Python Programming (DI02032011)

Summer 2025 Exam Solutions

Milav Dabgar

Lecturer, Dept of EC
GP Palanpur

Date of Exam: 04 June 2025

Exam Overview

Exam Details

Subject: Python Programming (DI02032011)
Date: 04-06-2025 **Time:** 10:30 AM – 01:00 PM
Marks: 70 **Questions:** Q1–Q5

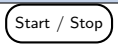
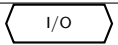
Topics Covered

Q1	Problem Solving, Flowcharts, Patterns	14 marks
Q2	Python Features, Data Types, if-else	14 marks
Q3	Applications, Loops, Operators	14 marks
Q4	Variables, Loops, Modules / UDFs	14 marks
Q5	Lists, Strings, Sets / Dicts	14 marks

Q.1(a) — Flowchart: Definition & Symbols [3 marks]

Definition

A **flowchart** is a diagrammatic representation of an algorithm using standard symbols connected by arrows to show the flow of execution.

Symbol	Name	Use
	Terminal (Oval)	Marks the **start** or **end** of the flowchart
	Process (Rectangle)	Represents a **calculation or operation**
	Decision (Diamond)	Represents a **condition / branching** (Yes/No)
	Input/Output (Parallelogram)	Represents **input** from user or **output** to screen

Q.1(b) — Algorithms [4 marks]

(1) Maximum of Two Numbers

1. START
2. Read A, B
3. IF A $>$ B THEN
   PRINT "A is maximum"
   ELSE
   PRINT "B is maximum"
   END IF
4. STOP

(2) Sum of Two Numbers

1. START
2. Read A, B
3. SUM $\leftarrow$ A + B
4. PRINT SUM
5. STOP

Q.1(c) — Pattern Programs [7 marks]

Pattern 1: & Triangle

```
for i in range(1, 6):
    print("& " * i)
```

Output:

```
&
& &
& & &
& & & &
& & & & &
```

Pattern 2: Number Triangle

```
for i in range(1, 6):
    for j in range(1, i+1):
        print(j, end=" ")
    print()
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Q.1(c) OR — Odd/Even & Average [7 marks]

(1) Odd or Even Check

```
n = int(input("Enter number: "))
if n % 2 == 0:
    print(n, "is Even")
else:
    print(n, "is Odd")
```

(2) Average of 1 to n

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

Q.2(a) — Features of Python [3 marks]

1. **Simple & Easy to Learn:** Clean syntax close to English
2. **Free & Open Source:** Available at no cost, source code is open
3. **High-Level Language:** Programmer does not manage memory manually
4. **Interpreted:** Code is executed line-by-line; easy to debug
5. **Portable / Platform Independent:** Runs on Windows, Linux, Mac
6. **Object-Oriented:** Supports classes, objects, inheritance
7. **Dynamically Typed:** No need to declare variable types
8. **Extensive Libraries:** Rich standard library (math, os, sys, ...)
9. **Embeddable & Extensible:** Can embed C/C++ code

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

Syntax

```
if condition:
    # block if True
elif condition2:
    # block if cond2
else:
    # block if all False
```

Example: Grade Check

```
marks = int(input("Enter marks: "))
if marks >= 70:
    print("Distinction")
elif marks >= 60:
    print("First Class")
elif marks >= 45:
    print("Pass")
else:
    print("Fail")
```

Key Points

- Uses indentation (no braces)
- `elif` is optional, can be many
- `else` is optional, at most one

Q.2(c) — Python Data Types [7 marks]

Type	Keyword	Description	Example
Integer	`int`	Whole numbers (positive/negative)	`x = 10`
Float	`float`	Numbers with decimal point	`x = 3.14`
Complex	`complex`	Numbers with real+imaginary part	`x = 2+3j`
String	`str`	Sequence of characters	`x = "Hello"`
Boolean	`bool`	True or False only	`x = True`
List	`list`	Ordered, mutable collection	`x = [1,2,3]`
Tuple	`tuple`	Ordered, immutable collection	`x = (1,2,3)`
Set	`set`	Unordered, unique elements	`x = {1,2,3}`
Dictionary	`dict`	Key-value pairs	`x = {"a":1}`

Check type with type()

```
print(type(10))       # <class 'int'>
print(type(3.14))     # <class 'float'>
print(type("Hi"))     # <class 'str'>
print(type(True))     # <class 'bool'>
```

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

1. **Web Development:** Django, Flask frameworks
2. **Data Science & Analysis:** NumPy, Pandas, Matplotlib
3. **Machine Learning / AI:** TensorFlow, Scikit-learn
4. **Desktop GUI Applications:** Tkinter, PyQt
5. **Scripting / Automation:** Automate repetitive tasks
6. **Scientific Computing:** SciPy, SymPy
7. **Game Development:** Pygame library
8. **Networking:** Socket programming, web scraping
9. **Cybersecurity:** Penetration testing tools
10. **Embedded Systems / IoT:** MicroPython on Raspberry Pi

Q.2 OR (b) — while Loop [4 marks]

Syntax

```
while condition:
    # body
    # update variable
```

Key Points

- Condition checked **before** each iteration
- Loop runs while condition is `True`
- Must update the variable to avoid infinite loop

Example: Print 1 to 5

```
i = 1
while i <= 5:
    print(i)
    i = i + 1
```

Example: Sum till 0

```
total = 0
n = int(input("Enter (0 to stop): "))
while n != 0:
    total += n
    n = int(input("Enter: "))
print("Sum =", total)
```

Q.2 OR (c) — Types of Operators in Python [7 marks]

Types of Operators

1. Arithmetic Operators
2. Comparison (Relational) Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operators
6. Membership Operators (`in`, `not in`)
7. Identity Operators (`is`, `is not`)

1. Arithmetic Operators

`+`	Addition	`5+3=8`
`-`	Subtraction	`5-3=2`
`*`	Multiplication	`5*3=15`
`/`	Division	`5/2=2.5`
`//`	Floor Division	`5//2=2`
`%`	Modulus	`5%2=1`
`**`	Exponent	`2**3=8`

2. Comparison Operators

`==`	Equal to	`5==5` → `True`
`!=`	Not equal	`5!=3` → `True`
`>`	Greater than	`5>3` → `True`
`<`	Less than	`3<5` → `True`
`>=`	Greater or equal	`5>=5` → `True`
`<=`	Less or equal	`3<=5` → `True`

Q.3(a) — Local Variable [3 marks]

Definition

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

Example

```
def greet():
    msg = "Hello, World!"  # local
    print(msg)

greet()
# print(msg)  # ERROR! msg
              # not accessible here
```

Key Points

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

Q.3(b) — for Loop [4 marks]

Syntax

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

Key Points

- Iterates over a sequence (list, string, range)
- range(n) gives 0 to n-1
- range(a,b) gives a to b-1
- Known number of iterations

Example 1: Print 1 to 5

```
for i in range(1, 6):
    print(i)
```

Example 2: Sum of 1 to n

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

Example 3: Iterate List

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

Q.3(c) — Math Module [7 marks]

Common math Functions

`math.sqrt(x)`	Square root
`math.pow(x,y)`	x^y
`math.floor(x)`	Floor value
`math.ceil(x)`	Ceiling value
`math.fabs(x)`	Absolute value
`math.factorial(x)`	Factorial
`math.log(x)`	Natural log
`math.sin(x)`	Sine (radians)
`math.pi`	$\pi = 3.14159\ldots$
`math.e`	$e = 2.71828\ldots$

Program demonstrating math module

```
import math

x = 16
print("sqrt(16) =", math.sqrt(x))
print("pow(2,3) =", math.pow(2, 3))
print("floor(4.7) =", math.floor(4.7))
print("ceil(4.2) =", math.ceil(4.2))
print("fabs(-5) =", math.fabs(-5))
print("factorial(5) =",math.factorial(5))
print("log(math.e) =", math.log(math.e))
print("pi =", math.pi)
```

Output:

```
sqrt(16) = 4.0    pow(2,3) = 8.0
floor(4.7) = 4    ceil(4.2) = 5
fabs(-5) = 5.0    factorial(5) = 120
log(math.e) = 1.0 pi = 3.14159...
```

Q.3 OR (a) — Global Variable [3 marks]

Definition

A **global variable** is declared **outside all functions**. It is accessible anywhere in the program. To modify it inside a function, use the `global` keyword.

Example

```
count = 0          # global variable

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

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

Key Points

- Accessible from any function
- Without `global` keyword, assignment creates a new local variable
- Overuse of globals makes code hard to debug

Q.3 OR (b) — Break Statement [4 marks]

Definition & Syntax

break terminates the loop immediately and execution continues after the loop body.

```
for/while ...:
    if condition:
        break
    # rest of body
```

Key Points

- Works in both `for` and `while` loops
- Only breaks the innermost loop
- Used to exit loop early when a condition is met

Example 1: Stop at 3

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
# Output: 1  2
```

Example 2: Search in list

```
nums = [5, 8, 12, 3, 7]
target = 12
for n in nums:
    if n == target:
        print("Found:", n)
        break
```

Q.3 OR (c) — Statistics Module [7 marks]

statistics Functions

`mean(data)`	Arithmetic mean
`median(data)`	Middle value
`mode(data)`	Most frequent
`stdev(data)`	Standard deviation
`variance(data)`	Variance
`median_low()`	Lower median
`median_high()`	Higher median
`harmonic_mean()`	Harmonic mean

Demonstration Program

```
import statistics as st

data = [4, 8, 6, 5, 3, 8, 7]

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     : 5.857142...
Median   : 6
Mode     : 8
Stdev    : 1.86
Variance: 3.47
```

Q.4(a) — Nested Tuple [3 marks]

Definition

A **nested tuple** is a tuple that contains one or more tuples as elements, used to represent multi-dimensional data.

Example

```
students = (
    ("Alice", 20, "CS"),
    ("Bob",   21, "IT"),
    ("Carol", 19, "EC")
)
# Access elements
print(students[0])     # ('Alice',20,'CS')
print(students[1][0]) # Bob
print(students[2][2]) # EC
# Iterate
for s in students:
    print(s[0], s[1])
```

Key Points

- Outer tuple contains inner tuples
- Access via `t[i][j]`
- Immutable – cannot be changed
- Useful for table-like data

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

Iterative Approach

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

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

Recursive Approach

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    return n * factorial(n - 1)

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

Sample Run

```
Enter number:  5
Factorial of 5 = 120
```

Q.4(c) — List Operations [7 marks]

List Operations with Methods

`append(x)`	Add x at end
`insert(i,x)`	Insert x at index i
`remove(x)`	Remove first x
`pop(i)`	Remove at index i
`sort()`	Sort in ascending order
`reverse()`	Reverse the list
`index(x)`	Return index of x
`count(x)`	Count occurrences of x
`len(list)`	Length of list
`+`	Concatenate two lists
`*`	Repeat list

Demonstration

```
L = [3, 1, 4, 1, 5]
L.append(9)
print("append:", L)
L.insert(2, 99)
print("insert:", L)
L.remove(1)
print("remove:", L)
L.sort()
print("sort:  ", L)
L.reverse()
print("reverse:", L)
print("index(5):", L.index(5))
print("count(1):", L.count(1))
print("len:    ", len(L))
print("concat: ", L + [10, 11])
print("repeat: ", [0]*3)
```

Q.4 OR (a) — String Indexing [3 marks]

Definition

Indexing allows accessing individual characters of a string using their position. Python supports both **positive** (left to right) and **negative** (right to left) indexing.

	P	Y	T	H	O	N
+	0	1	2	3	4	5
−	-6	-5	-4	-3	-2	-1

Example

```
s = "PYTHON"
#     P   Y   T   H   O   N
#     0   1   2   3   4   5    (positive)
#    -6  -5  -4  -3  -2  -1    (negative)

print(s[0])     # P
print(s[3])     # H
print(s[-1])    # N
print(s[-3])    # H
print(s[1:4])   # YTH   (slicing)
print(s[:3])    # PYT
print(s[::2])   # PTN   (step)
```

Slicing Syntax

`s[start:stop:step]`
Default: start=0, stop=end, step=1

Q.4 OR (b) — Prime Number Check using UDF [4 marks]

Program

```
def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

num = int(input("Enter number: "))
if is_prime(num):
    print(num, "is a Prime number")
else:
    print(num, "is NOT a Prime number")
```

Logic Explanation

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

Sample Run

```
Enter number:  7
7 is a Prime number

Enter number:  9
9 is NOT a Prime number
```

Q.4 OR (c) — Tuple Operations [7 marks]

Tuple Operations & Functions

`len(t)`	Length
`t[i]`	Indexing
`t[i:j]`	Slicing
`t1 + t2`	Concatenation
`t * n`	Repetition
`x in t`	Membership
`count(x)`	Count of x
`index(x)`	Index of x
`max(t)`	Maximum value
`min(t)`	Minimum value
`sorted(t)`	Returns sorted list

Tuples are Immutable

No append, remove, or sort in-place

Demonstration

```
t = (4, 2, 7, 2, 9, 1)
t2 = (10, 11)

print("len:     ", len(t))
print("index:   ", t[2])
print("slice:   ", t[1:4])
print("concat:  ", t + t2)
print("repeat:  ", t2 * 2)
print("in:      ", 7 in t)
print("count:   ", t.count(2))
print("index(): ", t.index(9))
print("max:     ", max(t))
print("min:     ", min(t))
print("sorted:  ", sorted(t))
```

Q.5(a) — Smallest Number in List [3 marks]

Using min() function

```
List1 = [21, 6, 3, 18, 12, 15]
smallest = min(List1)
print("Smallest number:", smallest)
# Output: Smallest number: 3
```

Without built-in min()

```
List1 = [21, 6, 3, 18, 12, 15]
smallest = List1[0]
for num in List1:
    if num < smallest:
        smallest = num
print("Smallest:", smallest)
```

Logic (Manual approach)

1. Assume first element is smallest
2. Loop through each element
3. If element $<$ current smallest, update
4. Print final smallest

Output

```
Smallest number:  3
```

Q.5(b) — Built-in String Functions [4 marks]

Program demonstrating 4 built-in string functions

```
s = "Hello World Python"

# 1. upper() - converts to UPPERCASE
print(s.upper())              # HELLO WORLD PYTHON

# 2. lower() - converts to lowercase
print(s.lower())              # hello world python

# 3. replace() - replaces a substring
print(s.replace("World", "GTU"))  # Hello GTU Python

# 4. split() - splits string into a list
words = s.split()
print(words)     # ['Hello', 'World', 'Python']
```

Other functions: `find(x)`, `count(x)`, `strip()`, `len(s)`, `title()`, `isdigit()`

Q.5(c) — Set Functions & Operations [7 marks]

Set Functions & Methods

`add(x)`	Add element
`remove(x)`	Remove (error if absent)
`discard(x)`	Remove (no error)
`pop()`	Remove random
`clear()`	Remove all
`union()`	$A \cup B$
`intersection()`	$A \cap B$
`difference()`	$A - B$
`issubset()`	$A \subseteq B$?
`len(s)`	Size of set

Demonstration Program

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

A.add(10)
print("add:          ", A)
A.discard(10)
print("discard:      ", A)
print("union:        ", A.union(B))
print("intersection:", A.intersection(B))
print("difference:   ", A.difference(B))
print("issubset:     ", {1,2}.issubset(A))
print("len:          ", len(A))
```

Output:

```
union:        {1,2,3,4,5,6,7,8}
intersection: {4,5}
difference:   {1,2,3}
issubset:     True
```

Q.5 OR (a) — Swap Two Elements in List [3 marks]

Program

```
List1 = [1, 3, 5, 7, 9, 11]
print("Original:", List1)

# Swap element at index 1 and index 4
# (i.e., swap 3 and 9)
i = 1
j = 4
List1[i], List1[j] = List1[j], List1[i]

print("After swap:", List1)
```

Output

```
Original:  [1, 3, 5, 7, 9, 11]
After swap:  [1, 9, 5, 7, 3, 11]
```

Using Temporary Variable

```
temp = List1[i]
List1[i] = List1[j]
List1[j] = temp
```

Python Trick

Python allows direct swap without a temp variable:
`a, b = b, a`

Q.5 OR (b) — Built-in Tuple Functions [4 marks]

Program demonstrating 4 built-in tuple functions

```
t = (5, 3, 8, 1, 9, 3, 7)

# 1. len() - number of elements
print("len()   :", len(t))     # 7

# 2. max() - maximum element
print("max()   :", max(t))     # 9

# 3. min() - minimum element
print("min()   :", min(t))     # 1

# 4. sum() - sum of all elements
print("sum()   :", sum(t))     # 36

print("count(3):", t.count(3))        # 2
print("index(8):", t.index(8))        # 2
print("sorted(): ", sorted(t))        # [1,3,3,5,7,8,9]
```

Q.5 OR (c) — Dictionary Functions [7 marks]

Dictionary Functions

`keys()`	All keys
`values()`	All values
`items()`	All key-value pairs
`get(k)`	Value for key k
`update()`	Add/update pairs
`pop(k)`	Remove key k
`clear()`	Remove all pairs
`copy()`	Shallow copy
`len(d)`	Number of pairs
`in`	Check key exists

Demonstration Program

```
d = {"name":"Alice","age":20,"city":"Surat"}

print("keys:  ", list(d.keys()))
print("values:", list(d.values()))
print("items: ", list(d.items()))
print("get:   ", d.get("age"))
d.update({"grade": "A"})
print("update:", d)
d.pop("city")
print("pop:   ", d)
print("len:   ", len(d))
print("in:    ", "name" in d)
d2 = d.copy()
print("copy:  ", d2)
```

Summary — All Questions Covered

Summer 2025 – Solutions Summary

Q1(a)	Flowchart definition & 4 symbols	3
Q1(b)	Algorithms: max of two / sum of two	4
Q1(c) / OR	Patterns / Odd-Even & Average	7
Q2(a)	Features of Python	3
Q2(b)	if. . . else with example	4
Q2(c)	Data types (all 9 types)	7
Q2 OR (a)	Applications of Python	3
Q2 OR (b)	while loop with example	4
Q2 OR (c)	Operators: 7 types, 2 explained	7
Q3(a)	Local variable with example	3
Q3(b)	for loop with examples	4
Q3(c)	math module functions & program	7
Q3 OR (a)	Global variable with example	3
Q3 OR (b)	break statement with examples	4
Q3 OR (c)	statistics module functions & program	7
Q4(a/b/c)	Nested tuple / Factorial UDF / List ops	3+4+7
Q4 OR (a/b/c)	String indexing / Prime UDF / Tuple ops	3+4+7
Q5(a)	Smallest in list / Swap elements	3
Q5(b)	String / Tuple built-in functions	4
Q5(c)	Set operations / Dictionary functions	7