Subject Name (Gujarati)

4351108 -- Summer 2024

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 ગુણ]

Python માં for loop નું કાર્ય સમજાવો.

જવાબ

For loop એ list, tuple અથવા string જેવા sequence ના દરેક item માટે code block ને repeat કરે છે.
સિન્ટેક્સ ટેબલ:

ઘટક	Syntax	ઉદાહરણ
મૂળભૂત	`for variable in sequence:`	`for i in [1,2,3]:`
Range	`for i in range(n):`	`for i in range(5):`
String	`for char in string:`	`for c in "hello":`

આકૃતિ:

```
Start {-{-} Check if items left in sequence}
         |
         v
    Execute loop body
         |
         v
    Move to next item {-{-} Check if items left}
         |                        |
         v                        v
    Items left? {-{-}{-}{-}No{-}{-}{-}{-} End}
         |
        Yes
         |
         v
    Back to Execute loop body
```

- **પુનરાવર્તન**: Loop variable ને sequence માંથી દરેક value એક પછી એક મળે છે
- **આપમેળે**: Python આપમેળે next item પર જવાનું handle કરે છે
- **લવચીક**: Lists, strings, tuples, ranges સાથે કામ કરે છે

યાદી કૌશલ્ય: ``દરેક Item માટે, Block Execute કરો''

પ્રશ્ન 1(બ) [4 ગુણ]

Python માં if-elif-else નું કાર્ય સમજાવો.

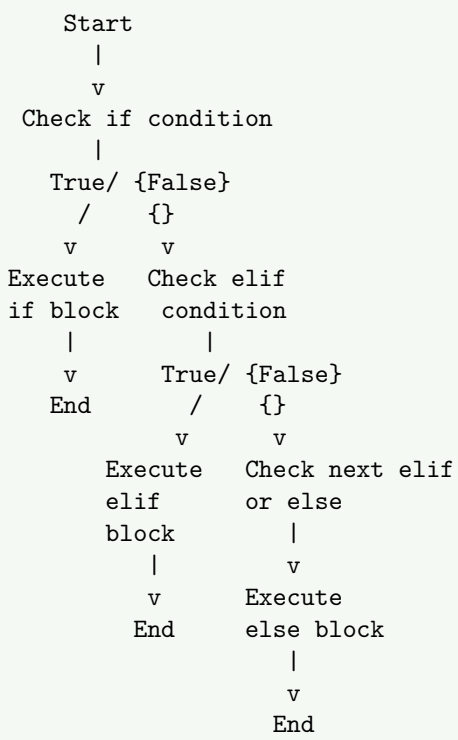
જવાબ

બહુ-માર્ગીય નિર્ણય માળખું જે sequence માં અનેક conditions ને ચકાસે છે.
માળખાકીય ટેબલ:

Statement	હેતુ	Syntax
if	પ્રથમ શરત	`if condition1:`

elif	વૈકલ્પિક શરતો	`elif condition2:`
else	મૂળભૂત કેસ	`else:`

પ્રવાહ આકૃતિ:



- **ક્રમબદ્ધ**: ઉપરથી નીચે conditions ને ચકાસે છે
- **વિશિષ્ટ**: માત્ર એક જ block execute થાય છે
- **વૈકલ્પિક**: elif અને else વૈકલ્પિક છે

યાદી કૌશલ્ય: ``જો આ, અથવા જો તે, અથવા Default''

પ્રશ્ન 1(ક) [7 ગુણ]

Python પ્રોગ્રામનું માળખું સમજાવો.

જવાબ

Python પ્રોગ્રામમાં તાર્કિક ક્રમમાં વિશિષ્ટ ઘટકો સાથે વ્યવસ્થિત માળખું હોય છે.

પ્રોગ્રામ માળખું ટેબલ:

ઘટક	હેતુ	ઉદાહરણ
Comments	દસ્તાવેજીકરણ	`# This is comment`
Import	બાહ્ય modules	`import math`
Constants	નિશ્ચિત વેલ્યુઝ	`PI = 3.14`
Functions	પુનઃઉપયોગી કોડ	`def function_name():`
Classes	Objects નો blueprint	`class ClassName:`
Main code	પ્રોગ્રામ execution	`if __name__ == "__main__":`

પ્રોગ્રામ આર્કિટેક્ચર:

```
  Comments
\# Documentation

      v

 Import Section
import modules

      v

 Constants \&
  Variables

      v

Function
Definitions

      v

Class
Definitions

      v

Main Program
  Execution
```

- **મોડ્યુલર**: દરેક વિભાગનો વિશિષ્ટ હેતુ હોય છે
- **વાંચવા યોગ્ય**: સ્પષ્ટ સંગઠન સમજવામાં મદદ કરે છે
- **જાળવણી યોગ્ય**: ફેરફાર અને debug કરવું સરળ
- **માનક**: Python conventions ને અનુસરે છે

સરળ ઉદાહરણ:

```
\# Program to calculate area
import math

PI = 3.14159

def calculate\_area(radius):
    return PI * radius * radius

\# Main execution
radius = float(input("Enter radius: "))
area = calculate\_area(radius)
print(f"Area = \{area\}")
```

યાદી કૌશલ્ય: ``Comment, Import, Constant, Function, Class, Main''

પ્રશ્ન 1(ક OR) [7 ગુણ]

Python પ્રોગ્રામિંગ લેંગવેજની વિશેષતાઓ સમજાવો.

જવાબ

Python ની અનન્ય લાક્ષણિકતાઓ છે જે તેને beginners અને professionals માટે લોકપ્રિય બનાવે છે.
Python વિશેષતાઓ ટેબલ:

વિશેષતા	વર્ણન	લાભ
સરળ	સરળ syntax	ઝડપી શીખવા
Interpreted	કોઈ compilation નહીં	ઝડપી development
Object-Oriented	Classes અને objects	કોડની પુનઃઉપયોગીતા
Open Source	ઉપયોગ માટે મફત	કોઈ licensing ખર્ચ નહીં
Cross-Platform	દરેક જગ્યાએ run થાય છે	ઉચ્ચ portability

વિશેષતા કેટેગરીઝ:

```
          Python Features



    v         v         v
Language   Technical  Community
Features   Features   Features

    v         v         v
{- Simple   {-} Interpreted {-} Open Source}
{- Readable {-} Portable    {-} Large Library}
{- Dynamic  {-} Extensible  {-} Active Support}
```

- **શિખાઉ-મિત્ર**: અંગ્રેજી ભાષા જેવું સરળ syntax
- **બહુમુખી**: web, AI, data science, automation માટે ઉપયોગ
- **સમૃદ્ધ લાયબ્રેરીઝ**: પ્રી-બિલ્ટ modules નો વિશાળ સંગ્રહ
- **ડાયનેમિક ટાઇપિંગ**: variable types declare કરવાની જરૂર નથી
- **ઇન્ટરેક્ટિવ**: interpreter માં લાઇન બાય લાઇન કોડ ટેસ્ટ કરી શકાય
- **હાઇ-લેવલ**: memory management આપમેળે handle કરે છે

કોડ ઉદાહરણ:

```
\# Simple Python syntax
name = "Python"
print(f"Hello, \{name\}!")
```

યાદી કૌશલ્ય: ``સરળ, Interpreted, Object-Oriented, Open, Cross-platform''

પ્રશ્ન 2(અ) [3 ગુણ]

સ્ટ્રિંગ પર થતાં કોઈ 3 ઓપરેશન સમજાવો.

જવાબ

String operations વિવિધ રીતે text data ને manipulate અને process કરે છે.
સ્ટ્રિંગ ઓપરેશન્સ ટેબલ:

ઓપરેશન	Method	ઉદાહરણ	પરિણામ
જોડવું	`+`	`"Hello" + "World"`	`"HelloWorld"`
લંબાઈ	`len()`	`len("Python")`	`6`
મોટા અક્ષર	`.upper()`	`"hello".upper()`	`"HELLO"`

ઓપરેશન ઉદાહરણો:

```
text = "Python"
\# 1.
result1 = text + " Programming"
\# 2.
result2 = len(text)
\# 3.
result3 = text.upper()
```

- **જોડવું**: બે અથવા વધુ strings ને સાથે જોડે છે
- **લંબાઈ**: string માં કુલ characters ગણે છે
- **કેસ કન્વર્ઝન**: અક્ષરોના કેસ બદલે છે (upper/lower)

યાદી કૌશલ્ય: ``જોડો, ગણો, કન્વર્ટ કરો''

પ્રશ્ન 2(બ) [4 ગુણ]

તાપમાનને ફેરનહાઇટથી સેલ્સિયસ એકમમાં (C=(F-32)/1.8 સમીકરણથી) પરિવર્તિત કરવા માટેનો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ user input સાથે ગાણિતિક formula વાપરીને temperature convert કરે છે.
એલ્ગોરિધમ ટેબલ:

પગલું	ક્રિયા	કોડ
1	Input લો	`fahrenheit = float(input())`
2	Formula લાગુ કરો	`celsius = (fahrenheit - 32) / 1.8`
3	પરિણામ દર્શાવો	`print(f"Celsius: {celsius}")`

સંપૂર્ણ પ્રોગ્રામ:

```
\# Temperature conversion program
fahrenheit = float(input("Enter temperature in Fahrenheit: "))
celsius = (fahrenheit {-} 32) / 1.8
print(f"Temperature in Celsius: \{celsius:.2f\}")
```

ટેસ્ટ કેસેસ:

- Input: $32 \rightarrow Output: 0.00$
- Input: $100 \rightarrow Output: 37.78$
- **યુઝર ઇનપુટ**: યુઝર પાસેથી Fahrenheit temperature લે છે
- **ફોર્મ્યુલા એપ્લિકેશન**: આપેલ conversion equation વાપરે છે
- **ફોર્મેટેડ આઉટપુટ**: decimal places સાથે પરિણામ બતાવે છે

યાદી કૌશલ્ય: ``Input, Calculate, Output''

પ્રશ્ન 2(ક) [7 ગુણ]

Python માં list ડેટા ટાઇપ વિસ્તૃત રીતે સમજાવો.

જવાબ

List એ ordered, mutable collection છે જે single variable માં multiple items store કરે છે.
લિસ્ટ લાક્ષણિકતાઓ ટેબલ:

પ્રોપર્ટી	વર્ણન	ઉદાહરણ
ક્રમબદ્ધ	Items નો position હોય છે	`[1, 2, 3]`

પરિવર્તનશીલ	બદલાઈ શકાય છે	`list[0] = 10`
ઇન્ડેક્સ્ડ	Position દ્વારા access	`list[0]`
મિશ્ર પ્રકારો	વિવિધ data types	`[1, "hello", 3.14]`

લિસ્ટ ઓપરેશન્સ આકૃતિ:

```
List: [10, 20, 30, 40]
       |   |   |   |
Index: 0   1   2   3

Operations:

    Access          Modify
   list[0]        list[0]=50


      v               v
    "10"          [50, 20, 30, 40]
```

સામાન્ય લિસ્ટ મેથડ્સ:

Method	હેતુ	ઉદાહરણ
append()	અંતે item ઉમેરો	`list.append(5)`
insert()	position પર ઉમેરો	`list.insert(1, 15)`
remove()	item ડિલીટ કરો	`list.remove(20)`
pop()	છેલ્લું item દૂર કરો	`list.pop()`
len()	લંબાઈ મેળવો	`len(list)`

ઉદાહરણ કોડ:

```
\# Creating and using lists
numbers = [1, 2, 3, 4, 5]
numbers.append(6)        \#   6
numbers.insert(0, 0)     \#     0
print(numbers[2])        \# 3  element access
numbers.remove(3)        \# value 3
```

- **ડાયનેમિક સાઇઝ**: execution દરમ્યાન વધી અથવા ઘટી શકે છે
- **ઝીરો ઇન્ડેક્સિંગ**: પ્રથમ element index 0 પર
- **સ્લાઇસિંગ**: [start:end] વાપરીને ભાગો extract કરી શકાય
- **નેસ્ટેડ લિસ્ટ્સ**: અન્ય lists સમાવી શકે છે

યાદી કૌશલ્ય: ``ક્રમબદ્ધ, પરિવર્તનશીલ, ઇન્ડેક્સ્ડ, મિશ્ર''

પ્રશ્ન 2(અ OR) [3 ગુણ]

Python માં સ્ટ્રિંગ ફોર્મેટિંગ સમજાવો.

જવાબ

String formatting એ templates માં values insert કરીને formatted strings બનાવે છે.
ફોર્મેટિંગ મેથડ્સ ટેબલ:

Method	Syntax	ઉદાહરણ
f-strings	`f"text {variable}"`	`f"Hello {name}"`
format()	`"text {}".format(value)`	`"Age: {}".format(25)`
% operator	`"text %s" % value`	`"Name: %s" % "John"`

ઉપયોગ ઉદાહરણ:

```
name = "Alice"
age = 25
\# f{-string formatting}
message = f"Hello \{name\}, you are \{age\} years old"
```

- **પ્લેસહોલ્ડર**: {} માર્ક કરે છે કે values ક્યાં જાય છે
- **ડાયનેમિક**: Runtime પર values insert થાય છે
- **વાંચવા યોગ્ય**: Concatenation કરતાં કોડ સાફ બનાવે છે

યાદી કૌશલ્ય: ``Format, Insert, Display''

પ્રશ્ન 2(બ OR) [4 ગુણ]

સ્કેન કરેલ નંબર એકી સંખ્યા છે કે બેકી સંખ્યા છે તે ઓળખી અને યોગ્ય મેસેજ પ્રિન્ટ કરતો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ number 2 થી divisible છે કે નહીં તે ચકાસીને even અથવા odd નક્કી કરે છે.
લૉજિક ટેબલ:

શરત	પરિણામ	મેસેજ
number % 2 == 0	Even	``Number is even''
number % 2 != 0	Odd	``Number is odd''

સંપૂર્ણ પ્રોગ્રામ:

```
\# Even/Odd checker program
number = int(input("Enter a number: "))
if number \% 2 == 0:
    print(f"\{number\} is even")
else:
    print(f"\{number\} is odd")
```

ટેસ્ટ કેસેસ:

- Input: 4 $\rightarrow$ $Output : "4 is even"$
- Input: 7 $\rightarrow$ $Output : "7 is odd"$
- **મોડ્યુલો ઓપરેટર**: % division પછીનો remainder આપે છે
- **કંડિશનલ લૉજિક**: if-else પરિણામ નક્કી કરે છે
- **યુઝર ફીડબેક**: પરિણામ વિશે સ્પષ્ટ મેસેજ

યાદી કૌશલ્ય: ``Input, Check Remainder, Display Result''

પ્રશ્ન 2(ક OR) [7 ગુણ]

Python માં Set ડેટા ટાઇપ વિસ્તૃત રીતે સમજાવો.

જવાબ

Set એ unordered collection છે જેમાં unique items હોય છે અને duplicate values નહીં.
સેટ લાક્ષણિકતાઓ ટેબલ:

પ્રોપર્ટી	વર્ણન	ઉદાહરણ
અક્રમ	કોઈ નિશ્ચિત position નથી	`{1, 3, 2}`
અનન્ય	કોઈ duplicates નથી	`{1, 2, 3}`
પરિવર્તનશીલ	ફેરફાર કરી શકાય	`set.add(4)`

પુનરાવર્તન યોગ્ય Loop કરી શકાય `for item in set:`

સેટ ઓપરેશન્સ આકૃતિ:

```
    Set A: \{1, 2, 3\    Set B: \{3, 4, 5\}}
         {                    /}
          {                  /}
           v                v

         Set Operations

      Union: \{1, 2, 3, 4, 5\  b    }
      Intersection: \{3\            }
      Difference: \{1, 2\           }
      Symmetric Diff: \{1,2,4,5\    }
```

સેટ મેથડ્સ ટેબલ:

Method	હેતુ	ઉદાહરણ
add()	single item ઉમેરો	`set.add(6)`
update()	multiple items ઉમેરો	`set.update([7, 8])`
remove()	item ડિલીટ કરો	`set.remove(3)`
union()	sets જોડો	`set1.union(set2)`
intersection()	સામાન્ય items	`set1.intersection(set2)`

ઉદાહરણ કોડ:

```
\# Creating and using sets
fruits = \{"apple", "banana", "orange"\}
fruits.add("mango")              \# single item
fruits.update(["grape", "kiwi"]) \# multiple
fruits.remove("banana")          \# item
print(len(fruits))               \# items
```

- **આપમેળે ડુપ્લિકેશન હટાવવું**: Duplicate values આપમેળે દૂર કરે છે
- **ઝડપી મેમ્બરશિપ**: Item exists છે કે નહીં ઝડપથી ચકાસી શકાય
- **ગાણિતિક ઓપરેશન્સ**: Union, intersection, difference
- **કોઈ ઇન્ડેક્સિંગ નહીં**: Position દ્વારા items access કરી શકાતાં નથી

યાદી કૌશલ્ય: ``અનન્ય, અક્રમ, પરિવર્તનશીલ, ગાણિતિક''

પ્રશ્ન 3(અ) [3 ગુણ]

math મૉડ્યુલની કોઈ પણ 3 મેથડ સમજાવો.

જવાબ

Math module જટિલ ગણતરીઓ માટે ગાણિતિક functions પ્રદાન કરે છે.
મેથ મેથડ્સ ટેબલ:

Method	હેતુ	ઉદાહરણ	પરિણામ
math.sqrt()	વર્ગમૂળ	`math.sqrt(16)`	`4.0`
math.pow()	પાવર ગણતરી	`math.pow(2, 3)`	`8.0`
math.ceil()	ઉપર રાઉન્ડ	`math.ceil(4.3)`	`5`

ઉપયોગ ઉદાહરણ:

```
import math
number = 16
result1 = math.sqrt(number)  \#
result2 = math.pow(2, 4)     \# 2       4
result3 = math.ceil(7.2)     \# 8
```

- **ચોકસાઈ**: Basic operators કરતાં વધુ accurate
- **ઇમ્પોર્ટ જરૂરી**: પહેલા math module import કરવું પડે
- **રિટર્ન વેલ્યુઝ**: સામાન્ય રીતે float numbers return કરે છે

યાદી કૌશલ્ય: ``વર્ગમૂળ, પાવર, સીલિંગ''

પ્રશ્ન 3(બ) [4 ગુણ]

for loop નો ઉપયોગ કરીને લિસ્ટમાં આવેલ તમામ ઘટકોનો સરવાળો શોધવા માટેનો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ list દ્વારા iterate કરે છે અને બધા elements નો sum accumulate કરે છે.
એલ્ગોરિધમ ટેબલ:

પગલું	ક્રિયા	કોડ
1	Sum initialize કરો	`total = 0`
2	List માં loop કરો	`for element in list:`
3	Sum માં ઉમેરો	`total += element`
4	પરિણામ દર્શાવો	`print(total)`

સંપૂર્ણ પ્રોગ્રામ:

```
\# Sum of list elements
numbers = [10, 20, 30, 40, 50]
total = 0
for element in numbers:
    total += element
print(f"Sum of all elements: \{total\}")
```

ટેસ્ટ કેસ:

- Input: [1, 2, 3, 4, 5] $\rightarrow Output : 15$
- **એક્યુમ્યુલેટર**: Variable running total store કરે છે
- **પુનરાવર્તન**: Loop દરેક element ની એકવાર મુલાકાત લે છે
- **ઉમેરો**: દરેક element ને running sum માં ઉમેરે છે

યાદી કૌશલ્ય: ``Initialize, Loop, Add, Display''

પ્રશ્ન 3(ક) [7 ગુણ]

બે list ની લંબાઈ સમાન છે કે નહીં તે ચકાસવા, અને જો હોય તો તેમને ભેગા કરીને તેમાંથી એક dictionary બનાવવાનો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ list lengths ની સરખામણી કરે છે અને જો તે match કરે તો dictionary બનાવે છે.
લૉજિક ફ્લો ટેબલ:

પગલું	શરત	ક્રિયા
1	લંબાઈ ચકાસો	`len(list1) == len(list2)`

2	જો સમાન	Merge અને dictionary બનાવો
3	જો અસમાન	Error message દર્શાવો

પ્રક્રિયા આકૃતિ:

```
    List1: [a, b, c]     List2: [1, 2, 3]
       |                      |
       v                      v
     len(List1) == len(List2) ?
             |
        Yes / { No}
           /   {}
          v     v
    Create Dict  Error
    \{a:1, b:2,   Message
     c:3\}
```

સંપૂર્ણ પ્રોગ્રામ:

```
\# Merge lists into dictionary
list1 = [{name}, {age}, {city}]
list2 = [{John}, 25, {Mumbai}]

if len(list1) == len(list2):
    \# Create dictionary using zip
    result\_dict = dict(zip(list1, list2))
    print("Dictionary created:", result\_dict)
else:
    print("Lists have different lengths, cannot merge")
```

અપેક્ષિત આઉટપુટ:

```
Dictionary created: {'name': 'John', 'age': 25, 'city': 'Mumbai'}
```

- **લંબાઈ સરખામણી**: લિસ્ટ્સ યોગ્ય રીતે pair કરી શકાય છે તેની ખાતરી કરે છે
- **zip() ફંક્શન**: બંને lists ના elements ને pair કરે છે
- **dict() કન્સ્ટ્રક્ટર**: Paired elements માંથી dictionary બનાવે છે
- **એરર હેન્ડલિંગ**: ખોટા pairing ને અટકાવે છે

વૈકલ્પિક મેથડ:

```
\# Manual dictionary creation
result\_dict = \{\}
for i in range(len(list1)):
    result\_dict[list1[i]] = list2[i]
```

યાદી કૌશલ્ય: ``લંબાઈ ચકાસો, Zip કરો, Dictionary બનાવો''

પ્રશ્ન 3(અ OR) [3 ગુણ]

statistics મૉડ્યુલની કોઈ પણ 3 મેથડ સમજાવો.

જવાબ

Statistics module numeric data પર statistical calculations માટે functions પ્રદાન કરે છે.
સ્ટેટિસ્ટિક્સ મેથડ્સ ટેબલ:

Method	હેતુ	ઉદાહરણ	પરિણામ
statistics.mean()	સરેરાશ value	`mean([1,2,3,4,5])`	`3.0`
statistics.median()	મધ્ય value	`median([1,2,3,4,5])`	`3`

statistics.mode()	સૌથી વધુ વારંવાર	`mode([1,1,2,3])`	`1`

ઉપયોગ ઉદાહરણ:

```
import statistics
data = [10, 20, 30, 40, 50]
avg = statistics.mean(data)      \#
mid = statistics.median(data)    \#    value
```

- **ડેટા એનાલિસિસ**: ડેટા patterns સમજવામાં મદદ કરે છે
- **બિલ્ટ-ઇન ફંક્શન્સ**: જટિલ formulas લખવાની જરૂર નથી
- **ચોક્કસ પરિણામો**: Edge cases ને યોગ્ય રીતે handle કરે છે

યાદી કૌશલ્ય: ``Mean, Median, Mode''

પ્રશ્ન 3(ક OR) [7 ગુણ]

આપેલ સ્ટ્રિંગમાં કોઈ અક્ષર કેટલી વાર આવે છે તે ગણવા માટેની dictionary બનાવવાનો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ dictionary બનાવે છે જ્યાં keys અક્ષરો છે અને values તેમની counts છે.
અક્ષર ગણતરી એલ્ગોરિધમ:

પગલું	ક્રિયા	કોડ
1	Dictionary initialize કરો	`char_count = {}`
2	String માં loop કરો	`for char in string:`
3	Occurrences ગણો	`char_count[char] = char_count.get(char, 0) + 1`
4	પરિણામો દર્શાવો	`print(char_count)`

ગણતરી પ્રક્રિયા:

```
String: "hello"
     |
     v
Loop through each character
     |

     h     e    l    l    o

     v
Dictionary: \{{h:1, e:1, l:2, o:1\}}
```

સંપૂર્ણ પ્રોગ્રામ:

```
\# Character frequency counter
text = input("Enter a string: ")
char\_count = \{\}

for char in text:
    if char in char\_count:
        char\_count[char] += 1
    else:
        char\_count[char] = 1

print("Character frequencies:")
for char, count in char\_count.items():
    print(f"{}\{char\}{: }\{count\}")
```

વૈકલ્પિક મેથડ (વધુ Pythonic):

```
\# Using get() method
text = "programming"
char\_count = \{\}

for char in text:
    char\_count[char] = char\_count.get(char, 0) + 1

print(char\_count)
```

ઉદાહરણ આઉટપુટ:

```
Input: "hello"
Output: {'h': 1, 'e': 1, 'l': 2, 'o': 1}
```

- **ડિક્શનરી કીઝ**: દરેક અનન્ય અક્ષર key બને છે
- **ડિક્શનરી વેલ્યુઝ**: અક્ષરની occurrences ની count
- **get() મેથડ**: Key exist નથી તો 0 return કરે છે, errors ટાળે છે
- **પુનરાવર્તન**: String ના દરેક અક્ષરને એકવાર process કરે છે

યાદી કૌશલ્ય: ``Loop, Check, Count, Store''

પ્રશ્ન 4(અ) [3 ગુણ]

Python ક્લાસ અને ઓબ્જેક્ટ્સનું કાર્ય ઉદાહરણ સાથે સમજાવો.

જવાબ

Class એ objects બનાવવા માટેનો blueprint છે. Objects એ classes ના instances છે.
ક્લાસ-ઓબ્જેક્ટ સંબંધ:

કન્સેપ્ટ	હેતુ	ઉદાહરણ
Class	Template/Blueprint	`class Car:`
Object	Class નો instance	`my_car = Car()`
Attributes	Class માં ડેટા	`self.color = "red"`
Methods	Class માં functions	`def start(self):`

ક્લાસ માળખું:

```
          Class: Car

        Attributes:
        {- color            }
        {- model            }

        Methods:
        {- start()          }
        {- stop()           }


                inherits
              v
    Object: my\_car = Car()
```

ઉદાહરણ કોડ:

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

    def display(self):    \# Method
        print(f"Name: \{self.name\}, Age: \{self.age\}")

\# Creating objects
student1 = Student("Alice", 20)
student1.display()
```

- **એન્કેપ્સુલેશન**: સંબંધિત data અને functions ને સાથે group કરે છે
- **પુનઃઉપયોગીતા**: એક class અનેક objects બનાવી શકે છે
- **સંગઠન**: બહેતર code structure અને maintenance

યાદી કૌશલ્ય: ``ક્લાસ Blueprint, ઓબ્જેક્ટ Instance''

પ્રશ્ન 4(બ) [4 ગુણ]

લિસ્ટમાં આવેલી તમામ એકી સંખ્યાઓ પ્રિન્ટ કરવા માટેનો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ list elements ને filter કરે છે અને માત્ર odd numbers દર્શાવે છે.
એકી સંખ્યા ચકાસણી ટેબલ:

સંખ્યા	number % 2	પરિણામ
1	1	એકી
2	0	બેકી
3	1	એકી
4	0	બેકી

સંપૂર્ણ પ્રોગ્રામ:

```
\# Print odd numbers from list
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

print("Odd numbers in the list:")
for number in numbers:
    if number \% 2 != 0:
        print(number, end=" ")
```

વૈકલ્પિક મેથડ્સ:

```
\# Method 2: List comprehension
odd\_numbers = [num for num in numbers if num \% 2 != 0]
print(odd\_numbers)

\# Method 3: Using filter
odd\_numbers = list(filter(lambda x: x \% 2 != 0, numbers))
print(odd\_numbers)
```

અપેક્ષિત આઉટપુટ:

```
Odd numbers in the list:
1 3 5 7 9
```

- **મોડ્યુલો ઓપરેશન**: % operator remainder શોધે છે
- **કંડિશન ચેક**: જો remainder 0 નથી તો number odd છે
- **લૂપ પુનરાવર્તન**: List ની દરેક number ચકાસે છે

યાદી કૌશલ્ય: ``Loop, Check Remainder, Print Odd''

પ્રશ્ન 4(ક) [7 ગુણ]

Python માં યુઝર ડિફાઇન્ડ ફંકશન્સનું કાર્ય સમજાવો.

જવાબ

User-defined functions એ programmers દ્વારા બનાવેલા custom functions છે જે વિશિષ્ટ કાર્યો કરે છે.

ફંકશન ઘટકો ટેબલ:

ઘટક	હેતુ	Syntax
def કીવર્ડ	Function declaration	`def function_name():`
Parameters	Input values	`def func(param1, param2):`
Body	Function code	Indented statements
return	Output value	`return value`

ફંક્શન માળખું:

```
    def function\_name(parameters):

                            Input values
                    Function identifier
           Keyword to define function

    Function Body (indented)

         v

       Local variables
       Processing logic
       Calculations


         v
    return result (optional)
```

ફંક્શન પ્રકારો:

પ્રકાર	વર્ણન	ઉદાહરણ
કોઈ parameters નહીં	કોઈ input લેતું નથી	`def greet():`
Parameters સાથે	Input લે છે	`def add(a, b):`
Return value	Output આપે છે	`return a + b`
કોઈ return નહીં	Action કરે છે	`print("Hello")`

ઉદાહરણ ફંક્શન્સ:

```
\# Function with no parameters
def greet():
    print("Hello, World!")

\# Function with parameters and return value
def calculate\_area(length, width):
    area = length * width
    return area

\# Function with default parameters
def introduce(name, age=18):
    print(f"My name is \{name\} and I am \{age\} years old")

\# Using functions
greet()
result = calculate\_area(5, 3)
print(f"Area: \{result\}")
introduce("Alice", 25)
introduce("Bob")  \# Uses default age
```

ફંક્શન લાભો:

- **પુનઃઉપયોગીતા**: એકવાર લખો, અનેકવાર વાપરો
- **મોડ્યુલરિટી**: જટિલ સમસ્યાઓને નાના ભાગોમાં તોડો
- **જાળવણીયોગ્યતા**: Update અને debug કરવું સરળ
- **વાંચવાયોગ્યતા**: કોડને વધુ organized અને સમજવાયોગ્ય બનાવે છે
- **ટેસ્ટિંગ**: અલગ અલગ functions ને અલગથી test કરી શકાય

વેરિયેબલ સ્કોપ:

- **લોકલ વેરિયેબલ્સ**: માત્ર function અંદર exist કરે છે
- **ગ્લોબલ વેરિયેબલ્સ**: આખા program માં accessible
- **પેરામીટર્સ**: Local variables તરીકે કામ કરે છે

યાદી કૌશલ્ય: ``Define, Parameters, Body, Return''

પ્રશ્ન 4(અ OR) [3 ગુણ]

Python માં કન્સ્ટ્રક્ટરનું કાર્ય સમજાવો.

જવાબ

Constructor એ special method છે જે objects બનાવવામાં આવે ત્યારે તેમને initialize કરે છે.
કન્સ્ટ્રક્ટર વિગતો ટેબલ:

પાસું	વર્ણન	Syntax
Method name	હંમેશા `__init__`	`def __init__(self):`
હેતુ	Object initialize કરવું	Initial values set કરવા
આપમેળે કૉલ	Object creation દરમ્યાન કૉલ થાય	`obj = Class()`
Parameters	Arguments લઈ શકે છે	`def __init__(self, param):`

કન્સ્ટ્રક્ટર ઉદાહરણ:

```
class Student:
    def \_\_init\_\_(self, name, age):
        self.name = name
        self.age = age
        print("Student object created")

\# Object creation automatically calls constructor
student1 = Student("Alice", 20)
```

- **આપમેળે એક્ઝિક્યુશન**: Object બનાવાતી વખતે તરત જ run થાય છે
- **ઇનિશિયલાઇઝેશન**: Object ની શરૂઆતી state set કરે છે
- **self પેરામીટર**: હાલનો object જે બનાવાઈ રહ્યો છે તેનો reference

યાદી કૌશલ્ય: ``Initialize, Automatic, Self''

પ્રશ્ન 4(બ OR) [4 ગુણ]

min ફંક્શનનો ઉપયોગ કર્યા વિના લિસ્ટમાંથી સૌથી નાનો નંબર શોધવા માટેનો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ manually બધા elements ની સરખામણી કરીને સૌથી નાની value શોધે છે.
મિનિમમ શોધવાનો એલ્ગોરિધમ:

પગલું	ક્રિયા	કોડ
1	પહેલું smallest માનો	`smallest = list[0]`
2	બીજાઓ સાથે સરખાવો	`for num in list[1:]:`
3	નાનું મળે તો અપડેટ કરો	`if num < smallest:`
4	પરિણામ દર્શાવો	`print(smallest)`

સંપૂર્ણ પ્રોગ્રામ:

```
\# Find smallest number without min()
numbers = [45, 23, 67, 12, 89, 5, 34]

smallest = numbers[0]  \#     smallest

for i in range(1, len(numbers)):
    if numbers[i] {} smallest:
        smallest = numbers[i]

print(f"Smallest number: \{smallest\}")
```

વૈકલ્પિક મેથડ:

```
\# Using for loop with list elements
numbers = [45, 23, 67, 12, 89, 5, 34]
smallest = numbers[0]

for num in numbers[1:]:
    if num {} smallest:
        smallest = num

print(f"Smallest number: \{smallest\}")
```

અપેક્ષિત આઉટપુટ:

```
Smallest number: 5
```

- **કમ્પેરિઝન લૉજિક**: દરેક element ને current smallest સાથે compare કરો
- **અપડેટ સ્ટ્રેટેજી**: નાનો number મળે ત્યારે smallest replace કરો
- **લિનિયર સર્ચ**: બધા elements ને એકવાર ચકાસો

યાદી કૌશલ્ય: ``માનો, સરખાવો, અપડેટ કરો, દર્શાવો''

પ્રશ્ન 4(ક OR) [7 ગુણ]

Python માં યુઝર ડિફાઇન્ડ મોડ્યુલ્સનું કાર્ય સમજાવો.

જવાબ

User-defined modules એ custom Python files છે જેમાં functions, classes અને variables હોય છે જે અન્ય programs માં import અને use કરી શકાય છે.

મોડ્યુલ ઘટકો ટેબલ:

ઘટક	હેતુ	ઉદાહરણ
Functions	પુનઃઉપયોગી કોડ blocks	`def calculate_area():`
Classes	Object blueprints	`class Shape:`
Variables	સ્પેર્ડ ડેટા	`PI = 3.14159`
Constants	નિશ્ચિત વેલ્યુઝ	`MAX_SIZE = 100`

મોડ્યુલ બનાવવાની પ્રક્રિયા:

```
    Step 1: .py file
         |
         v
    Step 2: Functions/classes
         |
         v
    Step 3: File save
         |
         v
    Step 4:    programs   import
         |
         v
    Step 5: Module functions
```

ઉદાહરણ મોડ્યુલ બનાવવું:
ફાઇલ: math_operations.py

```
\# User{-defined module}
PI = 3.14159

def calculate\_circle\_area(radius):
    return PI * radius * radius

def calculate\_rectangle\_area(length, width):
    return length * width

class Calculator:
    def add(self, a, b):
        return a + b

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

મોડ્યુલ વાપરવું:
ઇમ્પોર્ટ મેથડ્સ ટેબલ:

મેથડ	Syntax	ઉપયોગ
સંપૂર્ણ module import	`import math_operations`	`math_operations.calculate_circle_area(5)`
વિશિષ્ટ function import	`from math_operations import calculate_circle_area`	`calculate_circle_area(5)`
Alias સાથે import	`import math_operations as math_ops`	`math_ops.PI`
બધું import	`from math_operations import *`	`calculate_circle_area(5)`

મુખ્ય પ્રોગ્રામ:

```
\# main.py {- Module    }
import math\_operations

\# Module functions
radius = 5
area = math\_operations.calculate\_circle\_area(radius)
print(f"Circle area: \{area\}")

\# Module variables
print(f"PI value: \{math\_operations.PI\}")

\# Module classes
calc = math\_operations.Calculator()
result = calc.add(10, 20)
print(f"Addition result: \{result\}")
```

મોડ્યુલ લાભો:

- **કોડ પુનઃઉપયોગીતા**: એકવાર લખો, અનેક programs માં વાપરો
- **સંગઠન**: સંબંધિત functions એકસાથે રાખો
- **નેમસ્પેસ**: Naming conflicts ટાળો
- **જાળવણીયોગ્યતા**: Update અને debug કરવું સરળ
- **સહયોગ**: અન્ય developers સાથે modules share કરો

મોડ્યુલ સર્ચ પાથ:

1. વર્તમાન directory
2. PYTHONPATH environment variable
3. Standard library directories
4. Site-packages directory

બેસ્ટ પ્રેક્ટિસિસ:

- વર્ણનાત્મક module names વાપરો
- Documentation માટે docstrings include કરો
- સંબંધિત functionality એકસાથે રાખો
- Circular imports ટાળો

યાદી કૌશલ્ય: ``ફાઇલ બનાવો, ફંક્શન્સ ડિફાઇન કરો, ઇમ્પોર્ટ કરો, વાપરો''

પ્રશ્ન 5(અ) [3 ગુણ]

ઉદાહરણ સાથે Python માં સિંગલ ઇન્હેરિટન્સ સમજાવો.

જવાબ

Single inheritance એ જ્યારે એક class બરાબર એક parent class પાસેથી properties અને methods inherit કરે છે.

ઇન્હેરિટન્સ માળખું ટેબલ:

ઘટક	ભૂમિકા	ઉદાહરણ
Parent Class	Base/Super class	`class Animal:`
Child Class	Derived/Sub class	`class Dog(Animal):`
Inheritance	`class Child(Parent):`	`class Dog(Animal):`

ઇન્હેરિટન્સ આકૃતિ:

```
    Parent Class: Animal

        Attributes:
        {- name             }
        {- age              }

        Methods:
        {- eat()            }
        {- sleep()          }


                 inherits
                v
    Child Class: Dog

      Inherited:
        {- name, age        }
        {- eat(), sleep()   }

      Own Methods:
        {- bark()           }
```

ઉદાહરણ કોડ:

```
\# Parent class
class Animal:
    def \_\_init\_\_(self, name):
        self.name = name

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

\# Child class inheriting from Animal
class Dog(Animal):
    def bark(self):
        print(f"\{self.name\} is barking")

\# Using inheritance
my\_dog = Dog("Buddy")
my\_dog.eat()    \# Inherited method
my\_dog.bark()   \# Own method
```

- **કોડ પુનઃઉપયોગ**: Child class ને parent ની functionality આપમેળે મળે છે
- **વિસ્તરણ**: Child નવા methods અને attributes ઉમેરી શકે છે
- **Is-a સંબંધ**: Dog is-a Animal

યાદી કૌશલ્ય: ``એક Parent, એક Child''

પ્રશ્ન 5(બ) [4 ગુણ]

Python માં એબ્સ્ટ્રેક્શનની વિભાવના અને તેના લાભો સમજાવો.

જવાબ

Abstraction જટિલ implementation details છુપાવે છે અને user ને માત્ર આવશ્યક features બતાવે છે.
એબ્સ્ટ્રેક્શન કન્સેપ્ટ્સ ટેબલ:

કન્સેપ્ટ	વર્ણન	ઉદાહરણ
Abstract Class	Instantiate કરી શકાતું નથી	`class Shape(ABC):`
Abstract Method	Implement કરવું જ પડે	`@abstractmethod`
Interface	Method structure define કરે	`def area(self):`

એબ્સ્ટ્રેક્શન Implementation:

```
from abc import ABC, abstractmethod

\# Abstract class
class Shape(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

\# Concrete class
class Rectangle(Shape):
    def \_\_init\_\_(self, length, width):
        self.length = length
        self.width = width

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

    def perimeter(self):
        return 2 * (self.length + self.width)
```

લાભો ટેબલ:

લાભ	વર્ણન	ફાયદો
સરળતા	જટિલ details છુપાવે	વાપરવામાં સરળ
સુરક્ષા	આંતરિક implementation છુપાવે	ડેટા સુરક્ષા
જાળવણીયોગ્યતા	Implementation બદલી શકાય	લવચીક અપડેટ્સ
કોડ સંગઠન	સ્પષ્ટ માળખું	બહેતર ડિઝાઇન

- **જટિલતા છુપાવવી**: Users ને આંતરિક workings જાણવાની જરૂર નથી
- **સુસંગત Interface**: બધા child classes એક જ structure અનુસરે છે
- **Implementation ફરજિયાત**: Abstract methods child classes માં define કરવા જ પડે

યાદી કૌશલ્ય: ``વિગતો છુપાવો, Interface બતાવો''

પ્રશ્ન 5(ક) [7 ગુણ]

મલ્ટિપલ અને મલ્ટિ-લેવલ ઇન્હેરિટન્સનું કાર્ય દર્શાવતો Python પ્રોગ્રામ વિકસાવો.

જવાબ

પ્રોગ્રામ બંને inheritance types દર્શાવે છે: multiple (અનેક parents) અને multi-level (inheritance ની chain).

ઇન્હેરિટન્સ પ્રકારો સરખામણી:

પ્રકાર	માળખું	ઉદાહરણ
Multiple	Child અનેક parents પાસેથી inherit કરે	`class C(A, B):`

Multi-level	Grandparent $\rightarrow Parent \rightarrow Child$	`class C(B):` જ્યાં `class B(A):`

ઇન્હેરિટન્સ પદાનુક્રમ:

```
Multiple Inheritance:
    Father    Mother
      {       /}
       {     /}
        Child

Multi{-level Inheritance:}
    Animal
      |
      v
    Mammal
      |
      v
     Dog
```

સંપૂર્ણ પ્રોગ્રામ:

```
\# Multi{-level Inheritance Demo}
print("=== Multi{-level Inheritance ===")}

class Animal:
    def \_\_init\_\_(self, name):
        self.name = name

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

class Mammal(Animal):  \# Animal     inherit
    def breathe(self):
        print(f"\{self.name\} breathes air")

class Dog(Mammal):     \# Mammal     inherit     (  Animal     inherit    )
    def bark(self):
        print(f"\{self.name\} can bark")

\# Multi{-level inheritance      }
my\_dog = Dog("Buddy")
my\_dog.eat()     \# Animal     (grandparent)
my\_dog.breathe() \# Mammal     (parent)
my\_dog.bark()    \#     method

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

class Father:
    def father\_method(self):
        print("Method from Father class")

class Mother:
    def mother\_method(self):
        print("Method from Mother class")

class Child(Father, Mother):  \# Father   Mother       inherit
    def child\_method(self):
        print("Method from Child class")

\# Multiple inheritance
child = Child()
child.father\_method()  \# Father
child.mother\_method()  \# Mother
child.child\_method()   \#     method
```

```
\# Inheritance
print(f"{n}Child inherits from Father: \{issubclass(Child, Father)\}")
print(f"Child inherits from Mother: \{issubclass(Child, Mother)\}")
```

અપેક્ષિત આઉટપુટ:

```
=== Multi-level Inheritance ===
Buddy can eat
Buddy breathes air
Buddy can bark

=== Multiple Inheritance ===
Method from Father class
Method from Mother class
Method from Child class

Child inherits from Father: True
Child inherits from Mother: True
```

મુખ્ય તફાવતો:

પાસું	Multiple	Multi-level
Parents	2 અથવા વધુ direct parents	Single parent chain
Syntax	`class C(A, B):`	`class C(B):` જ્યાં `B(A):`
Inheritance	આડી	ઊભી
જટિલતા	વધુ (diamond problem)	ઓછી

મેથડ રિઝોલ્યુશન ઓર્ડર (MRO):
- **Multiple**: Python ડાબેથી-જમણે order અનુસરે છે
- **Multi-level**: Inheritance chain ઉપર જાય છે

યાદી કૌશલ્ય: ``અનેક Parents, મલ્ટિ-લેવલ Chain''

પ્રશ્ન 5(અ OR) [3 ગુણ]

Python માં આવતી 3 પ્રકારની મેથડ્સનું કાર્ય સમજાવો.

જવાબ

Python classes માં ત્રણ પ્રકારની methods છે જે class data ને કેવી રીતે access કરે છે તેના આધારે.
મેથડ પ્રકારો ટેબલ:

મેથડ પ્રકાર	ડેકોરેટર	પ્રથમ Parameter	હેતુ
Instance Method	કોઈ નહીં	`self`	Instance data access
Class Method	`@classmethod`	`cls`	Class data access
Static Method	`@staticmethod`	કોઈ નહીં	Utility functions

ઉદાહરણ કોડ:

```
class Student:
    school\_name = "ABC School"  \# Class variable

    def \_\_init\_\_(self, name):
        self.name = name        \# Instance variable

    \# Instance method
    def display\_info(self):
        print(f"Student: \{self.name\}")

    \# Class method
    @classmethod
    def get\_school(cls):
        return cls.school\_name

    \# Static method
    @staticmethod
    def is\_adult(age):
        return age {=} 18

\#
student = Student("Alice")
student.display\_info()           \# Instance method
print(Student.get\_school())      \# Class method
print(Student.is\_adult(20))      \# Static method
```

- **ઇન્સ્ટન્સ મેથડ્સ**: `self` વાપરીને object-specific data સાથે કામ કરે છે
- **ક્લાસ મેથડ્સ**: `cls` વાપરીને class-wide data સાથે કામ કરે છે
- **સ્ટેટિક મેથડ્સ**: સ્વતંત્ર utility functions

યાદી કૌશલ્ય: ``Instance Self, Class Cls, Static કોઈ નહીં''

પ્રશ્ન 5(બ OR) [4 ગુણ]

Python માં ઇન્હેરિટન્સ દ્વારા પોલીમોર્ફિઝમ સમજાવો.

જવાબ

Polymorphism વિવિધ classes ના objects ને સામાન્ય base class ના objects તરીકે treat કરવાની મંજૂરી આપે છે, દરેક પોતાની રીતે methods implement કરે છે.

પોલીમોર્ફિઝમ કન્સેપ્ટ ટેબલ:

પાસું	વર્ણન	ઉદાહરણ
સમાન Interface	સામાન્ય method names	`area()` method
અલગ Implementation	દરેક class નું પોતાનું version	Rectangle vs Circle area
Runtime Decision	Execution દરમ્યાન method પસંદ	Dynamic binding

પોલીમોર્ફિઝમ ઉદાહરણ:

```
\# Base class
class Shape:
    def area(self):
        pass

\#    implementations
class Rectangle(Shape):
    def \_\_init\_\_(self, length, width):
        self.length = length
        self.width = width

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

class Circle(Shape):
    def \_\_init\_\_(self, radius):
        self.radius = radius

    def area(self):
        return 3.14 * self.radius * self.radius

\# Polymorphic behavior
shapes = [Rectangle(5, 3), Circle(4)]

for shape in shapes:
    print(f"Area: \{shape.area()\}")  \#    method,
```

લાભો:

- **લવચીકતા:** સમાન કોડ વિવિધ object types સાથે કામ કરે છે
- **વિસ્તરણશીલતા:** વર્તમાન કોડ બદલ્યા વિના નવા classes ઉમેરવા સરળ
- **જાળવણીયોગ્યતા:** એક class માં ફેરફાર અન્યને અસર કરતો નથી

યાદી કૌશલ્ય: ``સમાન નામ, અલગ વર્તન''

પ્રશ્ન 5(ક OR) [7 ગુણ]

હાઇબ્રિડ ઇન્હેરિટન્સનું કાર્ય દર્શાવતો Python પ્રોગ્રામ વિકસાવો.

જવાબ

Hybrid inheritance એ single program structure માં multiple અને multi-level inheritance ને combine કરે છે.

હાઇબ્રિડ ઇન્હેરિટન્સ માળખું:

```
        Animal (Base)
          |
          v
        Mammal
       /      {}
      v        v
    Dog       Cat
      {       /}
       {    /}
        v  v
     Hybrid Pet
```

હાઇબ્રિડ માં ઇન્હેરિટન્સ પ્રકારો:

લેવલ	પ્રકાર	Classes
1	Single	Animal $\rightarrow Mammal$

2	Multiple	Mammal $\rightarrow Dog, Cat$
3	Multiple	Dog, Cat $\rightarrow Pet$

સંપૂર્ણ પ્રોગ્રામ:

```
\# Hybrid Inheritance Demonstration

print("=== Hybrid Inheritance Demo ===")

\# Base class (Level 1)
class Animal:
    def \_\_init\_\_(self, name):
        self.name = name

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

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

\# Single inheritance (Level 2)
class Mammal(Animal):
    def breathe(self):
        print(f"\{self.name\} breathes air")

    def give\_birth(self):
        print(f"\{self.name\} gives birth to babies")

\# Multiple inheritance branches (Level 3)
class Dog(Mammal):
    def bark(self):
        print(f"\{self.name\} barks: Woof!")

    def loyalty(self):
        print(f"\{self.name\} is loyal to owner")

class Cat(Mammal):
    def meow(self):
        print(f"\{self.name\} meows: Meow!")

    def independence(self):
        print(f"\{self.name\} is independent")

\# Hybrid class {- Multiple inheritance (Level 4)}
class HybridPet(Dog, Cat):
    def \_\_init\_\_(self, name, breed):
        super().\_\_init\_\_(name)
        self.breed = breed

    def play(self):
        print(f"\{self.name\} loves to play")

    def show\_info(self):
        print(f"Name: \{self.name\}, Breed: \{self.breed\}")

\# Creating and using hybrid inheritance
print("{n}{-{-}{-} Creating Hybrid Pet {-}{-}{-}"})
pet = HybridPet("Buddy", "Labrador{-Persian Mix"})

print("{n}{-{-}{-} Methods from Animal (Great{-}grandparent) {-}{-}{-}"})
pet.eat()
pet.sleep()

print("{n}{-{-}{-} Methods from Mammal (Grandparent) {-}{-}{-}"})
pet.breathe()
```

```
pet.give\_birth()

print("{n}{-{-}{-} Methods from Dog (Parent 1) {-}{-}{-}"})
pet.bark()
pet.loyalty()

print("{n}{-{-}{-} Methods from Cat (Parent 2) {-}{-}{-}"})
pet.meow()
pet.independence()

print("{n}{-{-}{-} Own Methods {-}{-}{-}"})
pet.play()
pet.show\_info()

print("{n}{-{-}{-} Inheritance Chain {-}{-}{-}"})
print(f"MRO (Method Resolution Order): \{HybridPet.\_\_mro\_\_\}")

\# Checking inheritance relationships
print(f"{n}Is HybridPet subclass of Animal? \{issubclass(HybridPet, Animal)\}")
print(f"Is HybridPet subclass of Dog? \{issubclass(HybridPet, Dog)\}")
print(f"Is HybridPet subclass of Cat? \{issubclass(HybridPet, Cat)\}")
```

અપેક્ષિત આઉટપુટ:

```
=== Hybrid Inheritance Demo ===

--- Creating Hybrid Pet ---

--- Methods from Animal (Great-grandparent) ---
Buddy can eat
Buddy can sleep

--- Methods from Mammal (Grandparent) ---
Buddy breathes air
Buddy gives birth to babies

--- Methods from Dog (Parent 1) ---
Buddy barks: Woof!
Buddy is loyal to owner

--- Methods from Cat (Parent 2) ---
Buddy meows: Meow!
Buddy is independent

--- Own Methods ---
Buddy loves to play
Name: Buddy, Breed: Labrador-Persian Mix

--- Inheritance Chain ---
MRO (Method Resolution Order): (<class '__main__.HybridPet'>, <class '__main__.Dog'>, <class '__main__.

Is HybridPet subclass of Animal? True
Is HybridPet subclass of Dog? True
Is HybridPet subclass of Cat? True
```

હાઇબ્રિડ ઇન્હેરિટન્સની મુખ્ય વિશેષતાઓ:

- **જટિલ માળખું**: વિવિધ inheritance types ને combine કરે છે
- **મેથડ રિઝોલ્યુશન ઓર્ડર**: Python method lookup માટે વિશિષ્ટ order અનુસરે છે
- **ડાયમંડ પ્રોબ્લેમ**: Python ના MRO દ્વારા આપમેળે handle થાય છે
- **લવચીકતા**: અનેક parent classes પાસેથી methods ને access

લાભો:

- **સમૃદ્ધ Functionality**: અનેક sources પાસેથી inherit કરે છે

- **કોડ પુનઃઉપયોગ**: વર્તમાન કોડનો મહત્તમ ઉપયોગ
- **સંબંધ મોડેલિંગ**: જટિલ વાસ્તવિક સંબંધો દર્શાવે છે

પડકારો:

- **જટિલતા**: સમજવું અને maintain કરવું કઠિન
- **નામ સંઘર્ષ**: અનેક parents પાસે સમાન method names હોઈ શકે
- **મેમોરી ઉપયોગ**: Objects વધુ overhead carry કરે છે

યાદી કૌશલ્ય: ``હાઇબ્રિડ બધા પ્રકારો Combine કરે છે''