Subject Name (Gujarati)

1323203 -- Summer 2023

Semester 1 Study Material

Detailed Solutions and Explanations

પ્રશ્ન 1(અ) [3 ગુણ]

અલગોરીધમ વ્યાખ્યાયિત કરો. અલગોરીધમનાં ફાયદા શું છે?

જવાબ

અલગોરીધમ એ ચોક્કસ સમસ્યાને ઉકેલવા માટે પગલાઓના ક્રમબદ્ધ સમૂહ અથવા નિયમોનો સેટ છે.
અલગોરીધમના ફાયદા:

- **સ્પષ્ટતા**: સ્પષ્ટ, અસંદિગ્ધ સૂચનાઓ પ્રદાન કરે છે
- **કાર્યક્ષમતા**: સમય અને સંસાધનોને અનુકૂળ બનાવવામાં મદદ કરે છે
- **પુન:ઉપયોગ**: સમાન સમસ્યાઓ માટે વારંવાર ઉપયોગ કરી શકાય છે
- **ચકાસણી**: અમલીકરણ પહેલાં પરીક્ષણ અને ડિબગ કરવું સરળ
- **સંદેશાવ્યવહાર**: ઉકેલને સંદેશાવ્યવહાર કરવા માટે બ્લુપ્રિન્ટ તરીકે કામ કરે છે

મેમરી ટ્રીક

``CERVC'' (Clarity, Efficiency, Reusability, Verification, Communication)

પ્રશ્ન 1(બ) [4 ગુણ]

ફ્લોચાર્ટનો ઉપયોગ કરીને સમસ્યા ઉકેલવાના નિયમો શું છે? સાદું વ્યાજ શોધવા માટેનો ફ્લોચાર્ટ ડીઝાઈન કરો.

જવાબ

ફ્લોચાર્ટનો ઉપયોગ કરીને સમસ્યા ઉકેલવાના નિયમો:

- **યોગ્ય સિમ્બોલ**: વિવિધ ઓપરેશન માટે માનક સિમ્બોલનો ઉપયોગ કરવો
- **દિશાનો પ્રવાહ**: હંમેશા ઉપરથી નીચે, ડાબેથી જમણે સ્પષ્ટ પ્રવાહ જાળવવો
- **એક એન્ટ્રી/એક્ઝિટ**: સ્પષ્ટ શરૂઆત અને અંત બિંદુ હોવા જોઈએ
- **સ્પષ્ટતા**: પગલાં સ્પષ્ટ અને સંક્ષિપ્ત રાખવા
- **સુસંગતતા**: વિગતોનું સુસંગત સ્તર જાળવવું

સાદું વ્યાજ ગણતરી માટેનો ફ્લોચાર્ટ:

```
flowchart LR
    A([  ]) {-{-} B[/     P,      R,    T      /]}
    B {-{-} C[SI = P * R * T / 100]}
    C {-{-} D[/SI      /]}
    D {-{-} E([ ])}
```

મેમરી ટ્રીક

``PDRSC'' (Proper symbols, Direction flow, Required entry/exit, Simplicity, Consistency)

પ્રશ્ન 1(ક) [7 ગુણ]

પાયથોનનાં અસાઇમેંટ ઓપરટેરની યાદી બનાવો અને કોઈપણ ત્રણ અસાઇમેંટ ઓપરટેરોની કામગીરી દશાર્વવા માટે પાયથોન કોડ બનાવો.

જવાબ

પાયથોન અસાઇમેંટ ઓપરેટર્સ:

ઓપરેટર	ઉદાહરણ	સમકક્ષ
=	x = 5	x = 5
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
//=	x //= 5	x = x // 5
**=	x **= 5	x = x ** 5
&=	x &= 5	x = x & 5
|=	x |= 5	x = x | 5
^=	x ^= 5	x = x ^ 5
>>=	x >>= 5	x = x >> 5
<<=	x <<= 5	x = x << 5

અસાઇમેંટ ઓપરેટર્સ દર્શાવતો કોડ:

```
\#
num = 10
print("       :", num)

\# +=
num += 5
print("+= 5   :", num)  \#    : 15

\# {-=         }
num {-=} 3
print("{-= 3   :"}, num)  \#    : 12

\# *=
num *= 2
print("*= 2   :", num)  \#    : 24
```

મેમરી ટ્રીક

``VALUE'' (Variable Assignment is Like Updating Existing values)

પ્રશ્ન 1(ક) OR [7 ગુણ]

પાયથોનનાં ડેટા ટાઇપ્સની યાદી બનાવો અને કોઈપણ ત્રણ ડેટા ટાઇપ્સને ઓળખવા માટેનો પાયથોન કોડ બનાવો.

જવાબ

પાયથોન ડેટા ટાઇપ્સ:

ડેટા ટાઇપ	વર્ણન	ઉદાહરણ
int	ઇન્ટીજર (પૂર્ણાંક સંખ્યાઓ)	42
float	ફ્લોટિંગ પોઇન્ટ (દશાંશ)	3.14
str	સ્ટ્રિંગ (ટેક્સ્ટ)	``Hello''
bool	બૂલિયન (True/False)	True
list	ક્રમિક, પરિવર્તનશીલ સંગ્રહ	[1, 2, 3]
tuple	ક્રમિક, અપરિવર્તનીય સંગ્રહ	(1, 2, 3)
set	અક્રમિક સંગ્રહ	{1, 2, 3}
dict	કી-વેલ્યુ જોડી	{``name'': ``John''}
complex	કોમ્પ્લેક્સ નંબર	2+3j
NoneType	None દર્શાવે છે	None

ત્રણ ડેટા ટાઇપ્સ ઓળખવા માટેનો કોડ:

```
\#
def identify\_data\_type(value):
    data\_type = type(value).\_\_name\_\_
    print(f"  : \{value\}")
    print(f"     : \{data\_type\}")
    print("{-"} * 20)

\# 3   {-              }
identify\_data\_type(42)            \# Integer
identify\_data\_type(3.14)          \# Float
identify\_data\_type("Hello World") \# String

\#    :
\#   : 42
\#      : int
\# {-{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}}
\#   : 3.14
\#      : float
\# {-{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}}
\#   : Hello World
\#      : str
\# {-{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}{-}}
```

મેમરી ટ્રીક

``TYPE-ID'' (Tell Your Python Elements - Identify Data)

પ્રશ્ન 2(અ) [3 ગુણ]

સ્યુડોકોડ વ્યાખ્યાયિત કરો. કોઈપણ બે સંખ્યા માંથી સૌથી નાની સંખ્યા શોધવા માટે સ્યુડોકોડ લખો.

જવાબ

સ્યુડોકોડ એ એલ્ગોરિધમનું ઉચ્ચ-સ્તરીય વર્ણન છે જે પ્રોગ્રામિંગ ભાષાના માળખાકીય સંકેતોનો ઉપયોગ કરે છે પરંતુ મશીન વાંચન કરતાં માનવ વાંચન માટે ડિઝાઇન કરેલ છે.
બે સંખ્યાઓમાંથી સૌથી નાની શોધવા માટે સ્યુડોકોડ:

```
BEGIN
    INPUT first_number, second_number
    IF first_number < second_number THEN
        smallest = first_number
    ELSE
        smallest = second_number
    END IF
    OUTPUT smallest
END
```

મેમરી ટ્રીક

``RISE'' (Read Input, Select smallest, Echo result)

પ્રશ્ન 2(બ) [4 ગુણ]

યુઝર્સ પાસેથી ત્રણ ઇનપુટ વાંચો અને સંખ્યાઓની સરેરાશ શોધવા માટેનો પાયથોન કોડ વિકસાવો.

જવાબ

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

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

\#
print(f"\{num1\}, \{num2\},    \{num3\}     : \{average\}")
```

આકૃતિ:

```
flowchart LR
    A([  ]) {-{-} B[/num1, num2, num3      /]}
    B {-{-} C["average = (num1 + num2 + num3) / 3"]}
    C {-{-} D[/average       /]}
    D {-{-} E([ ])}
```

મેમરી ટ્રીક

``I-ADD-D'' (Input three, ADD them up, Divide by 3)

પ્રશ્ન 2(ક) [7 ગુણ]

દાખલ કરેલ સંખ્યા prime છે કે નહીં તે બતાવવા પાયથોન કોડ લખો.

જવાબ

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

\# 2
if num {} 2:
    print(f"\{num\}            ")
else:
    \# is\_prime   True
    is\_prime = True

    \# 2   sqrt(num)
    for i in range(2, int(num**0.5) + 1):
if num \%

i == 0:

            is\_prime = False
            break

    \#
    if is\_prime:
        print(f"\{num\}           ")
    else:
        print(f"\{num\}            ")
```

આકૃતિ:

```
flowchart LR
    A("  ") {-{-} B("num      ")}
    B {-{-} C\{"num  2?"\}}
```

```
    C {-{-}| | D("num       ")}
    C {-{-}| | E("is\_prime = True")}
    E {-{-} F("i = 2")}
    F {-{-} G\{"(i * i) = num?"\}}
    G {-{-}| | J("is\_prime = True")}
G {-{-}| | H\{"num \%

i == 0?"\}}

    H {-{-}| | I("is\_prime = False")}
    I {-{-} J}
    H {-{-}| | K("i = i + 1")}
    K {-{-} F}
    J {-{-}| | L("num     ")}
    J {-{-}| | D}
    L {-{-} M(" ")}
    D {-{-} M}
```

મેમરી ટ્રીક

``PRIME'' (Positive number, Range check from 2 to , If divisible it's Multiple, Else it's prime)

પ્રશ્ન 2(અ) OR [3 ગુણ]

ફ્લોચાર્ટ અને એલ્ગોરિધમ વચ્ચેનો તફાવત લખો.

જવાબ

ફ્લોચાર્ટ	એલ્ગોરિધમ
માનક સિમ્બોલ અને આકારોનો ઉપયોગ કરીને **દૃશ્ય પ્રતિનિધિત્વ**	**લેખિત વર્ણન** માળખાકીય ભાષાનો ઉપયોગ કરીને
ગ્રાફિકલ પ્રકૃતિને કારણે **સમજવું સરળ**	સિન્ટેક્સ અને શબ્દાવલીનું જ્ઞાન જરૂરી
તાર્કિક પ્રવાહ અને સંબંધોને સ્પષ્ટ રીતે દર્શાવે	ક્રમિક ક્રમમાં **વિગતવાર પગલાં** પ્રદાન કરે
બનાવવા માટે **સમય-લેતી** પરંતુ સમજવા માટે સરળ	**ઝડપથી ડ્રાફ્ટ** પરંતુ સમજવામાં મુશ્કેલ હોઈ શકે
ફેરફાર કરવા કે અપડેટ કરવા વધુ મુશ્કેલ	ફેરફાર કરવા કે અપડેટ કરવા વધુ સરળ

મેમરી ટ્રીક

``VITAL'' (Visual vs Textual, Interpretation ease, Time to create, Alteration flexibility, Logical representation)

પ્રશ્ન 2(બ) OR [4 ગુણ]

નીચેનાં કોડનું આઉટપુટ શું છે?

```
x=10
y=2
print (x*y)
print (x ** y)
print (x//y)
print (x \% y)
```

જવાબ

ઓપરેશન	સમજૂતી	આઉટપુટ
x*y	ગુણાકાર: 10×2	20

x**y	ઘાતાંક: 10^2	100
x//y	પૂર્ણાંક ભાગાકાર: $10 \div 2$	5
x%y	મોડ્યુલસ (શેષ): $10 \div 2$	0

મેમરી ટ્રીક

``MEMO'' (Multiply, Exponent, Modulo, Operations)

પ્રશ્ન 2(ક) OR [7 ગુણ]

નીચેની પેટર્ન દર્શાવવા પાયથોન કોડ લખો:

```
A)                  B)
1                   * * * *
1 2                 * * *
1 2 3               * *
1 2 3 4             *
```

જવાબ

```
\#    A:
print("   A:")
for i in range(1, 5):
    for j in range(1, i + 1):
        print(j, end=" ")
    print()

\#    B:
print("{n}   B:")
for i in range(4, 0, {-}1):
    for j in range(i):
        print("*", end=" ")
    print()
```

આકૃતિ:

```
flowchart TD
    A([  ]) {-{-} B[   A]}
    B {-{-} C[i = 1   4]}
    C {-{-} D[j = 1   i]}
    D {-{-} E[j      ]}
    E {-{-} F[           ]}
    F {-{-} G[   B]}
    G {-{-} H[i = 4   1]}
    H {-{-} I[j = 0   i{-}1]}
    I {-{-} J["*      "]}
    J {-{-} K[           ]}
    K {-{-} L([ ])}
```

મેમરી ટ્રીક

``LOOP-NED'' (Loop Outer, Order Pattern, Nested loops, End with newline, Display)

પ્રશ્ન 3(અ) [3 ગુણ]

જરૂરી ઉદાહરણો સાથે break statementનાં ઉપયોગનું વર્ણન કરો.

જવાબ

break સ્ટેટમેન્ટનો ઉપયોગ લૂપને વચ્ચેથી સમાપ્ત કરવા માટે થાય છે, જ્યારે કોઈ ચોક્કસ શરત પૂરી થાય.
ઉદાહરણ:

```
\#
numbers = [2, 4, 6, 7, 8, 10]
for num in numbers:
    if num \% 2 != 0:
        print(f"        : \{num\}")
        break
    print(f"\{num\}          ")
```

આઉટપુટ:

```
2
4
6
        : 7
```

મેમરી ટ્રીક

``EXIT'' (EXecute until condition, Immediately Terminate)

પ્રશ્ન 3(બ) [4 ગુણ]

યોગ્ય ઉદાહરણ સાથે if...else statement સમજાવો.

જવાબ

if...else સ્ટેટમેન્ટ એ એક કન્ડિશનલ સ્ટેટમેન્ટ છે જે નિર્દિષ્ટ શરત True કે False હોવાના આધારે અલગ-અલગ કોડ બ્લોક્સ એક્ઝિક્યુટ કરે છે.
સિન્ટેક્સ:

```
if   :
    \#      True
else:
    \#      False
```

ઉદાહરણ:

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

if number \% 2 == 0:
    print(f"\{number\}         ")
else:
    print(f"\{number\}           ")
```

આકૃતિ:

```
flowchart LR
    A([  ]) {-{-} B[/        /]}
    B {-{-} C\{number \% 2 == 0?\}}
    C {-{-}| | D[/"number    "       /]}
    C {-{-}| | E[/"number     "       /]}
    D {-{-} F([ ])}
    E {-{-} F}
```

મેમરી ટ્રીક

``CITE'' (Check condition, If True Execute this, Else execute that)

પ્રશ્ન 3(ક) [7 ગુણ]

0 થી N સંખ્યા સુધીની ફીબોનાકી શ્રેણી પ્રિન્ટ કરવા માટે યુઝર ડેફાઇન ફંકશન બનાવો જેમાં N એ પૂર્ણાંક સંખ્યા છે અને આરગયુમેન્ટ તરીકે પાસ થાય છે.

જવાબ

```
\#
def print\_fibonacci(n):
    """
    0   n
        :
        n:         (inclusive)
    """
    \#
    a, b = 0, 1

    \# n
    if n {} 0:
        print("                      ")
        return

    \#
    print(n, "            :")

    if n {=} 0:
        print(a, end=" ")  \#

    if n {=} 1:
        print(b, end=" ")  \#

    \#
    while a + b {=} n:
        next\_term = a + b
        print(next\_term, end=" ")
        a, b = b, next\_term

\#
print\_fibonacci(55)
```

આકૃતિ:

```
flowchart LR
A([  ]) {-{-} B[a=0,

b=1         ]}

    B {-{-} C\{n  0?\}}
    C {-{-}| | D[           ]}
    D {-{-} K([ ])}
    C {-{-}| | E\{n = 0?\}}
    E {-{-}| | F[a       ]}
    E {-{-}| | G\{n = 1?\}}
    F {-{-} G}
    G {-{-}| | H[b       ]}
    G {-{-}| | I\{a + b = n?\}}
    H {-{-} I}
I {-{-}| | J[next\_term = a + bn next\_term       n

a = bn

b = next\_term]}

    J {-{-} I}
    I {-{-}| | K}
```

મેમરી ટ્રીક

``FIBER'' (First terms set, Initialize variables, Build next term, Echo results, Repeat until limit)

પ્રશ્ન 3(અ) OR [3 ગુણ]

જરૂરી ઉદાહરણો સાથે continue statementનાં ઉપયોગનું વર્ણન કરો.

જવાબ

continue સ્ટેટમેન્ટનો ઉપયોગ લૂપની વર્તમાન ઇટરેશન છોડીને આગળની ઇટરેશન પર જવા માટે થાય છે.
ઉદાહરણ:

```
\# 1   10
for i in range(1, 11):
    if i \% 2 == 0:
        continue  \#
    print(i)
```

આઉટપુટ:

```
1
3
5
7
9
```

મેમરી ટ્રીક

``SKIP'' (Skip current iteration, Keep looping, Ignore remaining statements, Proceed to next iteration)

પ્રશ્ન 3(બ) OR [4 ગુણ]

ઉદાહરણ સાથે For loop statement સમજાવો.

જવાબ

For લૂપનો ઉપયોગ કોઈ સિક્વન્સ (જેમ કે લિસ્ટ, ટપલ, સ્ટ્રિંગ) કે અન્ય ઇટરેબલ ઓબ્જેક્ટ પર ઇટરેશન કરવા અને દરેક આઇટમ માટે કોડનો બ્લોક એક્ઝિક્યુટ કરવા માટે થાય છે.
સિન્ટેક્સ:

```
for     in    :
    \#
```

ઉદાહરણ:

```
\# 1   5
for num in range(1, 6):
    square = num ** 2
    print(f"\{num\}     \{square\}  ")
```

આઉટપુટ:

```
1     1
2     4
3     9
4     16
5     25
```

આકૃતિ:

```
flowchart LR
```

```
    A([  ]) {-{-} B[num = 1   5   ]}
    B {-{-} C[square = num ** 2]}
    C {-{-} D[          ]}
    D {-{-} E\{     ?\}}
    E {-{-}| | B}
    E {-{-}| | F([ ])}
```

મેમરી ટ્રીક

``FIRE'' (For each Item, Run commands, Execute until end)

પ્રશ્ન 3(ક) OR [7 ગુણ]

યુઝર ડિફાઇન ફંકશનની મદદથી આપેલ નંબર આર્મસ્ટ્રોંગ નંબર છે કે પેલિન્ડ્રોમ તે નિર્ધારિત કરવા પાયથોન કોડ લખો.

જવાબ

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

    \#        n
    sum\_of\_powers = sum(int(digit) ** n for digit in num\_str)

    \#
    return sum\_of\_powers == num

\#
def is\_palindrome(num):
    \#
    num\_str = str(num)

    \#
    return num\_str == num\_str[::{-}1]

\#
def check\_number(num):
    if is\_armstrong(num):
        print(f"\{num\}           ")
    else:
        print(f"\{num\}            ")

    if is\_palindrome(num):
        print(f"\{num\}         ")
    else:
        print(f"\{num\}          ")

\#
number = int(input("          : "))
check\_number(number)
```

આકૃતિ:

```
flowchart LR
    A([  ]) {-{-} B[/        /]}
    B {-{-} C[check\_number        ]}
    C {-{-} D[is\_armstrong        ]}
    D {-{-} E\{sum\_of\_powers == num?\}}
    E {-{-}| | F[/"      "      /]}
```

```
    E {-{-}| | G[/"       "      /]}
    C {-{-} H[is\_palindrome        ]}
    H {-{-} I\{num == reversed num?\}}
    I {-{-}| | J[/"      "      /]}
    I {-{-}| | K[/"       "      /]}
    F {-{-} L([ ])}
    G {-{-} L}
    J {-{-} L}
    K {-{-} L}
```

મેમરી ટ્રીક

``APC'' (Armstrong check: Power sum of digits, Palindrome check: Compare with reverse)

પ્રશ્ન 4(અ) [3 ગુણ]

સ્કેન કરેલ નંબર even છે કે odd તે શોધવા પાયથોન કોડ વિકસાવો અને યોગ્ય મેસેજ પ્રિન્ટ કરો.

જવાબ

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

\#
if number \% 2 == 0:
    print(f"\{number\}         ")
else:
    print(f"\{number\}          ")
```

આકૃતિ:

```
flowchart LR
    A([  ]) {-{-} B[/        /]}
    B {-{-} C\{number \% 2 == 0?\}}
    C {-{-}| | D[/"number    "      /]}
    C {-{-}| | E[/"number     "      /]}
    D {-{-} F([ ])}
    E {-{-} F}
```

મેમરી ટ્રીક

``MODE'' (Modulo Operation Determines Even-odd)

પ્રશ્ન 4(બ) [4 ગુણ]

ફંકશનની વ્યાખ્યા આપો. યુઝર ડિફાઇન ફંકશન યોગ્ય ઉદાહરણ આપી સમજાવો.

જવાબ

ફંકશન એ કોડનો એવો બ્લોક છે જે ચોક્કસ કાર્ય કરવા માટે વ્યવસ્થિત અને ફરીથી ઉપયોગ કરી શકાય છે. યુઝર-ડિફાઇન ફંકશન એ પ્રોગ્રામર દ્વારા બનાવવામાં આવેલા ફંકશન છે જે કસ્ટમ ઓપરેશન કરે છે.

યુઝર-ડિફાઇન ફંકશનના ઘટકો:

- **def કીવર્ડ**: ફંકશન વ્યાખ્યાની શરૂઆત દર્શાવે છે
- **ફંકશન નામ**: ફંકશન માટે ઓળખકર્તા
- **પેરામીટર્સ**: ઇનપુટ વેલ્યુઝ (વૈકલ્પિક)
- **ડોકસ્ટ્રિંગ**: ફંકશનનું વર્ણન (વૈકલ્પિક)
- **ફંકશન બોડી**: એક્ઝિક્યુટ થનાર કોડ
- **રિટર્ન સ્ટેટમેન્ટ**: આઉટપુટ વેલ્યુ (વૈકલ્પિક)

ઉદાહરણ:

```
\#                  {-      }
def calculate\_area(length, width):
    """

        :
        length:
        width:
       :

    """
    area = length * width
    return area

\#
result = calculate\_area(5, 3)
print(f"        : \{result\}")
```

મેમરી ટ્રીક

``DRAPE'' (Define function, Receive parameters, Acquire result, Process data, End with return)

પ્રશ્ન 4(ક) [7 ગુણ]

વિવિધ સ્ટ્રિંગ ઓપરેશનની યાદી બનાવો અને કોઈપણ ત્રણ ઉદાહરણનો ઉપયોગ કરીને સમજાવો.

જવાબ

પાયથોનમાં સ્ટ્રિંગ ઓપરેશન્સ:

ઓપરેશન	વર્ણન
Concatenation	+ નો ઉપયોગ કરીને સ્ટ્રિંગ્સ જોડવી
Repetition	* નો ઉપયોગ કરીને સ્ટ્રિંગ રિપીટ કરવી
Indexing	પોઝિશન દ્વારા કેરેક્ટર એક્સેસ કરવા
Slicing	સ્ટ્રિંગનો ભાગ એક્સટ્રેક્ટ કરવો
Methods (len, upper, lower, વગેરે)	સ્ટ્રિંગ મેનિપ્યુલેશન માટે બિલ્ટ-ઇન ફંકશન્સ
Membership Testing	સ્ટ્રિંગમાં સબસ્ટ્રિંગ છે કે નહીં તે તપાસવું
Formatting	ફોર્મેટેડ સ્ટ્રિંગ્સ બનાવવી
Escape Sequences	થી શરૂ થતા સ્પેશિયલ કેરેક્ટર્સ

ત્રણ સ્ટ્રિંગ ઓપરેશન્સ વિથ ઉદાહરણ:

1. **સ્ટ્રિંગ Concatenation:**

```
first\_name = "John"
last\_name = "Doe"
full\_name = first\_name + " " + last\_name
print(full\_name)  \#    : John Doe
```

1. **સ્ટ્રિંગ Slicing:**

```
message = "Python Programming"
print(message[0:6])    \#    : Python
print(message[7:])     \#    : Programming
print(message[{-}11:])   \#    : Programming
```

1. **સ્ટ્રિંગ Methods:**

```
text = "python programming"
print(text.upper())    \#    : PYTHON PROGRAMMING
print(text.capitalize())  \#    : Python programming
print(text.replace("python", "Java"))  \#    : Java programming
```

મેમરી ટ્રીક

``CSM'' (Concatenate strings, Slice portions, Manipulate with methods)

પ્રશ્ન 4(અ) OR [3 ગુણ]

પોઝિટિવ અને નેગેટિવ નંબર તપાસવા પાયથોન કોડ બનાવો.

જવાબ

```
\#
\#
number = float(input("          : "))

\#        ,    ,
if number {} 0:
    print(f"\{number\}            ")
elif number {} 0:
    print(f"\{number\}           ")
else:
    print("       ")
```

આકૃતિ:

```
flowchart LR
    A([  ]) {-{-} B[/        /]}
    B {-{-} C\{number  0?\}}
    C {-{-}| | D[/"number       "       /]}
    C {-{-}| | E\{number  0?\}}
    E {-{-}| | F[/"number      "       /]}
    E {-{-}| | G[/"       "       /]}
    D {-{-} H([ ])}
    F {-{-} H}
    G {-{-} H}
```

મેમરી ટ્રીક

``SIGN'' (See If Greater than 0, Negative otherwise)

પ્રશ્ન 4(બ) OR [4 ગુણ]

યોગ્ય ઉદાહરણો સાથે local અને global વેરિએબલ સમજાવો.

જવાબ

પાયથોનમાં વેરિએબલ્સના અલગ-અલગ સ્કોપ્સ હોઈ શકે છે:

વેરિએબલ પ્રકાર	વર્ણન
Local Variable	ફંકશનની અંદર વ્યાખ્યાયિત અને માત્ર તે ફંકશનની અંદર જ એક્સેસિબલ
Global Variable	ફંકશનની બહાર વ્યાખ્યાયિત અને પ્રોગ્રામના તમામ ભાગમાં એક્સેસિબલ

ઉદાહરણ:

```
\# Global
count = 0  \#  Global

def update\_count():
    \# Local
    local\_var = 5  \#  Local

    \#       Global
    global count
    count += 1

    print(f"Local    : \{local\_var\}")
    print(f"Global     (      ): \{count\}")

\#
update\_count()

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

\#
\# print(local\_var)  \# NameError: name {local\_var is not defined}
```

આઉટપુટ:

```
Local    : 5
Global     (      ): 1
Global     (      ): 1
```

મેમરી ટ્રીક

``SCOPE'' (Some variables Confined to function Only, Program-wide Exposure for others)

પ્રશ્ન 4(ક) OR [7 ગુણ]

વિવિધ લિસ્ટ ઓપરેશનની યાદી બનાવો અને કોઈપણ ત્રણ ઉદાહરણનો ઉપયોગ કરીને સમજાવો.

જવાબ

પાયથોનમાં લિસ્ટ ઓપરેશન્સ:

ઓપરેશન	વર્ણન
લિસ્ટ બનાવવી	સ્ક્વેર બ્રેકેટ્સ [] નો ઉપયોગ
ઇન્ડેક્સિંગ	પોઝિશન દ્વારા એલિમેન્ટ એક્સેસ કરવા

સ્લાઇસિંગ	લિસ્ટના ભાગો એક્સટ્રેક્ટ કરવા
એપેન્ડ	છેલ્લે એલિમેન્ટ ઉમેરવા
ઇન્સર્ટ	ચોક્કસ પોઝિશન પર એલિમેન્ટ ઉમેરવા
રિમૂવ	ચોક્કસ એલિમેન્ટ દૂર કરવા
પોપ	એલિમેન્ટ દૂર કરવું અને પાછું મેળવવું
સોર્ટ	લિસ્ટ એલિમેન્ટ્સ ઓર્ડર કરવા
રિવર્સ	લિસ્ટનો ક્રમ ઊલટાવવો
એક્સ્ટેન્ડ	લિસ્ટ્સ જોડવી
લિસ્ટ કોમ્પ્રિહેન્શન્સ	એક્સપ્રેશન્સનો ઉપયોગ કરીને લિસ્ટ બનાવવી

ત્રણ લિસ્ટ ઓપરેશન્સ વિથ ઉદાહરણ:

1. **લિસ્ટ ઇન્ડેક્સિંગ અને સ્લાઇસિંગ:**

```
fruits = ["apple", "banana", "cherry", "orange", "kiwi"]
print(fruits[1])        \#    : banana
print(fruits[{-}1])       \#    : kiwi
print(fruits[1:4])      \#    : [{banana, cherry, orange]}
```

1. **લિસ્ટ મેથડ્સ (append, insert, remove):**

```
numbers = [1, 2, 3]
numbers.append(4)       \#    4
print(numbers)          \#    : [1, 2, 3, 4]

numbers.insert(0, 0)    \#      0   0
print(numbers)          \#    : [0, 1, 2, 3, 4]

numbers.remove(2)       \# 2
print(numbers)          \#    : [0, 1, 3, 4]
```

1. **લિસ્ટ કોમ્પ્રિહેન્શન્સ:**

```
\#
squares = [x**2 for x in range(1, 6)]
print(squares)  \#    : [1, 4, 9, 16, 25]

\#
numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
evens = [x for x in numbers if x \% 2 == 0]
print(evens)    \#    : [2, 4, 6, 8, 10]
```

મેમરી ટ્રીક

``AIM'' (Access with index, Insert/modify elements, Make using comprehensions)

પ્રશ્ન 5(અ) [3 ગુણ]

લિસ્ટમાં આપેલ બે એલિમેન્ટ્સને સ્વેપ કરવા પાયથોન કોડ લખો.

જવાબ

```
\#
def swap\_elements(my\_list, pos1, pos2):
    """
        pos1   pos2
    """
    \#
    if 0 {=} pos1 {} len(my\_list) and 0 {=} pos2 {} len(my\_list):
        \#
        my\_list[pos1], my\_list[pos2] = my\_list[pos2], my\_list[pos1]
```

```
        return True
    else:
        return False

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

\#     1   3
if swap\_elements(numbers, 1, 3):
    print("    :", numbers)
else:
    print("       ")
```

આઉટપુટ:

```
    : [10, 20, 30, 40, 50]
    : [10, 40, 30, 20, 50]
```

મેમરી ટ્રીક

``SWAP'' (Select positions, Watch boundaries, Assign simultaneously, Print result)

પ્રશ્ન 5(બ) [4 ગુણ]

પાયથોનનાં Math મોડ્યુલ અને random મોડ્યુલ ઉદાહરણનાં ઉપયોગ કરીને સમજાવો.

જવાબ

Math અને random મોડ્યુલ મેથેમેટિકલ ઓપરેશન્સ અને રેન્ડમ નંબર જનરેશન માટેના ફંકશન્સ પ્રદાન કરે છે.
Math મોડ્યુલ:

```
import math

\#
print(math.pi)          \#    : 3.141592653589793
print(math.e)           \#    : 2.718281828459045

\#
print(math.sqrt(16))    \#    : 4.0
print(math.ceil(4.2))   \#    : 5
print(math.floor(4.8))  \#    : 4
print(math.pow(2, 3))   \#    : 8.0
```

Random મોડ્યુલ:

```
import random

\# 0   1
print(random.random())       \#    : 0.123... (  )

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

\#
colors = ["red", "green", "blue"]
print(random.choice(colors)) \#    : "green" (  )

\#
numbers = [1, 2, 3, 4, 5]
random.shuffle(numbers)
print(numbers)               \#    : [3, 1, 5, 2, 4] (  )
```

મેમરી ટ્રીક

``MR-CS'' (Math for Calculations, Random for Choice and Shuffling)

પ્રશ્ન 5(ક) [7 ગુણ]

Tuple ફંકશન અને ઓપરેશન દર્શાવવા પાયથોન કોડ લખો.

જવાબ

```
\# Tuple

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

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

\# Tuple
print("{n}Tuple     :")
print(mixed\_tuple[1:3])    \#    : ("Hello", 3.14)

\# Tuple     (concatenation)
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
tuple3 = tuple1 + tuple2
print("{n}    tuple:", tuple3)  \#    : (1, 2, 3, 4, 5, 6)

\# Tuple
repeated\_tuple = tuple1 * 3
print("{n}      tuple:", repeated\_tuple)  \#    : (1, 2, 3, 1, 2, 3, 1, 2, 3)

\# Tuple
numbers = (1, 2, 3, 2, 4, 2)
print("{n}2    :", numbers.count(2))  \#    : 3
print("3     :", numbers.index(3))    \#    : 2

\# Tuple
print("{n}Tuple     :")
x, y, z = (10, 20, 30)
print(f"x=\{x\},

y=\{y\},

z=\{z\}")  \#    :

x=10,

y=20,

z=30


\# Tuple
print("{n}        :")
```

```
print(3 in numbers)     \#    : True
print(5 in numbers)     \#    : False

\#     tuple    tuple
my\_list = [1, 2, 3]
my\_tuple = tuple(my\_list)
print("{n}    tuple:", my\_tuple)

back\_to\_list = list(my\_tuple)
print("tuple    :", back\_to\_list)
```

આકૃતિ:

```
flowchart LR
    A[Tuples    ] {-{-} B[         ]}
    B {-{-} C[Tuples       ]}
    C {-{-} D[Tuples    ]}
    D {-{-} E[Tuples       ]}
    E {-{-} F[Tuple           ]}
    F {-{-} G[Tuples      ]}
    G {-{-} H[        ]}
    H {-{-} I[          ]}
```

મેમરી ટ્રીક

``CASC-RUMTC'' (Create, Access, Slice, Concatenate, Repeat, Use methods, Membership test, Tuple conversion)

પ્રશ્ન 5(અ) OR [3 ગુણ]

લિસ્ટમાં સામેલ એલિમેંટનો સરવાળો કરવા પાયથોન કોડ લખો.

જવાબ

```
\#
def sum\_of\_elements(numbers):
    """

    """
    total = 0
    for num in numbers:
        total += num
    return total

\#
my\_list = [10, 20, 30, 40, 50]
print("  :", my\_list)
print("         :", sum\_of\_elements(my\_list))  \#    : 150

\#   {-  sum()                  }
print("  {-                :"}, sum(my\_list))  \#    : 150
```

આકૃતિ:

```
flowchart LR
        direction LR
    A([  ]) {-{-} B[total = 0       ]}
    B {-{-} C[        num   ]}
    C {-{-} D[total += num]}
    D {-{-} E\{      ?\}}
    E {-{-}| | C}
    E {-{-}| | F[total     ]}
```

```
    F {-{-} G([ ])}
```

મેમરી ટ્રીક

``SITE'' (Sum Initialized To zero, Elements added one by one)

પ્રશ્ન 5(બ) OR [4 ગુણ]

નીચે આપેલ built in functionsનો ઉપયોગ સમજાવો: ૧) Print() ૨) Min() ૩) Sum() ૪) Input()

જવાબ

ફંકશન	હેતુ	ઉદાહરણ	આઉટપુટ
print()	કન્સોલ પર આઉટપુટ દર્શાવે છે	`print("Hello World")`	Hello World
min()	iterableમાંથી સૌથી નાના આઇટમને પરત કરે છે	`min([5, 3, 8, 1])`	1
sum()	iterableમાંના તમામ આઇટમ્સનો સરવાળો આપે છે	`sum([1, 2, 3, 4])`	10
input()	વપરાશકર્તા પાસેથી ઇનપુટ વાંચે છે	`name = input("  : ")`	(વપરાશકર્તાની ઇનપુટની રાહ જુએ છે)

ઉદાહરણ કોડ:

```
\# print()
print("  ,     !")  \#
print("a", "b", "c", sep="{-"})  \#          : a{-b{-}c}
print("        ", end=" ")  \#     end
print("       ")  \#    :

\# min()
numbers = [15, 8, 23, 4, 42]
print("      :", min(numbers))  \#    : 4
print("5, 2, 9       :", min(5, 2, 9))  \#    : 2
chars = "wxyz"
print("      :", min(chars))  \#    : w

\# sum()
print("        :", sum(numbers))  \#    : 92
print("              :", sum(numbers, 10))  \#    : 102

\# input()
user\_input = input("        : ")  \#
print("       :", user\_input)  \#
```

મેમરી ટ્રીક

``PMSI'' (Print to display, Min for smallest, Sum for total, Input for reading)

પ્રશ્ન 5(ક) OR [7 ગુણ]

સેટ ફંકશન અને ઓપરેશન દર્શાવવા પાયથોન કોડ લખો.

જવાબ

```
\#

\#
```

```
empty\_set = set()  \#
numbers = \{1, 2, 3, 4, 5\}
duplicates = \{1, 2, 2, 3, 4, 4, 5\  }\#
print("   :", numbers)
print("         :", duplicates)  \#    : \{1, 2, 3, 4, 5\}

\#
numbers.add(6)
print("{n}6      :", numbers)  \#    : \{1, 2, 3, 4, 5, 6\}

\#
numbers.update([7, 8, 9])
print("        :", numbers)  \#    : \{1, 2, 3, 4, 5, 6, 7, 8, 9\}

\#
numbers.remove(5)  \#
print("{n}5        :", numbers)

numbers.discard(10)  \#
print("10 discard      :", numbers)  \#

popped = numbers.pop()  \#
print("pop       :", popped)
print("pop   :", numbers)

\#
set1 = \{1, 2, 3, 4, 5\}
set2 = \{4, 5, 6, 7, 8\}

\#
union\_set = set1 | set2  \#    set1.union(set2)
print("{n}   :", union\_set)  \#    : \{1, 2, 3, 4, 5, 6, 7, 8\}

\#
intersection\_set = set1 \& set2  \#    set1.intersection(set2)
print("    :", intersection\_set)  \#    : \{4, 5\}

\#
difference\_set = set1 {-} set2  \#    set1.difference(set2)
print("    (set1 {- set2):"}, difference\_set)  \#    : \{1, 2, 3\}

\#
symmetric\_diff = set1 \^{} set2  \#    set1.symmetric\_difference(set2)
print("        :", symmetric\_diff)  \#    : \{1, 2, 3, 6, 7, 8\}

\#
subset = \{1, 2\}
print("{n} \{1, 2\   set1      ?"}, subset.issubset(set1))  \#    : True
print("  set1   \{1, 2\      ?"}, set1.issuperset(subset))  \#    : True
```

આકૃતિ:

```
flowchart TD
    A[     ] {-{-} B[         ]}
    B {-{-} C[       ]}
    B {-{-} D[         ]}
    A {-{-} E[     ]}
    E {-{-} F[   ]}
    E {-{-} G[    ]}
    E {-{-} H[   ]}
    E {-{-} I[        ]}
    E {-{-} J[  /   ]}
```

મેમરી ટ્રીક

``CARDS-UI'' (Create, Add, Remove, Discard elements, Set operations - Union, Intersection)