Subject Name (Gujarati)

4321602 -- Winter 2024

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 માર્ક્સ]

પાયથનમાં સેટ અને ડિક્શનરી વચ્ચેનો તફાવત લખો.

જવાબ

લક્ષણ	સેટ	ડિક્શનરી
ડેટા સ્ટોરેજ	ફક્ત યુનિક એલિમેન્ટ્સ સ્ટોર કરે	કી-વેલ્યુ પેર સ્ટોર કરે
ક્રમ	અનઓર્ડર્ડ કલેક્શન	ઓર્ડર્ડ (Python 3.7+)
ડુપ્લિકેટ્સ	ડુપ્લિકેટ્સની મંજૂરી નથી	કીઝ યુનિક હોવી જોઈએ
એક્સેસ	ઈન્ડેક્સ દ્વારા એક્સેસ કરી શકાતું નથી	કીઝ દ્વારા વેલ્યુઝ એક્સેસ કરવા
સિન્ટેક્સ	`{1, 2, 3}`	`{'key': 'value'}`

- **સેટ**: યુનિક, અનઓર્ડર્ડ એલિમેન્ટ્સનો કલેક્શન
- **ડિક્શનરી**: યુનિક કીઝ સાથે કી-વેલ્યુ પેરનો કલેક્શન

મેમરી ટ્રીક

``સેટ્સ યુનિક, ડિક્ટ્સ કીઝ વાળા''

પ્રશ્ન 1(બ) [4 માર્ક્સ]

પાયથોનમાં લિસ્ટ ઉદાહરણ સાથે સમજાવો.

જવાબ

લિસ્ટ એક ઓર્ડર્ડ, મ્યુટેબલ કલેક્શન છે જે વિવિધ ડેટા ટાઈપ્સ સ્ટોર કરી શકે છે.
લિસ્ટ ઓપરેશન્સનું ટેબલ:

ઓપરેશન	સિન્ટેક્સ	ઉદાહરણ
બનાવવું	`list_name = []`	`fruits = ['apple', 'banana']`
એક્સેસ	`list[index]`	`fruits[0]` રિટર્ન `apple'
ઉમેરવું	`append()`	`fruits.append('orange')`
હટાવવું	`remove()`	`fruits.remove('apple')`

```
\#
numbers = [1, 2, 3, 4, 5]
numbers.append(6)  \# [1, 2, 3, 4, 5, 6]
print(numbers[0])  \#    : 1
```

- **ઓર્ડર્ડ**: એલિમેન્ટ્સ તેમની પોઝિશન જાળવે છે
- **મ્યુટેબલ**: બનાવ્યા પછી મોડિફાઈ કરી શકાય છે
- **ફ્લેક્સિબલ**: કોઈપણ ડેટા ટાઈપ સ્ટોર કરે છે

મેમરી ટ્રીક

``લિસ્ટ્સ ઓર્ડર્ડ અને મોડિફાઈ કરી શકાય તેવી''

પ્રશ્ન 1(ક) [7 માર્ક્સ]

પાયથોનમાં ટપલ શું છે? બે ટપલ વેલ્યુને અદલાબદલી કરવાનો પાયથન પ્રોગ્રામ લખો.

જવાબ

ટપલ એક ઓર્ડર્ડ, ઈમ્યુટેબલ કલેક્શન છે જે મલ્ટિપલ આઈટમ્સ સ્ટોર કરે છે.
ટપલના લક્ષણોનું ટેબલ:

પ્રોપર્ટી	વર્ણન	ઉદાહરણ
ઈમ્યુટેબલ	બનાવ્યા પછી બદલી શકાતું નથી	`t = (1, 2, 3)`
ઓર્ડર્ડ	એલિમેન્ટ્સનો નિર્ધારિત ક્રમ	ઈન્ડેક્સ દ્વારા એક્સેસ
ડુપ્લિકેટ્સ	ડુપ્લિકેટ વેલ્યુઝની મંજૂરી	`(1, 1, 2)`
ઈન્ડેક્સિંગ	પોઝિશન દ્વારા એલિમેન્ટ્સ એક્સેસ	`t[0]`

```
\#
def swap\_tuple\_values(tup, pos1, pos2):
    \#
    temp\_list = list(tup)

    \#
    temp\_list[pos1], temp\_list[pos2] = temp\_list[pos2], temp\_list[pos1]

    \#
    return tuple(temp\_list)

\#
original\_tuple = (10, 20, 30, 40, 50)
print("    :", original\_tuple)

\#     1   3
swapped\_tuple = swap\_tuple\_values(original\_tuple, 1, 3)
print("     :", swapped\_tuple)
```

- **ઈમ્યુટેબલ**: એકવાર બનાવ્યા પછી મોડિફાઈ કરી શકાતું નથી
- **ઓર્ડર્ડ**: એલિમેન્ટ સિક્વન્સ જાળવે છે
- **હેટેરોજીનિયસ**: વિવિધ ડેટા ટાઈપ્સ સ્ટોર કરી શકે છે

મેમરી ટ્રીક

``ટપલ્સ ઈમ્યુટેબલ અને ઓર્ડર્ડ''

પ્રશ્ન 1(ક OR) [7 માર્ક્સ]

પાયથોનમાં ડિક્શનરી શું છે? લૂપની મદદથી ડિક્શનરીને ટ્રાવર્સ કરવાનો પાયથન પ્રોગ્રામ લખો.

જવાબ

ડિક્શનરી એક યુનિક કીઝ સાથે કી-વેલ્યુ પેરનો અનઓર્ડર્ડ કલેક્શન છે.
ડિક્શનરી મેથડ્સનું ટેબલ:

મેથડ	હેતુ	ઉદાહરણ
keys()	બધી કીઝ મેળવો	`dict.keys()`
values()	બધી વેલ્યુઝ મેળવો	`dict.values()`
items()	કી-વેલ્યુ પેર મેળવો	`dict.items()`
get()	સેફ કી એક્સેસ	`dict.get('key')`

```
\#
student\_marks = \{
    {Alice}: 85,
    {Bob}: 92,
    {Charlie}: 78,
    {Diana}: 96,
    {Eve}: 89
\}

print("            :")
print("{-"} * 30)

\#    1:
print("1.      :")
for key in student\_marks:
    print(f"    \{key\}")

\#    2:
print("{n}2.       :")
for value in student\_marks.values():
    print(f"    \{value\}")

\#    3:  {-           }
print("{n}3.  {-    :"})
for key, value in student\_marks.items():
    print(f"    \{key\}: \{value\}")

\#    4: keys()
print("{n}4. keys()        :")
for key in student\_marks.keys():
    print(f"    \{key\}   \{student\_marks[key]\}        ")
```

- **કી-વેલ્યુ સ્ટોરેજ**: દરેક કી એક વેલ્યુ સાથે મેપ થાય છે
- **યુનિક કીઝ**: ડુપ્લિકેટ કીઝની મંજૂરી નથી
- **ફાસ્ટ લુકઅપ**: O(1) એવરેજ ટાઈમ કોમ્પ્લેક્સિટી

મેમરી ટ્રીક

``ડિક્ટ્સ કીઝને વેલ્યુઝ સાથે મેપ કરે''

પ્રશ્ન 2(અ) [3 માર્ક્સ]

પેકેજ શું છે? પેકેજનો ઉપયોગ કરવાના ફાયદાઓની યાદી આપો.

જવાબ

પેકેજ એક ડિરેક્ટરી છે જેમાં મલ્ટિપલ મોડ્યુલ્સ એકસાથે ઓર્ગેનાઈઝ કરવામાં આવે છે.
પેકેજના ફાયદાઓનું ટેબલ:

ફાયદો	વર્ણન
ઓર્ગેનાઈઝેશન	સંબંધિત મોડ્યુલ્સને એકસાથે ગ્રુપ કરે
નેમસ્પેસ	નામિંગ કોન્ફ્લિક્ટ્સ ટાળે
રીયુઝેબિલિટી	કોડ પ્રોજેક્ટ્સમાં ફરીથી વાપરી શકાય
મેઈન્ટેનેબિલિટી	મોટા કોડબેસ મેનેજ કરવું સરળ
ડિસ્ટ્રિબ્યુશન	શેર કરવું અને ઈન્સ્ટોલ કરવું સરળ

- **મોડ્યુલર સ્ટ્રક્ચર**: વધુ સારું કોડ ઓર્ગેનાઈઝેશન
- **હાયરાર્કિકલ નેમસ્પેસ**: નેમ કોન્ફ્લિક્ટ્સ અટકાવે
- **કોડ રીયુઝ**: સોફ્ટવેર રીયુઝેબિલિટીને પ્રમોટ કરે

મેમરી ટ્રીક

``પેકેજિસ રિલેટેડ મોડ્યુલ્સ ઓર્ગેનાઈઝ કરે''

પ્રશ્ન 2(બ) [4 માર્ક્સ]

કોઈપણ બે પેકેજ આયાત પદ્ધતિઓ ઉદાહરણો સાથે સમજાવો.

જવાબ

આયાત મેથડ્સનું ટેબલ:

મેથડ	સિન્ટેક્સ	ઉપયોગ
નોર્મલ આયાત	`import package.module`	ફુલ પાથ સાથે એક્સેસ
ફ્રમ આયાત	`from package import module`	ડાયરેક્ટ મોડ્યુલ એક્સેસ
સ્પેસિફિક આયાત	`from package.module import function`	સ્પેસિફિક આઈટમ્સ આયાત
વાઈલ્ડકાર્ડ આયાત	`from package import *`	બધા મોડ્યુલ્સ આયાત

```
\#    1:
import mypackage.calculator
result = mypackage.calculator.add(5, 3)
print(f"          : \{result\}")

\#    2:
from mypackage import calculator
result = calculator.multiply(4, 6)
print(f"         : \{result\}")
```

- **નોર્મલ આયાત**: ફુલ પેકેજ પાથ જરૂરી
- **ફ્રમ આયાત**: ડાયરેક્ટ મોડ્યુલ એક્સેસની મંજૂરી
- **સ્પેસિફિક ફંક્શન આયાત**: ફક્ત જરૂરી ફંક્શન્સ આયાત

મેમરી ટ્રીક

``આયાત નોર્મલી અથવા પેકેજથી''

પ્રશ્ન 2(ક) [7 માર્ક્સ]

ઈન્ટ્રા-પેકેજ સંદર્ભ વિશે ઉદાહરણ સાથે સમજાવો.

જવાબ

ઈન્ટ્રા-પેકેજ રેફરન્સ પેકેજની અંદરના મોડ્યુલ્સને એકબીજાથી આયાત કરવાની મંજૂરી આપે છે.
પેકેજ સ્ટ્રક્ચરનો ડાયાગ્રામ:

```
mypackage/
  \_\_init\_\_.py
  math\_ops/
      \_\_init\_\_.py
      basic.py
      advanced.py
  utils/
       \_\_init\_\_.py
       helpers.py
```

રેફરન્સ ટાઈપ્સનું ટેબલ:

ટાઈપ	સિન્ટેક્સ	ઉપયોગ
એબ્સોલ્યુટ	`from mypackage.math_ops import basic`	પેકેજ રૂટથી ફુલ પાથ
રિલેટિવ	`from . import basic`	વર્તમાન પેકેજ
પેરન્ટ	`from .. import utils`	પેરન્ટ પેકેજ
સિબલિંગ	`from ..utils import helpers`	સિબલિંગ પેકેજ

```
\#
\# mypackage/math\_ops/advanced.py
from . import basic  \#
from ..utils import helpers  \#

def power\_operation(base, exp):
    \#
    if basic.is\_valid\_number(base) and basic.is\_valid\_number(exp):
        result = base ** exp
        \#
        return helpers.format\_result(result)
    return None

\# mypackage/math\_ops/basic.py
def is\_valid\_number(num):
    return isinstance(num, (int, float))

def add(a, b):
    return a + b

\# mypackage/utils/helpers.py
def format\_result(value):
    return f"    : \{value:.2f\}"
```

- **રિલેટિવ આયાત્સ**: વર્તમાન પેકેજ માટે ડોટ્સ (.) વાપરો
- **એબ્સોલ્યુટ આયાત્સ**: ફુલ પેકેજ પાથ
- **પેકેજ હાયરાર્કી**: ડોટ નોટેશન વાપરીને નેવિગેટ કરો

મેમરી ટ્રીક

``ડોટ્સ પેકેજ લેવલ્સ નેવિગેટ કરે''

પ્રશ્ન 2(અ OR) [3 માર્ક્સ]

મોડ્યુલ શું છે? મોડ્યુલનો ઉપયોગ કરવાના ફાયદાઓની યાદી આપો.

જવાબ

મોડ્યુલ એક Python ફાઈલ છે જેમાં ડેફિનિશન્સ, સ્ટેટમેન્ટ્સ અને ફંક્શન્સ હોય છે.
મોડ્યુલના ફાયદાઓનું ટેબલ:

ફાયદો	વર્ણન
કોડ રીયુઝેબિલિટી	એકવાર લખો, અનેક વાર વાપરો
નેમસ્પેસ	ફંક્શન્સ માટે અલગ નેમસ્પેસ
ઓર્ગેનાઈઝેશન	વધુ સારું કોડ સ્ટ્રક્ચર
મેઈન્ટેનેબિલિટી	ડિબગ અને અપડેટ કરવું સરળ
કોલેબોરેશન	મલ્ટિપલ ડેવલપર્સ કામ કરી શકે

- **રીયુઝેબલ કોડ**: ફંક્શન્સ ગમે ત્યાં આયાત કરી શકાય
- **મોડ્યુલર ડિઝાઈન**: મોટા પ્રોગ્રામ્સને નાના ભાગોમાં વહેંચો
- **સરળ મેઈન્ટેનન્સ**: એક જગ્યાએ ફેરફાર બધી આયાત્સને અસર કરે

મેમરી ટ્રીક

``મોડ્યુલ્સ કોડ રીયુઝેબલ બનાવે''

પ્રશ્ન 2(બ OR) [4 માર્ક્સ]

કોઈપણ બે મોડ્યુલ આયાત પદ્ધતિ ઉદાહરણ સાથે સમજાવો.

જવાબ

મોડ્યુલ આયાત મેથડ્સનું ટેબલ:

મેથડ	સિન્ટેક્સ	એક્સેસ પેટર્ન
ડાયરેક્ટ આયાત	`import module_name`	`module_name.function()`
ફ્રમ આયાત	`from module_name import function`	`function()`
એલિયાસ આયાત	`import module_name as alias`	`alias.function()`
વાઈલ્ડકાર્ડ આયાત	`from module_name import *`	`function()`

```
\#    1:
import math
result1 = math.sqrt(16)
print(f"       : \{result1\}")

\#    2:
from math import pi, sin
result2 = sin(pi/2)
print(f"     : \{result2\}")
```

- **ડાયરેક્ટ આયાત**: મોડ્યુલ નામ પ્રીફિક્સ સાથે એક્સેસ
- **ફ્રમ આયાત**: પ્રીફિક્સ વગર ડાયરેક્ટ ફંક્શન એક્સેસ
- **નેમસ્પેસ કંટ્રોલ**: યોગ્ય આયાત મેથડ પસંદ કરો

મેમરી ટ્રીક

``આયાત ડાયરેક્ટલી અથવા મોડ્યુલથી''

પ્રશ્ન 2(ક OR) [7 માર્ક્સ]

વતુળનું ક્ષેત્રફળ અને પરિઘ શોધવા માટેના મોડ્યુલનો પ્રોગ્રામ લખો.

જવાબ

```
\# circle\_operations.py (      )
import math

def area(radius):
    """          """
    if radius {=} 0:
        return 0
    return math.pi * radius * radius

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

```
    if radius {=} 0:
        return 0
    return 2 * math.pi * radius

def display\_info(radius):
    """            """
    print(f"    \{radius\}     :")
    print(f"    : \{area(radius):.2f\}")
    print(f"   : \{circumference(radius):.2f\}")

\#
PI = math.pi

\#  )

\# main\_program.py
import circle\_operations

radius = 5
print("   1:          ")
area\_result = circle\_operations.area(radius)
circumference\_result = circle\_operations.circumference(radius)

print(f"    : \{area\_result:.2f\}")
print(f"   : \{circumference\_result:.2f\}")

\#  )
\# specific\_import.py
from circle\_operations import area, circumference

radius = 7
print("{n}   2:          ")
area\_result = area(radius)
circumference\_result = circumference(radius)

print(f"    : \{area\_result:.2f\}")
print(f"   : \{circumference\_result:.2f\}")
```

મોડ્યુલ ફીચર્સનું ટેબલ:

ફીચર	ઇમ્પ્લિમેન્ટેશન
ફંક્શન્સ	area(), circumference()
એરર હેન્ડલિંગ	નેગેટિવ ત્રિજ્યા માટે ચેક
કોન્સ્ટન્ટ્સ	PI વેલ્યુ
ડોક્યુમેન્ટેશન	ફંક્શન ડોકસ્ટ્રિંગ્સ

- **મોડ્યુલ બનાવવું**: ફંક્શન્સને .py ફાઈલમાં સેવ કરો
- **આયાત લવચીકતા**: સંપૂર્ણ મોડ્યુલ અથવા ચોક્કસ ફંક્શન્સ
- **કોડ રીયુઝ**: એક જ ફંક્શન્સ મલ્ટિપલ પ્રોગ્રામ્સમાં વાપરો

મેમરી ટ્રીક

``મોડ્યુલ્સમાં રીયુઝેબલ ફંક્શન્સ હોય''

પ્રશ્ન 3(અ) [3 માર્ક્સ]

પાયથોનમાં ભૂલના પ્રકારો સમજાવો.

જવાબ

Python એરર ટાઈપ્સનું ટેબલ:

એરર ટાઈપ	વર્ણન	ઉદાહરણ
સિન્ટેક્સ એરર	ખોટું Python સિન્ટેક્સ	કોલન : ભૂલી જવું
રનટાઈમ એરર	એક્ઝિક્યુશન દરમિયાન થાય	શૂન્યથી ભાગાકાર
લોજિકલ એરર	ખોટું પ્રોગ્રામ લોજિક	ખોટું અલ્ગોરિધમ
નેમ એરર	અંડિફાઈન્ડ વેરિએબલ	અઘોષિત વેરિએબલ વાપરવું
ટાઈપ એરર	ખોટું ડેટા ટાઈપ ઓપરેશન	સ્ટ્રિંગ + ઇન્ટિજર

- **સિન્ટેક્સ એરર**: પ્રોગ્રામ રન થાય તે પહેલાં શોધાય
- **રનટાઈમ એરર**: પ્રોગ્રામ એક્ઝિક્યુશન દરમિયાન થાય
- **લોજિકલ એરર**: પ્રોગ્રામ રન થાય પણ ખોટા પરિણામ આપે

મેમરી ટ્રીક

``સિન્ટેક્સ, રનટાઈમ, લોજિક એરર્સ''

પ્રશ્ન 3(બ) [4 માર્ક્સ]

યુઝર-ડિફાઈન્ડ એક્સેપ્શન રેઈઝ સ્ટેટમેન્ટ સાથે સમજાવો.

જવાબ

યુઝર-ડિફાઈન્ડ એક્સેપ્શન્સ પ્રોગ્રામર્સ દ્વારા બનાવવામાં આવેલા કસ્ટમ એરર ક્લાસીસ છે.
એક્સેપ્શન કોમ્પોનન્ટ્સનું ટેબલ:

કોમ્પોનન્ટ	હેતુ	ઉદાહરણ
ક્લાસ ડેફિનિશન	કસ્ટમ એક્સેપ્શન બનાવો	`class CustomError(Exception):`
રેઈઝ સ્ટેટમેન્ટ	એક્સેપ્શન ટ્રિગર કરો	`raise CustomError("message")`
એરર મેસેજ	સમસ્યાનું વર્ણન	માહિતીપ્રદ ટેક્સ્ટ
એક્સેપ્શન હેન્ડલિંગ	કસ્ટમ એક્સેપ્શન પકડો	`except CustomError:`

```
\#
class AgeValidationError(Exception):
    def \_\_init\_\_(self, age, message="         "):
        self.age = age
        self.message = message
        super().\_\_init\_\_(self.message)

def validate\_age(age):
    if age {} 0:
        raise AgeValidationError(age, "            ")
    elif age {} 150:
        raise AgeValidationError(age, "  150            ")
    else:
        print(f"    : \{age\}")

\#
try:
    validate\_age({-}5)
except AgeValidationError as e:
    print(f"  : \{e.message\},   : \{e.age\}")
```

- **કસ્ટમ એક્સેપ્શન ક્લાસ**: Exception થી ઇન્હેરિટ કરે
- **રેઈઝ સ્ટેટમેન્ટ**: મેન્યુઅલી એક્સેપ્શન્સ ટ્રિગર કરે
- **અર્થપૂર્ણ મેસેજિસ**: ડિબગિંગમાં મદદ કરે

મેમરી ટ્રીક

``વેલિડેશન માટે કસ્ટમ એક્સેપ્શન રેઈઝ કરો''

પ્રશ્ન 3(ક) [7 માર્ક્સ]

ટ્રાય-એક્સેપ્ટ-ફાઈનલી ક્લોઝ ઉદાહરણ સાથે સમજાવો.

જવાબ

ટ્રાય-એક્સેપ્ટ-ફાઈનલી સંપૂર્ણ એક્સેપ્શન હેન્ડલિંગ મિકેનિઝમ પૂરું પાડે છે.
એક્સેપ્શન હેન્ડલિંગ બ્લોક્સનું ટેબલ:

બ્લોક	હેતુ	એક્ઝિક્યુશન
try	એક્સેપ્શન ઉઠાવી શકે તેવો કોડ	હંમેશા પહેલા એક્ઝિક્યુટ
except	ચોક્કસ એક્સેપ્શન્સ હેન્ડલ કરે	ફક્ત એક્સેપ્શન આવે તો
else	કોઈ એક્સેપ્શન નહીં આવે ત્યારે	ફક્ત કોઈ એક્સેપ્શન નહીં આવે તો
finally	ક્લીનઅપ કોડ	હંમેશા એક્ઝિક્યુટ થાય

```
\#
def divide\_numbers():
    try:
        print("                  ...")

        \#
        num1 = float(input("           : "))
        num2 = float(input("           : "))

        \#
        result = num1 / num2

    except ValueError:
        print("  :                     ")
        return None

    except ZeroDivisionError:
        print("  :                  ")
        return None

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

    else:
        print(f"       : \{num1\}  \{num2\} = \{result\}")
        return result

    finally:
        print("           ")
        print("                 ...")

\#
result = divide\_numbers()
if result:
    print(f"       : \{result\}")
```

ફ્લો ડાયાગ્રામ:

```
flowchart LR
    A[try block] {-{-} B\{      ?\}}
    B {-{-}| | C[except block]}
    B {-{-}| | D[else block]}
    C {-{-} E[finally block]}
    D {-{-} E}
    E {-{-} F[ ]}
```

- **try**: જોખમી કોડ હોય છે
- **except**: ચોક્કસ એરર્સ હેન્ડલ કરે
- **finally**: ક્લીનઅપ માટે હંમેશા એક્ઝિક્યુટ થાય

મેમરી ટ્રીક

``ટ્રાય-એક્સેપ્ટ-ફાઈનલી હંમેશા ક્લીન કરે''

પ્રશ્ન 3(અ OR) [3 માર્ક્સ]

બિલ્ટ-ઇન એક્સેપ્શન શું છે? કોઇ પણ બેની તેમના અર્થ સાથે યાદી બનાવો.

જવાબ

બિલ્ટ-ઇન એક્સેપ્શન્સ Python માં પૂર્વ-નિર્ધારિત એરર ટાઈપ્સ છે.
બિલ્ટ-ઇન એક્સેપ્શન્સનું ટેબલ:

એક્સેપ્શન	અર્થ	ઉદાહરણ
ValueError	યોગ્ય ટાઈપ પણ અમાન્ય વેલ્યુ	`int("abc")`
TypeError	ખોટું ડેટા ટાઈપ ઓપરેશન	`"5" + 5`
IndexError	લિસ્ટ ઇન્ડેક્સ રેન્જની બહાર	5-આઇટમ લિસ્ટ માટે `list[10]`
KeyError	ડિક્શનરી કી મળી નહીં	`dict["missing_key"]`
ZeroDivisionError	શૂન્યથી ભાગાકાર	`10 / 0`

બે મુખ્ય બિલ્ટ-ઇન એક્સેપ્શન્સ:

- **ValueError**: જ્યારે ફંક્શનને યોગ્ય ટાઈપ પણ અમાન્ય વેલ્યુ મળે
- **TypeError**: જ્યારે અયોગ્ય ડેટા ટાઈપ પર ઓપરેશન કરવામાં આવે

મેમરી ટ્રીક

``બિલ્ટ-ઇન એક્સેપ્શન્સ સામાન્ય એરર્સ હેન્ડલ કરે''

પ્રશ્ન 3(બ OR) [4 માર્ક્સ]

ટ્રાય-એક્સેપ્ટ ક્લોઝ ઉદાહરણ સાથે સમજાવો.

જવાબ

ટ્રાય-એક્સેપ્ટ પ્રોગ્રામ એક્ઝિક્યુશન દરમિયાન આવી શકે તેવા એક્સેપ્શન્સ હેન્ડલ કરે છે.
એક્સેપ્શન હેન્ડલિંગનું ટેબલ:

કોમ્પોનન્ટ	હેતુ	સિન્ટેક્સ
try	નિષ્ફળ થઈ શકે તેવો કોડ	`try:`
except	ચોક્કસ એક્સેપ્શન હેન્ડલ કરે	`except ErrorType:`
મલ્ટિપલ except	વિવિધ એરર્સ હેન્ડલ કરે	મલ્ટિપલ except બ્લોક્સ
જનરલ except	કોઈપણ એક્સેપ્શન પકડે	`except:`

```
\#   {-          }
def safe\_division():
    try:
        \#
        dividend = int(input("        : "))
        divisor = int(input("        : "))

        result = dividend / divisor
        print(f"    : \{dividend\}  \{divisor\} = \{result\}")

    except ValueError:
        print("  :                      ")

    except ZeroDivisionError:
        print("  :                 ")

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

    print("                     ")

\#
safe\_division()
```

- **try બ્લોક**: સંભવિત જોખમી કોડ હોય છે
- **except બ્લોક**: ચોક્કસ એક્સેપ્શન ટાઈપ્સ હેન્ડલ કરે
- **મલ્ટિપલ હેન્ડલર્સ**: વિવિધ એક્સેપ્શન્સ અલગ અલગ હેન્ડલ

મેમરી ટ્રીક

``જોખમી કોડ ટ્રાય કરો, એક્સેપ્ટ એરર્સ હેન્ડલ કરે''

પ્રશ્ન 3(ક OR) [7 માર્ક્સ]

ડિવાઇડ બાય ઝીરો એક્સેપ્શન ફાઈનલી ક્લોઝ સાથે કેચ કરવાનો પ્રોગ્રામ લખો.

જવાબ

```
\#
def advanced\_calculator():
    """                  """

    try:
        print("===          ===")
        print("               ")

        \#
        numerator = float(input("       : "))
        denominator = float(input("       : "))

        print(f"{n}\{numerator\}   \{denominator\}                    ...")

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

        result = numerator / denominator

        \#
        print(f"        !")
```

```
        print(f"     : \{numerator\}  \{denominator\} = \{result:.4f\}")

        return result

    except ZeroDivisionError as zde:
        print(f"           : \{zde\}")
        print("                    ")
        return None

    except ValueError as ve:
        print(f"      :          ")
        print("                       ")
        return None

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

    finally:
        print("{n}" + "="*40)
        print("        :")
        print("{-                   "})
        print("{-                  "})
        print("{-                   "})
        print("{-            "})
        print("="*40)

\#
def test\_calculator():
    """                      """

    test\_cases = [
        (10, 2),    \#
        (15, 0),    \#
        (7, 3),     \#
    ]

    for i, (num, den) in enumerate(test\_cases, 1):
        print(f"{n}{-{-}{-}      }\{i\} {-{-}{-}"})
        print(f"   : \{num\}  \{den\}")

        \#
        \#
        print(f"\{num\}   \{den\}           ")

        try:
            if den == 0:
                raise ZeroDivisionError("                ")
            result = num / den
            print(f"    : \{result\}")
        except ZeroDivisionError as e:
            print(f"     : \{e\}")
        finally:
            print("      ")

\#
if \_\_name\_\_ == "\_\_main\_\_":
    result = advanced\_calculator()

    if result is not None:
        print(f"{n}         : \{result\}")
```

```
    else:
        print("{n}              ")
```

એક્સેપ્શન હેન્ડલિંગ ફીચર્સનું ટેબલ:

ફીચર	ઇમ્પ્લિમેન્ટેશન
ZeroDivisionError	શૂન્યથી ભાગાકાર માટે ચોક્કસ હેન્ડલિંગ
ValueError	અમાન્ય ઇનપુટ ટાઈપ્સ હેન્ડલ કરે
જનરિક એક્સેપ્શન	અનપેક્ષિત એરર્સ પકડે
ફાઈનલી બ્લોક	હંમેશા ક્લીનઅપ કોડ એક્ઝિક્યુટ

એક્સેપ્શન હેન્ડલિંગ ફ્લો:

```
flowchart LR
    A[      ] {-{-} B[try block]}
    B {-{-} C\{   ?\}}
    C {-{-}|ZeroDivisionError| D[          ]}
    C {-{-}|ValueError| E[         ]}
    C {-{-}|  | F[         ]}
    C {-{-}|    | G[        ]}
    D {-{-} H[finally block]}
    E {-{-} H}
    F {-{-} H}
    G {-{-} H}
    H {-{-} I[        ]}
```

- **ચોક્કસ એક્સેપ્શન હેન્ડલિંગ**: ZeroDivisionError અલગથી પકડાય
- **ફાઈનલી ક્લોઝ**: ક્લીનઅપ માટે હંમેશા એક્ઝિક્યુટ થાય
- **રિસોર્સ મેનેજમેન્ટ**: એરર્સ છતાં યોગ્ય ક્લીનઅપ

મેમરી ટ્રીક

``ફાઈનલી હંમેશા રિસોર્સિસ ક્લીન કરે''

પ્રશ્ન 4(અ) [3 માર્ક્સ]

વ્યાખ્યાયિત કરો: ફાઇલ, બાઇનરી ફાઇલ, ટેક્સ્ટ ફાઇલ

જવાબ

ફાઈલ વ્યાખ્યાઓનું ટેબલ:

શબ્દ	વ્યાખ્યા	ઉદાહરણ
ફાઈલ	ડિસ્ક પર નામવાળું સ્ટોરેજ સ્થાન	document.txt, image.jpg
બાઈનરી ફાઈલ	બાઈનરી ફોર્મેટમાં નોન-ટેક્સ્ટ ડેટા સમાવે	.exe, .jpg, .mp3, .pdf
ટેક્સ્ટ ફાઈલ	માનવ-વાંચી શકાય તેવા ટેક્સ્ટ કેરેક્ટર્સ સમાવે	.txt, .py, .html, .csv

વિગતવાર વ્યાખ્યાઓ:

- **ફાઈલ**: સ્ટોરેજ ડિવાઈસ પર યુનિક નામ સાથે સ્ટોર કરેલો ડેટાનો કલેક્શન
- **બાઈનરી ફાઈલ**: બાઈનરી ફોર્મેટ (0s અને 1s) માં ડેટા સ્ટોર કરે, માનવ-વાંચી ન શકાય
- **ટેક્સ્ટ ફાઈલ**: ASCII અથવા Unicode કેરેક્ટર્સ સમાવે, માનવ-વાંચી શકાય તેવું ફોર્મેટ

મેમરી ટ્રીક

``ફાઈલ્સ ડેટા સ્ટોર કરે, બાઈનરી=મશીન, ટેક્સ્ટ=માનવ''

પ્રશ્ન 4(બ) [4 માર્ક્સ]

write() અને writelines() ફંક્શન ઉદાહરણ સાથે સમજાવો.

જવાબ

રાઈટ ફંક્શન્સનું ટેબલ:

ફંક્શન	હેતુ	પેરામીટર	ઉપયોગ
write()	સિંગલ સ્ટ્રિંગ લખે	સ્ટ્રિંગ	`file.write("Hello")`
writelines()	સ્ટ્રિંગ્સની લિસ્ટ લખે	લિસ્ટ/સિકવન્સ	`file.writelines(["line1", "line2"])`

```
\# write()   writelines()
def demonstrate\_write\_functions():

    \# write()
    with open("write\_demo.txt", "w") as file:
        file.write("     !{n}")
        file.write("        {n}")
        file.write("         {n}")

    \# writelines()
    lines = [
        "writelines          {n}",
        "writelines          {n}",
        "writelines           {n}"
    ]

    with open("writelines\_demo.txt", "w") as file:
        file.writelines(lines)

    print("             !")

\#
demonstrate\_write\_functions()
```

મુખ્ય તફાવતો:

- **write()**: એક સમયે એક સ્ટ્રિંગ લખે
- **writelines()**: સિકવન્સમાંથી મલ્ટિપલ સ્ટ્રિંગ્સ લખે
- **ન્યુલાઈન્સ**: `\n` મેન્યુઅલી ઉમેરવું પડે
- **રિટર્ન વેલ્યુ**: બંને લખાયેલા કેરેક્ટર્સની સંખ્યા રિટર્ન કરે

મેમરી ટ્રીક

``write() સિંગલ, writelines() મલ્ટિપલ''

પ્રશ્ન 4(ક) [7 માર્ક્સ]

tell() અને seek() ફંક્શન ઉદાહરણ સાથે સમજાવો.

જવાબ

ફાઈલ પોઈન્ટર ફંક્શન્સ ફાઈલની અંદર રીડિંગ/રાઈટિંગ માટે પોઝિશન કંટ્રોલ કરે છે.
પોઝિશન ફંક્શન્સનું ટેબલ:

ફંક્શન	હેતુ	રિટર્ન/પેરામીટર	ઉપયોગ
tell()	વર્તમાન પોઝિશન મેળવો	વર્તમાન બાઈટ પોઝિશન રિટર્ન	`pos = file.tell()`

seek(offset, whence)	ચોક્કસ પોઝિશન પર જાઓ	offset: બાઈટ્સ, whence: રેફરન્સ	`file.seek(10, 0)`

Seek Whence વેલ્યુઝ:

વેલ્યુ	રેફરન્સ પોઈન્ટ	વર્ણન
0	ફાઈલની શરૂઆત	એબ્સોલ્યુટ પોઝિશનિંગ
1	વર્તમાન પોઝિશન	વર્તમાનના સંબંધમાં
2	ફાઈલનો અંત	અંતના સંબંધમાં

```
\# tell()   seek()
def demonstrate\_file\_positioning():

    \#
    sample\_text = "     !   tell()   seek()                  ."

    with open("position\_demo.txt", "w", encoding={utf{-}8}) as file:
        file.write(sample\_text)

    \# tell()   seek()
    with open("position\_demo.txt", "r", encoding={utf{-}8}) as file:

        \#
        print(f"1.         : \{file.tell()\}")

        \#    5
        data1 = file.read(5)
        print(f"2. {}\{data1\}{   ,         : }\{file.tell()\}")

        \#     15
        file.seek(15)
        print(f"3. seek(15)   ,    : \{file.tell()\}")

        \#    10
        data2 = file.read(10)
        print(f"4. {}\{data2\}{   ,         : }\{file.tell()\}")

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

        \# seek(0, 2)
        file.seek(0, 2)
        print(f"6. seek(0,2)   ,    : \{file.tell()\}")

        \#
        file.seek({-}10, 1)
        print(f"7. seek({-10,1)   ,    : }\{file.tell()\}")

        \#
        remaining = file.read()
        print(f"8.        : {}\{remaining\}{"})

\#
def binary\_file\_positioning():

    \#
    data = b"Binary file positioning example"

    with open("binary\_demo.bin", "wb") as file:
        file.write(data)

    \#
    with open("binary\_demo.bin", "rb") as file:
        print(f"{n}           : \{file.tell()\}")

        \#    6
        chunk1 = file.read(6)
        print(f"  : \{chunk1\},    : \{file.tell()\}")

        \#     20
```

```
        file.seek(20)
        chunk2 = file.read(7)
        print(f"  : \{chunk2\},    : \{file.tell()\}")

\#
demonstrate\_file\_positioning()
binary\_file\_positioning()
```

પોઝિશન કંટ્રોલ ડાયાગ્રામ:

```
flowchart LR
    A[     ] {-{-} B["tell(): 0"]}
    B {-{-} C["read(5)"]}
    C {-{-} D["tell(): 5"]}
    D {-{-} E["seek(15)"]}
    E {-{-} F["tell(): 15"]}
    F {-{-} G["seek(0,2)"]}
    G {-{-} H["tell():  "]}
```

- **tell()**: ફાઈલમાં વર્તમાન બાઈટ પોઝિશન રિટર્ન કરે
- **seek()**: ફાઈલ પોઈન્ટરને સ્પેસિફાઈડ પોઝિશન પર મૂવ કરે
- **પોઝિશનિંગ**: રેન્ડમ ફાઈલ એક્સેસ માટે જરૂરી
- **બાઈનરી મોડ**: બાઈટ પોઝિશન્સ સાથે કામ કરે

મેમરી ટ્રીક

``tell() પોઝિશન, seek() મૂવમેન્ટ''

પ્રશ્ન 4(અ OR) [3 માર્ક્સ]

એબ્સોલ્યુટ અને રિલેટિવ પાથ શું છે?

જવાબ

પાથ ટાઈપ્સનું ટેબલ:

પાથ ટાઈપ	વર્ણન	ઉદાહરણ
એબ્સોલ્યુટ પાથ	રૂટ ડિરેક્ટરીથી સંપૂર્ણ પાથ	`/home/user/documents/file.txt`
રિલેટિવ પાથ	વર્તમાન ડિરેક્ટરીના સંબંધમાં પાથ	`../documents/file.txt`

પાથ સિમ્બોલ્સ:

સિમ્બોલ	અર્થ	ઉદાહરણ
`/`	રૂટ ડિરેક્ટરી (Linux/Mac)	`/home/user/`
`C:\`	ડ્રાઈવ લેટર (Windows)	`C:\\Users\\Documents\\`
`.`	વર્તમાન ડિરેક્ટરી	`./file.txt`
`..`	પેરન્ટ ડિરેક્ટરી	`../folder/file.txt`

- **એબ્સોલ્યુટ**: સિસ્ટમ રૂટથી સંપૂર્ણ પાથ
- **રિલેટિવ**: વર્તમાન વર્કિંગ ડિરેક્ટરીથી પાથ

મેમરી ટ્રીક

``એબ્સોલ્યુટ રૂટથી, રિલેટિવ વર્તમાનથી''

પ્રશ્ન 4(બ OR) [4 માર્ક્સ]

બાયનરી અને ટેક્સ્ટ ફાઇલ ખોલવાના વિવિધ મોડ સમજાવો.

જવાબ

ફાઈલ ઓપનિંગ મોડ્સનું ટેબલ:

મોડ	ટાઈપ	હેતુ	ફાઈલ પોઈન્ટર
`r'	ટેક્સ્ટ	ફક્ત વાંચવા	શરૂઆત
`w'	ટેક્સ્ટ	લખવા (ઓવરરાઈટ)	શરૂઆત
`a'	ટેક્સ્ટ	ઉમેરવા	અંત
`rb'	બાઈનરી	બાઈનરી વાંચવા	શરૂઆત
`wb'	બાઈનરી	બાઈનરી લખવા	શરૂઆત
`ab'	બાઈનરી	બાઈનરી ઉમેરવા	અંત
`r+'	ટેક્સ્ટ	વાંચવા અને લખવા	શરૂઆત
`w+'	ટેક્સ્ટ	લખવા અને વાંચવા	શરૂઆત

```
\#
def demonstrate\_file\_modes():

    \#
    with open("text\_file.txt", "w") as f:  \#
        f.write("     ")

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

    \#
    data = b"Binary data example"
    with open("binary\_file.bin", "wb") as f:  \#
        f.write(data)

    with open("binary\_file.bin", "rb") as f:  \#
        binary\_content = f.read()
        print(f"       : \{binary\_content\}")

demonstrate\_file\_modes()
```

- **ટેક્સ્ટ મોડ્સ**: એન્કોડિંગ સાથે સ્ટ્રિંગ ડેટા હેન્ડલ કરે
- **બાઈનરી મોડ્સ**: એન્કોડિંગ વગર રો બાઈટ્સ હેન્ડલ કરે
- **પ્લસ મોડ્સ**: રીડિંગ અને રાઈટિંગ બંનેની મંજૂરી

મેમરી ટ્રીક

``ટેક્સ્ટ સ્ટ્રિંગ્સ માટે, બાઈનરી બાઈટ્સ માટે''

પ્રશ્ન 4(ક OR) [7 માર્ક્સ]

વિદ્યાર્થીના વિષય રેકોર્ડ જેવાં કે શાખાનું નામ, સેમેસ્ટર, વિષયનો કોડ અને વિષયનું નામ બાઇનરી ફાઇલમાં લખવા માટેનો પ્રોગ્રામ લખો.

જવાબ

```
import pickle
import os

class StudentSubjectRecord:
    """                          """

    def \_\_init\_\_(self, branch\_name, semester, subject\_code, subject\_name):
        self.branch\_name = branch\_name
        self.semester = semester
```

```
        self.subject\_code = subject\_code
        self.subject\_name = subject\_name

    def \_\_str\_\_(self):
        return f"  : \{self.branch\_name\},    : \{self.semester\},   : \{self.subject\_code\},    : \{s

def write\_student\_records():
    """                    """

    \#
    records = [
        StudentSubjectRecord("          ", 2, "4321602", "             "),
        StudentSubjectRecord("          ", 2, "4321601", "           "),
        StudentSubjectRecord("          ", 3, "4330701", "      "),
        StudentSubjectRecord("          ", 2, "4321603", "      "),
        StudentSubjectRecord("          ", 3, "4330702", "        ")
    ]

    \# pickle
    try:
        with open("student\_records.bin", "wb") as binary\_file:
            pickle.dump(records, binary\_file)

        print("                              !")
        print(f"           : \{len(records)\}")

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

def read\_student\_records():
    """                      """

    try:
        if not os.path.exists("student\_records.bin"):
            print("             !")
            return

        with open("student\_records.bin", "rb") as binary\_file:
            records = pickle.load(binary\_file)

        print("{n}" + "="*60)
        print("                      ")
        print("="*60)

        for i, record in enumerate(records, 1):
            print(f"\{i\}. \{record\}")

        print("="*60)
        print(f"         : \{len(records)\}")

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

def add\_new\_record():
    """                           """

    try:
        \#
        records = []
        if os.path.exists("student\_records.bin"):
            with open("student\_records.bin", "rb") as binary\_file:
```

```
                records = pickle.load(binary\_file)

        \#
        print("{n}                      :")
        branch = input("     : ")
        semester = int(input("   : "))
        code = input("      : ")
        subject = input("     : ")

        \#
        new\_record = StudentSubjectRecord(branch, semester, code, subject)
        records.append(new\_record)

        \#
        with open("student\_records.bin", "wb") as binary\_file:
            pickle.dump(records, binary\_file)

        print("                !")

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

def search\_records\_by\_branch(branch\_name):
    """                 """

    try:
        if not os.path.exists("student\_records.bin"):
            print("             !")
            return

        with open("student\_records.bin", "rb") as binary\_file:
            records = pickle.load(binary\_file)

        \#
        filtered\_records = [record for record in records
                            if record.branch\_name.lower() == branch\_name.lower()]

        if filtered\_records:
            print(f"{n}\{branch\_name\}        :")
            print("{-"} * 40)
            for record in filtered\_records:
                print(f"  \{record\}")
        else:
            print(f"                : \{branch\_name\}")

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

\#
def main():
    """                         """

    print("===                 ===={n}")

    \#
    print("1.                           ...")
    write\_student\_records()

    \#
    print("{n}2.                       ...")
    read\_student\_records()
```

```
    \#
    print("{n}3.                    ...")
    search\_records\_by\_branch("          ")

    \#
    if os.path.exists("student\_records.bin"):
        file\_size = os.path.getsize("student\_records.bin")
        print(f"{n}          : \{file\_size\}    ")

\#
if \_\_name\_\_ == "\_\_main\_\_":
    main()
```

બાઈનરી ફાઈલ ઓપરેશન્સનું ટેબલ:

ઓપરેશન	મેથડ	હેતુ
લખવું	`pickle.dump()`	ઓબ્જેક્ટ્સને બાઈનરીમાં સીરિયલાઈઝ
વાંચવું	`pickle.load()`	બાઈનરીમાંથી ઓબ્જેક્ટ્સ ડીસીરિયલાઈઝ
ઉમેરવું	વાંચવું + ઉમેરવું + લખવું	નવા રેકોર્ડ્સ ઉમેરો
શોધવું	લોડ કરેલો ડેટા ફિલ્ટર	ચોક્કસ રેકોર્ડ્સ શોધો

બાઈનરી ફાઈલ સ્ટ્રક્ચર:

```
flowchart LR
    A[        ] {-{-} B["pickle.dump()"]}
    B {-{-} C[       .bin]}
    C {-{-} D["pickle.load()"]}
    D {-{-} E[Python    ]}
```

- **બાઈનરી સ્ટોરેજ**: ઓબ્જેક્ટ સીરિયલાઈઝેશન માટે pickle વાપરે
- **કાર્યક્ષમ સ્ટોરેજ**: કોમ્પેક્ટ બાઈનરી ફોર્મેટ
- **ઓબ્જેક્ટ પ્રિઝર્વેશન**: ડેટા સ્ટ્રક્ચર ઈન્ટેગ્રિટી જાળવે
- **ક્રોસ-પ્લેટફોર્મ**: વિવિધ ઓપરેટિંગ સિસ્ટમ્સ પર કામ કરે

મેમરી ટ્રીક

``Pickle Python ઓબ્જેક્ટ્સ પ્રિઝર્વ કરે''

પ્રશ્ન 5(અ) [3 માર્ક્સ]

વ્યાખ્યાયિત કરો: GUI, CLI

જવાબ

ઈન્ટરફેસ વ્યાખ્યાઓનું ટેબલ:

શબ્દ	પૂરું નામ	વર્ણન	ઉદાહરણ
GUI	Graphical User Interface	વિન્ડોઝ, બટન્સ, આઈકોન્સ સાથે વિઝ્યુઅલ ઈન્ટરફેસ	Windows, Mac ડેસ્કટોપ
CLI	Command Line Interface	કમાન્ડ્સ વાપરીને ટેક્સ્ટ-આધારિત ઈન્ટરફેસ	ટર્મિનલ, કમાન્ડ પ્રોમ્પ્ટ

મુખ્ય તફાવતો:

- **GUI**: યુઝર-ફ્રેન્ડલી, માઉસ-ડ્રિવન, વિઝ્યુઅલ એલિમેન્ટ્સ
- **CLI**: ટેક્સ્ટ-આધારિત, કીબોર્ડ-ડ્રિવન, કમાન્ડ સિન્ટેક્સ
- **ઈન્ટરેક્શન**: GUI ક્લિક્સ વાપરે, CLI ટાઈપ કરેલા કમાન્ડ્સ વાપરે

મેમરી ટ્રીક

``GUI ગ્રાફિક્સ, CLI કમાન્ડ્સ''

પ્રશ્ન 5(બ) [4 માર્ક્સ]

ટર્ટલનો ઉપયોગ કરીને for અને while લૂપનો ઉપયોગ કરીને ચોરસ આકાર દોરવા માટે પાયથોન પ્રોગ્રામ લખો.

જવાબ

```
import turtle

def draw\_square\_with\_for\_loop():
    """for             """

    \#
    screen = turtle.Screen()
    screen.bgcolor("white")
    square\_turtle = turtle.Turtle()
    square\_turtle.color("blue")
    square\_turtle.pensize(3)

    \# for
    print("for                    ...")
    side\_length = 100

    for i in range(4):
        square\_turtle.forward(side\_length)
        square\_turtle.right(90)

    square\_turtle.penup()
    square\_turtle.goto(150, 0)
    square\_turtle.pendown()

    return square\_turtle

def draw\_square\_with\_while\_loop(turtle\_obj):
    """while             """

    \#
    turtle\_obj.color("red")

    \# while
    print("while                   ...")
    side\_length = 100
    sides\_drawn = 0

    while sides\_drawn {} 4:
        turtle\_obj.forward(side\_length)
        turtle\_obj.right(90)
        sides\_drawn += 1

    \#
    turtle\_obj.penup()
    turtle\_obj.goto({-}50, {-}150)
    turtle\_obj.write("  : for  ,  : while  ",
                     font=("Arial", 12, "normal"))

\#
def main():
    \#
```

```
    turtle\_obj = draw\_square\_with\_for\_loop()
    draw\_square\_with\_while\_loop(turtle\_obj)

    \#
    turtle.Screen().exitonclick()

\#
main()
```

લૂપ કમ્પેરિઝનનું ટેબલ:

લૂપ ટાઈપ	સ્ટ્રક્ચર	ઉપયોગ	કંટ્રોલ
for લૂપ	`for i in range(4):`	જાણીતા ઈટરેશન્સ	કાઉન્ટર-આધારિત
while લૂપ	`while condition:`	કન્ડિશનલ ઈટરેશન્સ	કન્ડિશન-આધારિત

- **for લૂપ**: જાણીતી સંખ્યાના ઈટરેશન્સ માટે શ્રેષ્ઠ
- **while લૂપ**: કન્ડિશન-આધારિત રિપેટિશન માટે શ્રેષ્ઠ
- **બંને અચીવ કરે**: એક જ ચોરસ દોરવાનું પરિણામ

મેમરી ટ્રીક

``For કાઉન્ટ, While કન્ડિશન''

પ્રશ્ન 5(ક) [7 માર્ક્સ]

ટર્ટલનો ઉપયોગ કરીને ચેસબોર્ડ દોરવા માટે પાયથોન પ્રોગ્રામ લખો.

જવાબ

```
import turtle

def setup\_chessboard():
    """                          """

    screen = turtle.Screen()
    screen.bgcolor("white")
    screen.title("Python            ")
    screen.setup(width=600, height=600)

    \#
    chess\_turtle = turtle.Turtle()
    chess\_turtle.speed(0)  \#
    chess\_turtle.penup()

    return screen, chess\_turtle

def draw\_square(turtle\_obj, size, fill\_color):
    """               """

    turtle\_obj.pendown()
    turtle\_obj.fillcolor(fill\_color)
    turtle\_obj.begin\_fill()

    \#
    for \_ in range(4):
        turtle\_obj.forward(size)
        turtle\_obj.right(90)

    turtle\_obj.end\_fill()
```

```
    turtle\_obj.penup()

def draw\_chessboard():
    """   8x8       """

    screen, chess\_turtle = setup\_chessboard()

    \#
    square\_size = 40
    board\_size = 8
    start\_x = {-}160
    start\_y = 160

    print("            ...")

    \#
    for row in range(board\_size):
        for col in range(board\_size):

            \#
            x = start\_x + (col * square\_size)
            y = start\_y {-} (row * square\_size)

            \#
            chess\_turtle.goto(x, y)

            \#             (       )
            if (row + col) \% 2 == 0:
                color = "white"
            else:
                color = "black"

            \#
            draw\_square(chess\_turtle, square\_size, color)

    \#
    draw\_border(chess\_turtle, start\_x, start\_y, square\_size, board\_size)

    \#
    add\_chessboard\_labels(chess\_turtle, start\_x, start\_y, square\_size, board\_size)

    return screen

def draw\_border(turtle\_obj, start\_x, start\_y, square\_size, board\_size):
    """               """

    turtle\_obj.goto(start\_x {-} 5, start\_y + 5)
    turtle\_obj.pencolor("brown")
    turtle\_obj.pensize(3)
    turtle\_obj.pendown()

    \#
    border\_width = board\_size * square\_size + 10
    border\_height = board\_size * square\_size + 10

    for \_ in range(2):
        turtle\_obj.forward(border\_width)
        turtle\_obj.right(90)
        turtle\_obj.forward(border\_height)
        turtle\_obj.right(90)
```

```
    turtle\_obj.penup()
    turtle\_obj.pensize(1)
    turtle\_obj.pencolor("black")

def add\_chessboard\_labels(turtle\_obj, start\_x, start\_y, square\_size, board\_size):
    """                  """

    turtle\_obj.color("blue")

    \#       (A{-H)}
    columns = [{A}, {B}, {C}, {D}, {E}, {F}, {G}, {H}]
    for i, letter in enumerate(columns):
        x = start\_x + (i * square\_size) + (square\_size // 2)
        y = start\_y {-} (board\_size * square\_size) {-} 20
        turtle\_obj.goto(x, y)
        turtle\_obj.write(letter, align="center", font=("Arial", 12, "bold"))

    \#      (1{-8)}
    for i in range(board\_size):
        x = start\_x {-} 20
        y = start\_y {-} (i * square\_size) {-} (square\_size // 2)
        turtle\_obj.goto(x, y)
        turtle\_obj.write(str(8{-}i), align="center", font=("Arial", 12, "bold"))

    \#
    turtle\_obj.goto(0, start\_y + 30)
    turtle\_obj.write("Python       ", align="center",
                     font=("Arial", 16, "bold"))

def draw\_enhanced\_chessboard\_with\_pieces():
    """                    """

    screen = draw\_chessboard()

    \#
    piece\_turtle = turtle.Turtle()
    piece\_turtle.speed(0)
    piece\_turtle.penup()

    \#               (     )
    pieces = [
        ({-}120, 120, " "), ({-}80, 120, " "), ({-}40, 120, " "), (0, 120, " "),
        ({-}120, {-}120, " "), ({-}80, {-}120, " "), ({-}40, {-}120, " "), (0, {-}120, " ")
    ]

    piece\_turtle.color("red")
    for x, y, symbol in pieces:
        piece\_turtle.goto(x, y)
        piece\_turtle.write(symbol, align="center", font=("Arial", 20, "normal"))

    piece\_turtle.hideturtle()

    \#
    total\_squares = 64
    black\_squares = 32
    white\_squares = 32

    piece\_turtle.goto(0, {-}200)
    piece\_turtle.color("green")
    piece\_turtle.write(f"   : \{total\_squares\}    (\{black\_squares\}   , \{white\_squares\}   )",
                       align="center", font=("Arial", 12, "normal"))
```

```
    return screen

\#
def main():
    """                 """

    print("Python                      ...")
    screen = draw\_enhanced\_chessboard\_with\_pieces()

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

    \#
    screen.exitonclick()

\#
if \_\_name\_\_ == "\_\_main\_\_":
    main()
```

ચેસબોર્ડ કોમ્પોનન્ટ્સનું ટેબલ:

કોમ્પોનન્ટ	ઇમ્પ્લિમેન્ટેશન	હેતુ
ચોરસ	8x8 ગ્રિડ ઓલ્ટર્નેટિંગ કલર્સ	મુખ્ય બોર્ડ પેટર્ન
રંગો	કાળો અને સફેદ ઓલ્ટર્નેટિંગ	પરંપરાગત ચેસ પેટર્ન
બોર્ડર	બ્રાઉન રેક્ટેંગલ આઉટલાઈન	બોર્ડને ફ્રેમ કરો
લેબલ્સ	A-H કોલમ્સ, 1-8 રોઝ	ચેસ નોટેશન
પીસીસ	યુનિકોડ ચેસ સિમ્બોલ્સ	સેમ્પલ પીસ પ્લેસમેન્ટ

ચેસબોર્ડ પેટર્ન લોજિક:

```
flowchart LR
    A[  0  ] {-{-} B\{      \}}
    B {-{-} C\{       \}}
    C {-{-} D\{row + col  ?\}}
    D {-{-}| | E[    ]}
    D {-{-}| | F[     ]}
    E {-{-} G[     ]}
    F {-{-} G}
    G {-{-} C}
    C {-{-}|    | H[     ]}
    H {-{-} B}
    B {-{-}|      | I[     ]}
```

- **ઓલ્ટર્નેટિંગ પેટર્ન**: (row + col) % 2 રંગ નક્કી કરે
- **ગ્રિડ સિસ્ટમ**: ચોક્કસ પોઝિશનિંગ સાથે 8x8 ચોરસ
- **વિઝ્યુઅલ એન્હાન્સમેન્ટ્સ**: બોર્ડર, લેબલ્સ અને સેમ્પલ પીસીસ
- **સ્કેલેબલ ડિઝાઈન**: ચોરસ સાઈઝ મોડિફાઈ કરવું સરળ

મેમરી ટ્રીક

``ગ્રિડ પેટર્નમાં ઓલ્ટર્નેટ કલર્સ''

પ્રશ્ન 5(અ OR) [3 માર્ક્સ]

ટર્ટલમાં કેટલા પ્રકારના આકાર હોય છે? કોઈપણ એક આકાર યોગ્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

ટર્ટલ શેપ્સનું ટેબલ:

શેપ ટાઈપ	ઉદાહરણો	મેથડ
બેસિક શેપ્સ	વતુળ, ચોરસ, ત્રિકોણ	બિલ્ટ-ઇન ફંક્શન્સ
લાઈન પેટર્ન્સ	સીધી લાઈનો, કર્વ્સ	forward(), backward()
પોલિગોન્સ	પંચકોણ, ષટકોણ, અષ્ટકોણ	એંગલ્સ સાથે લૂપ
કોમ્પ્લેક્સ શેપ્સ	તારા, સ્પાઈરલ્સ, ફ્રેક્ટલ્સ	ગાણિતિક પેટર્ન્સ
કસ્ટમ શેપ્સ	યુઝર-ડિફાઈન્ડ પેટર્ન્સ	મૂવમેન્ટ્સનું કોમ્બિનેશન

વતુળ શેપ ઉદાહરણ:

```
import turtle

def draw\_circle\_example():
    screen = turtle.Screen()
    circle\_turtle = turtle.Turtle()

    \# 50
    circle\_turtle.circle(50)

    screen.exitonclick()

draw\_circle\_example()
```

- **બિલ્ટ-ઇન શેપ્સ**: વતુળ, ચોરસ, ત્રિકોણ સહેલાઈથી ઉપલબ્ધ
- **કસ્ટમ શેપ્સ**: મૂવમેન્ટ કોમ્બિનેશન વાપરીને બનાવાય
- **ગાણિતિક શેપ્સ**: ચોક્કસ ડ્રોઈંગ માટે ભૂમિતિ વાપરો

મેમરી ટ્રીક

``ટર્ટલ ઘણા શેપ ટાઈપ્સ દોરે''

પ્રશ્ન 5(બ OR) [4 માર્ક્સ]

ટર્ટલ મોડ્યુલની ચાર મૂળભૂત પદ્ધતિઓ વિશે સમજાવો.

જવાબ

બેસિક ટર્ટલ મેથડ્સનું ટેબલ:

મેથડ	હેતુ	પેરામીટર્સ	ઉદાહરણ
forward(distance)	ટર્ટલને આગળ ખસેડો	પિક્સલ્સમાં અંતર	`turtle.forward(100)`
backward(distance)	ટર્ટલને પાછળ ખસેડો	પિક્સલ્સમાં અંતર	`turtle.backward(50)`
right(angle)	ટર્ટલને જમણે ફેરવો	ડિગ્રીમાં એંગલ	`turtle.right(90)`
left(angle)	ટર્ટલને ડાબે ફેરવો	ડિગ્રીમાં એંગલ	`turtle.left(45)`

```
import turtle

def demonstrate\_basic\_methods():
    \#
    demo\_turtle = turtle.Turtle()

    \# 1.
    demo\_turtle.forward(100)  \# 100

    \# 2.
    demo\_turtle.right(90)     \# 90

    \# 3.
    demo\_turtle.backward(50)  \# 50

    \# 4.
    demo\_turtle.left(135)     \# 135

    turtle.done()

demonstrate\_basic\_methods()
```

- **મૂવમેન્ટ મેથડ્સ**: અંતર માટે forward() અને backward()
- **રોટેશન મેથડ્સ**: દિશા બદલવા માટે right() અને left()
- **કોઓર્ડિનેટ સિસ્ટમ**: વર્તમાન ટર્ટલ પોઝિશન અને હેડિંગ આધારિત
- **એંગલ મેઝરમેન્ટ**: ડિગ્રીઝ (0-360)

મેમરી ટ્રીક

``આગળ, પાછળ, જમણે, ડાબે બેસિક્સ''

પ્રશ્ન 5(ક OR) [7 માર્ક્સ]

ટર્ટલનો ઉપયોગ કરીને ચોરસ, લાંબચોરસ અને વતુળ દોરવા માટે પાયથોન પ્રોગ્રામ લખો.

જવાબ

```
import turtle
import math

def setup\_drawing\_environment():
    """                          """

    screen = turtle.Screen()
    screen.bgcolor("lightblue")
    screen.title("      :    ,     ,   ")
    screen.setup(width=800, height=600)

    \#
    shape\_turtle = turtle.Turtle()
    shape\_turtle.speed(3)
    shape\_turtle.pensize(2)

    return screen, shape\_turtle

def draw\_square(turtle\_obj, size, color, position):
    """                      """

    x, y = position
    turtle\_obj.penup()
```

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

    turtle\_obj.color(color)
    turtle\_obj.fillcolor(color)
    turtle\_obj.begin\_fill()

    \# 4
    for \_ in range(4):
        turtle\_obj.forward(size)
        turtle\_obj.right(90)

    turtle\_obj.end\_fill()

    \#
    turtle\_obj.penup()
    turtle\_obj.goto(x + size//2, y {-} 30)
    turtle\_obj.color("black")
    turtle\_obj.write(f"   (\{size\}x\{size\})", align="center",
                     font=("Arial", 10, "bold"))

def draw\_rectangle(turtle\_obj, width, height, color, position):
    """   dimensions               """

    x, y = position
    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(2):
        turtle\_obj.forward(width)
        turtle\_obj.right(90)
        turtle\_obj.forward(height)
        turtle\_obj.right(90)

    turtle\_obj.end\_fill()

    \#
    turtle\_obj.penup()
    turtle\_obj.goto(x + width//2, y {-} height {-} 20)
    turtle\_obj.color("black")
    turtle\_obj.write(f"     (\{width\}x\{height\})", align="center",
                     font=("Arial", 10, "bold"))

def draw\_circle(turtle\_obj, radius, color, position):
    """                    """

    x, y = position
    turtle\_obj.penup()
    turtle\_obj.goto(x, y {-} radius)  \#
    turtle\_obj.pendown()

    turtle\_obj.color(color)
    turtle\_obj.fillcolor(color)
    turtle\_obj.begin\_fill()
```

```
    \#
    turtle\_obj.circle(radius)

    turtle\_obj.end\_fill()

    \#
    area = math.pi * radius * radius
    turtle\_obj.penup()
    turtle\_obj.goto(x, y {-} radius {-} 30)
    turtle\_obj.color("black")
    turtle\_obj.write(f"   (r=\{radius\},    =\{area:.1f\})", align="center",
                      font=("Arial", 10, "bold"))

def draw\_all\_shapes():
    """                   """

    screen, shape\_turtle = setup\_drawing\_environment()

    print("                ...")

    \#
    print("1.             ...")
    draw\_square(shape\_turtle, 80, "red", ({-}300, 100))

    \#
    print("2.              ...")
    draw\_rectangle(shape\_turtle, 120, 80, "green", ({-}50, 100))

    \#
    print("3.            ...")
    draw\_circle(shape\_turtle, 60, "blue", (200, 100))

    \#
    add\_shape\_information(shape\_turtle)

    print("               !")
    return screen

def add\_shape\_information(turtle\_obj):
    """                 """

    \#
    turtle\_obj.penup()
    turtle\_obj.goto(0, 200)
    turtle\_obj.color("purple")
    turtle\_obj.write("Python       ", align="center",
                     font=("Arial", 18, "bold"))

    \#
    turtle\_obj.goto({-}350, {-}50)
    turtle\_obj.color("black")
    turtle\_obj.write("         :", font=("Arial", 12, "bold"))

    properties = [
        "•   : 4       , 4    ",
        "•     : 4   ,          ",
        "•   :                "
    ]

    for i, prop in enumerate(properties):
        turtle\_obj.goto({-}350, {-}80 {-} (i * 20))
```

```
        turtle\_obj.write(prop, font=("Arial", 10, "normal"))

    \#
    turtle\_obj.goto({-}350, {-}170)
    turtle\_obj.color("blue")
    turtle\_obj.write("  :", font=("Arial", 12, "bold"))

    formulas = [
        "•       :   ^{2"},
        "•         :         ",
        "•       :       ^{2"}
    ]

    for i, formula in enumerate(formulas):
        turtle\_obj.goto({-}350, {-}200 {-} (i * 20))
        turtle\_obj.write(formula, font=("Arial", 10, "normal"))

def interactive\_shape\_drawer():
    """                """

    screen, shape\_turtle = setup\_drawing\_environment()

    \#
    print("{n}===             ===")

    try:
        \#
        square\_size = int(input("            (50{-100): "}) or "80")
        square\_color = input("           (  /  /  ): ") or "red"

        \#
        rect\_width = int(input("              (80{-150): "}) or "120")
        rect\_height = int(input("             (60{-100): "}) or "80")
        rect\_color = input("           : ") or "green"

        \#
        circle\_radius = int(input("            (40{-80): "}) or "60")
        circle\_color = input("         : ") or "blue"

        \#
        draw\_square(shape\_turtle, square\_size, square\_color, ({-}300, 100))
        draw\_rectangle(shape\_turtle, rect\_width, rect\_height, rect\_color, ({-}50, 100))
        draw\_circle(shape\_turtle, circle\_radius, circle\_color, (200, 100))

        add\_shape\_information(shape\_turtle)

    except ValueError:
        print("      !                ...")
        return draw\_all\_shapes()

    return screen

\#
def main():
    """            """

    print("          :")
    print("1.        ")
    print("2.        ")

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

```
    if choice == "2":
        screen = interactive\_shape\_drawer()
    else:
        screen = draw\_all\_shapes()

    print("{n}                        .")
    screen.exitonclick()

\#
if \_\_name\_\_ == "\_\_main\_\_":
    main()
```

આકાર લક્ષણોનું ટેબલ:

આકાર	બાજુઓ	પ્રોપર્ટીઝ	ક્ષેત્રફળ સૂત્ર
ચોરસ	4 સમાન	બધા કોણ 90°	બાજુ^{2}
લાંબચોરસ	4 (2 જોડ)	સામેની બાજુઓ સમાન	લંબાઈ ×
વતુળ	0 (વક્ર)	બધા બિંદુઓ સમદૂર	□ ×^{2}

આકાર ડ્રોઇંગ પ્રક્રિયા:

```
flowchart LR
    A[     ] {-{-} B[     ]}
    B {-{-} C[     ]}
    C {-{-} D[       ] }
    D {-{-} E[     ]}
    E {-{-} F[     ]}
    F {-{-} G[       ]}
    G {-{-} H[ ]}
```

- **ભૌમિતિક ચોકસાઈ**: ચોક્કસ કોણ અને અંતર માપદંડો
- **વિઝ્યુઅલ અપીલ**: વિવિધ રંગો અને ભરેલા આકારો
- **શૈક્ષણિક મૂલ્ય**: સૂત્રો દર્શાવે છે
- **ગાણિતિક ગણતરીઓ**: ક્ષેત્રફળ સૂત્રો સામેલ
- **ઇન્ટરેક્ટિવ ફીચર્સ**: યુઝર પેરામીટર્સ કસ્ટમાઈઝ કરી શકે

મેમરી ટ્રીક

``ચોરસ સમાન, લાંબચોરસ સામેના, વતુળ ગોળ''