Python Introduction

- Python is a high-level, interpreted programming language.
- Every language has some set of rules & regulations, syntaxes, keywords, data types, data sets ....etc
- a word is a sequence of characters or form group of characters.
- In Python char is classified into two types 1. Numeric and 2. Non-Numeric
    - Numeric Data types (**int, float, complex**)
    - Non-Numeric Data types (**Alphabets (str) & Special char (#, _**)

```
x=100
y='100'
print(x,y,type(x),type(y))
```

```
100 100 <class 'int'> <class 'str'>
```

```
z=int('01')
print(x,type(z))
```

```
100 <class 'int'>
```

```
print(z,type(z))
```

```
1 <class 'int'>
```

int: it is a one kind of data type in python

- int('01') value is 1 , it is an integer class it stores the value as bits format, every value has base value by default value is 10 , int range is 2>= <=32.
- base value should any of these in number system : 2 -> binary, 8 -> Octal, 10 -> decimal, 16 ->hexa decimal.
- The int in python is a built-in method that converts a string or a number into an integer.
- We can also use int in python to convert binary numbers to decimal numbers, hexadecimal numbers to decimal numbers, and octal numbers into decimal numbers.

float: it is a used to declare fractional vlues or point values

- converts values into floating point numbers.
- Floating point numbers are decimal values or fractional numbers like 133.5, 2897.11, and 3571.213, whereas real numbers like 56, 2, and 33 are called integers.

```
x=1.34
print(x,type(x))
```

```
1.34 <class 'float'>
```

```
y=float(134)
print(y,type(y))
```

```
134.0 <class 'float'>
```

```
z=float('10.123')
print(z,type(z))
```

```
10.123 <class 'float'>
```

Complex: it is a combination of real and imaginary number

- Python complex() Function The complex() function returns a complex number by specifying a real number and an imaginary number.
- The complex data type in python consists of two values, the first one is the real part of the complex number, and the second one is the imaginary part of the complex number.
- We usually denote the real part using i and the imaginary part with j. For example, (3 + 7j) or (3i + 7j).

```
x=1+2j
print(x,type(x))
```

```
(1+2j) <class 'complex'>
```

string: it is a sequence of characters enclosed in a single or double or triple single quotes or triple double quotes.

- a string is a sequence of characters enclosed within either single quotes (' ') or double quotes (" ").
- It is an immutable data type, which means once a string is created, it cannot be modified.
- However, it is possible to create a new string by concatenating two or more strings.
- indexing is poaible for sequential or ordered data types

```
str1="Hello world"
print(str,type(str))
```

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

```
str1[0]
```

```
'H'
```

```
str1[-1]
```

```
'd'
```

```
print(str1[::], str1[::-1], str1[0:len(str1):2],str1[-1:-len(str1)-1:-1], sep='\n')
```

```
Hello world
dlrow olleH
Hlowrd
dlrow olleH
```

- indexing starts with '0' in python, defalut starting value is '0' end value is len(str) and step value is '1' in **Positive/ Left to right indexing**.
- indexing starts with '-1' in python, defalut starting value is '-1' end value is -len(str)-1 and step value is '-1' in **Negative/ Right to left indexing**.

```
print(len(str1),type(str1),id(str1))
```

```
11 <class 'str'> 132297045148080
```

Mutable & Immutable data types

- mutable means changable - after delcare data type we can make any changes like add/remove elements, immutable means unchangable once declared we can't make any changes.
- mutable data types in Python (**Lists, Sets and dictionaries**)
- immutable data types are all numeric data types **(int,float,complex), strings and tuples**.
- sequentail , ordered dat types are:
  - strings
  - Lists
  - Tuples
- dictionaries are not sequential but ordered
- Sets are neither sequential nor ordered

```
l=[1,'abc',1.25,1+4j,print,bool]
print(l,type(l),len(l),id(l))
```

```
[1, 'abc', 1.25, (1+4j), <built-in function print>, <class 'bool'>] <class 'list'> 6 132296812461888
```

```
x="Hello World"
print(x,type(x))
```

```
Hello World <class 'str'>
```

```
s1={1,'xyz',1.23,'hello',bool,print}
print(s1,type(s1))
```

```
{1, 1.23, 'hello', <built-in function print>, <class 'bool'>, 'xyz'} <class 'set'>
```

```
d={'name':'mahender','id':305001,'address':'Hyd'}
print(d,type(d))
```

```
{'name': 'mahender', 'id': 305001, 'address': 'Hyd'} <class 'dict'>
```

```
t=(1,'aaa',1.45,1+4j,print,bool)
```

```
print(t,type(t))
```

```
(1, 'aaa', 1.45, (1+4j), <built-in function print>, <class 'bool'>) <class 'tuple'>
```

Variable is a container or storage location which stores assigned value.

- Python is case sensitive language
- Python has no command for declaring a variable
- A variable is created the moment you first assign a value to it.

Rules to create a variable:

- 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)
- A variable name cannot be any of the Python keywords.
- Python has a set of keywords that are reserved words that cannot be used as variable names, function names, or any other identifiers Ex: and, del, for, if , in, is , True, ... etc

```
v1=123
var='abc'
_var1="a1"
var_2=1.25
newVar='hello'
```

Operators in Python

- Operator is nothing but a symbol or character which performs a predefined operation.
  - Arithmetic (+, -, *, /, //, %)
  - comparision (==,!=,>=,<=,>,<)
  - identity (is ,is not)
  - membership (in, not in)
  - Logical (and &, or |, not ~)
  - Bitwise (&, |, ~, <<, >>)
  - Arithmetic assignment (+=, -=, *=, /=, //=, %=)

```
x=100
y=200
z=x+y
z
```

```
300
```

```
z=x-y
z
```

```
-100
```

```
z=x*y
z
```

```
20000
```

```
z=x/y
z
```

```
0.5
```

```
z=x//y
z
```

```
0
```

```
z=x%y
z
```

```
100
```

```
x==y
```

```
False
```

```
x!=y
```

```
True
```

```
x>y
```

```
False
```

```
x<y
```

```
True
```

```
x>=y
```

```
False
```

```
x<=y
```

```
True
```

```
x is y
```

```
False
```

```
x is not y
```

```
True
```

```
x1=[1,2,3,4,5]
x2=[3,4]
x2 in x1
```

```
False
```

```
x1 in x2
```

```
False
```

```
3 in x1
```

```
True
```

```
3 in x2
```

```
True
```

```
100 not in x1
```

```
True
```

```
5 not in x1
```

```
False
```

Logical operators

- **AND** if both statements are true then it return True remaining case returns False. (T + T --> True)
- **OR** if any one statement is true returns True remining cases False (T + F --> True , F + T --> True)
- **NOT** reverse of statement

Bitwise Operators:

- **& AND** if both '1' returns '1' (1 + 1 --> 1)
- | **OR** if atleast one '1' return '1' (1 + 1--> 1 or 1 + 0 --> 1)
- **^ XOR** if 1 returns '0', if 0 returns 1
- **~ NOT** reverse to given statement if 1 return 0, if 0 returns 1.
- **<< Shift Left** Shift left by pushing zeros in from the right and let the leftmost bits fall off and adds zero at end
- **>> Shift Right** Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off and adds zero at begining.

Strings

- strings are a `seq` of `chars` enclosed in either `single qoutes` or `double qoutes` or `triple single or triple double qoutes`. Strings are `ordered`
- since strings are `seq` and `ordered` we can perform `indexing` and `slicing` of strings

---

- in strings, we can index the chars using either a
  - `positive indexing`
    - (starts qith `0` and ends with `len(stringObject) - 1`) `positive indexing` moves from `left to right`
  - `negative indexing`
    - (starts with `-1` and ends with `-len(stringObject)`) `negative indexing` moves from `right to left`

> Syntax for indexing

```
stringObject[indexValue]
```

- in strings, `slicing` is possible from `either side`. We control this by making use of the `step` value.

> Syntax for slicing

```
stringObject[startIndex : endIndex : stepValue]
```

- default value of stepValue is +1
  - when stepValue is `positive`,
    - default values of `startIndex` and `endIndex` are `0` and `len(stringObject)`
    - slicing happens from `left to right`
  - when the the `stepValue` is 'negative`
    - default `startIndex` and `endIndex` are `-1` and `-(len(stringObject))-1`
    - slicing happens from `right to left`
- `sign` of the `stepValue` indicates the direction.
  - `positive stepValue` indicates `left to right`
  - `negative stepValue` indicates 'right to left`
- `magnitude of stepValue` indicates how many `steps` to taken.

- `triple single or triple double qoutes` are used for muliti-line strings, multi-line comments and doc strings.
- `strings` are `immutable` in nature.

Creating a String:

singleQouteString = 'This string is enclosed in a single qoute'

doubleQouteString = "This string is enclosed in a double qoutes"

tripleSingleQouteString = ''' This string is a- -multi-line string enclosed in- -triple single qoutes. '''

tripleDoubleQouteString = """ This string is a- -multi-line string enclosed in- -triple double qoutes. """

escape char:

- /

```
#x = "this is Ravi's code. Ravi said "hello!"" #uncomment to see the error

x = 'this is Mahi\'s code'
x
```

```
'this is Mahi's code'
```

```
y = '\''
print(y)
```

```
'
```

```
z = '\\'
print(z)
```

```
\
```

```
x = '''this is line-1
this is line - 2
this is line - 3'''
print(x)
x
```

```
this is line-1
this is line - 2
this is line - 3
'this is line-1\nthis is line - 2\nthis is line - 3'
```

```
x = 'this is a string enclosed in single qoute\n line - 2 follows'
print(x)
x
```

```
this is a string enclosed in single qoute
 line - 2 follows
'this is a string enclosed in single qoute\n line - 2 follows'
```

```
x = 'this is a string enclosed in single qoute\tremaining string follows'
print(x)
x
```

```
this is a string enclosed in single qoute	remaining string follows
'this is a string enclosed in single qoute\tremaining string follows'
```

raw string:

- r
- it treats a string as it is.
- It won't consider the escape chars
- we write a raw - string as r'stringBody'

```
filePath = 'c:\Desktop\newFolder\newFile.txt'
print(filePath)
filePath
```

```
c:\Desktop
ewFolder
ewFile.txt
'c:\\Desktop\newFolder\newFile.txt'
```

```
filePath = r'c:\Desktop\newFolder\newFile.txt' # raw string
print(filePath)
filePath
```

```
c:\Desktop\newFolder\newFile.txt
'c:\\Desktop\\newFolder\\newFile.txt'
```

f-string:

- f

```
x, y = 5, 10
print('x has a value', x, 'and y has a value', y)
print(f'x has a value {x} and y has a value {y}. sum of x and y is {x + y}')
# flower braces '{}' will act as place holders for vars' values
```

```
x has a value 5 and y has a value 10
x has a value 5 and y has a value 10. sum of x and y is 15
```

creating an empty string:

- st=''
- x=str()

```
a, b = str(), '' # "", '''''', """""" ---> empty string
print(f'''a and b have values {a}, {b} and belongs to {type(a), type(b)}
      The len a an b are {len(a), len(b)} respectivly''')
```

```
a and b have values ,  and belongs to (<class 'str'>, <class 'str'>)
      The len a an b are (0, 0) respectivly
```

Indexing & Slicing:

```
s = 'hello world'
```

```
print(s[0],s[-1],s[::],s[::-1],s[0:9:2], sep='\n')
```

```
h
d
hello world
dlrow olleh
hlowr
```

Strings are immutable data types:

- At times, when a stringMethod is called on a stringObject, it'll alter the stringObject.
- this altering or changing of original stringObject will not reflect in the orginal stringObject memory location.
- That is why strings are immutable nature.

```
s, hex(id(s))
```

```
('hello world', '0x7852ce62ca70')
```

String Methods:

```
s.upper()
```

```
'HELLO WORLD'
```

```
s.lower()
```

```
'hello world'
```

```
x = [1, 2, 3]
print(x, hex(id(x)))
```

```
[1, 2, 3] 0x7852c06d7240
```

```
# adding 4 at the end of x
print(x.append(4), hex(id(x.append(4))))
```

```
None 0x585d4f7213e0
```

```
print(x, hex(id(x)))
```

```
[1, 2, 3, 4, 4] 0x7852c06d7240
```

```
dir(str)
```

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
 'isupper',
 'join',
```

```
len(dir(str))
```

```
80
```

```
Start coding or generate with AI.
```