Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #41
String and it's types

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M (Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Strings in Python

Introduction

String is a **Non-primitive Linear Data Structure**. It is a **collection of characters or a sequence of characters surrounded by quotes**. Let us try to understand why string is required through an example.

The requirement is to store the address of an employee or to store the srn of an employee. This data is a mixture of alphabets, spaces, punctuation marks like #. If we want to go with a list of characters, it is too much of headache when we need to change it. Hence the string type is really helpful. Also, creating strings is as straight forward as assigning a value to a variable in python.

Note: There is no character type in python.

Characteristics of Strings:

- It has **0 or more characters.**
- It allows duplicate characters within it.
- There is no **name for each element** in a string separately.
- Elements are accessed using **indexing operation or by subscripting**.
- String is **indexable** - **Index always starts with 0**. This is known as **zero based indexing.** Negative indices are also supported.
- String is **immutable**– Cannot grow or shrink as its size is fixed.
- String is **iterable –** Can get one element at a time.
- It has number of **built-in functions to manipulate and return the new string.**
- Strings cannot be nested.

Syntactically, characters of the string must be inside **quotes. Types of it are as below.**

- **Single quoted strings**
- **Double quoted strings**
- **Triple – Single quoted strings**
- **Triple – Double quoted strings.**

Single quoted strings and Double quoted strings

There is no difference between single and double quoted strings in python. **Escape sequences like \t, \n are expanded in both**. We can use double quotes in a single quoted string and single quote in double quoted string without escaping. These **strings can span just a line – cannot span multiple lines.**

```
s1 = 'I love PYthon'
print(type(s1))
s2 = "I love PYthon"
print(type(s2))
s3 = "'I love PythoN'"
print(s3)
s4 = '"I love PythoN"'
print(s4)
s5 = "I love\nPythoN"
print(s5)
s6 = 'I love\tPythoN'
print(s6)
```

```
C:\Users\Dell>test.py
<class 'str'>
<class 'str'>
'I love PythoN'
"I love PythoN"
I love
PythoN
I love	PythoN
```

Triple Single quoted strings and Triple Double quoted strings

We can create a **string spanning multiple lines** by using either three single quotes or three double quotes as delimiters. These strings are also **used for documentation purpose.** . **Escape sequences like \t, \n are also expanded in this.**

```
s1 = """I love PYthon"""
print(type(s1), s1)
s2 = '''I love PYthon'''
print(type(s2),s2)
s3 = "    I      love        PythoN   "
print(s3)
s4 = """    I      love        PythoN   """
print(s4)
s5 = '''I love\nPythoN'''
print(s5)
s6 = 'I love\tPythoN'
print(s6) #To span multiple lines we need to use \ in every line between single and double quotes
```

```
s7 = "I \
Love \
python \
"
print(s7)  # stored as one line
s8 = """I
Love python.
what about        you?"""
#s8 is spanned over multiple lines
print(s8)
```

```
C:\Users\Dell>test.py
<class 'str'> I love PYthon
<class 'str'> I love PYthon
    I       love          PythoN
    I       love          PythoN
I love
PythoN
I love  PythoN
I Love python
I
Love python.
what about          you?

C:\Users\Dell>
```

There is a special type of string called as **raw string -** There are cases where **the escape sequence should not be expanded**. In such cases, we prefix r to the string literal – it becomes a raw string.

```
>>> print ("this \t is python")
this     is python
>>> print (r"this \t is python")
this \t is python
>>>
```

Accessing individual elements of the string

Consider, s1 = "python programming"

The index begins with 0. To access h, we can use **s1[3].**If we want to use negative index, -1 is the index for the last element of the string. **Max value of index is length of the string – 1. Accessing outside this index results in Index Error. The last element of the list** can be accessed **using -1 as the index or len(s1) – 1**

```
>>> s1 = "python programming"
>>> s1[3]
'h'
>>> s1[-1]
'g'
```

```
>>> s1[-len(s1)]
'p'
>>> s1[20]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

A. **Common functions that can be applied are len(), max(), min().** These functions are self-explanatory

```
>>> sum(s1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> len(s1)
18
>>> max(s1)
'y'
>>> min(s1)
' '
>>>
```

B. **Common operators that can be applied are +, *, in, not in, slicing operator(:) within index operator([]) and relational operators**

- **+ ->**The **Concatenate operation** is used to merge two string and creates a new string.
- *** ->**The **Repetition operation** allows multiplying the string n times and create a new string.
- **in and not in ->**Used to **find whether the particular element exists in the string or not**.

```
>>> s1
'python programming'
>>> s2 = "is my favourite"
>>> s1+s2
'python programmingis my favourite'
>>> s1*3
'python programmingpython programmingpython programming'
>>> 'o' in s1
True
>>> 'z' not in s1
True
>>>
```

- **Relational operators ->**Compares the corresponding characters until a mismatch or one or both ends.

```
>>> s1
'python programming'
>>> s2
'is my favourite'
>>> s1 == s2
False
>>> s2 != s2
False
>>> s1 > s2
True
>>> s1 >= s2
True
>>> s1 = "python"
>>> s2 = "pyth"
>>> s1 > s2
True
>>> s1 >= s2
True
```

- **Slicing operator [:] ->**Used to **create a new string based on the indices of the existing** string. Works same as slice operator on lists.

Display the sliced string from the existing string

```
>>> s1 = "i am from pes university"
>>> s2 = s1[::]    # id is same as string is immutable when trying to copy all characters from the string
>>> id(s1)
2314690777200
>>> id(s2)
2314690777200
>>> s3 = s1[1:16]    #creates a copy. Id is different for s3 and s1
>>> s3
' am from pes un'
>>> id(s3)
2314691229680
>>> id(s1)
2314690777200
>>> s4 = s1[::-1]    # creates a copy with reversed string
>>> s4,id(s4)
('ytisrevinu sep morf ma i', 2314691289984)   #observe the i. different from s1
>>> s1, id(s1)
('i am from pes university', 2314690777200)  # string is immutable
>>>
```

-END-