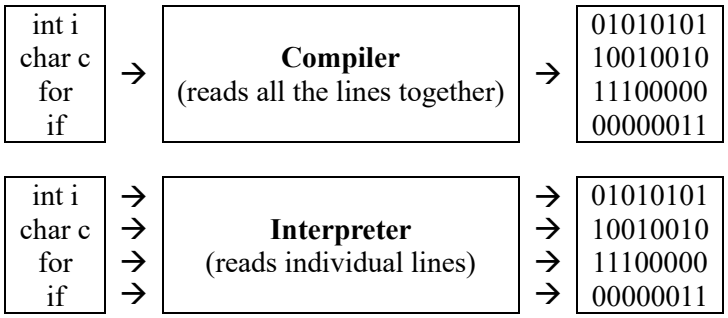
UNIT 1 – Introduction to Python and OOPs Concept

Differences between Programs and Script language

Program	Category	Scripting Language
A 'program' = Sequence of instructions written so that a computer can perform certain tasks	Definition	A 'script' = Code written in a scripting language → mostly used to control other applications
A program is **executed** (i.e., the source is first compiled, and the result of that compilation is expected).	Process	A script is **interpreted**.
More syntax – based		Less syntax – based
All lines → checked once and errors thrown		Interpreter → checks line by line errors
Faster	Speed	Slower



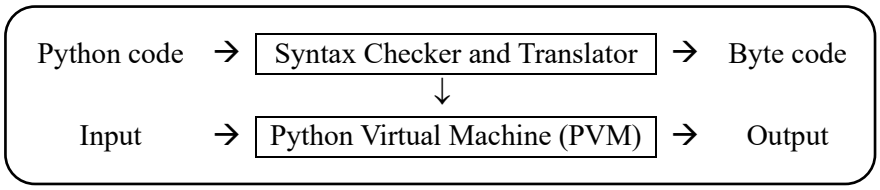
What is Python ?

- General purpose programming language that supports both → OOPs and procedural also
- Python = programming language + scripting language
- Also called → 'Interpreted language'

History

- Invented in Netherlands → early 90s by Guido van Rossum (fan of **Monty Python's flying circus** – a famous TV show in Netherlands)
- Python → conceived in the late 1980s
- Its implementation → started in December 1989

Steps



The python interpreter performs the following task to execute a python program –

Step 1

Interpreter → reads a python code / instruction → then it verifies that the instruction is well formatted i.e., it checks the syntax of each line → if it encounters any error, it immediately halts the translation and shows an error message.

↓

Step 2

If there is no error i.e., if the python code / instruction is well formatted → then the interpreter translates it into its equivalent form in intermediate language called 'byte code'

∴ After successful execution of python code / script → it is completely translated into byte code.

↓

Step 3

Byte code → sent to the Python Virtual Machine (PVM) → here again the bytecode is executed on PVM → if an error occurs during this execution, then the execution is halted with an error message.

Advantages of Python

1. Require considerably less number of lines of code to perform the same task compared to other languages like C
2. Less programming errors
3. Reduces the development time needed
4. Though Perl is a powerful language, it is highly syntax oriented and many modules are not completed

Scope of Python

1. Science (Bioinformatics)
2. System Administration
3. Web Application Development
4. CGI [Common Gateway Interface] → Testing scripts

Why do people use Python ? (Primary factors)

1. **Object oriented –** Structure supports such concepts as polymorphism, operations overloading and multiple inheritance

2. **Indentation –** one of the greatest futures in python

3. **Free (open source) –** Downloading and installing Python is free and easy. Source code is easily accessible.

4. **Powerful –** Dynamic Typing + Built – in types and tools + Library utilities + Third party utilities (Numeric, NumPy, SciPy) + Automatic memory management

5. **Portable –** Python runs virtually on every major platform used today.
   As long as you have a compatible python interpreter installed, python programs will run in exactly the same manner irrespective of platform.

6. **Mixable –** Python can be linked to components written in other languages easily.
   Linking to fast, compiled code is useful to computationally intensive problems.
   Python / C integration is quite common.

7. **Easy to use –** No intermediate compile and link steps as in C / C++.
   Python programs are compiled automatically to an intermediate form called 'byte code' which the interpreter then reads.
   This gives Python the development speed of an interpreter without the performance loss (inherent in purely interpreted languages).

8. **Easy to learn –** Structure and syntax are pretty easy to grasp.

Who uses Python today ?

Python is being applied in real revenue – generating products by real companies.

1. **Google** → makes extensive use of Python in its web search system and employs Python's creator.
2. **Intel, Cisco, Hewlett – Packard, Seagate, Qualcomm and IBM** → use Python for hardware testing
3. **ESRI (Environmental Systems Research Institute)** → uses Python as an end – user customization tool for its popular GIS mapping products
4. **YouTube's video sharing service** → is largely written in Python.

Installing Python

- Most Unix systems (including Linux, MAC OS) → Python = pre – installed
- Windows Operating System → user – download from https://www.python.org/downloads/ → download latest version of Python IDE and install

IDLE

- IDLE = Integrated Development Environment
- A graphical user interface – for doing Python development + Standard and free part of the Python system
- IDLE = like an editor that comes along with Python bundle → one can write the Python code or script using the IDLE

Python Basics

- Comment lines begin with → #
- File Naming scheme → filename.py → 'programs'
- Example program → >> print ("Hello World ! \n")

Python Data Types

1. Numeric 2. String 3. List 4. Tuple 5. Dictionary

I. Python Data Type – Numeric

Used to hold numeric values like –

Data Type	Description	Example
int	hold signed integers of non – limited length	a = 3
long	holds long integers exist in python 2.x ; deprecated in python 3.x	>>> print type(65536*65536) <type 'long'>
float	holds floating precision numbers accurate up to 15 decimal places	a = 5.6
complex	holds complex numbers	a = 2 + 4j

NOTE – type (a) → checks data type of 'a'

Example –	a = 100 → <class 'int'>
	a = 10.2345 → <class 'float'>
	a = 100 + 3j → <class 'complex'>

Taking Input in Python

- **input (prompt) →** this function first takes the input from the user and then evaluates the expression.
- Python automatically identifies whether user entered a string or a number or a list.

CODE	OUTPUT
val = input ("Enter your value: ") print (val)	Enter your value: 123 123

Basic Operators

Types	Operator	Symbol	Description	Syntax and Example
Arithmetic Operators	Addition	+	Adds two operands	x + y 7 + 2 = 9
	Subtraction	-	Subtracts two operands	x – y 7 – 2 = 5
	Multiplication	*	Multiplies two operands	x * y 7 * 2 = 14
	Division	/	Divides the first operand by the second	x / y 7 / 2 = 3.5
	Floor Division	//	Rounds off the answer to the nearest whole number	x // y 7 // 2 = 3
	Modulus	%	Returns the remainder when the first operand is divided by the second	x % y 7 % 2 = 1
	Exponent	**	Returns first raised to power second	x ** y 7 ** 2 = 49
Relational Operators (compares the values → returns either True or False according to the condition)	Less than	<	True if the left operand is less than the right	a < b
	Greater than	>	True if the left operand is greater than the right	a > b
	Less than or equal to	<=	True if left operand is less than or equal to the right	a <= b
	Greater than or equal to	>=	True if left operand is greater than or equal to the right	a >= b
	Equal to	==	True if both operands are equal	a == b
	Not equal to	!=	True if operands are not equal	a != b
Logical Operators	Logical AND	and	True → if both the operands are true	x and y True and True → True True and False → False False and True → False False and False → False

	Logical OR	or	True → if either of the operands is true	x or y True or True → True True or False → True False or True → True False or False → False
	Logical NOT	not	True → if operand is false	not x not True → False not False → True
Membership Operators (Tests for membership in a sequence such as strings, lists or tuples)	In	in	Evaluates to True if it finds a variable in the specified sequence and false otherwise	x in y ↓ 'in' results in → True → if x is a member of the sequence y ↓ else → False
	Not in	not in	Evaluates to True if it does not find a variable in the specified sequence and false otherwise	x not in y ↓ 'not in' results in → True → if x is not a member of the sequence y ↓ else → False

CODE	**OUTPUT**

a = 10
b = 20
list1 = [1, 2, 3, 4, 5]
if (a in list1):
print ("a is available")
else:
print ("a is not available")
if (b not in list1):
print ("b is not available)
else:
print ("b is available")
	a is not available
	b is not available

Assignment Operator	Assignment Operator	=	Used to assign values to the variable	x = 10 >> print (x) 10
	Addition Assignment Operator	+=	Add right side operand with left side operand and then assign the result to left operand	x += y
	Subtraction Assignment Operator	– =	Subtract right side operand from left side operand and then assign the result to left operand	x – = y

	Multiplication Assignment Operator	*=	Multiply right operand with left operand and then assign the result to the left operand	x *= y
	Division Assignment Operator	/=	Divide left operand with right operand and then assign the result to the left operand	x /= y
	Modulus Assignment Operator	%=	Divides the left operand with the right operand and then assign the remainder to the left operand	x //= y
	Floor Division Assignment Operator	//=	Divide left operand with right operand and then assign the value(floor) to left operand	x % = y
	Exponent Assignment Operator	**=	Calculate exponent (raise power) value using operands and then assign the result to left operand	x ** = y

II. Python Data Type – String

- String = sequence of characters → represented by either – single quotes / double quotes

Difference between single quoted and double quoted string	
CODE	**OUTPUT**
# single quotes print the statement as it is without interpreting print ('Hello World \n') # double quotes interpret and then print the statement print ("Hello World \n")	Hello World \n Hello World

- Python supports Unicode characters

String Special Operators

Operator	Symbol	Description	Example (a = "Hello" b = "Python")	Output
Concatenation	+	Adds / Merges values on either side of the operator	a + b	HelloPython
Repetition	*	Creates new strings, concatenating multiple copies of the same string	a*2	HelloHello
Slice	[ ]	Gives the character from the given index	a [1]	e
Range Slice	[:]	Gives the character from the given range	a [1:4]	ell
Membership	in	Returns True if a character exists in the given string	'H' in a	True
	not in	Returns True if a character does not exist in the given string	'M' not in a	True

CODE (str = "Hello World !")	OUTPUT
print str # Prints complete string	Hello World !
print str [0] # Prints 1st character of the string	H
print str [2:5] # Prints characters starting from 3rd to 5th character	llo
print str [2:] # Prints string starting from 3rd character	llo World !
print str*2 # Prints string 2 times	Hello World ! Hello World !
print str + "TEST" # Prints concatenated string	Hello World ! TEST

III. Python Data Type – List

- A versatile data type exclusive in python
- Same as the array in C or C++
- Can simultaneously hold different types of data
- List = ordered sequence of some data written using square brackets [ ] and commas (,)
- Example – userAge = [15, 25, 30, 45, 50, 65, 55]

Add and remove items

- To add item → append( ) function
  Example – userAge.append(100) → userAge = [15, 25, 30, 45, 50, 65, 55, 100]

- To remove any value from the list → del listName [index of the item to be deleted]
  Example – del userAge [2] → userAge = [15, 25, 45, 50, 65, 55, 100] # '30' was removed

CODE list1 = ['abcd', 786, 2.23, 'john', 70.2] tinylist = [123, 'john']	OUTPUT
print list1 # Prints complete list	['abcd', 786, 2.23, 'john', 70.2]
print list1 [0] # Prints 1st element of the list	'abcd'
print list1 [1:3] # Prints elements starting from 1st till 2nd index	786, 2.23
print list1 [2:] # Prints list starting from 3rd element	2.23, 'john', 70.2
print tinylist*2 # Prints list 2 times	[123, 'john', 123, 'john']
print list1 + tinylist # Prints concatenated lists	['abcd', 786, 2.23, 'john', 70.2, 123, 'john']

Updating lists

Can update single or multiple elements of list by giving the slice on the LHS of the assignment operator

CODE	OUTPUT
list1 = ['Physics', 'Chemistry', 1997, 2000] print ("Value available at index 2: ", list1 [2]) list [2] = 2001 print ("New value available at index 2: ", list [2])	Value available at index 2: 1997 New value available at index 2: 2001

Delete List Elements

- Can use either the 'del' statement → if you know exactly which element(s) you are deleting
- 'remove( )' method → if you do not know

CODE	OUTPUT
list1 = ['physics', 'chemistry', 1997, 2000] print (list1) del list1[2] print "After deleting value at index 2: " print (list1)	['physics', 'chemistry', 1997, 2000] After deleting value at index 2: ['physics', 'chemistry', 2000]

Built – in List functions and methods

Function / Method	Description
cmp (list1, list2)	Compares elements of both list
len (list)	Gives the total length of the list
max (list)	Returns item from the list with maximum value
min (list)	Returns item from the list with minimum value
list.append (obj)	Appends object obj to list returns
list.count (obj)	Count of how many times obj occurs in the list
list.extend (seq)	Appends the contents of sequence to list
list.index (obj)	Returns the lowest index in list that obj appears
list.insert (index, obj)	Inserts object obj into list at offset particular index
list.pop (obj = list [-1])	Removes and returns the last object or obj from list
list.remove (obj)	Removes object obj from list
list.reverse ( )	Reverses object of list in place
list.sort ([func])	Sorts objects of list use compare function if given

IV. Python Data Type – Tuples

- A sequence of data similar to list.
- Immutable → Data in a tuple is write protected.
- Data in a tuple → written using parenthesis and commas.

Basic Tuples Operations

- Tuples respond to the + (Concatenation) and * (Repetition) operators similar to strings → but the result is a tuple and not a string

Operator	Symbol	Description	Example	Output
Length		Prints the length of the tuple	len((1, 2, 3)	3
Concatenation	+	Adds / Merges values on either side of the operator	(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)
Repetition	*	Creates new strings, concatenating multiple copies of the same string	(‘Hi!’) * 4	(‘Hi!’, ‘Hi!’, ‘Hi!’, ‘Hi!’)
Membership	in	Returns True if a character exists in the given string	3 in (1, 2, 3)	True
	not in	Returns True if a character does not exist in the given string	4 in (1, 2, 3)	True
Iteration		Iterates through the tuple and prints each element	for x in (1, 2, 3): print x	1 2 3

CODE **tuple1 = (‘abcd’, 786, 2.23, ‘john’, 70.2)** **tinytuple = (123, ‘john’)**	**OUTPUT**
print tuple1 # Prints complete tuple	(‘abcd’, 786, 2.23, ‘john’, 70.2)
print tuple1 [0] # Prints 1st element of the tuple	‘abcd’
print tuple1 [1:3] # Prints elements starting from 1st till 2nd index	786, 2.23
print tuple1 [2:] # Prints tuple starting from 3rd element	2.23, ‘john’, 70.2
print tinytuple * 2 # Prints tuple 2 times	(123, ‘john’, 123, ‘john’)
print tuple1 + tinytuple # Prints concatenated tuples	(‘abcd’, 786, 2.23, ‘john’, 70.2, 123, ‘john’)

Built – in Tuple functions

Function / Method	Description
cmp (tuple1, tuple2)	Compares elements of both tuple
len (tuple)	Gives the total length of the tuple
max (tuple)	Returns item from the tuple with maximum value
min (tuple)	Returns item from the tuple with minimum value