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 બને છે 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:

```
# ત્રણ સંખ્યાઓ input લો
num1 = float(input("પ્રથમ સંખ્યા દાખલ કરો: "))
num2 = float(input("બીજી સંખ્યા દાખલ કરો: "))
num3 = float(input("ત્રીજી સંખ્યા દાખલ કરો: "))

# if-elif-else વાપરીને મહત્તમ શોધો
if num1 >= num2 and num1 >= num3:
    maximum = num1
elif num2 >= num1 and num2 >= num3:
    maximum = num2
else:
    maximum = num3

# પરિણામ દર્શાવો
print(f"મહત્તમ સંખ્યા છે: {maximum}")
```

max() ફંક્શન વાપરીને વૈકલ્પિક રીત:

```
num1, num2, num3 = map(float, input("3 સંખ્યાઓ દાખલ કરો: ").split())
maximum = max(num1, num2, num3)
print(f"મહત્તમ: {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:

```
# ખાલી ડિક્શનરી બનાવો
student = {}

# ઘટકો ઉમેરો
student['name'] = 'John'
student['age'] = 20
student['grade'] = 'A'

# ઘટક બદલો
student['age'] = 21

# ઘટક ડિલીટ કરો
del student['grade']

# ડિક્શનરી દર્શાવો
print(student)  # આઉટપુટ: {'name': 'John', 'age': 21}

# અન્ય methods
student.pop('age')           # Remove અને મૂલ્ય return કરે
student.update({'city': 'Mumbai'})  # અનેક items ઉમેરો
```

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:

```
# પેટર્ન પ્રોગ્રામ
for i in range(1, 6):
    for j in range(1, i + 1):
        print(j, end=" ")
    print()  # દરેક રો પછી નવી લાઇન
```

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:

```
# વપરાશકર્તા પાસેથી સંખ્યા input લો
number = int(input("સંખ્યા દાખલ કરો: "))
original_number = number
sum_digits = 0

# અંકો કાઢો અને સરવાળો કરો
while number > 0:
    digit = number % 10    # છેલ્લો અંક મેળવો
    sum_digits += digit    # સરવાળામાં ઉમેરો
    number = number // 10  # છેલ્લો અંક દૂર કરો

# પરિણામ દર્શાવો
print(f"{original_number} ના અંકોનો સરવાળો છે: {sum_digits}")
```

વૈકલ્પિક રીત:

```
number = input("સંખ્યા દાખલ કરો: ")
sum_digits = sum(int(digit) for digit in number)
print(f"અંકોનો સરવાળો: {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:

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

# સ્લાઇસિંગ ઓપરેશન્સ
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]

# કન્કેટનેશન ઓપરેશન્સ
result1 = list1 + list2          # [1, 2, 3, 4, 5, 6, 7, 8]
list1.extend(list2)              # list2 ને list1 માં ઉમેરે છે
combined = [*list1, *list2]      # Unpacking operator વાપરીને
```

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:

```
# લિસ્ટ બનાવો
fruits = ['apple', 'banana']

# અંતમાં ઘટક ઉમેરો
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:

```
# મેથડ 1: નેગેટિવ ઇન્ડેક્સ વાપરીને
my_tuple = (10, 20, 30, 40, 50)
last_element = my_tuple[-1]
print(last_element)  # આઉટપુટ: 50

# મેથડ 2: length વાપરીને
last_element = my_tuple[len(my_tuple) - 1]
print(last_element)  # આઉટપુટ: 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:

```
# ખાલી સેટ બનાવો
my_set = set()

# ઘટકો ઉમેરો
my_set.add(10)
my_set.add(20)

# ઘટક દૂર કરો
my_set.remove(10)

# ઓપરેશન્સ માટે બે સેટ બનાવો
A = {1, 2, 3, 4}
B = {3, 4, 5, 6}

# યુનિયન (બધા અનન્ય ઘટકો)
union_result = A.union(B)         # {1, 2, 3, 4, 5, 6}

# છેદ (સામાન્ય ઘટકો)
intersection_result = A.intersection(B)  # {3, 4}

# તફાવત (A - B)
difference_result = A.difference(B)      # {1, 2}

# સિમેટ્રિક તફાવત (A અથવા B માં છે, પરંતુ બંનેમાં નહીં)
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:

```
# સિંગલ અવતરણ
name = 'Python'

# ડબલ અવતરણ
message = "Hello World"

# ટ્રિપલ અવતરણ (મલ્ટિલાઇન)
text = """આ એક
મલ્ટિલાઇન સ્ટ્રિંગ છે"""
```

(ii) અક્ષરોને એક્સેસ કરવું:

```
word = "PYTHON"
print(word[0])    # P (પ્રથમ અક્ષર)
print(word[2])    # T (ત્રીજો અક્ષર)
print(word[-1])   # N (છેલ્લો અક્ષર)
print(word[-2])   # O (છેલ્લાથી બીજો)
```

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]

# મેથડ 1: સીધો iteration
for num in numbers:
    print(num)

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

વ્હાઇલ લૂપ ટ્રાવર્સિંગ:

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

while i < len(numbers):
    print(f"ઇન્ડેક્સ {i} પર ઘટક: {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:

```
# વિદ્યાર્થીઓની સંખ્યા input કરો
n = int(input("વિદ્યાર્થીઓની સંખ્યા દાખલ કરો: "))

# ખાલી ડિક્શનરી બનાવો
students = {}

# વિદ્યાર્થીઓનો ડેટા input કરો
for i in range(n):
    print(f"\nવિદ્યાર્થી {i + 1} ની વિગતો દાખલ કરો:")
    roll_no = int(input("રોલ નંબર: "))
    name = input("નામ: ")
    marks = float(input("માર્ક્સ: "))

    # ડિક્શનરીમાં સ્ટોર કરો
    students[roll_no] = {
        'name': name,
        'marks': marks
    }

# 75 થી વધુ માર્ક્સ વાળા વિદ્યાર્થીઓ દર્શાવો
print("\n75 થી વધુ માર્ક્સ વાળા વિદ્યાર્થીઓ:")
print("-" * 30)

high_performers = []
for roll_no, data in students.items():
```

```
    if data['marks'] > 75:
        high_performers.append(data['name'])
        print(f"નામ: {data['name']}, માર્ક્સ: {data['marks']}")

if not high_performers:
    print("કોઈ વિદ્યાર્થીએ 75 થી વધુ માર્ક્સ મેળવ્યા નથી")
else:
    print(f"\nકુલ હાઇ પર્ફોર્મર્સ: {len(high_performers)}")
```

સેમ્પલ આઉટપુટ:

```
વિદ્યાર્થીઓની સંખ્યા દાખલ કરો: 2

વિદ્યાર્થી 1 ની વિગતો દાખલ કરો:
રોલ નંબર: 101
નામ: John
માર્ક્સ: 80

વિદ્યાર્થી 2 ની વિગતો દાખલ કરો:
રોલ નંબર: 102
નામ: Alice
માર્ક્સ: 70

75 થી વધુ માર્ક્સ વાળા વિદ્યાર્થીઓ:
------------------------------
નામ: John, માર્ક્સ: 80.0

કુલ હાઇ પર્ફોર્મર્સ: 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 અને 1.0 વચ્ચે ફ્લોટ બનાવે છે
num = random.random()
print(num)  # ઉદાહરણ: 0.7234567

# randint() - આપેલ રેન્જ વચ્ચે ઇન્ટીજર બનાવે છે
dice = random.randint(1, 6)
print(dice)  # ઉદાહરણ: 4

# choice() - સિક્વન્સમાંથી રેન્ડમ ઘટક પસંદ કરે છે
colors = ['red', 'blue', 'green']
selected = random.choice(colors)
print(selected)  # ઉદાહરણ: '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:

```
# ફંક્શન વગર (પુનરાવર્તન)
num1 = 5
square1 = num1 * num1
print(square1)

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

# ફંક્શન સાથે (પુનઃઉપયોગ)
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:

```
# વપરાશકર્તા પાસેથી સ્ટ્રિંગ input લો
text = input("સ્ટ્રિંગ દાખલ કરો: ")

# 'a' ની બધી positions શોધો
positions = []
for i in range(len(text)):
    if text[i].lower() == 'a':  # 'a' અને 'A' બંને માટે ચેક કરો
        positions.append(i)

# પરિણામો દર્શાવો
if positions:
    print(f"અક્ષર 'a' આ positions પર મળ્યું: {positions}")
    print("વિગતવાર સ્થાનો:")
    for pos in positions:
        print(f"Position {pos}: '{text[pos]}'")
else:
    print("અક્ષર 'a' સ્ટ્રિંગમાં મળ્યું નથી")

# enumerate વાપરીને વૈકલ્પિક રીત
print("\nવૈકલ્પિક રીત:")
for index, char in enumerate(text):
    if char.lower() == 'a':
        print(f"'a' position {index} પર મળ્યું")
```

સેમ્પલ આઉટપુટ:

```
સ્ટ્રિંગ દાખલ કરો: Python Programming

અક્ષર 'a' આ positions પર મળ્યું: [12]
વિગતવાર સ્થાનો:
Position 12: 'a'

વૈકલ્પિક રીત:
'a' position 12 પર મળ્યું
```

સુધારેલું વર્ઝન:

```
text = input("સ્ટ્રિંગ દાખલ કરો: ")
count = 0

print(f"'{text}' માં 'a' શોધી રહ્યા છીએ")
print("-" * 30)

for i, char in enumerate(text):
    if char.lower() == 'a':
        count += 1
        print(f"'a' ઇન્ડેક્સ {i} પર મળ્યું (અક્ષર: '{char}')")

print(f"\n'a' ની કુલ આવૃત્તિઓ: {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:

```
# ગ્લોબલ વેરિયેબલ
global_var = "હું ગ્લોબલ છું"

def my_function():
    # લોકલ વેરિયેબલ
    local_var = "હું લોકલ છું"
    print(global_var)  # ગ્લોબલ એક્સેસ કરી શકાય છે
    print(local_var)   # લોકલ એક્સેસ કરી શકાય છે

my_function()
print(global_var)      # ગ્લોબલ એક્સેસ કરી શકાય છે
# print(local_var)     # એરર - લોકલ એક્સેસ કરી શકાતું નથી
```

ગ્લોબલ કીવર્ડ:

```
counter = 0  # ગ્લોબલ વેરિયેબલ

def increment():
    global counter  # બદલવા માટે ગ્લોબલ તરીકે declare કરો
    counter += 1

increment()
print(counter)  # આઉટપુટ: 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):
    """વૈકલ્પિક docstring"""
    # ફંક્શન બોડી
    return value  # વૈકલ્પિક
```

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:

```
# ફંક્શન વ્યાખ્યા
def greet_user(name, age):
    """નામ અને ઉંમર સાથે વપરાશકર્તાને શુભેચ્છા આપવાનું ફંક્શન"""
    message = f"હેલો {name}! તમારી ઉંમર {age} વર્ષ છે."
    return message

# ફંક્શન કૉલ
user_name = "જોહન"
user_age = 25
greeting = greet_user(user_name, user_age)
print(greeting)  # આઉટપુટ: હેલો જોહન! તમારી ઉંમર 25 વર્ષ છે.
```

```
# ડિફોલ્ટ પેરામીટર સાથે ફંક્શન
def calculate_power(base, exponent=2):
    return base ** exponent

print(calculate_power(5))      # 25 (ડિફોલ્ટ exponent=2 વાપરીને)
print(calculate_power(5, 3))   # 125 (exponent=3 વાપરીને)
```

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):
    """
    સંખ્યાનું ફેક્ટોરિયલ કેલ્ક્યુલેટ કરવાનું ફંક્શન
    ઇનપુટ: number (integer)
    આઉટપુટ: factorial (integer)
    """
    if number < 0:
        return "નેગેટિવ સંખ્યા માટે ફેક્ટોરિયલ વ્યાખ્યાયિત નથી"
    elif number == 0 or number == 1:
        return 1
    else:
        factorial = 1
        for i in range(2, number + 1):
            factorial *= i
        return factorial

# મુખ્ય પ્રોગ્રામ
try:
    # વપરાશકર્તા પાસેથી ઇનપુટ
    num = int(input("સંખ્યા દાખલ કરો: "))

    # ફંક્શન કૉલ
    result = calcFact(num)

    # પરિણામ દર્શાવો
    if isinstance(result, str):
        print(result)
    else:
        print(f"{num} નું ફેક્ટોરિયલ છે: {result}")

except ValueError:
    print("કૃપા કરીને યોગ્ય ઇન્ટીજર દાખલ કરો")
```

```
# વિવિધ મૂલ્યો સાથે ટેસ્ટ કરો
print("\nવિવિધ મૂલ્યો સાથે ટેસ્ટિંગ:")
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):
    """ફેક્ટોરિયલ કેલ્ક્યુલેટ કરવાનું રીકર્સિવ ફંક્શન"""
    if n < 0:
        return "નેગેટિવ સંખ્યા માટે અવ્યાખ્યાયિત"
    elif n == 0 or n == 1:
        return 1
    else:
        return n * calcFactRecursive(n - 1)

# ઉદાહરણ ઉપયોગ
number = int(input("સંખ્યા દાખલ કરો: "))
result = calcFactRecursive(number)
print(f"ફેક્ટોરિયલ: {result}")
```

સેમ્પલ આઉટપુટ:

```
સંખ્યા દાખલ કરો: 5
5 નું ફેક્ટોરિયલ છે: 120

વિવિધ મૂલ્યો સાથે ટેસ્ટિંગ:
calcFact(0) = 1
calcFact(1) = 1
calcFact(5) = 120
calcFact(10) = 3628800
calcFact(-3) = નેગેટિવ સંખ્યા માટે ફેક્ટોરિયલ વ્યાખ્યાયિત નથી
```

યાદી રાખવાની ટ્રિક: "વ્યાખ્યાયિત થેક 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:

```
# ક્લાસ વ્યાખ્યા (બ્લૂપ્રિન્ટ)
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

# ઓબ્જેક્ટ બનાવટ (ઇન્સ્ટન્સ)
student1 = Student("જોહન", 20)  # ઓબ્જેક્ટ 1
student2 = Student("આલિસ", 19) # ઓબ્જેક્ટ 2

print(student1.name)  # જોહન
print(student2.name)  # આલિસ
```

યાદી રાખવાની ટ્રિક: "ક્લાસ બ્લૂપ્રિન્ટ ઓબ્જેક્ટ ઇન્સ્ટન્સ"

પ્રશ્ન 5(બ) [4 ગુણ]

ક્લાસમાં 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):  # કન્સ્ટ્રક્ટર
        self.length = length  # એટ્રિબ્યુટ ઇનિશિયલાઇઝ કરો
        self.width = width    # એટ્રિબ્યુટ ઇનિશિયલાઇઝ કરો
        print("રેક્ટેંગલ ઓબ્જેક્ટ બન્યું!")

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

# ઓબ્જેક્ટ બનાવટ - કન્સ્ટ્રક્ટર આપોઆપ કૉલ થાય છે
rect1 = Rectangle(10, 5)  # આઉટપુટ: રેક્ટેંગલ ઓબ્જેક્ટ બન્યું!
rect2 = Rectangle()       # ડિફોલ્ટ મૂલ્યો વાપરે છે

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):
        """વિદ્યાર્થી એટ્રિબ્યુટ્સ ઇનિશિયલાઇઝ કરવા માટે કન્સ્ટ્રક્ટર"""
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_info(self):
        """વિદ્યાર્થીની માહિતી દર્શાવવાની મેથડ"""
        print("-" * 30)
        print("વિદ્યાર્થીની માહિતી")
        print("-" * 30)
        print(f"નામ: {self.name}")
        print(f"રોલ નંબર: {self.roll_number}")
        print(f"માર્ક્સ: {self.marks}")
        print("-" * 30)
```

```
    def calculate_grade(self):
        """માર્ક્સ આધારે ગ્રેડ કેલ્ક્યુલેટ કરવાની મેથડ"""
        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):
        """ગ્રેડ દર્શાવવાની મેથડ"""
        grade = self.calculate_grade()
        print(f"ગ્રેડ: {grade}")

# Student ક્લાસના ઓબ્જેક્ટ્સ બનાવવું
print("Student ઓબ્જેક્ટ્સ બનાવી રહ્યા છીએ:")
student1 = Student("જોહન દોય", 101, 85)
student2 = Student("આલિસ સ્મિથ", 102, 92)
student3 = Student("બોબ જોહન્સન", 103, 78)

# માહિતી દર્શાવવા માટે મેથડ્સનો ઉપયોગ
print("\n=== વિદ્યાર્થી 1 ની વિગતો ===")
student1.display_info()
student1.display_grade()

print("\n=== વિદ્યાર્થી 2 ની વિગતો ===")
student2.display_info()
student2.display_grade()

print("\n=== વિદ્યાર્થી 3 ની વિગતો ===")
student3.display_info()
student3.display_grade()

# એટ્રિબ્યુટ્સને સીધી એક્સેસ કરવું
print(f"\nસીધી એક્સેસ - વિદ્યાર્થી 1 નું નામ: {student1.name}")
print(f"સીધી એક્સેસ - વિદ્યાર્થી 2 ના માર્ક્સ: {student2.marks}")
```

સેમ્પલ આઉટપુટ:

```
Student ઓબ્જેક્ટ્સ બનાવી રહ્યા છીએ:

=== વિદ્યાર્થી 1 ની વિગતો ===
------------------------------
વિદ્યાર્થીની માહિતી
------------------------------
નામ: જોહન દોય
```

```
રોલ નંબર: 101
માર્ક્સ: 85
------------------------------
ગ્રેડ: A

=== વિદ્યાર્થી 2 ની વિગતો ===
------------------------------
વિદ્યાર્થીની માહિતી
------------------------------
નામ: આલિસ સ્મિથ
રોલ નંબર: 102
માર્ક્સ: 92
------------------------------
ગ્રેડ: 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  # પ્રાઇવેટ એટ્રિબ્યુટ
```

```
    def get_balance(self):        # Getter મેથડ
        return self.__balance

    def deposit(self, amount):    # નિયંત્રિત એક્સેસ
        if amount > 0:
            self.__balance += amount

account = BankAccount(1000)
print(account.get_balance())      # 1000
# print(account.__balance)        # એરર - સીધી એક્સેસ કરી શકાતી નથી
```

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:

```
    +----------+
    | GrandPa  |  (Base Class)
    +----------+
         ^
         |
    +----------+
    |  Parent  |  (Derived from GrandPa)
    +----------+
         ^
         |
    +----------+
    |  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:

```
# લેવલ 1 - મૂળ ક્લાસ
class Vehicle:
    def __init__(self, brand):
        self.brand = brand

    def start(self):
        print(f"{self.brand} વાહન શરૂ થયું")

# લેવલ 2 - Vehicle માંથી ઇન્હેરિટ
class Car(Vehicle):
    def __init__(self, brand, model):
        super().__init__(brand)
        self.model = model

    def drive(self):
        print(f"{self.brand} {self.model} ચાલી રહી છે")

# લેવલ 3 - Car માંથી ઇન્હેરિટ
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 ઝડપે રેસ કરી રહી છે")

# ઓબ્જેક્ટ બનાવીને મેથડ્સ વાપરવા
ferrari = SportsCar("Ferrari", "F8", 340)
ferrari.start()    # Vehicle ક્લાસમાંથી
ferrari.drive()    # Car ક્લાસમાંથી
ferrari.race()     # SportsCar ક્લાસમાંથી
```

યાદી રાખવાની ટ્રિક: "શૃંખલા ઇન્હેરિટ લેવલ એક્સેસ"

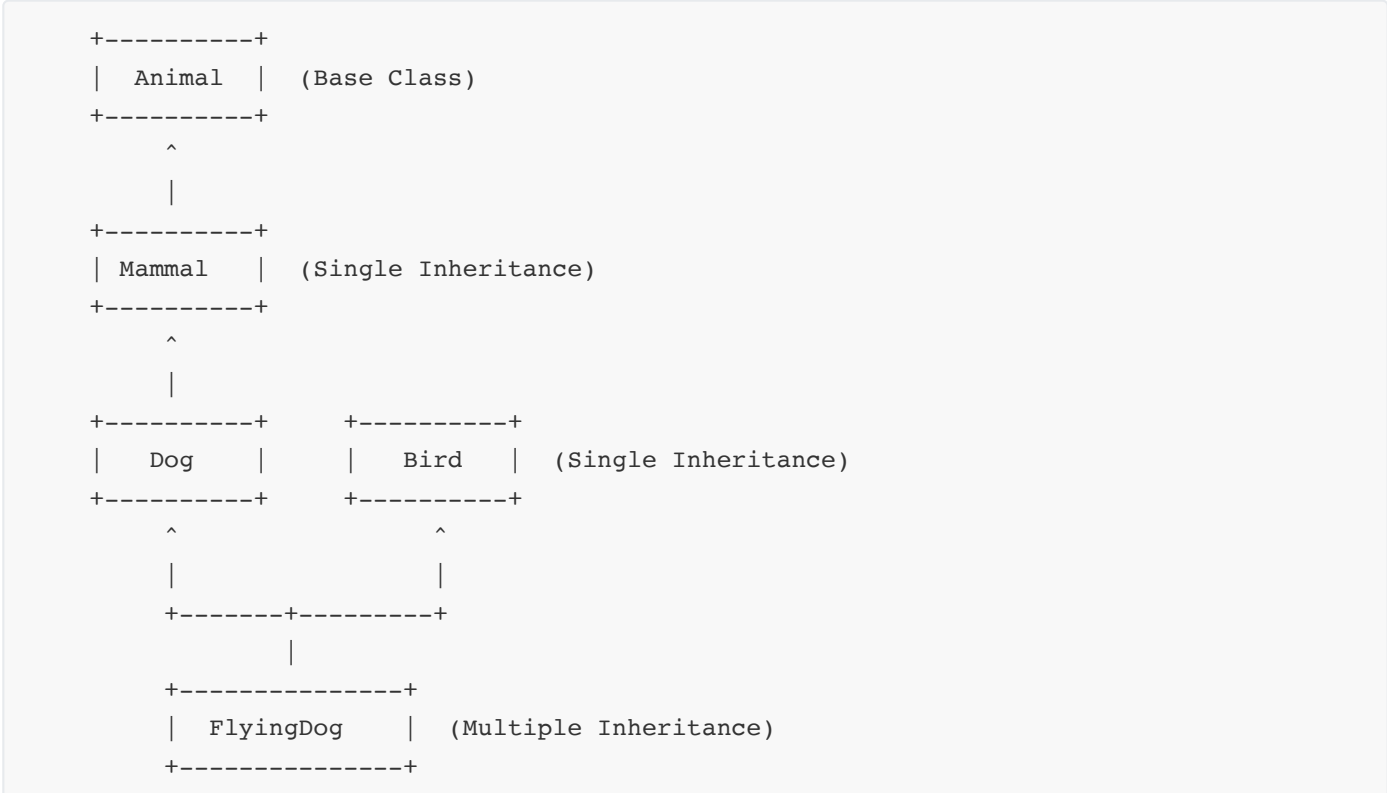
પ્રશ્ન 5(ક OR) [7 ગુણ]

હાઇબ્રિડ ઇન્હેરિટન્સનું કાર્ય દર્શાવતો પાયથોન પ્રોગ્રામ લખો.

જવાબ:

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

સ્ટ્રક્ચર ડાયાગ્રામ:



કોડ ઉદાહરણ:

```
# મૂળ ક્લાસ
class Animal:
    def __init__(self, name):
        self.name = name
        print(f"પ્રાણી {self.name} બન્યું")

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

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

# Animal માંથી સિંગલ ઇન્હેરિટન્સ
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} જીવતા બાળકોને જન્મ આપે છે")

# Animal માંથી સિંગલ ઇન્હેરિટન્સ
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 પાંખો સાથે ઉડે છે")

    def lay_eggs(self):
        print(f"{self.name} ઇંડા આપે છે")

# Mammal માંથી સિંગલ ઇન્હેરિટન્સ
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} ભૂકે છે")

    def guard(self):
        print(f"{self.name} ઘરની રક્ષા કરે છે")

# Dog અને Bird માંથી મલ્ટિપલ ઇન્હેરિટન્સ (હાઇબ્રિડ)
class FlyingDog(Dog, Bird):
    def __init__(self, name, fur_color, breed, wing_span):
        # બંને પેરેન્ટ ક્લાસને ઇનિશિયલાઇઝ કરો
        Dog.__init__(self, name, fur_color, breed)
        Bird.__init__(self, name, wing_span)
        print(f"જાદુઈ {self.name} બન્યું જેમાં mammal અને bird બંનેના લક્ષણ છે!")

    def fly_and_bark(self):
        print(f"{self.name} એક સાથે ઉડે છે અને ભૂકે છે!")

    def show_abilities(self):
        print(f"\n{self.name} ની ક્ષમતાઓ:")
        print("-" * 25)
        self.eat()           # Animal માંથી
        self.sleep()         # Animal માંથી
        self.give_birth()    # Mammal માંથી
        self.bark()          # Dog માંથી
        self.guard()         # Dog માંથી
        self.fly()           # Bird માંથી
        self.lay_eggs()      # Bird માંથી
        self.fly_and_bark() # પોતાની મેથડ

# પ્રદર્શન
print("=== હાઇબ્રિડ ઇન્હેરિટન્સ ડેમો ===\n")

# ઓબ્જેક્ટ્સ બનાવો
print("1. સામાન્ય કૂતરો બનાવી રહ્યા છીએ:")
dog1 = Dog("બડી", "સુવર્ણ", "રિટ્રીવર")
dog1.bark()
dog1.guard()
```

```
print("\n2. સામાન્ય પક્ષી બનાવી રહ્યા છીએ:")
bird1 = Bird("ગરુડ", 200)
bird1.fly()
bird1.lay_eggs()

print("\n3. જાદુઈ ઉડતો કૂતરો બનાવી રહ્યા છીએ:")
flying_dog = FlyingDog("સુપરડોગ", "રજતી", "હસ્કી", 150)
flying_dog.show_abilities()

# મેથડ રિઝોલ્યુશન ઓર્ડર
print(f"\nFlyingDog માટે મેથડ રિઝોલ્યુશન ઓર્ડર:")
for i, cls in enumerate(FlyingDog.__mro__):
    print(f"{i+1}. {cls.__name__}")
```

સેમ્પલ આઉટપુટ:

```
=== હાઇબ્રિડ ઇન્હેરિટન્સ ડેમો ===

1. સામાન્ય કૂતરો બનાવી રહ્યા છીએ:
પ્રાણી બડી બન્યું
બડી રિટ્રીવર કૂકે છે
બડી ઘરની રક્ષા કરે છે

2. સામાન્ય પક્ષી બનાવી રહ્યા છીએ:
પ્રાણી ગરુડ બન્યું
ગરુડ 200cm પાંખો સાથે ઉડે છે
ગરુડ ઇંડા આપે છે

3. જાદુઈ ઉડતો કૂતરો બનાવી રહ્યા છીએ:
પ્રાણી સુપરડોગ બન્યું
પ્રાણી સુપરડોગ બન્યું
જાદુઈ સુપરડોગ બન્યું જેમાં mammal અને bird બંનેના લક્ષણ છે!

સુપરડોગ ની ક્ષમતાઓ:
--------------------------
સુપરડોગ ખાય છે
સુપરડોગ સૂએ છે
સુપરડોગ જીવતા બાળકોને જન્મ આપે છે
સુપરડોગ હસ્કી કૂકે છે
સુપરડોગ ઘરની રક્ષા કરે છે
સુપરડોગ 150cm પાંખો સાથે ઉડે છે
સુપરડોગ ઇંડા આપે છે
સુપરડોગ એક સાથે ઉડે છે અને કૂકે છે!
```

આ ઉદાહરણમાં ઇન્હેરિટન્સના પ્રકારો:

1. **સિંગલ**: Mammal ← Animal, Bird ← Animal, Dog ← Mammal
2. **મલ્ટિપલ**: FlyingDog ← Dog + Bird
3. **મલ્ટિલેવલ**: FlyingDog ← Dog ← Mammal ← Animal

4. **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"