

Subject Name (Gujarati)

4321602 -- Summer 2023

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 માક્સ]

લિસ્ટ શું છે? તેનો પાયથનમાં ઉપયોગ શું છે અને તેની લાક્ષણિકતાઓ લખો.

જવાબ

લિસ્ટ એ એક ordered collection છે જે એક જ variable માં multiple values store કરી શકે છે. લિસ્ટ mutable છે અને duplicate elements ની મંજૂરી આપે છે.

ટેબલ: લિસ્ટ લાક્ષણિકતાઓ

ફીચર	વર્ણન
Ordered	Elements નો કમ નિર્ધારિત હોય છે
Mutable	બનાવ્યા પછી બદલી શકાય છે
Indexed	Index [0,1,2,...] વાપરીને access કરી શકાય
Duplicates	Duplicate values ની મંજૂરી છે

પાયથનમાં ઉપયોગ:

- Data Storage:** સંવધિત items નો સંગ્રહ
- Dynamic Arrays:** Runtime દરમિયાન size બદલી શકાય
- Iteration:** Elements માં આસાનીથી loop કરી શકાય

મેમરી ટ્રીક

“OMID - Ordered, Mutable, Indexed, Duplicates”

પ્રશ્ન 1(બ) [4 માક્સ]

પાયથનમાં String built-in functions સમજાવો.

જવાબ

String built-in functions પાયથન પ્રોગ્રામમાં text data ને efficiently manipulate અને process કરવામાં મદદ કરે છે.
ટેબલ: સામાન્ય String Functions

Function	હેતુ	ઉદાહરણ
upper()	Uppercase માં convert કરે	“hello”.upper() → “HELLO”
lower()	Lowercase માં convert કરે	“WORLD”.lower() → “world”
strip()	Whitespace remove કરે	“ hi ”.strip() → “ hi ”
split()	List માં split કરે	“a,b”.split(“,”) → [‘a’, ‘b’]
replace()	Substring replace કરે	“cat”.replace(‘c’, ‘b’) → “bat”
find()	Substring position શોધે	“hello”.find(‘e’) → 1

મુખ્ય મુદ્દાઓ:

- **Immutable:** Original string અપરિવર્તિત રહે છે
- **Return Values:** Functions નવી strings return કરે છે
- **Case Sensitive:** Functions case ને ધ્યાનમાં રાખે છે

મેમરી ટ્રીક

“ULSR-FR - Upper, Lower, Strip, Replace, Find, Replace”

પ્રશ્ન 1(ક OR) [7 માંકર્સ]

બિલ્ટ-ઇન Dictionary functions ની ઘાસી લખો. Dictionary ના functions અને operations દર્શાવવા માટે પ્રોગ્રામ લખો.

જવાબ

Dictionary એ key-value pairs નો collection છે જે fast lookup અને flexible data organization પ્રદાન કરે છે.
ટેબલ: Dictionary Functions

Function	હેતુ	Return કરે છે
keys()	બધી keys મેળવે	dict_keys object
values()	બધી values મેળવે	dict_values object
items()	Key-value pairs મેળવે	dict_items object
get()	Safe value retrieval	Value અથવા None
pop()	Remove કરીને value return કરે	Removed value
clear()	બધી items remove કરે	None
update()	Dictionaries merge કરે	None

પ્રોગ્રામ ઉદાહરણ:

```
\# Dictionary
student = \{{name}: {John}, {age}: 20, {grade}: {A}\}

# Dictionary operations
print("Keys:", list(student.keys()))
print("Values:", list(student.values()))
print("Items:", list(student.items()))

# Safe access
print("Age:", student.get({age}, {Not found}))

# Update add
student.update(\{{city}: {Mumbai}, {age}: 21\})
print("Updated:", student)

# Remove operations
grade = student.pop({grade})
print("Removed grade:", grade)
```

મુખ્ય ફીચર્સ:

- **Fast Lookup:** O(1) average time complexity
- **Flexible Keys:** Strings, numbers, tuples વાપરી શકાય
- **Dynamic:** કોઈ પણ સમયે items add/remove કરી શકાય

મેમરી ટ્રીક

“KVIGPCU - Keys, Values, Items, Get, Pop, Clear, Update”

પ્રશ્ન 2(અ) [3 માક્સ્]

Tuple ની વ્યાખ્યા લખો અને તે કઈ રીતે પાયથનમાં બનાવાય?

જવાબ

Tuple એ ordered collection છે જે immutable છે (બનાવ્યા પછી બદલી શકતી નથી).

ટેબલ: Tuple Creation Methods

Method	Syntax	ઉદાહરણ
Parentheses	(item1, item2)	(1, 2, 3)
Without Parentheses	item1, item2	1, 2, 3
Single Item	(item,)	(5,)
Empty Tuple	()	()

Code ઉદાહરણો:

```
\# Tuples
coordinates = (10, 20)           # Standard way
colors = {red}, {blue}, {green}   # Parentheses
single = (42,)                  # Single element (comma)
empty = ()                       # Empty tuple
```

મુખ્ય મુદ્દાઓ:

- Immutable:** બનાવ્યા પછી elements બદલી શકતા નથી
- Ordered:** Elements પોતાની position જાળવે છે
- Indexable:** Lists જેવી રીતે index વાપરીને access કરી શકાય

મેમરી ટ્રીક

"IOI - Immutable, Ordered, Indexed"

પ્રશ્ન 2(બ) [4 માક્સ્]

Module ના ફાયદાઓ સમજાવો.

જવાબ

Modules એ Python files છે જેમાં functions, classes, અને variables હોય છે જે બિજા programs માં import કરીને reuse કરી શકાય છે.

ટેબલ: Module ફાયદાઓ

ફાયદો	વર્ણન	લાભ
Reusability	Same code multiple times વાપરી શકાય	Development time બચાવે
Organization	Code ને logical units માં વિભાજિત કરે	Better code structure
Namespace	Naming conflicts ટાળે	Cleaner code
Maintainability	એક જ જગ્યાએ code update કરવું	Easy debugging

લાભો:

- **Code Reuse:** એક વાર લખો, ઘણી વાર વાપરો
- **Modularity:** મોટા programs ને નાના ભાગોમાં તોડો
- **Collaboration:** Multiple developers અલગ modules પર કામ કરી શકે
- **Testing:** Individual modules ને અલગથી test કરી શકાય

ઉદાહરણ Structure:

```
\# math\_utils.py (module)
def add(a, b):
    return a + b

\# main.py (module)
import math\_utils
result = math\_utils.add(5, 3)
```

મેમરી ટ્રીક

“RONM - Reusability, Organization, Namespace, Maintainability”

પ્રશ્ન 2(ક) [7 માંકર્સ]

ઓઝ ઉદાહરણ સાથે user defined package બનાવવા માટેના steps લખો.

જવાબ

Package એ directory છે જેમાં multiple modules હોય છે અને special `__init__.py` file હોય છે.

Package બનાવવાના Steps:

```
graph TD
    A[Package Directory] --> B["__init__.py file"]
    B --> C[Module Files]
    C --> D[Modules Functions]
    D --> E[Package Import]
```

ઉદાહરણ Package Structure:

```
mathtools/
    __init__.py
    basic.py
    advanced.py
```

Step-by-Step Implementation:

Step 1: Directory બનાવો

```
mkdir mathtools
```

Step 2: init.py બનાવો

```
\# mathtools/\_\_init\_\_.py
print("MathTools package loaded")
```

Step 3: basic.py બનાવો

```
\# mathtools/basic.py
def add(a, b):
    return a + b

def subtract(a, b):
    return a {-} b
```

Step 4: advanced.py બનાવો

```
\# mathtools/advanced.py
def power(base, exp):
    return base ** exp

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

Step 5: Package વાપરો

```
\# main.py
import mathtools.basic
from mathtools.advanced import power

result1 = mathtools.basic.add(5, 3)
result2 = power(2, 3)
print(f"Addition: {result1}, Power: {result2}")
```

મુખ્ય આવશ્યકતાઓ:

- **Directory:** Package directory હોવી જોઈએ
- **init.py:** જરૂરી file (ખાલી પણ હોઈ શકે)
- **Modules:** Package અંદર Python files
- **Import Path:** Python ને package path માં મળવું જોઈએ

મેમરી ટ્રીક

“DDMFU - Directory, Dunder-init, Modules, Functions, Use”

પ્રશ્ન 2(અ OR) [3 માંકર્સ]

Tuple અને List વચ્ચેનો તફાવત લખો.

જવાબ

Tuple અને List બંને sequence data types છે પરંતુ behavior અને usage માં મહત્વપૂર્ણ તફાવતો છે.
ટેબલ: Tuple vs List તુલના

કીયર	Tuple	List
Mutability	Immutable (બદલી શકતું નથી)	Mutable (બદલી શકાય છે)
Syntax	(1, 2, 3)	[1, 2, 3]
Performance	વધુ ઝડપી	ધીમું
Methods	મયોડિટ methods	ઘણો methods ઉપલબ્ધ
Use Case	Fixed data	Dynamic data
Memory	ઓછી memory	વધુ memory

Code ઉદાહરણ:

```
\# Tuple {- Immutable}
coordinates = (10, 20)
\# coordinates[0] = 15 \# Error!

\# List {- Mutable }
numbers = [1, 2, 3]
numbers[0] = 10      \#
numbers.append(4)   \# Elements add
```

ક્યારે વાપરવું:

- **Tuple:** Coordinates, database records, function arguments
- **List:** Shopping cart, student grades, dynamic collections

મેમરી ટ્રીક

“TIF-LIM - Tuple Immutable Fixed, List Mutable Dynamic”

પ્રશ્ન 2(બ OR) [4 માંકસ]

પાયથનમાં intra-package reference concept સમજાવો.

જવાબ

Intra-package references package અંદરના modules ને relative imports વાપરીને એકવીજાને import અને use કરવાની મંજૂરી આપે છે.

Import Types:

ટેબલ: Import પકારો

Type	Syntax	Usage
Absolute	from package.module import function	Root થી full path
Relative	from .module import function	Same package અંદર
Parent	from ..module import function	Parent package

Package Structure ઉદાહરણ:

```
calculator/
    __init__.py
    basic.py
    scientific.py
    utils/
        __init__.py
        helpers.py
```

Implementation:

```
\# calculator/basic.py
def add(a, b):
    return a + b

\# calculator/scientific.py
from .basic import add  \# Relative import
from .utils.helpers import validate  \# Sub{-package import}

def advanced\_add(a, b):
    if validate(a) and validate(b):
        return add(a, b)
    return None

\# calculator/utils/helpers.py
def validate(num):
    return isinstance(num, (int, float))
```

લાભો:

- **Clean Code:** ટૂંકા import statements
- **Package Independence:** Packages ને સરળતાથી relocate કરી શકાય
- **Clear Structure:** Package relationships દર્શાવે છે

મેમરી ટ્રીક

``RAP - Relative, Absolute, Parent imports''

પદ્ધતિ 2 (OR) [7 માંકર્ણ]

Module એટલે શું? વર્તુળનું ક્ષેત્રફળ અને પરિધ શોધવા માટે module બનાવવાનો પ્રોગ્રામ લખો. આ module ને પ્રોગ્રામમાં import કરો અને તેમાંથી functions call કરો.

જવાબ

Module એ Python file છે જેમાં functions, classes, અને variables હોય છે જે બીજા programs માં import કરીને વાપરી શકાય છે.

Circle Module (circle.py):

```
\# circle.py {- Circle operations module}
import math

def area(radius):
    """ calculate """
    if radius <= 0:
        return None
    return math.pi * radius * radius

def circumference(radius):
    """ calculate """
```

```

if radius <= 0:
    return None
return 2 * math.pi * radius

def diameter(radius):
    """ calculate """
    if radius <= 0:
        return None
    return 2 * radius

# Module constant
PI = math.pi

```

Main Program (main.py):

```

#!/usr/bin/python
# main.py {- Circle module      }
import circle
from circle import area, circumference

# Method 1: Module name
radius = 5
print("Module name      :")
print(f"    : \{circle.area(radius):.2f\}")
print(f"    : \{circle.circumference(radius):.2f\}")

# Method 2: Direct function import
print("{n}Direct import      :")
print(f"    : \{area(radius):.2f\}")
print(f"    : \{circumference(radius):.2f\}")

# Module constant
print(f"PI    : \{circle.PI:.4f\}")

```

Alternative Import Methods:

```

# functions import
from circle import *

# Alias      import
import circle as c
result = c.area(10)

# Specific function alias      import
from circle import area as circle\_area

```

Module લાભો:

- **Reusability:** Multiple programs માં વાપરી શકાય
- **Organization:** Related functions એકસાથે રાખી શકાય
- **Namespace:** Function name conflicts ટાળી શકાય
- **Testing:** Module functions ને અલગથી test કરી શકાય

Output ઉદાહરણ:

```

Module name      :
    : 78.54
    : 31.42

```

```

Direct import      :
    : 78.54
    : 31.42
PI    : 3.1416

```

મેમરી ટ્રીક

“IRUD - Import, Reuse, Use, Debug”) [7 માફર્સ]

સેટમાંથી કોઈ element કેવી રીતે ઉમેરતું, દૂર કરતું તે લખો. POP remove થી કઈ રીતે અલગ છે તે સમજાવો.

જવાબ

Sets એ unique elements નો unordered collection છે. પાયથન sets ને modify કરવા માટે વિવિધ methods પ્રદાન કરે છે.

ટેબલ: Set Operations

Operation	Method	Syntax	ઉદાહરણ
Add	add()	set.add(element)	s.add(5)
Remove	remove()	set.remove(element)	s.remove(3)
Remove Safe	discard()	set.discard(element)	s.discard(7)
Pop	pop()	set.pop()	s.pop()

Code ઉદાહરણ:

```
\# Set
my_set = \{1, 2, 3, 4\}

\# Element
my_set.add(5)          \# \{1, 2, 3, 4, 5\}

\# Elements
my_set.remove(2)      \# \{1, 3, 4, 5\}
my_set.discard(10)    \# Element      error

\# Pop operation
element = my_set.pop() \# Random element remove
```

POP vs REMOVE તફાવત:

પાસું	pop()	remove()
Target	Random element	Specific element
Parameter	Parameter જરૂરી નથી	Element value જરૂરી
Return	Removed element return કરે	None return કરે
Error	Set empty હોય તો error	Element ન મળે તો error

મુખ્ય મુદ્દાઓ:

- **Random Nature:** pop() unordered nature ને કારણે arbitrary element remove કરે
- **Predictability:** remove() specific known element ને target કરે
- **Error Handling:** KeyError ટાળવા માટે discard() વાપરો

મેમરી ટ્રીક

“PRRE - Pop Random, Remove Exact”

પ્રશ્ન 3(અ) [3 માફર્સ]

પાયથનમાં errors ના પ્રકારો સમજાવો.

જવાબ

Python errors ત્યારે આવે છે જ્યારે code properly execute ન થઈ શકે. Error types સમજવાથી debugging અને robust programs લખવામાં મદદ મળે છે.

ટેબલ: Python Error પ્રકારો

Error Type	વર્ણન	ઉદાહરણ
Syntax Error	Code structure ખોટું	Colon, brackets ગુમ
Runtime Error	Execution દરમિયાન error	Zero થી division
Logical Error	Code run થાય પણ wrong result	ખોટું formula

સામાન્ય ઉદાહરણો:

```
\# Syntax Error  
\# if x { 5 \# Colon }
```



```
\# Runtime Error  
\# result = 10 / 0 \# ZeroDivisionError
```



```
\# Logical Error  
def average(a, b):  
    return a + b / 2 \# (a + b) / 2
```

Error લાક્ષણિકતાઓ:

- Syntax:** Execution પહેલાં detect થાય
- Runtime:** Execution દરમિયાન detect થાય
- Logical:** Automatically detect નથી થતી

મેમરી ટ્રીક

“SRL - Syntax, Runtime, Logical”

પદ્ધતિ 3(બ્ય) [4 માફસ્ટ્]

try except નું structure સમજવો.

જવાબ

Try-except structure runtime errors ને gracefully handle કરે છે, program crashes અટકાવે છે અને user-friendly error messages પ્રદાન કરે છે.

Basic Structure:

```
flowchart LR  
A[try block] --> B[Code execution]  
B --> C{Error ?\}\| D[continue]  
C --> E[except block]\| F[Error handle ]  
F --> G[Program continue]
```

Syntax Structure:

```
try:  
    \# Code error  
    risky\_code()  
except ErrorType:  
    \# Specific error handle  
    handle\_error()  
except:  
    \# error handle  
    handle\_all\_errors()
```

```

finally:
    # execute
    cleanup\_code()

```

ટેબલ: Structure Components

Block	હેતુ	જરૂરી
try	Risky code સમાવે છે	હા
except	Specific errors handle કરે	હા
else	Error ન આવે તો run થાય	નહીં
finally	હંમેશા execute થાય	નહીં

ઉદાહરણ:

```

try:
    num = int(input("      : "))
    result = 100 / num
    print(f"      : \{result\}")
except ValueError:
    print("      format")
except ZeroDivisionError:
    print("Zero      ")
finally:
    print("Operation      ")

```

મેમરી ટ્રીક

“TEEF - Try, Except, Else, Finally”

પ્રશ્ન 3(ક) [7 માંકર્સ]

Marks Result માટે એક function બનાવો જેમાં English અને Maths marks ની બે arguments હોય, જો કોઈપણ argument નું value 0 કરતાં ઓછું હોય તો error generate કરાવો.

જવાબ

Custom error handling data validation ensure કરે છે અને invalid inputs માટે meaningful feedback પ્રદાન કરે છે.
Complete Implementation:

```

# Custom exception class
class InvalidMarksError(Exception):
    """Invalid marks      custom exception"""
    def __init__(self, subject, marks):
        self.subject = subject
        self.marks = marks
        super().__init__(f"      \{subject\} marks: \{marks\}. Marks negative      .")

def marks_result(english, maths):
    """
    English      Maths marks      result calculate

    Args:
        english (float): English subject marks
        maths (float): Mathematics subject marks

    Returns:
        dict: Total, percentage,      grade      result
    """

```

```

Raises:
    InvalidMarksError: Marks negative
    TypeError: Marks numeric
"""

\# Type validation
if not isinstance(english, (int, float)) or not isinstance(maths, (int, float)):
    raise TypeError("Marks numeric values      ")

\# Negative marks validation
if english < 0:
    raise InvalidMarksError("English", english)

if maths < 0:
    raise InvalidMarksError("Mathematics", maths)

\# Marks range validation (0{-100})
if english > 100:
    raise InvalidMarksError("English", english)

if maths > 100:
    raise InvalidMarksError("Mathematics", maths)

\# Results calculate
total = english + maths
percentage = (total / 200) * 100

\# Grade
if percentage == 90:
    grade = {A+}
elif percentage == 80:
    grade = {A}
elif percentage == 70:
    grade = {B}
elif percentage == 60:
    grade = {C}
elif percentage == 50:
    grade = {D}
else:
    grade = {F}

return \{
    {english}: english,
    {maths}: maths,
    {total}: total,
    {percentage}: round(percentage, 2),
    {grade}: grade,
    {status}: {Pass} if percentage >= 50 else {Fail}
\}

\# Usage examples with error handling
def main():
    """Marks\_result function demonstrate      main function"""

    test\_cases = [
        (85, 92),      \# Valid marks
        (-10, 85),    \# Negative English
        (75, -5),     \# Negative Maths
        (105, 80),    \# Marks > 100
        ("80", 90),   \# String input
    ]

```

```

for i, (eng, math) in enumerate(test\_cases, 1):
    print(f"\nTest Case {i}: English={eng}, Maths={math}")
    try:
        result = marks\_result(eng, math)
        print(f" : {result}")

    except InvalidMarksError as e:
        print(f"Custom Error: {e}")

    except TypeError as e:
        print(f"Type Error: {e}")

    except Exception as e:
        print(f"Unexpected Error: {e}")

# Interactive function
def get\_student\_result():
    """Student marks      interactive function"""

    while True:
        try:
            print("\n--- Student Result Calculator ---")
            english = float(input("English marks      (0{-100}: "))
            maths = float(input("Maths marks      (0{-100}: "))

            result = marks\_result(english, maths)

            print(f"\n--- --- ---")
            print(f"English: {result[english]}")
            print(f"Mathematics: {result[maths]}")
            print(f"Total: {result[total]}/200")
            print(f"Percentage: {result[percentage]}%")
            print(f"Grade: {result[grade]}")
            print(f"Status: {result[status]}")

            break

        except InvalidMarksError as e:
            print(f"Error: {e}")
            print("      valid marks      (0{-100})")

        except ValueError:
            print("Error:      numeric values      ")

        except KeyboardInterrupt:
            print("\nProgram user      terminate      ")
            break

    if __name__ == "__main__":
        main()
        get\_student\_result()

```

મુખ્ય ફુચક્સ:

- **Custom Exception:** InvalidMarksError specific validation માટે
- **Multiple Validations:** Negative, type, અને range checks
- **Comprehensive Results:** Total, percentage, grade calculation
- **User-Friendly:** Interactive input સાથે error handling

Error Handling લાભો:

- **Data Integrity:** Valid input data ensure કરે
- **User Experience:** Clear error messages
- **Program Stability:** Crashes અટકાડે
- **Debugging:** Issues identify કરવામાં સરળ

પ્રશ્ન 3(અ OR) [3 માફર્સ]

પાયથનમાં કોઈપણ પાંચ built-in exceptions ની યાદી લખો.

જવાબ

Built-in exceptions એ predefined error types છે જે Python specific error conditions દરમિયાન raise કરે છે.
ટેબલ: સામાન્ય Built-in Exceptions

Exception	કારણ	ઉદાહરણ
ValueError	Operation માટે invalid value	int("abc")
TypeError	ખોટો data type	"5" + 5
IndexError	Index range બહાર	list[10] 5-item list માટે
KeyError	Dictionary key ન મળે	dict["missing_key"]
ZeroDivisionError	Zero થી division	10 / 0

Code ઉદાહરણો:

```
\# ValueError
try:
    number = int("hello")  # Int    convert
except ValueError:
    print("    number format")

\# TypeError
try:
    result = "text" + 42  # String    int add
except TypeError:
    print("Type mismatch")

\# IndexError
try:
    mylist = [1, 2, 3]
    print(mylist[5])      # Index 5
except IndexError:
    print("Index range    ")
```

Additional સામાન્ય Exceptions:

- **FileNotFoundException:** File અસ્તિત્વમાં નથી
- **AttributeError:** Object માં attribute નથી
- **ImportError:** Module import ન થઈ શકે

પ્રશ્ન 3(બ OR) [4 માફર્સ]

Finally પર મુદ્દાઓ લખો અને ઉદાહરણ સાથે સમજાવો.

જવાબ

finally block એ special block છે જે exception આવે કે ન આવે તેની પરવા કર્યા વિના હંમેશા execute થાય છે.
ટેબ્લાનું: Finally Block લાક્ષણિકતાઓ

ફીચર	વર્ણન
હંમેશા Execute	Exception આવે તો પણ run થાય
Cleanup Code	Resource cleanup માટે perfect
Try/Except પછી	Try અને except blocks પછી execute
Skip ન થઈ શકે	Return statements પણ skip ન કરી શકે

મુખ્ય મુદ્દાઓ:

- **Guaranteed Execution:** બધા scenarios માં run થાય
- **Resource Management:** Files, database connections બંધ કરે
- **Cleanup Operations:** Memory free કરે, variables reset કરે
- **Return સાથે પણ:** Function return પહેલાં execute થાય

ઉદાહરણ પ્રોગ્રામ:

```
def file\_operations(filename):
    """File operations finally block demonstrate"""
    file\_handle = None

    try:
        print("File      ...")
        file\_handle = open(filename, {r})

        print("File content      ...")
        content = file\_handle.read()

        # Potential error simulate
        if len(content) == 0:
            raise ValueError("File      ")

        print(f"File content: \{content\}")
        return content

    except FileNotFoundError:
        print("Error: File      ")
        return None

    except ValueError as e:
        print(f"Error: \{e\}")
        return None

    finally:
        print("Finally block execute      ...")
        if file\_handle:
            file\_handle.close()
            print("File      ")
        else:
            print("      file      ")
        print("Cleanup      ")
```

Function ટેસ્ટ કરો:

```
print("== 1: Valid file ==")
result1 = file\_operations("test.txt")

print("{n}== 2: Non{-existent file ==}")
result2 = file\_operations("missing.txt")
```

Output ઉદાહરણ:

```
== 1: Valid file ==
File      ...
File content      ...
File content: Hello World
Finally block execute      ...
File
Cleanup

== 2: Non-existent file ==
File      ...
Error: File
```

```
Finally block execute      ...
    file
Cleanup
```

મેમરી ટ્રીક

“ARGC - Always Runs, Resource Cleanup”

પ્રશ્ન 3(ક OR) [7 માંકર્સ]

Finally clause સાથે Divide by Zero Exception catch કરવા માટેનો પ્રોગ્રામ લખો.

જવાબ

Divide by zero exception handling finally clause સાથે proper error management અને resource cleanup demonstrate કરે છે.

સંપૂર્ણ પ્રોગ્રામ:

```
import sys
import logging

# Logging configure
logging.basicConfig(level=logging.INFO, format='%(asctime)s [- %(levelname)s [- %(message)s')

class DivisionCalculator:
    """Divide by zero exception handling    calculator class"""

    def __init__(self):
        self.calculation_count = 0
        self.error_count = 0

    def safe_divide(self, dividend, divisor):
        """
        Exception handling    division

        Args:
            dividend (float):
            divisor (float):

        Returns:
            float or None: Division           error      None
        """
        operation_id = self.calculation_count + 1

        try:
            print(f"{operation_id} Operation : {dividend} / {divisor}")
            print(f"Division operation : {dividend} / {divisor}")

            # Type validation
            if not isinstance(dividend, (int, float)) or not isinstance(divisor, (int, float)):
                raise TypeError(" arguments numeric ")

            # Division
            result = dividend / divisor

            print(f" : {dividend} / {divisor} = {result}")
            logging.info(f"Division : {result}")

        except Exception as e:
            self.error_count += 1
            print(f"Error occurred: {e}")
            logging.error(str(e))

    def get_error_count(self):
        return self.error_count
```

```

        return result

    except ZeroDivisionError:
        error\_msg = f"Error: \{dividend\} zero !"
        print(error\_msg)
        logging.error(error\_msg)
        self.error\_count += 1
        return None

    except TypeError as e:
        error\_msg = f"Type Error: \{e\}"
        print(error\_msg)
        logging.error(error\_msg)
        self.error\_count += 1
        return None

    except Exception as e:
        error\_msg = f"Unexpected error: \{e\}"
        print(error\_msg)
        logging.error(error\_msg)
        self.error\_count += 1
        return None

    finally:
        \# execute {- cleanup logging}
        self.calculation\_count += 1
        print(f"Operation \{operation\_id\} ")
        print(f" operations: \{self.calculation\_count\}")
        print(f" errors: \{self.error\_count\}")
        logging.info(f"Operation \{operation\_id\} finalized")

        \# Resource cleanup simulation
        if hasattr(self, {temp\_data}):
            delattr(self, {temp\_data})
            print("Temporary data ")

def interactive\_calculator():
    """Interactive division calculator"""

    calc = DivisionCalculator()
    print("== Interactive Division Calculator ==")
    print("Program exit      {quit }")

    while True:
        try:
            print("{n}" + "="*40)

            \# Dividend
            dividend\_input = input("    (dividend) : ")
            if dividend\_input.lower() == {quit}:
                break

            dividend = float(dividend\_input)

            \# Divisor
            divisor\_input = input("    (divisor) : ")
            if divisor\_input.lower() == {quit}:
                break

            divisor = float(divisor\_input)

```

```

    \# Calculation
    result = calc.safe\divide(dividend, divisor)

    if result is not None:
        print(f"      : \{dividend\}  \{divisor\} = \{result\}")
    else:
        print(" Operation   ")

except ValueError:
    print("Error:      valid numeric values      ")
    calc.error\_count += 1

except KeyboardInterrupt:
    print("{nn}Program user      interrupt      ")
    break

finally:
    \#     iteration  final cleanup
    if {dividend\_input} in locals():
        del dividend\_input
    if {divisor\_input} in locals():
        del divisor\_input
    print("Input variables      ")

def test\_division\_cases():
    """  division scenarios      """

    print("==== Division Cases      ====")
    calc = DivisionCalculator()

    test\_cases = [
        (10, 2),      \# Normal division
        (15, 0),      \# Divide by zero
        (7.5, 2.5),   \# Float division
        ({-}20, 4),   \# Negative numbers
        (0, 5),       \# Zero dividend
        ("10", 2),    \# String input
        (100, 0.0),   \# Zero as float
    ]

    for dividend, divisor in test\_cases:
        result = calc.safe\divide(dividend, divisor)

    \# Final statistics
    print(f"\n====      ====")
    print(f"  operations  : \{calc.calculation\_count\}")
    print(f"  errors    : \{calc.error\_count\}")
    print(f"  : \{((calc.calculation\_count {-} calc.error\_count) / calc.calculation\_count * 100):.2f}\%")

if __name__ == "__main__":
    \# Test cases
    test\_division\_cases()

    \# Interactive calculator
    interactive\calculator()

```

મુખ્ય ફીચર્સ:

- **Comprehensive Error Handling:** Multiple exception types
- **Finally Clause:** Cleanup માટે હમેશા execute થાય
- **Logging:** Operations અને errors track કરે
- **Interactive Mode:** User-friendly interface
- **Statistics:** Operation success tracking

પ્રશ્ન 4(અ) [3 માંકર્સ]

File Handling શું છે? File Handling Operations ની ચારી બનાવો.

જવાબ

File Handling એ computer storage devices પર stored files સાથે કામ કરવાની process છે data read, write, અને manipulate કરવા માટે.

ટેબલ: File Handling Operations

Operation	હેતુ	Method
Open	Operations માટે file access કરે	open()
Read	File માંથી content retrieve કરે	read(), readline()
Write	File માં content add કરે	write(), writelines()
Close	File resources release કરે	close()
Seek	File pointer move કરે	seek()
Tell	Current position મેળવો	tell()

સામાન્ય Use Cases:

- **Data Storage:** Program data permanently save કરે
- **Configuration:** Files માંથી settings read કરે
- **Logging:** Program activities record કરે
- **Import/Export:** અન્ય programs સાથે data exchange કરે

Basic ઉદાહરણ:

```
\# Basic file operations
file = open("data.txt", "w")      # Open
file.write("Hello World")        # Write
file.close()                     # Close
```

મેમરી ટ્રીક

પ્રશ્ન 4(બ) [4 માંકર્સ]

Object Serialization સમજાવો.

જવાબ

Object Serialization એ Python objects ને એવા format માં convert કરવાની process છે જે files માં store કરી શકાય અથવા networks પર transmit કરી શકાય.

ટેબલ: Serialization Methods

Method	Module	હેતુ	File Type
Pickle	pickle	Python objects	Binary
JSON	json	Web-compatible data	Text
CSV	csv	Tabular data	Text
XML	xml	Structured documents	Text

Pickle ઉદાહરણ:

```
import pickle

# Serialization ( )
data = {{name}: {John}, {age}: 25, {scores}: [85, 92, 78]}

with open({data.pkl}, {wb}) as file:
    pickle.dump(data, file)

# Deserialization ( )
with open({data.pkl}, {rb}) as file:
    loaded_data = pickle.load(file)
    print(loaded_data)
```

લાભો:

- **Persistence:** Objects permanently store કરે
- **Data Transfer:** Programs દ્વારા objects send કરે
- **Caching:** Processed results save કરે
- **Backup:** Object snapshots બનાવે

મર્યાદાઓ:

- **Python Specific:** Pickle માત્ર Python સાથે કામ કરે
- **Security Risk:** Untrusted pickle files load ન કરો
- **Version Compatibility:** અલગ Python versions માં issues આવી શકે

મેમરી ટ્રીક

“SPDT - Store, Persist, Data Transfer”

પ્રશ્ન 4(ક) [7 માંકસી]

ફાઇલમાં ના તમામ સ્વરોની ગણતરી કરવા માટેનો પ્રોગ્રામ લખો.

જવાબ

Vowel counting program file reading અને text processing comprehensive error handling સાથે demonstrate કરે છે.

સંપૂર્ણ પ્રોગ્રામ:

```
import os
import string
from collections import Counter

class VowelCounter:
    """Text files    vowels count        class"""

    def __init__(self):
        self.vowels = set({aeiouAEIOU})
        self.total_files_processed = 0
        self.total_vowels_found = 0

    def count_vowels_in_text(self, text):
        """
            text    vowels count

        Args:
            text (str): Analyze      text

        Returns:
        """

        text    vowels count
```

```

        dict: Vowel counts      statistics
"""

vowel\_counts = \{vowel.lower(): 0 for vowel in {aeiou}\}
total\_vowels = 0
total\_characters = 0

for char in text:
    if char.isalpha():
        total\_characters += 1
        if char.lower() in vowel\_counts:
            vowel\_counts[char.lower()] += 1
            total\_vowels += 1

return \{
    {vowel\_counts}: vowel\_counts,
    {total\_vowels}: total\_vowels,
    {total\_characters}: total\_characters,
    {vowel\_percentage}: (total\_vowels / total\_characters * 100) if total\_characters > 0 else 0
\}

def count\_vowels\_in\_file(self, filename):
"""
Specific file    vowels count

Args:
    filename (str): File    path

Returns:
    dict or None: Vowel analysis results
"""

try:
    print(f"\n{---}{---}{---} File process : }\{filename\} {---}{---}{---}")

    #     file exists
    if not os.path.exists(filename):
        raise FileNotFoundError(f"File {}\\{filename}\\{}")

    #     file (directory)
    if not os.path.isfile(filename):
        raise ValueError(f"{}\\{filename}\\{} file   ")

    # File content read
    with open(filename, {r}, encoding={utf{-}8}) as file:
        content = file.read()

    print(f"File size: \{len(content)\} characters")

    if not content.strip():
        print("Warning: File      ")
        return None

    # Vowels count
    results = self.count\_vowels\_in\_text(content)

    # Results display
    print(f"  characters ( letters): \{results[{total\_characters}]\}")
    print(f"  vowels   : \{results[{total\_vowels}]\}")
    print(f"Vowel percentage: \{results[{vowel\_percentage}]\:.2f\}\%")

    print("\n      vowel counts:")
    for vowel, count in results[{vowel\_counts}].items():

```

```

percentage = (count / results[total\_vowels]) * 100 if results[total\_vowels] > 0
print(f" \{vowel.upper()\}: \{count\} (\{percentage:.1f\}\%)")

# Statistics update
self.total\_files\_processed += 1
self.total\_vowels\_found += results[total\_vowels]

return results

except FileNotFoundError as e:
    print(f"Error: \{e\}")
    return None

except PermissionError:
    print(f"Error: File \{filename\} has permission issue")
    return None

except UnicodeDecodeError:
    print(f"Error: File \{filename\} has decoding issue")
    return None

except Exception as e:
    print(f"Unexpected error: \{e\}")
    return None

finally:
    print(f"File processing : \{filename\}")

def create\_sample\_file(self, filename="sample.txt"):
    """Testing sample file"""
    sample\_content = """Python programming language . .
Python syntax libraries .
Python web development, data science, automation .
file vowels : a, e, i, o, u.
UPPER CASE VOWELS: A, E, I, O, U."""

try:
    with open(filename, "w", encoding="utf-8") as file:
        file.write(sample\_content)
    print(f"Sample file \{filename\} created")
    return True
except Exception as e:
    print(f"Sample file creation error: \{e\}")
    return False

def batch\_process\_files(self, file\_list):
    """Multiple files process"""
    print("== Batch Processing Files ==")

    all\_results = []

    for filename in file\_list:
        result = self.count\_vowels\_in\_file(filename)
        if result:
            all\_results.append((filename, result))

    # Summary statistics
    if all\_results:
        print(f"\n== Batch Processing Summary ==")
        print(f"Files processed : {len(all\_results)}")

```

```

total\_vowels = sum(result[{:total\_vowels}] for _, result in all\_results)
total\_chars = sum(result[{:total\_characters}] for _, result in all\_results)

print(f"  files      vowels: \{total\_vowels\}")
print(f"  files      characters: \{total\_chars\}")
print(f"  vowel percentage: \{(total\_vowels/total\_chars*100):.2f\}\%")

def interactive\_vowel\_counter():
    """Interactive vowel counter program"""

    counter = VowelCounter()

    while True:
        print("{n}" + "*50)
        print("VOWEL COUNTER PROGRAM")
        print("*50)
        print("1.      file      vowels count      ")
        print("2. Sample file      vowels count      ")
        print("3. Text      ")
        print("4. Multiple files process      ")
        print("5. Exit")

    try:
        choice = input("{n}          (1{-5}): ".strip())

        if choice == {1}:
            filename = input("Filename      : ").strip()
            counter.count\_vowels\_in\_file(filename)

        elif choice == {2}:
            filename = input("Sample      filename      (default: sample.txt): ").strip()
            if not filename:
                filename = "sample.txt"

            if counter.create\_sample\_file(filename):
                counter.count\_vowels\_in\_file(filename)

        elif choice == {3}:
            text = input("Analyze      text      : ")
            if text.strip():
                result = counter.count\_vowels\_in\_text(text)
                print(f"{n}      text      vowel analysis:")
                print(f"  vowels: \{result[{:total\_vowels}]\}")
                print(f"Vowel percentage: \{result[{:vowel\_percentage}]:.2f\}\%")
                for vowel, count in result[{:vowel\_counts}].items():
                    if count > 0:
                        print(f"  \{vowel.upper()\}: \{count\}")

            else:
                print("  text      ")

        elif choice == {4}:
            files\_input = input("Commas      filenames      : ")
            file\_list = [f.strip() for f in files\_input.split({,}) if f.strip()]
            if file\_list:
                counter.batch\_process\_files(file\_list)
            else:
                print("  files specify      ")

        elif choice == {5}:
            print("Vowel Counter      !")
            break

```

```

        else:
            print("      .      1{-5      . "})

    except KeyboardInterrupt:
        print("{nn}Program interrupt      .      !")
        break
    except Exception as e:
        print(f"Error: \{e\}")

if __name__ == "__main__":
    interactive_vowel_counter()

```

પ્રોગ્રામ ફીચર્સ:

- **File Validation:** File existence અને permissions check કરે
- **Error Handling:** Comprehensive exception management
- **Multiple Modes:** File input, text input, batch processing
- **Statistics:** Individual અને overall vowel counts
- **Interactive Interface:** User-friendly menu system

Output ઉદાહરણ:

```

--- File process      : sample.txt ---
File size: 245 characters
  characters (   letters): 195
  vowels      : 78
Vowel percentage: 40.00%

  vowel counts:
A: 15 (19.2%)
E: 20 (25.6%)
I: 12 (15.4%)
O: 18 (23.1%)
U: 13 (16.7%)

```

મેમરી ટ્રીક

“FVESI - File Validation, Vowel Extraction, Statistics, Interactive”

પ્રશ્ન 4(અ OR) [3 માંકસ્ય]

ફાઇલ કેવી રીતે open અને close કરવી? તેના માટે syntax પણ આપો.

જવાબ

File opening અને closing Python માં file handling ના fundamental operations છે specific syntax અને modes સાથે.

ટેબલ: File Opening Modes

Mode	હેતુ	વર્ણન
'r'	Read	Existing file read કરે (default)
'w'	Write	નવી file બનાવે અથવા existing ને overwrite કરે
'a'	Append	Existing file ના અંતે add કરે
'r+'	Read/Write	Existing file read અને write કરે

Syntax ઉદાહરણો:

```
\# Files opening
file = open("filename.txt", "r")          \# Read mode
file = open("data.txt", "w")              \# Write mode
file = open("log.txt", "a")              \# Append mode

\# Files closing
file.close()                           \# Manual closing

\# {with statement    automatic closing}
with open("filename.txt", "r") as file:
    content = file.read()
\#   file automatically close
```

Best Practices:

- હંમેશા Close કરો: Resource leaks અટકાવો
- `with` વાપરા: Automatic file closing
- Mode Specify કરો: File mode વિશે explicit રહો
- Errors Handle કરો: File operations માટે try-except વાપરો

મેમરી ટ્રીક

“ORWA - Open, Read, Write, Append modes”

પ્રશ્ન 4(બ OR) [4 માંકર્સ]

Text file અને Binary file નો તફાવત લખો.

જવાબ

Text અને Binary files અલગ formats માં data store કરે છે, Python programming માં અલગ handling approaches જરૂરી છે.

ટેબલ: Text vs Binary Files તુલના

પાસું	Text File	Binary File
Content	Human-readable characters	Machine-readable bytes
Mode	`r', `w', `a'	`rb', `wb', `ab'
Encoding	UTF-8, ASCII encoding	કોઈ encoding નથી
Size	Encoding ને કારણે મોટી	નાની, compact
ઉદાહરણો	.txt, .py, .html	.jpg, .exe, .pkl
Editing	કોઈપણ text editor	Specialized software

Code ઉદાહરણો:

```
\# Text File Operations
with open("text\_file.txt", "w") as file:
    file.write("Hello World!")

with open("text\_file.txt", "r") as file:
    content = file.read()
    print(content)  \# Output: Hello World!

\# Binary File Operations
import pickle

data = [1, 2, 3, 4, 5]

\# Binary write
with open("binary\_file.pkl", "wb") as file:
    pickle.dump(data, file)

\# Binary read
with open("binary\_file.pkl", "rb") as file:
    loaded\_data = pickle.load(file)
    print(loaded\_data)  \# Output: [1, 2, 3, 4, 5]
```

ક્યારે વાપરતું:

- **Text Files:** Configuration, logs, source code, documentation
- **Binary Files:** Images, videos, executables, serialized objects

મુખ્ય તકાવતો:

- **Portability:** Text files systems વર્ષે વધુ portable
- **Efficiency:** Binary files space અને time efficient
- **Human Readable:** Text files સીધી રીતે જોઈ શકાય

મેમરી ટ્રીક

“TCEB - Text Character Encoding Bigger, Binary Compact Efficient”

પ્રશ્ન 4(ક OR) [7 માંકર્સ]

સીટ નંબર અને નામ store કરવા માટે binary file બનાવવાનો પ્રોગ્રામ લખો. કોઈપણ સીટ નંબર શોધો અને જો સીટ નંબર મળે તો નામ display કરો અન્યથા “Seat no not found” display કરો.

જવાબ

Student record management માટે binary file program pickle serialization સાથે search functionality દર્શાવે છે.
સંપૂર્ણ પ્રોગ્રામ:

```
import pickle
import os
from typing import Dict, Optional

class StudentRecordManager:
    """Binary file student records manage"""
    def __init__(self, filename="students.pkl"):
        self.filename = filename
        self.records = {}
        self.load_records()

    def load_records(self):
```

```

"""Binary file      existing records load """
try:
    if os.path.exists(self.filename):
        with open(self.filename, {rb}) as file:
            self.records = pickle.load(file)
        print(f"\{len(self.records)\} existing records load    ")
    else:
        print("   existing record file      .          .")
        self.records = \{\}
except Exception as e:
    print(f"Records load      error: \{e\}")
    self.records = \{\}

def save\_records(self):
    """Records binary file  save """
    try:
        with open(self.filename, {wb}) as file:
            pickle.dump(self.records, file)
        print(f"Records      \{self.filename\}  save    ")
        return True
    except Exception as e:
        print(f"Records save      error: \{e\}")
        return False

def add\_student(self, seat\_no: int, name: str):
    """  student record add """
    try:
        if not isinstance(seat\_no, int) or seat\_no \{= 0:
            raise ValueError("Seat number positive integer      ")

        if not name or not name.strip():
            raise ValueError("Name      ")

        name = name.strip().title()

        if seat\_no in self.records:
            print(f"Warning: Seat \{seat\_no\}      \{self.records[seat\_no]\}\n")
            overwrite = input("      overwrite      ? (y/n): ").lower()
            if overwrite != {y}:
                print("Record add      ")
                return False

            self.records[seat\_no] = name
            self.save\_records()
            print(f"Student add      : Seat \{seat\_no\} \{- }\{name\}")
            return True

        except ValueError as e:
            print(f"Error: \{e\}")
            return False
    except Exception as e:
        print(f"Unexpected error: \{e\}")
        return False

def search\_student(self, seat\_no: int):
    """Seat number      student """
    try:
        if not isinstance(seat\_no, int):
            raise ValueError("Seat number integer      ")

        if seat\_no in self.records:

```

```

        name = self.records[seat\_no]
        print(f" : Seat \{seat\_no\} {- }\{\name\}")
        return name
    else:
        print("Seat no not found")
        return None

except ValueError as e:
    print(f"Error: \{e\}")
    return None
except Exception as e:
    print(f"Unexpected error: \{e\}")
    return None

def display\_all\_records(self):
    """ student records display """
    if not self.records:
        print(" records ")
        return

    print(f"\n{{'-'*25}} Student Records ( { }\\{len(self.records)} ) {{'-'*25}}")
    print("Seat No. | Name")
    print("-" * 25)

    # Better display seat number sort
    for seat\_no in sorted(self.records.keys()):
        print(f"\{seat\_no:8\} | \{self.records[seat\_no]\}")

def delete\_student(self, seat\_no: int):
    """Student record delete """
    try:
        if seat\_no in self.records:
            name = self.records[seat\_no]
            del self.records[seat\_no]
            self.save\_records()
            print(f"Delete : Seat \{seat\_no\} {- }\{\name\}")
            return True
        else:
            print("Seat no not found")
            return False
    except Exception as e:
        print(f"Record delete error: \{e\}")
        return False

def get\_statistics(self):
    """Record statistics """
    if not self.records:
        print("Statistics records ")
        return

    seat\_numbers = list(self.records.keys())
    print(f"\n{{'-'*25}} {{'-'*25} {{'-'*25}}")
    print(f" students: \{len(self.records)}")
    print(f" seat number: \{\min(seat\_numbers)\}")
    print(f" seat number: \{\max(seat\_numbers)\}")
    print(f"File size: \{os.path.getsize(self.filename) if os.path.exists(self.filename) else 0\} b")

def add\_sample\_data(manager):
    """Testing sample student data add """
    sample\_students = [
        (101, "Alice Johnson"),

```



```

seat\_no = int(input("Delete      seat number      : "))
confirm = input(f"      seat \{seat\_no\} delete      ? (y/n): ")
if confirm.lower() == {y}:
    manager.delete\_student(seat\_no)
except ValueError:
    print("Error:      valid seat number      ")

elif choice == {5}:
    confirm = input(" sample data add . Continue ? (y/n): ")
    if confirm.lower() == {y}:
        add\_sample\_data(manager)

elif choice == {6}:
    manager.get\_statistics()

elif choice == {7}:
    print("Student Record System      !")
    break

else:
    print("      .      1{-7}      ."))

except KeyboardInterrupt:
    print("{nn}Program interrupt      .      !")
    break
except Exception as e:
    print(f"Error: \{e\}")

def quick\_demo():
    """Program quick demonstration"""
    print("{n}{-{-}{-} QUICK DEMO {-}{-}{-}}")

    # Demo file manager
    demo\_manager = StudentRecordManager("demo\_students.pkl")

    # students add
    demo\_students = [
        (101, "John Doe"),
        (102, "Jane Smith"),
        (103, "Mike Johnson")
    ]

    print("Demo students add      .")
    for seat\_no, name in demo\_students:
        demo\_manager.add\_student(seat\_no, name)

    print("{n}Exists student search:")
    demo\_manager.search\_student(102)

    print("{n}Non-existing student search:")
    demo\_manager.search\_student(999)

    print("{n} records:")
    demo\_manager.display\_all\_records()

if __name__ == "__main__":
    # User demo full program
    mode = input("(d)elete (f)ull program? (d/f): ").lower()

    if mode == {d}:
        quick\ demo()

```

```
else:  
    main()
```

પ્રોગ્રામ ફીચર્સ:

- **Binary Storage:** Efficient data storage માટે pickle વાપરે
- **Search Functionality:** ઝડપી seat number lookup
- **Error Handling:** Comprehensive input validation
- **CRUD Operations:** Create, Read, Update, Delete records
- **Statistics:** File અને record information
- **Interactive Menu:** User-friendly interface

Sample Output:

```
seat number      : 102  
: Seat 102 - Jane Smith
```

```
seat number      : 999  
Seat no not found
```

મેમરી ટ્રીક

“BSECH - Binary Storage, Search Efficiently, CRUD Handling”

પ્રશ્ન 5(અ) [3 માંકર્સ]

Turtle શું છે અને તેનો ઉપયોગ objects દોરવા માટે કેવી રીતે થાય છે?

જવાબ

Turtle એ Python graphics module છે જે virtual drawing canvas પ્રદાન કરે છે turtle cursor સાથે programmatically graphics બનાવવા માટે.

ટેબલ: Turtle Basics

Component	વર્ણન	હેતુ
Canvas	Drawing surface	ગ્રાફિક્સ દેખાય તે વિસ્તાર
Turtle	Drawing cursor	ખરો છે અને lines દોરે છે
Pen	Drawing tool	Line appearance control કરે
Commands	Movement functions	Turtle actions control કરે

Basic Drawing Concept:

```
import turtle

# Screen turtle
screen = turtle.Screen()
pen = turtle.Turtle()

# square
for i in range(4):
    pen.forward(100)      # 100 units
    pen.right(90)        # 90 degrees

screen.exitonclick()    # Click
```

મુખ્ય ફીચર્સ:

- **Visual Programming:** તરત જ રીસલ્ઝ જોઈ શકાય
- **Educational:** Programming concepts શીખવા માટે સરસ
- **Interactive:** Real-time drawing feedback
- **Simple Syntax:** જાંટિલ ગ્રાફિક્સ માટે સરળ commands

સામાન્ય ઉપયોગો:

- **Geometric Shapes:** વર્ગ, વર્તુળ, બહુકોણ
- **Patterns:** Fractals, spirals, designs
- **Educational Graphics:** Geometry અને programming શીખવવા

મેમરી ટ્રીક

"CPTT - Canvas, Pen, Turtle, Teaching tool"

પ્રશ્ન 5(બ) [4 માંકસ]

Turtle ને બીજી સ્થિતિમાં ખસેડવાની વિવિધ રીતો સમજાવો.

જવાબ

Turtle drawing canvas પર positioning અને navigation માટે multiple movement methods પ્રદાન કરે છે.

ટેબલ: Turtle Movement Methods

Method	હેતુ	Pen State	ઉદાહરણ
forward(distance)	આગળ જાઓ	Line દોરે	forward(100)
backward(distance)	પાછળ જાઓ	Line દોરે	backward(50)
goto(x, y)	Coordinates પર જાઓ	Line દોરે	goto(100, 50)
penup()	Pen ઉઠાવો	Drawing નહીં	penup()
pendown()	Pen નીચે કરો	Line દોરે	pendown()
setx(x)	X coordinate set કરો	Line દોરે	setx(200)
sety(y)	Y coordinate set કરો	Line દોરે	sety(150)

Movement ઉદાહરણો:

```
import turtle

pen = turtle.Turtle()
pen.speed(3)

# Method 1: Forward/Backward movement
pen.forward(100)
pen.backward(50)

# Method 2: Drawing    direct positioning
pen.goto(200, 100)

# Method 3: Drawing    move
pen.penup()
pen.goto(-100, -100)
pen.pendown()

# Method 4: Coordinates    set
pen.setx(0)
pen.sety(0)
```

Rotation Methods:

- **right(angle):** Clockwise turn
- **left(angle):** Counterclockwise turn
- **setheading(angle):** Absolute direction set કરો

Position Control:

- **Drawing Mode:** Pen down, trail છાડે
- **Moving Mode:** Pen up, કોઈ trail નહોં
- **Coordinate System:** Center (0,0), positive Y ઊપર

મેમરી ટ્રીક

“FGPRS - Forward, Goto, Penup, Rotate, Set coordinates”

પ્રશ્ન 5(ક) [7 માંકસ્]

Turtle માં loops કેવી રીતે ઉપયોગી થઈ શકે તે ઉદાહરણ સાથે સમજાવો.

જવાબ

Turtle graphics માં loops repetitive patterns, complex shapes, અને geometric designs માટે efficient code enable કરે છે.

Loops ના Turtle માં ફાયદા:

ટેબલ: Loop Applications

Loop Type	Use Case	ઉદાહરણ Pattern
For Loop	Fixed repetitions	Regular polygons
While Loop	Conditional drawing	Spirals
Nested Loops	Complex patterns	Grids, fractals
Range Loop	Incremental changes	Color gradients

સંપૂર્ણ ઉદાહરણ પ્રોગ્રામ:

```
import turtle
import random
import math

def setup\_screen():
    """Turtle screen setup"""
    screen = turtle.Screen()
    screen.bgcolor("black")
    screen.title("Turtle Graphics with Loops")
    screen.setup(800, 600)
    return screen

def draw\_polygon(sides, size, pen):
    """For loop      regular polygon """
    angle = 360 / sides

    for i in range(sides):
        pen.forward(size)
        pen.right(angle)

def draw\_spiral(pen):
    """While loop      spiral """
    pen.color("cyan")
    pen.speed(10)

    distance = 1
    while distance < 100:
        pen.forward(distance)
        pen.right(91)
        distance += 2

def draw\_flower\_pattern(pen):
    """Nested loops      flower """
    pen.color("red")
    pen.speed(0)

    # Petals      outer loop
    for petal in range(36):
        pen.color(random.choice(["red", "pink", "yellow", "orange"]))

        #      petal shape      inner loop
        for side in range(4):
            pen.forward(50)
            pen.right(90)

        pen.right(10)  #      petal      rotate

def draw\_colorful\_squares(pen):
    """Colors      sizes      squares """
    colors = ["red", "blue", "green", "yellow", "purple", "orange"]
    pen.speed(0)

    for i in range(50):
        pen.color(colors[i \% len(colors)])
        pen.forward(i * 2)
        pen.right(91)

def draw\_geometric\_pattern(pen):
    """Nested loops      complex geometric pattern"""
    pen.speed(0)
```

```

\# Pattern repetition    outer loop
for pattern in range(6):
    pen.color(random.choice(["blue", "green", "purple", "orange"]))

    \# Shape creation    middle loop
    for shape in range(8):
        \# Individual shape    inner loop
        for side in range(6):
            pen.forward(30)
            pen.right(60)
            pen.right(45)

        pen.right(60)

def draw\_star\_with\_loop(pen):
    """Loop      star """
    pen.color("gold")
    pen.begin\_fill()

    for point in range(5):
        pen.forward(100)
        pen.right(144)

    pen.end\_fill()

def draw\_concentric\_circles(pen):
    """Loop      concentric circles """
    pen.speed(0)
    colors = ["red", "orange", "yellow", "green", "blue", "purple"]

    for i in range(6):
        pen.color(colors[i])
        pen.circle(20 + i * 15)
        pen.penup()
        pen.goto(0, {-}(10 + i * 15))
        pen.pendown()

def main\_demo():
    """Main demonstration function"""
    screen = setup\_screen()
    pen = turtle.Turtle()
    pen.pensize(2)

    while True:
        print("{n}== TURTLE GRAPHICS LOOP      ===")
        print("1. Regular Polygon (    ,    ,    ,    )")
        print("2. Spiral Pattern")
        print("3. Flower Pattern")
        print("4. Colorful Squares")
        print("5. Geometric Pattern")
        print("6. Star Shape")
        print("7. Concentric Circles")
        print("8. Screen      ")
        print("9. Exit")

        try:
            choice = input("      (1{-9}): ").strip()

            if choice == {1}:
                pen.clear()
                pen.home()

```

```

sides = int(input("Sides      (3{-10}: \""))
if 3 <= sides <= 10:
    size = int(input("Size      (50{-200}: \""))
    pen.color("blue")
    draw\_polygon(sides, size, pen)
    print(f"For loop      \{sides\}-sided polygon      !\"")
else:
    print("      sides      ")

elif choice == {2}:
    pen.clear()
    pen.home()
    draw\_spiral(pen)
    print("While loop      spiral      !\"")

elif choice == {3}:
    pen.clear()
    pen.home()
    draw\_flower\_pattern(pen)
    print("Nested loops      flower pattern      !\"")

elif choice == {4}:
    pen.clear()
    pen.home()
    draw\_colorful\_squares(pen)
    print("Colors      for loop      colorful squares      !\"")

elif choice == {5}:
    pen.clear()
    pen.home()
    draw\_geometric\_pattern(pen)
    print("Nested loops      complex geometric pattern      !\"")

elif choice == {6}:
    pen.clear()
    pen.home()
    draw\_star\_with\_loop(pen)
    print("For loop      star      !\"")

elif choice == {7}:
    pen.clear()
    pen.home()
    draw\_concentric\_circles(pen)
    print("For loop      concentric circles      !\"")

elif choice == {8}:
    pen.clear()
    pen.home()
    print("Screen      !\"")

elif choice == {9}:
    print("Turtle graphics explore      !\"")
    break

else:
    print("      !\"")

except ValueError:
    print("      valid numbers      !\"")
except Exception as e:
    print(f"Error: \{e\}")

```

```

screen.exitonclick()

if __name__ == "__main__":
    main_demo()

```

Turtle માં Loop ફાયદા:

ટેબલ: Loop Benefits

ફાયદો	વર્ણન	ઉદાહરણ
Code Efficiency	Repetitive code ઓછો	એક loop vs 100 lines
Pattern Creation	Regular geometric patterns	Polygons, spirals
Dynamic Graphics	Variable-based drawing	Size/color changes
Complex Designs	Nested loop patterns	Flowers, fractals

મુખ્ય Programming Concepts:

- **Iteration:** Drawing commands repeat કરવા
- **Variables:** Size, angle, color control કરવા
- **Nesting:** Complex multi-layer patterns બનાવવા
- **Conditionals:** Conditions આધારે behavior બદલતું

Mathematical Applications:

- **Geometry:** Regular polygons ($360^\circ/n$)
- **Trigonometry:** Angles વાપરીને circular patterns
- **Fibonacci:** Mathematical ratios સાથે spiral patterns
- **Fractals:** Self-repeating patterns

Performance Tips:

- **Speed Control:** જડપી drawing માટે `pen.speed(0)` વાપરો
- **Pen Movements ઓછા કરો:** Drawing operations group કરો
- **Color Efficiency:** Color lists pre-define કરો
- **Screen Updates:** Complex patterns માટે `screen.tracer(0)` વાપરો

મેમરી ટ્રીક

“LPDC - Loops, Patterns, Dynamic, Complex graphics”

પ્રશ્ન 5(અ OR) [3 માંકર્સ]

Turtle માં Shape function સમજાવો અને Turtle માં તેમના કેટલા પ્રકારના આકાર હોય છે?

જવાબ

Turtle shape function cursor appearance ને default arrow થી વિવિધ predefined shapes માં બદલે છે better visual representation માટે.

ટેબલ: Built-in Turtle Shapes

Shape Name	વર્ણન	Usage
“arrow”	Default arrow cursor	<code>turtle.shape("arrow")</code>
“turtle”	Turtle icon	<code>turtle.shape("turtle")</code>
“circle”	Circular cursor	<code>turtle.shape("circle")</code>
“square”	Square cursor	<code>turtle.shape("square")</code>
“triangle”	Triangle cursor	<code>turtle.shape("triangle")</code>
“classic”	Classic turtle shape	<code>turtle.shape("classic")</code>

Shape Function Usage:

```
import turtle

pen = turtle.Turtle()

# shapes
pen.shape("turtle")      # Turtle icon
pen.shape("circle")       # Circle cursor
pen.shape("square")       # Square cursor
pen.shape("triangle")     # Triangle cursor

# Current shape
current = pen.shape()
print(f"Current shape: {current}")

# Available shapes list
shapes = pen.getshapes()
print(f"Available shapes: {shapes}")
```

Custom Shapes:

- **Register New:** Custom polygon shapes બનાવી શકાય
- **Import Images:** External image files વાપરી શકાય
- **Shape Coordinates:** Coordinate points વાપરીને shape define કરી શકાય

ફાયદા:

- **Visual Appeal:** Default arrow કરતાં સાંચ લાગે
- **Orientation:** Turtle ની direction સ્પષ્ટ રીતે બતાવે
- **Thematic Design:** Project theme સાથે shape match કરી શકાય

મેમરી ટ્રીક

“ATCSTC - Arrow, Turtle, Circle, Square, Triangle, Classic”

પ્રશ્ન 5(બ OR) [4 માંકર્સ]

Turtle માં વિવિધ પ્રકારના pen command શું છે? તે સમજાવો.

જવાબ

Pen commands turtle movement દ્વારા બનાવાતી lines ની drawing behavior અને appearance control કરે છે.
ટેબલ: Pen Control Commands

Command Category	Commands	હેતુ
Pen State	penup(), pendown()	Drawing control
Pen Size	pensize(width)	Line thickness
Pen Color	pencolor(color)	Line color
Pen Speed	speed(value)	Drawing speed

બિગતવાર Pen Commands:

State Control:

```
import turtle

pen = turtle.Turtle()

# Pen state commands
pen.penup()      # Pen      {- drawing   }
pen.pendown()    # Pen      {- lines    }
pen.isdown()     # Check    pen      (True/False)
```

Appearance Control:

```
# Size control
pen.pensize(1)    # line
pen.pensize(5)    # line
pen.width(3)      # pensize alternative

# Color control
pen.pencolor("red")          # Single color
pen.pencolor(255, 0, 0)       # RGB values
pen.pencolor("\#FF0000")      # Hex color

# Current settings
current_size = pen.pensize()
current_color = pen.pencolor()
```

Speed Control:

```
# Speed settings (1{-10    string})
pen.speed(1)      #
pen.speed(5)      # Medium
pen.speed(10)     #
pen.speed(0)      #      ( animation )
pen.speed("slow") # String options
pen.speed("fast")
```

ટેબલ: Speed Values

Value	Speed	વર્ણન
1	સૌથી ધીમો	Step-by-step animation
3	ધીમો	Clear movement
6	સામાન્ય	Default speed
10	જડપી	Quick drawing
0	સૌથી જડપી	કોઈ animation delay નહીં

Fill Commands:

```
\# Shapes color fill
pen.fillcolor("blue")
pen.begin\_fill()      \# Fill
pen.circle(50)        \# Shape
pen.end\_fill()        \# Fill
```

ઉદાહરણ પ્રોગ્રામ:

```
import turtle

def demonstrate\_pen\_commands():
    screen = turtle.Screen()
    screen.bgcolor("white")

    pen = turtle.Turtle()

    \#      pen sizes demonstrate
    for size in range(1, 6):
        pen.pensize(size)
        pen.forward(50)
        pen.penup()
        pen.goto(0, size * {-}20)
        pen.pendown()

    \# Colors demonstrate
    colors = ["red", "blue", "green", "purple", "orange"]
    pen.goto({-}200, 100)

    for i, color in enumerate(colors):
        pen.pencolor(color)
        pen.circle(20)
        pen.penup()
        pen.forward(50)
        pen.pendown()

    screen.exitonclick()

demonstrate\_pen\_commands()
```

મેમરી ટ્રીક

“SSCSF - State, Size, Color, Speed, Fill commands”

પ્રશ્ન 5(ક OR) [7 માંકર્સ]

Turtle નો ઉપયોગ કરીને ભારતીય ધવજ દોરવા માટેનો પ્રોગ્રામ લખો.

જવાબ

ભારતીય ધવજ દોરવાનો પ્રોગ્રામ turtle graphics ને precise measurements, colors, અને geometric construction સાથે demonstrate કરે છે.

સંપૂર્ણ ભારતીય ધવજ પ્રોગ્રામ:

```
import turtle
import math

class IndianFlagDrawer:
```

```

"""    specifications                         class"""

def __init__(self):
    self.setup_screen()
    self.pen = turtle.Turtle()
    self.setup_pen()

    # dimensions (2:3 ratio)
    self.flag_width = 300
    self.flag_height = 200
    self.stripes_height = self.flag_height // 3

    #
    self.saffron = "#FF9933"
    self.white = "#FFFFFF"
    self.green = "#138808"
    self.navy_blue = "#000080"

def setup_screen(self):
    """Turtle screen setup"""
    self.screen = turtle.Screen()
    self.screen.bgcolor("lightblue")
    self.screen.title(" ")
    self.screen.setup(800, 600)

def setup_pen(self):
    """Turtle pen setup"""
    self.pen.speed(5)
    self.pen.pensize(2)

def draw_rectangle(self, width, height, color):
    """Filled rectangle"""
    self.pen.fillcolor(color)
    self.pen.begin_fill()

    for _ in range(2):
        self.pen.forward(width)
        self.pen.right(90)
        self.pen.forward(height)
        self.pen.right(90)

    self.pen.end_fill()

def draw_flag_stripes(self):
    """
    # starting position
    start_x = {-}self.flag_width // 2
    start_y = self.flag_height // 2

    #
    ( )
    self.pen.penup()
    self.pen.goto(start_x, start_y)
    self.pen.pendown()
    self.draw_rectangle(self.flag_width, self.stripes_height, self.saffron)

    #
    ( )
    self.pen.penup()
    self.pen.goto(start_x, start_y {-} self.stripes_height)
    self.pen.pendown()
    self.draw_rectangle(self.flag_width, self.stripes_height, self.white)
    """


```

```

\#           ( )
self.pen.penup()
self.pen.goto(start_x, start_y {-} 2 * self.stripes_height)
self.pen.pendown()
self.draw_rectangle(self.flag_width, self.stripes_height, self.green)

def draw_ashoka_chakra(self):
    """ (24{-spoke wheel}) """
    # center position
    center_x = 0
    center_y = 0
    chakra_radius = 30

    self.pen.penup()
    self.pen.goto(center_x, center_y)
    self.pen.pendown()

    # self.pen.color(self.navy_blue)
    self.pen.pensize(3)
    self.pen.circle(chakra_radius)

    # self.pen.penup()
    self.pen.goto(center_x, center_y + 5)
    self.pen.pendown()
    self.pen.circle(chakra_radius {-} 5)

    # 24 spokes
    self.pen.pensize(2)
    spoke_angle = 360 / 24 # spoke 15 degrees

    for spoke in range(24):
        # Spoke endpoints calculate
        angle_rad = math.radians(spoke * spoke_angle)

        # point
        inner_x = center_x + (chakra_radius {-} 10) * math.cos(angle_rad)
        inner_y = center_y + (chakra_radius {-} 10) * math.sin(angle_rad)

        # point
        outer_x = center_x + (chakra_radius {-} 3) * math.cos(angle_rad)
        outer_y = center_y + (chakra_radius {-} 3) * math.sin(angle_rad)

        # Spoke
        self.pen.penup()
        self.pen.goto(inner_x, inner_y)
        self.pen.pendown()
        self.pen.goto(outer_x, outer_y)

    # Center dot
    self.pen.penup()
    self.pen.goto(center_x, center_y {-} 2)
    self.pen.pendown()
    self.pen.begin_fill()
    self.pen.circle(2)
    self.pen.end_fill()

def draw_flag_pole(self):
    """
    pole_height = 400

```

```

pole\_width = 8

#           pole position
pole\_x = {-}self.flag\_width // 2 {-} 20
pole\_y = self.flag\_height // 2

self.pen.penup()
self.pen.goto(pole\_x, pole\_y)
self.pen.pendown()

#
self.pen.color("brown")
self.pen.pensize(pole\_width)
self.pen.setheading(270) #      point
self.pen.forward(pole\_height)

#   base
self.pen.penup()
self.pen.goto(pole\_x {-} 15, pole\_y {-} pole\_height)
self.pen.pendown()
self.pen.setheading(0)
self.pen.color("gray")
self.pen.pensize(4)
self.pen.forward(30)

def add\_title\_and\_info(self):
    """Title   information add """
    self.pen.penup()
    self.pen.goto(0, self.flag\_height // 2 + 50)
    self.pen.pendown()
    self.pen.color("black")
    self.pen.pensize(1)

    # Title
    self.pen.write("           ", align="center",
                  font=("Arial", 16, "bold"))

    # Information add
    self.pen.penup()
    self.pen.goto(0, {-}self.flag\_height // 2 {-} 50)
    self.pen.pendown()

    info\_text = " :       | :       | :       "
    self.pen.write(info\_text, align="center",
                  font=("Arial", 10, "normal"))

    #   info add
    self.pen.penup()
    self.pen.goto(0, {-}self.flag\_height // 2 {-} 70)
    self.pen.pendown()

    chakra\_text = "      : 24 spokes      24      "
    self.pen.write(chakra\_text, align="center",
                  font=("Arial", 9, "italic"))

def draw\_complete\_flag(self):
    """
    """
    print("           ...")

    #   components
    self.draw\_flag\_pole()

```

```

        self.draw\_flag\_stripes()
        self.draw\_ashoka\_chakra()
        self.add\_title\_and\_info()

        \#      border add
        self.pen.penup()
        self.pen.goto({-}self.flag\_width // 2, self.flag\_height // 2)
        self.pen.pendown()
        self.pen.color("black")
        self.pen.pensize(2)

        for \_ in range(2):
            self.pen.forward(self.flag\_width)
            self.pen.right(90)
            self.pen.forward(self.flag\_height)
            self.pen.right(90)

        \# Turtle hide
        self.pen.hideturtle()

        print("          !")
        print("      !  ")

def interactive\_demo(self):
    """Interactive demonstration"""
    print("{n}== DRAWING ===")
    print("      ")
    print("      ")

    input("Drawing      Enter    ...")

    self.draw\_complete\_flag()

    print("{n} components:")
    print("      ( )")
    print("      ( )")
    print("      ( )")
    print("      (24 spokes)")
    print("      ")
    print(" Title   information")

    self.screen.exitonclick()

def simple\_flag\_version():
    """Beginners simplified version"""
    screen = turtle.Screen()
    screen.bgcolor("lightblue")
    screen.title("")

    pen = turtle.Turtle()
    pen.speed(3)

    \#      rectangles
    colors = ["\#FF9933", "\#FFFFFF", "\#138808"]

    pen.penup()
    pen.goto({-}150, 100)
    pen.pendown()

    for i, color in enumerate(colors):
        pen.fillcolor(color)

```

```

pen.begin\_fill()

    for \_ in range(2):
        pen.forward(300)
        pen.right(90)
        pen.forward(66)
        pen.right(90)

    pen.end\_fill()
    pen.penup()
    pen.goto({-}150, 100 {-} (i + 1) * 66)
    pen.pendown()

\# chakra
pen.penup()
pen.goto(0, 33)
pen.pendown()
pen.color("\#000080")
pen.circle(20)

pen.hideturtle()
screen.exitonclick()

def main():
    """Main program"""
    print("      Drawing      :")
    print("1.          ")
    print("2.  version")

    choice = input("      (1      2): ").strip()

    if choice == "1":
        flag\_drawer = IndianFlagDrawer()
        flag\_drawer.interactive\_demo()
    elif choice == "2":
        simple\_flag\_version()
    else:
        print("      .      version      ...")
        flag\_drawer = IndianFlagDrawer()
        flag\_drawer.draw\_complete\_flag()
        flag\_drawer.screen.exitonclick()

if \_\_name\_\_ == "\_\_main\_\__":
    main()

```

પ્રોગ્રામ ફીચર્સ:

- ચોક્કસ Proportions: Specifications અનુસાર 2:3 દવજ ratio
- ચોંઘ રંગો: Official કેસરી, સફેદ, લીલા રંગો
- અશોક ચક્ક: Mathematical precision સાથે 24-spoke wheel
- દવજદં: Base સાથે સંપૂર્ણ
- શિક્ષણાત્મક માહિતી: રંગાના અર્થ અને મહત્વ
- Interactive: User-friendly demonstration

તકનીકી Concepts:

- Geometric Calculations: Spoke positioning માટે mathematical
- Color Management: Accuracy માટે hex color codes
- Modular Design: દરેક component માટે અલગ functions
- Object-Oriented: Class-based organization

Mathematical Elements:

- Circle Geometry: Chakra radius calculations
- Trigonometry: Spoke angle calculations ($360^\circ / 24 = 15^\circ$)
- Coordinate System: Precise positioning
- Proportional Scaling: દવજ ratios જાળવવા

આઉટપુટ ઉદાહરણ:

==== DRAWING ===

Drawing Enter ...

...

!

components:

()
()
()
(24 spokes)

Title information

મેમરી ટ્રીક

``SWACP - Stripes, White-chakra, Accurate, Colors, Proportional''