UNIT I: COMPUTATIONAL THINKING AND PROGRAMMING

PYTHON REVISION TOUR

About Python:

- Python programming language was developed by Guido Van Rossum in February 1991.
- Python is based on two programming languages, ABC language and Modula-3
- It is a **case-sensitive** programming language.
- Python is one of the languages that are **not strict about data types**.
- Python works in to modes – **interactive and script**
- **Python is** Interpreted and Cross Platform Language, Free and Open Source, used for both procedural and object-oriented programming.

Basic terminologies:

- **TOKEN / LEXICAL UNIT:** Smallest individual unit in a Programming Language.
- **Types of Tokens** (Keywords, Identifiers, Literals, Operators, Punctuators)
- **Keywords:** Reserved words having special meaning.

Literals	True, False, None
Operators	not, and, or, is, in
Empty statement	Pass
Conditional Statements	if, elif, else
Iterative statements	for, while, break, continue
Exception Handling	try, except, finally, assert, raise
Functions	def, return, global
File Handling	with
Others	del, import, from, as, lambda, nonlocal, yield, class

- **Identifiers:** These are the names given to **variables, objects, classes** or **functions** etc.

  Identifier Naming Rules:

  ✓ Python identifiers **must begin** with a letter **(A-Z or a-z)** or an **underscore (_)**
  ✓ It can be followed by **letters, digits (0-9),** or **underscores**.
  ✓ However, Python keywords **cannot** be used as Identifiers
  ✓ Special characters like @, #, or $ **cannot be** while naming Identifiers.
  ✓ Length of identifier is endless and is case sensitive.

- **Literals / Constants:** Data items that have a fixed value are called Literals.
  Example: 5 (integer), 5.9 (float), True (Boolean), "Hello" (string)

- **Operators:** Symbols that trigger some action.

Arithmetic Operators	**, *, /, //, %
Relational Operators	<, <=, >, >=, ==, !=
Logical Operators	not, and, or
Identity Operator	is, is not
Membership Operator	in, not in
Assignment Operator	=, +=, -=, *=, /=, //=, %=

- **Operator Precedence:** Operator precedence defines the priority of operators in an expression.
- **Operator Associativity:** It define the direction in which operators of same precedence are evaluated either from left to right or right to left.

Precedence Level	Operators	Description	Associativity
1 (Highest)	( )	Parentheses (grouping)	N/A
2	x[index], x[attr], x(...), x[...]	Subscription, attribute reference, call	Left to Right
3	**	Exponentiation	Right to Left
4	+x, -x, ~x	Unary plus, minus, bitwise NOT	Right to Left
5	*, /, //, %	Multiplication, division, floor division, modulo	Left to Right
6	+, -	Addition, subtraction	Left to Right
7	<<, >>	Bitwise shift operators	Left to Right
8	&	Bitwise AND	Left to Right
9	^	Bitwise XOR	Left to Right
10	|	Bitwise OR	Left to Right
11	in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, identity, membership tests	Left to Right
12	Not	Boolean NOT	Right to Left
13	And	Boolean AND	Left to Right
14	Or	Boolean OR	Left to Right
15(Lowest)	=	Assignment operator	Right to Left

Example : Evaluate 100 + 200 / 10 - 3 * 10
Here, the operators /, * have similar precedence and +, - have same precedence, hence the expression will be evaluated as ((100 + (200 / 10)) – (3 * 10)), which equals to 90.0

- **Punctuators:** Symbols that are used to organize sentence structure. These are used to give syntactic and semantic meaning to the program statement.
  Some commonly used punctuators are:

‘	“	#	\	(	)	[	]	{	}	@	,	:	.	=	;

- **Variables:** Variables are containers for storing data values.
  Key points:
  - Variables do not need to be declared or defined in advance.
  - A variable is created when you first assign a value to it.
  - Python is dynamically typed language where data type of the variable is not required statically.
  - **Example:**

```
# Initially assigning an integer value
x = 88                                  # x assigned as integer type
print ("x =", x, " Type:", type(x))
# Reassigning a string value to the same variable
x = "Dynamic Typing in Python"  # x assigned as string type
print ("x =", x, " Type:", type(x))
```

Note: Dynamic typing allows user to assign different values to same variable at different places in a program

- **Conversion of Data Types in Python**

Type Promotion (Implicit Type Conversion)	**Type Casting (Explicit Type Conversion):**
Python interpreter automatically converts/ promotes one data type to another higher type without any user involvement	The data type is manually changed by the user as per their requirement. With explicit type conversion, there is a risk of data loss since we are forcing an expression to be changed in some specific data type.

Example:
x = 100
y = 10.6
z = x + y
print(z)
print ("z is of type:", type(z))
# z is implicitly converted to floating type 	
	Example:
	a, b=5.88, ‘4’
	c=int(a)
	print(“c=”,c) #c=5
	d=int(b)
	print(“d=”,d) #d=4

Python Modules:

- A **module** is a file containing Python code (functions, variables, classes) that can be imported and used in other Python programs.

- Modules help organize code and reuse functionality.

Random Number generation

random Module:

The random module in Python is a powerful tool for generating random numbers and performing random operations.

- To use the random module, **you must import it first**:

  import random

- It provides several functions to produce random values, simulate randomness, and shuffle data.

Name of method	**Method**	**Description**	**Example**
Random Float	random ( )	Generate a random float number between 0.0 and 1.0.	import random random. random ( )
Random Integer	randint ( )	Generate a random integer within a specified range (both values inclusive)	import random random. randint (1, 10)
Random Range	randrange( )	Generate a random number within a specified range with a step.	import random random. randrange (0, 100, 5) # Random number between 0 and 100, step 5

STRING

- A string is a sequence of characters enclosed in **single (' ')**, **double (" ")**, or **triple (''' ''' / """ """)** quotes.
- Strings are **immutable** — once created, they **cannot be changed**.
- Strings are **indexed**, starting from 0 (left to right) and -1 (right to left).
- Use triple quotes (''' ''' or """ """) for **multiline text**.
- Use \ to include special characters (e.g., \n for newline, \t for tab).

LIST

- A list is a collection of items (elements) which are ordered.
- Lists are mutable, meaning their contents can be changed after creation.
- Lists can contain elements of different data types: integers, strings, floats, even other lists.
- Lists are indexed, with the first element at index 0.
- Defined using square brackets [].
- Example:
- my_list = [10, "Hello", 3.14, True]

TUPLE

- **Ordered –** Elements are stored in a specific order.
- **Immutable** – Once created, elements cannot be changed.
- **Allow duplicates** – Tuples can contain repeated values.
- **Can hold different data types** – Integers, strings, lists, etc.
- **Indexing** – Elements can be accessed using indexes.
- **Faster than lists** – Due to immutability.
- Tuples written in **parenthesis ( ).**

Multiple element tuple	Single element tuple	
t=(10,20,30) tup = 10, 20, 30	t = (10,) **Note:** t=10	# Comma is important #t is integer

- **Packing Unpacking of Tuple**

```
tup = 10, 20, 30
a, b, c = tup
print(a,b,c)  # 10 20 30
```

DICTIONARY

- A **dictionary** in Python is an **unordered collection** of items.
- Each item is a **key-value pair**.
- Dictionaries are **mutable** (can be changed).
- Defined using **curly braces** {} with key:value pairs.
- **Key**: Must be **unique** and **immutable** (string, number, tuple)
- **Value**: Can be **any datatype** and can be **duplicate**
- Keys are used to access values.

Data type	**Operations**				
	Traversal	**Concatenation (+)**	**Replication (*)**	**Membership (in, not in)**	**Slicing (start:stop:step)**

String 'Hello'	**Traversing using a for loop**
	for char in s:
	print(char)
	
	Traversing using indexing
	for i in range(len(s)):
	print(s[i])
		"Hello" + " World!"
		
		"Hello World!"
			"Hello" *2
			
			"HelloHello"
				"W" in "World!"
				
				True
				
				"P" not in "Hello World!"
				
				False
					(a)
					s[start:end]
					"Hello"[1:4]
					"ell"
					
					(b)
					s [: end]
					"Hello"[:3]
					"Hel"
					
					(c)
					s[start:]
					"Hello"[2:]
					"Hel"
					
					(d)
					s[start:end:step]
					"Hello"[::2]
					"Hlo"

List [1,2,3]	lst=[10,20,30,40]
	for i in lst:
	print(i)
	
	for i in range(len(lst)):
	print(lst[i])
		[1,2] + [3,4]
		
		[1,2,3,4]
			[0] * 3
			
			[0,0,0]
				3 in [1, 2, 3]
				
				True
					L=[11,12,14,17]
					L[1:3]
					[12,14]
					
					L[:3]
					[11,12,14]

Tuple (1,2,3)	tpl=(10,20,30,40)
	for i in tpl:
	print(i)
	
	for i in range(len(tpl)):
	print(tpl[i])
		(1,2) + (3,4)
		
		(1,2,3,4)
			(1) * 3
			
			(1,1,1)
				3 in (1, 2, 3)
				
				True
					L=(11,12,14,17)
					L(1:3)
					(12,14)
					
					L(:3)
					(11,12,14)

String Functions and Methods

Method	Explanation	Example	Output
len( )	Returns number of characters in a string	len("Hello")	5
capitalize ( )	Capitalizes first letter, rest lowercase	"hello".capitalize( )	"Hello"
title ( )	Capitalizes the first letter of each word	"hello world".title( )	"Hello World"
lower ( )	Converts all characters to lowercase	"HELLO".lower( )	"hello"
upper ( )	Converts all characters to uppercase	"hello".upper( )	"HELLO"
count ( )	Counts occurrences of a substring	"banana".count("a")	3
find ( )	Finds first index of substring (-1 if not found)	"hello".find("e")	1
index ( )	Like find ( ) but error if not found	"hello".index("o")	4
endswith( )	Checks if string ends with specified value	"hello". endswith("lo")	True
startswith( )	Checks if string starts with specified value	"hello". startswith("he")	True
isalnum( )	Checks if all characters are alphanumeric	"abc123". isalnum( )	True
isalpha( )	Checks if all characters are alphabetic	"abc". isalpha( )	True
isdigit( )	Checks if all characters are digits	"123". isdigit( )	True

islower( )	Checks if all characters are lowercase	"hello". islower( )	True
isupper( )	Checks if all characters are uppercase	"HELLO". isupper( )	True
isspace( )	Checks if string contains only whitespace	" ". isspace( )	True
lstrip( )	Removes leading spaces or characters	" hello". lstrip( )	"hello"
rstrip( )	Removes trailing spaces or characters	"hello ". rstrip( )	"hello"
strip( )	Removes both leading and trailing spaces	" hello ". strip ( )	"hello"
replace( )	Replaces all occurrences of a substring	"hello".replace("l", "x")	"hexxo"
join ( )	Joins elements of a sequence with string as separator	"*". join (["a", "b", "c"])	'a*b*c'
partition ( )	Splits string into 3 parts: before, separator, after	"hello world".partition(" ")	('hello', ' ','world')
split ( )	Splits string into list at separator	"a,b,c".split(",")	['a', 'b', 'c']

List Functions and Methods

Function / Method	Explanation	Syntax	Example	Output
len( )	Returns the number of elements in the list	len(list)	len([1, 2, 3])	3
list ( )	Converts an iterable (like string, tuple) to a list	list(iterable)	list("abc")	['a', 'b', 'c']
append ( )	Adds a single element at the end of the list	list.append(item)	a = [1, 2] a.append(3)	[1, 2, 3]
extend ( )	Adds elements of another list to the end	list.extend(iterable)	a = [1] a.extend([2, 3])	[1, 2, 3]
insert ( )	Inserts an element at a given position	list.insert(index, item)	a = [1, 3] a.insert(1, 2)	[1, 2, 3]
count ( )	Returns number of times an element appears	list.count(item)	[1, 2, 2, 3]. count (2)	2
index ( )	Returns index of first occurrence of element	list.index(item)	a=[10, 20, 30] a.index(20)	1
remove ( )	Removes first occurrence of a specified element	list.remove(item)	a = [1, 2, 3, 2]; a.remove(2)	[1,3,2]
pop ( )	Removes and returns element at given index (default: last)	list.pop([index])	a = [1, 2, 3]; a.pop( )	3 list becomes [1, 2]
reverse ( )	Reverses the list in place	list.reverse( )	a = [1, 2, 3]; a.reverse( )	[3, 2, 1]
sort ( )	Sorts the list in ascending order (modifies original list)	list.sort( )	a = [3, 1, 2]; a.sort( )	[1, 2, 3]
sorted ( )	Returns a new sorted list (original list unchanged)	sorted(list)	sorted ([3, 1, 2])	[1, 2, 3]
min ( )	Returns the smallest item in the list	min(list)	min ([2, 3, 1])	1
max ( )	Returns the largest item in the list	max(list)	max ((2, 3, 1])	3
sum ( )	Returns the sum of all numeric items in the list	sum(list)	sum ((1, 2, 3])	6

Tuple Functions and Methods

Function	Description	Syntax	Example	Output
len( )	Returns the number of elements in the tuple	len(t)	len((10, 20, 30))	3
max ( )	Returns the maximum element	max(t)	max((5, 12, 8))	12
min ( )	Returns the minimum element	min(t)	min((5, 12, 8))	5
sum ( )	Returns the sum of all elements	sum(t)	sum((10, 20, 30))	60
sorted ( )	Returns a sorted list (not tuple) of elements	sorted(t)	sorted((50, 10,20))	[10, 20,50]
tuple( )	Converts a sequence into a tuple	tuple(seq)	tuple([1, 2, 3])	(1, 2, 3)
count(x)	Returns the **number of times** the value x appears in the tuple	tuple.count(x)	(1, 2, 2, 3).count(2)	2
index(x)	Returns the **index of the first occurrence** of the value x in the tuple	tuple.index(x)	(10, 20,30).index(20)	1

WORKING WITH DICTIONARIES

d = {'name': 'John', 'age': 16, 'grade': 'A'}

Operation	Code Example	Description	Output
Access a value	d['name']	Returns the value of the key	'John'
Add a new pair	D['city'] = 'Delhi'	Adds new key-value to the dictionary	{'city': 'Delhi'}
Modify a value	d['age'] = 17	Changes the value of existing key	{'age': 17}
Delete a pair	del d['grade']	Deletes key-value pair	Removes 'grade' key
Check key existence	'name' in d	Returns True if key exists	True
Loop through keys	for k in d:	Iterates over keys	'name', 'age', etc.
Loop through items	for k, v in d.items( ): print(k, v)	Iterates through key- value pairs	name John age 16 city Delhi

Dictionary Functions & Methods

Function/ Method	Example	Output
len( )	len({'a':1, 'b':2})	2
dict( )	dict(a=1, b=2)	{'a': 1, 'b': 2}
keys ( )	{'x':1, 'y':2}.keys( )	dict_keys(['x', 'y'])
values ( )	{'x':1, 'y':2}.values( )	dict_values([1, 2])
items ( )	{'x':1, 'y':2}.items( )	dict_items([('x', 1), ('y', 2)])
get ( )	{'name':'John'}.get('name')	'John'
update ( )	{'a':1}.update({'b':2})	{'a':1, 'b':2}
Del	d={'a':1}; del d['a']; print(d)	{}
clear ( )	d={'a':1}; d.clear( ); print(d)	{}
fromkeys( )	dict.fromkeys(['a', 'b'], 0) ** Creates a new dictionary from a sequence of keys with a common value.*	{'a': 0, 'b': 0}
copy ( )	d={'a':1}; d2=d.copy( ); print(d2)	{'a': 1}
pop ( )	d={'a':1,'b':2}; print(d.pop('a')); print(d) ** Removes the item with the specified key and returns its value.*	1 {'b':2}
popitem( )	{'a':1,'b':2}.popitem( )	('b', 2)
setdefault( )	d={'a':1}; d.setdefault('b',2); print(d) ** Returns value of key. If key not found, inserts it with a specified default value.*	{'a':1,'b':2}
max ( )	max({'a':1, 'b':2})	'b'
min ( )	min({'a':1, 'b':2})	'a'
sorted ( )	sorted({'c':3, 'a':1, 'b':2})	['a', 'b', 'c']