Python Programming (DI02032011)

Winter 2025 Exam Solutions

Milav Dabgar

Lecturer, Dept of EC
GP Palanpur

Date of Exam: 20 January 2026

Exam Overview

Exam Details

Subject:	Python Programming
Code:	DI02032011
Date:	20-01-2026
Time:	10:30 AM – 01:00 PM
Marks:	70

Questions at a Glance

Q1	Algorithm, Flowchart, Data Types
Q2	Features, if-else, Patterns
Q3	Strings, Loops, Variables
Q4	List, Fibonacci, Sets / Tuples
Q5	List, Random, Dictionary

Q.1(a) — Algorithm: Definition & Example [3 marks]

Definition

An **algorithm** is a finite, ordered set of well-defined steps to solve a problem. It must be: **Unambiguous** · **Finite** · **Effective** · **Input/Output defined**.

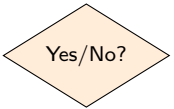
Algorithm: Area of a Circle

1. START
2. Read the radius `r`
3. Compute `area` $\leftarrow \pi \times r^2$ (use $\pi = 3.14159$)
4. PRINT `area`
5. STOP

Q.1(b) — Flowchart Symbols [4 marks – 1/2]

Oval
Terminal
Marks **start** or **end**

Rectangle
Process
Computation or operation

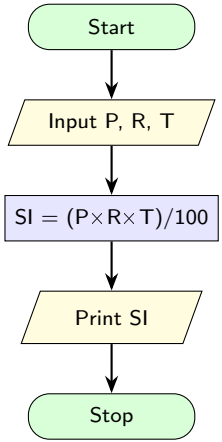


Diamond
Decision
Conditional branch

Parallelogram
Input / Output
Read input or print output

Flow Line (Arrow) — Connects symbols and shows the direction of execution. Every symbol must have at least one incoming or outgoing arrow.

Q.1(b) — Flowchart: Simple Interest [4 marks – 2/2]



Formula

$$\text{SI} = \frac{P \times R \times T}{100}$$

P	Principal amount
R	Rate of interest (%)
T	Time (years)

Python Code

```
P = float(input("Principal: "))
R = float(input("Rate %: "))
T = float(input("Time (yrs): "))
SI = (P * R * T) / 100
print("Simple Interest =", SI)
```

Q.1(c) — Python Data Types: Complete List [7 marks – 1/2]

Category	Type	Description	Example
Numeric	`int`	Whole numbers	`x = 42`
	`float`	Decimal numbers	`x = 3.14`
	`complex`	Real + imaginary	`x = 2+3j`
Text	`str`	Sequence of characters	`x = "Hi"`
Boolean	`bool`	`True` or `False` only	`x = True`
Collection	`list`	Ordered, mutable	`[1,2,3]`
	`tuple`	Ordered, immutable	`(1,2,3)`
	`set`	Unordered, unique	`{1,2,3}`
	`dict`	Key-value pairs	`{"a":1}`

Q.1(c) — Explaining Any Two Data Types [7 marks – 2/2]

int — Integer

Stores whole numbers. No decimal point. Can be positive, negative, or zero.

```
age = 20
temp = -5
print(type(age))    # <class 'int'>
print(age + 10)     # 30
print(age * 2)      # 40
```

str — String

Sequence of characters enclosed in quotes. Supports indexing and slicing.

```
name = "Python"
print(type(name))   # <class 'str'>
print(name[0])      # P
print(name[1:4])    # yth
print(len(name))    # 6
print(name.upper())# PYTHON
```

Key Rule

Use `type(var)` to check the data type of any variable at runtime.

Q.1(c) OR — Arithmetic Operators [7 marks – 1/2]

Arithmetic Operators

Op	Operation	Example
+	Addition	5+3 = 8
-	Subtraction	5-3 = 2
*	Multiplication	5*3 = 15
/	Division	7/2 = 3.5
//	Floor Division	7//2 = 3
%	Modulus	7%2 = 1
**	Exponent	2**3 = 8

Python Code

```
a = 10
b = 3
print(a + b)    # 13
print(a - b)    # 7
print(a * b)    # 30
print(a / b)    # 3.333...
print(a // b)   # 3
print(a % b)    # 1
print(a ** b)   # 1000
```

Q.1(c) OR — Relational Operators [7 marks – 2/2]

Relational (Comparison) Operators

Op	Meaning	Result
==	Equal to	True/False
!=	Not equal to	True/False
>	Greater than	True/False
<	Less than	True/False
>=	Greater or equal	True/False
<=	Less or equal	True/False

Note

Relational operators always return a bool (True or False). Used mainly in conditions.

Python Code

```
a = 10
b = 20
print(a == b)   # False
print(a != b)   # True
print(a > b)    # False
print(a < b)    # True
print(a >= 10)  # True
print(b <= 20)  # True

# Practical use in if:
if a < b:
    print("a is smaller")
```

Q.2(a) — Features of Python [3 marks]

- **Simple & Easy to Learn** — clean, English-like syntax
- **Free & Open Source** — no license cost
- **Interpreted** — line-by-line execution
- **Platform Independent** — runs on Windows, Linux, Mac
- **Dynamically Typed** — no need to declare types
- **Object-Oriented** — supports classes and objects
- **High-Level Language** — hides hardware details
- **Extensive Standard Library** — math, os, sys, ...
- **Embeddable** — can mix C/C++ code
- **Large Community** — active support ecosystem

Q.2(b) — if. . . else Statement [4 marks]

Syntax

```
if condition:
    # runs when True
elif condition2:
    # optional extra check
else:
    # runs when all False
```

- Uses **indentation** (no braces)
- `elif` and `else` are optional
- Conditions use relational / logical operators

Example: Positive, Negative or Zero

```
n = int(input("Enter number: "))

if n > 0:
    print("Positive number")
elif n < 0:
    print("Negative number")
else:
    print("Zero")
```

Example: Largest of two

```
a = int(input("a: "))
b = int(input("b: "))
if a > b:
    print("a is larger")
else:
    print("b is larger")
```

Q.2(c) — Pattern 1: @ Triangle [7 marks – 1/2]

Program

```
n = 5
for i in range(1, n+1):
    print("@ " * i)
```

Expected Output

```
@
@ @
@ @ @
@ @ @ @
@ @ @ @ @
```

Logic Explained

Iteration	i	Output
1st	1	@
2nd	2	@ @
3rd	3	@ @ @
4th	4	@ @ @ @
5th	5	@ @ @ @ @

The string `"@ "` is repeated `i` times using the `*` operator.

Q.2(c) — Pattern 2: Squares (1, 4, 9, 16...) [7 marks – 2/2]

Program

```
n = 5
for i in range(1, n+1):
    for j in range(1, i+1):
        print(j*j, end=" ")
    print()
```

Expected Output

```
1
1 4
1 4 9
1 4 9 16
1 4 9 16 25
```

Logic Explained

Outer loop controls rows (1 to n). Inner loop prints `j*j` for each column from 1 to `i`.
`end=" "` keeps values on the same line; `print()` moves to the next row.

Q.2 OR (a) — Applications of Python [3 marks]

- **Web Development** — Django, Flask
- **Data Science** — Pandas, NumPy, Matplotlib
- **Machine Learning / AI** — TensorFlow, Scikit-learn
- **Desktop GUI** — Tkinter, PyQt
- **Scientific Computing** — SciPy, SymPy
- **Game Development** — Pygame
- **Scripting / Automation** — automate tasks
- **Networking** — socket programming
- **Cybersecurity** — penetration testing
- **IoT / Embedded** — MicroPython, Raspberry Pi

Q.2 OR (b) — Nested if [4 marks]

Concept

A **nested if** is an `if` statement placed **inside another if** block. Used when a condition depends on a previous condition being true.

Syntax

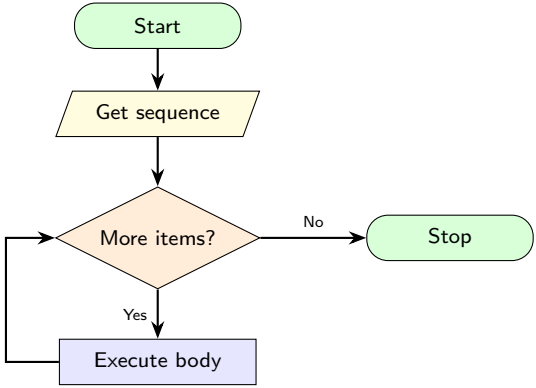
```
if condition1:
    if condition2:
        # both True
    else:
        # only cond1 True
else:
    # cond1 False
```

Example: Largest of 3 Numbers

```
a = int(input("a: "))
b = int(input("b: "))
c = int(input("c: "))

if a > b:
    if a > c:
        print("a is largest")
    else:
        print("c is largest")
else:
    if b > c:
        print("b is largest")
    else:
        print("c is largest")
```

Q.2 OR (c) — for Loop: Syntax & Flowchart [7 marks – 1/2]



Syntax

```
for variable in sequence:
    # loop body
```

Q.2 OR (c) — for Loop: Code Examples [7 marks – 2/2]

Example 1: Print 1 to 5

```
for i in range(1, 6):
    print(i, end=" ")
# Output: 1 2 3 4 5
```

Example 2: Sum of n numbers

```
n = int(input("Enter n: "))
total = 0
for i in range(1, n+1):
    total += i
print("Sum =", total)
```

Example 3: Iterate over a list

```
fruits = ["mango","apple","grape"]
for f in fruits:
    print(f)
```

Example 4: Multiplication table

```
n = int(input("Table of: "))
for i in range(1, 11):
    print(n, "x", i, "=", n*i)
```

Q.3(a) — Six Built-in String Functions [3 marks]

Function	Description
`upper()`	Convert to UPPERCASE
`lower()`	Convert to lowercase
`len()`	Return length of string
`replace(old,new)`	Replace a substring
`split()`	Split string into list
`strip()`	Remove leading/trailing whitespace

Quick Demo

```
s = "Hello World"
print(s.upper())           # HELLO WORLD
print(s.lower())           # hello world
print(len(s))              # 11
print(s.replace("World","GTU")) # Hello GTU
print(s.split())           # ['Hello', 'World']
```

Q.3(b) — for vs while & Program [4 marks]

for loop	while loop
Count known in advance	Count may be unknown
Iterates over a sequence	Runs while condition is True
Initialisation is implicit	Must initialise variable
Counter auto-updates	Must update counter manually
More compact syntax	More flexible

(2) Print 1 to 25 using for loop

```
for i in range(1, 26):
    print(i, end=" ")
# Output: 1 2 3 4 5 ... 24 25
```

Q.3(c) — Local Variables [7 marks – 1/2]

Definition

A **local variable** is declared **inside a function**. It exists only within that function and is destroyed when the function returns.

- Created when function is called
- Not accessible outside the function
- Two functions can have the same local variable name

Example

```
def add(a, b):
    result = a + b     # local variable
    print("Sum:", result)

def greet():
    msg = "Hello!"     # local variable
    print(msg)

add(3, 5)      # Sum: 8
greet()        # Hello!

# print(result)  --> NameError!
# 'result' is not defined here
```

Q.3(c) — Global Variables [7 marks – 2/2]

Definition

A **global variable** is declared **outside all functions**. It is accessible throughout the program. Use the `global` keyword to *modify* it inside a function.

Important Rule

Without `global` keyword, assigning inside a function creates a *new local* variable, not modifying the global.

Example

```
count = 0        # global variable

def increment():
    global count
    count = count + 1

increment()
increment()
print("Count:", count)  # Count: 2

# Without 'global' keyword:
x = 10
def demo():
    x = 99      # creates a NEW local x
    print(x)    # 99
demo()
print(x)        # 10  (global unchanged)
```

Q.3 OR (a) — String Operation: Concatenation [3 marks]

Concatenation

Joining two or more strings using the + operator (or * for repetition).

Concatenation with +

```
first  = "Hello"
second = " World"
result = first + second
print(result)       # Hello World

name = "GTU"
greeting = "Welcome to " + name
print(greeting)     # Welcome to GTU
```

Repetition with *

```
s = "Ha"
print(s * 3)        # HaHaHa

line = "-" * 20
print(line)         #
    --------------------
```

Note

You cannot concatenate a string and a number directly. Use `str()` to convert first.

Q.3 OR (b) — Prime Number Program [4 marks]

Program

```
n = int(input("Enter a number: "))

if n < 2:
    print(n, "is NOT prime")
else:
    is_prime = True
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            is_prime = False
            break

    if is_prime:
        print(n, "is a PRIME number")
    else:
        print(n, "is NOT a prime number")
```

Logic

1. Numbers < 2 are not prime
2. Check divisors from 2 to $\sqrt{n}$
3. If any divisor divides n exactly $\rightarrow$ not prime
4. No divisor found $\rightarrow$ prime

Sample Output

```
Enter a number:  7
7 is a PRIME number

Enter a number:  12
12 is NOT a prime number
```

Q.3 OR (c) — Python Math Library: Functions List [7 marks – 1/2]

Trigonometric & Logarithmic

`math.sin(x)`	Sine of x (radians)
`math.cos(x)`	Cosine of x
`math.tan(x)`	Tangent of x
`math.log(x)`	Natural log ($\ln x$)
`math.log10(x)`	Base-10 log

Constants

`math.pi`	$\pi = 3.14159\ldots$
`math.e`	$e = 2.71828\ldots$
`math.inf`	Infinity

Numeric / Rounding

`math.sqrt(x)`	Square root
`math.pow(x, y)`	x^y (returns float)
`math.fabs(x)`	Absolute value
`math.floor(x)`	Round down
`math.ceil(x)`	Round up
`math.factorial(n)`	$n!$
`math.gcd(a, b)`	GCD of a and b

Q.3 OR (c) — Explaining Any Three Math Functions [7 marks – 2/2]

`math.sqrt()`

Returns the square root of x.

```
import math
print(math.sqrt(25))
# 5.0
print(math.sqrt(2))
# 1.4142...
```

`math.floor()` & `ceil()`

`floor` rounds down; `ceil` rounds up.

```
import math
print(math.floor(4.9))
# 4
print(math.ceil(4.1))
# 5
```

`math.factorial()`

Returns $n!$ for non-negative integer.

```
import math
print(math.factorial(5))
# 120   (5x4x3x2x1)
print(math.factorial(0))
# 1
```

Q.4(a) — Six Built-in List Functions [3 marks]

Function	Description
`append(x)`	Add x at the end of list
`insert(i,x)`	Insert x at index i
`remove(x)`	Remove first occurrence of x
`pop(i)`	Remove and return element at i
`sort()`	Sort list in ascending order
`reverse()`	Reverse the list in-place

Quick Demo

```
L = [3, 1, 4, 1, 5]
L.append(9)      # [3,1,4,1,5,9]
L.insert(2, 0)   # [3,1,0,4,1,5,9]
L.remove(1)      # [3,0,4,1,5,9]
L.sort()         # [0,1,3,4,5,9]
L.reverse()      # [9,5,4,3,1,0]
print(L)
```

Q.4(b) — Fibonacci Series using UDF [4 marks]

Program

```
def fibonacci(n):
    a, b = 0, 1
    print("Fibonacci series:")
    while a <= n:
        print(a, end=" ")
        a, b = b, a + b
    print()

N = int(input("Enter N: "))
fibonacci(N)
```

Series Logic

$F_0 = 0$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$

Each term is the sum of the previous two.

Sample Output

```
Enter N: 20
Fibonacci series:
0 1 1 2 3 5 8 13
```

Q.4(c) — Set Functions & Operations: Reference [7 marks – 1/2]

Set Methods

Method	Description
`add(x)`	Add element x
`remove(x)`	Remove x (error if absent)
`discard(x)`	Remove x (no error)
`pop()`	Remove a random element
`clear()`	Remove all elements
`copy()`	Return a copy

Set Operations

Operation	Description
`union()`	$A \cup B$ — all elements
`intersection()`	$A \cap B$ — common only
`difference()`	$A - B$ — in A, not B
`issubset()`	Is $A \subseteq B$?
`issuperset()`	Is $A \supseteq B$?
`symmetric_difference()`	in A or B, not both

Q.4(c) — Set Demonstration Program [7 marks – 2/2]

Program

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}

print("A =", A)
print("B =", B)

A.add(10)
print("After add(10):", A)
A.discard(10)

print("union:         ", A | B)
print("intersection:  ", A & B)
print("difference:    ", A - B)
print("sym_diff:      ", A ^ B)
print("issubset:      ", {1,2}.issubset(A))
print("len(A):        ", len(A))
```

Output

```
A = {1, 2, 3, 4, 5}
B = {4, 5, 6, 7, 8}
After add(10):  {1,2,3,4,5,10}
union:  {1,2,3,4,5,6,7,8}
intersection:  {4, 5}
difference:  {1, 2, 3}
sym_diff:  {1,2,3,6,7,8}
issubset:  True
len(A): 5
```

Q.4 OR (a) — Count Vowels in a String [3 marks]

Program

```
s = input("Enter a string: ")
count = 0
vowels = "aeiouAEIOU"

for ch in s:
    if ch in vowels:
        count += 1

print("Number of vowels:", count)
```

Logic

1. Read the string
2. Loop through each character
3. If character is a vowel, increment counter
4. Print the final count

Sample Output

```
Enter:  Hello World
Number of vowels:  3
```

Q.4 OR (b) — Factorial using User-Defined Function [4 marks]

Iterative UDF

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

num = int(input("Enter number: "))
print("Factorial of", num, "=",
    factorial(num))
```

Formula

$$n! = n \times (n-1) \times \cdots \times 2 \times 1$$

$0! = 1$ (by definition)

Trace for n=5

i	result
1	1
2	2
3	6
4	24
5	**120**

Q.4 OR (c) — Tuple Operations & Functions: Reference [7 marks – 1/2]

Tuple Operations

Operation	Description
`t[i]`	Indexing (access element)
`t[a:b]`	Slicing (sub-tuple)
`t1 + t2`	Concatenation
`t * n`	Repetition
`x in t`	Membership test
`len(t)`	Length
`max(t)`	Maximum value
`min(t)`	Minimum value

Tuple Methods

Method	Description
`count(x)`	Count occurrences of x
`index(x)`	First index of x

Tuples are Immutable

No append, remove, or sort in-place.
Use `sorted(t)` for a sorted list.

Q.4 OR (c) — Tuple Operations: Code Demo [7 marks – 2/2]

Program

```
t1 = (10, 20, 30, 40, 50)
t2 = (60, 70)
print(t1[0])        # 10
print(t1[1:4])      # (20, 30, 40)
print(t1[-1])       # 50
print(t1 + t2)      # (10, 20, 30, 40, 50,
    60, 70)
print(t2 * 2)       # (60, 70, 60, 70)
print(30 in t1)     # True
print(len(t1))      # 5
print(max(t1))      # 50
print(min(t1))      # 10
t3 = (1, 2, 2, 3, 2)
print(t3.count(2)) # 3
print(t3.index(3)) # 3
```

Output

```
10
(20, 30, 40)
50
(10, 20, 30, 40, 50, 60, 70)
(60, 70, 60, 70)
True
5
50
10
3
3
```

Q.5(a) — Multiplication of All List Elements [3 marks]

Program

```
nums = [2, 3, 4, 5, 6]
product = 1

for n in nums:
    product = product * n

print("Product of all elements:", product)
```

Logic

1. Initialise `product = 1`
2. Loop through each element
3. Multiply element into `product`
4. Print result

Trace

n	product
2	2
3	6
4	24
5	120
6	**720**

Q.5(b) — Random Module: Functions [4 marks – 1/2]

What is the `random` module?

Python's built-in `random` module provides functions for generating **pseudo-random numbers** and making random selections. Import it with `import random`.

Function	Description
`random()`	Returns float in $[0.0, 1.0)$
`randint(a, b)`	Random integer from a to b (inclusive)
`randrange(a, b)`	Random integer from a to b-1
`choice(seq)`	Random element from a sequence
`choices(seq, k=n)`	n random elements (with replacement)
`shuffle(lst)`	Shuffle a list in-place
`sample(seq, k)`	k unique random elements
`uniform(a, b)`	Random float between a and b

Q.5(b) — Random Module: Code Examples [4 marks – 2/2]

Demonstration Program

```
import random

# Random float [0, 1)
print(random.random())

# Random integer 1 to 10
print(random.randint(1, 10))

# Random choice from list
colors = ["red", "blue", "green"]
print(random.choice(colors))

# Shuffle a list
cards = [1, 2, 3, 4, 5]
random.shuffle(cards)
print("Shuffled:", cards)

# 3 unique samples
print(random.sample(cards, 3))
```

Sample Output

```
0.7324561...
7
blue
Shuffled:  [3, 1, 5, 2, 4]
[3, 5, 2]
```

Note

Results vary each run since they are random. Use `random.seed(n)` to get reproducible results.

Q.5(c) — Dictionary Built-in Functions [7 marks – 1/2]

Core Functions

Function	Description
`keys()`	Returns all keys
`values()`	Returns all values
`items()`	Returns (key, value) pairs
`get(k)`	Value for key k (no KeyError)
`update(d2)`	Merge dict d2 into dict
`pop(k)`	Remove and return value for k
`clear()`	Remove all items
`copy()`	Shallow copy
`len(d)`	Number of key-value pairs
`in`	Check if key exists

`get()` vs direct access

`d["x"]` raises `KeyError` if `"x"` missing.
`d.get("x")` returns `None` safely.
`d.get("x", "N/A")` returns default.

Creating a Dictionary

```
student = {
    "name":  "Alice",
    "age":   20,
    "grade": "A"
}
```

Q.5(c) — Dictionary Functions: Code Demo [7 marks – 2/2]

Program

```
d = {"name":"Alice","age":20,"city":"Surat"}
print(list(d.keys()))      # ['name','age','city']
print(list(d.values()))    # ['Alice', 20, 'Surat']
print(list(d.items()))
print(d.get("age"))        # 20
print(d.get("marks","N/A"))  # N/A
d.update({"grade":"A"})
print(d)
d.pop("city")
print(d)
print(len(d))              # 3
print("name" in d)         # True
d2 = d.copy()
print(d2)
```

Output

```
keys:  ['name', 'age', 'city']
values:  ['Alice', 20, 'Surat']
items:  [('name','Alice'),('age',20),('city','Surat')]
get:  20
get:  N/A
update:  {'name':'Alice','age':20,'city':'Surat','grade':'A'}
pop:  {'name':'Alice','age':20,'grade':'A'}
len:  3
in:  True
copy:  {'name':'Alice','age':20,'grade':'A'}
```

Q.5 OR (a) — Dictionary: Students Scoring above 70 [3 marks]

Program

```
n = int(input("Number of students: "))
students = {}

for i in range(n):
    roll  = int(input("Roll No: "))
    name  = input("Name: ")
    marks = int(input("Marks: "))
    students[roll] = {"name": name, "marks":
        marks}

print("\nStudents with marks above 70:")
for roll, info in students.items():
    if info["marks"] > 70:
        print(info["name"], "-", info["marks"
            ])
```

Sample Output

```
Number of students:  3
Roll:  1   Name:  Alice   Marks:  85
Roll:  2   Name:  Bob    Marks:  60
Roll:  3   Name:  Carol   Marks:  92

Students above 70:
Alice - 85
Carol - 92
```

Q.5 OR (b) — Statistics Module [4 marks]

Key Functions

Function	Returns
`mean(data)`	Arith. mean
`median(data)`	Middle value
`mode(data)`	Most frequent
`stdev(data)`	Std. deviation
`variance(data)`	Variance
`harmonic_mean(data)`	Harmonic mean

Program

```
import statistics as st
data = [4, 8, 6, 5, 3, 8, 7, 2]
print("Mean     :", st.mean(data))
print("Median   :", st.median(data))
print("Mode     :", st.mode(data))
print("Stdev    :", round(st.stdev(data), 2))
print("Variance:", round(st.variance(data),
    2))
```

Output

```
Mean :  5.375
Median :  5.5
Mode :  8
Stdev :  2.13
Variance:  4.55
```

Q.5 OR (c) — List Indexing [7 marks – 1/2]

Concept

Each element in a list has an **index**. Python supports both **positive** (left-to-right) and **negative** (right-to-left) indexing.

	10	20	30	40	50
+	0	1	2	3	4
–	-5	-4	-3	-2	-1

Indexing Examples

```
L = [10, 20, 30, 40, 50]
#     0   1   2   3   4    positive
#    -5  -4  -3  -2  -1    negative

print(L[0])     # 10
print(L[3])     # 40
print(L[-1])    # 50
print(L[-3])    # 30
```

Index Error

Accessing an index outside range raises `IndexError`.

Q.5 OR (c) — List Slicing [7 marks – 2/2]

Slicing Syntax

`L[start : stop : step]`

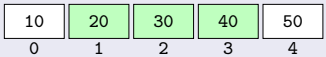
`start`	Index to begin (inclusive)
`stop`	Index to end (exclusive)
`step`	Increment (default 1)

Slicing Examples

```
L = [10, 20, 30, 40, 50]

print(L[1:4])    # [20, 30, 40]
print(L[:3])     # [10, 20, 30]
print(L[2:])     # [30, 40, 50]
print(L[::2])    # [10, 30, 50]
print(L[::-1])   # [50, 40, 30, 20, 10]
print(L[1:4:2]) # [20, 40]
```

Visual: `L[1:4]`



Highlighted = selected

Key Rule

`stop` index is **excluded**. `L[1:4]` gives elements at indices 1, 2, 3 — not 4.

`L[::-1]` reverses the list

Step $= -1$ walks from end to start.

Summary — Winter 2025 Solutions

Q	Topic	Marks
Q1(a)	Algorithm definition + area of circle	3
Q1(b)	Flowchart symbols + simple interest FC	4
Q1(c)	Data types list + explain two	7
Q1(c) OR	Arithmetic + Relational operators	7
Q2(a)	Features of Python	3
Q2(b)	if...else with examples	4
Q2(c)	Patterns: @ triangle + squares	7
Q2 OR	Applications / Nested if / for loop	3+4+7
Q3(a)	6 built-in string functions	3
Q3(b)	for vs while + print 1-25	4
Q3(c)	Local and global variables	7
Q3 OR	String op / Prime / Math functions	3+4+7
Q4(a)	6 built-in list functions	3
Q4(b)	Fibonacci UDF	4
Q4(c)	Set operations + program	7
Q4 OR	Vowels / Factorial / Tuple ops	3+4+7
Q5(a)	Product of list elements	3
Q5(b)	Random module + examples	4
Q5(c)	Dictionary functions + demo	7
Q5 OR	Student dict / Statistics / List index & slice	3+4+7