CHAPTER-INTRODUCTION TO PYTHON

INTRODUCTION

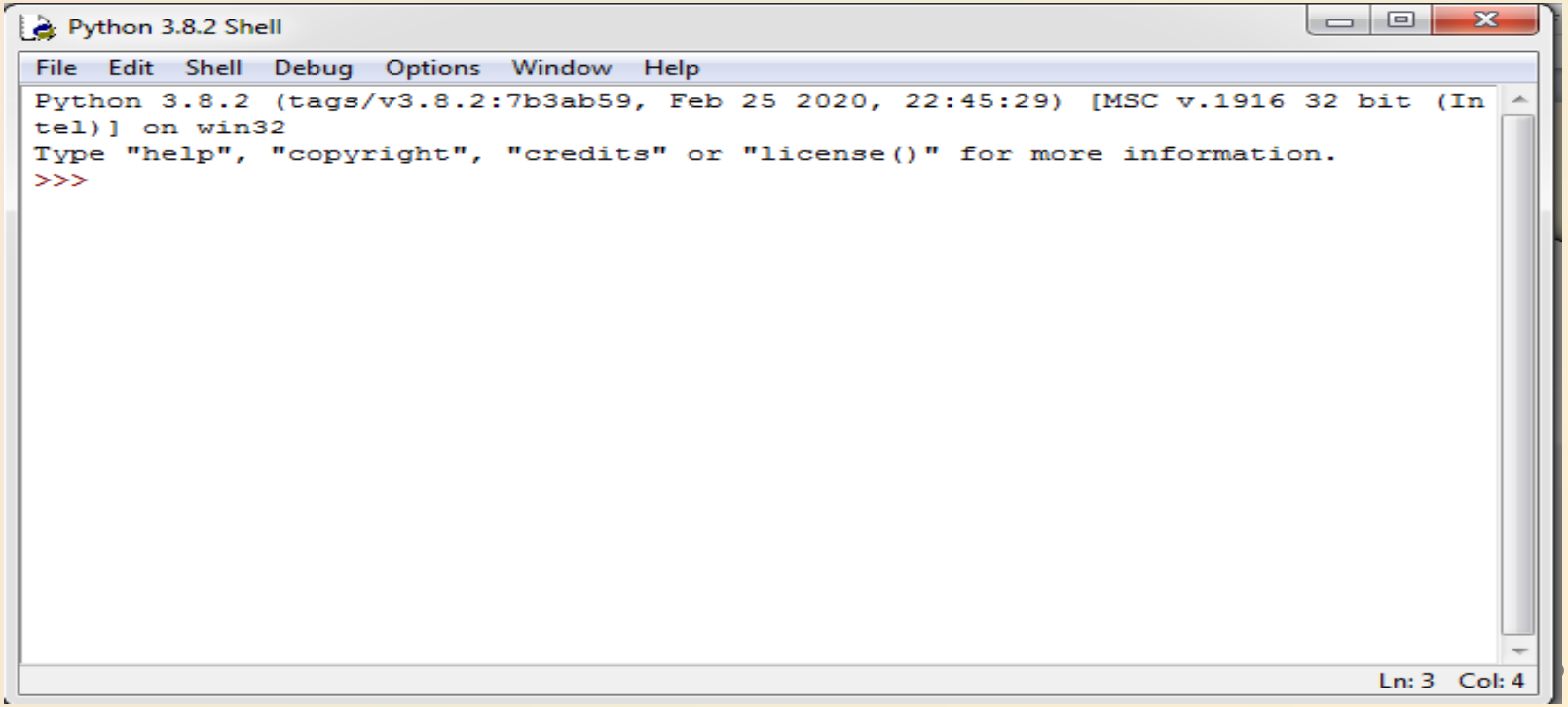
- **It is widely used general purpose, high level programming language. Developed by Guido van Rossum in 1991.**
- **It is used for:**
  software development,
  web development (server-side),
  system scripting,
  Mathematics.

Features of Python

- **Easy to use** – Due to simple syntax rule.
- **Interpreted language** – Code execution & interpretation line by line.
- **Cross-platform language** – It can run on windows, Linux, Macintosh, etc. equally.
- **Expressive language** – Less code to be written as it itself express the purpose of the code.
- **Completeness** – Support wide range of library
- **Free & Open Source** – Can be downloaded freely and source code can be modify for improvement

Starting with Python

- To write and run (execute) a Python program, we need to have a Python interpreter installed on our computer or we can use any online Python interpreter. The interpreter is also called Python shell.



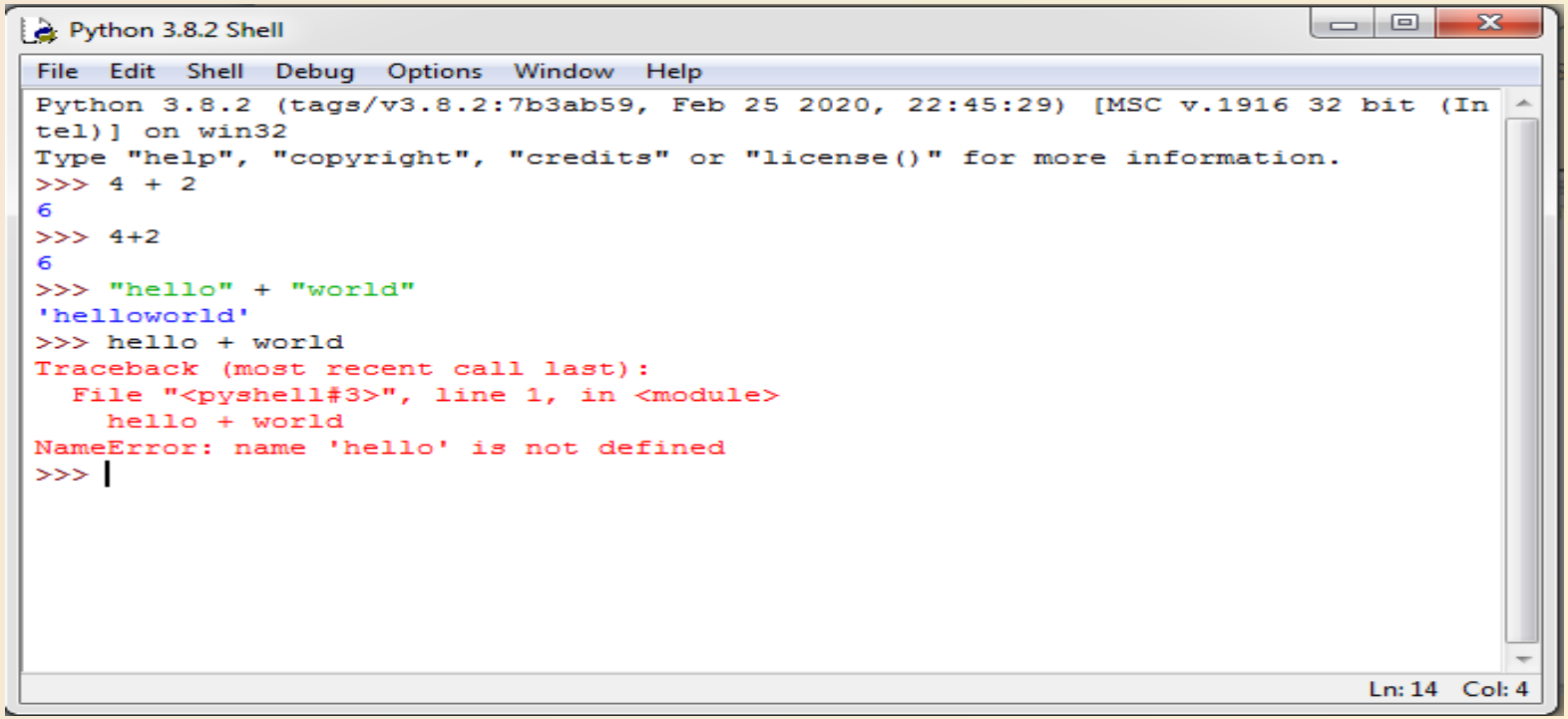
Execution modes

- There are two ways to use the Python interpreter:
  a) Interactive mode
  b) Script mode

- Interactive mode allows execution of individual statement instantaneously.
- Script mode allows us to write more than one instruction in a file called Python source code file that can be executed.

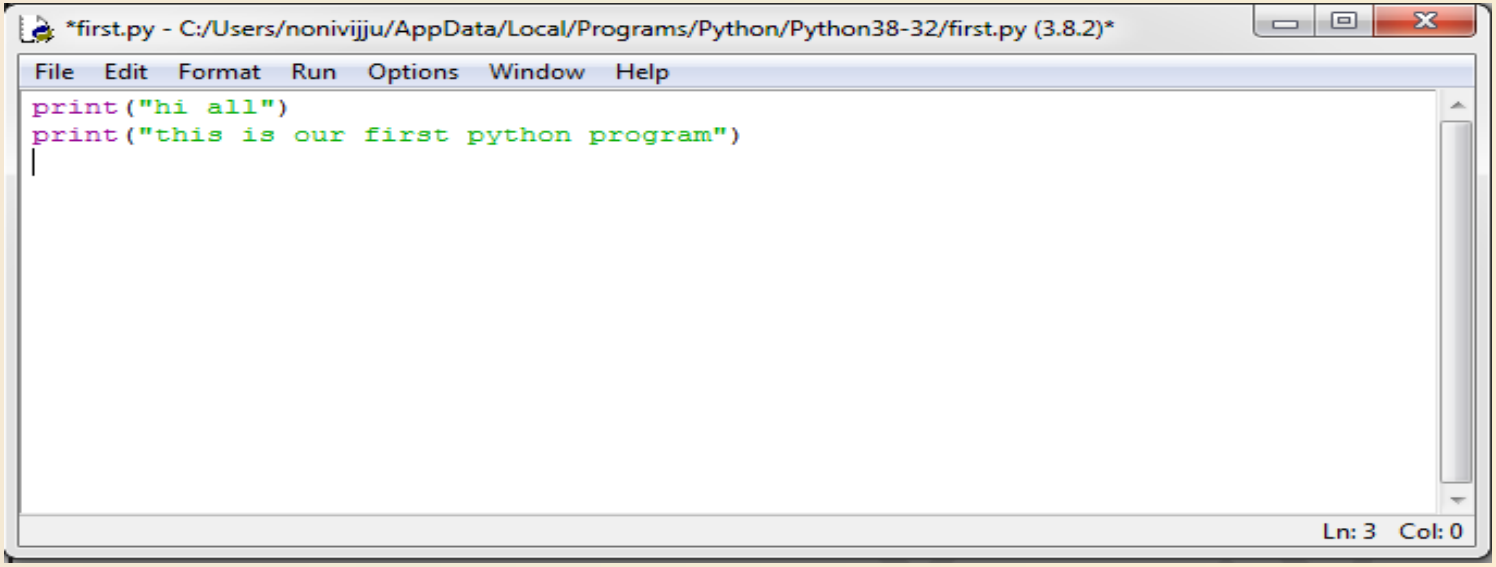
Interactive mode

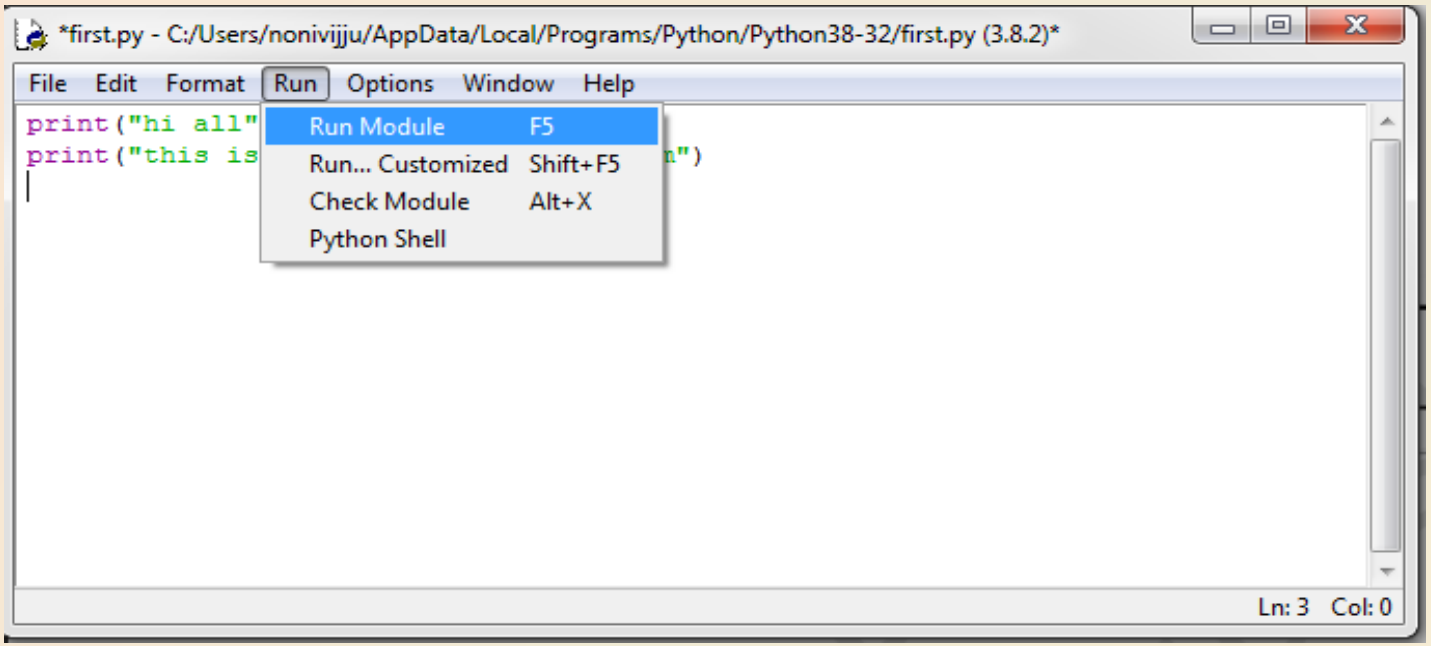
- Working in the interactive mode is convenient for testing a single line code for instant execution. **But**, we cannot save the statements for future use and we have to retype the statements to run them again.
- **Advantages:**
- Helpful when script is extremely short and need immediate results.
- Faster as we only have to type a command and then press the enter key to get the results.

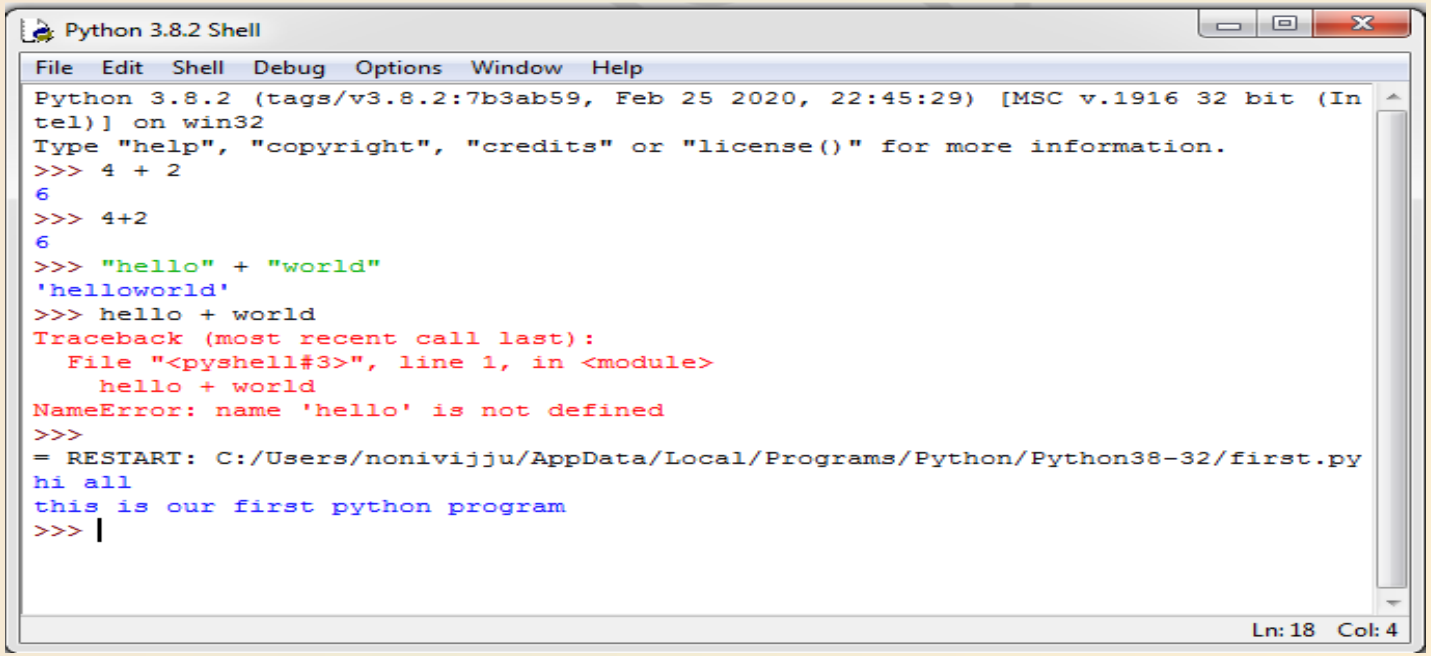


Script Mode

- In the script mode, we can write a Python program in a file, save it and then use the interpreter to execute it. Python scripts are saved as files where file name has extension ".py". By default, the Python scripts are saved in the Python installation folder.




first.py - C:/Users/nonivijju/AppData/Local/Programs/Python/Python38-32/first.py (3.8.2)
File Edit Format Run Options Window Help
print("hi all"
print("this is
n")
Run Module F5
Run... Customized Shift+F5
Check Module Alt+X
Python Shell
Ln: 3 Col: 0


Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 4 + 2
6
>>> 4+2
6
>>> "hello" + "world"
'helloworld'
>>> hello + world
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
hello + world
NameError: name 'hello' is not defined
>>>
= RESTART: C:/Users/nonivijju/AppData/Local/Programs/Python/Python38-32/first.py
hi all
this is our first python program
>>>
Ln: 18 Col: 4

Python Character Set

- A set of valid characters recognized by python. Python uses the traditional ASCII character set. The latest version recognizes the Unicode character set. The ASCII character set is a subset of the Unicode character set
  Letters :– A-Z, a-z
  Digits :– 0-9
  Special symbols :– Special symbol available over keyboard
  White spaces:– blank space, tab, carriage return, new line, form feed
  Other characters:- Unicode

Token

- Smallest individual unit in a program is known as token.
  1. Keywords
  2. Identifiers
  3. Literals
  4. Operators
  5. punctuators / Delimiters

Python Keywords

- Reserve word of the compiler/interpreter which can't be used as identifier.

and	exec	not
as	finally	or
assert	for	pass
break	from	print
class	global	raise
continue	if	return
def	import	try
del	in	while
elif	is	with
else	lambda	yield
except		

Identifiers

- A Python identifier is a name used to identify a variable, function, class, module or other object.
  * An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores
  and digits (0 to 9).
  * Python does not allow special characters
  * Identifier must not be a keyword of Python.
  * Python is a case sensitive programming language.
  Thus, **Rollnumber** and rollnumber are two different identifiers in Python.
  Some valid identifiers : Mybook, file123, z2td, date_2, _no
  Some invalid identifier : 2rno,break,my.book,data-cs

Some additional naming conventions

1. Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
2. Starting an identifier with a single leading underscore indicates that the identifier is private.
3. Starting an identifier with two leading underscores indicates a strong private identifier.
4. If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Literals

- Literals in Python can be defined as number, text, or other data that represent values to be stored in variables.
- Example of String Literals in Python
  name = ‘Johni’ , fname =“johny”
- Example of Integer Literals in Python(numeric literal)
  age = 22
- Example of Float Literals in Python(numeric literal)
  height = 6.2
- Example of Special Literals in Python
  name = None

Escape sequence

Escape Sequence	Description
\\	Backslash (\)
\'	Single quote (')
\"	Double quote (")
\a	ASCII Bell (BEL)
\b	ASCII Backspace (BS)
\f	ASCII Formfeed (FF)
\n	ASCII Linefeed (LF)
\r	ASCII Carriage Return (CR)
\t	ASCII Horizontal Tab (TAB)
\v	ASCII Vertical Tab (VT)
\ooo	Character with octal value ooo
\xhh	Character with hex value hh

Variables

- Variable in Python refers to an object which is an item or element that is stored in the memory. Value of a variable can be a string (e.g., ‘b’, ‘Global Citizen’), numeric (e.g., 345) or any combination of alphanumeric characters (CD67)

- Example: gender = 'M'
  message = "Keep Smiling"
  price = 987.9

Variable(contd.)

- **Variable Scope And Lifetime in Python Program**
  1. Local Variable

```
def fun():
x=8
print(x)
fun()
print(x) #error will be shown
```

  2. Global Variable

```
x = 8
def fun():
print(x) # Calling variable ‘x’ inside fun()
fun()
print(x) # Calling variable ‘x’ outside fun()
```

Practice time

Q) Write a Python program to find the sum of two numbers.

Q) Write a program in python to find the area of a rectangle given that its length is 10 units and breadth is 20 units.

Q) Write a program to swap two numbers using a third variable.

Operators

- Operators can be defined as symbols that are used to perform operations on operands.

- **Types of Operators**
  1. Arithmetic Operators.
  2. Relational Operators.
  3. Assignment Operators.
  4. Logical Operators.
  5. Membership Operators.
  6. Identity Operators.

Arithmetic Operators

Arithmetic Operators are used to perform arithmetic operations like addition, multiplication, division etc.

+	perform addition of two number	a+b
-	perform subtraction of two number	a-b
/	perform division of two number	a/b
*	perform multiplication of two number	a*b
%	Modulus = returns remainder	a%b
//	Floor Division = remove digits after the decimal point	a//b
**	Exponent = perform raise to power	a**b

Relational operators

- Relational operators are used to compare two values.

==	Equal to, return true if a equals to b	a == b
!=	Not equal, return true if a is not equals to b	a != b
>	Greater than, return true if a is greater than b	a > b
>=	Greater than or equal to , return true if a is greater than b or a is equals to b	a >= b
<	Less than, return true if a is less than b	a < b
<=	Less than or equal to , return true if a is less than b or a is equals to b	a <= b

Assignment Operators

- Used to assign values to the variables.

=	Assigns values from right side operands to left side operand	a=b
+=	Add 2 numbers and assigns the result to left operand.	a+=b
/=	Divides 2 numbers and assigns the result to left operand.	a/=b
*=	Multiply 2 numbers and assigns the result to left operand.	A*=b
-=	Subtracts 2 numbers and assigns the result to left operand.	A-=b
%=	modulus 2 numbers and assigns the result to left operand.	a%=b
//=	Perform floor division on 2 numbers and assigns the result to left operand.	a//=b
**=	calculate power on operators and assigns the result to left operand.	a**=b

Logical Operators

- Are used to perform logical operations on the given two variables or values.

and	return true if both condition are true	x and y
or	return true if either or both condition are true	x or y
not	reverse the condition	not(a>b)

- a=30
  b=20
  if(a==30 and b==20):
  print('hello')
  Output :-
  hello

Membership Operators

- Are used to validate whether a value is found within a sequence such as such as strings, lists, or tuples.

in	return true if value exists in the sequence, else false.	a in list
not in	return true if value does not exists in the sequence, else false.	a not in list

E.g.

```
a = 22
list = [22,99,27,31]
In_Ans = a in list
NotIn_Ans = a not in list
print(In_Ans)
print(NotIn_Ans)
```

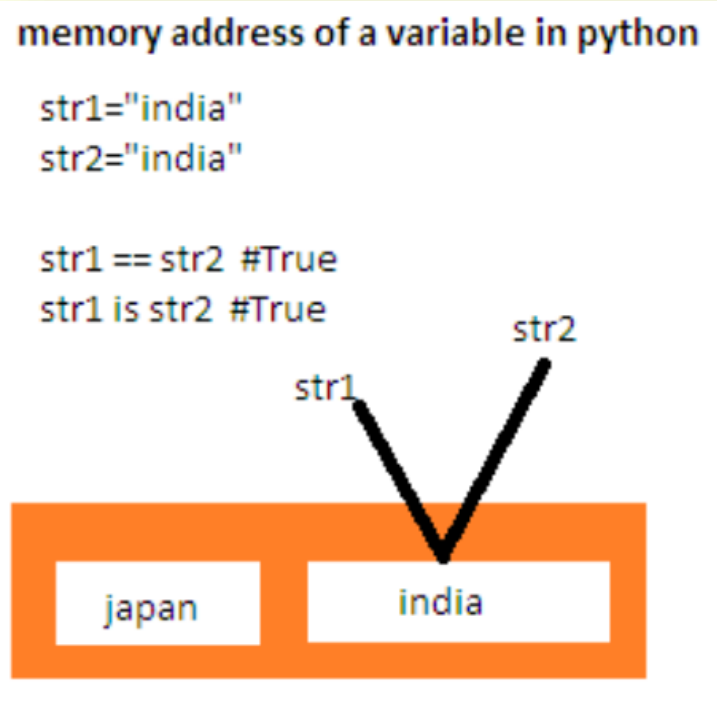
Output :-

```
True
False
```

Identity Operators

- Are used to compare the memory locations of two objects

is	returns true if two variables point the same object, else false	a is b
iSnot	returns true if two variables point the different object, else false	a is not b



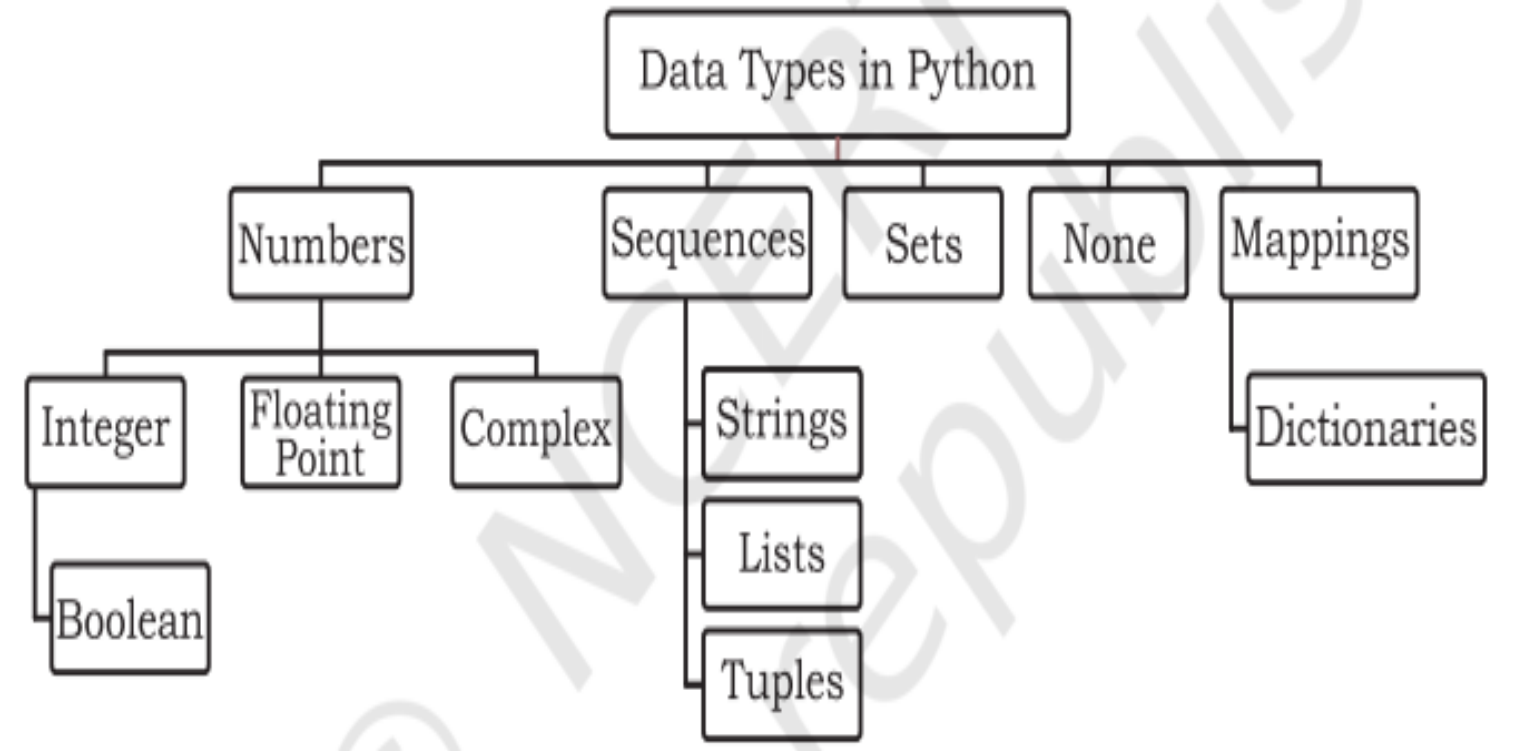
Punctuators / Delimiters

- They are used to implement the grammatical and structure of the syntax.

‘	“	#	\			
(	)	[	]	{	}	@
,	:	.	`	=	;	
+=	-=	*=	/=	//=	%=	
&=	|=	^=	>>=	<<=	**=	

Data types

- Data type identifies the type of data values a variable can hold and the operations that can be performed on that data.



Number

- Number data type stores numerical values only. It is further classified into three different types: int, float and complex. Boolean data type (bool) is a subtype of integer. It is a unique data type, consisting of two constants, True and False. Boolean True value is non-zero, non-null and non-empty. Boolean False is the value zero.

Type/ Class	Description	Examples
int	integer numbers	–12, –3, 0, 125, 2
float	real or floating point numbers	–2.04, 4.0, 14.23
complex	complex numbers	3 + 4i, 2 – 2i

- We can use the function type() to determine the data type of the variable.

Sequence

A Python sequence is an ordered collection of items, where each item is indexed by an integer.

- ***String:*** String is a group of characters. These characters may be alphabets, digits or special characters including spaces. String values are enclosed either in single quotation marks (e.g., 'Hello') or in double quotation marks (e.g., "Hello").

  For example,

```
>>> str1 = 'Hello Friend'
>>> str2 = "452"
```

List: List is a sequence of items separated by commas and the items are enclosed in square brackets [ ].

- To create a list

```
>>> list1 = [5, 3.4, "New Delhi", "20C", 45]
#print the elements of the list list1
>>> print(list1)
[5, 3.4, 'New Delhi', '20C', 45]
```

Sequence (contd.)

- ***Tuple***
  Tuple is a sequence of items separated by commas and
  items are enclosed in parenthesis ( ). This is unlike list, where values are enclosed in brackets [ ]. Once created, we cannot change the tuple.
- #create a tuple tuple1
  >>> tuple1 = (10, 20, "Apple", 3.4, 'a')
  #print the elements of the tuple tuple1
  >>> print(tuple1)
  (10, 20, "Apple", 3.4, 'a')

Set

- Set is an unordered collection of items separated by commas and the items are enclosed in curly brackets { }. A set is similar to list, except that it cannot have duplicate entries. Once created, elements of a set cannot be changed.

- Example: to create a set.

```
>>> set1 = {10,20,3.14,"New Delhi"}
>>> print(type(set1))
<class 'set'>
>>> print(set1)
{10, 20, 3.14, "New Delhi"}
```

- Example: to show that sets does not include duplicate items.

```
>>> set2 = {1,2,1,3}
>>> print(set2)
{1, 2, 3}
```

None

- None is a special data type with a single value. It is used to signify the absence of value in a situation. None supports no special operations, and it is neither False nor 0 (zero).
- Example:

```
>>> myVar = None
>>> print(type(myVar))
<class 'NoneType'>
>>> print(myVar)
None
```

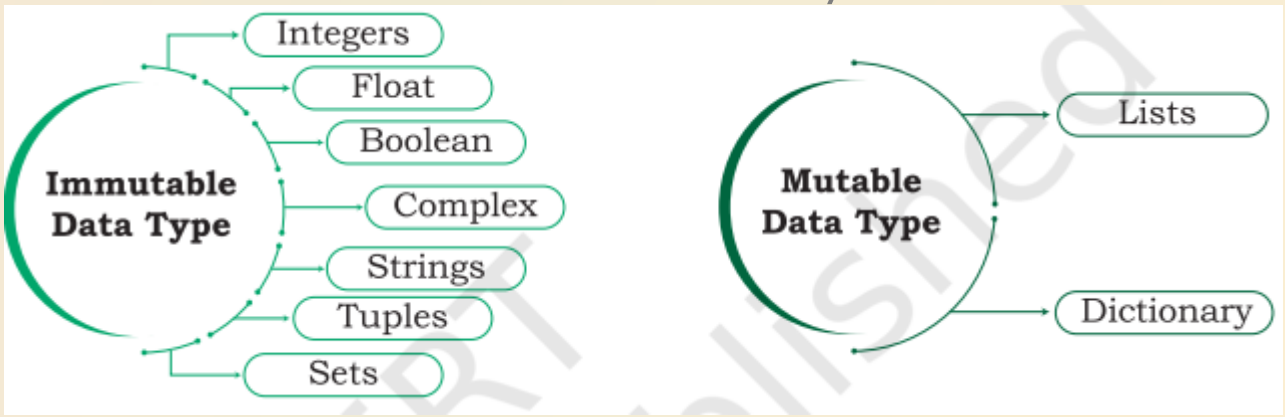
Mapping

- Mapping is an unordered data type in Python. Currently, there is only one standard mapping data type in Python called dictionary.
- ***Dictionary:*** holds data items in key-value pairs. Items in a dictionary are enclosed in curly brackets { }. Dictionaries permit faster access to data. Every key is separated from its value using a colon (:) sign. The keys are usually strings and their values can be any data type.
- Example: to create a dictionary

```
>>> dict1 = {'Fruit':'Apple', 'Climate':'Cold', 'Price(kg)':120}
>>> print(dict1)
{'Fruit': 'Apple', 'Climate': 'Cold', 'Price(kg)': 120}
>>> print(dict1['Price(kg)'])
120
```

Mutable and Immutable Data Types

- Variables whose values can be changed after they are created and assigned are called mutable.
- Variables whose values cannot be changed after they are created and assigned are called immutable. When an attempt is made to update the value of an immutable variable, the old variable is destroyed and a new variable is created by the same name in memory.



Deciding Usage of Python Data Types

Q1) When we need a simple iterable collection of data that may go for frequent modifications?

Q2) When we do not need any change in the data.
For example, names of months in a year.

Q3) If our data is being constantly modified or we need a fast lookup based on a custom key or we need a logical association between the key : value pair.

Comments

- Comments are used to add a remark or a note in the source code. Comments are not executed by interpreter. They are added with the purpose of making the source code easier for humans to understand.

Expressions

- An expression is defined as a combination of constants, variables, and operators. An expression always evaluates to a value.
- A value or a standalone variable is also considered as an expression **but** a standalone operator is not an expression.
- Example: are they valid. Give resons?

(i) 100 (ii) num (iii) num – 20.4	(iv) 3.0 + 3.14 (v) 23/3 -5 * 7(14 -2) (vi) "Global" + "Citizen"

Precedence of Operators

- Evaluation of the expression is based on precedence of operators. When an expression contains different kinds of operators, precedence determines which operator should be applied first.
- Higher precedence operator is evaluated before the lower precedence operator.

**	Exponentiation (raise to the power)
~ + -	Complement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'td>
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Precedence (contd.)

Q1) How will Python evaluate the following expression?

(20 + 30) * 40

Ans) = (20+30) * 40 #step1

= 50 * 40 #step2

= 2000 #step3

Q2) How will the following expression be evaluated in Python?

15.0 / 4 + (8 + 3.0)

Statement

- In Python, a statement is a unit of code that the Python interpreter can execute.

- >>> x = 4 #assignment statement
- >>> cube = x ** 3 #assignment statement
- >>> print (x, cube) #print statement

- 4 64

Input and Output

- In Python, we have the input() function for taking the user input. The input() function prompts the user to enter data. It accepts all user input as string. The user may enter a number or a string but the input() function treats them as strings only. The syntax for input() is:

  input ([Prompt])

  Prompt is the string we may like to display on the screen prior to taking the input, and it is optional.

The input() takes exactly what is typed from the keyboard, converts it into a string and assigns it to the variable on left-hand side of the assignment operator (=). Entering data for the input function is terminated by pressing the enter key.

Example:
```
>>> fname = input("Enter your frst name: ")
Enter your frst name: Arnab
>>> age = input("Enter your age: ")
 Enter your age: 19
>>> type(age)
<class 'str'>
```

Input and Output(contd.)

- Python uses the print() function to output data to standard output device — the screen. The function print() evaluates the expression before displaying it on the screen. The print() outputs a complete line and then moves to the next line for subsequent output.
- The syntax for print() is:
  print(value [, ..., sep = ' ', end = '\n'])
- sep: The optional parameter sep is a separator between the output values. We can use a character, integer or a string as a separator. The default separator is space.
- end: This is also optional and it allows us to specify any string to be appended after the last value. The default is a new line.
- Examples: print("hello")
  print(20* 1.5)
  print("I" + "Love" + "Python" + "Programming")

Type Conversion

- The process of converting the value of one data type (integer, string, float, etc.) to another data type is called type conversion. Python has two types of type conversion.
- Implicit Type Conversion
- Explicit Type Conversion

- **Implicit Type Conversion:**
  In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.
- Example: num_int = 12
  num_flo = 10.23
  num_new = num_int + num_flo
  print("datatype of num_int:",type(num_int))
  print("datatype of num_flo:",type(num_flo))
  print("Value of num_new:",num_new)
  print("datatype of num_new:",type(num_new))

Type Conversion (contd.)

- **Explicit Type Conversion:**
  In Explicit Type Conversion, users convert the data type of an object to required data type. We use the predefined functions like int(),float(),str() etc.
- Example:

```
        num_int = 12
       num_str = "45"
      print("Data type of num_int:",type(num_int))
print("Data type of num_str before Type Casting:“ ,type(num_str))

      num_str = int(num_str)
print("Data type of num_str after Type Casting:",type(num_str))

      num_sum = num_int + num_str
      print("Sum of num_int and num_str:",num_sum)
      print("Data type of the sum:",type(num_sum))
```

Debugging

- The process of identifying and removing such mistakes, also known as bugs or errors, from a program is called debugging. Errors occurring in programs can be categorized as:
  i) Syntax errors
  ii) Logical errors
  iii) Runtime errors

- **Syntax errors:** Python has its own rules that determine its syntax. The interpreter interprets the statements only if it is syntactically (as per the rules of Python) correct. If any syntax error is present, the interpreter shows error message(s) and stops the execution there.

- **Logical errors:** A logical error is a bug in the program that causes it to behave incorrectly. A logical error produces an undesired output but without abrupt termination of the execution of the program. They are also called semantic errors as they occur when the meaning of the program is not correct.

- **Runtime errors:** A runtime error causes abnormal termination of program while it is executing. Runtime error is when the statement is correct syntactically, but the interpreter cannot execute it. Runtime errors do not appear until after the program starts running or executing.

Thank you