Subject Name (Gujarati)

1323203 -- Winter 2023

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(a) [3 ગુણ]

આપેલ નંબર પોઝિટિવ છે કે નેગેટિવ તે તપાસવા માટે સ્યૂડો કોડ લખો

જવાબ

```
BEGIN
    Input number
    IF number > 0 THEN
        Display "Number is positive"
    ELSE IF number < 0 THEN
        Display "Number is negative"
    ELSE
        Display "Number is zero"
    END IF
END
```

મેમરી ટ્રીક

``શૂન્ય સાથે સરખાવો''

પ્રશ્ન 1(b) [4 ગુણ]

એલ્ગોરિધમ વ્યાખ્યાયિત કરો અને ત્રણ નંબર માંથી મહત્તમ નંબર શોધવાનો એલ્ગોરિધમ બનાવો.

જવાબ

Algorithm વ્યાખ્યા: એલ્ગોરિધમ એટલે ચોક્કસ સમસ્યાને ઉકેલવા માટે અથવા ગણતરી કરવા માટે બનાવેલ સ્ટેપ-બાય-સ્ટેપ પ્રક્રિયા અથવા નિયમોનો સેટ.

ત્રણ નંબરમાંથી મહત્તમ શોધવાનો એલ્ગોરિધમ:

```
BEGIN
    Input num1, num2, num3
    Set max = num1
    IF num2 > max THEN
        Set max = num2
    END IF
    IF num3 > max THEN
        Set max = num3
    END IF
    Display max
END
```

ડાયાગ્રામ:

```
flowchart LR
    A[Start] {-{-} B[Input num1, num2, num3]}
    B {-{-} C[Set max = num1]}
    C {-{-} D\{Is num2  max?\}}
    D {-{-} Yes {-}{-} E[Set max = num2]}
    D {-{-} No {-}{-} F\{Is num3  max?\}}
    E {-{-} F}
    F {-{-} Yes {-}{-} G[Set max = num3]}
    F {-{-} No {-}{-} H[Display max]}
```

```
    G {-{-} H}
    H {-{-} I[End]}
```

મેમરી ટ્રીક

``સરખામણી અને બદલો''

પ્રશ્ન 1(c) [7 ગુણ]

તાપમાન ના સેલ્સિયસ ને ફેરનહાઇટ માં કન્વર્ટ કરવાનો પાયથોન કોડ લખો.

જવાબ

```
\#

\#
celsius = float(input("              : "))

\#                    : F = (C * 9/5) + 32
fahrenheit = (celsius * 9/5) + 32

\#
print(f"\{celsius\}^   \{fahrenheit\}^       ")
```

ટેબલ: તાપમાન રૂપાંતરણ:

ઘટક	વર્ણન
ઇનપુટ	સેલ્સિયસમાં તાપમાન
સૂત્ર	F = (C $\times 9/5) + 32$
આઉટપુટ	ફેરનહાઇટમાં તાપમાન

મેમરી ટ્રીક

``9થી ગુણાકાર, 5થી ભાગાકાર, 32 ઉમેરો''

પ્રશ્ન 1(c OR) [7 ગુણ]

કંપેરિઝન ઓપરેટર નું લિસ્ટ આપો અને દરેકને પાયથોન કોડના ઉદાહરણ સાથે સમજાવો.

જવાબ

ટેબલ: પાયથોન કંપેરિઝન ઓપરેટર્સ

ઓપરેટર	વર્ણન	ઉદાહરણ	પરિણામ
==	બરાબર છે	`5 == 5`	`True`
!=	બરાબર નથી	`5 != 6`	`True`
>	કરતાં મોટું	`6 > 3`	`True`
<	કરતાં નાનું	`3 < 6`	`True`
>=	કરતાં મોટું અથવા બરાબર	`5 >= 5`	`True`
<=	કરતાં નાનું અથવા બરાબર	`5 <= 5`	`True`

કોડ ઉદાહરણ:

```
\#
a = 10
b = 5

\#
print(f"\{a\} == \{b\}: \{a == b\}")  \# False

\#
print(f"\{a\} != \{b\}: \{a != b\}")  \# True

\#
print(f"\{a\} { }\{b\}: \{a {} b\}")    \# True

\#
print(f"\{a\} { }\{b\}: \{a {} b\}")    \# False

\#
print(f"\{a\} {= }\{b\}: \{a {=} b\}")  \# True

\#
print(f"\{a\} {= }\{b\}: \{a {=} b\}")  \# False
```

મેમરી ટ્રીક

``સરખાવો'' (સમાન, રિલેશનલ, ખાસ સરખામણી, અસમાનતા, વધુ ઓછું)

પ્રશ્ન 2(a) [3 ગુણ]

પાયથોન ના ડેટા ટાઇપ સમજાવો.

જવાબ

ટેબલ: પાયથોન ડેટા ટાઇપ્સ

ડેટા ટાઇપ	વર્ણન	ઉદાહરણ
int	પૂર્ણાંક મૂલ્યો	`x = 10`
float	દશાંશ બિંદુ મૂલ્યો	`y = 10.5`
str	ટેક્સ્ટ અથવા અક્ષર મૂલ્યો	`name = "Python"`
bool	તાર્કિક મૂલ્યો (True/False)	`is_valid = True`
list	ક્રમબદ્ધ, બદલી શકાય તેવો સંગ્રહ	`nums = [1, 2, 3]`
tuple	ક્રમબદ્ધ, ન બદલી શકાય તેવો સંગ્રહ	`point = (5, 10)`
dict	કી-વેલ્યુ જોડી	`student = {"name": "John"}`

મેમરી ટ્રીક

``NIFTY SLD'' (નંબર્સ, ઇન્ટીજર્સ, ફ્લોટ્સ, ટેક્સ્ટ, યસ/નો, સીક્વન્સીસ, લિસ્ટ્સ, ડિક્શનરીઝ)

પ્રશ્ન 2(b) [4 ગુણ]

Nested If પાયથોન કોડ ના ઉદાહરણ સાથે સમજાવો.

જવાબ

Nested if: એક conditional statement ની અંદર બીજું conditional statement લખવાને nested if કહેવામાં આવે છે. તે ઘણી શરતોને ક્રમમાં તપાસવાની મંજૂરી આપે છે.

```
\#        ,                    nested if
\#             ,

num = int(input("          : "))

if num {} 0:
    print("       ")
    \# nested if
    if num \% 2 == 0:
        print("    ")
    else:
        print("     ")
elif num {} 0:
    print("      ")
else:
    print("  ")
```

ડાયાગ્રામ:

```
flowchart LR
    A[Start] {-{-} B[Input num]}
    B {-{-} C\{Is num  0?\}}
    C {-{-} Yes {-}{-} D[Print Positive number]}
    D {-{-} E\{Is num \% 2 == 0?\}}
    E {-{-} Yes {-}{-} F[Print Even number]}
    E {-{-} No {-}{-} G[Print Odd number]}
    C {-{-} No {-}{-} H\{Is num  0?\}}
    H {-{-} Yes {-}{-} I[Print Negative number]}
    H {-{-} No {-}{-} J[Print Zero]}
    F {-{-} K[End]}
    G {-{-} K}
    I {-{-} K}
    J {-{-} K}
```

મેમરી ટ્રીક

``ચેક અંદર ચેક''

પ્રશ્ન 2(c) [7 ગુણ]

ઉદાહરણ સાથે વિવિધ પ્રકારના પસંદગી/નિર્ણય લેવાના ફ્લો-ઓફ-કંટ્રોલ સ્ટ્રક્ચર ઉપયોગ સમજાવો

જવાબ

ટેબલ: પાયથોનમાં સિલેક્શન કંટ્રોલ સ્ટ્રક્ચર્સ

સ્ટ્રક્ચર	હેતુ	વપરાશ
if	શરત સાચી હોય ત્યારે કોડ ચલાવવા	સરળ શરત ચકાસણી
if-else	સાચી શરત માટે એક કોડ, ખોટી માટે બીજો	દ્વિ નિર્ણય લેવા
if-elif-else	ઘણી શરતો ચકાસવી	ઘણા સંભવિત પરિણામો
Nested if	શરત અંદર બીજી શરત	જટિલ શ્રેણીબદ્ધ નિર્ણયો
Ternary operator	એક લાઇન if-else	સરળ શરતી નિયુક્તિ

કોડ ઉદાહરણ:

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

\#    if
if score {=} 90:
    print("  !")

\# if{-else}
if score {=} 60:
    print("         .")
else:
    print("          .")

\# if{-elif{-}else}
if score {=} 90:
    grade = "A"
elif score {=} 80:
    grade = "B"
elif score {=} 70:
    grade = "C"
elif score {=} 60:
    grade = "D"
else:
    grade = "F"
print(f"       \{grade\}  ")

\# Ternary operator
result = "  " if score {=} 60 else "   "
print(result)
```

મેમરી ટ્રીક

``SCENE'' (સિમ્પલ if, કન્ડિશન્સ વિથ else, Elif ફોર મલ્ટિપલ, Nested ફોર કોમ્પ્લેક્સ, એક્સપ્રેસ વિથ ટર્નરી)

પ્રશ્ન 2(a) [3 ગુણ] - OR ઓપ્શન

વેરિયેબલ વ્યાખ્યાયિત કરવાના નિયમો લિસ્ટ કરો.

જવાબ

ટેબલ: પાયથોનમાં વેરિએબલ્સ વ્યાખ્યાયિત કરવાના નિયમો

નિયમ	વર્ણન	ઉદાહરણ
અક્ષર અથવા અન્ડરસ્કોરથી શરૂ કરો	પ્રથમ અક્ષર એક લેટર અથવા અન્ડરસ્કોર હોવો જોઈએ	`name = "John"`, `_count = 10`
કોઈ ખાસ અક્ષરો નહીં	માત્ર અક્ષરો, અંકો અને અન્ડરસ્કોર માન્ય	`user_name` (માન્ય), `user-name` (અમાન્ય)
કેસ સેન્સિટિવ	મોટા અક્ષરો અને નાના અક્ષરો અલગ	`age` અને `Age` અલગ વેરિએબલ્સ છે
રિઝર્વ્ડ કીવર્ડ્સ નહીં	પાયથોન કીવર્ડ્સને વેરિએબલ નામ તરીકે ઉપયોગ ન કરી શકાય	`if`, `for`, `while`, વગેરે ઉપયોગ ન કરી શકાય
સ્પેસ નહીં	સ્પેસને બદલે અન્ડરસ્કોર વાપરો	`first_name` (`first name` નહીં)

મેમરી ટ્રીક

``SILKS'' (શરૂઆત યોગ્ય રીતે, ઇગ્નોર સ્પેશિયલ કેરેક્ટર, લૂક એટ કેસ, કીવર્ડ્સ અવોઇડ, સ્પેસ નોટ અલાઉડ)

પ્રશ્ન 2(b) [4 ગુણ] - OR ઓપ્શન

ફોર લૂપ ને જરૂરી ઉદાહરણ સાથે સમજાવો.

જવાબ

પાયથોનમાં For Loop: for લૂપનો ઉપયોગ કોઈ sequence (લિસ્ટ, ટપલ, સ્ટ્રીંગ) અથવા અન્ય iterable ઓબ્જેક્ટ પર પુનરાવર્તન કરવા માટે થાય છે. તે sequence ના દરેક આઇટમ માટે કોડનો એક બ્લોક ચલાવે છે.

```
\#      for
\#
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)

\# range     for
print("1   5      :")
for i in range(1, 6):
    print(i)

\#       for
name = "Python"
for char in name:
    print(char)
```

ડાયાગ્રામ:

```
flowchart LR
    A[Start] {-{-} B[Initialize sequence]}
    B {-{-} C[Get first item]}
    C {-{-} D[Execute code block]}
    D {-{-} E\{More items?\}}
    E {-{-} Yes {-}{-} F[Get next item]}
    F {-{-} D}
    E {-{-} No {-}{-} G[End]}
```

મેમરી ટ્રીક

``ITEM'' (Iterate Through Each Member) - દરેક સભ્ય પર પુનરાવર્તન કરો

પ્રશ્ન 2(c) [7 ગુણ] - OR ઓપ્શન

Break અને continue સ્ટેટમેન્ટને સંક્ષિપ્તમાં સમજાવો.

જવાબ

ટેબલ: Break અને Continue સ્ટેટમેન્ટ્સ

સ્ટેટમેન્ટ	હેતુ	અસર
break	લૂપમાંથી તરત જ બહાર નીકળો	વર્તમાન લૂપને અટકાવે છે અને લૂપ પછીના સ્ટેટમેન્ટ પર કંટ્રોલ ટ્રાન્સફર કરે છે
continue	વર્તમાન પુનરાવર્તન છોડી દો	લૂપના આગલા પુનરાવર્તન પર જાય છે, continue સ્ટેટમેન્ટ પછીના કોઈપણ કોડને છોડી દે છે

કોડ ઉદાહરણ:

```
\# Break
print("Break    :")
for i in range(1, 11):
if

i == 6:

        print("i =", i, "          ")
        break
    print(i, end=" ")
print("{n}       ")

\# Continue
print("{n}Continue     :")
for i in range(1, 11):
    if i \% 2 == 0:
        continue
    print(i, end=" ")
print("{n}               ")
```

ડાયાગ્રામ:

```
flowchart LR
    A[Start Loop] {-{-} B\{Condition met for break?\}}
    B {-{-} Yes {-}{-} C[Exit Loop]}
    B {-{-} No {-}{-} D\{Condition met for continue?\}}
    D {-{-} Yes {-}{-} E[Skip to next iteration]}
    E {-{-} A}
    D {-{-} No {-}{-} F[Execute remaining code in loop body]}
    F {-{-} A}
    C {-{-} G[Continue execution after loop]}
```

મેમરી ટ્રીક

``EXIT SKIP'' (EXIT with break, SKIP with continue) - બ્રેક સાથે બહાર નીકળો, કન્ટિન્યુ સાથે છોડી દો

પ્રશ્ન 3(a) [3 ગુણ]

1 થી 10 નંબર ને લૂપથી પ્રિન્ટ કરવા માટેનો પાયથન કોડ બનાવો.

જવાબ

```
\# 1   10              for
print("for          :")
for i in range(1, 11):
    print(i, end=" ")

print("{nn}while          :")
\# 1   10              while
counter = 1
while counter {=} 10:
    print(counter, end=" ")
    counter += 1
```

ટેબલ: લૂપ અભિગમ

અભિગમ	ફાયદો
range સાથે For લૂપ	સરળ, સંક્ષિપ્ત, આપોઆપ કાઉન્ટર મેનેજ કરે છે

While લૂપ	જટિલ શરતો માટે વધુ લવચીક

મેમરી ટ્રીક

``COUNT UP'' (Counter દરેક પુનરાવર્તનમાં અપડેટ થાય છે)

પ્રશ્ન 3(b) [4 ગુણ]

નીચેની પેટર્ન પ્રિન્ટ કરવા માટેનો પાયથન કોડ લખો.

```
*
**
***
****
*****
```

જવાબ

```
\# for
rows = 5

for i in range(1, rows + 1):
    \#        i
    print("*" * i)
```

વૈકલ્પિક ઉકેલ નેસ્ટેડ લૂપ્સ સાથે:

```
\#
rows = 5

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

ડાયાગ્રામ:

```
flowchart LR
    A[Start] {-{-} B[Set rows = 5]}
    B {-{-} C[Initialize i = 1]}
    C {-{-} D\{"Is i = rows?"\}}
    D {-{-} Yes {-}{-} E["Print * * i"]}
    E {-{-} F[Increment i]}
    F {-{-} D}
    D {-{-} No {-}{-} G[End]}
```

મેમરી ટ્રીક

``RISE UP'' (Row Increases, Stars Expand Upward Progressively) - રો વધે છે, સ્ટાર ઊપર તરફ વિસ્તરે છે

પ્રશ્ન 3(c) [7 ગુણ]

આપેલા નંબર નો factorial શોધવા માટેનું યુઝર ડિફાઇન ફંક્શન બનાવો.

જવાબ

```
\#
def factorial(n):
    \#
```

```
    if not isinstance(n, int) or n {} 0:
        return "      .        {-            ."}

    \#    : 0    1        1
if

n == 0 or

n == 1:

        return 1

    \#
    result = 1
    for i in range(2, n + 1):
        result *= i

    return result

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

ડાયાગ્રામ:

```
flowchart LR
    A[Start] {-{-} B[Define factorial function]}
    B {-{-} C[Check if n is valid]}
    C {-{-} Invalid {-}{-} D[Return error message]}
    C {-{-} Valid {-}{-} E\{Is n 0 or 1?\}}
    E {-{-} Yes {-}{-} F[Return 1]}
    E {-{-} No {-}{-} G[Set result = 1]}
    G {-{-} H[Loop from 2 to n]}
    H {-{-} I[result = result * i]}
    I {-{-} J[Return result]}
    J {-{-} K[End]}
```

ટેબલ: ફેક્ટોરિયલ ઉદાહરણો

નંબર	ગણતરી	ફેક્ટોરિયલ
0	0! = 1	1
1	1! = 1	1
3	3! = 3 $\times 2 \times 1$	6
5	5! = 5 $\times 4 \times 3 \times 2 \times 1$	120

મેમરી ટ્રીક

``1 સુધી ગુણાકાર કરો'' (બધા આંકડાને 1 સુધી ગુણાકાર કરો)

પ્રશ્ન 3(a) [3 ગુણ] - OR ઓપ્શન

1 થી N માંથી odd અને even નંબર શોધવાનો પાયથન કોડ બનાવો.

જવાબ

```
\# 1   N     odd    even

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

```
print("1  ", N, "    even    :")
for i in range(1, N + 1):
    if i \% 2 == 0:
        print(i, end=" ")

print("{n}1  ", N, "    odd    :")
for i in range(1, N + 1):
    if i \% 2 != 0:
        print(i, end=" ")
```

ટેબલ: Even અને Odd ચેક

નંબર	ચેક	પ્રકાર
Even નંબર	`number % 2 == 0`	2, 4, 6, ...
Odd નંબર	`number % 2 != 0`	1, 3, 5, ...

મેમરી ટ્રીક

``MOD-2'' (Modulo 2 જે even કે odd નક્કી કરે છે)

પ્રશ્ન 3(b) [4 ગુણ] - OR ઓપ્શન

Nested લિસ્ટ અને તેના એલિમેન્ટ ડિસ્પ્લે કરવા માટેનો પાયથન કોડ બનાવો.

જવાબ

```
\# Nested

\# Nested
nested\_list = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

\# Nested
print("Nested   :", nested\_list)

\# Nested
print("{n}Nested         :")
for i in range(len(nested\_list)):
    for j in range(len(nested\_list[i])):
        print(f"nested\_list[\{i\}][\{j\}] = \{nested\_list[i][j]\}")

\# enumerate
print("{n}enumerate         :")
for i, inner\_list in enumerate(nested\_list):
    for j, value in enumerate(inner\_list):
        print(f"    (\{i\}, \{j\}): \{value\}")
```

ડાયાગ્રામ:

```
flowchart TD
    A[Nested List] {-{-} B[Row 0]}
    A {-{-} C[Row 1]}
    A {-{-} D[Row 2]}
    B {-{-} B1[1]}
    B {-{-} B2[2]}
    B {-{-} B3[3]}
    C {-{-} C1[4]}
    C {-{-} C2[5]}
```

```
    C {-{-} C3[6]}
    D {-{-} D1[7]}
    D {-{-} D2[8]}
    D {-{-} D3[9]}
```

મેમરી ટ્રીક

``ROWS COLS'' (રો અને કોલમ માળખું બનાવે છે)

પ્રશ્ન 3(c) [7 ગુણ] - OR ઓપ્શન

Local અને Global વેરિયેબલ ઉદાહરણ સાથે સમજાવો.

જવાબ

ટેબલ: Local vs Global વેરિએબલ્સ

પ્રકાર	સ્કોપ	એક્સેસિબિલિટી	ઘોષણા
Local વેરિએબલ્સ	માત્ર જે ફંક્શનમાં ઘોષિત થયા છે ત્યાં	માત્ર ઘોષિત કરનાર ફંક્શનની અંદર	ફંક્શનની અંદર
Global વેરિએબલ્સ	સમગ્ર પ્રોગ્રામમાં	બધા ફંક્શન એક્સેસ કરી શકે	કોઈપણ ફંક્શનની બહાર

કોડ ઉદાહરણ:

```
\# Global
total = 0

def add\_numbers(a, b):
    \# Local
    sum\_result = a + b
    print(f"Local      sum\_result: \{sum\_result\}")

    \# Global
    print(f"Global     total          : \{total\}")

    \#        Global
    global total
    total = sum\_result
    print(f"Global     total         : \{total\}")

    return sum\_result

\#
x = 5  \#          Local
y = 10  \#          Local

result = add\_numbers(x, y)
print(f"    : \{result\}")
print(f"    global total: \{total\}")

\#            sum\_result   add\_numbers    Local
\# print(sum\_result)  \# NameError: name {sum\_result is not defined}
```

ડાયાગ્રામ:

```
flowchart LR
    A[Program Scope] {-{-} B[Global Variables: total]}
    A {-{-} C[Function Scope: add\_numbers]}
    C {-{-} D[Local Variables: sum\_result, a, b]}
    A {-{-} E[Main Program Variables: x, y, result]}
    B {{-}.{-} C}
    D {-.{-} C}
    E {-.{-} A}
```

મેમરી ટ્રીક

``GLOBAL SEES ALL'' (Global વેરિએબલ્સ બધે જોઈ શકે છે)

પ્રશ્ન 4(a) [3 ગુણ]

પાયથન ની સ્ટાન્ડર્ડ લાઇબ્રેરી ના મેથેમેટિકલ ફંક્શન લિસ્ટ કરો.

જવાબ

ટેબલ: પાયથોન Math મોડ્યુલ ફંક્શન્સ

ફંક્શન	વર્ણન	ઉદાહરણ
abs()	એબ્સોલ્યુટ વેલ્યુ આપે છે	`abs(-5)` → 5
pow()	x ને y ની ઘાત આપે છે	`pow(2, 3)` → 8
max()	સૌથી મોટી વેલ્યુ આપે છે	`max(5, 10, 15)` → 15
min()	સૌથી નાની વેલ્યુ આપે છે	`min(5, 10, 15)` → 5
round()	નજીકના પૂર્ણાંક સુધી રાઉન્ડ કરે છે	`round(4.6)` → 5

math.sqrt()	વર્ગમૂળ	`math.sqrt(16)` → `4.0`
math.sin()	સાઇન ફંક્શન	`math.sin(math.pi/2)` → `1.0`

મેમરી ટ્રીક

``PEARS Math'' (Power, Exponents, Arithmetic, Roots, Sine functions in Math)

પ્રશ્ન 4(b) [4 ગુણ]

પાયથન મોડ્યુલ કોડ સાથે સમજાવો.

જવાબ

મોડ્યુલ: પાયથોનમાં મોડ્યુલ એટલે પાયથોન વ્યાખ્યાઓ અને સ્ટેટમેન્ટ્સ ધરાવતી ફાઇલ. ફાઇલનું નામ .py સફિક્સ સાથેનું મોડ્યુલનું નામ છે.

```
\# math
import math

\# math
radius = 5
area = math.pi * math.pow(radius, 2)
print(f"    \{radius\}            \{area:.2f\} ")

\#     import
from math import sqrt, sin
angle = math.pi / 4
print(f"25      \{sqrt(25)\} ")
print(f"\{angle\}          \{sin(angle):.4f\} ")

\# alias    import
import random as rnd
random\_number = rnd.randint(1, 100)
print(f"1   100          : \{random\_number\}")
```

ટેબલ: મોડ્યુલ Import ટેકનિક્સ

પદ્ધતિ	સિન્ટેક્સ	ઉદાહરણ
આખો મોડ્યુલ import કરો	`import module_name`	`import math`
ચોક્કસ આઇટમ્સ import કરો	`from module_name import item1, item2`	`from math import sqrt, sin`
alias સાથે import કરો	`import module_name as alias`	`import random as rnd`

મેમરી ટ્રીક

``CODE-LIB'' (Code Libraries for reuse) - ફરીથી ઉપયોગ માટે કોડ લાઇબ્રેરીઓ

પ્રશ્ન 4(c) [7 ગુણ]

એક પાયથન પ્રોગ્રામ લખો જે નિર્ધારિત કરે છે કે આપેલ નંબર `આર્મસ્ટ્રોંગ નંબર' છે કે વપરાશકર્તા-વ્યાખ્યાયિત કાર્યનો ઉપયોગ કરીને પેલિન્ડ્રોમ છે.

જવાબ

```
\#
def is\_armstrong(num):
    \#
    num\_str = str(num)
    n = len(num\_str)
```

```
    \#
    armstrong\_sum = 0
    for digit in num\_str:
        armstrong\_sum += int(digit) ** n

    \#
    return armstrong\_sum == num

\#
def is\_palindrome(num):
    \#
    num\_str = str(num)
    return num\_str == num\_str[::{-}1]

\#
number = int(input("          : "))

\#
if is\_armstrong(number):
    print(f"\{number\}            ")
else:
    print(f"\{number\}           ")

\#
if is\_palindrome(number):
    print(f"\{number\}          ")
else:
    print(f"\{number\}         ")
```

ટેબલ: ઉદાહરણો

નંબર	આર્મસ્ટ્રોંગ ચેક	પેલિન્ડ્રોમ ચેક
153	$1^3 + 5^3 + 3^3 = 1 + 125 + 27 = 153$	$153 \neq 351$
121	$1^3 + 2^3 + 1^3 = 1 + 8 + 1 = 10 \neq 121$	121 = 121
1634	$1^4 + 6^4 + 3^4 + 4^4 =$ $1 + 1296 + 81 + 256 = 1634$	$1634 \neq 4361$

ડાયાગ્રામ:

```
flowchart LR
    A[Start] {-{-} B[Input number]}
    B {-{-} C\{Check Armstrong\}}
    C {-{-} D\{Check Palindrome\}}
    C {-{-} Yes {-}{-} E[Print {-} Is Armstrong]}
    C {-{-} No {-}{-} F[Print {-} Not Armstrong]}
    D {-{-} Yes {-}{-} G[Print {-} Is Palindrome]}
    D {-{-} No {-}{-} H[Print {-} Not Palindrome]}
    E {-{-} D}
    F {-{-} D}
    G {-{-} I[End]}
    H {-{-} I}
```

મેમરી ટ્રીક

``SAME SUM'' (SAME આગળ-પાછળ પેલિન્ડ્રોમ માટે, SUM ઘાતના અંકોનો આર્મસ્ટ્રોંગ માટે)

પ્રશ્ન 4(a) [3 ગુણ] - OR ઓપ્શન

પાયથોનમાં બિલ્ટ ઇન ફંક્શન સમજાવો.

જવાબ

Built-in Functions: આ ફંક્શન્સ પાયથોનના સ્ટાન્ડર્ડ લાઇબ્રેરીનો ભાગ છે અને કોઈપણ મોડ્યુલ import કર્યા વિના ઉપલબ્ધ છે.
ટેબલ: સામાન્ય પાયથોન Built-in Functions

ફંક્શન	હેતુ	ઉદાહરણ
print()	આઉટપુટ ડિસ્પ્લે	`print("Hello")`
input()	યુઝર ઇનપુટ લે	`name = input("Name: ")`
len()	ઓબ્જેક્ટની લંબાઈ આપે	`len([1, 2, 3])` → `3`
type()	ઓબ્જેક્ટનો પ્રકાર આપે	`type(5)` → `<class 'int'>`
int(), float(), str()	ચોક્કસ પ્રકારમાં રૂપાંતર	`int("5")` → `5`
range()	સીક્વન્સ જનરેટ કરે	`list(range(3))` → `[0, 1, 2]`
sum()	સરવાળો ગણે	`sum([1, 2, 3])` → `6`

મેમરી ટ્રીક

``PITS LCR'' (Print, Input, Type, Sum, Len, Convert, Range)

પ્રશ્ન 4(b) [4 ગુણ] - OR ઓપ્શન

એક પાયથોન કોડનું ઉદાહરણ આપીને પાયથોન મેથ મોડ્યુલનું વર્ણન કરો.

જવાબ

પાયથોન Math મોડ્યુલ: math મોડ્યુલ C સ્ટાન્ડર્ડ દ્વારા વ્યાખ્યાયિત ગાણિતિક ફંક્શન્સની એક્સેસ પ્રદાન કરે છે.

```
\# math
import math

\#
print(f"pi      : \{math.pi\}")
print(f"e      : \{math.e\}")

\#           (          )
angle = math.pi / 3  \# 60
print(f"\{angle:.2f\}         : \{math.sin(angle):.4f\}")
print(f"\{angle:.2f\}          : \{math.cos(angle):.4f\}")
print(f"\{angle:.2f\}         : \{math.tan(angle):.4f\}")

\#
x = 10
print(f"\{x\}           : \{math.log(x):.4f\}")
print(f"\{x\}          10: \{math.log10(x):.4f\}")
print(f"e   \{x\}   : \{math.exp(x):.4f\}")

\#
print(f"25      : \{math.sqrt(25)\}")
print(f"4.3       : \{math.ceil(4.3)\}")
print(f"4.7      : \{math.floor(4.7)\}")
```

ટેબલ: Math મોડ્યુલ કેટેગરીઝ

કેટેગરી	ફંક્શન્સ
સ્થિરાંકો	`math.pi`, `math.e`
ત્રિકોણમિતિ	`sin()`, `cos()`, `tan()`
લોગરિધમિક	`log()`, `log10()`, `exp()`
ન્યુમેરિક	`sqrt()`, `ceil()`, `floor()`

મેમરી ટ્રીક

``PENT'' (Pi/constants, Exponents, Numbers, Trigonometry)

પ્રશ્ન 4(c) [7 ગુણ] - OR ઓપ્શન

પાયથોનમાં વેરીએબલના અવકાશનો કોન્સેપ્ટ સમજાવો અને પાયથોન પ્રોગ્રામમાં વૈશ્વિક અને સ્થાનિક વેરીએબલ કોન્સેપ્ટ લાગુ કરો.

જવાબ

પાયથોનમાં વેરિએબલનો સ્કોપ: વેરિએબલનો સ્કોપ નક્કી કરે છે કે પ્રોગ્રામમાં ક્યાં વેરિએબલ એક્સેસિબલ કે દેખાય છે.
ટેબલ: વેરિએબલ સ્કોપના પ્રકારો

સ્કોપ	વર્ણન	એક્સેસ
Local	ફંક્શનની અંદર વ્યાખ્યાયિત વેરિએબલ્સ	માત્ર ફંક્શનની અંદર
Global	ટોપ લેવલ પર વ્યાખ્યાયિત વેરિએબલ્સ	સમગ્ર પ્રોગ્રામમાં
Enclosing	નેસ્ટેડ ફંક્શન્સના બાહ્ય ફંક્શનના વેરિએબલ્સ	બાહ્ય અને અંદરના ફંક્શનમાં
Built-in	પાયથોનમાં પહેલેથી વ્યાખ્યાયિત વેરિએબલ્સ	સમગ્ર પ્રોગ્રામમાં

કોડ ઉદાહરણ:

```
\#

\# Global
count = 0

def outer\_function():
    \# Enclosing
    name = "Python"

    def inner\_function():
        \# Local
        age = 30
        \# Global
        global count
        count += 1
        \# Enclosing
        print(f"inner\_function     : name is \{name\}")
        print(f"inner\_function     : age is \{age\}")
        print(f"inner\_function     : count is \{count\}")

    \# outer\_function    Local
    language = "Programming"
    print(f"outer\_function     : name is \{name\}")
    print(f"outer\_function     : language is \{language\}")
    print(f"outer\_function     : count is \{count\}")

    \#
    inner\_function()

    \#       {- age   inner\_function    Local  }
    \# print(age)

\#
print(f"Global    : count is \{count\}")
outer\_function()
print(f"         Global    : count is \{count\}")

\#       {-          Local  }
\# print(name)
\# print(language)
```

ડાયાગ્રામ:

```
flowchart LR
    A[Global Scope] {-{-} B[count]}
    A {-{-} C[outer\_function]}
    C {-{-} D[Enclosing Scope: name, language]}
    D {-{-} E[inner\_function]}
    E {-{-} F[Local Scope: age]}
    B {{-}.{-} E}
    D {{-}.{-} E}
```

મેમરી ટ્રીક

``LEGB'' (Local, Enclosing, Global, Built-in - સ્કોપ લુકઅપનો ક્રમ)

પ્રશ્ન 5(a) [3 ગુણ]

આપેલ સૂચિમાં બે ઘટકોને સ્વેપ કરવા માટે પાયથોન પ્રોગ્રામ બનાવો.

જવાબ

```
\#

\#
my\_list = [10, 20, 30, 40, 50]
print("    :", my\_list)

\#
pos1 = int(input("             (    0        ): "))
pos2 = int(input("             (    0        ): "))

\#
if 0 {=} pos1 {} len(my\_list) and 0 {=} pos2 {} len(my\_list):
    \#
    temp = my\_list[pos1]
    my\_list[pos1] = my\_list[pos2]
    my\_list[pos2] = temp

    print(f"    \{pos1\}   \{pos2\}                  :", my\_list)
else:
    print("       !                    .")
```

વૈકલ્પિક પદ્ધતિ:

```
\#      tuple               (      )
if 0 {=} pos1 {} len(my\_list) and 0 {=} pos2 {} len(my\_list):
    my\_list[pos1], my\_list[pos2] = my\_list[pos2], my\_list[pos1]
    print(f"    \{pos1\}   \{pos2\}                  :", my\_list)
```

ટેબલ: સ્વેપિંગ પદ્ધતિઓ

પદ્ધતિ	કોડ
ટેમ્પ વેરિએબલનો ઉપયોગ	`temp = a; a = b; b = temp`
પાયથોન ટપલ અનપેકિંગ	`a, b = b, a`

મેમરી ટ્રીક

``TEMP SWAP'' (ટેમ્પરરી વેરિએબલ સલામત સ્વેપિંગમાં મદદ કરે છે)

પ્રશ્ન 5(b) [4 ગુણ]

ઉદાહરણ આપીને નેસ્ટેડ લિસ્ટ સમજાવો

જવાબ

Nested List: Nested list એટલે એવી લિસ્ટ જેના એલિમેન્ટ્સ તરીકે અન્ય લિસ્ટ હોય, જે મલ્ટી-ડાયમેન્શનલ ડેટા સ્ટ્રક્ચર બનાવે છે.

```
\# Nested list     (3x3    )
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

\#
print("      :", matrix)
print("    :", matrix[0])
print("  1,    2      :", matrix[0][1])  \#    : 2

\#
```

```
matrix[1][1] = 50
print("            :", matrix)

\# Nested list
print("{n}                :")
for row in matrix:
    for element in row:
        print(element, end=" ")
    print()  \#
```

ડાયાગ્રામ:

```
flowchart TD
    A[matrix] {-{-} B[Row 0]}
    A {-{-} C[Row 1]}
    A {-{-} D[Row 2]}
    B {-{-} B1[1]}
    B {-{-} B2[2]}
    B {-{-} B3[3]}
    C {-{-} C1[4]}
    C {-{-} C2[50]}
    C {-{-} C3[6]}
    D {-{-} D1[7]}
    D {-{-} D2[8]}
    D {-{-} D3[9]}
```

ટેબલ: Nested List ઓપરેશન્સ

ઓપરેશન	સિન્ટેક્સ	ઉદાહરણ
એલિમેન્ટ એક્સેસ	`list[row][col]`	`matrix[0][1]`
એલિમેન્ટ મોડિફાય	`list[row][col] = new_value`	`matrix[1][1] = 50`
નવી રો ઉમેરવી	`list.append([...])`	`matrix.append([10, 11, 12])`

મેમરી ટ્રીક

``MARS'' (Matrix Access with Row and column Structure) - મેટ્રિક્સ એક્સેસ રો અને કોલમ માળખા સાથે

પ્રશ્ન 5(c) [7 ગુણ]

ઉદાહરણો સાથે સ્ટ્રિંગ ઓપરેશન્સ સમજાવો

જવાબ

ટેબલ: પાયથોનમાં સ્ટ્રિંગ ઓપરેશન્સ

ઓપરેશન	વર્ણન	ઉદાહરણ
કન્કેટેનેશન	સ્ટ્રિંગ્સ જોડવી	`"Hello" + " World"` → `"Hello World"`
રિપિટિશન	સ્ટ્રિંગ્સ પુનરાવર્તિત કરવી	`"Python" * 3` → `"PythonPythonPython"`
સ્લાઇસિંગ	સબસ્ટ્રિંગ એક્સટ્રેક્ટ	`"Python"[1:4]` → `"yth"`
ઇન્ડેક્સિંગ	એક્સેસ કેરેક્ટર	`"Python"[0]` → `"P"`
લેન્થ	કેરેક્ટર્સ ગણો	`len("Python")` → `6`
મેમ્બરશિપ	ચેક કરો કે હાજર છે	`"P" in "Python"` → `True`
કમ્પેરિઝન	સ્ટ્રિંગ્સ સરખાવો	`"apple" < "banana"` → `True`

કોડ ઉદાહરણ:

```
\#
text = "Python Programming"

\#
print("      :", text[0])
print("      :", text[{-}1])

\#
print("     :", text[:6])
print("     :", text[7:])
print("       :", text[3:10])
print("   :", text[::{-}1])

\#
print("   :", text.upper())
print("   :", text.lower())
print("{P   J      :"}, text.replace("P", "J"))
print("          :", text.split())
print("{m      :"}, text.count({m}))
print("{gram    :"}, text.find("gram"))

\#
print("       ?", text.isalnum())
print("{Py         ?"}, text.startswith("Py"))
print("{ing           ?"}, text.endswith("ing"))
```

ડાયાગ્રામ:

```
flowchart TD
    A["Python Programming"] {-{-} B["Indexing: P (0), g ({-}1)"]}
    A {-{-} C["Slicing: Python (0:6), Programming (7:)"]}
    A {-{-} D["Methods: PYTHON PROGRAMMING (upper())"]}
    A {-{-} E["Checks: startswith, endswith, isalnum, etc"]}
```

મેમરી ટ્રીક

``SCREAM'' (Slice, Concat, Replace, Extract, Access, Methods)

પ્રશ્ન 5(a) [3 ગુણ] - OR ઓપ્શન

આપેલ સૂચિમાં તમામ ઘટકોનો સરવાળો શોધવા માટે પાયથોન પ્રોગ્રામ બનાવો.

જવાબ

```
\#

\#     1:   {-  sum()        }
def sum\_list\_builtin(numbers):
    return sum(numbers)

\#     2:
def sum\_list\_loop(numbers):
    total = 0
    for num in numbers:
        total += num
    return total

\#
```

```
my\_list = [10, 20, 30, 40, 50]
print("  :", my\_list)

\#   {-                  }
print("  {-                 :"}, sum\_list\_builtin(my\_list))

\#
print("             :", sum\_list\_loop(my\_list))
```

ટેબલ: સરવાળા પદ્ધતિઓની તુલના

પદ્ધતિ	ફાયદો
બિલ્ટ-ઈન sum()	સરળ, કાર્યક્ષમ, ઝડપી
લૂપ અભિગમ	કસ્ટમ સમિંગ લોજિક માટે કામ કરે છે

મેમરી ટ્રીક

``ADD ALL'' (દરેક એલિમેન્ટને ઉમેરો)

પ્રશ્ન 5(b) [4 ગુણ] - OR ઓપ્શન

પાયથોન લિસ્ટમાં ઈન્ડેક્સીંગ અને સ્લાઇસિંગ ઓપરેશન્સ સમજાવો

જવાબ

ટેબલ: ઇન્ડેક્સિંગ અને સ્લાઇસિંગ ઓપરેશન્સ

ઓપરેશન	સિન્ટેક્સ	વર્ણન	ઉદાહરણ
પોઝિટિવ ઇન્ડેક્સિંગ	`list[i]`	પોઝિશન i પર આઇટમ એક્સેસ કરો (0-બેઝ્ડ)	`fruits[0]` $\rightarrow$
નેગેટિવ ઇન્ડેક્સિંગ	`list[-i]`	અંતથી આઇટમ એક્સેસ કરો (-1 છેલ્લું છે)	`fruits[-1]` $\rightarrow$
બેઝિક સ્લાઇસિંગ	`list[start:end]`	start થી end-1 સુધીના આઇટમ્સ	`fruits[1:3]` $\rightarrow 1, 2$
સ્ટેપ સાથે સ્લાઇસ	`list[start:end:step]`	step ના અંતરાલ સાથે આઇટમ્સ	`nums[1:6:2]` $\rightarrow 1, 3, 5$
ઇન્ડિસીસ છોડવા	`list[:end]`, `list[start:]`	શરૂઆતથી અથવા અંત સુધી	`fruits[:3]` $\rightarrow 3$
નેગેટિવ સ્લાઇસિંગ	`list[-start:-end]`	અંતથી સ્લાઇસ	`fruits[-3:-1]` $\rightarrow 32$
રિવર્સ	`list[::-1]`	લિસ્ટ રિવર્સ કરો	`fruits[::-1]` $\rightarrow$

કોડ ઉદાહરણ:

```
\#
fruits = ["apple", "banana", "cherry", "date", "elderberry", "fig"]
print("    :", fruits)

\#
print("{n}         :")
print("     :", fruits[0])  \# apple
print("     :", fruits[{-}1])  \# fig
print("      :", fruits[2])  \# cherry

\#
print("{n}         :")
print("         :", fruits[:3])  \# [{apple, banana, cherry]}
print("         :", fruits[{-}3:])  \# [{date, elderberry, fig]}
print("       :", fruits[2:4])  \# [{cherry, date]}
print("         :", fruits[::2])  \# [{apple, cherry, elderberry]}
print("      :", fruits[::{-}1])  \# [{fig, elderberry, date, cherry, banana, apple]}
```

ડાયાગ્રામ:

```
flowchart TD
    A["List: fruits"] {-{-} B\{\{"Indexing"\}\}}
    A {-{-} C\{\{"Slicing"\}\}}
    B {-{-} D["Positive: fruits[0], fruits[1], ..."]}
    B {-{-} E["Negative: fruits[{-}1], fruits[{-}2], ..."]}
    C {-{-} F["Basic: fruits[1:3]"]}
    C {-{-} G["With step: fruits[::2]"]}
    C {-{-} H["Reverse: fruits[::{-}1]"]}
```

મેમરી ટ્રીક

``START-END-STEP'' (સ્લાઇસિંગ સિન્ટેક્સ: [start:end:step])

પ્રશ્ન 5(c) [7 ગુણ] - OR ઓપ્શન

જરૂરી ઉદાહરણ સાથે tuple ને ટૂંકમાં સમજાવો.

જવાબ

Tuple: Tuple એ એલિમેન્ટ્સનો ક્રમબદ્ધ, અપરિવર્તનીય સંગ્રહ છે. એકવાર બનાવ્યા પછી, એલિમેન્ટ્સ બદલી શકાતા નથી.
ટેબલ: Tuple vs List

ફીચર	Tuple	List
સિન્ટેક્સ	`(item1, item2)`	`[item1, item2]`
પરિવર્તનશીલતા	Immutable (બદલી શકાતી નથી)	Mutable (બદલી શકાય છે)
પરફોર્મન્સ	ઝડપી	ધીમું
ઉપયોગ કેસ	ફિક્સ્ડ ડેટા, ડિક્શનરી કીઝ	ડેટા જેને મોડિફિકેશનની જરૂર પડે
મેથડ્સ	ઓછી મેથડ્સ	ઘણી મેથડ્સ

કોડ ઉદાહરણ:

```
\# Tuples
empty\_tuple = ()
single\_item\_tuple = (1,)  \#
mixed\_tuple = (1, "Hello", 3.14, True)
nested\_tuple = (1, 2, (3, 4), 5)

\# Tuple
print("     :", mixed\_tuple[0])  \# 1
print("     :", mixed\_tuple[{-}1])  \# True
print("Nested tuple    :", nested\_tuple[2][0])  \# 3

\# Tuple
print("       :", mixed\_tuple[:2])  \# (1, "Hello")

\# Tuple
a, b, c, d = mixed\_tuple
print("        :", a, b, c, d)

\# Tuple
print("1      :", mixed\_tuple.count(1))  \# 1
print("{Hello      :"}, mixed\_tuple.index("Hello"))  \# 1

\# Tuple
combined\_tuple = mixed\_tuple + nested\_tuple
repeated\_tuple = mixed\_tuple * 2
print("     tuple:", combined\_tuple)
print("    tuple:", repeated\_tuple)

\#           tuples immutable
\# mixed\_tuple[0] = 100  \# TypeError: {tuple object does not support item assignment}
```

ડાયાગ્રામ:

```
flowchart TD
    A["Tuple (1, {Hello, 3.14, True)"] {-}{-}|"index[0]"| B[1]}
    A {-{-}|"index[1]"| C["Hello"]}
    A {-{-}|"index[2]"| D[3.14]}
    A {-{-}|"index[3]"| E[True]}
    F["Operations"] {-{-} G["Access: tuple[i]"]}
    F {-{-} H["Slice: tuple[i:j]"]}
    F {-{-} I["Concatenate: tuple1 + tuple2"]}
    F {-{-} J["Repeat: tuple * n"]}
    K["Methods"] {-{-} L["count()"]}
    K {-{-} M["index()"]}
```

મેમરી ટ્રીક

``IPAC'' (Immutable, Parentheses, Access only, Cannot modify) - અપરિવર્તનીય, કૌંસ, માત્ર એક્સેસ, મોડિફાય ન કરી શકાય