STRINGS • LISTS • TUPLES • DICTIONARY • MODULES

Class XII Computer Science (083) | every function of the syllabus with a worked example and its exact output

PART 1 — STRINGS

A string is a sequence of characters written inside single, double or triple quotes. Strings are IMMUTABLE — once made, a character cannot be changed. S[0] = 'X' gives an error; you must build a new string instead.

1.1 String operations

Operation	Symbol	Example	Output
Concatenation (joining)	+	"Hello" + "World"	HelloWorld
Repetition	*	"Ha" * 3	HaHaHa
Membership	in	"am" in "I am fine"	True
Membership (negative)	not in	"xy" not in "Python"	True
Comparison	== < >	"apple" < "banana"	True

- Both sides of + must be strings. "Class" + 12 is an error; write "Class" + str(12).

1.2 Indexing and slicing S = "PYTHON"

```
Index  :   0    1    2    3    4    5
String :   P    Y    T    H    O    N
Index  :  -6   -5   -4   -3   -2   -1
```

Slice	Meaning	Output
S[0]	first character	P
S[-1]	last character	N
S[1:4]	from 1 up to 3 (4 is left out)	YTH
S[:3]	from the start up to 2	PYT
S[3:]	from 3 to the end	HON
S[:]	the whole string	PYTHON
S[0:6:2]	from 0 to 5, step 2	PTO
S[::-1]	the string reversed	NOHTYP
S[-4:-1]	from -4 up to -2	THO

Rule for every slice: the START is included, the STOP is left out. This one line answers most output questions.

1.3 Traversing a string

```
S = "Python"

# method 1 – directly, character by character
for ch in S:
    print(ch, end=" ")            # P y t h o n

# method 2 – using the index, when the position is needed
for i in range(len(S)):
    print(i, S[i])

# method 3 – backwards
for i in range(len(S)-1, -1, -1):
    print(S[i], end="")           # nohtyP
```

1.4 String functions and methods

Function / Method	What it does	Example	Output
len(S)	number of characters	len("Python")	6
capitalize()	first letter capital, everything else small	"pYTHON".capitalize()	Python
title()	first letter of EVERY word capital	"hello world".title()	Hello World
lower()	all letters small	"PYTHON".lower()	python
upper()	all letters capital	"python".upper()	PYTHON
count(sub)	how many times sub occurs	"banana".count("a")	3

Function / Method	What it does	Example	Output
find(sub)	position of the first sub; −1 if not found	"Computer".find("put")	3
find(sub)	not present — no error	"Computer".find("xyz")	-1
index(sub)	same as find, but ERROR if not found	"Computer".index("put")	3
endswith(sub)	does the string end with sub	"file.txt".endswith(".txt")	True
startswith(sub)	does the string start with sub	"Python".startswith("Py")	True
isalnum()	only letters and digits, nothing else	"Python3".isalnum()	True
isalnum()	a space makes it False	"Py 3".isalnum()	False
isalpha()	only letters	"Python".isalpha()	True
isdigit()	only digits	"12345".isdigit()	True
islower()	all letters are small	"python".islower()	True
isupper()	all letters are capital	"PYTHON".isupper()	True
isspace()	only spaces	" ".isspace()	True
lstrip()	removes spaces from the LEFT	" hi ".lstrip()	'hi '
rstrip()	removes spaces from the RIGHT	" hi ".rstrip()	' hi'
strip()	removes spaces from BOTH ends	" hi ".strip()	'hi'
replace(a,b)	replaces every a with b	"I am a boy".replace("a","A")	I Am A boy
join(list)	joins the items using the string as glue	"-".join(["a","b","c"])	a-b-c
partition(sub)	splits into 3 parts at the FIRST sub	"Comp Sci".partition(" ")	('Comp', ' ', 'Sci')
split(sub)	breaks into a LIST at every sub	"a,b,c".split(",")	['a', 'b', 'c']
split()	with nothing inside, breaks at spaces	"I am fine".split()	['I', 'am', 'fine']

- Remember the difference: find() returns −1 when the text is missing, index() stops the program with an error.
- No string method ever changes the original string. They all RETURN a new string, which is why we write S = S.upper().
- partition() returns a tuple of 3 items, split() returns a list.

PART 2 — LISTS

A list is an ordered collection written inside square brackets. It can hold items of different types, and it is MUTABLE — items can be changed, added and removed.

2.1 List operations L = [10, 20, 30, 40, 50]

Operation	Example	Output
Concatenation	[1,2] + [3,4]	[1, 2, 3, 4]
Repetition	[0] * 4	[0, 0, 0, 0]
Membership	30 in L	True
Indexing	L[0] and L[-1]	10 and 50
Slicing	L[1:4]	[20, 30, 40]
Slicing with step	L[::2]	[10, 30, 50]
Reversed	L[::-1]	[50, 40, 30, 20, 10]
Changing an item	L[0] = 99	[99, 20, 30, 40, 50]
Changing a slice	L[0:2] = [1, 2]	[1, 2, 30, 40, 50]

A string cannot be changed, a list can. That is the single biggest difference between them.

2.2 Traversing a list

```
L = [10, 20, 30]

for item in L:            # by value
    print(item)

for i in range(len(L)):   # by index, when the position is needed
    print(i, L[i])
```

2.3 List functions and methods

Function / Method	What it does	Example	Output
`len(L)`	number of items	`len([10,20,30])`	`3`
`list(seq)`	makes a list out of a string or tuple	`list("abc")`	`['a', 'b', 'c']`
`append(x)`	adds ONE item at the end	`L=[1,2]; L.append(3)`	`[1, 2, 3]`
`append(list)`	a whole list is added as ONE item	`L=[1,2]; L.append([3,4])`	`[1, 2, [3, 4]]`
`extend(L2)`	adds every item of L2 separately	`L=[1,2]; L.extend([3,4])`	`[1, 2, 3, 4]`
`insert(i,x)`	puts x at position i	`L=[1,2,3]; L.insert(1,99)`	`[1, 99, 2, 3]`
`count(x)`	how many times x occurs	`[1,2,2,3].count(2)`	`2`
`index(x)`	position of the FIRST x	`[10,20,30].index(20)`	`1`
`remove(x)`	removes the first x by VALUE	`L=[1,2,3,2]; L.remove(2)`	`[1, 3, 2]`
`pop()`	removes and RETURNS the last item	`L=[1,2,3]; L.pop()`	`3, list becomes [1, 2]`
`pop(i)`	removes and returns the item at i	`L=[1,2,3]; L.pop(0)`	`1, list becomes [2, 3]`
`reverse()`	reverses the list itself	`L=[1,2,3]; L.reverse()`	`[3, 2, 1]`
`sort()`	sorts the list itself, smallest first	`L=[3,1,2]; L.sort()`	`[1, 2, 3]`
`sort(reverse=True)`	sorts largest first	`L=[3,1,2]; L.sort(reverse=True)`	`[3, 2, 1]`
`sorted(L)`	returns a NEW sorted list, L is unchanged	`sorted([3,1,2])`	`[1, 2, 3]`
`min(L) / max(L)`	smallest / largest item	`min([4,9,2]) , max([4,9,2])`	`2 , 9`
`sum(L)`	total of all the numbers	`sum([10,20,30])`	`60`

- remove() needs the VALUE, pop() needs the POSITION. Mixing these two is the most common mistake.
- sort() and reverse() change the list and return None. print(L.sort()) prints None — write L.sort() first, then print(L).
- sorted() does not disturb the original list, which is why it is used when both versions are needed.
- min(), max() and sum() work only when all the items can be compared. sum() needs numbers only.

2.4 Nested lists

```
L = [[1, 2], [3, 4], [5, 6]]      # a list inside a list
print(len(L))          # 3   – three items, not six
print(L[1])            # [3, 4]
print(L[1][0])         # 3   – row 1, column 0

for row in L:                     # traversing a nested list
    for item in row:
        print(item, end=" ")      # 1 2 3 4 5 6
```

PART 3 — TUPLES

A tuple is an ordered collection written inside round brackets. It works exactly like a list except for one thing — a tuple is IMMUTABLE. No item can be changed, added or removed after it is created.

```
T = (10, 20, 30)
T = 10, 20, 30            # brackets are optional
T = (10,)                 # ONE item tuple – the comma is compulsory
T = (10)                  # this is NOT a tuple, it is the integer 10
T = ()                    # empty tuple

T[0] = 99                 # ERROR : tuples do not support item assignment
```

3.1 Tuple operations T = (10, 20, 30, 40, 50)

Operation	Example	Output
Concatenation	`(1,2) + (3,4)`	`(1, 2, 3, 4)`
Repetition	`(0,) * 3`	`(0, 0, 0)`
Membership	`30 in T`	`True`
Indexing	`T[0]  and  T[-1]`	`10  and  50`
Slicing	`T[1:4]`	`(20, 30, 40)`
Reversed	`T[::-1]`	`(50, 40, 30, 20, 10)`

3.2 Tuple functions

Function / Method	What it does	Example	Output
`len(T)`	number of items	`len((10,20,30))`	`3`
`tuple(seq)`	makes a tuple from a string or list	`tuple("abc")`	`('a', 'b', 'c')`
`tuple(seq)`	from a list	`tuple([1,2,3])`	`(1, 2, 3)`
`count(x)`	how many times x occurs	`(1,2,2,3).count(2)`	`2`
`index(x)`	position of the first x	`(10,20,30).index(30)`	`2`
`sorted(T)`	returns a sorted LIST, not a tuple	`sorted((3,1,2))`	`[1, 2, 3]`
`min(T) / max(T)`	smallest / largest item	`min((4,9,2)) , max((4,9,2))`	`2 , 9`
`sum(T)`	total of the numbers	`sum((10,20,30))`	`60`

- A tuple has only TWO methods of its own — count() and index(). Everything else is a built-in function used on it.
- sorted() on a tuple gives back a LIST. To get a tuple write tuple(sorted(T)).

3.3 Tuple assignment (unpacking)

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

x, y = 5, 8               # packing two values
x, y = y, x               # swapping in ONE line, no temporary variable
print(x, y)               # 8 5

# the number of variables must match the number of items
a, b = (1, 2, 3)          # ERROR : too many values to unpack
```

3.4 Nested tuples

```
T = ((1, "Aarav", 85), (2, "Siya", 92))
print(len(T))             # 2
print(T[1])               # (2, "Siya", 92)
print(T[1][1])            # Siya

for rec in T:
    print(rec[1], rec[2]) # Aarav 85   then   Siya 92
```

3.5 List against tuple

Point	List	Tuple
Brackets	[ ] square	( ) round
Can be changed	yes (mutable)	no (immutable)
Methods available	many — append, remove, sort ...	only count() and index()
Speed	slower	faster
Used when	the data has to change	the data must stay safe and fixed

PART 4 — DICTIONARY

A dictionary stores data in KEY : VALUE pairs inside curly brackets. The value is reached through its key, not through a position number. Keys must be unique and immutable (a string, number or tuple); values can be anything.

```
D = {"name": "Aarav", "age": 17, "marks": 85}

print(D["name"])          # Aarav      — access by key
print(D["class"])         # ERROR : KeyError, that key does not exist

D["age"] = 18             # existing key  -> the value is MODIFIED
D["city"] = "Delhi"       # new key       -> a new pair is ADDED

for k in D:                           # traversing : k is the KEY
    print(k, ":", D[k])

for k, v in D.items():                # traversing key and value together
    print(k, ":", v)
```

4.1 Dictionary functions and methods D = {'a': 1, 'b': 2}

Function / Method	What it does	Example	Output
`len(D)`	number of key–value pairs	`len({'a':1,'b':2})`	`2`
`dict(seq)`	builds a dictionary	`dict([('a',1),('b',2)])`	`{'a': 1, 'b': 2}`
`keys()`	all the keys	`D.keys()`	`dict_keys(['a', 'b'])`
`values()`	all the values	`D.values()`	`dict_values([1, 2])`
`items()`	every pair as a tuple	`D.items()`	`dict_items([('a', 1), ('b', 2)])`
`get(k)`	value of k; None if absent, NO error	`D.get('z')`	None
`get(k, default)`	value of k, or the default given	`D.get('z', 0)`	`0`
`update(D2)`	adds D2's pairs; common keys are overwritten	`D.update({'c':3})`	`{'a': 1, 'b': 2, 'c': 3}`
`del D[k]`	removes that one pair	`del D['a']`	`{'b': 2}`
`clear()`	empties the dictionary, D still exists	`D.clear()`	`{}`
`fromkeys(seq, v)`	makes a dictionary with the same value for every key	`dict.fromkeys(['a','b'], 0)`	`{'a': 0, 'b': 0}`
`copy()`	a separate copy of the dictionary	`D2 = D.copy()`	`{'a': 1, 'b': 2}`
`pop(k)`	removes k and RETURNS its value	`D.pop('a')`	`1, D becomes {'b': 2}`
`popitem()`	removes and returns the LAST pair	`D.popitem()`	`('b', 2)`
`setdefault(k, v)`	key absent : adds it and returns v	`D.setdefault('c', 3)`	`3, D gets 'c': 3`
`setdefault(k, v)`	key present : returns the old value, changes nothing	`D.setdefault('a', 9)`	`1`
`max(D) / min(D)`	largest / smallest KEY	`max({'a':5,'b':2})`	`b`
`max(D.values())`	largest VALUE	`max({'a':5,'b':2}.values())`	`5`
`sorted(D)`	list of the keys in order	`sorted({'b':2,'a':1})`	`['a', 'b']`

- D['z'] stops the program with a KeyError; D.get('z') quietly returns None. Use get() whenever the key may be missing.
- max(D) works on the KEYS. To work on the values write max(D.values()).
- clear() empties the dictionary, del D deletes the whole variable.
- A key cannot repeat. Writing the same key twice simply overwrites the earlier value.

PART 5 — PYTHON MODULES

A module is a ready-made file of functions. Importing it saves you from writing those functions yourself.

5.1 Two ways to import

```
# 1. import the whole module — the module name must be written every time
import math
print(math.sqrt(25))          # 5.0
print(math.pi)                # 3.141592653589793

# 2. from ... import — the module name is NOT written afterwards
from math import sqrt, pi
print(sqrt(25))               # 5.0
print(pi)                     # 3.141592653589793

from math import *            # imports everything
import math as m              # a short nickname
print(m.sqrt(25))             # 5.0
```

- With 'import math' you MUST write math.sqrt(25). Writing sqrt(25) alone gives a NameError.
- With 'from math import sqrt' you MUST write sqrt(25). Writing math.sqrt(25) then gives an error.

5.2 math module

Function	What it does	Example	Output
`math.pi`	the value of π	`math.pi`	`3.141592653589793`
`math.e`	the value of e	`math.e`	`2.718281828459045`
`sqrt(x)`	square root — the answer is always a float	`math.sqrt(25)`	`5.0`
`ceil(x)`	rounds UP to the next whole number	`math.ceil(4.2)`	`5`
`ceil(x)`	with a negative number	`math.ceil(-4.2)`	`-4`

Function	What it does	Example	Output
floor(x)	rounds DOWN to the whole number below	math.floor(4.8)	4
floor(x)	with a negative number	math.floor(-4.2)	-5
pow(x,y)	x raised to y — always a float	math.pow(2,3)	8.0
fabs(x)	absolute value (removes the minus sign)	math.fabs(-7)	7.0
sin(x)	sine — x must be in RADIANS	math.sin(0)	0.0
cos(x)	cosine	math.cos(0)	1.0
tan(x)	tangent	math.tan(0)	0.0
sin(x)	using pi for 90 degrees	math.sin(math.pi/2)	1.0

- ceil goes UP, floor goes DOWN. With negative numbers ceil(-4.2) is -4 because -4 is higher than -4.2.
- math.sqrt(25) gives 5.0 with a decimal point, not 5. Same for pow() and fabs().
- The trigonometric functions take radians. For degrees use math.sin(math.radians(90)).

5.3 random module

Function	What it does	Example	Output
random()	a decimal number from 0.0 up to but NOT including 1.0	random.random()	0.7325... (changes every run)
randint(a,b)	a whole number from a to b, BOTH included	random.randint(1,6)	any one of 1 2 3 4 5 6
randrange(a,b)	a whole number from a to b−1, b is LEFT OUT	random.randrange(1,6)	any one of 1 2 3 4 5
randrange(b)	from 0 to b−1	random.randrange(4)	any one of 0 1 2 3
randrange(a,b,s)	from a to b−1 in steps of s	random.randrange(0,10,2)	any one of 0 2 4 6 8

The one line that is tested every year: randint INCLUDES the last number, randrange LEAVES IT OUT.

```
import random

print(random.randint(2, 5))          # possible : 2, 3, 4, 5
print(random.randrange(2, 5))        # possible : 2, 3, 4
print(int(random.random() * 10))     # possible : 0 to 9
print(random.randint(1, 3) + 10)     # possible : 11, 12, 13

# picking a random item from a list
L = ["RED", "GREEN", "BLUE"]
print(L[random.randint(0, 2)])       # any one colour
```

5.4 statistics module

Function	What it does	Example	Output
mean(L)	the average — total divided by how many	statistics.mean([1,2,3,4])	2.5
median(L)	the middle value after sorting	statistics.median([1,3,2])	2
median(L)	even count : average of the middle two	statistics.median([1,2,3,4])	2.5
mode(L)	the value that occurs most often	statistics.mode([1,2,2,3])	2
mode(L)	works on text too	statistics.mode(["a","b","a"])	a

- median() sorts the data itself — you do not have to sort the list first.
- If two values occur equally often, older versions of Python raise an error for mode(); newer ones return the first of them.

PART 6 — The programs named in the syllabus

6.1 Maximum, minimum and mean of a list

```
L = [45, 12, 78, 30, 66]

# using the built-in functions
print("Maximum :", max(L))              # 78
print("Minimum :", min(L))              # 12
print("Mean    :", sum(L) / len(L))     # 46.2

# doing it without built-in functions (often asked this way)
big = small = L[0]
```

```
total = 0
for x in L:
    if x > big:   big = x
    if x < small: small = x
    total = total + x
print(big, small, total/len(L))          # 78 12 46.2
```

6.2 Linear search in a list or a tuple

```
L = [45, 12, 78, 30, 66]
key = int(input("Enter the number to search : "))

found = False
for i in range(len(L)):
    if L[i] == key:
        print("Found at position", i)
        found = True
        break
if found == False:
    print("Not found")

# the same code works on a tuple T = (45, 12, 78, 30, 66)
```

6.3 Frequency of every item of a list, using a dictionary

```
L = [1, 2, 2, 3, 1, 2]
D = {}

for x in L:
    if x in D:
        D[x] = D[x] + 1
    else:
        D[x] = 1
print(D)                    # {1: 2, 2: 3, 3: 1}

# shorter, with get()
D = {}
for x in L:
    D[x] = D.get(x, 0) + 1
print(D)                    # {1: 2, 2: 3, 3: 1}
```

6.4 Counting how many times each character appears in a string

```
S = "banana"
D = {}

for ch in S:
    D[ch] = D.get(ch, 0) + 1

print(D)                    # {'b': 1, 'a': 3, 'n': 2}

for k, v in D.items():
    print(k, "occurs", v, "time(s)")
```

6.5 Employees with their salary, kept in a dictionary

```
emp = {}
n = int(input("How many employees : "))

for i in range(n):
    name = input("Enter name   : ")
    sal  = float(input("Enter salary : "))
    emp[name] = sal

print(emp)

nm = input("Whose salary do you want : ")
if nm in emp:
    print(nm, "earns", emp[nm])
else:
    print("No such employee")

# highest paid employee
print("Highest salary :", max(emp.values()))
for k, v in emp.items():
    if v == max(emp.values()):
        print("Earned by :", k)
```

PART 7 — The four collections at a glance

Point	String	List	Tuple	Dictionary
Brackets	' ' or " "	[ ]	( )	{ }
Ordered	yes	yes	yes	yes (as entered)
Can be changed	no	yes	no	yes
Items reached by	index	index	index	key
Duplicates allowed	yes	yes	yes	values yes, keys no
Empty one	S = ""	L = []	T = ()	D = {}

Mistakes that cost marks

- In every slice the stop value is LEFT OUT : S[1:4] gives three characters, not four.
- Trying to change a string or a tuple : S[0] = 'X' and T[0] = 9 are both errors.
- print(L.sort()) prints None. sort(), reverse(), append() and insert() return nothing.
- append() adds a list as ONE item, extend() adds its items separately.
- remove() takes the value, pop() takes the position.
- D['key'] raises an error when the key is missing; D.get('key') returns None instead.
- randint(1,6) can give 6, randrange(1,6) can never give 6.
- math.sqrt() and math.pow() always give a float — write 5.0, not 5, in an output question.
- ceil(-4.2) is -4 and floor(-4.2) is -5. Negative numbers reverse what students expect.
- After 'from math import sqrt', writing math.sqrt() is an error, and the other way round as well.