Topics -

1. About Python.
2. Data Types.
3. Variables, keywords and identifiers.
4. Python input/output.

Why Python ?

Although Python (1980) was created before Java (1995), it was not as popular.
But, in the current times, Python is close to (if not) ahead of Java.

Note - Python is an interpreted language which means the code is executed line by line (not at once).

Sequence -
Step 1 - Convert source code to byte code (of first line).
Step 2 - Execute byte code (of first line) using Python Virtual Machine.

What made Python so popular ?

1.Design philosophy -
Python contains all those things which are required in a programming language.
It is so easy to learn that it can be learned even by a class 6 student.
Python code is very well-indented, readable and looks pretty aesthetic.

2.Batteries included -
Python always comes with batteries included - this means that it is already equipped with various tools, libraries, functions, operators etc. which makes it very easy to use.
For example - To reverse a string in C/C++, we need to write an entire code from scratch.
But this can be easily done in python with just one line of code.
Python works on the philosophy of - "Don't rebuild the wheel".
(If the wheel is already made, then why recreate the wheel, instead use the wheel to make a car.)

3.General Purpose -
Python is a general purpose programming language which means that python supports various kinds of programming styles such as procedural programmng, object-oriented programming (OOP), functional programming.
There is also no limitation of the kind of software that can be created.
For example - We can create desktop apps, websites, data science coding etc.

4.Libraries/Community -
The community of python developers is actually very helpful and keeps on working to provide us with new libraries and other stuff.
This is the reason that python is used in data science because there are a number of libraries such as pandas, numpy, seaborn, matplotlib etc. which are very helpful in making sense of data.
This helps us a lot as we don't need to write any code from scratch, instead we can use some library.

Why python for Data Science ?

Note - On an industrial level, python is used but on a research/academic level, "R" is used.

Reason 1- Easy to use and learn.

Data Science was originally used by people working in the domain of mathematics and statistics.
Their goal was to convert that knowledge (algorithms) into code, so they wanted a language which is very easy to use and learn (as they are mathematicians, not programmers) - hence, the most obvious choice was python.

Reason 2- Proximity with maths.

Python is filled with libraries which are related to the mathematics domain such as numpy (which literally stands for numerical python), scipy etc.

Reason 3- Community.

The community of python is very big and keeps on creating newer libraries for everyone to use.
This is not present in Java or C++. (It will take them a long time to reach the level of python)

Note - The python which we will be learning is going to be from the perspective of Data Science and not Software Development.

Topic 1 - Python Output /print function.

The most basic thing to be taught at first in programming is how to display something on screen.
In python, to display something on screen, we use the print( ) function.
Whatever, we need to dsiplay is put inside the print function.

Note - A function is something that takes an input and then does some computation, then returns an output.
Any function has the form - function_name(input).
The input is given in the brackets after the function name is used.
"print" is an inbuilt function i.e it comes with python.

Comments - They are used to simplify the code by adding a code description, so if someone else reads our code, it can be very easily understood by them.
They are written after writing a # symbol.
They are ignored by the python interpreter, so they create no errors or problems.

In [85]:
```
# This is a comment and will be ignored, hence it won't have an output
```

In [4]:
```
# Note - Anything written after # symbol is a comment and will be ignored
# Since "Hello World" is a string/text, we pass it inside inverted commas

print("Hello World")
```

```
Hello World
```

If we are displaying numbers, then we can pass them inside python as is (without inverted/double invert commas).
However, if we are displaying a string i.e a collection of words, then it needs to be passed inside inverted commas inside the print function.

In [3]:
```
# If we want to display numbers, then we can display them as is (no inverted commas)

print(24)
```

```
24
```

Note - Python is a case sensitive language which means print( ) and Print( ) are different.
We must always use the lowercase one i.e print( ).

In [7]:
```
# We can print a variety of different types of data.

# String/Text -
print("Hello World")

# Number -
print(69)

# Decimal/Float -
print(69.69)

# Boolean - (True/False or 0/1)
print(True)

# We can print different data types together also -
print("Hello", 1, True, 3.14)

# Note - We can see that only strings require the input to be sent in
```

```
Hello World
69
69.69
True
Hello 1 True 3.14
```

In [11]:
```
# When printing multiple data types together, we can see that they are
# This is because print function has a parameter called as sep=' ', wh
# We can override it as follows -

# sep=',' - Here, we are going to set sep to a comma(,)
print("Hello", 1, True, 3.14, sep=',')

# sep='/' - Here, we are going to set sep to a slash(/)
print("Hello", 1, True, 3.14, sep='/')

# sep='-' - Here, we are going to set sep to a hyphen(-)
print("Hello", 1, True, 3.14, sep='-')

# sep='....', Here, we are going to set sep to four dots(....)
print("Hello", 1, True, 3.14, sep='....')


# As we can see, we can set the separator/sep to any value of our choi
```

```
Hello,1,True,3.14
Hello/1/True/3.14
Hello-1-True-3.14
Hello....1....True....3.14
```

In [12]:
```
# Whenever we are printing something together, it will automatically b
# For example -

print("Hello")
print("World")
```

```
Hello
World
```

In [14]:
```
# As we can see that both "Hello" and "World" were printed on differer
# If we want we can change this behaviour using the end='' parameter.
# It is set to end='\n' by default, which means a new line is entered
# \n means line change.

print("Hello", end='-')
print("World")

# Here, we have set the value of end to hyphen(-).
# So, this print statement will be connected to the next print stateme
# or we can say that the first print will end with hyphen(-).
```

```
Hello-World
```

Topic 2 - Data Types

Every programming language has some data types.
A data types tells us about the types of data that a language can work with.

In [15]:
```
# Type 1 - Integer (both positive and negative)
# This is a numeric datatype.
# Python can handle integer values up to 1e308 (means 1*10^308)

print(75)
```

```
75
```

In [16]:
```
#Type 2 - Decimal/Float
# This is also a numeric datatype.
# Python can handle decimal values up to 1.7e308

print(748.1234)
```

```
748.1234
```

In [17]:
```
# Type 3 - Boolean (True/False or 0/1)
# Note - Boolean is not text (so, no need to place it in inverted comm

print(True)
print(False)
```

```
True
False
```

In [18]:
```
# Type 4 - Text/String
# This needs to be placed in inverted/double inverted commas.
# Note - Python does not have 'char' data type like C language, instea

print("My name is Ayush")
```

```
My name is Ayush
```

In [20]:
```
# Type 5 - Complex
# This is a numeric datatype.
# This is used very less and comes from mathematics of imaginary numbe

print(5 + 6j)

# Here, 5 is the real part and 6 is the imaginary part of the complex
```

```
(5+6j)
```

In [21]:
```
# Type 6 - List
# This is a sequence data type and serves the same purpose which is se
# A list is displayed inside square brackets which contain comma separ
# For example - [1,2,3,4]

print([1,2,3,4])
```

```
[1, 2, 3, 4]
```

In [22]:
```
# Type 7 - Tuple
# This is also a sequence data type but it is displayed using regular
# The differences between list and tuple will be coming in future.

print((1,2,3,4))
```

```
(1, 2, 3, 4)
```

In [23]:
```
# Type 8 - Sets
# This is the same set which is present in mathematics.
# A set is displayed using curly brackets.

print({1,2,3,4})
```

```
{1, 2, 3, 4}
```

In [24]:
```
# Type 9 - Dictionary
# This is a mapping data type and uses key value pairs to store inform
# A dictionary is displayed using curly brackets.

print({'Name':"Ayush", "Gender":"Male"})
```

```
{'Name': 'Ayush', 'Gender': 'Male'}
```

To check the datatype of any data, we use the type( ) function.

In [4]:
```
# To check the data type of a number such as 2.
# That number is then passed as an argument inside type() in brackets.

type(2)

# As we can see that 2 is an integer.
```

Out[4]: int

In [27]:
```
type(2+3j)
```

Out[27]: complex

In [28]:
```
type("Hello")

# Hello is a string i.e text
```

Out[28]: str

In [29]:
```
type(True)

# True is boolean.
```

Out[29]: bool

In [30]:
```
type(4.5)
```

Out[30]: float

In [31]:
```
type([1,2,3])
```

Out[31]: list

Topic 3 - Variables, Keywords and Identifiers.

Variables are a way to store data, they act as a container to store data.
They are used extensively in programming where we don't know what input will be given by the user.

For example-
We are trying to create a simple program that greets a user such as "Hello,user_name"
Each time, someone logs in, the name of the user will be different such as "Hello, Ayush" or "Hello, Anshu" etc.

Here, since user name is going to be different each time, we can use a container/variable to get the name of the user and store it after "Hello", so that each time a new value comes, it can be printed as such.

In [33]:
```
# For example - Here we are going to create a variable called as "name
# Note - values are stored from right to left which means that "Ayush"

name = "Ayush"

# Now, we can print this variable using the print function.
print(name)

# Note - this time we did not print the value ("Ayush") directly, inst
# which contained the value.
```

```
Ayush
```

In [34]:
```
# Other examples of variables -

# Here 3 is stored in variable "a" and 9 is stored in variable "b"
a = 3
b = 9

print(a+b)
```

```
12
```

Interview Question - What is dynamic typing in python ?

In other languages like C/C++, we have to declare the datatype along with variable name.
For example - "int a = 7", this means we have created a variable "a", which can store integer values.
We do not need to do this in python, since python supports dynamic typing i.e we do not need to tell the data type of the variable, we just need to tell the value to be stored and python automatically understands the datatype which is stored in it.

Instead of doing "int a = 7", we can just do "a=7" in python.

Interview Question - What is dynamic binding in python ?

In python, the datatype of the variable is not fixed, instead it depends on the value stored in the data stored in the variable.

For example -
a = 10, here, the data type of a is integer as it has stored an integer (10) inside it.
If we now do, a = "Hello", Now, the data type of the same variable (a) is str/string.
In doing so, we will not encounter any error.

Other languages like C, C++, Java etc have static binding, which means that once the data type is set at the beginning, it cannot be changed for that particular integer.

For example -
If we write - int a = 10
Then, since "a" is now set as an integer, we can now no longer store any value of any other data type.
However, we can still store other integers if we want.

In [35]:
```
a = 10
print(a)

a = "Hello"
print(a)
```

```
10
Hello
```

Some stylish ways to declare variables in python.

In [37]:
```
# Method 1

a = 1
b = 2
c = 3

print(a, b, c)
```

```
1 2 3
```

In [39]:
```
# Method 2

a, b, c = 1, 2, 3
print(a, b, c)

# Here, all the variables will be assigned in their respective order.
# a will be given 1, b will be given 2 and c will be given 3.
```

```
1 2 3
```

In [40]:
```
# Method 3
# This is used to give the same value to multiple variables.

a=b=c=10

print(a, b, c)
```

```
10 10 10
```

What is compilation ?

We as programmers use English language to write the source code which is high level in nature.
But, the computer/processor only understands binary language (1's and 0's) which is low level in nature.
So, we need to convert this high level language to low level language.
This process is known as compilation.

Who does this compilation ?

This compilation is done either by a compiler or an interpreter.
Some languages which are called as compiled languages such as C/C++, Java have a compiler.
Whereas some languages are called as interpreted languages such as Python, PHP etc have an interpreter.

What is the difference between both ?

Compiler translates high level source code (ours) into binary at once, then executes the program.
Whereas interpreter translates the source code to binary code line by line.

Keywords

What are keywords ?

There are certain words which are reserved by the compiler/interpreter which help it to understand what the programmer is trying to do.
Here, reserved means that they can't be created/defined by the user as they are already in use by the compiler.
If they are used forcefully, then it will create an error as python will get confused.
For example -
"print" is a keyword as it is reserved by python and python knows that the user is trying to display something.
In python, there are a total of 32 keywords.

The whole idea is that we should not use keywords as variable names.

Identifiers

What are identifiers ?

The names that we use inside a program are called as identifiers.
For example - If we write a=10, Here, "a" is an identifier (variable name).
These names can be variable names, class names, function names etc.

There are certain rules to be used while creating an identifier-

1. An identifier can't start with a digit.
2. We can use uppercase, lowercase letters etc, but from special characters, we can only use underscore(_).
3. We can even use just an underscore as a variable name.
4. Keywords can't be used as an identifier (as they are already reserved by the interpreter).

Topic 4 - Python Input/Output

To take the input from the user, we use the input function, which returns us a string.
For example - If we want to take the name of the user and then store it in a variable called as user_name.

Note - whatever we write inside the input function will be displayed as a message to the user

In [45]:
```
user_name = input("Please enter your name = ")
print(user_name)
```

```
Please enter your name = Ayush
Ayush
```

Now, let's write a simple addition program which takes two numbers from the user and returns their sum.

Here, we want to get a number as an input, but input function returns a string.
So, to avoid this problem, we convert the string into a number, by using the int() function.
Anything that goes into int( ) is converted into a number(if it is already a number but in string format.

This is known as type conversion.

Why does python return in string format ?
It does so because string is a universal format.
Any datatype can be stored in a string but the reverse is not true.
For example - We can store "4"(number) as a string, but we can't store "Kolkata"(string) as a number.

Type conversion - This is used to convert one datatype into another.
To do this, we simply pass the input into the respective datatype.

For example -
If we want to convert anything to a number, we pass it inside the int( ) function.
If we want to convert anything to a string, we pass it inside the str( ) function.
If we want to convert anything to a float, we pass it inside the float( ) function.

Note - All these int( ), str( ), float( ) etc are built-in functions.

Very Important - Type conversion is only possible, when it is logically possible.
For example - We can't convert a string to an integer or complex to an integer.

In [52]:
```
# First we take a number but in string format (i.e in double quotes),

# String format
a = "74"
print(a, type(a))

# Number format
b = int(a)
print(b, type(b))

# Although, they both look like numbers, but a contains a string and b
```

```
74 <class 'str'>
74 <class 'int'>
```

Now, let's write a simple addition program which takes two numbers from the user and returns their sum.

In [54]:
```
# Take the first number from user.
a = input("Please enter the first number = ")

# Take the second number from the user.
b = input("Please enter the second number = ")

# Do the type conversion of  a and b into numbers before addition, sir
c = int(a) + int(b)

print("The result is = ", c)
```

```
Please enter the first number = 5
Please enter the second number = 16
The result is =  21
```

Note - Type conversion does not change the data type permanently, rather it changes it for that particular instance by creating a new value.
No change takes place to the original data.

In [56]:
```
# For example - If we check the datatype of our variable a and b, they

print(type(a))
print(type(b))
```

```
<class 'str'>
<class 'str'>
```

Literals

As we already know that a variable is a container for storing a value.
This value that we store (in raw form) is called as a literal.

For example -
a = 2, Here 'a' is the variable, and '2' is the literal.

Integer Literals - This means that in how many ways can we give an integer value as an input.

In [57]:
```
# Decimal literal
a = 100
print(a)
```

```
100
```

In [58]:
```
# Binary literal
# This 0b tells python that we are giving a binary number(1010) as an
# 1010 is the binary equivalent of 10.

a = 0b1010
print(a)
```

```
10
```

In [59]:
```
# Octal literal
# This 0o tells python that we are giving an octal number(310) as an i
# 310 is the octal equivalent of 200

a = 0o310
print(a)
```

```
200
```

In [61]:
```
# Hexadecimal literal
# This 0x tells python that we are giving a hexadecimal number(12c) as
# 12c is the hexadecimal equivalent of 300

a = 0x12c
print(a)
```

```
300
```

Float Literals - This means that in how many ways can we give an float value as an input.

In [62]:
```
# Method 1 - used for smaller numbers.

a = 10.5
print(a)
```

```
10.5
```

In [63]:
```
# Method 2 - used for larger numbers.

# This means 3.5*10^4
a = 3.5e4
print(a)
```

```
35000.0
```

In [65]:
```
# Method 3 - used for very small numbers.

a = 3.5e-4
print(a)
```

```
0.00035
```

Complex Literals - This means that in how many ways can we give an complex value as an input.

In [66]:
```
a = 3 + 4j
print(a)
```

```
(3+4j)
```

In [67]:
```
# To get the real and imaginary part separately

print(a.real)
print(a.imag)
```

```
3.0
4.0
```

String Literals - This means that in how many ways can we give an string value as an input.

In [68]:
```
# Method 1 - Single inverted commas.

a = 'Hello'
print(a)
```

```
Hello
```

In [69]:
```
# Method 2 - Double inverted commas.

a = "Hello"
print(a)
```

```
Hello
```

In [70]:
```
# Method 3 - Single character.

a = "X"
print(a)
```

```
X
```

In [73]:
```
# Method 4 - Unicode characters (used for emojis and emoticons)
# Here, the 'u' written outside inverted commas is used to tell pythor

a = u"\U0001F923"
print(a)
```

```
🤣
```

In [77]:
```
# Method 5 - Raw strings (used for \n, \t etc)
# Note - raw strings can't be printed without this method.

a = r"raw \n string"
print(a)
```

```
raw \n string
```

Boolean Literals - This means that in how many ways can we give an boolean value as an input.
Note - Internally, python considers True as "1" and False as "0".

In [78]:
```
# Here, True is "1", which when added to 2 gives 3.
print(True + 2)
```

```
3
```

In [82]:
```
# Here, False means 0, which when added to 5 gives 5.
print(False + 5)
```

```
5
```

In python, there is no way to declare a variable beforehand. Whenever we need a variable, we can declare it on the spot.
So, if we want to declare a variable but we don't have a use for it, then we can assign None to that variable (as the variable can't be kept empty).

In [84]:
```
k = None
print(k)

# If we want, then in future, we can change the value of k from None t
```

```
None
```

END OF SESSION 1.