Subject Name (Gujarati)

4321602 -- Winter 2023

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 ગુણ]

ડિક્શનરી શું છે? ઉદાહરણ સાથે સમજાવો.

જવાબ

ડિક્શનરી એ Python માં key-value pairs નો collection છે જે mutable અને ordered હોય છે.

Table 1: ડિક્શનરીની વિશેષતાઓ

વિશેષતા	વર્ણન
Mutable	Values ને change કરી શકાય છે
Ordered	Python 3.7+ માં insertion order maintain રહે છે
Indexed	Keys દ્વારા access કરાય છે
No Duplicates	Duplicate keys allow નથી

```
\#
student = \{
    "name": "  ",
    "age": 20,
    "course": "IT"
\}
print(student["name"])  \#    :
```

- **Key-Value Structure**: દરેક element માં key અને value હોય છે
- **Fast Access**: O(1) time complexity માં data access
- **Dynamic Size**: Runtime માં size વધારી-ઘટાડી શકાય છે

મેમરી ટ્રીક

``Dictionary = Key Value Treasure''

પ્રશ્ન 1(બ) [4 ગુણ]

ટ્યુપલ બિલ્ટ-ઇન ફંકશન અને મેથડોનો વર્ણન કરો.

જવાબ

ટ્યુપલ માં limited built-in methods છે કારણ કે તે immutable છે.

Table 2: ટ્યુપલ મેથડો

મેથડ	વર્ણન	ઉદાહરણ
count()	Element ની frequency return કરે છે	`t.count(5)`
index()	Element નું first index return કરે છે	`t.index('a')`
len()	ટ્યુપલ નું length return કરે છે	`len(t)`
max()	Maximum value return કરે છે	`max(t)`
min()	Minimum value return કરે છે	`min(t)`

```
\#
numbers = (1, 2, 3, 2, 4, 2)
print(numbers.count(2))      \#    : 3
print(numbers.index(3))      \#    : 2
print(len(numbers))          \#    : 6
```

- **Immutable Nature**: Methods ટ્યુપલ ને modify નથી કરતા
- **Return Values**: બધા methods નવી values return કરે છે
- **Type Conversion**: tuple() function થી list ને tuple માં convert કરી શકાય

મેમરી ટ્રીક

``Count Index Length Max Min''

પ્રશ્ન 1(ક) [7 ગુણ]

સેટ ઓપરેશન્સ દર્શાવવા માટે પાયથન પ્રોગ્રામ લખો.

જવાબ

Set operations mathematics ના set theory પર આધારિત છે.

Table 3: સેટ ઓપરેશન્સ

ઓપરેશન	Symbol	Method	વર્ણન
Union	`|`	`union()`	બન્ને sets ના elements
Intersection	`&`	`intersection()`	Common elements
Difference	`-`	`difference()`	First set માંથી second ને minus
Symmetric Difference	`^`	`symmetric_difference()`	Unique elements only

```
\#
set1 = \{1, 2, 3, 4, 5\}
set2 = \{4, 5, 6, 7, 8\}

print("Set 1:", set1)
print("Set 2:", set2)

\#
union\_result = set1 | set2
print("Union:", union\_result)

\#
intersection\_result = set1 \& set2
print("Intersection:", intersection\_result)

\#
difference\_result = set1 {-} set2
print("Difference:", difference\_result)

\#
sym\_diff\_result = set1 \^{} set2
print("Symmetric Difference:", sym\_diff\_result)

\#
set3 = \{1, 2\}
print("  set3   set1   subset  ?", set3.issubset(set1))
print("  set1   set3   superset  ?", set1.issuperset(set3))
```

- **Mathematical Operations**: Set theory ના operations implement કરે છે
- **Efficient Processing**: Duplicate elements automatically remove થાય છે
- **Boolean Results**: Subset/superset operations boolean return કરે છે

મેમરી ટ્રીક

``Union Intersection Difference Symmetric''

પ્રશ્ન 1(ક OR) [7 ગુણ]

ડિક્શનરી ફંક્શન અને ઓપરેશન્સ દર્શાવવા માટે પાયથન પ્રોગ્રામ લખો.

જવાબ

ડિક્શનરી ઓપરેશન્સ data manipulation માટે powerful tools પ્રદાન કરે છે.

Table 4: ડિક્શનરી મેથડો

મેથડ	વર્ણન	ઉદાહરણ
keys()	બધી keys return કરે છે	`dict.keys()`
values()	બધા values return કરે છે	`dict.values()`
items()	Key-value pairs return કરે છે	`dict.items()`
get()	Safe value retrieval	`dict.get('key')`
update()	Dictionary merge કરે છે	`dict.update()`

```
\#
student\_data = \{
    "name": "  ",
    "age": 21,
    "course": "IT",
    "semester": 2
\}

print("  Dictionary:", student\_data)

\# Values   access
print("       :", student\_data.get("name"))
print("        :", student\_data["age"])

\#    key{-value pair    }
student\_data["city"] = "    "
print("        :", student\_data)

\#         value
student\_data.update(\{"age": 22, "semester": 3\})
print("        :", student\_data)

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

\# Elements   remove
removed\_value = student\_data.pop("semester")
print("Remove    value:", removed\_value)
print("   Dictionary:", student\_data)
```

- **Dynamic Operations**: Runtime માં keys અને values add/remove કરી શકાય
- **Safe Access**: get() method KeyError prevent કરે છે
- **Iteration Support**: keys(), values(), items() methods loop માટે useful

મેમરી ટ્રીક

``Get Keys Values Items Update Pop''

પ્રશ્ન 2(અ) [3 ગુણ]

પાયથનમાં ટ્યુપલ અને લિસ્ટ વચ્ચે તફાવત આપો.

જવાબ

Table 5: ટ્યુપલ vs લિસ્ટ તુલના

વિશેષતા	ટ્યુપલ	લિસ્ટ
Mutability	Immutable (બદલાઈ શકતું નથી)	Mutable (બદલાઈ શકે છે)
Syntax	કૌંસ `()`	ચોરસ કૌંસ `[]`
Performance	ઝડપી	ધીમી
Memory	ઓછી memory	વધુ memory
Methods	મર્યાદિત (count, index)	ઘણી methods ઉપલબ્ધ
Use Case	Fixed data	Dynamic data

- **Immutable Nature**: ટ્યુપલ એકવાર create થયા પછી change થઈ શકતું નથી
- **Performance**: ટ્યુપલ operations લિસ્ટ કરતાં ઝડપી છે
- **Memory Efficient**: ટ્યુપલ ઓછી memory વાપરે છે

મેમરી ટ્રીક

``Tuple Tight, List Light''

પ્રશ્ન 2(બ) [4 ગુણ]

પાયથનમાં dir() ફંકશન શું છે? ઉદાહરણ સાથે સમજાવો.

જવાબ

dir() function એ built-in function છે જે object ના attributes અને methods ની list return કરે છે.

Table 6: dir() ફંકશનની વિશેષતાઓ

વિશેષતા	વર્ણન
Object Inspection	Object ના attributes show કરે છે
Method Discovery	Available methods list કરે છે
Namespace Exploration	Current namespace ના variables show કરે છે
Module Analysis	Module ના contents explore કરે છે

```
\# dir()
\# String object
text = "Hello"
string\_methods = dir(text)
print("String methods:", string\_methods[:5])

\# List object
my\_list = [1, 2, 3]
list\_methods = dir(my\_list)
print("List methods:", [m for m in list\_methods if not m.startswith({\_})][:5])

\# Current namespace
print("Current namespace:", dir()[:3])

\# Built{-in functions   }
import math
print("Math module:", dir(math)[:5])
```

- **Interactive Development**: Objects ના capabilities જાણવા માટે useful
- **Debugging Tool**: Available methods quickly identify કરવા માટે
- **Learning Aid**: નવી libraries explore કરવા માટે helpful

મેમરી ટ્રીક

``Dir = Directory of Methods''

પ્રશ્ન 2(ક) [7 ગુણ]

સર્કલનો એરિયા અને સિર્કમફ્રેન્સ શોધવા માટે મોડ્યુલ બનાવો અને બીજા પ્રોગ્રામમાં મોડ્યુલ ઇમ્પોર્ટ કરો.

જવાબ

Module approach કોડ reusability અને organization સુધારે છે.
ડાયાગ્રામ: મોડ્યુલ સ્ટ્રક્ચર

```
    circle.py             main.py
    (Module)              (Main Program)

  • area()            import circle
  • circumference       use functions
  • PI constant
```

ફાઇલ 1: circle.py (મોડ્યુલ)

```
\# circle.py {-            }
import math

\# Constants
PI = math.pi

def area(radius):
    """               """
    if radius {} 0:
        return "Radius negative       "
    return PI * radius * radius

def circumference(radius):
    """    circumference        """
    if radius {} 0:
        return "Radius negative       "
    return 2 * PI * radius

def display\_info():
    """            """
    print("Circle Module {- Version 1.0"})
    print("Functions: area(), circumference()")
```

ફાઇલ 2: main.py (મુખ્ય પ્રોગ્રામ)

```
\# main.py {- circle module            }
import circle

\#       radius
radius = float(input("Radius      : "))

\#    functions
circle\_area = circle.area(radius)
circle\_circumference = circle.circumference(radius)

\#
print(f"Radius \{radius\}        :")
print(f"Area: \{circle\_area:.2f\}")
print(f"Circumference: \{circle\_circumference:.2f\}")

\#    info
circle.display\_info()
```

- **Modular Design**: Functions ને separate file માં organize કરે છે
- **Reusability**: Module ને multiple programs માં use કરી શકાય
- **Namespace Management**: Module prefix થી function access કરાય છે

મેમરી ટ્રીક

``Import Calculate Display''

પ્રશ્ન 2(અ OR) [3 ગુણ]

નેસ્ટેડ ટ્યુપલને ઉદાહરણ સાથે સમજાવો.

જવાબ

Nested Tuple એ ટ્યુપલ અંદર બીજા tuples હોય છે, જે hierarchical data structure બનાવે છે.

Table 7: નેસ્ટેડ ટ્યુપલની વિશેષતાઓ

વિશેષતા	વર્ણન
Multi-dimensional	2D અથવા 3D data structure
Immutable	બધા levels પર immutable
Indexing	Multiple square brackets વાપરીને access
Heterogeneous	અલગ-અલગ data types store કરી શકાય

```
\#
student\_records = (
    ("  ", 20, ("IT", 2)),
    ("   ", 19, ("CS", 1)),
    ("  ", 21, ("IT", 3))
)

\#    elements   access
print("        :", student\_records[0])
print("           :", student\_records[0][0])
print("            :", student\_records[0][2][0])

\#          iterate
for student in student\_records:
    name, age, (course, semester) = student
    print(f"\{name\} {-   : }\{age\},    : \{course\},   : \{semester\}")
```

- **Data Organization**: સંબંધિત data ને group કરવા માટે useful
- **Immutable Structure**: એકવાર create થયા પછી structure change થઈ શકતું નથી
- **Efficient Access**: Index-based fast access

મેમરી ટ્રીક

``Nested = Tuple Inside Tuple''

પ્રશ્ન 2(બ OR) [4 ગુણ]

PIP શું છે? પાયથન પેકેજને ઇન્સ્ટોલ અને અનઇન્સ્ટોલ કરવા માટે સિન્ટેક્સ લખો.

જવાબ

PIP (Pip Installs Packages) એ Python package installer છે જે PyPI થી packages download અને install કરે છે.

Table 8: PIP કમાન્ડો

કમાન્ડ	સિન્ટેક્સ	વર્ણન
Install	`pip install package_name`	Package install કરે છે

Uninstall	`pip uninstall package_name`	Package remove કરે છે
List	`pip list`	Installed packages show કરે છે
Show	`pip show package_name`	Package info display કરે છે
Upgrade	`pip install --upgrade package_name`	Package update કરે છે

```
\# PIP          (Terminal/Command Prompt    run    )

\# Package install
\# pip install requests

\# Specific version install
\# pip install Django==3.2.0

\# Package uninstall
\# pip uninstall numpy

\#    installed packages list
\# pip list

\# Package information show
\# pip show matplotlib

\# Package upgrade
\# pip install {-{-}upgrade pandas}

\# Requirements file   install
\# pip install {-r requirements.txt}
```

- **Package Management**: Third-party libraries easily manage કરી શકાય
- **Version Control**: Specific versions install કરી શકાય
- **Dependency Resolution**: જરૂરી dependencies automatically install થાય

મેમરી ટ્રીક

``PIP = Package Install Python''

પ્રશ્ન 2(ક OR) [7 ગુણ]

પેકેજ ઇમ્પોર્ટ કરવાની વિવિધ રીતો સમજાવો. મોડ્યુલ અને પેકેજ એકબીજાની સાથે કેવી રીતે જોડાયેલા છે?

જવાબ

Python માં imports ના વિવિધ ways છે જે code organization અને namespace management માટે મહત્વપૂર્ણ છે.
ડાયાગ્રામ: પેકેજ સ્ટ્રક્ચર

```
MyPackage/
   \_\_init\_\_.py
   module1.py
   module2.py
   subpackage/
       \_\_init\_\_.py
       module3.py
```

Table 9: ઇમ્પોર્ટ મેથડો

મેથડ	સિન્ટેક્સ	ઉપયોગ
Basic Import	`import module`	સંપૂર્ણ module name જરૂરી

From Import	`from module import function`	Direct function access
Alias Import	`import module as alias`	Module માટે ટૂંકું નામ
Star Import	`from module import *`	બધા functions import કરવા
Package Import	`from package import module`	Package માંથી import કરવા

```
\#

\# 1. Basic Import
import math
result = math.sqrt(16)

\# 2. From Import
from math import sqrt, pi
result = sqrt(16)
area = pi * 5 * 5

\# 3. Alias Import
import numpy as np
array = np.array([1, 2, 3])

\# 4. Star Import (     )
from math import *
result = cos(0)

\# 5. Package Import
from mypackage import module1
from mypackage.subpackage import module3

\# 6. Relative Import (package   )
\# from . import module1
\# from ..parent\_module import function
```

મોડ્યુલ-પેકેજ કનેક્શન:

- **Modules**: Python code ધરાવતી single .py files
- **Packages**: __init__.py સાથે multiple modules ધરાવતી directories
- **Namespace**: Packages hierarchical namespace structure બનાવે છે
- **__init__.py**: Directory ને package બનાવે છે અને imports control કરે છે

મેમરી ટ્રીક

``Import From As Star Package''

પ્રશ્ન 3(અ) [3 ગુણ]

રનટાઇમ એરર અને સિન્ટેક્સ એરરનું વર્ણન કરો. ઉદાહરણ સાથે સમજાવો.

જવાબ

Table 10: એરર પ્રકારોની તુલના

એરર પ્રકાર	ક્યારે થાય છે	Detection	ઉદાહરણ
Syntax Error	Code parsing time	Execution પહેલાં	Missing colon, brackets
Runtime Error	Execution દરમિયાન	Program run કરતી વખતે	Division by zero, file not found
Logic Error	હંમેશા	Execution પછી	ખોટી calculation logic

```
\#
\# print("Hello World"  \# Missing closing parenthesis
\# SyntaxError: unexpected EOF while parsing

\#
try:
    \# ZeroDivisionError
    result = 10 / 0
except ZeroDivisionError:
    print("               ")

try:
    \# FileNotFoundError
    file = open("nonexistent.txt", "r")
except FileNotFoundError:
    print("        ")
```

- **Syntax Errors**: કોડ run થવા પહેલા જ detect થાય છે
- **Runtime Errors**: Program execution દરમિયાન થાય છે
- **Prevention**: Exception handling runtime errors ને handle કરે છે

મેમરી ટ્રીક

``Syntax Before, Runtime During''

પ્રશ્ન 3(બ) [4 ગુણ]

પાયથનમાં Exception હેન્ડલિંગ શું છે? ઉદાહરણ સાથે સમજાવો.

જવાબ

Exception handling એ mechanism છે જે runtime errors ને gracefully handle કરે છે અને program crash થવાથી prevent કરે છે.

Table 11: Exception Handling Keywords

Keyword	હેતુ	વર્ણન
try	Exception માં થઈ શકે એવો code	Risk code block
except	Exception handle કરવા માટે	Error handling block
finally	હંમેશા execute થાય	Cleanup code
else	Exception ન આવે તો	Success code block
raise	Manual exception raise કરવા	Custom error throwing

```
\# Exception Handling
def safe\_division(a, b):
    try:
        \#     exception raise
        result = a / b
        print(f"       : \{result\}")

    except ZeroDivisionError:
        \# Specific exception handle
        print("  :               ")
        result = None

    except TypeError:
        \# Type errors handle
        print("  :     data types")
        result = None

    else:
        \# Exception       execute
        print("              ")

    finally:
        \#    execute
        print("     operation    ")

    return result

\#
safe\_division(10, 2)   \#
safe\_division(10, 0)   \# Zero division
safe\_division(10, "a") \# Type error
```

- **Error Prevention**: Program crash થવાથી prevent કરે છે
- **Graceful Handling**: User-friendly error messages આપે છે
- **Resource Management**: finally block માં cleanup operations

મેમરી ટ્રીક

``Try Except Finally Else Raise''

પ્રશ્ન 3(ક) [7 ગુણ]

બે સંખ્યાઓના division માટે ફંકશન બનાવો, જો કોઈપણ argument ની value non-integer હોય તો એરર રેઇઝ થાય અથવા જો બીજી argument 0 હોય તો એરર રેઇઝ થાય.

જવાબ

Custom exception handling function બનાવવું validation અને error control માટે મહત્વપૂર્ણ છે.
ડાયાગ્રામ: ફંકશન ફ્લો

```
flowchart LR
    A[Input Numbers] {-{-} B\{    integers  ?\}}
    B {-{-}| | C[Raise TypeError]}
    B {-{-}| | D\{        0  ?\}}
    D {-{-}| | E[Raise ZeroDivisionError]}
    D {-{-}| | F[Division   ]}
    F {-{-} G[Result Return   ]}
    C {-{-} H[Error Handle   ]}
    E {-{-} H}
```

```
    H {-{-} I[ ]}
    G {-{-} I}

def safe\_integer\_division(num1, num2):
    """
          validation
      arguments integers      TypeError raise
         argument 0     ZeroDivisionError raise
    """

    \#         arguments integers
    if not isinstance(num1, int):
        raise TypeError(f"   argument integer      ,    \{type(num1).\_\_name\_\_\}")

    if not isinstance(num2, int):
        raise TypeError(f"   argument integer      ,    \{type(num2).\_\_name\_\_\}")

    \#
    if num2 == 0:
        raise ZeroDivisionError("              ")

    \#
    result = num1 / num2
    return result

\#
def test\_division():
    test\_cases = [
        (10, 2),      \#
        (15, 3),      \#
        (10, 0),      \# Zero division error
        (10.5, 2),    \# Non{-integer    argument}
        (10, 2.5),    \# Non{-integer    argument}
        ("10", 2),    \# String argument
    ]

    for num1, num2 in test\_cases:
        try:
            result = safe\_integer\_division(num1, num2)
            print(f"\{num1\}  \{num2\} = \{result\}")

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

        except ZeroDivisionError as e:
            print(f"Zero Division Error: \{e\}")

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

        print("{-"} * 40)

\#    run
test\_division()
```

- **Input Validation**: Arguments ના type અને value check કરે છે
- **Custom Errors**: Specific exceptions raise કરે છે
- **Error Messages**: સ્પષ્ટ અને વર્ણનાત્મક error messages

મેમરી ટ્રીક

``Validate Type, Check Zero, Divide Safe''

પ્રશ્ન 4(અ) [3 ગુણ]

ટેક્સ્ટ ફાઇલ અને બાયનરી ફાઇલ વચ્ચેના તફાવત પર પાંચ પોઇન્ટ્સ લખો.

જવાબ

Table 12: ટેક્સ્ટ ફાઇલ vs બાયનરી ફાઇલ

વિશેષતા	ટેક્સ્ટ ફાઇલ	બાયનરી ફાઇલ
Content	Human-readable characters	Binary data (0s અને 1s)
Encoding	Character encoding (UTF-8, ASCII)	કોઈ character encoding નથી
Opening Mode	`r', `w', `a'	`rb', `wb', `ab'
File Size	સામાન્યતે મોટી	સામાન્યતે નાની
Platform	Platform dependent	Platform independent

```
\#    vs
\#
with open("sample.txt", "w") as f:
    f.write("Hello World")

\#
with open("sample.bin", "wb") as f:
    f.write(b{}{x48x65x6cx6cx6f}{})
```

- **Readability**: ટેક્સ્ટ ફાઇલો editor માં read કરી શકાય, બાયનરી ફાઇલો માટે special software જોઈએ
- **Portability**: બાયનરી ફાઇલો different platforms પર easily transfer થાય
- **Processing**: ટેક્સ્ટ ફાઇલો string operations માટે, બાયનરી ફાઇલો exact data storage માટે

મેમરી ટ્રીક

``Text Human, Binary Machine''

પ્રશ્ન 4(બ) [4 ગુણ]

ફાઇલમાંથી ડેટા read કરવા માટે પ્રોગ્રામ લખો અને Uppercase કેરેક્ટર અને Lowercase કેરેક્ટરને બે અલગ ફાઇલોમાં separate કરો.

જવાબ

ફાઇલ processing માં character-based operations સામાન્ય requirements છે.

Table 13: ફાઇલ ઓપરેશન્સ

ઓપરેશન	મેથડ	હેતુ
Read	`read()`	સંપૂર્ણ ફાઇલ content
Write	`write()`	ફાઇલમાં string લખવા
Character Check	`isupper()`, `islower()`	Character case detection
File Handling	`with open()`	Safe ફાઇલ operations

```
def separate\_case\_characters(input\_file, upper\_file, lower\_file):
    """
       read    uppercase/lowercase characters
    """
    try:
        \# Input      read
        with open(input\_file, {r}) as infile:
            content = infile.read()

        \# Characters
        uppercase\_chars = ""
        lowercase\_chars = ""

        for char in content:
            if char.isupper():
                uppercase\_chars += char
            elif char.islower():
                lowercase\_chars += char

        \# Uppercase
        with open(upper\_file, {w}) as upfile:
            upfile.write(uppercase\_chars)

        \# Lowercase
        with open(lower\_file, {w}) as lowfile:
            lowfile.write(lowercase\_chars)

        print(f"  Characters            !")
        print(f"Uppercase characters: \{len(uppercase\_chars)\}")
        print(f"Lowercase characters: \{len(lowercase\_chars)\}")

    except FileNotFoundError:
        print(f"  :    {}\{input\_file\}{     "})
    except Exception as e:
        print(f"  : \{e\}")

\# Sample input
def create\_sample\_file():
    sample\_text = """Hello World!     SAMPLE Text     .
   UPPERCASE   lowercase Characters  .
Python Programming     FUN   Educational  ."""

    with open("input.txt", "w") as f:
        f.write(sample\_text)
    print("Sample input       : input.txt")

\#    execution
create\_sample\_file()
separate\_case\_characters("input.txt", "uppercase.txt", "lowercase.txt")

\#
print("{n}   Contents:")
print("{-"} * 30)

try:
    with open("uppercase.txt", "r") as f:
        print(f"Uppercase    : \{f.read()\}")

    with open("lowercase.txt", "r") as f:
        print(f"Lowercase    : \{f.read()\}")
except FileNotFoundError:
```

```
print("Output          ")
```

- **Character Processing**: દરેક character ની case individually check કરાય છે
- **File Safety**: with statement automatic ફાઇલ closing ensure કરે છે
- **Error Handling**: ફાઇલ operations માં proper exception handling

મેમરી ટ્રીક

``Read Separate Write''

પ્રશ્ન 4(ક) [7 ગુણ]

dump() અને load() મેથડનું વર્ણન કરો. ઉદાહરણ સાથે સમજાવો.

જવાબ

dump() અને load() methods pickle module ના part છે જે object serialization માટે વાપરાય છે.

Table 14: Pickle મેથડો

મેથડ	હેતુ	File Mode	વર્ણન
dump()	Object ને file માં serialize કરવા	`wb'	Object ને binary file માં store કરે
load()	File માંથી object deserialize કરવા	`rb'	ફાઇલમાંથી object retrieve કરે
dumps()	Bytes માં serialize કરવા	N/A	Object ને bytes માં convert કરે
loads()	Bytes માંથી deserialize કરવા	N/A	Bytes માંથી object બનાવે

ડાયાગ્રામ: Serialization Process

```
flowchart LR
    A[Python Object] {-{-}|"dump()"| B[Binary File]}
    B {-{-}|"load()"| C[Python Object]}
    A {-{-}|"dumps()"| D[Bytes String]}
    D {-{-}|"loads()"| C}

import pickle

\#    data types
def demonstrate\_pickle():
    \# Serialize       sample data
    student\_data = \{
        {name}: {    },
        {age}: 20,
        {grades}: [85, 92, 78, 96],
        {subjects}: ({Math}, {Python}, {Database}),
        {is\_active}: True
    \}

    class Student:
        def \_\_init\_\_(self, name, roll\_no):
            self.name = name
            self.roll\_no = roll\_no

        def \_\_str\_\_(self):
            return f"Student: \{self.name\} (Roll: \{self.roll\_no\})"

    \# Objects
    student\_obj = Student("     ", 101)
    data\_list = [student\_data, student\_obj, [1, 2, 3, 4, 5]]

    \# DUMP {- Objects   file   serialize   }
    print("=== DUMP Operation ===")
    try:
        with open({student\_data.pkl}, {wb}) as f:
            pickle.dump(data\_list, f)
        print("  Data       student\_data.pkl   dump    ")

        \# dumps()   demonstrate
        serialized\_bytes = pickle.dumps(student\_data)
        print(f"  Data   bytes   serialize    : \{len(serialized\_bytes)\} bytes")

    except Exception as e:
        print(f"  Dump error: \{e\}")

    \# LOAD {- File    objects deserialize     }
    print("{n}=== LOAD Operation ===")
    try:
        with open({student\_data.pkl}, {rb}) as f:
            loaded\_data = pickle.load(f)

        print("  Data       student\_data.pkl    load    ")
        print("{n}Loaded Data:")
        print("{-"} * 20)

        \# Loaded data display
        for i, item in enumerate(loaded\_data):
            print(f"Item \{i+1\}: \{item\}")
            print(f"Type: \{type(item)\}")
            print()
```

```
        \# loads()   demonstrate
        deserialized\_data = pickle.loads(serialized\_bytes)
        print(f"  Bytes    data deserialize    : \{deserialized\_data\}")

    except FileNotFoundError:
        print("  Pickle        ")
    except Exception as e:
        print(f"  Load error: \{e\}")

\# Advanced     custom class
def advanced\_pickle\_example():
    class BankAccount:
        def \_\_init\_\_(self, account\_no, holder\_name, balance):
            self.account\_no = account\_no
            self.holder\_name = holder\_name
            self.balance = balance
            self.transactions = []

        def deposit(self, amount):
            self.balance += amount
            self.transactions.append(f"Deposit: +\{amount\}")

        def withdraw(self, amount):
            if self.balance {=} amount:
                self.balance {-=} amount
                self.transactions.append(f"Withdraw: {-}\{amount\}")
            else:
                print("   balance")

        def \_\_str\_\_(self):
            return f"Account \{self.account\_no\}: \{self.holder\_name\} {- Balance:  }\{self.balance\}

    \# Account
    account = BankAccount("12345", "     ", 5000)
    account.deposit(1500)
    account.withdraw(800)

    print("=== Advanced Pickle     ===")
    print(f" : \{account\}")
    print(f"Transactions: \{account.transactions\}")

    \# Account object serialize
    with open({bank\_account.pkl}, {wb}) as f:
        pickle.dump(account, f)

    \# Account object load
    with open({bank\_account.pkl}, {rb}) as f:
        loaded\_account = pickle.load(f)

    print(f"Loaded: \{loaded\_account\}")
    print(f"Loaded transactions: \{loaded\_account.transactions\}")

    \# Object functionality verify
    loaded\_account.deposit(200)
    print(f"   deposit   : \{loaded\_account\}")

\# Demonstrations run
demonstrate\_pickle()
print("{n}" + "="*50 + "{n}")
advanced\_pickle\_example()
```

ફાયદા અને મર્યાદાઓ:

```
\#
benefits = [
    "   object state preservation",
    "Complex nested objects         ",
    "Object relationships maintain    ",
    "Fast serialization/deserialization"
]

\#
limitations = [
    "Python{-specific format"},
    "      data    security risks",
    "Version compatibility issues",
    "Human{-readable   "}
]

print("   :", benefits)
print("    :", limitations)
```

- **Object Persistence**: Python objects ને ફાઇલમાં permanently store કરી શકાય
- **Complete State**: Object ની complete state including methods preserve થાય છે
- **Binary Format**: Efficient storage પણ human-readable નથી

મેમરી ટ્રીક

``Dump Store, Load Restore''

પ્રશ્ન 4(અ OR) [3 ગુણ]

ફાઇલ ઓપરેશન માટે પાયથન દ્વારા આપેલ વિવિધ પ્રકારના ફાઇલ modes ની સૂચિ બનાવો અને તેમના ઉપયોગો સમજાવો.

જવાબ

Table 15: Python ફાઇલ Modes

Mode	પ્રકાર	વર્ણન	Pointer Position
`r'	Text Read	માત્ર read કરવા, ફાઇલ હોવી જરૂરી	શરૂઆત
`w'	Text Write	માત્ર write કરવા, creates/overwrites	શરૂઆત
`a'	Text Append	માત્ર write કરવા, creates if not exist	અંત
`x'	Text Create	નવી ફાઇલ બનાવે, exists હોય તો fail	શરૂઆત
`rb'	Binary Read	Binary data read કરવા	શરૂઆત
`wb'	Binary Write	Binary data write કરવા	શરૂઆત
`ab'	Binary Append	Binary data append કરવા	અંત
`r+'	Text Read/Write	Read અને write, ફાઇલ હોવી જરૂરી	શરૂઆત
`w+'	Text Write/Read	Write અને read, creates/overwrites	શરૂઆત

```
\#    Modes
import os

\# Demonstration    sample
with open({demo.txt}, {w}) as f:
    f.write("  content{n}Line 2{n}Line 3")

\# Read mode ({r})}
with open({demo.txt}, {r}) as f:
    content = f.read()
    print("Read mode:", content)

\# Append mode ({a)  }
with open({demo.txt}, {a}) as f:
    f.write("{n}Appended line")

\# Read+Write mode ({r+)}
with open({demo.txt}, {r+}) as f:
    f.seek(0)  \#
    f.write("Modified")

print("   modes       demonstrate   ")
```

- **Safety**: `x' mode accidental ફાઇલ overwriting prevent કરે છે
- **Efficiency**: Binary modes non-text data માટે ઝડપી છે
- **Flexibility**: Combined modes બંને read અને write operations allow કરે છે

મેમરી ટ્રીક

``Read Write Append Create Binary Plus''

પ્રશ્ન 4(બ OR) [4 ગુણ]

ફાઇલના readline() અને writeline() ફંકશનનું વર્ણન કરો.

જવાબ

નોંધ: Python માં `writeline()` ફંકશન અસ્તિત્વમાં નથી. યોગ્ય ફંકશન `writelines()` છે.

Table 16: Line-based ફાઇલ ફંકશન્સ

ફંકશન	હેતુ	Return Type	ઉપયોગ
readline()	એક line read કરવા	String	Sequential line reading
readlines()	બધી lines read કરવા	List of strings	Complete ફાઇલ as list
writelines()	Multiple lines write કરવા	None	List of strings લખવા
write()	Single string write કરવા	Number of chars	Basic writing

```
def demonstrate\_line\_functions():
    \# Multiple lines   sample
    lines\_to\_write = [
        "     {n}",
        "     {n}",
        "      {n}",
        "      newline   "
    ]

    print("=== WRITELINES() Demonstration ===")
    \# writelines()     multiple lines
    with open({sample\_lines.txt}, {w}) as f:
        f.writelines(lines\_to\_write)
    print("  writelines()     multiple lines   ")

    print("{n}=== READLINE() Demonstration ===")
    \# readline()      {-  line read   }
    with open({sample\_lines.txt}, {r}) as f:
        line\_count = 0
        while True:
            line = f.readline()
            if not line:  \#
                break
            line\_count += 1
            print(f"Line \{line\_count\}: \{line.strip()\}")

    print(f"  lines read   : \{line\_count\}")

    print("{n}=== READLINES() Demonstration ===")
    \# readlines()        lines     read
    with open({sample\_lines.txt}, {r}) as f:
        all\_lines = f.readlines()

    print("List       lines:")
    for i, line in enumerate(all\_lines, 1):
        print(f"  [\{i\}] \{repr(line)\}")

    \# Practical    :     line by line process
    print("{n}=== Practical     ===")
    student\_data = [
        "  ,20,IT{n}",
        "   ,19,CS{n}",
        "   ,21,EC{n}",
        "   ,20,IT{n}"
    ]

    \# Student data
    with open({students.txt}, {w}) as f:
        f.writelines(student\_data)

    \# Line by line read   process
    print("Student Information:")
    with open({students.txt}, {r}) as f:
        while True:
            line = f.readline()
            if not line:
                break

            \#    line   process
            parts = line.strip().split({,})
            if len(parts) == 3:
```

```
                name, age, course = parts
                print(f"  \{name\} (  : \{age\},    : \{course\})")

\# Demonstration run
demonstrate\_line\_functions()

\#    pointer behavior
def file\_pointer\_demo():
    print("{n}===    Pointer Behavior ===")

    with open({sample\_lines.txt}, {r}) as f:
        print(f"    position: \{f.tell()\}")

        line1 = f.readline()
        print(f"readline()   : position \{f.tell()\}")
        print(f"Read: \{repr(line1)\}")

        line2 = f.readline()
        print(f"   readline()   : position \{f.tell()\}")
        print(f"Read: \{repr(line2)\}")

file\_pointer\_demo()
```

- **Sequential Access**: readline() sequential manner માં lines read કરે છે
- **Memory Efficient**: મોટી ફાઇલો માટે readline() memory-efficient છે
- **List Operations**: writelines() list of strings ને efficiently write કરે છે

મેમરી ટ્રીક

``Read Line, Write Lines''

પ્રશ્ન 4(ક OR) [7 ગુણ]

seek() અને tell() methods ને demonstrate કરવા માટે પાયથન પ્રોગ્રામ લખો.

જવાબ

seek() અને tell() methods ફાઇલ pointer manipulation માટે વાપરાય છે.

Table 17: ફાઇલ Pointer મેથડો

મેથડ	હેતુ	Parameters	Return Value
tell()	Current position	None	Integer (byte position)
seek()	Pointer move કરવા	offset, whence	New position
whence=0	શરૂઆતથી	Default	Absolute position
whence=1	Current થી	Relative	Current + offset
whence=2	અંતથી	End relative	End + offset

ડાયાગ્રામ: ફાઇલ Pointer Movement

```
  : "Hello World"
     \^{             \^{}         \^{}}
   Position 0   Position 6  Position 11 (EOF)

seek(0)    {-     move   }
seek(6)    {- Position 6   move     }
seek({-5,2) {-}    5 positions    move   }

def demonstrate\_seek\_tell():
    \# Known content    sample
    sample\_text = "Hello Python Programming World!"

    with open({pointer\_demo.txt}, {w}) as f:
        f.write(sample\_text)

    print("===    Pointer Demonstration ===")
    print(f"   content: {}\{sample\_text\}{"})
    print(f"   length: \{len(sample\_text)\} characters")
    print()

    with open({pointer\_demo.txt}, {r}) as f:
        \#     position
        print(f"1.     position: \{f.tell()\}")

        \#     characters read
        first\_part = f.read(5)  \# "Hello" read
        print(f"2. {}\{first\_part\}{ read      : position }\{f.tell()\}")

        \# Specific position   move
        f.seek(6)  \# Position 6   move    ("Python"     )
        print(f"3. seek(6)   : position \{f.tell()\}")

        \#    position   read
        next\_part = f.read(6)  \# "Python" read
        print(f"4. {}\{next\_part\}{ read    : position }\{f.tell()\}")

        \#
        f.seek(0)  \#
        print(f"5. seek(0)   : position \{f.tell()\}")

        \#
        f.seek(0, 2)  \#    0 offset (position 2 = end)
        print(f"6. seek(0,2) {-        : position }\{f.tell()\}")

        \#       move
        f.seek({-}6, 2)  \#    6 positions
        print(f"7. seek({-6,2)   : position }\{f.tell()\}")

        \#     content read
        remaining = f.read()
        print(f"8.     {}\{remaining\}{ read    : position }\{f.tell()\}")

def practical\_seek\_tell\_example():
    print("{n}=== Practical    :    Editor Simulation ===")

    \# Structured data
    data\_lines = [
        "NAME:John Doe{n}",
        "AGE:25{n}",
        "CITY:Mumbai{n}",
```

```
        "PHONE:9876543210{n}",
        "EMAIL:john@example.com{n}"
    ]

    with open({person\_data.txt}, {w}) as f:
        f.writelines(data\_lines)

    \# Specific data find   modify    demonstrate
    with open({person\_data.txt}, {r+}) as f:  \# Read+Write mode
        \#    positions find   display
        positions = \{\}

        while True:
            pos = f.tell()
            line = f.readline()
            if not line:
                break

            field = line.split({:})[0]
            positions[field] = pos
            print(f"Field {}\{field\}{ position }\{pos\}        ")

        print(f"{n}   positions: \{positions\}")

        \# Specific field (AGE) modify
        if {AGE} in positions:
            f.seek(positions[{AGE}])
            print(f"{n}AGE field   position \{f.tell()\}   move   ")

            \# Current line read
            current\_line = f.readline()
            print(f"Current line: \{current\_line.strip()\}")

            \# Overwrite    position calculate
            f.seek(positions[{AGE}])
            new\_age\_line = "AGE:26{n}"  \#   length
            f.write(new\_age\_line)
            print(f"AGE field      ")

    \# Changes verify
    print("{n}         content:")
    with open({person\_data.txt}, {r}) as f:
        print(f.read())

def binary\_seek\_tell\_demo():
    print("{n}=== Binary    Seek/Tell Demo ===")

    \# Binary
    binary\_data = b{}{x48x65x6cx6cx6fx20x57x6fx72x6cx64}{}  \# "Hello World"

    with open({binary\_demo.bin}, {wb}) as f:
        f.write(binary\_data)

    with open({binary\_demo.bin}, {rb}) as f:
        print(f"Binary    size: \{len(binary\_data)\} bytes")

        \# Binary mode      seek modes demonstrate
        print(f"    position: \{f.tell()\}")

        \#    5 bytes read
        first\_bytes = f.read(5)
```

```
        print(f"   5 bytes read    : \{first\_bytes\} at position \{f.tell()\}")

        \# Current position   relative seek (binary mode        )
        f.seek(1, 1)  \# Current   1 byte    move
        print(f"seek(1,1)   : position \{f.tell()\}")

        \#    seek
        f.seek({-}3, 2)  \#    3 bytes
        print(f"seek({-3,2)   : position }\{f.tell()\}")

        \#     bytes
        remaining\_bytes = f.read()
        print(f"    bytes: \{remaining\_bytes\}")

\#   demonstrations run
demonstrate\_seek\_tell()
practical\_seek\_tell\_example()
binary\_seek\_tell\_demo()

\# Cleanup
import os
try:
    os.remove({pointer\_demo.txt})
    os.remove({person\_data.txt})
    os.remove({binary\_demo.bin})
    print("{n}Demo          ")
except:
    pass
```

- **ફાઇલ Navigation**: seek() arbitrary position પર move કરવા માટે વાપરાય છે
- **Position Tracking**: tell() current position track કરવા માટે useful છે
- **ફાઇલ Editing**: Specific locations પર data modify કરવા માટે જરૂરી

મેમરી ટ્રીક

``Tell Position, Seek Destination''

પ્રશ્ન 5(અ) [3 ગુણ]

ટર્ટલનો ઉપયોગ કરીને circle અને rectangle ના આકાર દોરો અને તેમને લાલ રંગથી ભરો.

જવાબ

Turtle graphics module માં shapes draw કરવા અને fill કરવા માટે specific methods છે.

Table 18: Turtle Shape મેથડો

મેથડ	હેતુ	ઉદાહરણ
circle()	Circle draw કરવા	`turtle.circle(50)`
forward()	આગળ move કરવા	`turtle.forward(100)`
right()	જમણે turn કરવા	`turtle.right(90)`
begin_fill()	Fill શરૂ કરવા	`turtle.begin_fill()`
end_fill()	Fill સમાપ્ત કરવા	`turtle.end_fill()`
fillcolor()	Fill color set કરવા	`turtle.fillcolor("red")`

```
import turtle

def draw\_filled\_shapes():
    \# Screen   turtle
    screen = turtle.Screen()
    screen.title("Turtle   Filled Shapes")
    screen.bgcolor("white")
    screen.setup(800, 600)

    \# Turtle object
    painter = turtle.Turtle()
    painter.speed(3)

    \# Filled circle draw
    print("Filled circle draw        ...")
    painter.penup()
    painter.goto({-}150, 0)  \#       move
    painter.pendown()

    painter.fillcolor("red")
    painter.begin\_fill()
    painter.circle(80)  \# Radius = 80
    painter.end\_fill()

    \# Filled rectangle draw
    print("Filled rectangle draw        ...")
    painter.penup()
    painter.goto(50, 50)  \#       move
    painter.pendown()

    painter.fillcolor("red")
    painter.begin\_fill()

    \# Rectangle draw    (100x80)
    for \_ in range(2):
        painter.forward(100)
        painter.right(90)
        painter.forward(80)
        painter.right(90)

    painter.end\_fill()

    \# Labels
    painter.penup()
    painter.goto({-}150, {-}120)
    painter.write("   Circle", align="center", font=("Arial", 14, "normal"))

    painter.goto(100, {-}50)
    painter.write("   Rectangle", align="center", font=("Arial", 14, "normal"))

    \# Turtle hide      result display
    painter.hideturtle()
    print("Shapes       draw   !")

    \# Window open
    screen.exitonclick()

\#      run
draw\_filled\_shapes()
```

- **Fill Process**: begin_fill() અને end_fill() વચ્ચે drawn shape automatically fill થાય છે
- **Color Setting**: fillcolor() method fill color set કરે છે

- **Shape Drawing**: Geometric shapes માટે specific turtle movements

મેમરી ટ્રીક

``Begin Fill Draw End''

પ્રશ્ન 5(બ) [4 ગુણ]

ટર્ટલની Direction બદલવાની વિવિધ inbuilt પદ્ધતિઓ સમજાવો.

જવાબ

Table 19: Turtle Direction મેથડો

મેથડ	Parameters	વર્ણન	ઉદાહરણ
right()	angle	Degrees માં જમણે turn કરે છે	`turtle.right(90)`
left()	angle	Degrees માં ડાબે turn કરે છે	`turtle.left(45)`
setheading()	angle	Absolute direction set કરે છે	`turtle.setheading(0)`
towards()	x, y	Coordinates તરફ point કરે છે	`turtle.towards(100, 50)`
home()	none	Center પર return કરે, પૂર્વ તરફ face કરે	`turtle.home()`

```
import turtle

def demonstrate\_direction\_methods():
    screen = turtle.Screen()
    screen.setup(600, 600)
    screen.title("Turtle Direction   ")

    t = turtle.Turtle()
    t.speed(2)
    t.shape("turtle")

    \# 1. right()
    t.write("1. right(90)", font=("Arial", 10, "normal"))
    t.forward(50)
    t.right(90)
    t.forward(50)

    \# 2. left()
    t.penup()
    t.goto({-}100, 100)
    t.pendown()
    t.write("2. left(45)", font=("Arial", 10, "normal"))
    t.forward(50)
    t.left(45)
    t.forward(50)

    \# 3. setheading()
    t.penup()
    t.goto(100, 100)
    t.pendown()
    t.write("3. setheading(180)", font=("Arial", 10, "normal"))
    t.setheading(180)  \#       face
    t.forward(50)

    \# 4. towards()
    t.penup()
    t.goto({-}100, {-}100)
    t.pendown()
    target\_x, target\_y = 100, {-}100
    t.write("4. towards(100,{-100)"}, font=("Arial", 10, "normal"))
    angle = t.towards(target\_x, target\_y)
    t.setheading(angle)
    t.goto(target\_x, target\_y)

    \# 5. home()
    t.write("5. home()", font=("Arial", 10, "normal"))
    t.home()  \# Center   return           face

    t.hideturtle()
    screen.exitonclick()

demonstrate\_direction\_methods()
```

- **Relative Turns**: right() અને left() current direction થી relative turn કરે છે
- **Absolute Direction**: setheading() absolute compass direction set કરે છે
- **Smart Pointing**: towards() specific coordinates તરફ point કરે છે

મેમરી ટ્રીક

``Right Left Set Towards Home''

પ્રશ્ન 5(ક) [7 ગુણ]

ટર્ટલનો ઉપયોગ કરીને rainbow દોરવા માટે પાયથન પ્રોગ્રામ લખો.

જવાબ

Rainbow drawing માં multiple colored arcs અને proper positioning જરૂરી છે.

ડાયાગ્રામ: Rainbow Structure

```
    Red (  )
   Orange
  Yellow
 Green
Blue
Indigo
Violet (   )
```

```
import turtle

def draw\_rainbow():
    \# Screen setup
    screen = turtle.Screen()
    screen.title("   Rainbow")
    screen.bgcolor("lightblue")
    screen.setup(800, 600)

    \# Turtle setup
    rainbow\_turtle = turtle.Turtle()
    rainbow\_turtle.speed(8)
    rainbow\_turtle.pensize(8)

    \# Rainbow colors (ROYGBIV)
    colors = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"]

    \# Rainbow parameters
    start\_radius = 200
    radius\_decrease = 15
    start\_y = {-}150

    print("Rainbow draw         ...")

    \#    color arc draw
    for i, color in enumerate(colors):
        \# Color set
        rainbow\_turtle.pencolor(color)

        \# Current radius calculate
        current\_radius = start\_radius {-} (i * radius\_decrease)

        \# Semi{-circle    turtle position   }
        rainbow\_turtle.penup()
        rainbow\_turtle.goto({-}current\_radius, start\_y)
        rainbow\_turtle.pendown()
        rainbow\_turtle.setheading(0)  \#      face

        \# Semi{-circle draw    (180 degrees)}
        rainbow\_turtle.circle(current\_radius, 180)

        print(f"\{color\} arc radius \{current\_radius\}    draw    ")

    \# Rainbow     clouds
    draw\_clouds(rainbow\_turtle)

    \# Sun
```

```
    draw\_sun(rainbow\_turtle)

    \# Text
    rainbow\_turtle.penup()
    rainbow\_turtle.goto(0, {-}250)
    rainbow\_turtle.pencolor("black")
    rainbow\_turtle.write("    Rainbow  ", align="center",
                         font=("Arial", 16, "bold"))

    rainbow\_turtle.hideturtle()
    print("Rainbow     !")

    screen.exitonclick()

def draw\_clouds(turtle\_obj):
    """Rainbow       clouds draw    """
    turtle\_obj.pensize(3)
    turtle\_obj.pencolor("white")
    turtle\_obj.fillcolor("lightgray")

    \# Clouds   positions
    cloud\_positions = [({-}250, {-}100), (250, {-}100)]

    for x, y in cloud\_positions:
        turtle\_obj.penup()
        turtle\_obj.goto(x, y)
        turtle\_obj.pendown()

        \# Multiple circles     cloud draw
        turtle\_obj.begin\_fill()
        for i in range(3):
            turtle\_obj.circle(20)
            turtle\_obj.left(120)
        turtle\_obj.end\_fill()

def draw\_sun(turtle\_obj):
    """Corner   sun draw    """
    turtle\_obj.penup()
    turtle\_obj.goto(300, 200)
    turtle\_obj.pendown()
    turtle\_obj.pencolor("orange")
    turtle\_obj.fillcolor("yellow")

    \# Sun body draw
    turtle\_obj.begin\_fill()
    turtle\_obj.circle(30)
    turtle\_obj.end\_fill()

    \# Sun rays draw
    turtle\_obj.pensize(2)
    for angle in range(0, 360, 45):
        turtle\_obj.setheading(angle)
        turtle\_obj.forward(45)
        turtle\_obj.backward(45)

\# Alternative rainbow with gradient effect
def draw\_gradient\_rainbow():
    screen = turtle.Screen()
    screen.title("Gradient Rainbow")
    screen.bgcolor("skyblue")
    screen.setup(800, 600)
```

```
    t = turtle.Turtle()
    t.speed(0)
    t.pensize(5)

    \# Gradient effect    color variations
    rainbow\_colors = [
        "\#FF0000", "\#FF4500", "\#FFD700", "\#32CD32",
        "\#0000FF", "\#4B0082", "\#8B00FF"
    ]

    \# Varying thickness    rainbow draw
    for i, color in enumerate(rainbow\_colors):
        t.pencolor(color)
        t.pensize(12 {-} i)  \# Decreasing thickness

        radius = 150 {-} (i * 10)
        t.penup()
        t.goto({-}radius, {-}100)
        t.pendown()
        t.setheading(0)
        t.circle(radius, 180)

    t.hideturtle()
    screen.exitonclick()

\# Rainbow programs run
print("Rainbow         :")
print("1. Standard Rainbow")
print("2. Gradient Rainbow")

choice = input("          (1    2): ")
if choice == "2":
    draw\_gradient\_rainbow()
else:
    draw\_rainbow()
```

- **Color Sequence**: ROYGBIV (Red Orange Yellow Green Blue Indigo Violet) નો proper order
- **Radius Management**: દરેક arc નો radius gradually decrease કરાય છે
- **Positioning**: Proper positioning માટે penup/pendown અને goto methods

મેમરી ટ્રીક

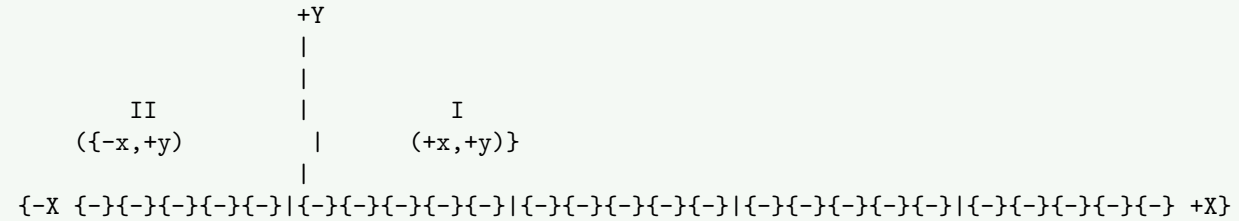
``ROYGBIV Arc Radius Position''

પ્રશ્ન 5(અ OR) [3 ગુણ]

ટર્ટલ સ્ક્રીનનો ડાયાગ્રામ દોરો અને x અને y કોઓર્ડિનેટ્સના તમામ 4 quadrants સમજાવો.

જવાબ

ડાયાગ્રામ: Turtle Coordinate System



```
       {-200  {-}100   0   100  200}
                     |
        III          |          IV
    ({-x,{-}y)          |       (+x,{-}y)}
                     |
                     |
                    {-Y}

Default Screen: 400x300 pixels
Center: (0, 0)
```

Table 20: Coordinate Quadrants

Quadrant	X Value	Y Value	વર્ણન	ઉદાહરણ
I	Positive (+)	Positive (+)	ઉપર-જમણે	(100, 50)
II	Negative (-)	Positive (+)	ઉપર-ડાબે	(-100, 50)
III	Negative (-)	Negative (-)	નીચે-ડાબે	(-100, -50)
IV	Positive (+)	Negative (-)	નીચે-જમણે	(100, -50)

```
import turtle

def demonstrate\_coordinate\_system():
    screen = turtle.Screen()
    screen.title("Turtle Coordinate System")
    screen.setup(600, 500)
    screen.bgcolor("white")

    t = turtle.Turtle()
    t.speed(3)
    t.shape("turtle")

    \# Coordinate axes draw
    t.pencolor("gray")
    t.pensize(2)

    \# X{-axis}
    t.penup()
    t.goto({-}250, 0)
    t.pendown()
    t.goto(250, 0)

    \# Y{-axis}
    t.penup()
    t.goto(0, {-}200)
    t.pendown()
    t.goto(0, 200)

    \# Center mark
    t.penup()
    t.goto(0, 0)
    t.dot(8, "red")
    t.write("(0,0)", font=("Arial", 12, "normal"))

    \#    quadrant demonstrate
    quadrants = [
        (100, 100, "I", "red"),     \# Quadrant I
        ({-}100, 100, "II", "blue"),  \# Quadrant II
        ({-}100, {-}100, "III", "green"), \# Quadrant III
        (100, {-}100, "IV", "orange")   \# Quadrant IV
    ]

    for x, y, quad\_name, color in quadrants:
        t.pencolor(color)
        t.penup()
        t.goto(x, y)
        t.pendown()
        t.dot(10, color)
        t.write(f"Q\{quad\_name\}{n}(\{x\},\{y\})", align="center",
                font=("Arial", 10, "bold"))

    t.hideturtle()
    screen.exitonclick()

demonstrate\_coordinate\_system()
```

- **Origin**: (0,0) screen ના center પર આવેલું છે
- **Positive Direction**: X-axis જમણે તરફ, Y-axis ઉપર તરફ positive
- **Navigation**: goto(x, y) method specific coordinates પર move કરે છે

મેમરી ટ્રીક

``Right Up Positive, Left Down Negative''

પ્રશ્ન 5(બ OR) [4 ગુણ]

Background color, title, screensize અને shapesize ને બદલવા માટે વિવિધ ટર્ટલ સ્ક્રીન પદ્ધતિઓનું વર્ણન કરો.

જવાબ

Table 21: Turtle Screen મેથડો

મેથડ	હેતુ	Parameters	ઉદાહરણ
bgcolor()	Background color set કરવા	color name/hex	`screen.bgcolor("blue")`
title()	Window title set કરવા	string	`screen.title("My Program")`
setup()	Screen size set કરવા	width, height	`screen.setup(800, 600)`
screensize()	Canvas size set કરવા	width, height	`screen.screensize(400, 300)`
shapesize()	Turtle size set કરવા	stretch_wid, stretch_len	`turtle.shapesize(2, 3)`

```
import turtle

def demonstrate\_screen\_methods():
    \# Screen object
    screen = turtle.Screen()

    \# 1. Title
    screen.title("Screen Methods Demonstration")
    print("  Title set   : {Screen Methods Demonstration"})

    \# 2. Background Color
    screen.bgcolor("lightgreen")
    print("  Background color set    : lightgreen")

    \# 3. Setup    (window size)
    screen.setup(width=800, height=600)
    print("  Window size set   : 800x600 pixels")

    \# 4. Screen Size    (canvas size)
    screen.screensize(canvwidth=400, canvheight=300)
    print("  Canvas size set   : 400x300")

    \# Shapesize demonstrate       turtle
    demo\_turtle = turtle.Turtle()
    demo\_turtle.speed(3)

    \# 5. Shape Size
    demo\_turtle.shape("turtle")
    demo\_turtle.shapesize(stretch\_wid=3, stretch\_len=2, outline=3)
    print("  Turtle shape size: width=3x, length=2x, outline=3")

    \#     background colors demonstrate
    colors = ["lightblue", "lightyellow", "lightpink", "lightcoral"]

    for i, color in enumerate(colors):
        screen.bgcolor(color)
        demo\_turtle.write(f"Background: \{color\}",
                          font=("Arial", 14, "normal"))
        demo\_turtle.forward(50)
        demo\_turtle.right(90)
        screen.ontimer(lambda: None, 1000)  \# 1

    \# Final state   reset
    screen.bgcolor("white")
    demo\_turtle.penup()
    demo\_turtle.goto(0, {-}50)
    demo\_turtle.write("Screen Methods Demo   !",
                      align="center", font=("Arial", 16, "bold"))

    demo\_turtle.hideturtle()
    screen.exitonclick()

def advanced\_screen\_customization():
    """Advanced screen customization    """
    screen = turtle.Screen()

    \#    parameters    advanced setup
    screen.setup(width=0.8, height=0.8, startx=100, starty=50)
    screen.title("  Advanced Turtle Graphics  ")
    screen.bgcolor("\#2E8B57")  \# Sea Green
```

```
    \# Custom color palette
    screen.colormode(255)  \# RGB mode enable

    \#     sizes    multiple turtles
    turtles = []
    shapes = ["turtle", "circle", "square", "triangle"]
    sizes = [(1, 1), (2, 1), (1, 2), (3, 3)]
    colors = [(255, 0, 0), (0, 255, 0), (0, 0, 255), (255, 255, 0)]

    for i in range(4):
        t = turtle.Turtle()
        t.shape(shapes[i])
        t.shapesize(sizes[i][0], sizes[i][1])
        t.color(colors[i])
        t.penup()
        t.goto({-}150 + i*100, 0)
        turtles.append(t)

        \#    turtle   label
        t.write(f"\{shapes[i]\}{n}\{sizes[i]\}",
               align="center", font=("Arial", 10, "normal"))

    \# Instructions
    instruction\_turtle = turtle.Turtle()
    instruction\_turtle.hideturtle()
    instruction\_turtle.penup()
    instruction\_turtle.goto(0, {-}100)
    instruction\_turtle.color("white")
    instruction\_turtle.write("    turtle shapes    sizes",
                          align="center", font=("Arial", 16, "bold"))

    screen.exitonclick()

\# Demonstrations run
print("Screen Methods Demo run         ...")
demonstrate\_screen\_methods()

print("{n}Advanced Customization run         ...")
advanced\_screen\_customization()
```

- **Window vs Canvas**: setup() window size, screensize() canvas size control કરે છે
- **Color Modes**: bgcolor() color names અથવા hex values accept કરે છે
- **Shape Scaling**: shapesize() turtle appearance ને scale કરે છે

મેમરી ટ્રીક

``Title Background Setup Size Shape''

પ્રશ્ન 5(ક OR) [7 ગુણ]

ટર્ટલનો ઉપયોગ કરીને સ્ટાર, ત્રિકોણ અને અષ્ટકોણ દોરવા માટે પાયથન પ્રોગ્રામ લખો.

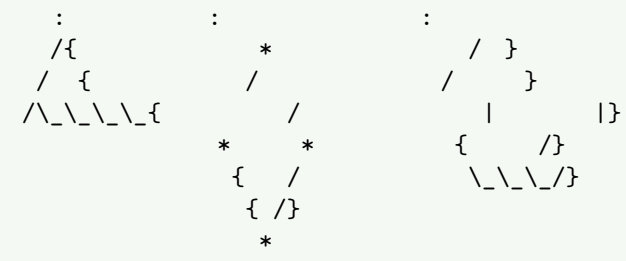
જવાબ

Geometric shapes drawing માં angles અને sides ની proper calculation જરૂરી છે.

Table 22: Shape Properties

આકાર	બાજુઓ	External Angle	Internal Angle	Turn Angle
ત્રિકોણ	3	120°	60°	120°
સ્ટાર (5-point)	5	144°	36°	144°
અષ્ટકોણ	8	45°	135°	45°

ડાયાગ્રામ: Shape Construction



```
import turtle
import math

def draw\_geometric\_shapes():
    \# Screen setup
    screen = turtle.Screen()
    screen.title("Geometric Shapes:   ,    ,    ")
    screen.bgcolor("black")
    screen.setup(900, 600)

    \# Turtle setup
    artist = turtle.Turtle()
    artist.speed(6)
    artist.pensize(3)

    \#    1:
    draw\_triangle(artist, {-}250, 100, 80, "cyan")

    \#    2:  {-         }
    draw\_star(artist, 0, 100, 80, "yellow")

    \#    3:
    draw\_octagon(artist, 250, 100, 60, "magenta")

    \# Labels
    add\_labels(artist)

    artist.hideturtle()
    print("               !")
    screen.exitonclick()

def draw\_triangle(turtle\_obj, x, y, size, color):
    """           """
    print(f"    (\{x\}, \{y\})   draw        ")

    turtle\_obj.penup()
    turtle\_obj.goto(x, y)
    turtle\_obj.pendown()
    turtle\_obj.color(color)
    turtle\_obj.fillcolor(color)

    turtle\_obj.begin\_fill()
    for \_ in range(3):
        turtle\_obj.forward(size)
        turtle\_obj.left(120)  \#        external angle
    turtle\_obj.end\_fill()

def draw\_star(turtle\_obj, x, y, size, color):
    """  {-           """}
    print(f"   (\{x\}, \{y\})   draw        ")
```

```
    turtle\_obj.penup()
    turtle\_obj.goto(x, y)
    turtle\_obj.pendown()
    turtle\_obj.color(color)
    turtle\_obj.fillcolor(color)

    turtle\_obj.begin\_fill()
    for \_ in range(5):
        turtle\_obj.forward(size)
        turtle\_obj.right(144)  \# 5{-pointed       144^ turn}
    turtle\_obj.end\_fill()

def draw\_octagon(turtle\_obj, x, y, size, color):
    """            """
    print(f"    (\{x\}, \{y\})   draw        ")

    turtle\_obj.penup()
    turtle\_obj.goto(x, y)
    turtle\_obj.pendown()
    turtle\_obj.color(color)
    turtle\_obj.fillcolor(color)

    turtle\_obj.begin\_fill()
    for \_ in range(8):
        turtle\_obj.forward(size)
        turtle\_obj.right(45)  \#        360^/8 = 45^
    turtle\_obj.end\_fill()

def add\_labels(turtle\_obj):
    """         labels    """
    turtle\_obj.color("white")

    labels = [
        ({-}250, 30, "   {n}3   {n}120^ turns"),
        (0, 30, "  {n}5   {n}144^ turns"),
        (250, 30, "   {n}8   {n}45^ turns")
    ]

    for x, y, text in labels:
        turtle\_obj.penup()
        turtle\_obj.goto(x, y)
        turtle\_obj.write(text, align="center", font=("Arial", 12, "normal"))

def draw\_advanced\_shapes():
    """Animations   multiple variations   advanced version"""
    screen = turtle.Screen()
    screen.title("Advanced Geometric Shapes")
    screen.bgcolor("navy")
    screen.setup(1000, 700)

    artist = turtle.Turtle()
    artist.speed(8)
    artist.pensize(2)

    \# Animated     variations
    triangle\_sizes = [40, 60, 80]
    triangle\_colors = ["red", "orange", "yellow"]

    for i, (size, color) in enumerate(zip(triangle\_sizes, triangle\_colors)):
        x = {-}300 + i * 30
        y = 200 {-} i * 20
```

```
        draw\_triangle(artist, x, y, size, color)

    \# Animated    variations
    star\_sizes = [30, 50, 70, 90]
    star\_colors = ["pink", "lightblue", "lightgreen", "gold"]

    for i, (size, color) in enumerate(zip(star\_sizes, star\_colors)):
        angle = i * 90
        x = 150 + math.cos(math.radians(angle)) * 80
        y = 100 + math.sin(math.radians(angle)) * 80

        artist.penup()
        artist.goto(x, y)
        artist.setheading(angle)
        artist.pendown()
        artist.color(color)
        artist.fillcolor(color)

        artist.begin\_fill()
        for \_ in range(5):
            artist.forward(size)
            artist.right(144)
        artist.end\_fill()

    \#     pattern
    for i in range(3):
        size = 40 + i * 15
        color\_intensity = 0.3 + i * 0.2
        draw\_octagon(artist, {-}100, {-}100 + i * 80, size,
                     (color\_intensity, 0, color\_intensity))

    \#
    artist.penup()
    artist.goto(0, {-}250)
    artist.color("white")
    artist.write("Geometric Shapes {-           "},
                 align="center", font=("Arial", 16, "bold"))

    artist.goto(0, {-}280)
    artist.write("   :          = 180^,    : 36^ points,     :      = 1080^",
                 align="center", font=("Arial", 12, "normal"))

    artist.hideturtle()
    screen.exitonclick()

def calculate\_shape\_properties():
    """           calculate   display    """
    shapes\_info = \{
        "   ": \{
            "  ": 3,
            "   \_  ": 180 * (3{-}2) / 3,
            "  \_  ": 360 / 3,
            "    \_    ": 180 * (3{-}2)
        \,}
        "   (5{-point)"}: \{
            "  ": 5,
            "point\_angle": 36,
            "turn\_angle": 144,
            " \_rotation": 720
        \,}
        "   ": \{
```

```
            "  ": 8,
            "   \_  ": 180 * (8{-}2) / 8,
            "  \_  ": 360 / 8,
            "   \_   ": 180 * (8{-}2)
        \}
    \}

    print("{n}" + "="*50)
    print("GEOMETRIC SHAPES {-           "})
    print("="*50)

    for shape, props in shapes\_info.items():
        print(f"{n}\{shape\}:")
        for prop, value in props.items():
            print(f"  \{prop.replace({\_}, { }).title()\}: \{value\}^" if {  } in prop else f"  \{prop.

\#      run
print("Drawing mode     :")
print("1. Basic Shapes")
print("2. Advanced Shapes with Variations")

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

if choice == "2":
    draw\_advanced\_shapes()
else:
    draw\_geometric\_shapes()

\#            display
calculate\_shape\_properties()
```

- **Angle Calculation**: દરેક shape માટે correct turn angles ની calculation જરૂરી
- **Fill Technique**: begin_fill() અને end_fill() વચ્ચે shape automatically fill થાય
- **Mathematical Foundation**: Geometry ના principles આધારે shapes construct થાય

મેમરી ટ્રીક

``Triangle 120, Star 144, Octagon 45''