Subject Name (Gujarati)

4351108 -- Winter 2023

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 ગુણ]

પાયથન પ્રોગ્રામિંગ લેન્ગવેજના કોઈ પણ 6 ઉપયોગો લખો.

જવાબ

પાયથનના ઉપયોગોનું ટેબલ:

ઉપયોગ ક્ષેત્ર	વર્ણન
વેબ ડેવેલપમેન્ટ	Django, Flask frameworks
ડેટા સાયન્સ	Analysis અને visualization
મશીન લર્નિંગ	AI model development
ડેસ્કટોપ એપ્લિકેશન	GUI using Tkinter, PyQt
ગેમ ડેવેલપમેન્ટ	Pygame library
ઑટોમેશન	Scripting અને testing

સ્મરણ સૂત્ર: ``Web Data Machine Desktop Game Auto''

પ્રશ્ન 1(બ) [4 ગુણ]

પાયથન પ્રોગ્રામિંગ લેન્ગવેજની કોઈ પણ 8 વિશેષતાઓ લખો.

જવાબ

પાયથનની વિશેષતાઓનું ટેબલ:

વિશેષતા	વર્ણન
સરળ સિન્ટેક્સ	વાંચવા અને લખવામાં સરળ
ઇન્ટરપ્રિટેડ	Compilation ની જરૂર નથી
ઑબ્જેક્ટ-ઓરિએન્ટેડ	OOP concepts સપોર્ટ કરે છે
ડાયનેમિક ટાઇપિંગ	Variables ને type declaration જરૂરી નથી
ક્રોસ-પ્લેટફોર્મ	Multiple OS પર ચાલે છે
મોટી લાઇબ્રેરીઓ	Rich standard library
ઓપન સોર્સ	ઉપયોગ અને modify કરવા માટે મફત
ઇન્ટરેક્ટિવ	REPL environment

સ્મરણ સૂત્ર: ``Simple Interpreted Object Dynamic Cross Large Open Interactive''

પ્રશ્ન 1(ક) [7 ગુણ]

પાયથનની for અને while લૂપનું કાર્ય સમજાવો.

જવાબ

For Loop:

- **પુનરાવર્તન**: Sequences પર પુનરાવર્તન કરે છે (lists, strings, ranges)
- **સિન્ટેક્સ**: `for variable in sequence:`
- **આપોઆપ**: Iteration આપોઆપ handle કરે છે

While Loop:

- **શરત આધારિત**: જ્યાં સુધી શરત સાચી રહે છે
- **મેન્યુઅલ નિયંત્રણ**: Programmer iteration નો નિયંત્રણ કરે છે
- **જોખમ**: શરત કદી false ન બને તો infinite loop બની શકે છે

ડાયાગ્રામ:

```
    Start
      |
   Initialize
      |
    Condition? {-{-}{-}{-}{-}No{-}{-}{-}{-}{-} End}
      |Yes
    Execute
      |
   Update
      |
    (loop back)
```

કોડ ઉદાહરણ:

```
\# For loop
for i in range(5):
    print(i)

\# While loop
i = 0
while i {} 5:
    print(i)
    i += 1
```

સ્મરણ સૂત્ર: ``For આપોઆપ, While મેન્યુઅલ''

પ્રશ્ન 1(ક OR) [7 ગુણ]

પાયથનના break, continue અને pass સ્ટેટમેન્ટના કાર્ય સમજાવો.

જવાબ

Break Statement:
- **બહાર નીકળવું**: સંપૂર્ણ loop ને terminate કરે છે
- **ઉપયોગ**: જ્યારે કોઈ specific condition મળે છે
- **અસર**: Control loop પછીના statement પર જાય છે

Continue Statement:
- **છોડીને આગળ**: ફક્ત current iteration skip કરે છે
- **ઉપયોગ**: Iteration માં specific values skip કરવા માટે
- **અસર**: Next iteration પર જાય છે

Pass Statement:
- **Placeholder**: કંઈ કરતું નથી, syntactic placeholder
- **ઉપયોગ**: જ્યારે syntax statement જોઈએ પણ કોઈ action નહીં
- **અસર**: કોઈ operation perform કરતું નથી

કોડ ઉદાહરણો:

```
\# Break
for i in range(10):
if

i == 5:

        break
    print(i)  \# prints 0,1,2,3,4

\# Continue
for i in range(5):
```

```
if

i == 2:

        continue
    print(i)  \# prints 0,1,3,4

\# Pass
if True:
    pass  \# placeholder
```

સ્મરણ સૂત્ર: ``Break બહાર, Continue છોડીને, Pass રાહ''

પ્રશ્ન 2(અ) [3 ગુણ]

લિસ્ટના દરેક ઘટકનું મૂલ્ય 1 થી વધારવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Method 1 {- Using for loop}
numbers = [1, 2, 3, 4, 5]
for i in range(len(numbers)):
    numbers[i] += 1
print(numbers)

\# Method 2 {- List comprehension}
numbers = [1, 2, 3, 4, 5]
result = [x + 1 for x in numbers]
print(result)
```

સ્મરણ સૂત્ર: ``Loop Index અથવા Comprehension''

પ્રશ્ન 2(બ) [4 ગુણ]

વપરાશકર્તા પાસેથી 3 સંખ્યા લઈ તેની સરેરાશ શોધવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Input three numbers
num1 = float(input("Enter first number: "))
num2 = float(input("Enter second number: "))
num3 = float(input("Enter third number: "))

\# Calculate average
average = (num1 + num2 + num3) / 3

\# Display result
print(f"Average is: \{average\}")
```

મુખ્ય મુદ્દાઓ:

- **ઇનપુટ**: દશાંશ સંખ્યાઓ માટે `float()` ઉપયોગ કરો
- **સૂત્ર**: બધાનો સરવાળો કરીને સંખ્યા વડે ભાગો
- **આઉટપુટ**: Formatting માટે f-string ઉપયોગ કરો

સ્મરણ સૂત્ર: ``Input Float, Sum Divide, Format Output''

પ્રશ્ન 2(ક) [7 ગુણ]

પાયથનનો list ડેટા ટાઈપ વિસ્તારથી સમજાવો.

જવાબ

લિસ્ટની લાક્ષણિકતાઓ:

- **ક્રમબદ્ધ**: Elements sequence જાળવે છે
- **બદલાવ પાત્ર**: બનાવ્યા પછી modify કરી શકાય છે
- **વિવિધ પ્રકારની**: વિવિધ data types store કરી શકે છે
- **ઇન્ડેક્સવાળી**: Index વડે elements ને access કરી શકાય છે (0-based)

લિસ્ટ ઑપરેશન્સ ટેબલ:

ઑપરેશન	Syntax	વર્ણન
બનાવવું	`list = [1,2,3]`	નવી list બનાવો
એક્સેસ	`list[0]`	Index વડે element મેળવો
Append	`list.append(4)`	અંતે element ઉમેરો
Insert	`list.insert(1,5)`	Specific position પર ઉમેરો
Remove	`list.remove(2)`	પહેલું occurrence દૂર કરો
Pop	`list.pop()`	છેલ્લું element દૂર કરીને return કરો
Slice	`list[1:3]`	Sublist મેળવો

કોડ ઉદાહરણ:

```
\# List creation and operations
fruits = [{apple}, {banana}, {orange}]
fruits.append({mango})
fruits.insert(1, {grape})
print(fruits[0])  \# apple
print(len(fruits))  \# 5
```

સ્મરણ સૂત્ર: ``ક્રમબદ્ધ બદલાવપાત્ર વિવિધપ્રકારની ઇન્ડેક્સવાળી''

પ્રશ્ન 2(અ OR) [3 ગુણ]

for લૂપની મદદથી લિસ્ટના દરેક ઘટકનો સરવાળો શોધવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Method 1 {- Traditional for loop}
numbers = [10, 20, 30, 40, 50]
total = 0
for num in numbers:
    total += num
print(f"Sum is: \{total\}")

\# Method 2 {- Using range and index}
numbers = [10, 20, 30, 40, 50]
total = 0
for i in range(len(numbers)):
    total += numbers[i]
print(f"Sum is: \{total\}")
```

સ્મરણ સૂત્ર: ``શૂન્ય શરૂઆત, લૂપ ઉમેરો, કુલ પ્રિન્ટ''

પ્રશ્ન 2(બ OR) [4 ગુણ]

વપરાશકર્તા પાસેથી લીધેલા મૂડલ, વ્યાજદર અને વર્ષ પરથી સાદું વ્યાજ શોધવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Get input from user
principal = float(input("Enter principal amount: "))
rate = float(input("Enter rate of interest: "))
time = float(input("Enter time in years: "))

\# Calculate simple interest
simple\_interest = (principal * rate * time) / 100

\# Display results
print(f"Principal: \{principal\}")
print(f"Rate: \{rate\}\%")
print(f"Time: \{time\} years")
print(f"Simple Interest: \{simple\_interest\}")
print(f"Total Amount: \{principal + simple\_interest\}")
```

સૂત્ર:

- **સાદું વ્યાજ** = (P $\times R \times T)/100$
- **કુલ રકમ** = મૂડલ + સાદું વ્યાજ

સ્મરણ સૂત્ર: ``મૂડલ દર સમય, ગુણો ભાગો સો''

પ્રશ્ન 2(ક OR) [7 ગુણ]

પાયથનનો tuple ડેટા ટાઈપ વિસ્તારથી સમજાવો.

જવાબ

ટ્યૂપલની લાક્ષણિકતાઓ:

- **ક્રમબદ્ધ**: Elements sequence જાળવે છે
- **અપરિવર્તનીય**: બનાવ્યા પછી modify કરી શકાતું નથી
- **વિવિધ પ્રકારની**: વિવિધ data types store કરી શકે છે
- **ઇન્ડેક્સવાળી**: Index વડે access કરી શકાય છે (0-based)

ટ્યૂપલ ઑપરેશન્સ ટેબલ:

ઑપરેશન	Syntax	વર્ણન
બનાવવું	`tuple = (1,2,3)`	નવું tuple બનાવો
એક્સેસ	`tuple[0]`	Index વડે element મેળવો
Count	`tuple.count(2)`	Occurrences ગિનો
Index	`tuple.index(3)`	પહેલો index શોધો
Slice	`tuple[1:3]`	Sub-tuple મેળવો
Length	`len(tuple)`	Tuple નું size મેળવો
Concatenate	`tuple1 + tuple2`	Tuples જોડો

કોડ ઉદાહરણ:

```
\# Tuple creation and operations
coordinates = (10, 20, 30)
print(coordinates[0])  \# 10
print(len(coordinates))  \# 3
x, y, z = coordinates  \# tuple unpacking
new\_tuple = coordinates + (40, 50)
```

લિસ્ટ સાથે મુખ્ય તફાવત:

- **અપરિવર્તનીય**: Elements બદલી શકાતા નથી
- **પરફોર્મન્સ**: Lists કરતા વધુ ઝડપી
- **ઉપયોગ**: Fixed data collections માટે

સ્મરણ સૂત્ર: ``ક્રમબદ્ધ અપરિવર્તનીય વિવિધપ્રકારની ઇન્ડેક્સવાળી''

પ્રશ્ન 3(અ) [3 ગુણ]

random મોડ્યૂલની કોઈ પણ 3 મેથડ સમજાવો.

જવાબ

Random મોડ્યૂલ મેથડ્સ ટેબલ:

મેથડ	Syntax	વર્ણન
random()	`random.random()`	0.0 થી 1.0 વચ્ચે float
randint()	`random.randint(1,10)`	આપેલી range વચ્ચે integer
choice()	`random.choice(list)`	Sequence માંથી random element

કોડ ઉદાહરણ:

```
import random

\# Generate random float
print(random.random())  \# 0.7234567

\# Generate random integer
print(random.randint(1, 10))  \# 7

\# Choose random element
colors = [{red}, {blue}, {green}]
print(random.choice(colors))  \# blue
```

સ્મરણ સૂત્ર: ``Random Float, Randint Integer, Choice Select''

પ્રશ્ન 3(બ) [4 ગુણ]

વપરાશકર્તા પાસેથી એક સ્ટ્રિંગ લઈને એમાંના દરેક `a' નું સ્થાન પ્રિન્ટ કરવાનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Get string from user
text = input("Enter a string: ")

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

\# Display results
if positions:
    print(f"Letter {a found at positions: }\{positions\}")
else:
    print("Letter {a not found in the string"})

\# Alternative method using enumerate
text = input("Enter a string: ")
for index, char in enumerate(text):
    if char.lower() == {a}:
        print(f"{a found at position }\{index\}")
```

મુખ્ય મુદ્દાઓ:

- **Case-insensitive**: `a' અને `A' બંને શોધવા માટે `.lower()` ઉપયોગ કરો
- **Index tracking**: range અથવા enumerate ઉપયોગ કરો
- **આઉટપુટ ફોર્મેટ**: સ્પષ્ટ position indication

સ્મરણ સૂત્ર: ``લૂપ ઇન્ડેક્સ ચેક ઉમેરો પ્રિન્ટ''

પ્રશ્ન 3(ક) [7 ગુણ]

પાયથનનો string ડેટા ટાઈપ વિસ્તારથી સમજાવો.

જવાબ

સ્ટ્રિંગની લાક્ષણિકતાઓ:
- **અપરિવર્તનીય**: બનાવ્યા પછી બદલી શકાતું નથી
- **અનુક્રમ**: Characters નો ordered collection
- **ઇન્ડેક્સવાળી**: Index વડે characters ને access કરી શકાય છે
- **યુનિકોડ**: બધી ભાષાઓ અને symbols સપોર્ટ કરે છે

સ્ટ્રિંગ મેથડ્સ ટેબલ:

મેથડ	ઉદાહરણ	વર્ણન
upper()	`"hello".upper()`	Uppercase માં convert કરો
lower()	`"HELLO".lower()`	Lowercase માં convert કરો
strip()	`" hello ".strip()`	Whitespace દૂર કરો
split()	`"a,b,c".split(",")`	List માં split કરો
replace()	`"hello".replace("l","x")`	Substring replace કરો
find()	`"hello".find("e")`	Substring index શોધો
join()	`",".join(["a","b"])`	List elements join કરો

સ્ટ્રિંગ ઑપરેશન્સ:

```
\# String creation
name = "Python Programming"

\# String indexing and slicing
print(name[0])      \# P
print(name[0:6])    \# Python
print(name[{-}1])     \# g

\# String formatting
age = 25
message = f"I am \{age\} years old"
```

મુખ્ય વિશેષતાઓ:
- **જોડાણ**: + operator વડે
- **પુનરાવર્તન**: * operator વડે
- **સભ્યપદ**: `in' operator વડે
- **ફોર્મેટિંગ**: f-strings, .format(), % formatting

સ્મરણ સૂત્ર: ``અપરિવર્તનીય અનુક્રમ ઇન્ડેક્સવાળી યુનિકોડ''

પ્રશ્ન 3(અ OR) [3 ગુણ]

math મોડ્યૂલની કોઈ પણ 3 મેથડ સમજાવો.

જવાબ

Math મોડ્યૂલ મેથડ્સ ટેબલ:

મેથડ	Syntax	વર્ણન
sqrt()	`math.sqrt(16)`	Square root ગણતરી
pow()	`math.pow(2,3)`	Power ગણતરી
ceil()	`math.ceil(4.3)`	Integer માં round up

કોડ ઉદાહરણ:

```
import math

\# Square root
print(math.sqrt(25))    \# 5.0

\# Power
print(math.pow(2, 3))   \# 8.0

\# Ceiling
print(math.ceil(4.2))   \# 5
```

સ્મરણ સૂત્ર: ``Square Root, Power Up, Ceiling Round''

પ્રશ્ન 3(બ OR) [4 ગુણ]

વપરાશકર્તા પાસેથી એક સ્ટ્રિંગ લઈને એમાં રહેલા અંગ્રેજી સ્વરોની સંખ્યા શોધવાનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Get string from user
text = input("Enter a string: ")

\# Define vowels
vowels = "aeiouAEIOU"

\# Count vowels
vowel\_count = 0
for char in text:
    if char in vowels:
        vowel\_count += 1

\# Display result
print(f"Total vowels in {}\{text\}{: }\{vowel\_count\}")

\# Alternative method using list comprehension
text = input("Enter a string: ")
vowels = "aeiouAEIOU"
count = sum(1 for char in text if char in vowels)
print(f"Total vowels: \{count\}")
```

મુખ્ય મુદ્દાઓ:
- **સ્વર વ્યાખ્યા**: બંને cases શામેલ કરો
- **લૂપ કરો**: String ના દરેક character માં
- **ગિનતી તર્ક**: Membership check કરો અને increment કરો

સ્મરણ સૂત્ર: ``સ્વર વ્યાખ્યા, લૂપ ચેક, ગિનતી વધારો''

પ્રશ્ન 3(ક OR) [7 ગુણ]

પાયથનનો set ડેટા ટાઈપ વિસ્તારથી સમજાવો.

જવાબ

સેટની લાક્ષણિકતાઓ:
- **અક્રમ**: Elements નો કોઈ નિશ્ચિત sequence નથી
- **બદલાવ પાત્ર**: Elements ઉમેરી/દૂર કરી શકાય છે
- **અનન્ય**: Duplicate elements allowed નથી

- **પુનરાવર્તનીય**: Elements માં loop કરી શકાય છે

સેટ ઑપરેશન્સ ટેબલ:

ઑપરેશન	Syntax	વર્ણન
બનાવવું	`set = {1,2,3}`	નવો set બનાવો
Add	`set.add(4)`	Single element ઉમેરો
Remove	`set.remove(2)`	Element દૂર કરો (error if not found)
Discard	`set.discard(2)`	Element દૂર કરો (no error)
સંયોજન	`set1 | set2`	Sets જોડો
છેદ	`set1 & set2`	સામાન્ય elements
તફાવત	`set1 - set2`	ફક્ત set1 માંના elements

સેટ ગણિતીય ઑપરેશન્સ:

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

\# Set operations
print(A | B)    \# Union: \{1,2,3,4,5,6\}
print(A \& B)    \# Intersection: \{3,4\}
print(A {-} B)    \# Difference: \{1,2\}
print(A \^{} B)    \# Symmetric difference: \{1,2,5,6\}
```

મુખ્ય ઉપયોગો:
- **Duplicates દૂર કરવા**: Lists માંથી
- **ગણિતીય ઑપરેશન્સ**: સંયોજન, છેદ
- **સભ્યપદ પરીક્ષા**: ઝડપી lookup

સ્મરણ સૂત્ર: ``અક્રમ બદલાવપાત્ર અનન્ય પુનરાવર્તનીય''

પ્રશ્ન 4(અ) [3 ગુણ]

પાયથનમાં ક્લાસ શું છે? તે ઓબ્જેક્ટથી કઈ રીતે અલગ છે?

જવાબ

ક્લાસ વિ. ઓબ્જેક્ટ સરખામણી:

પાસું	ક્લાસ	ઓબ્જેક્ટ
વ્યાખ્યા	Blueprint અથવા template	ક્લાસનું instance
મેમરી	કોઈ memory allocate થતી નથી	Memory allocate થાય છે
અસ્તિત્વ	Logical entity	Physical entity
બનાવટ	class keyword ઉપયોગ કરીને	Class constructor ઉપયોગ કરીને

ઉદાહરણ:

```
\# Class definition (blueprint)
class Car:
    def \_\_init\_\_(self, brand):
        self.brand = brand

\# Object creation (instances)
car1 = Car("Toyota")  \# Object 1
car2 = Car("Honda")   \# Object 2
```

મુખ્ય મુદ્દાઓ:
- **ક્લાસ**: Properties અને methods વ્યાખ્યાયિત કરતું template
- **ઓબ્જેક્ટ**: Specific values સાથેનું actual instance
- **સંબંધ**: એક ક્લાસ, અનેક ઓબ્જેક્ટ્સ

સ્મરણ સૂત્ર: ``ક્લાસ Blueprint, ઓબ્જેક્ટ Instance''

પ્રશ્ન 4(બ) [4 ગુણ]

dictionary ડેટા ટાઈપની કોઈ પણ 4 મેથડ સમજાવો.

જવાબ

Dictionary મેથડ્સ ટેબલ:

મેથડ	Syntax	વર્ણન
keys()	`dict.keys()`	બધી keys મેળવો
values()	`dict.values()`	બધી values મેળવો
items()	`dict.items()`	Key-value pairs મેળવો
get()	`dict.get('key')`	Value સુરક્ષિત રીતે મેળવો

કોડ ઉદાહરણ:

```
student = \{{name}: {John}, {age}: 20, {grade}: {A}\}

\# Dictionary methods
print(student.keys())    \# dict\_keys([{name, age, grade])}
print(student.values())  \# dict\_values([{John, 20, A])}
print(student.items())   \# dict\_items([({name, John), ...])}
print(student.get({name}))  \# John
```

સ્મરણ સૂત્ર: ``Keys Values Items Get''

પ્રશ્ન 4(ક) [7 ગુણ]

કોઈ કાર્યો કરવા માટે યુઝર ડિફાઈન્ડ મોડ્યૂલ બનાવી તેને ઈમ્પોર્ટ કરી તેના ફંક્શનનો ઉપયોગ કરવાનો પાયથન પ્રોગ્રામ લખો.

જવાબ

મોડ્યૂલ બનાવવું (math_operations.py):

```
\# math\_operations.py
def add(a, b):
    """Add two numbers"""
    return a + b

def multiply(a, b):
    """Multiply two numbers"""
    return a * b
```

```
def factorial(n):
    """Calculate factorial"""
    if n {=} 1:
        return 1
    return n * factorial(n {-} 1)

PI = 3.14159

def circle\_area(radius):
    """Calculate circle area"""
    return PI * radius * radius
```

મુખ્ય પ્રોગ્રામ (main.py):

```
\# Import entire module
import math\_operations

\# Use module functions
result1 = math\_operations.add(5, 3)
result2 = math\_operations.multiply(4, 6)
result3 = math\_operations.factorial(5)
area = math\_operations.circle\_area(5)

print(f"Addition: \{result1\}")
print(f"Multiplication: \{result2\}")
print(f"Factorial: \{result3\}")
print(f"Circle Area: \{area\}")

\# Import specific functions
from math\_operations import add, multiply
print(f"Direct call: \{add(10, 20)\}")
```

મુખ્ય મુદ્દાઓ:
- **મોડ્યૂલ બનાવવું**: Functions સાથે અલગ .py ફાઈલ
- **Import પદ્ધતિઓ**: import module અથવા from module import function
- **ઉપયોગ**: module.function() અથવા direct function() વડે access

સ્મરણ સૂત્ર: ``બનાવો Import ઉપયોગ''

પ્રશ્ન 4(અ OR) [3 ગુણ]

પાયથન ક્લાસની મેથડ્સના પ્રકારો ટૂંકમાં સમજાવો.

જવાબ

મેથડ્સના પ્રકારોનું ટેબલ:

મેથડ પ્રકાર	Syntax	વર્ણન
Instance Method	`def method(self):`	Instance variables ને access કરે છે
Class Method	`@classmethod def method(cls):`	Class variables ને access કરે છે
Static Method	`@staticmethod def method():`	Class/instance થી સ્વતંત્ર

ઉદાહરણ:

```
class MyClass:
    class\_var = "Class Variable"

    def instance\_method(self):  \# Instance method
        return "Instance method"

    @classmethod
    def class\_method(cls):      \# Class method
        return cls.class\_var

    @staticmethod
    def static\_method():        \# Static method
        return "Static method"
```

સ્મરણ સૂત્ર: ``Instance Self, Class Cls, Static કંઈ નહીં''

પ્રશ્ન 4(બ OR) [4 ગુણ]

string ડેટા ટાઈપની કોઈ પણ 4 મેથડ સમજાવો.

જવાબ

String મેથડ્સ ટેબલ:

મેથડ	Syntax	વર્ણન
startswith()	`str.startswith('pre')`	Substring થી શરૂ થાય છે કે ચેક કરો
endswith()	`str.endswith('suf')`	Substring થી અંત થાય છે કે ચેક કરો
isdigit()	`str.isdigit()`	બધા digits છે કે ચેક કરો
count()	`str.count('sub')`	Substring ની occurrences ગિનો

કોડ ઉદાહરણ:

```
text = "Hello World 123"

\# String methods
print(text.startswith({Hello}))  \# True
print(text.endswith({123}))      \# True
print({123}.isdigit())           \# True
print(text.count({l}))           \# 3
```

સ્મરણ સૂત્ર: ``Start End Digit Count''

પ્રશ્ન 4(ક OR) [7 ગુણ]

રિકર્સીવ ફંક્શનની મદદથી આપેલ નંબરનો ફેક્ટોરીયલ શોધવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
def factorial(n):
    """
    Recursion   factorial
    Base case: factorial(0) = 1, factorial(1) = 1
    Recursive case: factorial(n) = n * factorial(n{-1)}
    """
    \# Base case
if
```

```
n == 0 or

n == 1:

        return 1

    \# Recursive case
    else:
        return n * factorial(n {-} 1)

\# Main program
try:
    num = int(input("Enter a number: "))

    if num {} 0:
        print("Factorial not defined for negative numbers")
    else:
        result = factorial(num)
        print(f"Factorial of \{num\} is \{result\}")

except ValueError:
    print("Please enter a valid integer")

\# Test cases
print(f"Factorial of 5: \{factorial(5)\}")  \# 120
print(f"Factorial of 0: \{factorial(0)\}")  \# 1
```

Recursion Flow:

```
factorial(5)
    |
5 * factorial(4)
        |
    4 * factorial(3)
            |
        3 * factorial(2)
                |
            2 * factorial(1)
                    |
                return 1

Result: 5 * 4 * 3 * 2 * 1 = 120
```

મુખ્ય મુદ્દાઓ:
- **Base case**: Recursion બંધ કરે છે (n=0 અથવા n=1)
- **Recursive case**: Function પોતાને call કરે છે
- **Error handling**: Negative input માટે ચેક કરો

સ્મરણ સૂત્ર: ``Base બંધ, Recursive Call, Error ચેક''

પ્રશ્ન 5(અ) [3 ગુણ]

સિંગલ ઇન્હેરિટન્સ બતાવવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

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

```
    def speak(self):
        print(f"\{self.name\} makes a sound")

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

\# Child class inheriting from Animal
class Dog(Animal):
    def \_\_init\_\_(self, name, breed):
        super().\_\_init\_\_(name)  \# Call parent constructor
        self.breed = breed

    def bark(self):
        print(f"\{self.name\} is barking")

    def speak(self):  \# Override parent method
        print(f"\{self.name\} says Woof!")

\# Create objects and test
dog = Dog("Buddy", "Golden Retriever")
dog.speak()  \# Buddy says Woof!
dog.eat()    \# Buddy is eating (inherited)
dog.bark()   \# Buddy is barking (own method)
```

સ્મરણ સૂત્ર: ``Parent Child Inherit Override''

પ્રશ્ન 5(બ) [4 ગુણ]

પાયથન ક્લાસમાં કન્સ્ટ્રક્ટરનું મહત્વ સમજાવો.

જવાબ

કન્સ્ટ્રક્ટરનું મહત્વ:

પાસું	વર્ણન
ઇનિશિયલાઇઝેશન	ઓબ્જેક્ટ બનાવવામાં આવે ત્યારે આપોઆપ call થાય છે
સેટઅપ	Instance variables ને values સાથે initialize કરે છે
મેમરી	Object attributes માટે memory allocate કરે છે
વેલિડેશન	Creation દરમિયાન input parameters validate કરે છે

કન્સ્ટ્રક્ટરના પ્રકારો:

```
class Student:
    \# Default constructor
    def \_\_init\_\_(self):
        self.name = "Unknown"
        self.age = 0

    \# Parameterized constructor
    def \_\_init\_\_(self, name, age):
        self.name = name
        self.age = age
        print(f"Student \{name\} created")

    \# Constructor with default parameters
    def \_\_init\_\_(self, name="Unknown", age=0):
        self.name = name
        self.age = age
```

મુખ્ય ફાયદાઓ:
- **આપોઆપ execution**: Manual call કરવાની જરૂર નથી
- **ઓબ્જેક્ટ state**: યોગ્ય initialization ensure કરે છે
- **કોડ પુનઃઉપયોગ**: એક જગ્યાએ common setup code

સ્મરણ સૂત્ર: ``Initialize Setup Memory Validate''

પ્રશ્ન 5(ક) [7 ગુણ]

ઇન્હેરિટન્સ દ્વારા થતું મેથડ ઓવરરાઇડિંગ બતાવવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# Base class
class Shape:
    def \_\_init\_\_(self, name):
        self.name = name

    def area(self):
        print(f"Area calculation for \{self.name\}")
        return 0

    def display(self):
        print(f"This is a \{self.name\}")

\# Derived class 1
class Rectangle(Shape):
    def \_\_init\_\_(self, length, width):
        super().\_\_init\_\_("Rectangle")
        self.length = length
        self.width = width

    \# Override area method
    def area(self):
        area\_value = self.length * self.width
        print(f"Rectangle area: \{area\_value\}")
        return area\_value

\# Derived class 2
class Circle(Shape):
    def \_\_init\_\_(self, radius):
```

```
        super().\_\_init\_\_("Circle")
        self.radius = radius

    \# Override area method
    def area(self):
        area\_value = 3.14 * self.radius * self.radius
        print(f"Circle area: \{area\_value\}")
        return area\_value

    \# Override display method
    def display(self):
        super().display()  \# Call parent method
        print(f"Radius: \{self.radius\}")

\# Test method overriding
shapes = [
    Rectangle(5, 4),
    Circle(3),
    Shape("Generic Shape")
]

for shape in shapes:
    shape.display()
    shape.area()
    print("{-"} * 20)
```

મેથડ ઓવરરાઇડિંગ ડાયાગ્રામ:

```
    Shape (Base)
    |{-{-} area()}
    |{-{-} display()}
         |
    Rectangle    Circle
    |{-{-} area()   |{-}{-} area()}
                 |{-{-} display()}
```

મુખ્ય મુદ્દાઓ:

- **સમાન મેથડ નામ**: Parent અને child classes માં
- **અલગ implementation**: Child class specific logic આપે છે
- **Runtime નિર્ણય**: Object type આધારે યોગ્ય method call થાય છે
- **Super() ઉપયોગ**: Parent class method ને access કરવા માટે

સ્મરણ સૂત્ર: ``સમાન નામ અલગ તર્ક Runtime નિર્ણય''

પ્રશ્ન 5(અ OR) [3 ગુણ]

પાયથનમાં ડેટા એન્કેપ્સ્યુલેશનનો ખ્યાલ સમજાવો.

જવાબ

ડેટા એન્કેપ્સ્યુલેશન:

પાસું	વર્ણન
વ્યાખ્યા	Data અને methods ને એકસાથે બાંધવું
એક્સેસ કન્ટ્રોલ	Internal data ને direct access પર પ્રતિબંધ
ડેટા છુપાવવું	Internal implementation બહારથી છુપાવવું
ઇન્ટરફેસ	Methods દ્વારા controlled access પ્રદાન કરવું

અમલીકરણ:

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

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

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

    def \_\_validate(self):         \# Private method
        return self.\_\_balance {=} 0

\# Usage
account = BankAccount(1000)
account.deposit(500)
print(account.get\_balance())  \# 1500
\# print(account.\_\_balance)    \# Error {- cannot access private}
```

સ્મરણ સૂત્ર: ``બાંધો ડેટા છુપાવો ઇન્ટરફેસ''

પ્રશ્ન 5(બ OR) [4 ગુણ]

પાયથનમાં એબ્સ્ટ્રેક્ટ ક્લાસનો ખ્યાલ સમજાવો.

જવાબ

એબ્સ્ટ્રેક્ટ ક્લાસ:

કન્સેપ્ટ	વર્ણન
વ્યાખ્યા	સીધા instantiate ન થઈ શકતો ક્લાસ
એબ્સ્ટ્રેક્ટ મેથડ્સ	Declared પણ implemented નથી
અમલીકરણ	Subclasses એ abstract methods implement કરવા જોઈએ
હેતુ	Related classes માટે common interface વ્યાખ્યાયિત કરવો

ABC વડે અમલીકરણ:

```
from abc import ABC, abstractmethod

class Animal(ABC):  \# Abstract class
    @abstractmethod
    def make\_sound(self):  \# Abstract method
        pass

    def sleep(self):       \# Concrete method
        print("Animal is sleeping")

class Dog(Animal):
    def make\_sound(self):  \# Must implement
        print("Woof!")

class Cat(Animal):
    def make\_sound(self):  \# Must implement
        print("Meow!")

\# Usage
dog = Dog()
dog.make\_sound()  \# Woof!
\# animal = Animal()  \# Error {- cannot instantiate}
```

મુખ્ય વિશેષતાઓ:
- **Instantiate ન થઈ શકે**: Abstract class objects બનાવી શકાતા નથી
- **અમલીકરણ દબાણ**: Subclasses એ abstract methods implement કરવા જોઈએ
- **કોમન ઇન્ટરફેસ**: સુસંગત method signatures ensure કરે છે

સ્મરણ સૂત્ર: ``Instantiate ન થાય અમલીકરણ દબાણ કોમન ઇન્ટરફેસ''

પ્રશ્ન 5(ક OR) [7 ગુણ]

મલ્ટિપલ ઇન્હેરિટન્સ બતાવવા માટેનો પાયથન પ્રોગ્રામ લખો.

જવાબ

કોડ:

```
\# First parent class
class Father:
    def \_\_init\_\_(self):
        self.father\_name = "John"
        print("Father constructor called")

    def show\_father(self):
        print(f"Father: \{self.father\_name\}")

    def work(self):
        print("Father works as Engineer")

\# Second parent class
class Mother:
    def \_\_init\_\_(self):
        self.mother\_name = "Mary"
        print("Mother constructor called")

    def show\_mother(self):
        print(f"Mother: \{self.mother\_name\}")

    def work(self):
```

```
        print("Mother works as Doctor")

\# Child class inheriting from both parents
class Child(Father, Mother):
    def \_\_init\_\_(self):
        Father.\_\_init\_\_(self)  \# Call father{s constructor}
        Mother.\_\_init\_\_(self)  \# Call mother{s constructor}
        self.child\_name = "Alice"
        print("Child constructor called")

    def show\_child(self):
        print(f"Child: \{self.child\_name\}")

    def show\_family(self):
        self.show\_father()
        self.show\_mother()
        self.show\_child()

\# Create child object and test
child = Child()
print("{n}Family Details:")
child.show\_family()
print("{n}Method Resolution:")
child.work()  \# Calls Father{s work method (MRO)}

\# Check Method Resolution Order
print(f"{n}MRO: \{Child.\_\_mro\_\_\}")
```

મલ્ટિપલ ઇન્હેરિટન્સ ડાયાગ્રામ:

```
    Father        Mother
    |              |
    |              |
    +{-{-}{-}{-}{-}{-}+{-}{-}{-}{-}{-}{-}{-}+}
           |
         Child
```

મુખ્ય મુદ્દાઓ:

- **અનેક પેરેન્ટ્સ**: Child બંને Father અને Mother થી inherit કરે છે
- **મેથડ રિઝોલ્યુશન ઓર્ડર (MRO)**: કયો method call થશે તે નક્કી કરે છે
- **કન્સ્ટ્રક્ટર કોલ્સ**: Parent constructors ને સ્પષ્ટપણે call કરવા
- **ડાયમંડ પ્રોબ્લેમ**: Python MRO વડે handle કરે છે

આઉટપુટ:

```
Father constructor called
Mother constructor called
Child constructor called

Family Details:
Father: John
Mother: Mary
Child: Alice

Method Resolution:
Father works as Engineer
```

સ્મરણ સૂત્ર: ``અનેક પેરેન્ટ્સ MRO કન્સ્ટ્રક્ટર ડાયમંડ''