Python

Definition of program: A program is a set of instructions for completing and exacts well defined tasks. Every instruction has to be exats also. Instructions are normally very low level.

There are three types of programming language:

1. High level language
2. Low level language
3. Machine lanaguage

Interpreter: It works line by line in all of the program.

Compiler: It works all program at once

IDE(Integrated Development Environment):This is a software that provides facilities for development,testing & defined of software.-
Python as many IDE's like pycharm, Jupyter, Spyder ,Eclips etc.

Parts of Program------

- General part of a program is input,processing and output.
- Every program must have a well defined aim.

Data structure :Data structures are part of programs that stores data for processing and also for storage.

Comments in python :
Single line comment -- #
Multi line comment ---- “”” parts of program “”” or ‘’’ parts of program ‘’’

1. Python indentation

Indentation means the distance of the line from its left beginning.
This defines a block statement in Python. In other language it is simply ignored.

The block statement is defined in other languages by putting the statements in curly brackets.
{

}
For example
If(a-b)
{
s1:
s2:
}
will cause the statements s1 and s2 to be excused if
a>b.
In python this will be written as
If(a>b):
s1
s2
The shifting from the end defines the block.
Brackets are not required in Python conditions.
Python uses indentation to indicate a block of code.
Line termination doesn't require ;.';' is optional.

Example:
```
If a>b:
    print (”A-B”)
```
Python will give you an error if you skip the indentation:
Example
Syntax error:

```
if a>b:
print("A>B")
```
Number of spaces has to be at least one.
 Example
```
if a>b:
 print("a>b")
 if a>b:
 print("A>B")
```
 The number of spaces has to be the same.
 Example
Syntax Error:
```
if a>b:
 print("A>B")
print("B<A”)
```

2. Python Comments

Comments in a program are code that isn't executed. Comments are for explaining the code, for remembering and also for stopping the execution of code sometimes..
Creating a Comment
Comments starts with a # and Python will ignore them:
Example #this is a comment print ("The previous statement was a comment") Comments can be placed at start of a line or at some other place in a line and Python will ignore the rest of the line:
Example
```
 print("Rest of the line is a comment")
 #This is a comment
```
 Comments does not have to be text to explain the code,
 it can also be used to prevent Python from executing code:
 Example
#print("This will not be executed") print("This will be executed") Multi Line Comments Python does not really have a syntax for multi line comments. To add a multiline comment you could insert a # for each line:

Example
#This is a comment
#in
#multiple lines print("Many commented lines")
A multiline string can be used as a multiline comment.

Example
""" First line comment
 Second line comment
 Third line comment """
print("Multiline comment")
As long as the string is not assigned to a variable, Python will read the code, but then ignore it, and you have made a multiline comment.

3. Python Data Types

Data types define the possible values that can be stored in variables of the type and also the range of values. Data types also define the possible operations and also the type of operation that willbe performed when overloaded operators are used.

Variables can store data of different types, and different types can do different things.

Python has the following data types available with the language, in these categories:

Text Type: str Numeric
Types: int, float, complex
Sequence Types: list, tuple, range
Mapping Type: dict
Set Types: set, frozenset
Boolean Type: bool
Binary Types: bytes, bytearray, memoryview
Getting the Data Type
You can get the data type of any object by using the type() function:

Example
Print the data type of the variable
x: x = 5
print(type(x))
Setting the Data Type In Python, the data type is set when you assign a value to a variable:
x = "Hello Champak"
str x = 20
int x = 20.5
float x = 1j
complex x = ["Varanasi", "Lucknow", "Prayagraj"]
list x = ("Varanasi", "Lucknow", "Prayagraj") tuple
x = range(6) range
x = {"name" : "Champak", "Grade" : 1} dict
x = {"Varanasi", "Prayagraj", "Lucknow"} set
x = frozenset({"Varanasi", "Prayagraj", "Lucknow"}) frozenset
x = True bool
x = b"Hello" bytes
x = bytearray(5) bytearray
x = memoryview(bytes(5)) memoryview

Setting the Specific Data Type If you want to specify the data type, you can use the following constructor functions:
Example
x = str("Champak") str
x = int(20) int
x = float(20.5) float
x = complex(1j) complex
x = list(("Varanasi", "Lucknow", "Prayagraj")) list
x = tuple(("Varanasi", "Lucknow", "Prayagraj")) tuple
x = range(6) range

x = dict(name="Champak", grade=1) dict
x = set(("Varanasi", "Prayagraj", "Lucknow")) set
x = frozenset(("Varanasi", "Prayagraj", "Lucknow")) frozenset
x = bool(5) bool
x = bytes(5) bytes
x = bytearray(5) bytearray
x = memoryview(bytes(5)) Frozen set is immutable while set can be changed.

Therefore frozenset can be used as a dictionary key. Memoryview is like a C memory pointer.

4. Python Numbers

There are three numeric types in Python:

Int
float
complex
Variables of numeric types are created when you assign a value to them:
Example
x = 1 # int
y = 2.8 # float
z = 1j # complex
To verify the type of any object in Python, use the type() function:
Example
print(type(x))
print(type(y))
print(type(z))
Int Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Example
Integers:
x = 1
y = 35656222554887711
z = -3255522
print(type(x))
print(type(y))
print(type(z))
Float Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Example
Floats:
x = 1.10
y = 1.0
z = -35.59
print(type(x))
print(type(y))
print(type(z))
Float can also be scientific numbers with an "e" to indicate the power of 10.
Example
Floats:
x = 35e3
y = 12E4
z = -87.7e100
print(type(x))
print(type(y))
print(type(z))

Complex
Complex numbers are written with a "j" as the imaginary part:
Example

Comple
x: x = 3+5j
y = 5j
z = -5j
print(type(x))

print(type(y))
print(type(z))

Type Conversion You can convert from one type to another with the int(), float(), and complex() methods:

Example
Convert from one type to another:
x = 1   # int
y = 2.8   # float
z = 1j   # complex

#convert from int to float:
a = float(x)
#convert from float to int:
b = int(y)
#convert from int to complex:
c = complex(x)
print(a)
print(b)
print(c)
print(type(a))
print(type(b))
print(type(c))

Note: You cannot convert complex numbers into another number type.

Random Number Python does not have a random() function to make a random number, but Python has a built-in module called random that can be used to make random numbers:

Example
Import the random module, and display a random number between 1 and 9:

import random

print(random.randrange(1,10))

5. Python Variables

Creating Variables

Variables are containers for storing data values. Unlike other programming languages, Python has no command for declaring a variable.
A variable is created the moment you first assign a value to it.
Example
x = 5
y = "Champak"
print(x)
print(y)

Variables do not need to be declared with any particular type and can even change type after they have been set.
Example
x = 4 # x is of type int
x = "Champak" # x is now of type str
print(x)

Variable Names
A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume). Rules for Python variables:
A variable name must start with a letter or the underscore character
A variable name cannot start with a number
A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
Variable names are case-sensitive (age, Age and AGE are three different variables)

Example
```
#Legal variable names:
myvar = "Champak"
my_var = "Champak"
_my_var = "Champak"
myVar = "Champak"
MYVAR = "Champak"
myvar2 = "Champak"

#Illegal variable names:
2myvar = "10"
my-var = "20"
my var = "40"
```
Remember that variable names are case-sensitive Assign Value to Multiple Variables Python allows you to assign values to multiple variables in one line:
Example
```
x, y, z = "1", "2", "3"
print(x)
print(y)
print(z)
```

Output 1 2 3

```
x = y = z = "1"
print(x)
print(y)
print(z)
```

Output 1 1 1

Output Variables
The Python print statement is often used to output variables.
To combine both text and a variable, Python uses the + character:
Example
```
x = "city"
print("Varanasi is a " + x)
```

Output
```
Varanasi is a city
```
You can also use the + character to add a variable to another variable:
Example
```
x = "Varanasi is " y = "city"
z = x + y
print(z)
```

Output
```
Varanasi is a city
```

For numbers, the + character works as a mathematical operator:
Example
```
x = 5
y = 10
print(x + y)
```

Output

```
15
```

If you try to combine a string and a number, Python will give you an error:

Example

```
x = 5
y = "City"
print(x + y)
```

Output

```
Error
```

Global Variables

Variables that are created outside of a function (as in all of the examples above) are known as global variables.

Global variables can be used by everyone, both inside of functions and outside.

Example

Create a variable outside of a function, and use it inside the function

```
x = "city"
 def f():
print("Varanasi is a " + x)
f()
```

Output

```
Varanasi is a city
```

If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example

Create a variable inside a function, with the same name as the global variable

```
 x = "Varanasi"
def f():
x = "Lucknow"
print( x + " is a city")
 f()
print( x + " is a city")
```

Output

```
 Lucknow is a city
 Varanasi is a city
```

The global Keyword Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function. To create a global variable inside a function, you can use the global keyword.

Example

If you use the global keyword, the variable belongs to the global scope:

```
def myfunc():
 global x x = "Varanasi"
f()
 print(x + " is a city")
```

Output

```
 Varanasi is a city
```

Also, use the global keyword if you want to change a global variable inside a function.

Example

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

```
x = "Lucknow"

def f():
```

```
global x

x = "Varanasi"

 f()
print(x + " is a city")
```

Output
Varanasi is a city

6. Python Casting

Specify a Variable Type
There may be times when you want to specify a type on to a variable. This can be done with casting.
Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

int() - constructs an integer number from an integer literal, a float literal (by rounding down to the previous whole number), or a string literal (providing the string represents a whole number)
float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

Example
Integers:

```
x = int(1)    # x will be 1
y = int(2.8)    # y will be 2
z = int("3")    # z will be 3
```

Example
Floats:
```
x = float(1)    # x will be 1.0
y = float(2.8)   # y will be 2.8
z = float("3")    # z will be 3.0
w = float("4.2")    # w will be 4.2
```

Example
Strings:
```
x = str("s1") # x will be 's1'
y = str(2)   # y will be '2'
z = str(3.0) # z will be '3.0'
```

7. Python Booleans

Booleans represent one of two values: True or False.

Boolean Values In programming you often need to know if an expression is True or False.

You can evaluate any expression in Python, and get one of two answers, True or False.

When you compare two values, the expression is evaluated and Python returns the Boolean answer:

Example
```
print(10 > 9)
print(10 == 9)
 print(10 < 9)
```

When you run a condition in an if statement, Python returns True or False:

Example

Print a message based on whether the condition is True or False:

```
 a = 200
b = 33
 if b > a:
print("b is greater than a")
else:
 print("b is not greater than a")
```

Evaluate Values and Variables

The bool() function allows you to evaluate any value, and give you True or False in return,

Example

Evaluate a string and a number:

```
 print(bool("Hello"))
print(bool(15))
```

Example

Evaluate two variables:

```
 x = "Hello"
y = 15

print(bool(x))
 print(bool(y))
```

Most Values are True

Almost any value is evaluated to True if it has some sort of content.

Any string is True, except empty strings.

Any number is True, except 0. Any list, tuple, set, and dictionary are True, except empty ones.

Example

The following will return True:

```
 bool("abc")
 bool(123)
 bool(["apple", "cherry", "banana"])
```

Some Values are False In fact, there are not many values that evaluates to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

Example

The following will return False:

```
bool(False)
bool(None)
 bool(0)
 bool("")
bool(())
 bool([])
bool({})
```

One more value, or object in this case, evaluates to False, and that is if you have an object that is made from a class with a __len__ function that returns 0 or False:

Example

```
 class myclass():
 def __len__(self):
```

```
 return 0
 myobj = myclass()
 print(bool(myobj))
```

Functions can Return a Boolean You can create functions that returns a Boolean Value:

Example
Print the answer of a function:

```
def myFunction() :
 return True
 print(myFunction())
```

You can execute code based on the Boolean answer of a function:

Example
Print "YES!" if the function returns True, otherwise print "NO!":

```
def myFunction() :
 return True

if myFunction():
print("YES!")
 else:
 print("NO!")
```

Python also has many built-in functions that returns a boolean value, like the isinstance() function, which can be used to determine if an object is of a certain data type:

Example
Check if an object is an integer or not:

```
x = 200
print(isinstance(x, int))
```

8. Python Operators

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:
Arithmetic operators
Assignment operators
Comparison operators
Logical operators
Identity operators
Membership operators
Bitwise operators

Python Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator		
+	Addition	x + y
-	Subtraction	x – y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators Assignment operators are used to assign values to variables:

Operator		
=	x = 5	x = 5

+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators Comparison operators are used to compare two values:

Operator		
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators
Logical operators are used to combine conditional statements:

Operator
and Returns True if both statements are true x < 5 and x < 10
or Returns True if one of the statements is true x < 5 or x < 4
not Reverse the result, returns False if the result is true not (x < 5 and x < 10)

Python Identity Operators
Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator
is Returns True if both variables are the same object x is y
is not Returns True if both variables are not the same object x is not y

Python Membership Operators
Membership operators are used to test if a sequence is presented in an object:

Operator
In Returns True if a sequence with the specified value is present in the object x in y not
in Returns True if a sequence with the specified value is not present in the object x not in y

Python Bitwise Operators
Bitwise operators are used to compare (binary) numbers:

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1
^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

9. Python Strings

String Literals
String literals in python are surrounded by either single quotation marks, or double quotation marks.

'Champak Roy' is the same as "Champak Roy".

You can display a string literal with the print() function:
Example

```
print("Champak Roy")
print('Champak Roy')
```

Assign String to a Variable Assigning a string to a variable is done with the variable name followed by an equal sign and the string:
Example

```
a = "Champak"
print (a)
```

multiline Strings You can assign a multiline string to a variable by using three quotes: Example You can use three double quotes:

```
a = """Champak lives in Varanasi. He writes books"""
print (a)
```

Or three single quotes:
Example

```
a = '''Champak lives in Varanasi, he writes books, he makes programs. '''
print(a)
```

Note: in the result, the line breaks are inserted at the same position as in the code.

Strings are Arrays
Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.

However, Python does not have a character data type, a single character is simply a string with a length of 1.
Square brackets can be used to access elements of the string.

Example
Get the character at position 1 (remember that the first character has the position 0):

```
a = "Hello, World!"
print(a[1])
```

Slicing You can return a range of characters by using the slice syntax.

Specify the start index and the end index, separated by a colon, to return a part of the string.
Example
Get the characters from position 2 to position 5 (not included):

```
b = "Hello, World!"
print(b[2:5])
```

Negative Indexing Use negative indexes to start the slice from the end of the string:
Example
Get the characters from position 5 to position 1,
starting the count from the end of the string:

```
b = "Hello, World!"
print(b[-5:-2])
```

String Length To get the length of a string, use the len() function.
Example
The len() function returns the length of a string:

```
a = "Hello, World!"
print(len(a))
```

String Methods Python has a set of built-in methods that you can use on strings.
Example
The strip() method removes any whitespace from the beginning or the end:

```
a = " Hello, World! "
print(a.strip())
```

# returns "Hello, World!"
Example
The lower() method returns the string in lower case:
a = "Hello, World!"
print(a.lower())
Example
The upper() method returns the string in upper case:
a = "Hello, World!"
print(a.upper())
Example
The replace() method replaces a string with another string:
a = "Hello, World!"
print(a.replace("H", "J"))
Example
The split() method splits the string into substrings if it finds instances of the separator:
a = "Hello, World!"
print(a.split(","))
# returns ['Hello', ' World!']

Learn more about String Methods with our String Methods Reference Check String To check if a certain phrase or character is present in a string, we can use the keywords in or not in.
Example
Check if the phrase "ain" is present in the following text:
txt = "The rain in Spain stays mainly in the plain"
x = "rain" in txt print(x)

Example
Check if the phrase "rain" is NOT present in the following text:
txt = "The rain in Spain stays mainly in the plain"
x = "rain" not in txt
print(x)
String Concatenation To concatenate, or combine, two strings you can use the + operator.

Example Merge variable a with variable b into variable
c: a = "Hello"
b = "World"
c = a + b
print(c)

Example
To add a space between them, add a " ":
a = "Hello"
b = "World"
c = a + " " + b
print(c)

String Format As we learned in the Python Variables chapter, we cannot combine strings and numbers like this:

Example
age = 36
txt = "My name is John, I am " + age print(txt)
But we can combine strings and numbers by using the format() method!
The format() method takes the passed arguments, formats them, and places them in the string where the placeholders {} are:

Example
Use the format() method to insert numbers into strings:
age = 36
txt = "My name is John,
and I am {}"
print(txt.format(age))
The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example

```
quantity = 3
item no = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} rupees."
print(myorder.format(quantity, itemno, price))
```

You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} rupees for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Escape Character To insert characters that are illegal in a string, use an escape character. An escape character is a backslash \ followed by the character you want to insert.
An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example
You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called "Vikings" from the north."
```

To fix this problem, use the escape character \":

Example
The escape character allows you to use double quotes when you normally would not be allowed:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Other escape characters used in Python:

Code	Result	Try it
\'	Single Quote	
\\	Backslash	
\n	New Line	
\r	Carriage Return	
\t	Tab	
\b	Backspace	
\f	Form Feed	
\ooo	Octal value	
\xhh	Hex value	

String Methods Python has a set of built-in methods that you can use on strings.

Note: All string methods returns new values. They do not change the original string. Method Description
capitalize() Converts the first character to upper case
casefold() Converts string into lower case center() Returns a centered string
count() Returns the number of times a specified value occurs in a string
encode() Returns an encoded version of the string
endswith() Returns true if the string ends with the specified value
expandtabs() Sets the tab size of the string
find() Searches the string for a specified value and returns the position of where it was found
format() Formats specified values in a string format_
map() Formats specified values in a string
index() Searches the string for a specified value and returns the position of where it was found
isalnum() Returns True if all characters in the string are alphanumeric isalpha() Returns True if all characters in the string are in the alphabet
isdecimal() Returns True if all characters in the string are decimals
isdigit() Returns True if all characters in the string are digits
isidentifier() Returns True if the string is an identifier
islower() Returns True if all characters in the string are lower case
isnumeric() Returns True if all characters in the string are numeric
isprintable() Returns True if all characters in the string are printable
isspace() Returns True if all characters in the string are whitespaces istitle() Returns True if the string follows the rules of a title
isupper() Returns True if all characters in the string are upper case join() Joins the elements of an iterable to the end of the string

ljust() Returns a left justified version of the string lower() Converts a string into lower case
lstrip() Returns a left trim version of the string maketrans() Returns a translation table to be used in translations
partition() Returns a tuple where the string is parted into three parts replace() Returns a string where a specified value is replaced with a specified value
rfind() Searches the string for a specified value and returns the last position of where it was found
rindex() Searches the string for a specified value and returns the last position of where it was found rjust() Returns a right justified version of the string
rpartition() Returns a tuple where the string is parted into three parts rsplit() Splits the string at the specified separator, and returns a list .
rstrip() Returns a right trim version of the string split() Splits the string at the specified separator, and returns a list
splitlines() Splits the string at line breaks and returns a list startswith() Returns true if the string starts with the specified value
strip() Returns a trimmed version of the string swapcase() Swaps cases, lower case becomes upper case and vice versa
title() Converts the first character of each word to upper case translate() Returns a translated string
upper() Converts a string into upper case
zfill() Fills the string with a specified number of 0 values at the beginning