

# Subject Name (Gujarati)

4311601 -- Winter 2023

Semester 1 Study Material

*Detailed Solutions and Explanations*

## પ્રશ્ન 1(અ) [3 ગુણ]

ફલો ચાર્ટ શું છે? ફલો ચાર્ટમાં વપરાતા પ્રતીકોની યાદી બનાવો.

### જવાબ

ફલો ચાર્ટ એ અલોરિધમની ગ્રાફિકલ રજૂઆત છે જે પ્રક્રિયાના પગલાંઓ અને નિર્ણય બિંદુઓ દર્શાવે છે.

ફલો ચાર્ટ પ્રતીકોનું ટેબલ:

પ્રતીક	નામ	ઉપયોગ
અંડાકાર	ટર્મિનલ	પ્રારંભ/અંત
લંબચોરસ	પ્રોસેસ	પ્રક્રિયા/ગણતરી
હીરો	નિર્ણય	શરતી નિવેદનો
સમાંતર ચતુર્ભકોણ	ઇનપુટ/આઉટપુટ	ડેટા લેવો/આપવો
વૃત્ત	કનેક્ટર	ભાગોને જોડવા
તીર	ફલો લાઇન	દિશા

### મુખ્ય બિંદુઓ:

- વિઝ્યુઅલ રજૂઆત: પ્રોગ્રામ લોજિક ગ્રાફિકલી દર્શાવે
- પગલાં દર પગલાં: ક્રમિક ઓપરેશનનો ફલો
- નિર્ણય લેવો: હીરા શરતી શાખાઓ દર્શાવે

### મેમરી ટ્રીક

"ફલો ચાર્ટ્સ પ્રોગ્રામ સ્ટેપ્સ વિઝ્યુઅલી દર્શાવે"

## પ્રશ્ન 1(બ) [4 ગુણ]

for લૂપ માટે ટૂકી નોંધ લખો.

### જવાબ

for લૂપ Python માં સિક્વન્સ (list, tuple, string, range) પર iterate કરવા માટે વપરાય છે.

For લૂપ ટેબલ:

ઘટક	સિન્ટેક્સ	ઉદાહરણ
મૂળભૂત	for variable in sequence:	for i in range(5):
રેન્જ	range(start, stop, step)	range(1, 10, 2)
યાદી	for item in list:	for x in [1,2,3]:
સ્ટ્રિંગ	for char in string:	for c in "hello":

## સરળ કોડ ઉદાહરણ:

```
for i in range(3):
    print(i)
\#      : 0, 1, 2
```

### મુખ્ય લક્ષણો:

- ઓટોમેટિક iteration: મેન્યુઅલ કાઉન્ટરની જરૂર નથી
- સિકવન્સ ટ્રાવર્સલ: કોઈપણ iterable ઓફ્જેક્ટ સાથે કામ કરે
- રેન્જ ફુલ્ફલન: નંબર સિકવન્સ સરળતાથી બનાવે

### મેમરી ટ્રીક

“For લૂપ્સ સિકવન્સમાં iterate કરે”

## પ્રશ્ન 1(ક) [7 ગુણ]

ફિબોનાકી શ્રેણીને નમી ટર્મ સુધી દર્શાવવા માટે એક પ્રોગ્રામ લખો જ્યાં યુઝર દ્વારા n આપવામાં આવે છે.

### જવાબ

#### ફિબોનાકી શ્રેણી પ્રોગ્રામ:

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

\#           initialize
a, b = 0, 1

\#
if n == 1:
    print(a, end=" ")

\#
if n == 2:
    print(b, end=" ")

\#
for i in range(2, n):
    c = a + b
    print(c, end=" ")
    a, b = b, c
```

#### અણોરિધમ ફ્લો:

```
flowchart LR
    A[ ] --{-{-}} B[n ]
    B --{-{-}} C{n = 1?}
    C --{-{-}}| | D[0 ]
    C --{-{-}}| | H[ ]
    D --{-{-}} E{n = 2?}
    E --{-{-}}| | F[1 ]
    E --{-{-}}| | H
    F --{-{-}} G[i=2 n{-}1]
    G --{-{-}} I[c = a + b]
    I --{-{-}} J[c ]
    J --{-{-}} K[a = b,
    b = c]
    K --{-{-}} L{i n{-}1?}
```

```
L { -{-} | | G}
L { -{-} | | H[ ] }
```

#### મુખ્ય કોન્સ્ટ્રુક્શન્સ:

- સિક્વેન્શિયલ જનરેશન: દરેક ટર્મ = પાછળી બે ટર્મનો સરવાતો
- વેરિયેબલ સ્ટેર્ચિંગ: a, b વેલ્યુઝ અસરકારક રીતે અપડેટ કરો
- યુઝર ઇનપુટ: ડાયનમિક શ્રેણી લેન્થ

#### મેમરી ટ્રીક

“ફિલોનાકી: પાછળા બે નંબરો ઉમેરો”

### પ્રશ્ન 1(ક OR) [7 ગુણ]

1 થી 100 સુધીના ODD નંબરો પ્રિન્ટ કરવા માટે ફ્લોચાર્ટ.

#### જવાબ

1 થી 100 ODD નંબરો માટે ફ્લોચાર્ટ:

```
flowchart LR
    A[ ] { -{-} } B[i = 1]
    B { -{-} } C\{i = 100?\}
    C { -{-} | | D\{i \% 2 != 0?\}}
    D { -{-} | | E[i ] }
    D { -{-} | | F[i = i + 1] }
    E { -{-} F }
    F { -{-} C }
    C { -{-} | | G[ ] }
```

અનુસરિક Python કોડ:

```
for i in range(1, 101):
    if i \% 2 != 0:
        print(i, end=" ")
```

વૈકલ્પિક પદ્ધતિ:

```
for i in range(1, 101, 2):
    print(i, end=" ")
```

#### મુખ્ય તત્ત્વો:

- લૂપ કંટ્રોલ: i 1 થી 100 સુધી
- વિષમ ચેક: i \% 2 != 0 શરત
- સ્ટેપ વધારો: આગલા નંબર પર જાઓ

#### મેમરી ટ્રીક

“વિષમ નંબરો: 2 થી ભાગ્યે 1 બાકી”

### પ્રશ્ન 2(અ) [3 ગુણ]

નંબર પેલિન્ડ્રોમ છે કે નહીં તે શોધવા માટે પ્રોગ્રામ લખો.

#### જવાબ

પેલિન્ડ્રોમ ચેક પ્રોગ્રામ:

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

```

reverse = 0

# 
while temp != 0:
    reverse = reverse * 10 + temp % 10
    temp = temp // 10

```

```

# 
if num == reverse:
    print(f"\{num\}      ")
else:
    print(f"\{num\}      ")

```

અલોરિધમ ટેબલ:

પગલું	ઓપરેશન	ઉદાહરણ (121)
1	છેલ્લો અંક મેળવો	$121 \% 10 = 1$
2	રિવર્સ બનાવો	$0 * 10 + 1 = 1$
3	છેલ્લો અંક દૂર કરો	$121 // 10 = 12$
4	0 સુધી પુનરાવર્તન	પ્રક્રિયા ચાલુ રાખો

મુખ્ય બિંદુઓ:

- ડિજિટ એક્સ્ટ્રેક્શન: મોડ્યુલો (%) ઓપરેટર વાપરો
- રિવર્સ બિલ્ડિંગ: 10 થી ગુણા કરી ડિજિટ ઉમેરો
- સરખામણી: મૂળ બરાબર રિવર્સ

### મેમરી ટ્રીક

“પેલિન્ડ્રોમ આગળ પાછળ સરખું વાંચાય”

## પ્રશ્ન 2(બ) [4 ગુણ]

Python પ્રોગ્રામિંગની વિશેષતાઓ સમજાવો.

### જવાબ

Python વિશેષતાઓનું ટેબલ:

વિશેષતા	વર્ણન	ફાયદો
સરળ સિન્ટેક્સ	સાદો, વાંચી શકાય તેવો કોડ	જડપી ડેવલપમેન્ટ
ઇન્ટરપ્રૈટર	કમ્પાઇલેશનની જરૂર નથી	જડપી ટેસ્ટિંગ
ઓફ્જિક્ટ-ઓરિએન્ટેડ	કલાસ અને ઓફ્જિક્ટ સપોર્ટ	કોડ રિયુઝેબિલિટી
ઓપન સોર્સ	વાપરવા માટે ફ્રી	લાઇસન્સિંગ કોસ્ટ નથી
ક્રોસ-પ્લેટફોર્મ	મલ્ટિપ્લાન OS પર ચાલે	વ્યાપક કમ્પેટિબિલિટી
મોટી લાઇબ્રેરીઓ	વ્યાપક બિલ્ટ-ઇન મોડ્યુલ્સ	સમૃદ્ધ કાર્યક્ષમતા

મુખ્ય ફાયદાઓ:

- શિખાઉ-મિત્ર: શીખવામાં અને સમજવામાં સરળ
- વર્સોટાઇલ: વેબ ડેવલપમેન્ટ, AI, ડેટા સાયન્સ
- કોડ્યુનિટી સપોર્ટ: મોટો ડેવલપર કોડ્યુનિટી
- ડાયનેમિક ટાઇપિંગ: વેરિયેબલ ટાઇપ ડિક્લેરેશનની જરૂર નથી

### મેમરી ટ્રીક

“Python: સરળ, શક્તિશાળી, લોકપ્રિય પ્રોગ્રામિંગ”

## પ્રશ્ન 2(ક) [7 ગુણ]

Python પ્રોગ્રામની બેસિક સ્ટ્રક્ચર સમજાવો.

### જવાબ

#### Python પ્રોગ્રામ સ્ટ્રક્ચર:

```
\#!/usr/bin/env python3
# Shebang      ( )

"""
    (docstring)

"""

# Import
import math
from datetime import date

#
PI = 3.14159
count = 0

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

#
class Calculator:
    def \_\_init\_\_(self):
        self.result = 0

#
if \_\_name\_\_ == "\_\_main\_\__":
    #
    radius = 5
    area = calculate\_area(radius)
    print(f"    : {area}")
```

#### સ્ટ્રક્ચર કમ્પોનન્ટ્સ ટેબલ:

કમ્પોનન્ટ	હેતુ	ઉદાહરણ
Shebang	સિસ્ટમ ઇન્ફ્રાસ્ટ્રક્ચર	#!/usr/bin/env python3
Docstring	પ્રોગ્રામ ડોક્યુમેન્ટેશન	"""     """      (docstring)
Imports	બાધ મોડ્યુલ્સ	import math
વેરિયેબલ્સ	ગ્લોબલ ડેટા સ્ટોરેજ	PI = 3.14159
ફંક્શન્સ	પુનઃવપરાશ કોડ બ્લોક્સ	def function_name():
કલાસીસ	ઓફ્ઝેક્ટ ટેમ્પલેટ્સ	class ClassName:
મેઇન બ્લોક	પ્રોગ્રામ એક્ઝિક્યુશન	if __name__ == "__main__":

#### મુખ્ય સિદ્ધાંતો:

- ઇન્કોરેશન: કોડ બ્લોક્સ વ્યાખ્યાપિત કરે (4 સ્પેસીસ આગ્રહિય)
- કોમેન્ટ્સ: સિંગલ લાઇન માટે #, મલ્ટિ-લાઇન માટે """ """"
- મોડ્યુલેરીટી: ફંક્શન અને કલાસમાં કોડ ગોઠવો

### મેમરી ટ્રીક

“સ્ટ્રક્ચર: ઇમ્પોર્ટ, ડિફાઇન, એક્ઝિક્યુટ"

## પ્રશ્ન 2(અ OR) [3 ગુણ]

સ્ટ્રિંગને રિવર્સ કરવા માટે પ્રોગ્રામ લખો.

### જવાબ

#### સ્ટ્રિંગ રિવર્સલ પ્રોગ્રામ:

```
\#      1:
string = input("      : ")
reversed\_string = string[::-1]
print(f"      : \{reversed\_string\}")

\#      2:
string = input("      : ")
reversed\_string = ""
for char in string:
    reversed\_string = char + reversed\_string
print(f"      : \{reversed\_string\}")
```

#### રિવર્સલ પદ્ધતિઓનું ટેબલ:

પદ્ધતિ	સિન્ટેક્સ	ઉદાહરણ
સ્લાઇસિંગ	string[::-1]	“hello” → “olleh”
લૂપ	કેરેક્ટર દ્વારા કેરેક્ટર બનાવો	દરેક char આગળ ઉમેરો
બિલ્ટ-ઇન	"".join(reversed(string))	રિવર્સ સિક્વન્સ જોડો

#### મુખ્ય કોન્સોપ્ટ્સ:

- સ્લાઇસિંગ: સૌથી અસરકારક પદ્ધતિ
- કન્કેનેશન: કેરેક્ટર દ્વારા કેરેક્ટર સ્ટ્રિંગ બનાવો
- ઇન્ડિક્સિંગ: સ્ટ્રિંગ પોજિશન્સ એક્સેસ કરો

### મેમરી ટ્રીક

“રિવર્સ: છેલ્લો કેરેક્ટર પહેલો”

## પ્રશ્ન 2(બ OR) [4 ગુણ]

લોજિકલ ઓપરેટરને ઉદાહરણ સાથે સમજાવો.

### જવાબ

#### Python લોજિકલ ઓપરેટર્સ:

ઓપરેટર	સિમ્બોલ	વર્ણન	ઉદાહરણ	પરિણામ
AND	and	બંને શરતો સાચી	True and False	False
OR	or	ઓછામાં ઓછી એક શરત સાચી	True or False	True
NOT	not	શરતની વિરુદ્ધ	not True	False

## ઉદાહરણ કોડ:

```
a = 10  
b = 5  
  
\# AND  
if a {} 5 and b {} 10:  
    print("      ")  
  
\# OR  
if a {} 15 or b {} 10:  
    print("      ")  
  
\# NOT  
if not (a {} 5):  
    print("a 5      ")
```

## ટ્રૂથ ટેબલ:

A	B	A and B	A or B	not A
T	T	T	T	F
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

## મુખ્ય ઉપયોગો:

- જાટિલ શરતો: બહુવિધ ચેકસ કંબાઇન કરો
- નિર્ણય લેવો: પ્રોગ્રામ ફલો કંટ્રોલ કરો
- બુલિયન લોજિક: True/False ઓપરેશન્સ

## મેમરી ટ્રીક

“AND બધાની જરૂર, OR એકની જરૂર, NOT ઉલટાવે”

## પ્રશ્ન 2(ક) OR) [7 ગુણ]

Python માં વિવિધ ડેટા પ્રકારો સમજાવો

### જવાબ

#### Python ડેટા ટાઇપ્સ વર્ગીકરણ:

#### Mermaid Diagram (Code)

```
{Shaded}  
{Highlighting} []  
graph TD  
    A[Python] --{-{-}{}}--> B[]  
    A --{-{-}{}}--> C[]  
    A --{-{-}{}}--> D[]  
    A --{-{-}{}}--> E[]  
    A --{-{-}{}}--> F[]  
    B --{-{-}{}}--> G[int]  
    B --{-{-}{}}--> H[float]  
    B --{-{-}{}}--> I[complex]  
    C --{-{-}{}}--> J[str]  
    C --{-{-}{}}--> K[list]  
    C --{-{-}{}}--> L[tuple]  
{Highlighting}  
{Shaded}
```

## ડેટા ટાઇપ્સ ટેબલ:

ટાઇપ	ઉદાહરણ	વર્ણન	Mutable
int	42	પૂર્ણ સંખ્યાઓ	ના
float	3.14	દશાંશ સંખ્યાઓ	ના
str	"hello"	ટેક્સ્ટ ડેટા	ના
list	[1,2,3]	કમાંકિત સંગ્રહ	હા
tuple	(1,2,3)	કમાંકિત અપરિવર્તનીય	ના
dict	{"a":1}	કી-વેલ્યુ જોડીઓ	હા
bool	True/False	બુલિયન વેલ્યુઝ	ના
set	{1,2,3}	ચુનિક તત્વો	હા

## ઉદાહરણ કોડ:

```
\#
age = 25          # int
price = 99.99    # float
complex\_num = 3+4j # complex

\#
name = "Python"      # string
numbers = [1,2,3,4]   # list
coordinates = (10,20) # tuple

\#
is\_active = True     # boolean
unique\_items = \{1,2,3\} # set
student = \{"name":"John", "age":20\} # dict
```

### મુખ્ય લક્ષણો:

- ડાયનેમિક ટાઇપિંગ: વેરિયેબલ ટાઇપ ડિક્લાર કરવાની જરૂર નથી
- ટાઇપ કન્વર્ટન: સુસંગત ટાઇપ્સ વચ્ચે કન્વર્ટ કરો
- બિલ્ટ-ઇન ફંક્શન્સ: ચેકિંગ માટે type(), isinstance()

## મેમરી ટ્રીક

"Python ટાઇપ્સ: નંબર્સ, સિકવન્સીસ, કલેક્શન્સ"

## પ્રશ્ન 3(અ) [3 ગુણ]

Python માં ફ્લો કંટ્રોલ શું છે? ઉદાહરણ સાથે સમજાવો

### જવાબ

ફ્લો કંટ્રોલ શરતી અને લૂપ સ્ટ્રક્ચર્સ વાપરીને પ્રોગ્રામ સ્ટેટમેન્ટ્સનો એક્ઝિક્યુશન ઓર્ડર મેનેજ કરે છે.  
ફ્લો કંટ્રોલ પ્રકારોનું ટેબલ:

પ્રકાર	સ્ટેટમેન્ટ	હેતુ	ઉદાહરણ
સિકવેન્શિયલ	સામાન્ય એક્ઝિક્યુશન	લાઇન બાય લાઇન	print("Hello")
સિલેક્શન	if, elif, else	નિર્ણય લેવો	if x > 0:
Iteration	for, while	પુનરાવર્તન	for i in range(5):
Jump	break, continue	લૂપ કંટ્રોલ	break

## ઉદાહરણ કોડ:

```
\#
age = 18
if age == 18:
    print(" ")
else:
    print(" ")

\# Iteration
for i in range(3):
    print(f" : {i}")
```

### મુખ્ય કોન્સોટ્ટ્સ:

- શરતી એક્ઝિક્યુશન: શરતોના આધારે કોડ ચાલે
- લૂપ સ્ટ્રક્ચર્સ: કોડ બ્લોક્સ પુનરાવર્તન
- પ્રોગ્રામ ફ્લો: એક્ઝિક્યુશન પાથ કંટ્રોલ

## મેમરી ટ્રીક

“ફ્લો કંટ્રોલ: નિર્ણય, પુનરાવર્તન, Jump”

## પ્રશ્ન 3(બ) [4 ગુણ]

નેરેડ if સ્ટેપેન્ટ સમજાવવા માટે પ્રોગ્રામ લખો.

### જવાબ

#### નેરેડ If સ્ટેપેન્ટ પ્રોગ્રામ:

```
\#     if
marks = int(input(" : "))

if marks == 0 and marks == 100:
    if marks == 90:
        grade = "A+"
    elif marks == 80:
        if marks == 85:
            grade = "A"
        else:
            grade = "B+"
    elif marks == 70:
        grade = "B"
    elif marks == 60:
        grade = "C"
    else:
        grade = "F"
    print(f" : {grade}")
else:
    print(" ")
```

#### નેરેડ સ્ટ્રક્ચર ડાયાગ્રામ:

```
marks input
|
0{=marks=100 }
```

True

False

```
marks{=90           Invalid }
```

A+

...

#### મુખ્ય લક્ષણો:

- બહુવિધ સ્તરો: if સ્ટેટમેન્ટ્સ અંદર if સ્ટેટમેન્ટ્સ
- જાટેલ શરતો: બહુવિધ માપદંડો હેન્ડલ કરો
- લોજિકલ સ્ટ્રક્ચર: નિર્ણય વૃક્ષો ગોડવો

#### મેમરી ટ્રીક

“નેસ્ટેડ If: નિર્ણયોની અંદર નિર્ણયો”

### પ્રશ્ન 3(ક) [7 ગુણ]

Arguments અને Parameters ના પ્રકારો સમજાવવા માટે એક પ્રોગ્રામ લખો.

#### જવાબ

##### Arguments અને Parameters ના પ્રકારો:

```
\# 1.      Arguments
def greet(name, age):
    print(f"  \{name\},     \{age\}      ")

greet(" ", 25)  #      arguments

\# 2.      Arguments
greet(age=30, name=" ")  #      arguments

\# 3.      Parameters
def introduce(name, city=" "):
    print(f"\{name\} \{city\}      ")

introduce(" ")  #
introduce(" ", " ")  #

\# 4.      {- Arguments (*args)}
def sum\_all(*numbers):
    return sum(numbers)

result = sum\_all(1, 2, 3, 4, 5)
print(f" : \{result\}")

\# 5.      Arguments (**kwargs)
def display\_info(**info):
    for key, value in info.items():
        print(f"\{key\}: \{value\}")

display\_info(name=" ", age=28, city=" ")
```

##### Parameters પ્રકારોનું ટેબલ:

પ્રકાર	સિન્ટેક્સ	ઉદાહરણ	વર્ણન
પોઝિશનલ	def func(a, b):	func(1, 2)	ક્રમ મહત્વનો
કીવર્ટ	def func(a, b):	a=1)func(b=2, a=1)	નામ સ્પેસિફિકેશન
ડિફોલ્ટ	def func(a, b=10):	func(5)	ડિફોલ્ટ વેલ્યુ

```
*args      def func(*args):      func(1,2,3)      વેરિયેબલ પોજિશનલ
**kwargs  def func(**kwargs):   b=2)func(a=1, b=2)  વેરિયેબલ કીવર્ડ
```

#### મુખ્ય કોન્સોપ્ટ્સ:

- લવચીકતા: ડેટા પાસ કરવાની વિવિધ રીતો
- કમ મહત્વ: પોજિશનલ vs કીવર્ડ
- વેરિયેબલ arguments: અજાણી સંખ્યાના ઇનપુટ્સ હેન્ડ કરો

#### મેમરી ટ્રીક

“Parameters: પોજિશન, કીવર્ડ્સ, ડિફોલ્ટ્સ, વેરિયેબલ્સ”

## પ્રશ્ન 3(અ OR) [3 ગુણ]

**break** અને **continue** statement ને ઉદાહરણ સાથે સમજાવો.

#### જવાબ

##### Break અને Continue સ્ટેટમેન્ટ્સ:

##### Break સ્ટેટમેન્ટ:

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

i == 5:

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

##### Continue સ્ટેટમેન્ટ:

```
\# Continue    {- iteration   }
for i in range(5):
if

i == 2:

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

##### સરખામણી ટેબ્લ:

સ્ટેટમેન્ટ	હેતુ	કિયા	ઉદાહરણ ઉપયોગ
break	લૂપમાંથી બહાર નીકળો	સંપૂર્ણ લૂપ સમાપ્ત કરે	શરત પર બહાર નીકળો
continue	iteration છોડો	આગલા iteration પર જાઓ	સ્પેસિફિક વેલ્યુઝ છોડો

##### મુખ્ય તફાવતો:

- Break: લૂપમાંથી સંપૂર્ણ બહાર નીકળો
- Continue: માત્ર વર્તમાન iteration છોડો
- ફલો કંટ્રોલ: લૂપ એક્ઝિક્યુશન મેનેજ કરે

#### મેમરી ટ્રીક

“Break બહાર નીકળે, Continue છોડે”

### પ્રશ્ન 3(બ) OR) [4 ગુણ]

નીચેની પેટન દર્શાવવા માટે એક પ્રોગ્રામ બનાવો

1  
12  
123  
1234  
12345

#### જવાબ

નંબર પેટન પ્રોગ્રામ:

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

  

```
\#      2:
for i in range(1, 6):
    line = ""
    for j in range(1, i + 1):
        line += str(j)
    print(line)
```

  

```
\#      3: join
for i in range(1, 6):
    numbers = [str(j) for j in range(1, i + 1)]
    print("".join(numbers))
```

પેટન લોજિક ટેબલ:

પંક્તિ	નંબર્સ	રેન્જ	આઉટપુટ
1	1	1 થી 1	1
2	1,2	1 થી 2	12
3	1,2,3	1 થી 3	123
4	1,2,3,4	1 થી 4	1234
5	1,2,3,4,5	1 થી 5	12345

મુખ્ય કોન્સેપ્ટ્સ:

- નેસ્ટેડ લૂપ્સ: બાધ્ય પંક્તિઓ માટે, અંદરુની નંબર્સ માટે
- રેન્જ ફંક્શન: નંબર સિક્વન્સ જનરેટ કરે
- પ્રિન્ટ કંટ્રોલ: નવી લાઇનો ટાઇવા માટે end=" " વાપરો

#### મેમરી ટ્રીક

“પેટન: પંક્તિ નંબર કોલમ કાઉન્ટ નક્કી કરે”

### પ્રશ્ન 3(ક) OR) [7 ગુણ]

દરેક માટે કોડ લખીને નીચેના ગાણ્યિતિક કાર્યો સમજાવો: 1. abs() 2. max() 3. pow() 4. sum()

#### જવાબ

Python માં ગાણ્યિતિક ફંક્શન્સ:

```
\# 1. abs() {-      }
numbers = [-5, 3.7, -10.2, 0]
```

```

print("abs()           :")
for num in numbers:
    print(f"abs(\{num\}) = \{abs(num)\}")

# 2. max() {-}
list1 = [4, 7, 2, 9, 1]
print(f"\{n}max()       :")
print(f"max(\{list1\}) = \{max(list1)\}")
print(f"max(10, 25, 5) = \{max(10, 25, 5)\}")
print(f"max({hello}) = \{max({hello})\}"  #)

# 3. pow() {-}
print(f"\{n}pow()       :")
print(f"pow(2, 3) = \{pow(2, 3)\}"      # 2^3 = 8)
print(f"pow(5, 2) = \{pow(5, 2)\}"      # 5^2 = 25)
print(f"pow(8, 1/3) = \{pow(8, 1/3)\}"  # 8

# 4. sum() {-}
numbers = [1, 2, 3, 4, 5]
print(f"\{n}sum()       :")
print(f"sum(\{numbers\}) = \{sum(numbers)\}")
print(f"sum(\{numbers\}, 10) = \{sum(numbers, 10)\}"  #

```

#### ફંક્શન સારાંશ ટેબલ:

ફંક્શન	સિન્ટેક્સ	હેતુ	ઉદાહરણ	પરિણામ
abs()	abs(x)	એબ્સોલ્યુટ વેલ્યુ	abs(-5)	5
max()	max(iterable)	મહત્તમ વેલ્યુ	max([1,5,3])	5
pow()	pow(x, y)	x ને y ની પાવર	pow(2, 3)	8
sum()	sum(iterable)	વેલ્યુનો સરવાળો	sum([1,2,3])	6

#### મુખ્ય ઉપયોગો:

- **abs()**: અંતર ગણતરી, એરર હેન્ડલિંગ
- **max()**: મહત્તમ શોધવું, સ્પર્ધાના પરિણામો
- **pow()**: વૈજ્ઞાનિક ગણતરી, ચકવૃદ્ધિ વ્યાજ
- **sum()**: કુલ ગણતરી, આંકડાશાસ્ત્ર

#### મેમરી ટ્રીક

“Math ફંક્શન્સ: એબ્સોલ્યુટ, મહત્તમ, પાવર, સરવાળો”

### પ્રશ્ન 4(અ) [3 ગુણ]

Variables નો scope સમજાવો.

#### જવાબ

વેરિયેબલ સ્કોપ એ પ્રોગ્રામમાં તે પ્રદેશનો સંદર્ભ આપે છે જ્યાં વેરિયેબલ એક્સેસ કરી શકાય.

સ્કોપ પ્રકારોનું ટેબલ:

સ્કોપ	વર્ણન	જીવનકાળ	એક્સેસ
લોકલ	ફંક્શનની અંદર	ફંક્શન એક્ઝિક્યુશન	માત્ર ફંક્શન
ગ્લોબલ	ફંક્શનોની બહાર	પ્રોગ્રામ એક્ઝિક્યુશન	આખો પ્રોગ્રામ
બિન્ડ-ઇન	Python કીવર્ડ્સ	Python સેશન	બધે

## ઉદાહરણ કોડ:

```
x = 10  #  
  
def my\_function():  
    y = 20  #  
    print(f"    y: {y}")  
    print(f"    x: {x}")  
  
my\_function()  
print(f"    x: {x}")  
# print(y)  # : y
```

### મુખ્ય નિયમો:

- લોકલ વેરિયેબલ્સ: ફંક્શનોની અંદર બનાવાયાં
- ગ્લોબલ વેરિયેબલ્સ: સમગ્ર પ્રોગ્રામમાં એક્સોસિબલ
- **LEGB નિયમ:** Local → Enclosing → Global → Built – in

## મેમરી ટ્રીક

"સ્કોપ: લોકલ ફંક્શનમાં રહે, ગ્લોબલ બધે રહે"

## પ્રશ્ન 4(બ) [4 ગુણ]

નેસ્ટેડ LOOP અને નંબર્સ ડિસ્પ્લે કરવા માટે પ્રોગ્રામ ડેવલપ કરો.

### જવાબ

#### નેસ્ટેડ લૂપ પ્રોગ્રામ:

```
\# 1:  
print("      :")  
for i in range(1, 4):  
    for j in range(1, 5):  
        print(f"\{i}\{j}", end=" ")  
    print()  #  
  
\# 2:  
print("{n}      :")  
for i in range(1, 4):  
    for j in range(1, 6):  
        result = i * j  
        print(f"\{result:3\}", end=" ")  
    print()  
  
\# 3:  
print("{n}      :")  
for i in range(1, 5):  
    for j in range(1, i + 1):  
        print(j, end=" ")  
    print()
```

#### નેસ્ટેડ લૂપ સ્ટ્રક્ચર:

(i)

```
i = 1
```

(j)

j=1,2,3,4

i = 2

#### મુખ્ય કોન્સેપ્ટ્સ:

- બાધ્ય લૂપ: પંક્તિઓ/મુખ્ય iterations કર્ટ્રોલ કરે
- અંદરૂની લૂપ: કોલમ્સ/નાના iterations કર્ટ્રોલ કરે
- એક્ઝિક્યુશન ફલો: અંદરૂનું પૂર્ણ થાય પછી બાધ્ય વધે

#### મેમરી ટ્રીક

“નેસ્ટેડ લૂપ્સ: બાધ્ય અંદરૂનીને કર્ટ્રોલ કરે”

### પ્રશ્ન 4(ક) [7 ગુણ]

1 થી 50 ની રેઝમાં ODD અને EVEN નંબરોની LIST બનાવવા માટે પ્રોગ્રામ લખો.

#### જવાબ

##### ODD અને EVEN નંબર્સ પ્રોગ્રામ:

```
\#      1:
odd\_numbers = []
even\_numbers = []

for i in range(1, 51):
    if i \% 2 == 0:
        even\_numbers.append(i)
    else:
        odd\_numbers.append(i)

print("      (1{-50}:")
print(odd\_numbers)
print(f"  : \{len(odd\_numbers)\}")

print("{n}      (1{-50}:")
print(even\_numbers)
print(f"  : \{len(even\_numbers)\}")

\#      2:
odd\_list = [i for i in range(1, 51) if i \% 2 != 0]
even\_list = [i for i in range(1, 51) if i \% 2 == 0]

print(f"{n}      (      ): \{odd\_list[:10]\}...")  \#      10
print(f" (      ): \{even\_list[:10]\}...")  \#      10

\#      3:
odd\_range = list(range(1, 51, 2))  \#      1,      2
even\_range = list(range(2, 51, 2))  \#      2,      2

print(f"{n}      (      ): \{odd\_range[:10]\}...")
print(f" (      ): \{even\_range[:10]\}...")
```

#### નંબર વર્ગીકરણ ટેબલ:

પ્રકાર	શરત	રેઝ 1-10	કાઉન્ટ (1-50)
વિષમ	n \% 2 != 0	1,3,5,7,9	25

**મુખ્ય તકનિકો:**

- મોડ્યુલો ઓપરેટર: બાકીની ચેક માટે %
- લિસ્ટ કમ્પીઝન: સંક્ષિપ્ત લિસ્ટ સર્જન
- રેન્જ ફુંક્શન: સિકવન્સ અસરકારક રીતે જનરેટ કરે

**મેમરી ટ્રીક**

“વિષમ/સમ: 2 થી ભાગ્યે બાકી 1/0”

**પ્રશ્ન 4(અ OR) [3 ગુણ]**

સ્ટ્રિંગ સ્લાઇસિંગને ઉદાહરણ સાથે સમજાવો.

**જવાબ**

સ્ટ્રિંગ સ્લાઇસિંગ [start:stop:step] સિન્ટેક્સ વાપરીને સ્ટ્રિંગના ભાગો એકસ્ટ્રેક્ટ કરે છે.  
સ્લાઇસિંગ સિન્ટેક્સ ટેબલ:

સિન્ટેક્સ	વર્ણન	ઉદાહરણ	પરિણામ
s[start:stop]	start થી stop-1 સુધી	"hello"[1:4]	"ell"
s[start:]	start થી અંત સુધી	"hello"[2:]	"llo"
s[:stop]	શરૂઆતથી stop-1 સુધી	"hello"[:3]	"hel"
s[::step]	દરેક step કેરેક્ટર	"hello"[:2]	"hlo"
s[::-1]	સ્ટ્રિંગ રિવર્સ	"hello"[:-1]	"olleh"

**ઉદાહરણ કોડ:**

```
text = "Python Programming"

# print(f"    6      : \{text[:6]\}")      # "Python"
print(f"    11     : \{text[7:]\}")      # "Programming"
print(f"      : \{text[2:8]\}")        # "thon P"

# print(f"    2      : \{text[::2]\}")      # "Pto rgamn"

# print(f"      : \{text[{-}1]\}")        # "g"
print(f"      : \{text[::{-}1]\}")      # "gnimmargorP nohtyP"
```

**મુખ્ય લક્ષણો:**

- ગ્રિરો-બેરડ ઇન્ડેક્સિંગ: 0 થી શરૂ
- નેગેટિવ ઇન્ડેક્સિંગ: અંતથી ગણતરી (-1)
- અપરિવર્તનીય: મૂળ સ્ટ્રિંગ અપરિવર્તિત

**મેમરી ટ્રીક**

“સ્લાઇસ: શરૂ, બંધ, સ્ટેપ”

**પ્રશ્ન 4(બ OR) [4 ગુણ]**

આપેલ સંખ્યાના ફેક્ટોરિયલ શોધવા માટે user defined function નો ઉપયોગ કરીને પ્રોગ્રામ લખો.

## જવાબ

ફેક્ટોરિયલ ફંક્શન પ્રોગ્રામ:

```
def factorial(n):
    """
    """
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

def factorial_iterative(n):
    """
    """
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# number = int(input(" : "))
if number < 0:
    print(" ")
else:
    result1 = factorial(number)
    result2 = factorial_iterative(number)
    print(f"\{number\} = \{result1\}")


```

ફેક્ટોરિયલ ટેબલ:

n	ફેક્ટોરિયલ	ગણતરી
0	1	બેઝ કેસ
1	1	બેઝ કેસ
3	6	$3 \times 2 \times 1$
5	120	$5 \times 4 \times 3 \times 2 \times 1$

મુખ્ય કોન્સોપ્ટ્સ:

- રિક્રૂન: ફંક્શન પોતાને કોલ કરે
- બેઝ કેસ: રિકર્સિવ કોલ્સ બંધ કરે
- યુગર-ડિફાઇન્ડ: કસ્ટમ ફંક્શન સર્જન

## મેમરી ટ્રીક

“ફેક્ટોરિયલ: નીચેના બધા નંબર્સ ગુણા કરો”

## પ્રશ્ન 4(ક) OR [7 ગુણ]

આપેલ સ્ટ્રિંગમાં સબ સ્ટ્રિંગ હાજર છે કે કેમ તે તપાસવા માટે user defined function લખો.

## જવાબ

સબસ્ટ્રિંગ ચેક ફંક્શન:

```
def find_substring(main_string, sub_string):
    """
    """
    if sub_string in main_string:
```

```

        index = main\_string.find(sub\_string)
        return True, index
    else:
        return False, {-}1

def count\_substring(main\_string, sub\_string):
    """
    """
    return main\_string.count(sub\_string)

def find\_all\_positions(main\_string, sub\_string):
    """
    """
    positions = []
    start = 0
    while True:
        pos = main\_string.find(sub\_string, start)
        if pos == {-}1:
            break
        positions.append(pos)
        start = pos + 1
    return positions

\#
text = input("          : ")
search = input("          : ")

found, position = find\_substring(text, search)
if found:
    print(f"  {{}}\{{search}\}{      }\{{position}\}      ")
    count = count\_substring(text, search)
    all\_pos = find\_all\_positions(text, search)
    print(f"  : {{count}\}")
    print(f"  : {{all\_pos}\}")
else:
    print(f"  {{}}\{{search}\}{      }")

```

### સ્ટ્રિંગ મેથડ્સ ટેબલ:

મેથડ	હેતુ	ઉદાહરણ	પરિણામ
find()	પ્રથમ પોઝિશન શોધો	"hello".find("ll")	2
count()	ઘટનાઓ ગણો	"hello".count("l")	2
in	અસ્તિત્વ ચેક કરો	"ll" in "hello"	True
index()	પોઝિશન શોધો (ન મળે તો એરર)	"hello".index("e")	1

### મુખ્ય લક્ષણો:

- બાબુબિધ પદ્ધતિઓ: શોધવાની વિવિધ રીતો
- પોઝિશન ટ્રેકિંગ: મળેલ સબસ્ટ્રિંગનો ઇન્ડેક્સ પરત કરે
- એરર હેન્ડલિંગ: પ્રોસેસિંગ પહેલાં ચેક કરે

### મેમરી ટ્રીક

“સબસ્ટ્રિંગ: શોધ, મેળવ, ગણ, પોઝિશન”

### પ્રશ્ન 5(અ) [3 ગુણ]

ઉદાહરણ સાથે List કેવી રીતે બનાવવી અને એક્સેસ કરવી તે સમજાવો.

## જવાબ

લિસ્ટ સર્જન અને એક્સેસ:

```
\#
empty_list = []
numbers = [1, 2, 3, 4, 5]
mixed = [1, "hello", 3.14, True]
nested = [[1, 2], [3, 4], [5, 6]]

\#
print(f" : \{numbers[0]\}")      # 1
print(f" : \{numbers[-1]\}")     # 5
print(f" : \{numbers[1:4]\}")    # [2, 3, 4]
```

લિસ્ટ એક્સેસ પદ્ધતિઓ:

પદ્ધતિ	સિન્ટેક્સ	ઉદાહરણ	પરિણામ
ઇન્ડેક્સ	list[i]	[1,2,3] [1]	2
નેગેટિવ	list[-i]	[1,2,3] [-1]	3
સ્લાઇસ	list[start:stop]	[1,2,3,4] [1:3]	[2,3]

મુખ્ય લક્ષણો:

- કમાંકિત સંગ્રહ: તત્વોની પોઝિશન-સ છે
- પરિવર્તનશીલ: સર્જન પછી સુધારી શકાય
- મિશ્ર પ્રકાર: વિવિધ ડેટા ટાઇપ્સની મંજૂરી

## મેમરી ટ્રીક

"લિસ્ટ્સ: બનાવો, ઇન્ડેક્સ, એક્સેસ"

## પ્રશ્ન 5(બ) [4 ગુણા]

LIST પર કરી શકાય તેવી કામગીરીની યાદી બનાવો. એક લિસ્ટને બીજુ લિસ્ટમાં બનાવવા અને કોપી કરવા માટે પ્રોગ્રામ લખો.

## જવાબ

લિસ્ટ ઓપરેશન્સ અને કોપી પ્રોગ્રામ:

```
\#
original = [1, 2, 3, 4, 5]
print(f" : \{original\}")

\#
shallow_copy = original.copy()
slice_copy = original[:]
list_copy = list(original)

\#
original.append(6)
print(f"append : \{original\}")
print(f" : \{shallow_copy\}")

\#
numbers = [10, 20, 30]
numbers.append(40)          #
numbers.insert(1, 15)       #
numbers.remove(20)         #
popped = numbers.pop()     #
```

લિસ્ટ ઓપરેશન્સ ટેબલ:

ઓપરેશન	મેથડ	ઉદાહરણ	પરિણામ
ઉમેરો	append()	[1, 2].append(3)	[1, 2, 3]
ઇન્સર્ટ	insert(i, x)	[1, 3].insert(1, 2)	[1, 2, 3]
દૂર કરો	remove(x)	[1, 2, 3].remove(2)	[1, 3]
પોપ	pop()	[1, 2, 3].pop()	[1, 2]

#### મુખ્ય કોન્સોપ્ટ્સ:

- શેલો કોપી: સમાન તત્વો સાથે સ્વતંત્ર લિસ્ટ
- ડીપ કોપી: નેરટેડ સ્ટ્રક્ચર માટે જરૂરી
- બહુવિધ પદ્ધતિઓ: કોપી કરવાની વિવિધ તકનીકો

#### મેમરી ટ્રીક

“લિસ્ટ ઓપરેશન્સ: ઉમેરો, ઇન્સર્ટ, દૂર કરો, પોપ, કોપી”

### પ્રશ્ન 5(ક) [7 ગુણ]

LIST ની વિવિધ બિલ્ટ-ઇન methods ની સૂચિ બનાવો અને ઉપયોગ દર્શાવો

#### જવાબ

##### બિલ્ટ-ઇન લિસ્ટ મેથડ્સ:

```
\#
fruits = [{apple}, {banana}, {cherry}, {apple}]
numbers = [3, 1, 4, 1, 5, 9, 2]

\#
fruits.append({date})                      #
fruits.insert(1, {avocado})                 #
fruits.remove({apple})                     # occurrence
last\_fruit = fruits.pop()                  #
fruits.clear()                            #

\#
fruits = [{apple}, {banana}, {apple}, {cherry}]
count = fruits.count({apple})            # occurrences
index = fruits.index({banana})          #

\#
numbers.sort()                           # in place
numbers.reverse()                         # in place
sorted\_copy = sorted(fruits)           #

\#
more\_fruits = [{grape}, {orange}]
fruits.extend(more\_fruits)             #
```

##### લિસ્ટ મેથડ્સ સારાંશ:

ક્રેટેગરી	મેથડ	હેતુ	પરત કરે	મૂળ સુધારે
ઉમેરો	append(x)	અંતે આઈટમ ઉમેરો	None	હા
ઉમેરો	insert(i, x)	પોર્ઝિશન પર ઇન્સર્ટ કરો	None	હા
ઉમેરો	extend(list)	બહુવિધ આઈટમ્સ ઉમેરો	None	હા
દૂર કરો	remove(x)	પ્રથમ x દૂર કરો	None	હા
દૂર કરો	pop(i)	ઇન્ડેક્સ પર દૂર કરો	દૂર કરેલ આઈટમ	હા
દૂર કરો	clear()	બધું દૂર કરો	None	હા
શોધ	index(x)	પોર્ઝિશન શોધો	ઇન્ડેક્સ	ના

શોધ	count(x)	occurrences ગણો	કાઉન્ટ	ના
સોર્ટ	sort()	in place સોર્ટ કરો	None	હા
સોર્ટ	reverse()	કમ ઉલટાવો	None	હા
કોપી	copy()	શેલો કોપી	નવી લિસ્ટ	ના

### યવહારિક ઉદાહરણો:

```
\#
cart = []
cart.append({ })
cart.extend([{ }, { }, { }])
print(f"      : \{len(cart)\}")

if { } in cart:
    cart.remove({ })
    print("      ")

cart.sort()
print(f"      : \{cart\}")

મુખ્ય ઉપયોગો:
• ડેટા મેનેજમેન્ટ: આઇટમ્સ ઉમેરો, ફૂર કરો, ગોઠવો
• શોધ ઓપરેશન્સ: તત્ત્વો શોધો અને ગણો
• સોર્ટિંગ: ડેટાને કમમાં ગોઠવો
```

### મેમરી ટ્રીક

“લિસ્ટ મેથ્ડ્સ: ઉમેરો, ફૂર કરો, શોધો, સોર્ટ, કોપી”

## પ્રશ્ન 5(અ OR) [3 ગુણ]

ઉદાહરણ આપીને string ને કેવી રીતે create અને traverse કરવી તે સમજાવો.

### જવાબ

#### સ્લિંગ સર્જન અને ટ્રાવર્સલ:

```
\#
string1 = "Hello World"      #
string2 = {Python Programming} #
string3 = """  {-  }
"""\n

\#
text = "Python"

\#      1: for
for char in text:
    print(char, end=" ")
print()

\#      2:
for i in range(len(text)):
    print(f"\{text[i]\} \{i\}  ")

\#      3: enumerate
for index, char in enumerate(text):
    print(f" \{index\}: \{char\}")


ટ્રાવર્સલ પદ્ધતિઓનું ટેબલ:
```

પદ્ધતિ	સિન્ટેક્સ	ઉપયોગ કરું
ડાયરેક્ટ	for char in string:	સાદી કેરેક્ટર એક્સેસ
ઇન્ડેક્સ	for i in range(len(s)):	પોઝિશન માહિતી જોઈએ
Enumerate	for i, char in enumerate(s):	ઇન્ડેક્સ અને કેરેક્ટર બને

#### મુખ્ય કોન્સ્ટ્રુક્ટ્સ:

- અપરિવર્તનીય: સ્ટ્રિંગ્સ બદલી શકતી નથી
- Iterable: કેરેક્ટર્સમાં લૂપ કરી શકાય
- ઇન્ડેક્સિંગ: વ્યક્તિગત કેરેક્ટર્સ એક્સેસ કરી શકાય

#### મેમરી ટ્રીક

"સ્ટ્રિંગ્સ: બનાવો, લૂપ, એક્સેસ"

### પ્રશ્ન 5(બ OR) [4 ગુણ]

સ્ટ્રિંગ પર કરી શકાય તેવી કામગીરીની યાદી બનાવો. કોઈપણ 2 કામગીરી માટે કોડ લખો

#### જવાબ

##### સ્ટ્રિંગ ઓપરેશન્સ:

```
\#
text = "Python Programming"

\#    1:
first\_name = " "
last\_name = " "
full\_name = first\_name + " " + last\_name
formatted = f" , {full\_name}!"
print(f" : {full\_name}")
print(f" : {formatted}")

\#    2:
sentence = "python programming"
title\_case = sentence.title()
upper\_case = sentence.upper()
words = sentence.split()
print(f" : {title\_case}")
print(f" : {upper\_case}")
print(f" : {words}")
```

##### સ્ટ્રિંગ ઓપરેશન્સ ટેબલ:

ક્રેટેગરી	ઓપરેશન	ઉદાહરણ	પરિણામ
જોડાણ	કન્કેનેશન	"Hello" + " World"	"Hello World"
કેસ	upper()	"hello".upper()	"HELLO"
કેસ	lower()	"HELLO".lower()	"hello"
કેસ	title()	"hello world".title()	"Hello World"
સ્પ્લિટ	split()	"a,b,c".split(",")	[ 'a', 'b', 'c' ]
રિપ્લેસ	replace()	"hello".replace("l","x")	"hexxo"
સ્ટ્રિપ	strip()	" hello ".strip()	"hello"
શોધ	find()	"hello".find("e")	1

##### મુખ્ય લક્ષણો:

- અપરિવર્તનીય: ઓપરેશન્સ નવી સ્ટ્રિંગ્સ પરત કરે
- મેથડ રેટનિંગ: બહુવિધ ઓપરેશન્સ કંબાઇન કરો
- લવચીક: ઘણા બિલ્ટ-ઇન ઓપરેશન્સ ઉપલબ્ધ

## પ્રશ્ન 5(ક OR) [7 ગુણ]

સ્ટ્રિંગની વિવિધ બિલ્ટ-ઇન methods ની સૂચિ બનાવો અને ઉપયોગ દર્શાવો.

## જવાબ

## બિલ્ટ-ઇન સ્ટ્રિંગ મેથ્ડ્સ:

```
\#
text = " Python Programming Language "
sample = "Hello World Programming"

\#
print(f" : {}\\{text\\}{\"})
print(f"upper(): \\{text.upper()\\}")
print(f"lower(): \\{text.lower()\\}")
print(f"title(): \\{text.title()\\}")
print(f"capitalize(): \\{text.capitalize()\\}")
print(f"swapcase(): \\{{Hello}.swapcase()\\}")

\#
print(f"strip(): {}\\{text.strip()\\}{\"})
print(f"lstrip(): {}\\{text.lstrip()\\}{\"})
print(f"rstrip(): {}\\{text.rstrip()\\}{\")

\#
print(f"find({Python}): }\\{text.find({Python})\\}")
print(f"count({o}): }\\{sample.count({o})\\}")
print(f"startswith({ Py}): }\\{text.startswith({ Py})\\}")
print(f"endswith({ge }): }\\{text.endswith({ge })\\")

\#
test\_string = "Python123"
print(f"isalpha(): \\{{Python}.isalpha()\\}")
print(f"isdigit(): \\{{123}.isdigit()\\}")
print(f"isalnum(): \\{test\_string.isalnum()\\}")

\#
words = sample.split()
joined = "-".join(words)
print(f"split(): \\{words\\}")
print(f"join(): \\{joined\\}")

\#
replaced = sample.replace("World", "Universe")
print(f"replace(): \\{replaced\\}")
```

## સ્ટ્રિંગ મેથ્ડ્સ વર્ગીકરણ:

કેટેગરી	મેથ્ડ્સ	હેતુ	ઉદાહરણ
કેસ	upper(), lower(), title(), capitalize()	કેસ બદલો	"hello".upper() → "HELLO"
હાઇટસ્પેસ	strip(), lstrip(), rstrip()	સ્પેસીસ દૂર કરો	" hi ".strip() → "hi"
શોધ	find(), index(), count()	સબસ્ટ્રિંગ શોધો	"hello".find("e") → 1

ચેક	<code>startswith()</code> , <code>endswith()</code>	સ્ટ્રીંગ અંત ટેસ્ટ કરો	"hello".startswith("h") → <i>True</i>
ટાઇપ ચેક	<code>isalpha()</code> , <code>isdigit()</code> , <code>isalnum()</code>	કોરેક્ટર પ્રકાર	"123".isdigit() → <i>True</i>
સ્પલટ/જોઇન	<code>split()</code> , <code>join()</code>	તોડો/જોડો	"a-b".split("-") → ['a', 'b']
રિપ્લેસ	<code>replace()</code>	ટેક્સ્ટ બદલો	"hi".replace("i", "o") → "ho"

### વાસ્તવિક જીવનના ઉદાહરણો:

```
\#
email = " USER@EXAMPLE.COM "
clean_email = email.strip().lower()
is_valid = "@" in clean_email and "." in clean_email
print(f"      : \{clean_email\}")
print(f"      : \{is_valid\}")

\#
user_input = "python programming"
formatted_title = user_input.title()
word_count = len(user_input.split())
print(f"      : \{formatted_title\}")
print(f"      : \{word_count\}")
```

### મુખ્ય ઉપયોગો:

- ડેટા કલીનિંગ: અનાયાસ સ્પેસીસ દૂર કરો, કેસ ઢીક કરો
- ટેક્સ્ટ પ્રોસેસિંગ: સર્ચ, રિપ્લેસ, સ્પલટ કન્ટેન્ટ
- વ૆લિડેશન: સ્ટ્રીંગ ફિર્મેટ અને કન્ટેન્ટ ચેક કરો
- ફોર્મેટિંગ: ડિસ્પ્લે માટે ટેક્સ્ટ તૈયાર કરો

### મેમરી ટ્રીક

“સ્ટ્રીંગ મેથ્ડ્સ: કેસ, સાફ્, ચેક, બદલો”