Subject Name (Gujarati)

4351108 -- Winter 2024

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 ગુણ]

પાયથોન પ્રોગ્રામિંગ ભાષાના લક્ષણોની યાદી બનાવો.

જવાબ

લક્ષણ	વર્ણન
સરળ અને સહેલું	સ્વચ્છ, વાંચી શકાય તેવું syntax
મફત અને ઓપન સોર્સ	કોઈ કિંમત નહીં, community driven
ક્રોસ-પ્લેટફોર્મ	Windows, Linux, Mac પર ચાલે છે
Interpreted	compilation ની જરૂર નથી
Object-Oriented	classes અને objects ને support કરે છે
મોટી લાઇબ્રેરીઓ	સમૃદ્ધ standard library

યાદી રાખવાની ટ્રિક: ``સરળ મફત ક્રોસ Interpreted ઓબ્જેક્ટ મોટી''

પ્રશ્ન 1(બ) [4 ગુણ]

પાયથોન પ્રોગ્રામિંગ ભાષાની એપ્લિકેશનો લખો.

જવાબ

એપ્લિકેશન ક્ષેત્ર	ઉદાહરણો
વેબ ડેવલપમેન્ટ	Django, Flask frameworks
ડેટા સાયન્સ	NumPy, Pandas, Matplotlib
મશીન લર્નિંગ	TensorFlow, Scikit-learn
ડેસ્કટોપ GUI	Tkinter, PyQt applications
ગેમ ડેવલપમેન્ટ	Pygame library
ઓટોમેશન	Scripting અને testing

યાદી રાખવાની ટ્રિક: ``વેબ ડેટા મશીન ડેસ્કટોપ ગેમ ઓટો''

પ્રશ્ન 1(ક) [7 ગુણ]

પાયથોનમાં વિવિધ ડેટાટાઇપ્સ સમજાવો.

જવાબ

ડેટા ટાઇપ	ઉદાહરણ	વર્ણન
int	`x = 5`	પૂર્ણાંક સંખ્યાઓ
float	`y = 3.14`	દશાંશ સંખ્યાઓ
str	`name = "John"`	ટેક્સ્ટ ડેટા
bool	`flag = True`	True/False મૂલ્યો

list	`[1, 2, 3]`	ક્રમબદ્ધ, બદલી શકાય તેવું
tuple	`(1, 2, 3)`	ક્રમબદ્ધ, બદલી ન શકાય તેવું
dict	`{"a": 1}`	Key-value જોડી
set	`{1, 2, 3}`	અનન્ય ઘટકો

કોડ ઉદાહરણ:

```
\# Numeric types
age = 25          \# int
price = 99.99     \# float

\# Text type
name = "Python"   \# str

\# Boolean type
is\_valid = True   \# bool

\# Collection types
numbers = [1, 2, 3]        \# list
coordinates = (10, 20)     \# tuple
student = \{"name": "John"\ }\# dict
unique\_ids = \{1, 2, 3\     }\# set
```

યાદી રાખવાની ટ્રિક: ``Integer Float String Boolean List Tuple Dict Set''

પ્રશ્ન 1(ક OR) [7 ગુણ]

એરિથમેટિક, એસાઇનમેન્ટ અને આઇડેન્ટિટી ઓપરેટરો ઉદાહરણ સાથે સમજાવો.

જવાબ

એરિથમેટિક ઓપરેટરો:

ઓપરેટર	ઓપરેશન	ઉદાહરણ
`+`	બાકીદારી	`5 + 3 = 8`
`-`	બાદબાકી	`5 - 3 = 2`
`*`	ગુણાકાર	`5 * 3 = 15`
`/`	ભાગાકાર	`10 / 3 = 3.33`
`//`	Floor Division	`10 // 3 = 3`
`%`	બાકી	`10 % 3 = 1`
`**`	ઘાત	`2 ** 3 = 8`

એસાઇનમેન્ટ ઓપરેટરો:

ઓપરેટર	ઉદાહરણ	સમકક્ષ
`=`	`x = 5`	મૂલ્ય આપો
`+=`	`x += 3`	`x = x + 3`
`-=`	`x -= 2`	`x = x - 2`
`*=`	`x *= 4`	`x = x * 4`

આઇડેન્ટિટી ઓપરેટરો:

ઓપરેટર	હેતુ	ઉદાહરણ
`is`	સમાન ઓબ્જેક્ટ	`x is y`
`is not`	વિવિધ ઓબ્જેક્ટ	`x is not y`

કોડ ઉદાહરણ:

```
\# Arithmetic
a = 10 + 5    \# 15
b = 10 // 3   \# 3

\# Assignment
x = 5
x += 3        \# x     8

\# Identity
list1 = [1, 2, 3]
list2 = [1, 2, 3]
print(list1 is list2)      \# False
print(list1 is not list2)  \# True
```

યાદી રાખવાની ટ્રિક: ``Add Assign Identity''

પ્રશ્ન 2(અ) [3 ગુણ]

નીચેનામાંથી કયા આઇડેન્ટિફાયર્સ ના નામો અમાન્ય છે? **(i) Total Marks (ii)Total_Marks (iii)total-Marks (iv) Hundred$ (v) _Percentage (vi) True**

જવાબ

આઇડેન્ટિફાયર	માન્ય/અમાન્ય	કારણ
Total Marks	**અમાન્ય**	સ્પેસ છે
Total_Marks	માન્ય	અન્ડરસ્કોર મંજૂર છે
total-Marks	**અમાન્ય**	હાઇફન મંજૂર નથી
Hundred$	**અમાન્ય**	$ સિમ્બોલ મંજૂર નથી
_Percentage	માન્ય	અન્ડરસ્કોરથી શરૂ થઈ શકે છે
True	**અમાન્ય**	આરક્ષિત કીવર્ડ છે

અમાન્ય આઇડેન્ટિફાયર્સ: Total Marks, total-Marks, Hundred$, True
યાદી રાખવાની ટ્રિક: ``સ્પેસ હાઇફન ડોલર કીવર્ડ = અમાન્ય''

પ્રશ્ન 2(બ) [4 ગુણ]

આપેલ ત્રણ સંખ્યાઓમાંથી મહત્તમ સંખ્યા શોધવા માટે પ્રોગ્રામ લખો.

જવાબ

```
\#        input
num1 = float(input("            : "))
num2 = float(input("            : "))
num3 = float(input("             : "))

\# if{-elif{-}else           }
if num1 {=} num2 and num1 {=} num3:
    maximum = num1
elif num2 {=} num1 and num2 {=} num3:
    maximum = num2
else:
    maximum = num3

\#
print(f"        : \{maximum\}")
```

max() ફંક્શન વાપરીને વૈકલ્પિક રીત:

```
num1, num2, num3 = map(float, input("3          : ").split())
maximum = max(num1, num2, num3)
print(f"   : \{maximum\}")
```

યાદી રાખવાની ટ્રિક: ``Input Compare Display''

પ્રશ્ન 2(ક) [7 ગુણ]

પાયથોનમાં ડિક્શનરી સમજાવો. ડિક્શનરીમાં ઘટકો ઉમેરવા, બદલવા અને કાઢી નાખવા માટેના સ્ટેટમેન્ટ લખો.

જવાબ

ડિક્શનરી એ key-value જોડીઓનો સંગ્રહ છે જે ક્રમબદ્ધ, બદલાય તેવો અને ડુપ્લિકેટ keys નથી મંજૂર કરે છે.
ઓપરેશન્સ ટેબલ:

ઓપરેશન	સિન્ટેક્સ	ઉદાહરણ
બનાવો	`dict_name = {}`	`student = {}`
ઉમેરો	`dict[key] = value`	`student['name'] = 'John'`
બદલો	`dict[key] = new_value`	`student['name'] = 'Jane'`
ડિલીટ કરો	`del dict[key]`	`del student['name']`
એક્સેસ કરો	`dict[key]`	`print(student['name'])`

કોડ ઉદાહરણ:

```
\#
student = \{\}

\#
student[{name}] = {John}
student[{age}] = 20
student[{grade}] = {A}

\#
student[{age}] = 21

\#
del student[{grade}]

\#
print(student)  \#    : \{{name: John, age: 21\}}

\#    methods
student.pop({age})           \# Remove      return
student.update(\{{city}: {Mumbai}\)  }\#    items
```

ડિક્શનરીના ગુણધર્મો:

- **ક્રમબદ્ધ**: insertion order જાળવે છે (Python 3.7+)
- **બદલાય તેવું**: બનાવ્યા પછી બદલી શકાય છે
- **ડુપ્લિકેટ્સ નહીં**: Keys અનન્ય હોવા જરૂરી છે

યાદી રાખવાની ટ્રિક: ``Key-Value ક્રમબદ્ધ બદલાય અનન્ય''

પ્રશ્ન 2(અ OR) [3 ગુણ]

નીચેની પેટર્ન દર્શાવવા માટેનો પ્રોગ્રામ લખો.

જવાબ

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

આઉટપુટ:

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

યાદી રાખવાની ટ્રિક: ``બાહ્ય રો આંતરિક કોલમ પ્રિન્ટ''

પ્રશ્ન 2(બ OR) [4 ગુણ]

વપરાશકર્તા દ્વારા દાખલ કરેલ પૂર્ણાંક સંખ્યાના અંકોનો સરવાળો શોધવા માટે પ્રોગ્રામ લખો.

જવાબ

```
\#              input
number = int(input("        : "))
original\_number = number
sum\_digits = 0

\#
while number {} 0:
    digit = number \% 10    \#
    sum\_digits += digit    \#
    number = number // 10  \#

\#
print(f"\{original\_number\}           : \{sum\_digits\}")
```

વૈકલ્પિક રીત:

```
number = input("        : ")
sum\_digits = sum(int(digit) for digit in number)
print(f"       : \{sum\_digits\}")
```

યાદી રાખવાની ટ્રિક: ``Input કાઢો સરવાળો દર્શાવો''

પ્રશ્ન 2(ક OR) [7 ગુણ]

લિસ્ટમાં સ્લાઇસિંગ અને કન્કેટનેશન ઓપરેશન સમજાવો.

જવાબ

લિસ્ટ સ્લાઇસિંગ: લિસ્ટનો ભાગ કાઢવા માટે `[start:stop:step]` સિન્ટેક્સ વાપરવું.
સ્લાઇસિંગ સિન્ટેક્સ ટેબલ:

સિન્ટેક્સ	વર્ણન	ઉદાહરણ
`list[start:stop]`	start થી stop-1 સુધીના ઘટકો	`nums[1:4]`
`list[:stop]`	શરૂઆતથી stop-1 સુધી	`nums[:3]`
`list[start:]`	start થી અંત સુધી	`nums[2:]`
`list[::step]`	step સાથે બધા ઘટકો	`nums[::2]`
`list[::-1]`	રિવર્સ લિસ્ટ	`nums[::-1]`

કન્કેટનેશન: બે અથવા વધુ લિસ્ટને + ઓપરેટર અથવા `extend()` મેથડ વાપરીને જોડવું.
કોડ ઉદાહરણ:

```
\#
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8]

\#
print(list1[1:4])    \# [2, 3, 4]
print(list1[:3])     \# [1, 2, 3]
print(list1[2:])     \# [3, 4, 5]
print(list1[::2])    \# [1, 3, 5]
print(list1[::{-}1])   \# [5, 4, 3, 2, 1]

\#
result1 = list1 + list2           \# [1, 2, 3, 4, 5, 6, 7, 8]
list1.extend(list2)               \# list2  list1
combined = [*list1, *list2]       \# Unpacking operator
```

મુખ્ય મુદ્દા:
- **સ્લાઇસિંગ**: મૂળ લિસ્ટ બદલ્યા વગર નવી લિસ્ટ બનાવે છે
- **કન્કેટનેશન**: લિસ્ટને એક લિસ્ટમાં જોડે છે
- **નેગેટિવ ઇન્ડેક્સિંગ**: `list[-1]` છેલ્લો ઘટક આપે છે

યાદી રાખવાની ટ્રિક: ``સ્લાઇસ કાઢો કન્કેટ જોડો''

પ્રશ્ન 3(અ) [3 ગુણ]

પાયથોનમાં લિસ્ટ વ્યાખ્યાયિત કરો. લિસ્ટના અંતમાં એલિમેન્ટ ઉમેરવા માટે વપરાતા ફંક્શનનું નામ લખો.

જવાબ

લિસ્ટ વ્યાખ્યા: એક **લિસ્ટ** એ આઇટમ્સનો ક્રમબદ્ધ સંગ્રહ છે જે બદલાય તેવો અને ડુપ્લિકેટ મૂલ્યો મંજૂર કરે છે.
ગુણધર્મો ટેબલ:

ગુણધર્મ	વર્ણન
ક્રમબદ્ધ	આઇટમ્સનો નિશ્ચિત ક્રમ છે
બદલાય તેવું	બનાવ્યા પછી બદલી શકાય છે
ડુપ્લિકેટ્સ	ડુપ્લિકેટ મૂલ્યો મંજૂર કરે છે
ઇન્ડેક્સ્ડ	ઇન્ડેક્સ દ્વારા આઇટમ્સ access કરવાય છે

ઘટક ઉમેરવા માટેનું ફંક્શન: `append()`
ઉદાહરણ:

```
\#
fruits = [{apple}, {banana}]

\#
fruits.append({orange})
print(fruits)  \# [{apple, banana, orange]}
```

યાદી રાખવાની ટ્રિક: ``લિસ્ટ append અંત''

પ્રશ્ન 3(બ) [4 ગુણ]

પાયથોનમાં ટ્યુપલ વ્યાખ્યાયિત કરો. ટ્યુપલના છેલ્લા એલિમેન્ટને એક્સેસ કરવા માટેનું સ્ટેટમેન્ટ લખો.

જવાબ

ટ્યુપલ વ્યાખ્યા: એક **ટ્યુપલ** એ આઇટમ્સનો ક્રમબદ્ધ સંગ્રહ છે જે બદલાય તેવો નથી અને ડુપ્લિકેટ મૂલ્યો મંજૂર કરે છે.
ગુણધર્મો ટેબલ:

ગુણધર્મ	વર્ણન
ક્રમબદ્ધ	આઇટમ્સનો નિશ્ચિત ક્રમ છે
બદલાય તેવું નથી	બનાવ્યા પછી બદલી શકાય નહીં
ડુપ્લિકેટ્સ	ડુપ્લિકેટ મૂલ્યો મંજૂર કરે છે
ઇન્ડેક્સ્ડ	ઇન્ડેક્સ દ્વારા આઇટમ્સ access કરવાય છે

છેલ્લા એલિમેન્ટને એક્સેસ કરવું:

```
\#    1:
my\_tuple = (10, 20, 30, 40, 50)
last\_element = my\_tuple[{-}1]
print(last\_element)  \#    : 50

\#    2: length
last\_element = my\_tuple[len(my\_tuple) {-} 1]
print(last\_element)  \#    : 50
```

યાદી રાખવાની ટ્રિક: ``ટ્યુપલ બદલાય નહિ નેગેટિવ ઇન્ડેક્સ''

પ્રશ્ન 3(ક) [7 ગુણ]

નીચેના સેટ ઓપરેશન્સ માટે સ્ટેટમેન્ટ લખો: ખાલી સેટ બનાવો, સેટમાં એક ઘટક ઉમેરો, સેટમાંથી એક ઘટક દૂર કરો, બે સેટનું યુનિયન, બે સેટનું છેદ, બે સેટ વચ્ચેનો તફાવત અને બે સેટ વચ્ચે સિમેટ્રિક તફાવત.

જવાબ

સેટ ઓપરેશન્સ ટેબલ:

ઓપરેશન	મેથડ	ઓપરેટર	ઉદાહરણ
ખાલી બનાવો	`set()`	-	`s = set()`
ઘટક ઉમેરો	`add()`	-	`s.add(5)`
ઘટક દૂર કરો	`remove()`	-	`s.remove(5)`
યુનિયન	`union()`	`|`	`A.union(B)` અથવા `A | B`
છેદ	`intersection()`	`&`	`A.intersection(B)` અથવા `A & B`
તફાવત	`difference()`	`-`	`A.difference(B)` અથવા `A - B`
સિમેટ્રિક તફાવત	`symmetric_difference()`		`A.symmetric_difference(B)` અથવા `A ^ B`

કોડ ઉદાહરણ:

```
\#
my\_set = set()

\#
my\_set.add(10)
my\_set.add(20)

\#
my\_set.remove(10)

\#
A = \{1, 2, 3, 4\}
B = \{3, 4, 5, 6\}

\#     (        )
union\_result = A.union(B)        \# \{1, 2, 3, 4, 5, 6\}

\#   (       )
intersection\_result = A.intersection(B)  \# \{3, 4\}

\#     (A {- B)}
difference\_result = A.difference(B)      \# \{1, 2\}

\#          (A    B    ,         )
sym\_diff\_result = A.symmetric\_difference(B)  \# \{1, 2, 5, 6\}
```

યાદી રાખવાની ટ્રિક: ``બનાવો ઉમેરો દૂર કરો યુનિયન છેદ તફાવત સિમેટ્રિક''

પ્રશ્ન 3(અ OR) [3 ગુણ]

પાયથોનમાં સ્ટ્રિંગ વ્યાખ્યાયિત કરો. ઉદાહરણ વાપરીને સમજાવો (i) સ્ટ્રિંગ કેવી રીતે બનાવવી. (ii) ઇન્ડેક્સિંગનો ઉપયોગ કરીને વ્યક્તિગત અક્ષરોને એક્સેસ કરવું.

જવાબ

સ્ટ્રિંગ વ્યાખ્યા: એક **સ્ટ્રિંગ** એ અવતરણચિહ્ન (સિંગલ અથવા ડબલ) માં બંધ કરેલા અક્ષરોનો ક્રમ છે.
(i) સ્ટ્રિંગ બનાવવું:

```
\#
name = {Python}

\#
message = "Hello World"

\#         (     )
text = """
          """
```

(ii) અક્ષરોને એક્સેસ કરવું:

```
word = "PYTHON"
print(word[0])    \# P (      )
print(word[2])    \# T (       )
print(word[{-}1])   \# N (      )
print(word[{-}2])   \# O (       )
```

યાદી રાખવાની ટ્રિક: ``સ્ટ્રિંગ અવતરણ ઇન્ડેક્સ એક્સેસ''

પ્રશ્ન 3(બ OR) [4 ગુણ]

ફોર લૂપ અને વ્હાઇલ લૂપનો ઉપયોગ કરીને લિસ્ટ ટ્રાવર્સિંગ સમજાવો.

જવાબ

લિસ્ટ ટ્રાવર્સિંગ મતલબ લિસ્ટના દરેક ઘટકને એક પછી એક મુલાકાત લેવી.
ફોર લૂપ ટ્રાવર્સિંગ:

```
numbers = [10, 20, 30, 40, 50]

\#    1:    iteration
for num in numbers:
    print(num)

\#    2:
for i in range(len(numbers)):
    print(f"    \{i\}: \{numbers[i]\}")
```

વ્હાઇલ લૂપ ટ્રાવર્સિંગ:

```
numbers = [10, 20, 30, 40, 50]
i = 0

while i {} len(numbers):
    print(f"    \{i\}     : \{numbers[i]\}")
    i += 1
```

તુલના ટેબલ:

લૂપ પ્રકાર	ફાયદો	ઉપયોગ
ફોર લૂપ	સરળ સિન્ટેક્સ	જ્યારે iteration ની સંખ્યા ખબર હોય
વ્હાઇલ લૂપ	વધુ નિયંત્રણ	જ્યારે શરત આધારિત iteration જોઈએ

યાદી રાખવાની ટ્રિક: ``ફોર સરળ વ્હાઇલ નિયંત્રણ''

પ્રશ્ન 3(ક OR) [7 ગુણ]

એક એવો પ્રોગ્રામ લખો કે જેનાથી વર્ગમાં n વિદ્યાર્થીઓના રોલ નંબર, નામ અને માર્ક્સ સાથેની ડિક્શનરી બનાવી શકાય અને 75 થી વધુ ગુણ મેળવનારા વિદ્યાર્થીઓના નામ ડિસ્પ્લે કરી શકાય.

જવાબ

```
\#           input
n = int(input("               : "))

\#
students = \{\}

\#           input
for i in range(n):
    print(f"{n}      \{i + 1\}            :")
    roll\_no = int(input("     : "))
    name = input("  : ")
    marks = float(input("   : "))

    \#
```

```
    students[roll\_no] = \{
        {name}: name,
        {marks}: marks
    \}

\# 75
print("{n}75                :")
print("{-"} * 30)

high\_performers = []
for roll\_no, data in students.items():
    if data[{marks}] {} 75:
        high\_performers.append(data[{name}])
        print(f"  : \{data[{name}]\},    : \{data[{marks}]\}")

if not high\_performers:
    print("        75             ")
else:
    print(f"{n}         : \{len(high\_performers)\}")
```

સેમ્પલ આઉટપુટ:

```
               : 2

     1           :
    : 101
  : John
   : 80

     2           :
    : 102
  : Alice
   : 70

75                :
------------------------------
  : John,    : 80.0

         : 1
```

યાદી રાખવાની ટ્રિક: ``Input સ્ટોર ફિલ્ટર ડિસ્પ્લે''

પ્રશ્ન 4(અ) [3 ગુણ]

રેન્ડમ મોડ્યુલમાં ઉપલબ્ધ કોઈપણ ત્રણ ફંક્શન લખો. દરેક ફંક્શનનું સિન્ટેક્સ અને ઉદાહરણ લખો.

જવાબ

રેન્ડમ મોડ્યુલ ફંક્શન્સ:

ફંક્શન	સિન્ટેક્સ	હેતુ	ઉદાહરણ
random()	`random.random()`	0.0 થી 1.0 સુધી રેન્ડમ ફ્લોટ	`0.7534`
randint()	`random.randint(a, b)`	a થી b સુધી રેન્ડમ ઇન્ટીજર	`randint(1, 10)`
choice()	`random.choice(seq)`	સિક્વન્સમાંથી રેન્ડમ ઘટક	`choice(['a', 'b', 'c'])`

કોડ ઉદાહરણ:

```
import random

\# random() {- 0.0   1.0          }
num = random.random()
print(num)  \#     : 0.7234567

\# randint() {-                }
dice = random.randint(1, 6)
print(dice)  \#     : 4

\# choice() {-                  }
colors = [{red}, {blue}, {green}]
selected = random.choice(colors)
print(selected)  \#     : {blue}
```

યાદી રાખવાની ટ્રિક: ``Random Randint Choice''

પ્રશ્ન 4(બ) [4 ગુણ]

ફંક્શનના ફાયદા લખો.

જવાબ

ફંક્શનના ફાયદા:

ફાયદો	વર્ણન
કોડ પુનઃઉપયોગ	એકવાર લખો, અનેકવાર વાપરો
મોડ્યુલરિટી	મોટા પ્રોગ્રામને નાના ભાગોમાં વહેંચો
સરળ ડીબગિંગ	એરર્સ અલગ પાડીને સહેલાઈથી ઠીક કરો
વાંચવાની સુવિધા	કોડ વધુ સંગઠિત અને સ્પષ્ટ બનાવે છે
જાળવણી	સહેલાઈથી અપડેટ અને બદલાવ કરી શકાય છે
પુનરાવર્તન ટાળો	ડુપ્લિકેટ કોડ ઓછો કરે છે

ઉદાહરણ:

```
\#       (     )
num1 = 5
square1 = num1 * num1
print(square1)

num2 = 8
square2 = num2 * num2
print(square2)

\#       (     )
def calculate\_square(num):
    return num * num

print(calculate\_square(5))  \# 25
print(calculate\_square(8))  \# 64
```

યાદી રાખવાની ટ્રિક: ``પુનઃઉપયોગ મોડ્યુલર ડીબગ વાંચો જાળવો ટાળો''

પ્રશ્ન 4(ક) [7 ગુણ]

એક પ્રોગ્રામ લખો જે વપરાશકર્તાને સ્ટ્રિંગ માટે પૂછે અને સ્ટ્રિંગમાં દરેક `a' નું સ્થાન પ્રિન્ટ કરે.

જવાબ

```
\#              input
text = input("         : ")

\# {a      positions   }
positions = []
for i in range(len(text)):
    if text[i].lower() == {a}:  \# {a   A         }
        positions.append(i)

\#
if positions:
    print(f"   {a   positions     : }\{positions\}")
    print("        :")
    for pos in positions:
        print(f"Position \{pos\}: {}\{text[pos]\}{"})
else:
    print("   {a          "})

\# enumerate
print("{n}      :")
for index, char in enumerate(text):
    if char.lower() == {a}:
        print(f"{a position }\{index\}     ")
```

સેમ્પલ આઉટપુટ:

```
         : Python Programming

   'a'   positions     : [12]
        :
Position 12: 'a'

      :
'a' position 12
```

સુધારેલું વર્ઝન:

```
text = input("         : ")
count = 0

print(f"{}\{text\}{   a         "})
print("{-"} * 30)

for i, char in enumerate(text):
    if char.lower() == {a}:
        count += 1
        print(f"{a     }\{i\}      (   : {}\{char\}{)"})

print(f"{n}{a         : }\{count\}")
```

યાદી રાખવાની ટ્રિક: ``Input લૂપ ચેક સ્ટોર ડિસ્પ્લે''

પ્રશ્ન 4(અ OR) [3 ગુણ]

લોકલ અને ગ્લોબલ વેરિયેબલ સમજાવો.

જવાબ

વેરિયેબલ સ્કોપ પ્રકારો:

વેરિયેબલ પ્રકાર	સ્કોપ	એક્સેસ	ઉદાહરણ
લોકલ	ફંક્શનની અંદર જ	ફંક્શનની અંદર	`def func(): x = 5`
ગ્લોબલ	આખા પ્રોગ્રામમાં	પ્રોગ્રામમાં ગમે ત્યાં	`x = 5` (ફંક્શનની બહાર)

કોડ ઉદાહરણ:

```
\#
global\_var = "      "

def my\_function():
    \#
    local\_var = "      "
    print(global\_var)  \#
    print(local\_var)   \#

my\_function()
print(global\_var)      \#
\# print(local\_var)     \#    {-               }
```

ગ્લોબલ કીવર્ડ:

```
counter = 0  \#

def increment():
    global counter  \#              declare
    counter += 1

increment()
print(counter)  \#    : 1
```

યાદી રાખવાની ટ્રિક: ``લોકલ અંદર ગ્લોબલ બધે''

પ્રશ્ન 4(બ OR) [4 ગુણ]

યુઝર ડિફાઇન્ડ ફંક્શનનું બનાવટ અને ઉપયોગ ઉદાહરણ સાથે સમજાવો.

જવાબ

ફંક્શન બનાવટ સિન્ટેક્સ:

```
def function\_name(parameters):
    """    docstring"""
    \#
    return value  \#
```

ફંક્શનના ઘટકો:

ઘટક	હેતુ	ઉદાહરણ
def	ફંક્શન વ્યાખ્યાયિત કરવાનું કીવર્ડ	`def`
function_name	ફંક્શનનું નામ	`calculate_area`
parameters	ઇનપુટ મૂલ્યો	`(length, width)`
return	આઉટપુટ મૂલ્ય	`return result`

ઉદાહરણ:

```
\#
def greet\_user(name, age):
    """                              """
    message = f"   \{name\}!        \{age\}     ."
    return message

\#
user\_name = "   "
user\_age = 25
greeting = greet\_user(user\_name, user\_age)
print(greeting)  \#    :      !        25     .

\#
def calculate\_power(base, exponent=2):
    return base ** exponent

print(calculate\_power(5))      \# 25 (     exponent=2     )
print(calculate\_power(5, 3))  \# 125 (exponent=3     )
```

યાદી રાખવાની ટ્રિક: ``વ્યાખ્યાયિત કૉલ રિટર્ન પેરામીટર''

પ્રશ્ન 4(ક OR) [7 ગુણ]

calcFact() નામનું યુઝર ડિફાઇન્ડ ફંક્શન બનાવવા માટેનો પ્રોગ્રામ લખો કે જે આર્ગ્યુમેન્ટ તરીકે આપેલ સંખ્યાના ફેક્ટોરિયલની ગણતરી કરી તેને ડિસ્પ્લે કરે.

જવાબ

```
def calcFact(number):
    """

      : number (integer)
       : factorial (integer)
    """
    if number {} 0:
        return "                        "
    elif number == 0 or number == 1:
        return 1
    else:
        factorial = 1
        for i in range(2, number + 1):
            factorial *= i
        return factorial

\#
try:
    \#
    num = int(input("        : "))

    \#
    result = calcFact(num)

    \#
    if isinstance(result, str):
        print(result)
    else:
        print(f"\{num\}         : \{result\}")
```

```
except ValueError:
    print("                  ")

\#
print("{n}            :")
test\_values = [0, 1, 5, 10, {-}3]
for val in test\_values:
    result = calcFact(val)
    print(f"calcFact(\{val\}) = \{result\}")
```

રીકર્સિવ વર્ઝન:

```
def calcFactRecursive(n):
    """                    """
    if n {} 0:
        return "               "
elif

n == 0 or

n == 1:

        return 1
    else:
        return n * calcFactRecursive(n {-} 1)

\#
number = int(input("        : "))
result = calcFactRecursive(number)
print(f"     : \{result\}")
```

સેમ્પલ આઉટપુટ:

```
        : 5
5         : 120

            :
calcFact(0) = 1
calcFact(1) = 1
calcFact(5) = 120
calcFact(10) = 3628800
calcFact(-3) =
```

યાદી રાખવાની ટ્રિક: ``વ્યાખ્યાયિત ચેક લૂપ ગુણાકાર રિટર્ન''

પ્રશ્ન 5(અ) [3 ગુણ]

ક્લાસ અને ઓબ્જેક્ટ વચ્ચે તફાવત આપો.

જવાબ

ક્લાસ વર્સિસ ઓબ્જેક્ટ તુલના:

બાબત	ક્લાસ	ઓબ્જેક્ટ
વ્યાખ્યા	બ્લૂપ્રિન્ટ/ટેમ્પ્લેટ	ક્લાસનું ઇન્સ્ટન્સ
મેમરી	મેમરી એલોકેટ નથી	મેમરી એલોકેટ છે
બનાવટ	`class` કીવર્ડ વાપરીને વ્યાખ્યાયિત	ક્લાસના નામ વાપરીને બનાવાય છે
એટ્રિબ્યુટ્સ	વ્યાખ્યાયિત પરંતુ ઇનિશિયલાઇઝ નથી	વાસ્તવિક મૂલ્યો છે

ઉદાહરણ `class Car:` `my_car = Car()`

કોડ ઉદાહરણ:

```
\#        (    )
class Student:
    def \_\_init\_\_(self, name, age):
        self.name = name
        self.age = age

\#        (    )
student1 = Student("  ", 20)  \#     1
student2 = Student("  ", 19) \#     2

print(student1.name)  \#
print(student2.name)  \#
```

યાદી રાખવાની ટ્રિક: ``કલાસ બ્લૂપ્રિન્ટ ઓબ્જેક્ટ ઇન્સ્ટન્સ''

પ્રશ્ન 5(બ) [4 ગુણ]

કલાસમાં કન્સ્ટ્રક્ટરનો હેતુ જણાવો.

જવાબ

કન્સ્ટ્રક્ટરનો હેતુ:

હેતુ	વર્ણન
ઓબ્જેક્ટ ઇનિશિયલાઇઝ કરો	એટ્રિબ્યુટ્સને પ્રારંભિક મૂલ્યો આપો
ઓટોમેટિક એક્ઝીક્યુશન	ઓબ્જેક્ટ બનાવતી વખતે આપોઆપ કૉલ થાય છે
મેમરી સેટઅપ	ઓબ્જેક્ટ એટ્રિબ્યુટ્સ માટે મેમરી એલોકેટ કરે છે
ડિફોલ્ટ મૂલ્યો	એટ્રિબ્યુટ્સને ડિફોલ્ટ મૂલ્યો આપે છે

કન્સ્ટ્રક્ટરના પ્રકારો:

પ્રકાર	વર્ણન	ઉદાહરણ
ડિફોલ્ટ	કોઈ પેરામીટર નથી	`def __init__(self):`
પેરામીટરાઇઝ્ડ	પેરામીટર લે છે	`def __init__(self, name):`

ઉદાહરણ:

```
class Rectangle:
    def \_\_init\_\_(self, length=0, width=0):  \#
        self.length = length  \#
        self.width = width    \#
        print("         !")

    def area(self):
        return self.length * self.width

\#        {-               }
rect1 = Rectangle(10, 5)  \#    :          !
rect2 = Rectangle()       \#

print(rect1.area())       \# 50
print(rect2.area())       \# 0
```

યાદી રાખવાની ટ્રિક: ``ઇનિશિયલાઇઝ ઓટોમેટિક મેમરી ડિફોલ્ટ''

પ્રશ્ન 5(ક) [7 ગુણ]

``Student'' નામનો ક્લાસ બનાવવા માટે પ્રોગ્રામ લખો જેમાં નામ, રોલ નંબર અને માર્ક્સ જેવા એટ્રિબ્યુટ્સ હોય. વિદ્યાર્થીની માહિતી પ્રદર્શિત કરવાની મેથડ બનાવો. ``Student'' ક્લાસનો ઓબ્જેક્ટ બનાવો અને મેથડનો ઉપયોગ કેવી રીતે કરવો તે બતાવો.

જવાબ

```
class Student:
    def \_\_init\_\_(self, name, roll\_number, marks):
        """                              """
        self.name = name
        self.roll\_number = roll\_number
        self.marks = marks

    def display\_info(self):
        """                    """
        print("{-"} * 30)
        print("          ")
        print("{-"} * 30)
        print(f"  : \{self.name\}")
        print(f"     : \{self.roll\_number\}")
        print(f"   : \{self.marks\}")
        print("{-"} * 30)

    def calculate\_grade(self):
        """                      """
        if self.marks {=} 90:
            return {A+}
        elif self.marks {=} 80:
            return {A}
        elif self.marks {=} 70:
            return {B}
```

```
        elif self.marks {=} 60:
            return {C}
        else:
            return {F}

    def display\_grade(self):
        """          """
        grade = self.calculate\_grade()
        print(f"  : \{grade\}")

\# Student
print("Student              :")
student1 = Student("     ", 101, 85)
student2 = Student("      ", 102, 92)
student3 = Student("      ", 103, 78)

\#
print("{n}===      1       ===")
student1.display\_info()
student1.display\_grade()

print("{n}===      2       ===")
student2.display\_info()
student2.display\_grade()

print("{n}===      3       ===")
student3.display\_info()
student3.display\_grade()

\#
print(f"{n}      {-      1     : }\{student1.name\}")
print(f"      {-      2      : }\{student2.marks\}")
```

સેમ્પલ આઉટપુટ:

```
Student              :

===      1       ===
------------------------------

------------------------------
  :
     : 101
   : 85
------------------------------
  : A

===      2       ===
------------------------------

------------------------------
  :
     : 102
   : 92
------------------------------
  : A+
```

ક્લાસના ઘટકો:

- **એટ્રિબ્યુટ્સ**: name, roll_number, marks
- **કન્સ્ટ્રક્ટર**: `__init__()` મેથડ
- **મેથડ્સ**: display_info(), calculate_grade(), display_grade()
- **ઓબ્જેક્ટ્સ**: student1, student2, student3

યાદી રાખવાની ટ્રિક: ``ક્લાસ એટ્રિબ્યુટ્સ કન્સ્ટ્રક્ટર મેથડ્સ ઓબ્જેક્ટ્સ''

પ્રશ્ન 5(અ OR) [3 ગુણ]

એન્કેપ્સ્યુલેશનનો હેતુ જણાવો.

જવાબ

એન્કેપ્સ્યુલેશનનો હેતુ:

હેતુ	વર્ણન
ડેટા છુપાવવું	આંતરિક implementation વિગતો છુપાવે છે
ડેટા સુરક્ષા	અનધિકૃત એક્સેસથી ડેટાને બચાવે છે
નિયંત્રિત એક્સેસ	મેથડ્સ દ્વારા નિયંત્રિત એક્સેસ આપે છે
કોડ સિક્યુરિટી	ડેટાના આકસ્મિક ફેરફારને અટકાવે છે
મોડ્યુલરિટી	સંબંધિત ડેટા અને મેથડ્સ એકસાથે રાખે છે

અમલીકરણ ઉદાહરણ:

```
class BankAccount:
    def \_\_init\_\_(self, balance):
        self.\_\_balance = balance  \#

    def get\_balance(self):       \# Getter
        return self.\_\_balance

    def deposit(self, amount):   \#
        if amount {} 0:
            self.\_\_balance += amount

account = BankAccount(1000)
print(account.get\_balance())     \# 1000
\# print(account.\_\_balance)       \#    {-               }
```

ફાયદા:
- **સિક્યુરિટી**: ડેટાને સીધી એક્સેસ કરી શકાતી નથી
- **જાળવણી**: આંતરિક implementation સહેલાઈથી બદલી શકાય છે
- **વેલિડેશન**: getter/setter મેથડ્સમાં વેલિડેશન ઉમેરી શકાય છે

યાદી રાખવાની ટ્રિક: ``છુપાવો સુરક્ષા નિયંત્રણ સિક્યુર મોડ્યુલર''

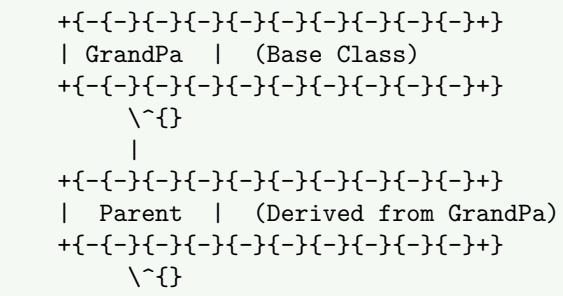
પ્રશ્ન 5(બ OR) [4 ગુણ]

મલ્ટિલેવલ ઇન્હેરિટન્સ સમજાવો.

જવાબ

મલ્ટિલેવલ ઇન્હેરિટન્સ એ જ્યારે એક ક્લાસ બીજી ક્લાસમાંથી ઇન્હેરિટ કરે છે, જે બદલામાં બીજી ક્લાસમાંથી ઇન્હેરિટ કરે છે, આમ એક શૃંખલા બને છે.

સ્ટ્રક્ચર ડાયાગ્રામ:



```
        |
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
    |  Child   |  (Derived from Parent)
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
```

લક્ષણો ટેબલ:

લેવલ	ક્લાસ	ઇન્હેરિટ કરે છે	એક્સેસ કરે છે
લેવલ 1	GrandPa	કોઈથી નહીં	પોતાની મેથડ્સ
લેવલ 2	Parent	GrandPa	GrandPa + પોતાની મેથડ્સ
લેવલ 3	Child	Parent	GrandPa + Parent + પોતાની

કોડ ઉદાહરણ:

```
\#    1 {-      }
class Vehicle:
    def \_\_init\_\_(self, brand):
        self.brand = brand

    def start(self):
        print(f"\{self.brand\}       ")

\#    2 {- Vehicle       }
class Car(Vehicle):
    def \_\_init\_\_(self, brand, model):
        super().\_\_init\_\_(brand)
        self.model = model

    def drive(self):
        print(f"\{self.brand\} \{self.model\}       ")

\#    3 {- Car       }
class SportsCar(Car):
    def \_\_init\_\_(self, brand, model, top\_speed):
        super().\_\_init\_\_(brand, model)
        self.top\_speed = top\_speed

    def race(self):
        print(f"\{self.brand\} \{self.model\} \{self.top\_speed\} km/h           ")

\#
ferrari = SportsCar("Ferrari", "F8", 340)
ferrari.start()     \# Vehicle
ferrari.drive()     \# Car
ferrari.race()      \# SportsCar
```

યાદી રાખવાની ટ્રિક: ``શૃંખલા ઇન્હેરિટ લેવલ એક્સેસ''

પ્રશ્ન 5(ક OR) [7 ગુણ]

હાઇબ્રિડ ઇન્હેરિટન્સનું કાર્ય દર્શાવતો પાયથોન પ્રોગ્રામ લખો.

જવાબ

હાઇબ્રિડ ઇન્હેરિટન્સ એક પ્રોગ્રામમાં બહુવિધ પ્રકારની ઇન્હેરિટન્સ (સિંગલ, મલ્ટિપલ, મલ્ટિલેવલ) ને જોડે છે.
સ્ટ્રક્ચર ડાયાગ્રામ:

```
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
    |  Animal  |  (Base Class)
```

```
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
         \^{}
         |
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
    | Mammal   |  (Single Inheritance)
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
         \^{}
         |
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+     +{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
    |   Dog    |     |   Bird   |  (Single Inheritance)
    +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}+     +{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
         \^{                 \^{}}
         |                 |
         +{-{-}{-}{-}{-}{-}{-}+{-}{-}{-}{-}{-}{-}{-}{-}+}
                 |
         +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
         |  FlyingDog    |  (Multiple Inheritance)
         +{-{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}+}
```

કોડ ઉદાહરણ:

```
\#
class Animal:
    def \_\_init\_\_(self, name):
        self.name = name
        print(f"    \{self.name\}   ")

    def eat(self):
        print(f"\{self.name\}    ")

    def sleep(self):
        print(f"\{self.name\}    ")

\# Animal
class Mammal(Animal):
    def \_\_init\_\_(self, name, fur\_color):
        super().\_\_init\_\_(name)
        self.fur\_color = fur\_color

    def give\_birth(self):
        print(f"\{self.name\}              ")

\# Animal
class Bird(Animal):
    def \_\_init\_\_(self, name, wing\_span):
        super().\_\_init\_\_(name)
        self.wing\_span = wing\_span

    def fly(self):
        print(f"\{self.name\} \{self.wing\_span\}cm         ")

    def lay\_eggs(self):
        print(f"\{self.name\}      ")

\# Mammal
class Dog(Mammal):
    def \_\_init\_\_(self, name, fur\_color, breed):
        super().\_\_init\_\_(name, fur\_color)
        self.breed = breed

    def bark(self):
        print(f"\{self.name\} \{self.breed\}    ")
```

```
    def guard(self):
        print(f"\{self.name\}          ")

\# Dog   Bird              (    )
class FlyingDog(Dog, Bird):
    def \_\_init\_\_(self, name, fur\_color, breed, wing\_span):
        \#
        Dog.\_\_init\_\_(self, name, fur\_color, breed)
        Bird.\_\_init\_\_(self, name, wing\_span)
        print(f"   \{self.name\}      mammal   bird        !")

    def fly\_and\_bark(self):
        print(f"\{self.name\}              !")

    def show\_abilities(self):
        print(f"{n}\{self.name\}      :")
        print("{-"} * 25)
        self.eat()          \# Animal
        self.sleep()        \# Animal
        self.give\_birth()   \# Mammal
        self.bark()         \# Dog
        self.guard()        \# Dog
        self.fly()          \# Bird
        self.lay\_eggs()     \# Bird
        self.fly\_and\_bark() \#

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

\#
print("1.                  :")
dog1 = Dog("  ", "   ", "    ")
dog1.bark()
dog1.guard()

print("{n}2.                 :")
bird1 = Bird("  ", 200)
bird1.fly()
bird1.lay\_eggs()

print("{n}3.                    :")
flying\_dog = FlyingDog("   ", "   ", "   ", 150)
flying\_dog.show\_abilities()

\#
print(f"{n}FlyingDog              :")
for i, cls in enumerate(FlyingDog.\_\_mro\_\_):
    print(f"\{i+1\}. \{cls.\_\_name\_\_\}")
```

સેમ્પલ આઉટપુટ:

```
===              ===

1.                 :




2.                 :

   200cm
```

```
3.                    :


             mammal    bird         !

          :
-------------------------




     150cm

                  !
```

આ ઉદાહરણમાં ઇન્હેરિટન્સના પ્રકારો:

1. **સિંગલ**: Mammal $\leftarrow Animal, Bird \leftarrow Animal, Dog \leftarrow Mammal$
1. **મલ્ટિપલ**: FlyingDog $\leftarrow Dog + Bird$
1. **મલ્ટિલેવલ**: FlyingDog $\leftarrow Dog \leftarrow Mammal \leftarrow Animal$
1. **હાઇબ્રિડ**: ઉપરોક્ત બધાનું સંયોજન

મુખ્ય લક્ષણો:

- **મલ્ટિપલ પેરેન્ટ ક્લાસ**: FlyingDog Dog અને Bird બંનેમાંથી ઇન્હેરિટ કરે છે
- **મેથડ રિઝોલ્યુશન ઓર્ડર**: Python MRO ને અનુસરીને મેથડ conflicts ને હલ કરે છે
- **super() નો ઉપયોગ**: પેરેન્ટ ક્લાસના યોગ્ય ઇનિશિયલાઇઝેશન માટે
- **સંયુક્ત કાર્યક્ષમતા**: બધી પેરેન્ટ ક્લાસની મેથડ્સ તકે પહોંચ

યાદી રાખવાની ટ્રિક: ``હાઇબ્રિડ મલ્ટિપલ સિંગલ મલ્ટિલેવલ સંયુક્ત''