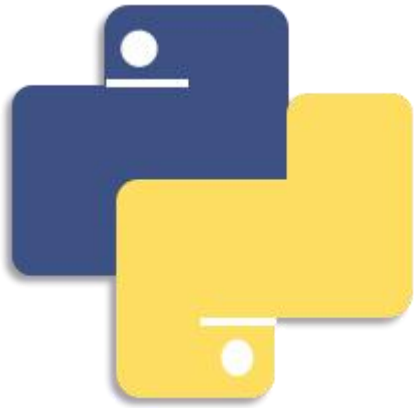

Python Programming - V

The Game of Strings

CONTENTS

ACCESS, UPDATE, ESCAPE

TRIPLE QUOTES

SPECIAL OPERATOR,
FORMAT OPERATOR

UNICODE STRINGS

METHODS

ENDSWITH

REPLACE

MAX MIN

IS a-to-z

MAKE TRANS

EXPAND TABS

CENTER

LENGTH

COUNT

ENCODE

DECODE

UNICODE

CAPATALIZE

JOIN

FIND

JUST
STRIP

INDEX

LOWER

R
FIND
INDEX
JUST
STRIP

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring. To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring.

Example :

```
s1 = "H3LL0 W0RLD"
s2 = "MITU SKILLOLOGIES"

print("s1[0]:", s1[0])
print("s2[3:10]:", s2[3:10])
```

Output:

```
s1[0]: H
s2[3:10]: U SKILL
```

Example : Updating Strings

```
s1 = 'TUSHAR'

print("Updated String: ",s1[:6]+' Sir')
```

Output:

```
Updated string is TUSHAR Sir
```

NOTE:
The String s1 remains unchanged until the operation performed, is stored into a variable.

Escape Characters

Example : BACKSPACE (\b)

```
s1 = 'MITU'

print(s1 + '\b' + 'SKILLOLOGIES')
```

Output:

```
MITSKILLOLOGIES
```

Example : NEWLINE (\n)

```
s1 = 'MITU'

print(s1,'\n',"Skillologies")
```

Output:

```
MITU
Skillologies
```

SPECIAL STRING OPERATORS

Considering a & b are two Strings

OPERATOR	DESCRIPTION	EXAMPLE
+	Concatenation - Adds values on either side of the operator	a + b
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a * 2
[]	Slice - Gives the character from the given index	a[1]
[:]	Range Slice - Gives the characters from the given range	a[1:4]
in	Membership - Returns true if a character exists in the given string	'H' in a
not in	Membership - Returns true if a character does not exist in the given string	'M' not in a

OPERATOR	DESCRIPTION	EXAMPLE
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	R"\t Hello"
%	Format - Performs String Formatting	Similar to Access specifier

EXAMPLE:

```
a = "MITU"
b = "Skillologies"

print("Concatenation:\t\t",       a + b)
print("Repetition:\t\t\t",        a * 3)
print("Slice:\t\t\t\t",           a[1])
print("Range Slice:\t\t",         b[3:9])
print("H In String var:\t",       'H'in a)
print("M Not In String var:",     'M'not in a)
print("R/r for Raw String:\t",    R"\t Hello")
```

Output:

```
Concatenation:                  MITUSkillologies
Repetition:                     MITUMITUMITU
Slice:                          I
Range Slice:                    llolog
H In String var:                False
M Not In String var:            False
R/r for Raw String:             \t Hello
```

STRING FORMAT SPECIFIER

One of Python's coolest features is the string format operator %. This operator is unique
to strings and makes up for the pack of having functions from C's printf() family.

Example :

```
print("___String Format Specifier___")
name = "Tejas"
surname = "Rawal"
age = 24
print("My Name is %s %s and I'm %d years old" %
(name, surname, age))
```

Output:

```
My Name is Tejas Rawal and I'm 24 years old
```

Example : Try the highlighted String Format Specifiers

```
print("\n\t___String Format Specifier___\n")
name = "Tejas"
surname = "Rawal"
age = 24

print("1> My Name is '%s' '%s' and I'm '%d' years
old" % (name, surname, age))

print("2> I am '%c' Researcher at MITU SKILLOLOGIES"
% (surname[1]))

print("3> This is my Age in OCTAL Value '%o'" % (age))
print("4> and this one in HEX '%x'" % (age))
print("5> Exponential Notation '%E'" % (age**2))
print("6> Floating point '%f'" % (age *
8552146632.11))

print("7> Short Floating Point '%G'" %
(age*8552146632.11))
```

Output:

```
___String Format Specifier___

1> My Name is 'Tejas' 'Rawal' and I'm '24' years old
2> I am 'a' Researcher at MITU SKILLOLOGIES
3> This is my Age in OCTAL Value '30'
4> and this one in HEX '18'
5> Exponential Notation '5.760000E+02'
6> Floating point '205251519170.639984'
7> Short Floating Point '2.05252E+11'
```

UNICODE STRING

In Python 3, all strings are represented in Unicode. In Python 2 are stored internally as 8 - bit ASCII, hence it is required to attach 'u' to make it Unicode. It is no longer necessary now.

1. CAPITALIZE

Capitalizes first letter of string

Syntax: capitalize()

Example :

```
name = "dyanEASHWARI"
print("name.capitalize() : ", name.capitalize())
```

Output:

```
name.capitalize() : Dyaneashwari
```

2. CENTER

Returns a string padded with fillchar with the original string centered to a total of width columns.

Syntax: center(width, fillchar)

Example :

```
name = "dyanEASHWARI"
print("name.center(30,'_') :", name.center(30, '_'))
```

Output:

```
name.center(30,'_') : __________dyanEASHWARI__________
```

3. COUNT

Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.

Syntax: center(width, fillchar)

Example :

```
name = "dyanEASHWARI"
print("name.count('A') : ", name.count('A'))
```

Output:

```
name.count('A') : 2
```

4. ENCODE

Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.

Syntax: encode(encoding='UTF-8',errors='strict')

Example :

```
import base64 as b6
name = "dyanEASHWARI"

print("\n", " ENCODING & DECODING ".center(40, '~'))

ename = b6.b64encode(name.encode('UTF-8', 'strict'))

print("\n>> ENCODE - b6.b64encode( name.encode('UTF-
8','strict') ) ", ename)
```

Output:

```
~~~~~~~~~~ ENCODING & DECODING ~~~~~~~~~~

>> ENCODE - b6.b64encode( name.encode('UTF-8','strict') )
b'Z3lhbkVBU0hXQVJJ'
```

5. DECODE

Decodes the string using the codec registered for encoding. encoding defaults to the default string encoding.

Syntax: decode(encoding='UTF-8',errors='strict')

Example :

```
import base64 as b6
name = "dyanEASHWARI"

print("\n", " ENCODING & DECODING ".center(40, '~'))

ename = b6.b64encode(name.encode('UTF-8', 'strict'))


print ename.decode('base64', 'strict')
```

Output:

```
 ~~~~~~~~~~ ENCODING & DECODING ~~~~~~~~~~

>> ENCODE - b6.b64encode( name.encode('UTF-8','strict') )
b'Z3lhbkVBU0hXQVJJ'

>> DECODE - b6.b64decode( ename.decode('UTF-8','ignore') )
b'gyanEASHWARI'
```

6. STARTS WITH

Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring str; returns true if so and false otherwise.

Syntax: startswith(str, beg=0,end=len(string))

Example :

```
print("\n", " STARTS WITH ".center(40, '~'))
stri = "This is some string for testing"

print("String:", stri, " \nStarts with: This in 0-3",
stri.startswith('This', 0, 3))

print("Starts with: This in 0-",
stri.startswith('This', 0))
```

Output:

```
~~~~~~~~~~~~~~ STARTS WITH ~~~~~~~~~~~~~~
String: This is some string for testing
Starts with: This in 0-3 False
Starts with: This in 0- True
```

7. ENDS WITH

Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.

Syntax: endswith(suffix, beg=0, end=len(string))

Example :

```
name = "dyanEASHWARI"

print("\n", " ENDSWITH ".center(40, '~'))

print("name.endswith('RI')\t\t ", name.endswith('RI'))

print("name.endswith('A', 2, 6)", name.endswith('A', 2,
6))
```

Output:

```
 ~~~~~~~~~~~~~~~~ ENDSWITH ~~~~~~~~~~~~~~~~
name.endswith('RI')           True
name2.endswith('A', 2, 6)     True
```

8. EXPANDING TABS

Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided.

Syntax: expandtabs(tabsize=8)

Example :

```
print("\n", " EXPANDING TABS ".center(40, '~'))

name1 = "UNICODE\tSTRINGS"

print("\t", name1.expandtabs(tabsize=20))
```

Output:

```
~~~~~~~~~~~~ EXPANDING TABS ~~~~~~~~~~~~
        UNICODE             STRINGS
```

9. FIND

Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

Syntax: find(str, beg=0 end=len(string))

Example :

```
print("\n", " FINDING ".center(40, '~'))
name2 = "Rubber baby buggy bumper"
print("name2.find('buggy', 5)  > ", name2.find('buggy',
5))

print("name2.find('buggy', 5)  > ", name2.find('buggy',
13))
```

Output:

```
 ~~~~~~~~~~~~~~~ FINDING ~~~~~~~~~~~~~~~
name2.find('buggy', 5)  >  12
name2.find('buggy', 5)  >  -1
```

10. INDEX

Same as find(), but raises an exception if str not found.

Syntax: index(str, beg=0, end=len(string))

Example :

```
print("\n", " INDEXING ".center(40, '~'))
name2 = "Rubber baby buggy bumper"
print(name2.index("Rubber"))

print(name2.index("bumper", 15))

print(name2.index("bumper",18,20))
```

Output:

```
 ~~~~~~~~~~~~~~~~ INDEXING ~~~~~~~~~~~~~~~~
0
18
Traceback (most recent call last):
  File "python/prog.py", line 7, in <module>
    print(name2.index("bumper",18,20))
ValueError: substring not found
```

11. IS ALPHANUMERIC, NUMERIC, DIGIT, DECIMAL & ALPHABETIC

(A) IS ALPHANUMERIC

Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.

Syntax: isalnum()

(B) IS NUMERIC

Returns true if a unicode string contains only numeric characters and false otherwise.

Syntax: isnumeric()

(C) IS ALPHABETIC

Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.

Syntax: isalpha()

(D) IS DIGIT

Returns true if the string contains only digits and false otherwise.

Syntax: isdigit()

(E) IS DECIMAL

The isdecimal() method checks whether the string consists of only decimal characters.
This method are present only on unicode objects.

Syntax: isdecimal()

Example :

```
# IS ALPHA NUM
print("1. CE1G2P017", "CE1G2P017".isalnum())
print("2. 17", "17".isalnum())
print("3. An Apple", "An Apple".isalnum())

# IS ALPHABETIC
print("\n1. ALPHA-BETA", "ALPHA-BETA".isalpha())
print("2. BETA", "BETA".isalpha())

# IS NUMERIC
print("\n11254", "11254".isnumeric())

# IS DIGIT
print("\n", " IS DIGIT ".center(40, '~'))
print("125463", "125463".isdigit())
print("1254a1", "1254a1".isdigit())

print("\n", " IS DECIMAL ".center(40, '~'))
print("12553.1".isdecimal())
print("4755133".isdecimal())
```

Output:

```
 ~~~ IS ALPHANUMERIC, NUMERIC & ALPHABETIC ~~~
1. CE1G2P017 True
2. 17 True
3. An Apple False

1. ALPHA-BETA False
2. BETA True

11254 True

 ~~~~~~~~~~~~~~~ IS DIGIT ~~~~~~~~~~~~~~~
125463 True
1254a1 False
```

12. IS LOWER / UPPER

(A) IS LOWER

Returns true if string has at least one cased character and all cased characters are in lowercase and false otherwise.

Syntax: islower()

(B) IS UPPER

Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

Syntax: isupper()

Example :

```
print("\n", " IS LOWER / UPPER ".center(40, '~'))
print("MITU", "MITU".islower())
print("MITU", "MITU".isupper())
```

Output:

```
 ~~~~~~~~~~~ IS LOWER / UPPER ~~~~~~~~~~~
MITU False
MITU True
```

13. IS SPACE

Returns true if string contains only whitespace characters and false otherwise.

Syntax: isspace()

Example :

```
print("\n", " IS SPACE ".center(40, '~'))
str1 = ' '
print(str1.isspace())
str1 = '_'
print(str1.isspace())
```

Output:

```
 ~~~~~~~~~~~~~~~~ IS SPACE ~~~~~~~~~~~~~~~~
True
False
```

14. IS TITLE

Returns true if string is properly "titlecased" and false otherwise.

Syntax: istitle()

Example:

```
print("\n", " IS TITLE ".center(40, '~'))
str1 = 'This Is A Title'
print(str1, ">", str1.istitle())
str1 = "This is a title"
print(str1, ">", str1.istitle())
```

Output:

```
 ~~~~~~~~~~~~~~~~ IS TITLE ~~~~~~~~~~~~~~~~
This Is A Title > True
This is a title > False
```

15. JOIN

The join() method returns a string in which the string elements of sequence have been joined by str separator.

Syntax: str.join(sequence)

Example:

```
print("\n", " JOIN ".center(40, '~'))
sequence = ("Apple", "Mango", "Strawberry")
sep = ", "
print(sequence, "Is now written with separator as:",
sep.join(sequence))
```

Output :

```
 ~~~~~~~~~~~~~~~~~ JOIN ~~~~~~~~~~~~~~~~~
('Apple', 'Mango', 'Strawberry') Is now written with
separator as: Apple, Mango, Strawberry
```

16. LENGTH

The len() method returns the length of the string.

Syntax: len(str)

Example:

```
name2 = "Rubber baby buggy bumper"

print("\n", " CAL LENGTH ".center(40, '~'))

print("Size of ", name2, "is", len(name2))
```

Output :

```
 ~~~~~~~~~~~~~~~ CAL LENGTH ~~~~~~~~~~~~~~~
Size of  Rubber baby buggy bumper is 24
```

17. JUSTIFICATION

(A) LJUST

Returns a space-padded string with the original string left-justified to a total of width columns.

Syntax: ljust(width[, fillchar])

(B) RJUST

Returns a space-padded string with the original string right-justified to a total of width columns.

Syntax: rjust(width,[, fillchar])

Example :

```
print("\n", " L R JUSTIFICATION ".center(40, '~'))
print("HELLO".ljust(10, "-"))
print("WORLD".rjust(10, "~"))
```

Output :

```
 ~~~~~~~~~~ L R JUSTIFICATION ~~~~~~~~~~
HELLO-----
~~~~~WORLD
```

18. CASE CONVERSION

(A) LOWER

Converts all uppercase letters in string to lowercase.

Syntax: ljust(width[, fillchar])

(B) UPPER

Converts lowercase letters in string to uppercase.

Syntax: rjust(width,[, fillchar])

Example :

```
print("\n", " LOWER UPPER CONVERSION ".center(40, '~'))
print("THIS WAS IN UPPERCASE".lower())
print("this was in lowercase".upper())
```

Output :

```
 ~~~~~~~~ LOWER UPPER CONVERSION ~~~~~~~~
this was in uppercase
THIS WAS IN LOWERCASE
```

19. STRING STRIPPING

(A) LSTRIP

Removes all leading whitespace in string.

Syntax: lstrip()

(B) RSTRIP

Removes all trailing whitespace of string.

Syntax: rstrip()

Example :

```
print("\n", " L-R STRIPPING ".center(40, '~'))
print("**MESSAGE**  >", "**MESSAGE**".lstrip('*'))
print("**MESSAGE**  >", "**MESSAGE**".rstrip('*'))
print("**MESSAGE**  >", "**MESSAGE**".strip('*'))
```

Output :

```
 ~~~~~~~~~~~~ L-R STRIPPING ~~~~~~~~~~~~~
**MESSAGE**  > MESSAGE**
**MESSAGE**  > **MESSAGE
**MESSAGE**  > MESSAGE
```

20. MAKE TRANS

Returns a translation table to be used in translate function.

Syntax: endswith(suffix, beg=0, end=len(string))

Example :

```
print("\n", " MAKING TRANSLATION ".center(40, '~'))
msg = "TEJAS is my code name"

intab = 'AEGHIOSX'
outtab = "436#!05*"

print("Message is ", msg, "Translated in CODE WRD as
\n", msg.maketrans(intab, outtab))

transl8 = msg.maketrans(intab, outtab)

print("Let's Translate using .translate()",
msg.translate(transl8))
```

Output :

```
Message is  TEJAS is my code name Translated in CODE WRD
as
 {65: 52, 83: 53, 69: 51, 71: 54, 72: 35, 73: 33, 88: 42,
79: 48}

Let's Translate using .translate() T3J45 is my code name
```

21. MAX MIN CHARACTER

(A) MAX

Returns the max alphabetical character from the string str.

Syntax: max(str)

(B) MIN

Returns the min alphabetical character from the string str.

Syntax: min(str)

Example :

```
print("\n", " MAX & MIN CHAR ".center(40, '~'))
alphabet = "AtoZ"

print(alphabet, "'s MAX character is", max(alphabet),
"and MIN is", min(alphabet))
```

Output :

```
 ~~~~~~~~~~~~ MAX & MIN CHAR ~~~~~~~~~~~~
AtoZ 's MAX character is t and MIN is A
```

22. REPLACE

Replaces all occurrences of old in string with new or at most max occurrences if max given.

Syntax: replace(old, new [, max])

Example :

```
print("\n", " REPLACING ".center(40, '~'))

print("BEFORE:", name2)
print("AFTER:", name2.replace("buggy", "'buddy'"))
```

Output :

```
 ~~~~~~~~~~~~~~~ REPLACING ~~~~~~~~~~~~~~~
BEFORE: Rubber baby buggy bumper
AFTER: Rubber baby 'buddy' bumper
```

23. FINDING FROM L & R

(A) FIND

Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

Syntax: find(str, beg=0 end=len(string))

(B) R FIND

Same as find(), but search backwards in string.

Syntax: rfind(str)

Example :

```
print("\n", " FINDING FROM R ".center(40, '~'))
str1 = "Linux is an Open Source System Software"

print(str1, "Finding: Source Located at",
str1.rfind('Source', 18))

print(str1, "Finding: Source Located at",
str1.find('nux'))
```

Output :

```
~~~~~~~~~~~~ FINDING FROM R ~~~~~~~~~~~~
Linux is an Open Source System Software Finding: Source
Located at -1
Linux is an Open Source System Software Finding: Source
Located at 2
```

24. RINDEX

Same as index(), but search backwards in string.

Syntax: rindex(str, beg=0, end=len(string))

Example :

```
print("\n", " REVERSE INDEXING ".center(40, '~'))
str1 = "Being a Researcher is cool"

print(str1, "\n Finding last INDEX of 'Researcher'",
str1.rindex('Researcher', 7))
```

Output :

```
 ~~~~~~~~~~~~ REVERSE INDEXING ~~~~~~~~~~~~~
Being a Researcher is cool
 Finding last INDEX of 'Researcher' 8
```

25. SPLIT

Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.

Syntax: split(str="", num=string.count(str))

Example :

```
print("\n", " SPLITTING STRING ".center(40, '~'))
str1 = "This is a beautiful string!"

print(str1)

print("Splitting from 'i' \n>", str1.split('i'))
print("Min 2 split from 'i' \n>", str1.split('i', 2))
```

Output :

```
 ~~~~~~~~~~ SPLITTING STRING ~~~~~~~~~~
This is a beautiful string!
Splitting from 'i'
> ['Th', 's ', 's a beaut', 'ful str', 'ng!']
Min 2 split from 'i'
> ['Th', 's ', 's a beautiful string!']
```

26. SPLIT LINES

Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.

Syntax: splitlines(num=string.count('\n'))

Example :

```
para = '''This string as introduced is a
Paragraph\nthis will have multiple Lines in it \n The
way a paragraphmust be \n'''

print("\n Splitting paragraph from NEW LINE \n",
para.splitlines())
```

Output :

```
 Splitting paragraph from NEW LINE
 ['This string as introduced is a Paragraph', 'this will
have multiple Lines in it ', ' The way a paragraphmust be
']
```

27. SWAPCASE

The swapcase() method returns a copy of the string in which all the case-based characters have had their case swapped.

Syntax: str.swapcase()

Example :

```
print("\n", " SWAPCASE ".center(40, '~'))
string1 = "This is a GOOD EXAMPLE To Show!"

print("ACTUAL STRING >", string1, "\n",
string1.swapcase())
```

Output :

```
 ~~~~~~~~~~~~~~~ SWAPCASE ~~~~~~~~~~~~~~~
ACTUAL STRING > This is a GOOD EXAMPLE To Show!
 tHIS IS A good example tO sHOW!
```

28. ZERO FILL

The zfill() method pads string on the left with zeros to fill width.

Syntax: str.zfill(<width>)

Example :

```
print("\n", " ZERO FILL ".center(40, '~'))
example = "SAMPLE TEXT"

print("ZERO FILLING IN",example, "\n",example.zfill(20))
print("ZERO FILLING IN",example, "\n",example.zfill(12))
```

Output :

```
 ~~~~~~~~~~~~~~~ ZERO FILL ~~~~~~~~~~~~~~~
ZERO FILLING IN SAMPLE TEXT
 000000000SAMPLE TEXT
ZERO FILLING IN SAMPLE TEXT
 0SAMPLE TEXT
```

TOO MUCH OF COFFEE IS BAD FOR HUMANS

THANK GOD, WE ARE CODERS!