Subject Name Solutions

4311601 – Winter 2023

Semester 1 Study Material

Detailed Solutions and Explanations

Question 1(a) [3 marks]

What is Flow chart? List out symbols used in Flow chart.

Solution

A **flowchart** is a graphical representation of an algorithm that shows the sequence of steps and decision points in a process using standardized symbols.
Flowchart Symbols Table:

Symbol	Name	Purpose
Oval	Terminal	Start/End of program
Rectangle	Process	Processing/Calculation steps
Diamond	Decision	Conditional statements
Parallelogram	Input/Output	Data input or output
Circle	Connector	Connect flowchart parts
Arrow	Flow line	Direction of flow

Key Points:

- **Visual representation**: Shows program logic graphically
- **Step-by-step**: Displays sequential flow of operations
- **Decision making**: Diamond symbols show conditional branches

Mnemonic

"Flow Charts Show Program Steps Visually"

Question 1(b) [4 marks]

Write a short note on for loop.

Solution

The **for loop** is used to iterate over a sequence (list, tuple, string, range) in Python.
For Loop Table:

Component	Syntax	Example
Basic	`for variable in sequence:`	`for i in range(5):`
Range	`range(start, stop, step)`	`range(1, 10, 2)`
List	`for item in list:`	`for x in [1,2,3]:`
String	`for char in string:`	`for c in "hello":`

Simple Code Example:

```
for i in range(3):
    print(i)
\# Output: 0, 1, 2
```

Key Features:

- **Automatic iteration**: No manual counter needed
- **Sequence traversal**: Works with any iterable object
- **Range function**: Creates number sequences easily

Mnemonic

"For Loops Iterate Through Sequences"

Question 1(c) [7 marks]

Write a program to display Fibonacci series up to nth term where n is provided by the user.

Solution

Fibonacci Series Program:

```
\# Get number of terms from user
n = int(input("Enter number of terms: "))

\# Initialize first two terms
a, b = 0, 1

\# Display first term
if n {=} 1:
    print(a, end=" ")

\# Display second term
if n {=} 2:
    print(b, end=" ")

\# Generate remaining terms
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a, b = b, c
```

Algorithm Flow:

```
flowchart LR
    A[Start] {-{-} B[Input n]}
    B {-{-} C\{n = 1?\}}
    C {-{-}|Yes| D[Print 0]}
    C {-{-}|No| H[End]}
    D {-{-} E\{n = 2?\}}
    E {-{-}|Yes| F[Print 1]}
    E {-{-}|No| H}
    F {-{-} G[Loop i=2 to n{-}1]}
    G {-{-} I[c = a + b]}
    I {-{-} J[Print c]}
J {-{-} K[a = b,

b = c]}

    K {-{-} L\{i  n{-}1?\}}
    L {-{-}|Yes| G}
    L {-{-}|No| H[End]}
```

Key Concepts:
- **Sequential generation**: Each term = sum of previous two
- **Variable swapping**: Update a, b values efficiently
- **User input**: Dynamic series length

Mnemonic

"Fibonacci: Add Previous Two Numbers"

Question 1(c OR) [7 marks]

Draw a flow chart to print ODD numbers from 1 to 100.

Solution

Flowchart for ODD Numbers 1 to 100:

```
flowchart LR
    A[Start] {-{-} B[i = 1]}
    B {-{-} C\{i = 100?\}}
    C {-{-}|Yes| D\{i \% 2 != 0?\}}
    D {-{-}|Yes| E[Print i]}
    D {-{-}|No| F[i = i + 1]}
    E {-{-} F}
    F {-{-} C}
    C {-{-}|No| G[End]}
```

Corresponding Python Code:

```
for i in range(1, 101):
    if i \% 2 != 0:
        print(i, end=" ")
```

Alternative Method:

```
for i in range(1, 101, 2):
    print(i, end=" ")
```

Key Elements:
- **Loop control**: i from 1 to 100
- **Odd check**: i % 2 != 0 condition
- **Step increment**: Move to next number

Mnemonic

"Odd Numbers: Remainder 1 When Divided by 2"

Question 2(a) [3 marks]

Write a Program to find whether a number is Palindrome or not.

Solution

Palindrome Check Program:

```
\# Input number
num = int(input("Enter a number: "))
temp = num
reverse = 0

\# Reverse the number
while temp {} 0:
```

```
    reverse = reverse * 10 + temp \% 10
    temp = temp // 10

\# Check palindrome
if num == reverse:
    print(f"\{num\} is palindrome")
else:
    print(f"\{num\} is not palindrome")
```

Algorithm Table:

Step	Operation	Example (121)
1	Get last digit	121 % 10 = 1
2	Build reverse	0*10 + 1 = 1
3	Remove last digit	121 // 10 = 12
4	Repeat until 0	Continue process

Key Points:

- **Digit extraction**: Use modulo (%) operator
- **Reverse building**: Multiply by 10 and add digit
- **Comparison**: Original equals reversed

Mnemonic

"Palindrome Reads Same Forward Backward"

Question 2(b) [4 marks]

Explain features of Python Programming.

Solution

Python Features Table:

Feature	Description	Benefit
Easy Syntax	Simple, readable code	Faster development
Interpreted	No compilation needed	Quick testing
Object-Oriented	Classes and objects support	Code reusability
Open Source	Free to use	No licensing cost
Cross-Platform	Runs on multiple OS	Wide compatibility
Large Libraries	Extensive built-in modules	Rich functionality

Key Advantages:

- **Beginner-friendly**: Easy to learn and understand
- **Versatile**: Web development, AI, data science
- **Community support**: Large developer community
- **Dynamic typing**: No variable type declaration needed

Mnemonic

"Python: Easy, Powerful, Popular Programming"

Question 2(c) [7 marks]

Explain basic structure of Python Program.

Solution

Python Program Structure:

```
\#!/usr/bin/env python3
\# Shebang line (optional)

"""
Documentation string (docstring)
Describes program purpose
"""

\# Import statements
import math
from datetime import date

\# Global variables
PI = 3.14159
count = 0

\# Function definitions
def calculate\_area(radius):
    """Calculate circle area"""
    return PI * radius * radius

\# Class definitions
class Calculator:
    def \_\_init\_\_(self):
        self.result = 0

\# Main program execution
if \_\_name\_\_ == "\_\_main\_\_":
    \# Program logic here
    radius = 5
    area = calculate\_area(radius)
    print(f"Area: \{area\}")
```

Structure Components Table:

Component	Purpose	Example
Shebang	System interpreter	`#!/usr/bin/env python3`
Docstring	Program documentation	`"""Program description"""`
Imports	External modules	`import math`
Variables	Global data storage	`PI = 3.14159`
Functions	Reusable code blocks	`def function_name():`
Classes	Object templates	`class ClassName:`
Main block	Program execution	`if __name__ == "__main__":`

Key Principles:

- **Indentation**: Defines code blocks (4 spaces recommended)
- **Comments**: Use # for single line, “ “ ” “ “ ” for multi-line
- **Modularity**: Organize code in functions and classes

Mnemonic

“Structure: Import, Define, Execute”

Question 2(a OR) [3 marks]

Write a Program to reverse a string.

Solution

String Reversal Program:

```
\# Method 1: Using slicing
string = input("Enter a string: ")
reversed\_string = string[::{-}1]
print(f"Reversed: \{reversed\_string\}")

\# Method 2: Using loop
string = input("Enter a string: ")
reversed\_string = ""
for char in string:
    reversed\_string = char + reversed\_string
print(f"Reversed: \{reversed\_string\}")
```

Reversal Methods Table:

Method	Syntax	Example
Slicing	`string[::-1]`	"hello" → *"olleh"*
Loop	Build character by character	Add each char to front
Built-in	`"".join(reversed(string))`	Join reversed sequence

Key Concepts:

- **Slicing**: Most efficient method
- **Concatenation**: Build string character by character
- **Indexing**: Access string positions

Mnemonic

"Reverse: Last Character First"

Question 2(b OR) [4 marks]

Explain Logical Operators with example.

Solution

Python Logical Operators:

Operator	Symbol	Description	Example	Result
AND	`and`	Both conditions true	`True and False`	`False`
OR	`or`	At least one condition true	`True or False`	`True`
NOT	`not`	Opposite of condition	`not True`	`False`

Example Code:

```
a = 10
b = 5

\# AND operator
if a {} 5 and b {} 10:
    print("Both conditions true")

\# OR operator
if a {} 15 or b {} 10:
    print("At least one condition true")

\# NOT operator
if not (a {} 5):
    print("a is not less than 5")
```

Truth Table:

A	B	A and B	A or B	not A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

Key Uses:

- **Complex conditions**: Combine multiple checks
- **Decision making**: Control program flow
- **Boolean logic**: True/False operations

Mnemonic

"AND needs All, OR needs One, NOT reverses"

Question 2(c OR) [7 marks]

Explain different Data Types in Python Programming language

Solution

Python Data Types Classification:

Mermaid Diagram (Code)

```
{Shaded}
{Highlighting}[]
graph TD
    A[Python Data Types] {-{-}{} B[Numeric]}
    A {-{-}{} C[Sequence]}
    A {-{-}{} D[Boolean]}
    A {-{-}{} E[Set]}
    A {-{-}{} F[Dictionary]}
    B {-{-}{} G[int]}
    B {-{-}{} H[float]}
    B {-{-}{} I[complex]}
    C {-{-}{} J[str]}
    C {-{-}{} K[list]}
    C {-{-}{} L[tuple]}
{Highlighting}
{Shaded}
```

Data Types Table:

Type	Example	Description	Mutable
int	`42`	Whole numbers	No
float	`3.14`	Decimal numbers	No
str	`"hello"`	Text data	No
list	`[1,2,3]`	Ordered collection	Yes
tuple	`(1,2,3)`	Ordered immutable	No
dict	`{"a":1}`	Key-value pairs	Yes
bool	`True/False`	Boolean values	No
set	`{1,2,3}`	Unique elements	Yes

Example Code:

```
\# Numeric types
age = 25          \# int
price = 99.99     \# float
complex\_num = 3+4j \# complex

\# Sequence types
name = "Python"          \# string
numbers = [1,2,3,4]      \# list
coordinates = (10,20)    \# tuple

\# Other types
is\_active = True         \# boolean
unique\_items = \{1,2,3\  }\# set
student = \{"name":"John", "age":20\  }\# dict
```

Key Features:

- **Dynamic typing**: No need to declare variable types
- **Type conversion**: Convert between compatible types
- **Built-in functions**: `type()`, `isinstance()` for checking types

Mnemonic

"Python Types: Numbers, Sequences, Collections"

Question 3(a) [3 marks]

What is flow control in Python? Explain with example

Solution

Flow control manages the execution order of program statements using conditional and loop structures.
Flow Control Types Table:

Type	Statement	Purpose	Example
Sequential	Normal execution	Line by line	`print("Hello")`
Selection	if, elif, else	Decision making	`if x > 0:`
Iteration	for, while	Repetition	`for i in range(5):`
Jump	break, continue	Loop control	`break`

Example Code:

```
\# Selection example
age = 18
if age {=} 18:
    print("Adult")
else:
    print("Minor")

\# Iteration example
for i in range(3):
    print(f"Count: \{i\}")
```

Key Concepts:

- **Conditional execution**: Code runs based on conditions
- **Loop structures**: Repeat code blocks
- **Program flow**: Control execution path

Mnemonic

"Flow Control: Decide, Repeat, Jump"

Question 3(b) [4 marks]

Write a program to explain nested if statement.

Solution

Nested If Statement Program:

```
\# Grade calculation using nested if
marks = int(input("Enter marks: "))

if marks {=} 0 and marks {=} 100:
    if marks {=} 90:
        grade = "A+"
    elif marks {=} 80:
        if marks {=} 85:
            grade = "A"
        else:
            grade = "B+"
    elif marks {=} 70:
        grade = "B"
    elif marks {=} 60:
        grade = "C"
    else:
        grade = "F"
    print(f"Grade: \{grade\}")
else:
    print("Invalid marks")
```

Nested Structure Diagram:

```
                    marks input
                         |

                    0{=marks=100 }

              True               False
```