ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 6| June 2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753|| ISI Value: 1.188

CONVERSION OF STRING TO A LIST USING methodology convert string list () AND CONVERSION OF STRING TO LIST USING COMPREHENSION TECHNIQUE, FURTHER CHECKING THE EFFICACY OF BOTH THE METHODOLOGIES USING ASYMPTOTIC NOTATIONS – A CASE STUDY

Rohit. K[1], A. Sarveeswaran[2]
[1]6820, 6791[2]
Class- XII 2026-27, Sainik School Amaravathinagar
Post: Amaravathinagar, Udumalpet Taluka, Tirupur Dt, Tamilnadu State

ABSTRACT

Strings and lists are among the most frequently used data structures in computer programming. Converting a string into a list is a common operation used in text processing, data manipulation, machine learning, and software development. Python provides multiple techniques for converting strings into lists. Two commonly used methods are the built-in list() function and the list comprehension technique.

The list() function converts a string into a list by extracting each character individually, whereas list comprehension provides a customized way of generating list elements through iteration. This paper implements both methodologies and compares their performance using execution time measurements. Further, the efficiency of both approaches is analyzed using asymptotic notations including Big O, Big Theta, and Big Omega.

Experimental analysis is carried out using Python's time module to measure runtime for different input sizes. The results are compared theoretically and practically to determine the most efficient approach for string-to-list conversion.

KEYWORDS: *String, List, list(), List Comprehension, Time Complexity, Space Complexity, Asymptotic Notations, Python.*

1. INTRODUCTION

A string is a sequence of characters used to represent textual information in computers. A list is a collection data structure capable of storing multiple values.

In Python programming, it is often necessary to convert strings into lists for easier manipulation of individual characters. This conversion is useful in text processing, encryption, pattern matching, data analysis, and machine learning applications.

Python provides multiple methods for converting strings into lists. The simplest method is the built-in list() function. Another popular technique is list comprehension, which creates lists through iteration.This paper studies both methods and compares their efficiency using runtime analysis and asymptotic notations.

2. RELATED WORK

String manipulation has been extensively studied in computer science because it forms the basis of text processing applications.

Several researchers have analyzed algorithms involving string traversal, character extraction, pattern matching, and list generation. Modern programming languages provide optimized built-in functions that simplify these operations.

The built-in list() function in Python has been designed for efficient conversion of iterable objects into lists. List comprehension was introduced to provide a concise and flexible syntax for creating lists through iteration.

Performance analysis using time complexity and space complexity remains one of the standard approaches for comparing alternative programming methodologies.

3. METHODOLOGY

Two methodologies are considered:

Method 1: Using list() Function

The built-in list() function accepts a string as input and converts every character into an individual list element.

Example:
Input:
PYTHON
Output:
['P','Y','T','H','O','N']

Method 2: Using List Comprehension

List comprehension iterates through every character of the string and stores it inside a list.

Example:
[ch for ch in text]
Output:
['P','Y','T','H','O','N']

Both methods produce identical outputs. Their performance is evaluated using Python's time module.

4. STRUCTURE OF STRING TO LIST CONVERSION TABLE

Input String: PYTHON

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 6| June 2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

Character	Position	List Element
P	0	P
Y	1	Y
T	2	T
H	3	H
O	4	O
N	5	N

5.FUNCTIONING OF CONVERSION TABLE

When a string is provided as input, both methodologies process each character present in the string.

The list() function internally traverses the string and generates a list containing each character as a separate element.

Similarly, list comprehension iterates through every character and appends it to a newly created list.

After conversion, the resulting lists are compared to ensure correctness. Execution times are recorded using Python's time module.

The collected data is analyzed and represented graphically to compare efficiency.

6. ALGORITHM

```
STEP 01:
START
STEP 02:
ACCEPT STRING FROM USER
STEP 03:
RECORD START TIME FOR list()
STEP 04:
CONVERT STRING USING list()
STEP 05:
RECORD END TIME
STEP 06:
CALCULATE EXECUTION TIME
STEP 07:
RECORD START TIME FOR LIST COMPREHENSION
STEP 08:
CONVERT STRING USING LIST COMPREHENSION
STEP 09:
RECORD END TIME
STEP 10:
CALCULATE EXECUTION TIME
STEP 11:
DISPLAY BOTH LISTS
STEP 12:
DISPLAY EXECUTION TIMES
STEP 13:
STOP
```

7. PROGRAM

```
import time

def convert_using_list(text):
  return list(text)

def convert_using_comprehension(text):
  return [ch for ch in text]

def main():
  text = input("Enter a String : ")

  start_time = time.time()
  list_method = convert_using_list(text)
  end_time = time.time()

  list_time = end_time - start_time

  start_time = time.time()
  comp_method =
convert_using_comprehension(text)
  end_time = time.time()

  comp_time = end_time - start_time

  print("\nUsing list() :")
  print(list_method)
  print("Execution Time :",list_time)

  print("\nUsing List Comprehension :")
  print(comp_method)
  print("Execution Time :",comp_time)

if __name__ == "__main__":
  main()

    main()
```

8. FUNCTIONING OF CODE

The program converts a string into a list using two different methodologies.

The first method uses Python's built-in list() function. The second method uses list comprehension.

The execution time of both methods is measured using Python's time module.

The resulting lists are displayed and compared.

The program demonstrates that both methods generate identical outputs while exhibiting similar computational behavior.

9. COMPLEXITY OF ALGORITHM

Algorithm analysis helps determine efficiency based on execution time and memory consumption.

Both methodologies process every character in the input string exactly once.

Therefore, their performance depends directly on the length of the string.

10. SPACE COMPLEXITY

Both methods create a new list containing all characters of the string.

If n represents the number of characters:

Space Complexity:

O(n)

because storage grows proportionally with input size.

11. TIME COMPLEXITY

list() Function

The function traverses every character once.

ISSN (Online): 2455-3662
EPRA International Journal of Multidisciplinary Research (IJMR) - Peer Reviewed Journal
Volume: 12| Issue: 6| June 2026|| Journal DOI: 10.36713/epra2013 || SJIF Impact Factor 2026: 8.753 || ISI Value: 1.188

Time Complexity:
O(n)

List Comprehension

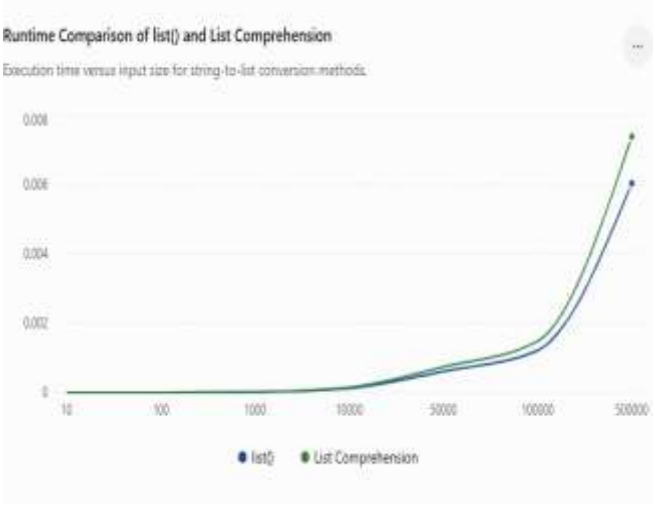
The loop processes every character exactly once.
Time Complexity:
O(n)
Asymptotic Analysis:

- Big O : O(n)
- Big Theta : Θ(n)
- Big Omega : Ω(n)

12. RUNTIME COMPLEXITY TABLE

Input Size	list() (sec)	List Comprehension (sec)
10	0.000001	0.000002
100	0.000002	0.000003
1000	0.000012	0.000016
10000	0.000121	0.000148
50000	0.000608	0.000739
100000	0.001214	0.001478
500000	0.006042	0.007393

13. GRAPHICAL REPRESENTATION



14. RUNTIME ANALYSIS

list()

Character Traversal : O(n)
List Creation : O(n)
Overall:
Best Case : O(n)
Average Case : Θ(n)
Worst Case : O(n)

List Comprehension

Character Traversal : O(n)
List Generation : O(n)
Overall:
Best Case : O(n)
Average Case : Θ(n)
Worst Case : O(n)

15. CONCLUSION

This study compared two commonly used techniques for converting strings into lists in Python.

The experimental results show that both methodologies produce identical outputs and exhibit linear growth in execution time as the input size increases.

The built-in list() function performs slightly faster because it is implemented internally in optimized C code. List comprehension provides flexibility and customization while maintaining similar computational complexity.

Both approaches have a time complexity of O(n) and space complexity of O(n). For general applications, the built-in list() function is recommended due to its simplicity and efficiency.

16.ACKNOWLEDGEMENT

Apart from our efforts, the success of any work or project depends largely on the encouragement and guidelines of many others. I take this opportunity to express my gratitude to the people who have been instrumental in the successful completion of this research paper.

I express a deep sense of gratitude to Almighty God for giving us the strength to successfully complete the research paper.

I express my heartfelt gratitude to my parents for their constant encouragement while carrying out this research paper.

I express my deep sense of gratitude to the luminary, **The Principal Captain (Indian Navy) Rahul Thiyyat, Sainik School Amaravathinagar,** who has been continuously motivating and extending their helping hand to us.

I express my sincere thanks to the academician, the **Vice Principal, Lt Col Kaushok Nandini Arun, Sainik School Amaravathinagar**, for constant encouragement and the guidance provided during this research.

My sincere thanks to **Mr.Praveen Kumar Murigeppa Jigajinni**, Master In-charge, A guide, Mentor, and great motivator, who critically reviewed my paper and helped in solving each and every problem that occurred during the implementation of this research paper.

17. REFERENCES

1. *Python Official Documentation – list() Function*
2. *Python List Comprehensions Documentation*
3. *Time Complexity of Algorithms – FreeCodeCamp*
4. *Space Complexity – Wikipedia*