BCSE101E
Computer Programming: Python

PROF. SARAH PRITHVIKA P.C.
ASSISTANT PROFESSOR
SCOPE

Strings

Strings

- Immutable (unchangeable) sequence of characters
- A string literal uses quotes
- 'Hello' or "Hello" or '"Hello"'
- For strings, + means “concatenate”
- When a string contains numbers, it is still a string
- We can convert numbers in a string into a number using int()

```
s = "123"
num = int(s)
print(num)      # Output: 123
print(type(num))  # Output: <class 'int'>
```

Immutable

String is an “**immutable sequence**”

S = 'spam'

print(id(S))

S= ' VIT’

print(id(S))

OUTPUT

```
2110878431216
2110878431344
```

When you “change” string variable (e.g., from 'spam' to 'VIT'), you are not updating the old string but rather creating a new string object in memory and making S point to it.

Hence, the two print(id(S)) output different IDs

Changing Strings

String - “**immutable sequence**”

Immutable - you cannot change a string in place

```
>>> S = 'spam'
>>> S[0] = 'x'                  # Raises an error!
```

TypeError: 'str' object does not support item assignment

Example Strings

- Single quotes: 'spa"m'

```
>>>S= 'spa"m'

>>>print(S)

spa"m
```

- Double quotes: "spa'm"

```
>>> S="spa'm"

>>> print(S)

spa'm
```

Triple Quotes

Triple Quotes “”” or ‘’’ allow you to write strings that span multiple lines.

```
a = """Python was designed for readability, and has some similarities to the
English language with influence from mathematics. Python uses new lines to
complete a command, as opposed to other programming languages which often use
semicolons or parentheses. Python relies on indentation, using whitespace,
to define scope; such as the scope of loops, functions and classes."""
print(a)
```

Output

```
Python was designed for readability, and has some similarities to the
English language with influence from mathematics. Python uses new lines to
complete a command, as opposed to other programming languages which often use
semicolons or parentheses. Python relies on indentation, using whitespace,
to define scope; such as the scope of loops, functions and classes.
```

Example Strings

Triple Quotes “”” or ‘’’

- When placed at the beginning of a function, class, or module, triple-quoted strings serve as **documentation**.
- Triple-quoted strings **can be used like comments** because Python will ignore them if not assigned or used

```
'''
This looks like a comment,
but it's actually a string that is ignored.
'''
```

Example Strings

Eg1

txt = "Hello\nWorld!“ #insert new line.

Output

Hello
World!

Eg2

txt = "Hello\tWorld!"
print(txt)

Output

Hello	World!

To insert characters that are illegal in a string, use an escape character.

Eg 3

a='It's a nice day outside.' #Error, since we have 3 single quotes
a='It\'s a nice day outside.' #No error, since we have used escape character \
print(a)

Eg 4

txt = txt ="Who\"s at the door?“#3 double quotes, but no error, since we have used escape character \

Escape Sequences

```
s = 'a\nb\tcs‘
print(len(s))
for x in s:
    print(x)
```

Output

```
6
a

b

c
s
```

Backslash in Strings

- if Python does not recognize the character after a \ as being a valid escape code, it simply keeps the backslash in the resulting string:

```
>>> x = "C:\py\code"

>>> x
'C:\py\code'
>>> len(x)
10
```

Raw strings

Python raw string is created by prefixing a string literal with ‘r’ or ‘R’.

Python raw string treats backslash (\) as a literal character.

```
>>> s = "C:\new\text.dat"
>>> print(s)
C:
ew      ext.dat
>>> s1 = r"C:\new\text.dat"
>>> print(s1)
C:\new\text.dat
```

Double slash \\ can also be used

```
>>>s2 = "C:\\new\\text.dat“
>>>print(s2)
C:\new\text.dat
```

Length of a String

```
>>> s = ‘abc'
>>> s
‘abc’
>>> len(s)
3
>>> print(s)
abc
```

Basic Operations

```
>>> 'Ni!' * 4

'Ni!Ni!Ni!Ni!‘

>>> print('-' * 10)                # 10 dashes

----------

>>>x = "Python is "

>>>y = "awesome"

>>>z =  x + y

print(z)

Python is awesome
```

Basic Operations

If you try to combine a string and a number, Python will give you an error:

TypeError: unsupported operand type(s) for +: 'int' and 'str'

```
>>>x = 5
>>>y = “6"
>>>print(x + int(y)) #convert y to int
```

Using ‘in’ Operator in Strings

```
>>> myjob = "hacker"
>>> "k" in myjob                    # Found
True
>>> "z" in myjob                    # Not found
False
>>> 'spam' in 'abcspamdef'
# Substring search, no position returned
True
```

Counting

- Count the number of ‘a’

Example

```
word = 'Btechallbranches'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
#Output 2
```

Counting

```
>>>txt = "I love apples, apple are my favorite fruit"
>>>x = txt.count("apple")
>>>print(x)
2
```

string.count(value, start, end)

value - Required. A String. The string to value to search for

start - Optional. An Integer. The position to start the search. Default is 0

end - Optional. An Integer. The position to end the search. Default is the end of the string

Counting

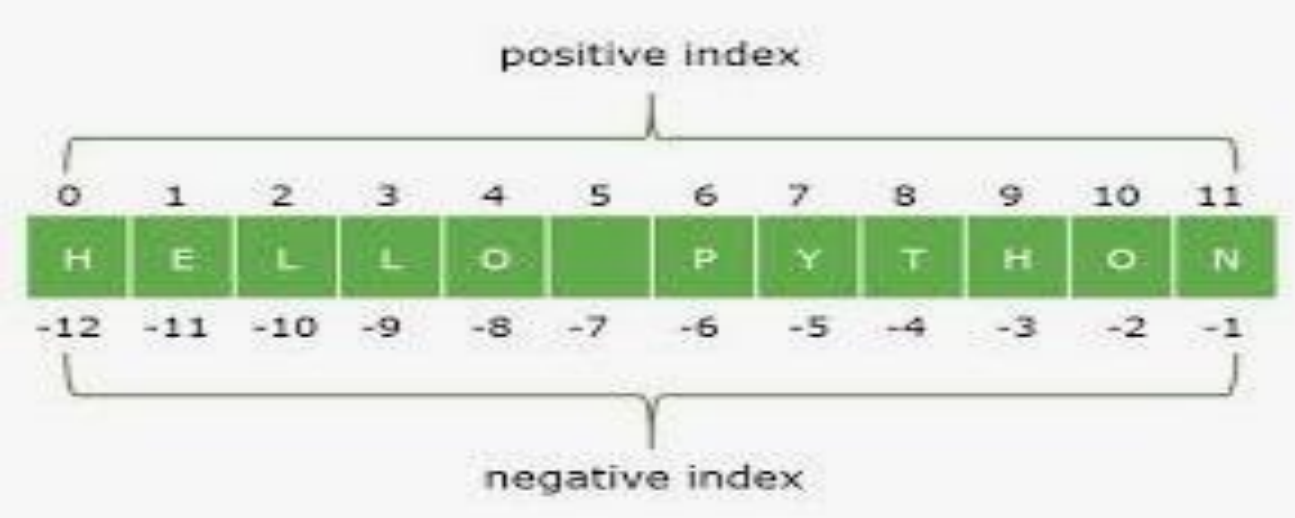
```
>>>txt = "I love apples, apple are my favorite fruit"

>>>x = txt.count("apple", 10, 24)

>>>print(x)

1
```

Indexing and Slicing

```
>>> S = 'spam'
```

Last character in the string has index -1 and the one before it has index -2 and so on

Indexing and Slicing



Indexing and Slicing

- Take one letter from a word at a time
- Use square bracket and give the index of the letter to be extracted
- Indexing can be done either from front or from end

```
>>> s='spam'
>>> S[0], S[−2]
('s', 'a')
```

Slicing

- Take a part of a word
- Square bracket with two arguments with a colon
- First value indicates the starting position of the slice and second value indicates the stop position of the slice
- Character at the stop position is not included in the slice

>>>s = ‘spam’

>>> **S[1:3]**

'pa’

Slicing

- If the second number is beyond the end of the string, it stops at the end.
- If we leave off the first or last number of the slice, it is assumed to be beginning or end of the string respectively

```
s = ‘spam’
print(s[:3])      #Output ‘spa'
print(s[1:])      #Output ‘pam'
print(s[:5])      #Output ‘spam'
```

Properties of Slicing

- S[1:3] - fetches items at index 1 up to but **not including 3**.
- S[1:] - fetches items at index **1 through the end**
- S[:3] - fetches items at index **0 up to but not including 3**
- S[:−1] - fetches items at index **0 up to but not including last item**
- S[:] - fetches items at index **0 through the end** —making a top-level copy of S

Extended slicing

- **X[I:J:K]** - means “extract all the items in X, from index **I through J−1, by K**.”
- Third limit, K, **defaults to +1**
- If you specify an explicit value it is used to skip items
- Extraction is reversed when negative value is given for K, -1
- Each time K-1 items are skipped

Extended slicing Examples

```
>>> S = 'abcdefghijklmnop'
>>> S[1:10:2]              # Skipping items
'bdfhj'
>>> S[::2]
'acegikmo‘
>>> S = 'hello'
>>> S[::−1]                # Reversing items
'olleh‘
>>>S = 'abcdefghijklmnop'
>>>print(S[-1::-2])
pnljhfdb
```

```
>>> str1 = '0123456789'
>>> str1[:]
'0123456789'
>>> str1[::2]
'02468'
>>> str1[::3]
'0369'
>>> str1[1::2]
'13579'
>>> str1[1:6:2]
'135'
>>>
```

```
>>>S = 'abcdefghijklmnop'
>>>print(S[-1::2])
p
```

String Conversion Tools

- >>> **"42" + 1**
- **TypeError:** Can't convert 'int' object to str implicitly
- >>> **int("42"), str(42)** # Convert from/to string
- (42, '42')
- int("42") + 1
- 43
- >>> "42" + str(1)
- '421'

ASCII (American Standard Code for Information Interchange)

ASCII is a character encoding standard that assigns numeric codes to characters so computers can store and process text.

Suppose you want to send a text message to your friend that reads, “Hello!” When you type this message on your phone or computer, each character is converted into its corresponding ASCII value. In this case, the ASCII values for “Hello!” are 72, 101, 108, 108, 111, and 33.

These numerical values are then translated into binary code, which is transmitted to your friend’s device. Upon receiving the message, their device converts the binary code back into ASCII values and finally displays the original text, “Hello!”

Character	ASCII
a	97
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
j	106
k	107
l	108
m	109

Character	ASCII
n	110
o	111
p	112
q	113
r	114
s	115
t	116
u	117
v	118
w	119
x	120
y	121
z	122

Character	ASCII
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77

Character	ASCII
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90

Character	ASCII
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57

Character code Conversions

- **ord () - Convert a single character** to its **underlying integer code** (e.g., its **ASCII** byte value)— this value is used to represent the corresponding character in memory.

- >>> **ord('s')**

  115

Character code Conversions

chr () – Does **inverse of ord**

```
>>> chr(115)
's'
```

Character code Conversions - Example

```
>>> c = '5'
>>> ord(c)
53
>>> x = chr(ord(c) + 1)
>>> x
'6'
```

Character code Conversions - Example

```
>>> ord('5') - ord('0')     # 53 - 48
5
```

int() Function

The int() function converts the specified value into an integer number.

int(value, base)

value - A number or a string that can be converted into an integer number

base - A number representing the number format. Default value: 10

```
>>>x = int("12")
>>>print(x)
12
```

```
num = 13
Str = '187'
r = int(Str) + num
print(res)
200
```

```
>>> int('1101', 2) # Convert binary to integer
13
```

Concatenation

```
>>>S1 = ‘Welcome’
>>>S2 = ‘Python’
>>>S3 = S1 + S2
>>>S3
'WelcomePython'
```

String.replace()

```
>>> S = 'splot'
>>> S = S.replace('pl', 'pamal')
>>> S
'spamalot'
```

Formatting Strings

>>> 'That is **%d %s** bird!' **% (1, ‘nice')**

That is 1 nice bird!

>>> 'That is {0} {1} bird! {2} {3}' .format(1, ‘nice‘, 23.4, ‘raj’)

'That is 1 nice bird! 23.4 raj'

Formatting Strings

```
print('That is {0} {1} bird! {2} {3}'.format(1, 'nice', 23.5, 'raj'))
print('That is {} {} bird! {} {}'.format(1, 'nice', 23.5, 'raj'))
print('That is {no} {adj} bird! {mark} {name}'.format(no=1, adj='nice',
mark=23.5, name='raj'))
 #Output
That is 1 nice bird! 23.5 raj
That is 1 nice bird! 23.5 raj
That is 1 nice bird! 23.5 raj
```

```
>>> greet = 'Hello Arun'
>>> zap = greet.lower()
>>> print (zap)
      hello arun


>>> print (‘Hi There'.upper())
            HI THERE
```

Searching a String

find() - finds the first occurrence of the substring

If the substring is not found, find() returns -1

```
>>> name = 'kumar‘
>>> pos = name.find(‘ma')
>>> print (pos)
2
```

Searching a String

```
>>> aa = “fruit”.find('z')
>>> print (aa)
-1
>>>txt = "Hello, welcome to my world."
>>>x = txt.find("welcome")
>>>print(x)
7
```

Searching a String

A=“Hello there, how are you?”

Pos = A.find(‘how’)

print(A[Pos:]) #Output how are you?

s="element"
print(s.find('e',3,6)) #Output 4

strip()

The strip() method in Python removes leading and trailing characters from a string.
By default, it removes whitespace (spaces, tabs, newlines) from both ends.

```
Str1='This is some special text This'
print(Str1.strip('This'))
```

```
===== RESTART: C:/Users/
 is some special text
>>>
```

```
sentence = ' hello  apple  '
print(sentence)
print(sentence.strip())
```

```
 hello  apple
hello  apple
```

string.rstrip()

rstrip() - Remove any white spaces at the end of the string.

```
>>>txt = "     banana     "

>>>x = txt.rstrip()

>>>print("of all fruits", x, " is my favorite")

of all fruits     banana is my favorite
```

string.rstrip()

string.rstrip(characters)

characters - Optional. A set of characters to remove as trailing characters

#Remove the trailing characters if they are commas, s, q, or w:

txt = **"banana,,,,,"**

x = txt.rstrip(",")

print(x)

banana

>>>txt = ",,,banana,,,,,ssqqqww....."

>>>x = txt.rstrip(",.qsw")

>>>print(x)

,,,banana

strip() vs rstrip() vs lstrip()

```
Str1='Some text.Some'

print(Str1.strip('Some'))

print(Str1.lstrip('Some'))

print(Str1.rstrip('Some'))
```

Output

```
 text.

 text.Some

Some text.
```

Other Common String Methods in Action

- Str = "A:Some text.A"
- position=Str.rfind('A')
- print(position)
- position=Str.find('A')
- print(position)
- Output
- 12
- 0

Other Common String Methods in Action

- >>> line = "The knights who say Ni!\n“
- >>> line**.isalpha()**
- **# Check if all the characters in the string are letters**
- False
- >>> line**.endswith('Ni!\n')**
- **#Check if the string ends with the given string**
- True
- >>> line**.startswith('The')**
- **#Check if the string starts with “The"**
- True

- txt = "50800"

```
x = txt.isdigit()

print(x)
```

- Output

- True

Other Common String Methods in Action

```
line = 'The knights who say MyVIT'

sub="MyVIT"

print(-len(sub))

print(line[-len(sub):])

print(line[-len(sub):] == sub)
```

Output

```
-5

MyVIT

True
```

Other Common String Methods in Action

```
print(line.endswith(sub))
```

Output

```
True
```

Other Common String Methods in Action

```
Str='vit'

print(Str.endswith(('e','t'))) #Output True

Str='vit'

print(Str.endswith(('e',’i'))) #Output False
```

Other Common String Methods in Action

expandtabs()- replace tab characters (\t) in a string with spaces

Str='text\ttext2\ttext3'

print(Str)

print(Str.expandtabs(20))

Output

```
text    text2   text3
text                text2                   text3
```

To check if all letters in a String are in Uppercase

- **isupper()** function is used to check if all letters in a string are in upper case.
- **Examples**

```
>>> 'a'.isupper()
False
>>> 'A'.isupper()
True
>>> 'AB'.isupper()
True
>>> 'ABc'.isupper()
False
```

To check if all letters in a String are in Lowercase

- **islower()** function is used to check if all letters in a string are in lower case.

- **Examples**

```
>>> 'a'.islower()
True
>>> 'aB'.islower()
False
```

To check if a sentence is in Title case

- **istitle()** method returns True if all words in a text start with a upper case letter, AND the rest of the word are lower case letter.
- **Examples**

```
>>> 'Apple Is A Tree'.istitle()
True


>>> 'Apple Is A tree'.istitle()
False
```

- **capitalize()**

The capitalize() method returns a string where the first character of first word is upper case, and the rest is lower case.

```
txt1='hello'
print(txt1.capitalize())
```

Output

Hello

- **capitalize()**

```
txt1='hello how are you'
print(txt1.capitalize())
```

Output

```
Hello how are you
```

swapcase()

```
Str1='This is some special text'
print(Str1.swapcase())
print(Str1.title())
```

Output

```
tHIS IS SOME SPECIAL TEXT
This Is Some Special Text
```

camelcase

Camel case is a casing style where words are joined together without any spaces, and each new word starts with a capital letter except for the first word. The name "camelCase" comes from the hump-like appearance of the capital letters

```
s= "hello world example"
words = s.split()
print(words)

# Capitalize each word except the first one and join them
print(words[0].lower() + ''.join(word.capitalize() for word in words[1:]))
```

Output

```
['hello', 'world', 'example']
helloWorldExample
```

camelcase

- **words[0].lower()**
- Takes the first word and converts it to lowercase.
- This is the start of the **camelCase** format.

- **''.join(word.capitalize() for word in words[1:])**
- Iterates over all words except the first one (words[1:]).

- word.capitalize() converts each of these words to **title case** (first letter uppercase, rest lowercase).

- ''.join(...) joins them into a single string **without spaces**.

PascalCase

Pascal case is similar to camel case, but unlike camel case, the first letter of each word is capitalized, such as: "PascalCase".

```
s= "hello world example"
words = s.split()
print(''.join(word.capitalize() for word in words))
```

Output

```
HelloWorldExample
```

- **center**

Str='vit'

print(Str.center(20))

vit

- **center**

Str='vit'

print(Str.center(20),'.')

```
>>>
========== RESTART: C:/Users
.........vit.........
>>>
```

Index()

```
Str='What did the astronauts discover'
Pos=Str.index('astronauts')
print(Pos)
print(Str[Pos:])
```

Output

13

astronauts discover

Note- if we use for a string that is not present index function will result in error. Str.index('x') will result in error, since x is not there in the string.

index() vs find()

Both index() and find() are identical in that they return the index position of the first occurrence of the substring from the main string. The main difference is that Python find() produces -1 as output if it is unable to find the substring, whereas index() throws a ValueError exception.

rindex()

```
Str = "B: Some text.B"
print(Str.rindex('B'))
print(Str.index('B'))
```

Output

```
13
0
```

```
Str=‘hello2'
print(Str.isalnum())
#Output True
Str=‘hello2!'
print(Str.isalnum())
#Output False
```

isspace()

Check if all the characters in the text are whitespaces.

```
txt = "    "
x = txt.isspace()
print(x)
#Output True
```

join()

```
s='-'
print(s.join(['text1','text2','text3']))
```

Output

```
text1-text2-text3
```

ljust()

- Left-aligns the string within the given width
- Pads extra spaces (or a specified character) on the **right**.

Str1=‘text’

print(Str1.ljust(20,’-’))

Output

text----------

rjust()

- Right-aligns the string within the given width.
- Pads extra spaces (or a specified character) on the **left**.

```
Str = 'text'
print(Str.rjust(20,'_'))
```

Output

```
________________text
```

partition()

```
Str = ‘a+b=c’
print(Str.partition(‘=‘))
```

Output

```
('a+b', '=', 'c')
```

rpartition()

```
Str = 'text=text2=text3'
print(Str.rpartition('='))
```

Output

```
('text=text2', '=', 'text3')
```

removeprefix()

```
Str = "What's up"
print(Str.removeprefix('What'))
```

Output

's up

```
Str = "WhatWhat's What up What"
print(Str.removeprefix('What'))
```

Output

What's What up What

removesuffix()

```
Str = "What's up"
print(Str.removesuffix('up'))
```

Output What's

```
Str = "What's up up up"
print(Str.removesuffix('up'))
```

Output

```
What's up up
```

split()

```
Str1='This is some special text'
print(Str1.split(sep=' '))
```

```
Str1='This is some special text'
print(Str1.split(maxsplit=2))
# maxsplit=2 means split only 2 times from the left.
```

```
['This', 'is', 'some', 'special', 'text']
```

```
Str1='This is some special text'
print(Str1.rsplit(maxsplit=2))
```

```
['This', 'is', 'some special text']
['This is some', 'special', 'text']
```

```
# maxsplit=2 → split two times from the right.
```

zfill()

The **zfill(width)** method in Python pads a numeric string on the **left with zeros** until it reaches the specified **width**.

```
Str1='This is some special text'

print(Str1.zfill(50))
```

Output

```
000000000000000000000000This is some special text
```

sorted()

```
input_sentence = "the quick brown fox jumps over the lazy dog"
words = input_sentence.split()
print(words)
print(sorted(words))
```

Output

```
['the', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
['brown', 'dog', 'fox', 'jumps', 'lazy', 'over', 'quick', 'the', 'the']
```