **= a: {}”.format(a,c))
```

```
c //= a
print (“a: {} c //= a: {}”.format(a,c))
```

Output:

```
a: 21 b: 10 c : 0
a: 21  c = a + b: 31
a: 21 c += a: 52
a: 21 c *= a: 1092
a: 21 c /= a : 52.0
a: 21 b: 10 c : 2
a: 21 c %= a: 2
a: 21 c **= a: 2097152
a: 21 c //= a: 99864
```

✓ **Python bitwise operator**

Python bitwise operator are normally used to perform bitwise operations on integer-type objects. However, instead of treating the object as a whole, it is treated as a string of bits. Different operations are done on each bit in the string.

Python has six bitwise operators - &, |, ^, ~, << and >>. All these operators (except ~) are binary in nature, in the sense they operate on two operands. Each operand is a binary digit (bit) 1 or 0.

Operator	Name	Example
&	AND	a & b
|	OR	a | b
^	XOR	a ^ b
~	NOT	~a
<<	Zero fill left shift	a << 3
>>	Signed right shift	a >> 3

- **Bitwise AND operator**

Bitwise AND operator is somewhat similar to logical and operator. It returns True only if both the bit operands are 1 (i.e. True). All the combinations are –

```
0 & 0 is 0
1 & 0 is 0
0 & 1 is 0
1 & 1 is 1
```

Example:

```
a=60
b=13
c=a&b
print(c)
```

Output:

```
12
```

To understand how Python performs the operation, obtain the binary equivalent of each variable.

Example:

```
print ("a:", bin(a))
print ("b:", bin(b))
```

Output:

```
a: 0b111100
b: 0b1101
```

- **Bitwise OR operator**

The "|" symbol (called pipe) is the bitwise OR operator. If any bit operand is 1, the result is 1 otherwise it is 0.

Example:

```
a=60
b=13
print ("a:",a, "b:",b, "a|b:",a|b)
print ("a:", bin(a))
print ("b:", bin(b))
```

Output:

```
a: 60 b: 13 a|b: 61
a: 0b111100
b: 0b1101
```

- **Bitwise XOR operator**

The term XOR stands for exclusive OR. It means that the result of OR operation on two bits will be 1 if only one of the bits is 1.

```
0 ^ 0 is 0
0 ^ 1 is 1
1 ^ 0 is 1
1 ^ 1 is 0
```

Example:

```
a=60
b=13
print ("a:",a, "b:",b, "a^b:",a^b)
```

Output:

```
a: 60 b: 13 a^b: 49
```

- **Bitwise NOT operator**

This operator is the binary equivalent of logical NOT operator. It flips each bit so that 1 is replaced by 0, and 0 by 1

Example:

```
a=60
print ("a:",a, "~a:", ~a)
```

Output:

```
a: 60 ~a: -61
```

- **Bitwise Left wise operator**

Left shift operator shifts most significant bits to right by the number on the right side of the "<<" symbol. Hence, "x << 2" causes two bits of the binary representation of to right.

Bitwise left operator.

Example:

```
a=60
print ("a:",a, "a<<2:", a<<2)
```

Output:

```
a: 60 a<<2: 240
```

- **Bitwise right operator**

Right shift operator shifts least significant bits to left by the number on the right side of the ">>" symbol. Hence, "x >> 2" causes two bits of the binary representation of to left.

Example:

```
a=60
print ("a:",a, "a>>2:", a>>2)
```

Output:

```
a: 60 a>>2: 15
```

✓ **Membership operator**

The membership operators in Python help us determine whether an item is present in a given container type object, or in other words, whether an item is a member of the given container type object.

Python has two membership operators: in and not in. Both return a Boolean result. The result of in operator is opposite to that of not in operator.

- **in operator**

  The “in” operator is used to check whether a substring is present in a bigger string any item is present in a list or tuple, or a sub-list or sub-tuple is included in a list or tuple.

Example:

```
var = "Hello world"
a = "H"
b = "o"
c = "rld"
d = "lwo"
print (a, "in", var, ":", a in var)
print (b, "in", var, ":", b in var)
print (c, "in", var, ":", c in var)
print (d, "in", var, ":", d in var)
```

Output:

```
H in Hello world True
O in Hello world True
rld in Hello world True
lwo in Hello world False
```

- **not in operator**

The "not in" operator is used to check a sequence with the given value is not present in the object like string, list, tuple, etc.

Example:

```
var = "Hello world"
a = "H"
b = "o"
c = "rld"
d = "lwo"
print (a, "in", var, ":", a in var)
print (b, "in", var, ":", b in var)
print (c, "in", var, ":", c in var)
print (d, "in", var, ":", d in var)
```

Output:

```
False
False
False
True
```

✓ **Python identity operator**

The identity operators compare the objects to determine whether they share the same memory and refer to the same object type data type.

Python provided two identity operators; we have listed them as follows:

'is' Operator
'is not' Operator

- **Python 'is' operator**

The 'is' operator evaluates to True if both the operand objects share the same memory location. The memory location of the object can be obtained by the "id()" function. If the "id()" of both variables is same, the "is" operator returns True.

Example:

```
a = [1, 2, 3, 4, 5]
b = [1, 2, 3, 4, 5]
c = a

# Comparing and printing return values
print(a is c)
print(a is b)

# Printing IDs of a, b, and c
print("id(a) : ", id(a))
print("id(b) : ", id(b))
print("id(c) : ", id(c))
```

Output:

```
True
False
id(a) :  140114091859456
id(b) :  140114091906944
id(c) :  140114091859456
```

- **'is not' operator**

The 'is not' operator evaluates to True if both the operand objects do not share the same memory location or both operands are not the same objects.

Example:

```
a = [1, 2, 3, 4, 5]
b = [1, 2, 3, 4, 5]
c = a

# Comparing and printing return values
print(a is not c)
print(a is not b)

# Printing IDs of a, b, and c
print("id(a) : ", id(a))
print("id(b) : ", id(b))
print("id(c) : ", id(c))
```

Output:

```
False
True
id(a) :  140559927442176
id(b) :  140559925598080
id(c) :  140559927442176
```

✓ Operator Precedence

An expression may have multiple operators to be evaluated. The operator precedence defines the order in which operators are evaluated. In other words, the order of operator evaluation is determined by the operator precedence.

If a certain expression contains multiple operators, their order of evaluation is determined by the order of precedence. For example, consider the following expression

```
a = 2+3*5
```

Sr.No.	Operator & Description
1	**(),[], {}** Parentheses and braces
2	**[index], [index:index]** Subscription, slicing,
3	**await x** Await expression
4	****** Exponentiation
5	**+x, -x, ~x** Positive, negative, bitwise NOT
6	***, @, /, //, %** Multiplication, matrix multiplication, division, floor division, remainder
7	**+, -** Addition and subtraction
8	**<<, >>** Left Shifts, Right Shifts
9	**&** Bitwise AND
10	**^** Bitwise XOR
11	**|** Bitwise OR
12	**in, not in, is, is not, <, <=, >, >=, !=, ==** Comparisons, including membership tests and identity tests
13	**not x** Boolean NOT
14	**and** Boolean AND
15	**or** Boolean OR
16	**if else** Conditional expression
17	**lambda** Lambda expression
18	**:=** Walrus operator

User input

Python provides us with two built-in functions to read the input from the keyboard.

input () Function

Example:

```
a=int(input("Enter first no:"))
b=int(input("Enter second number"))
sum=a+b
print("sum of two no:'s",sum)
```

Output:

```
Enter first no: 3
Enter second no: 5
Sum of two no:'s 8
```

Example2:

```
a=input("Enter first string")
b=input("Enter second string")
print(a+b)
```

Output:

```
Enter first string Hello
Enter second string world
Helloworld
```

Control flow

Python program control flow is regulated by various types of conditional statement, loops, and function calls. By default, the instructions in a computer program are executed in a sequential manner, from top to bottom, or from start to end. However, such sequentially executing programs can perform only simplistic tasks. We would like the program to have a decision-making ability, so that it performs different steps depending on different conditions

Decision making

Decision making statements are used in the Python programs to make them able to decide which of the alternative group of instructions to be executed, depending on value of a certain Boolean expression.

- ✓ **Python if statement**

The if statement in python evaluates whether a condition is true or false. It contains a logical expression that compares data, and a decision is made based on the result of the comparison.

Syntax of the if Statement

```
if expression:
  # statement(s) to be executed
```

Example:

First set value 10 to a variable ,Then use an if statement to check whether the amount is greater than 0. If this condition is true, then execute content inside if statement

```
a=10
if a>0:
   print(“The given number is greater than 10”)
print(“outer of if statement”)
```

Output:

```
The given number is greater than 10
```

Example: Program for check given number is equal to zero,positive or negative

```
a=10
if a==0:
    print(“The given no: is equal to zero”)
if a>10:
    print(“The given numer is greater than zero”)
if a<0:
    print(“The given number is less than zero”)
```

Output:

```
The given number is less than zero
```

- ✓ **Python if else statement**

The if-else statement in python is used to execute a block of code when the condition in the if statement is true, and another block of code when the condition is false.

Syntax of if-else Statement

The syntax of an if-else statement in Python is as follows

```
if boolean_expression:
  # code block to be executed
  # when boolean_expression is true
else:
  # code block to be executed
  # when boolean_expression is false
```

Example: Check which number is greater

```
a=10
b=20
if(a>b):
    print("a is greater")
else:
  print("b is greater")
```

Output:

```
b is greater
```

- ✓ **Python if elif else Statement**

The if elif else statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Syntax of Python if elif else Statement

```
if expression1:
  statement(s)
elif expression2:
  statement(s)
elif expression3:
  statement(s)
else:
  statement(s)
```

Set amount to test all possible conditions: 800, 2500, 7500 and 15000. The outputs will vary accordingly

```
```

```
amount = 2500
print('Amount = ',amount)
if amount > 10000:
  discount = amount * 20 / 100
else:
  if amount > 5000:
    discount = amount * 10 / 100
  else:
    if amount > 1000:
      discount = amount * 5 / 100
    else:
      discount = 0
print('Payable amount = ',amount - discount)
```

Output:

Set amount to test all possible conditions: 800, 2500, 7500 and 15000. The outputs will vary accordingly

```
Amount: 800
Payable amount = 800
Amount: 2500
Payable amount = 2375.0
Amount: 7500
Payable amount = 6750.0
Amount: 15000
Payable amount = 12000.0
```

- ✓ **Nested if Statement with else Condition**

Syntax

The syntax of the nested if construct with else condition will be like this

```
if expression1:
   statement(s)
   if expression2:
      statement(s)
   else
      statement(s)
else:
   if expression3:
      statement(s)
   else:
      statement(s)
```

Example:

```
num=9
print ("num = ",num)
if num%2==0:
   if num%3==0:
      print ("Divisible by 3 and 2")
   else:
      print ("divisible by 2 not divisible by 3")
else:
   if num%3==0:
      print ("divisible by 3 not divisible by 2")
   else:
      print ("not Divisible by 2 not divisible by 3")
```

Output:

```
Num=9
Divisible by 3 not divisible by 2
```

Python match-case Statement

A Python match-case statement takes an expression and compares its value to successive patterns given as one or more case blocks. Only the first pattern that matches gets executed. It is also possible to extract components (sequence elements or object attributes) from the value into variable.

Syntax

The following is the syntax of match-case statement in Python

```
match variable_name:
  case 'pattern 1' : statement 1
  case 'pattern 2' : statement 2
    ...
  case 'pattern n' : statement n
```

Example:

```
day = 4
match day:
  case 1:
    print("Monday")
  case 2:
    print("Tuesday")
  case 3:
    print("Wednesday")
  case 4:
    print("Thursday")
  case 5:
    print("Friday")
  case 6:
    print("Saturday")
  case 7:
    print("Sunday")
```

Output:

```
Thursday
```

Python – Loops

Python loops allow us to execute a statement or group of statements multiple times.

In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on. There may be a situation when you need to execute a block of code several number of times.

- ✓ **Python while Loop**

A while loop in Python programming language repeatedly executes a target statement as long as the specified boolean expression is true. This loop starts with while keyword followed by a boolean expression and colon symbol (:). Then, an indented block of statements starts.

Syntax of while Loop

```
while expression:
   statement(s)
```

Example:

The following example illustrates the working of while loop. Here, the iteration run till value of count will become 5.

```
count=0
while count<5:
   count+=1
   print ("Iteration no. {}".format(count))

print ("End of while loop")
```

Output:

```
Iteration no. 1
Iteration no. 2
Iteration no. 3
Iteration no. 4
Iteration no. 5
End of while loop
```

Python - break Statement

Python break statement is used to terminate the current loop and resumes execution at the next statement, just like the traditional break statement in c.

Syntax of break Statement

```
looping statement:
  condition check:
    break
```

Example:

```
a=1
while a<0:
     print("value is",a)
      if a==5:
          break
      a=a+1
```

Output:

```
value is 1
value is 2
value is 3
value is 4
value is 5
```

Continue statement

Python continue statement is used to skip the execution of the program block and returns the control to the beginning of the current loop to start the next iteration. When encountered, the loop starts next iteration without executing the remaining statements in the current iteration.

Syntax of continue statment

```
looping statement:
   condition check:
      continue
```

Example:

```
for letter in 'Python':
   if letter == 'h':
      continue
   print ('Current Letter :', letter)
print ("Good bye!")
```

Python - pass Statement

Python pass statement is used when a statement is required syntactically but you do not want any command or code to execute. It is a null which means nothing happens when it executes. This is also useful in places where piece of code will be added later, but a placeholder is required to ensure the program runs without errors.

Syntax of pass Statement

```
Pass
```

Example:

```
for letter in 'Python':
   if letter == 'h':
      pass
      print ('This is pass block')
   print ('Current Letter :', letter)
print ("Good bye!")
```

Python - Nested Loops

In Python, when you write one or more loops within a loop statement that is known as a nested loop. The main loop is considered as outer loop and loop(s) inside the outer loop are known as inner loops

✓ **Python Nested for Loop**

The for loop with one or more inner for loops is called nested for loop. A for loop is used to loop over the items of any sequence, such as a list, tuple or a string and performs the same action on each item of the sequence.

Python Nested for Loop Syntax

The syntax for a Python nested for loop statement in Python programming language is as follows

```
for  iterating_var in sequence:
   for  iterating_var in sequence:
      statements(s)
   statements(s)
```

Example:

```
months = ["jan", "feb", "mar"]
days = ["sun", "mon", "tue"]
for x in months:
  for y in days:
    print(x, y)
print("Good bye!")
```

Output:

```
jan sun
jan mon
jan tue
feb sun
feb mon
feb tue
mar sun
mar mon
mar tue
Good bye!
```

Python Nested while Loop

The while loop having one or more inner while loops are nested while loop. A while loop is used to repeat a block of code for an unknown number of times until the specified boolean expression becomes TRUE.

Python Nested while Loop Syntax

The syntax for a nested while loop statement in Python programming language is as follows

```
while expression:
   while expression:
      statement(s)
   statement(s)
```