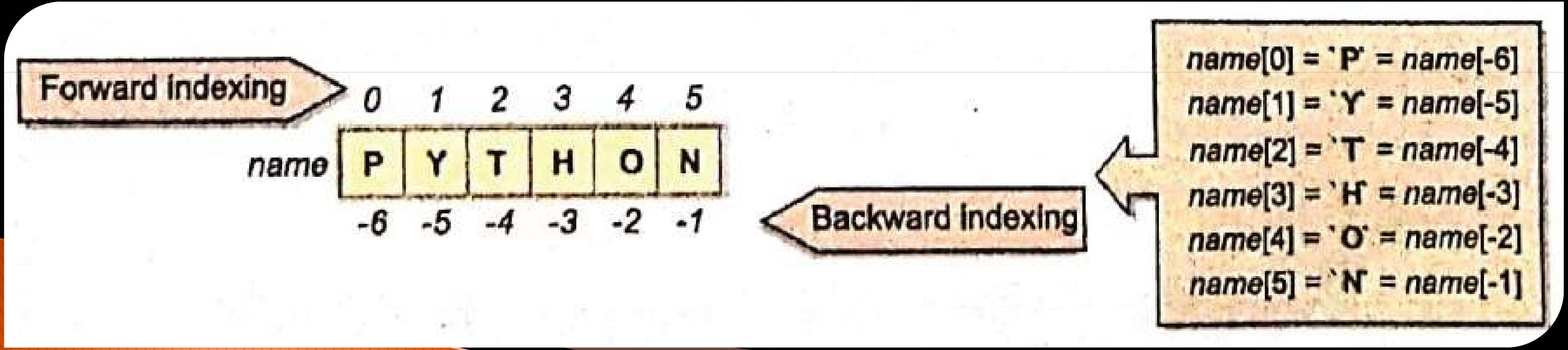
Strings in Python

Strings in Python

- Strings in Python are stored as individual characters in contiguous locations,
- with two-way index for each location.
- consider the following figure.



001 FORWARD INDEXING

```
name="Python"
print(name[0])
print(name[1])
print(name[2])
print(name[3])
print(name[4])
print(name[5])
```

002 BACKWARD INDEXING

```python
name="Python"
print(name[-6])
print(name[-5])
print(name[-4])
print(name[-3])
print(name[-2])
print(name[-1])
```

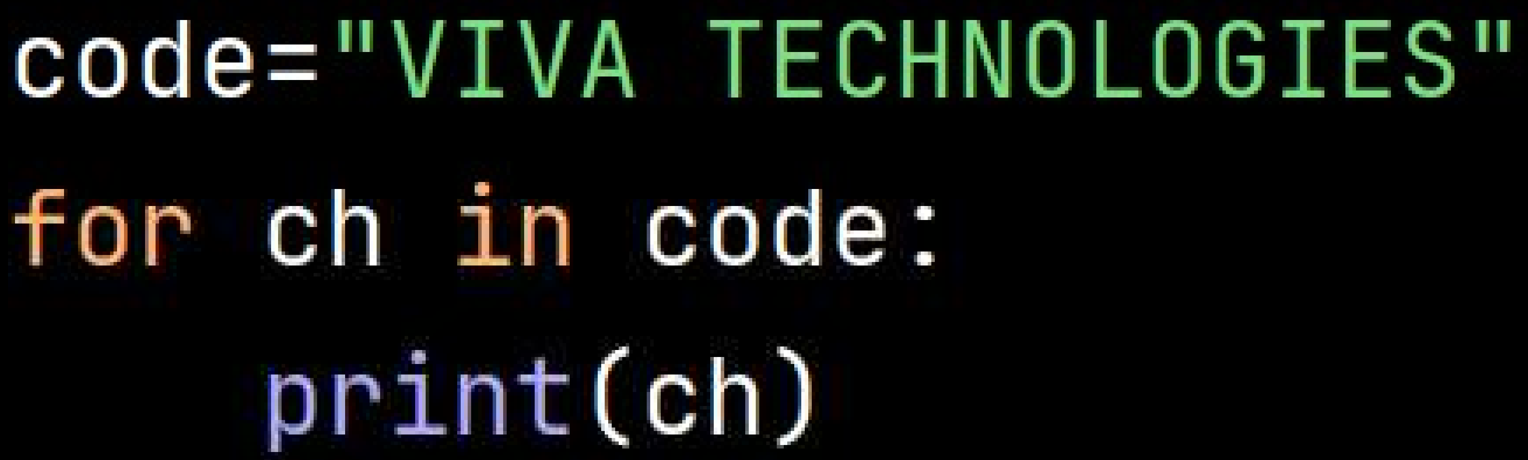
003 MUTATION IN STRING

```
name="hello"

name[0]='p'
```

NOTE : Strings are Immutable

Traversing a String

- Traversing refers to iterating through the element of a string character at a time.
- To traverse through string you can write a loop like



```
code="VIVA TECHNOLOGIES"
for ch in code:
    print(ch)
```

004 TRAVERSING STRING

```
code="VIVA TECHNOLOGIES"
for ch in code:
    print(ch)
```

```
code="VIVA TECHNOLOGIES"
for ch in code:
    print(ch, end=' ')
```

String Operators

- In this section you will be learning to work with various operators
- that can be used to manipulate string in multiple ways:

1. Concatenation Operator (+)
2. Replication Operators (*)
3. Membership Operators (in, not in)
4. Comparison Operators (<, <=, >, >=, ==, !=)

String Operators

1. Concatenation Operator +

- The + operator creates a new spring by joining the two operands strings, e.g.,

  “power" + "ful“

- Will result into:

  'power ful'

Caution !

- The + operator has to have both operand of the same type either of number types (for Addition) or of string types (for multiplication).
- It cannot work with one operands as string and one as a number.

005 CONCATENATION OPERATOR +

```
print("power"+"ful")
```

```
a="Viva "
b="Technologies"
print(a+b)
```

String Operators

2. replication operators *

- To use a * operator with strings you need 2 types of operands a string and a number,
- i.e., as Number * string or string * number
- where is string operands tell the string to be replicated and number operand tells the number of times, it is to be repeated

String Operators

- For example, 3 * “Ha!”
- Will return:, “Ha! Ha! Ha!”
- **Caution !**
- The * operator has to either have both operands of the number types (for multiplication) Or one string type and one number type (for replication).
- It cannot work with both operands of string types.

006 REPLICATION OPERATOR *

```
print(2 * "Wah! ")
print(3 * "Ha! ")
```

String Operators

3. Membership Operators

- There are two membership Operators for strings (in fact, for all sequence types).
- These are in and not in:
- In : returns true if a character or a substring exist in the given string ; false otherwise.
- Not in : not in returns true if a character or a substring does not exit in the given string ; falls otherwise.

String Operators

- Both membership operators (when used with string), required that both operands used with them are of string type, i.e.,

  <string> in <string>

  <string> not in <string>

007 MEMBERSHIP OPERATORS

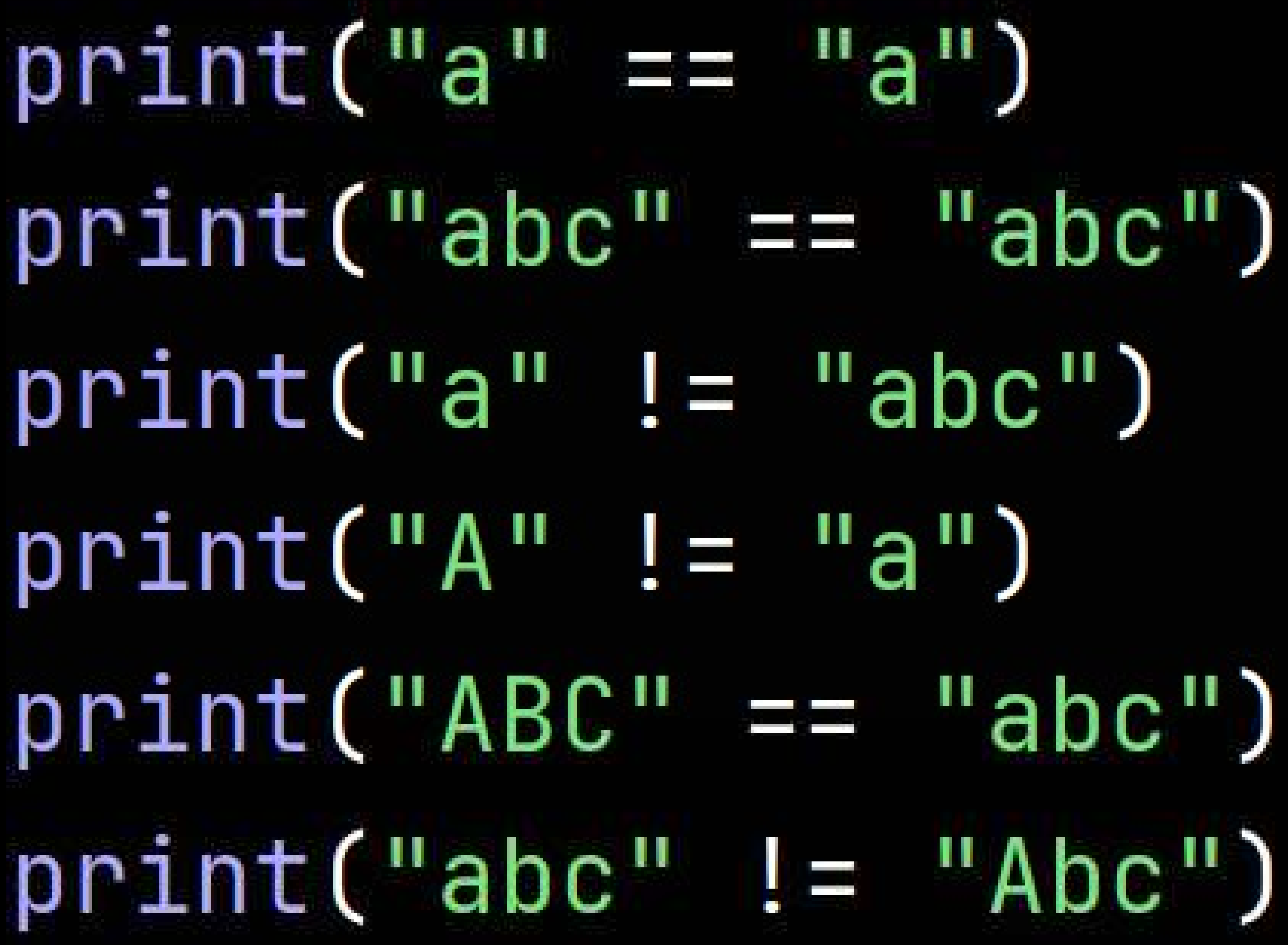
```
a="Viva Technologies"
print('V' in a)
print('T' not in a)
print('Techno' in a)
```

String Operators

4. Comparison Operators

- all relational operators (<, <=, >, >=, ==, !=) apply to string also.
- The comparisons using these operators are based on the standard character by character comparison rules for ASCII or Unicode (i.e., dictionary order).

008 COMPARISON OPERATORS



```
print("a" == "a")
print("abc" == "abc")
print("a" != "abc")
print("A" != "a")
print("ABC" == "abc")
print("abc" != "Abc")
```

String Operators

- **Determining ASCII/Unicode Value of a single character**
- Python offers are built in function ord ( ) that takes a single character and
- returns the corresponding ASCII value of Unicode value:

009 ASCII OR UNICODE VALUE – ORD OPERATOR

```
print(ord ('A'))
print(ord ('B'))
print(ord ('Z'))
print(ord ('a'))
print(ord ('b'))
print(ord ('z'))
```

String Operators

- The opposite of ord ( ) function is chr ( ),
- while ord ( ) returns the ASCII value of a character
- the chr( ) takes the ASCII value in integer form and returns the character corresponding to that ASCII value.

010 CHR OPERATOR

```
print(chr (65))
print(chr (66))
print(chr (97))
print(chr (98))
```

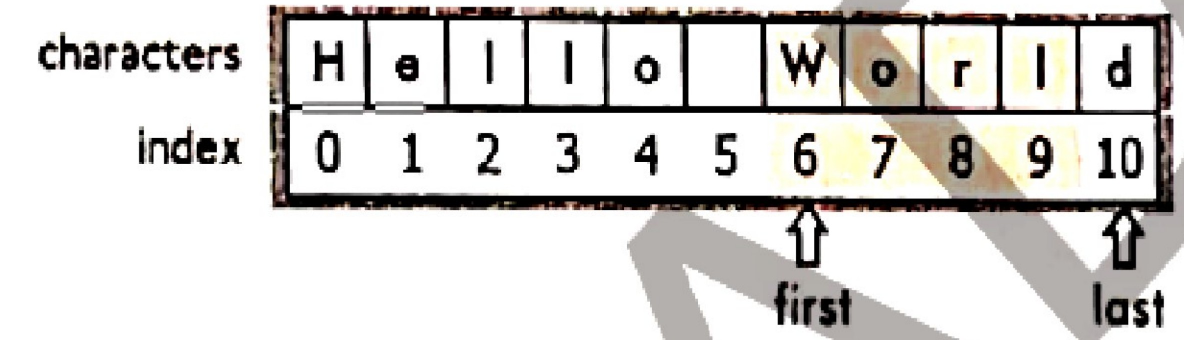
String Slices

- The term ‘String Slice’ refers to a part of the string where strings are sliced using a range of indices.

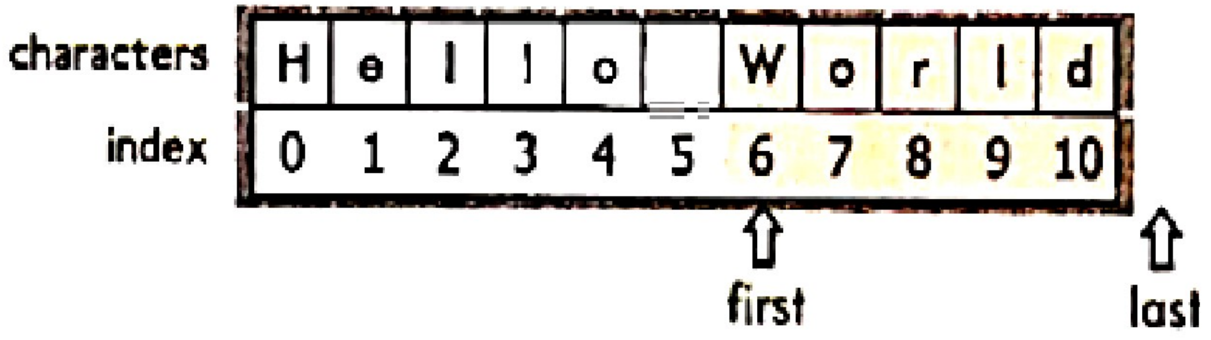
String Slices

helloString ="Hello World" :

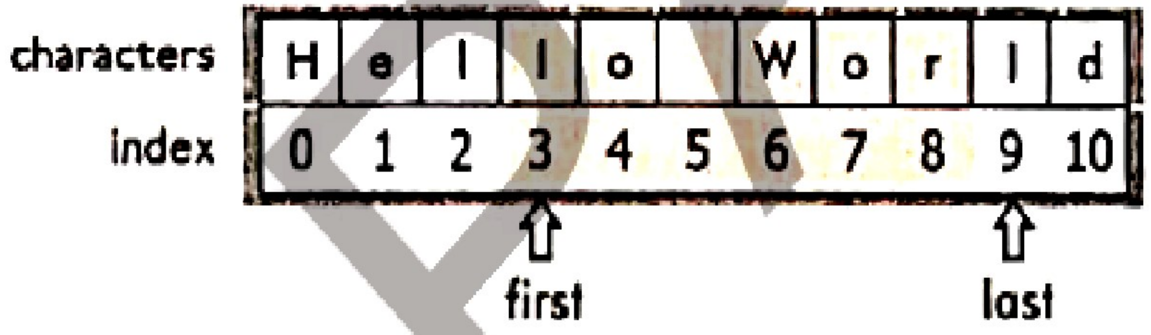
helloString[6:10]

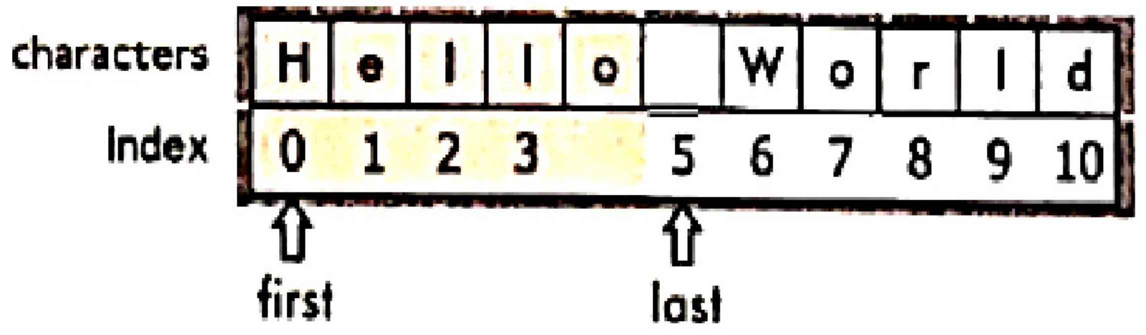


helloString[6:]



helloString[3:-2]





helloString[:5]

NOTE : For any index n, s [:n] + s[n:] will give you original string s.

011 STRING SLICING

```
a="Viva Technologies"
print(a[5:11])
print(a[5:])
print(a[5:-2])
print(a[:5])
```

V	I	V	A		T	E	C	H	N	O	L	O	G	I	E	S
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
-17	-16	-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

String Slices

❖ **Interesting Inference**

❖ Using the same string slicing techniques, you will find that

For any index n, s[:n] + s[:n] will give you original strings.

NOTE : string [:-1] is an easy way to receive a string

012 SLICING PART - 2

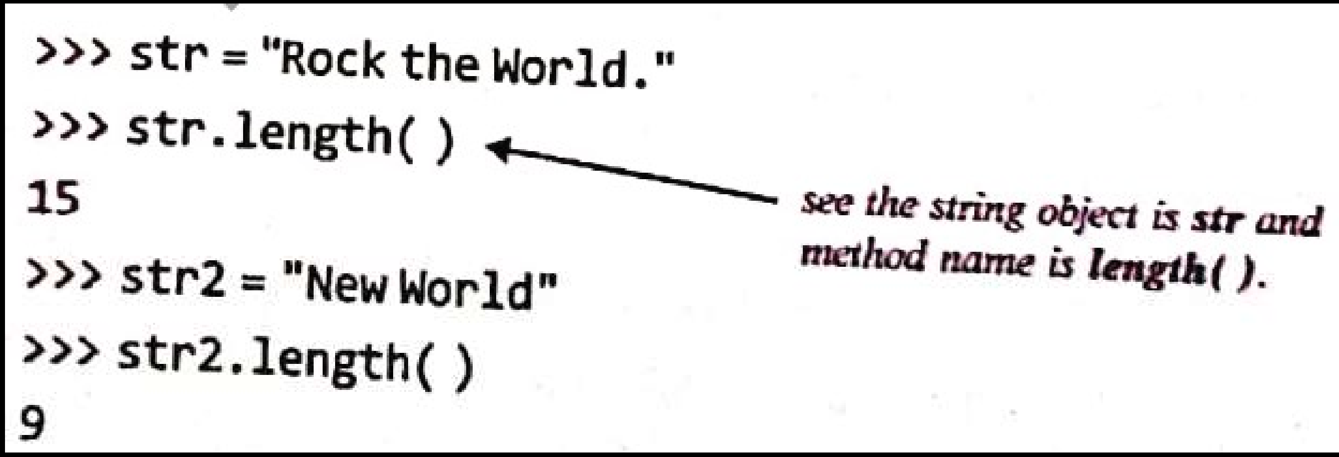
```
a="amazing"

print(a[3:], a[:3])

print(a[:3] + a[3:])

print(a[:-7], a[-7:])

print(a[:-7] + a[-7:])
```

String Functions

- For instance, if you have a string namely

  str="Rock the World"

- and you want to find its length, you will write the code somewhat like shown below:



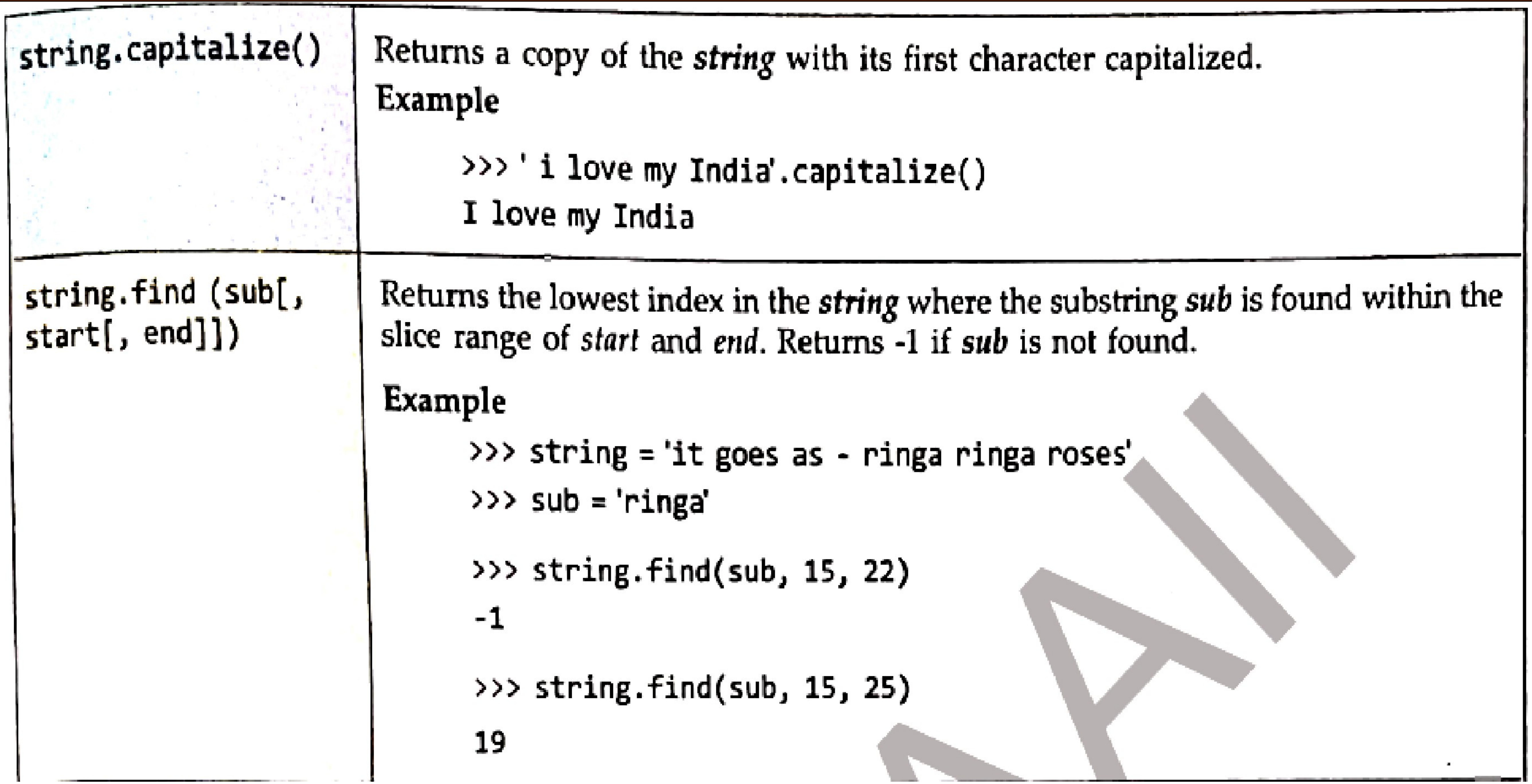
```
>>> str = "Rock the World."
>>> str.length( )
15
>>> str2 = "New World"
>>> str2.length( )
9
```

NOTE : Do you know that following website and web applications have used python extensively : Instagram, Drop box, Google, Netflix, spotify, Quora, reedit, Face book, and many others?

013 LENGTH STRING FUNCTIONS

```
import string
a="Viva Technologies"
print(len(a))
b="An ISO Certified Institute"
print(len(b))
```

Python's built-in string manipulation methods



`string.capitalize()`	Returns a copy of the *string* with its first character capitalized.
	Example
	`>>> ' i love my India'.capitalize()`
	`I love my India`

`string.find (sub[, start[, end]])`	Returns the lowest index in the *string* where the substring *sub* is found within the slice range of *start* and *end*. Returns -1 if *sub* is not found.
	Example
	`>>> string = 'it goes as - ringa ringa roses'`
	`>>> sub = 'ringa'`
	`>>> string.find(sub, 15, 22)`
	`-1`
	`>>> string.find(sub, 15, 25)`
	`19`

014 STRING CAPITALIZE

```
a="viva technologies"
print(a.capitalize())
```

015 STRING FIND

```
a="Viva Technologies, An ISO Certified Institute"
sub="Techno"
print(a.find(sub,  __start: 7,  __end: 18))
print(a.find(sub,  __start: 3,  __end: 18))
```

Python's built-in string manipulation methods

`string.isalnum()`	Returns **True** if the characters in the ***string*** are alphanumeric (alphabets or numbers) and there is at least one character, **False** otherwise.
`string.isalpha()`	Returns **True** if all characters in the ***string*** are alphabetic and there is at least one character, **False** otherwise.
`string.isdigit()`	Returns **True** if all the characters in the ***string*** are digits. There must be at least one digit, otherwise it returns **False**.

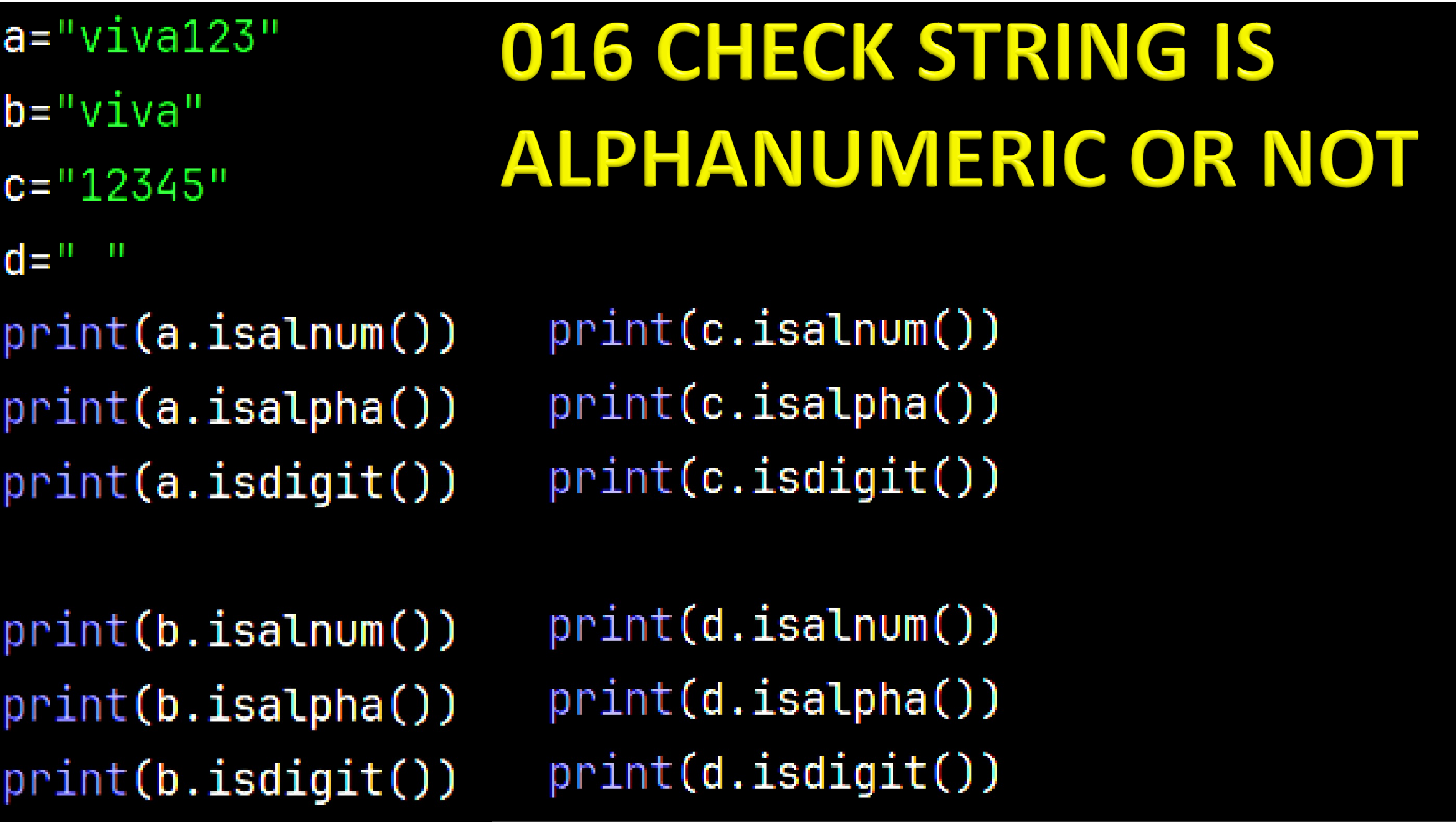
Examples

```
>>> string = "abc123"
>>> string2 = 'hello'
>>> string3 = '12345'
>>> string4 = ' '
```

```
>>> string.isalnum()
True
>>> string2.isalnum()
True
>>> string3.isalnum()
True
>>> string4.isalnum()
False
```

```
>>> string.isalpha()
False
>>> string2.isalpha()
True
>>> string3.isalpha()
False
>>> string4.isalpha()
False
```

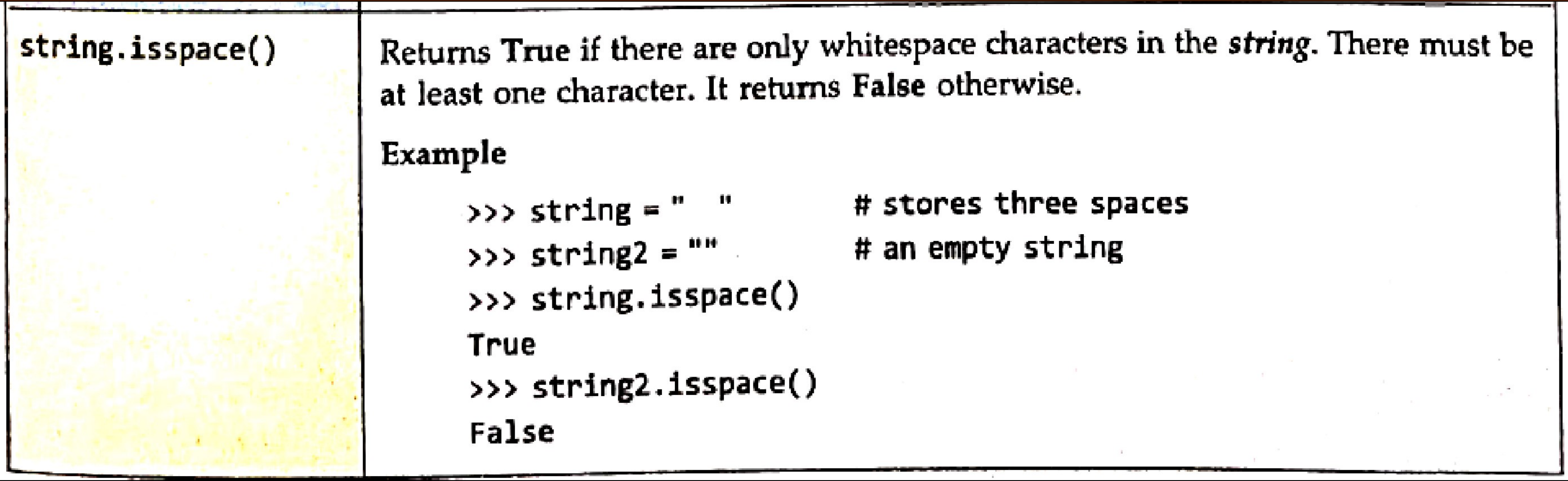
```
>>> string.isdigit()
False
>>> string2.isdigit()
False
>>> string3.isdigit()
False
>>> string4.isdigit()
True
```



016 CHECK STRING IS ALPHANUMERIC OR NOT

```
a="viva123"
b="viva"
c="12345"
d=" "
print(a.isalnum())    print(c.isalnum())
print(a.isalpha())    print(c.isalpha())
print(a.isdigit())    print(c.isdigit())


print(b.isalnum())    print(d.isalnum())
print(b.isalpha())    print(d.isalpha())
print(b.isdigit())    print(d.isdigit())
```

Python's built-in string manipulation methods



`string.isspace()`	Returns **True** if there are only whitespace characters in the *string*. There must be at least one character. It returns **False** otherwise.

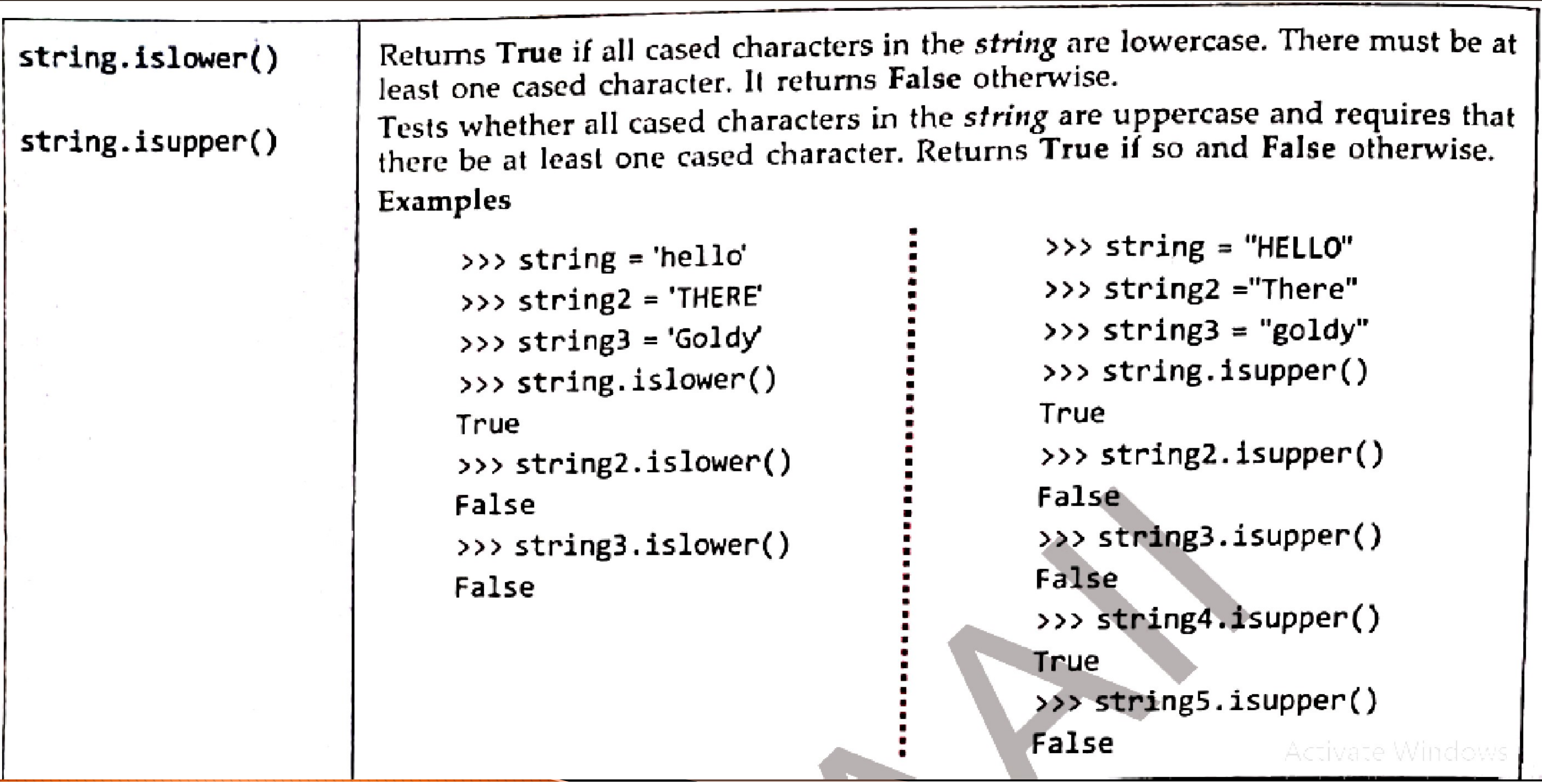
Example

```
>>> string = "   "          # stores three spaces
>>> string2 = ""            # an empty string
>>> string.isspace()
True
>>> string2.isspace()
False
```

017 STRING SPACE

```
a=""

b=" "

print(a.isspace())

print(b.isspace())
```

Python's built-in string manipulation methods



`string.islower()`	Returns **True** if all cased characters in the *string* are lowercase. There must be at least one cased character. It returns **False** otherwise.
`string.isupper()`	Tests whether all cased characters in the *string* are uppercase and requires that there be at least one cased character. Returns **True** if so and **False** otherwise.

Examples

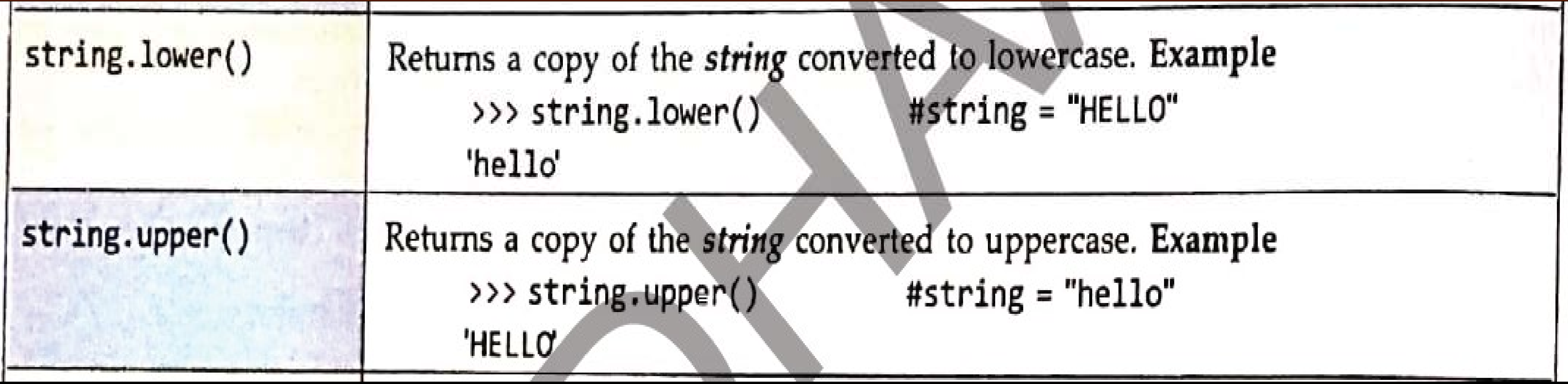
```
>>> string = 'hello'
>>> string2 = 'THERE'
>>> string3 = 'Goldy'
>>> string.islower()
True
>>> string2.islower()
False
>>> string3.islower()
False
```

```
>>> string = "HELLO"
>>> string2 ="There"
>>> string3 = "goldy"
>>> string.isupper()
True
>>> string2.isupper()
False
>>> string3.isupper()
False
>>> string4.isupper()
True
>>> string5.isupper()
False
```

018 STRING IS LOWER OR NOT

```
a="viva technologies"
b="VIVA TECHNOLOGIES"
c="Viva Technologies"
print(a.islower())
print(a.isupper())

print(b.islower())
print(b.isupper())

print(c.islower())
print(c.isupper())
```

Python's built-in string manipulation methods



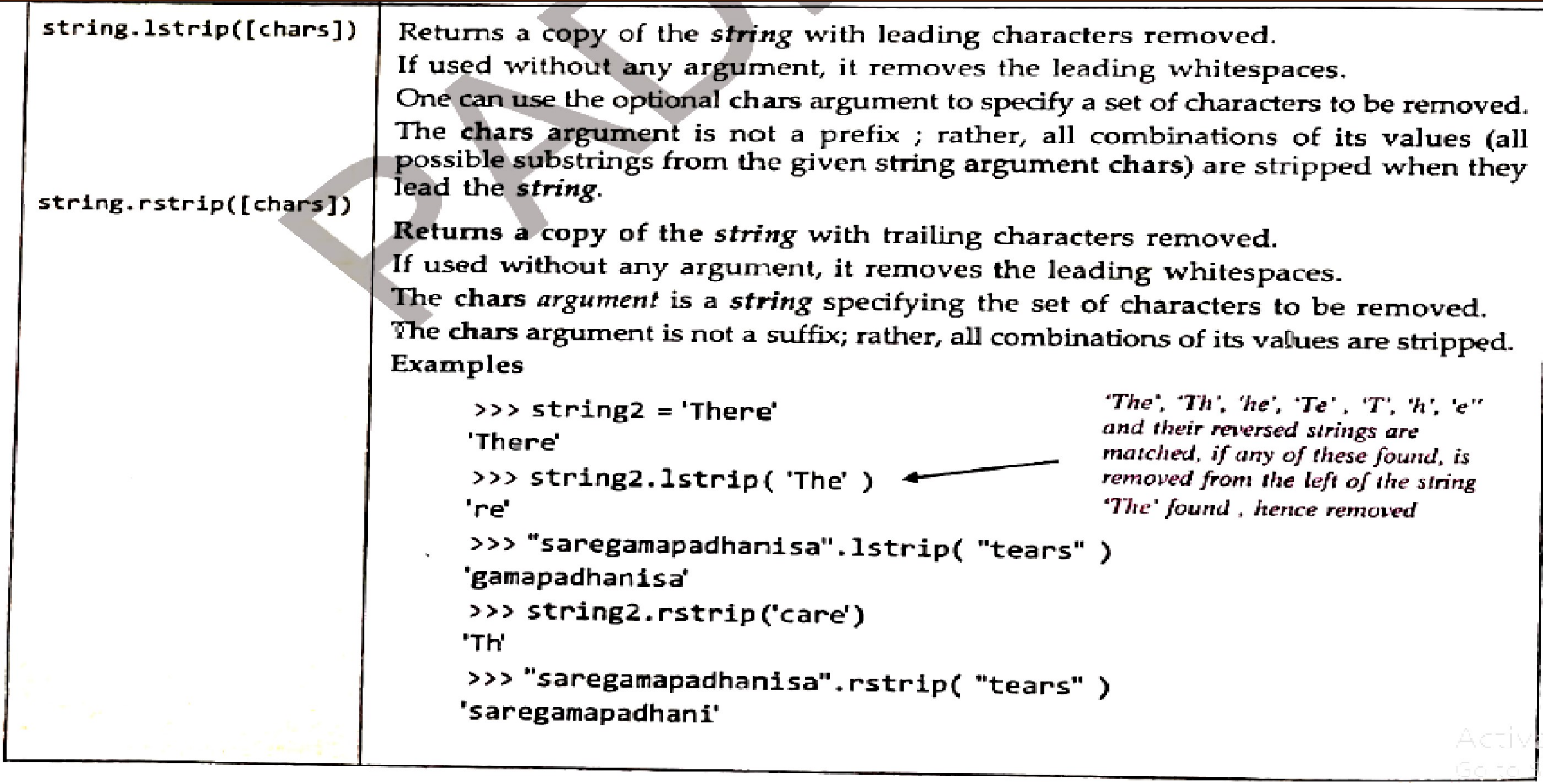
`string.lower()`	Returns a copy of the *string* converted to lowercase. Example
	`>>> string.lower()          #string = "HELLO"`
	`'hello'`

`string.upper()`	Returns a copy of the *string* converted to uppercase. Example
	`>>> string.upper()          #string = "hello"`
	`'HELLO'`

019 STRING CONVERTED INTO LOWER AND UPPER

```
a="viva technologies"
print(a.upper())
print(a.lower())
```

Python's built-in string manipulation methods



`string.lstrip([chars])`	Returns a copy of the ***string*** with leading characters removed. If used without any argument, it removes the leading whitespaces. One can use the optional chars argument to specify a set of characters to be removed. The **chars argument** is not a prefix ; rather, all combinations of its values (all possible substrings from the given string argument chars) are stripped when they lead the ***string***.
`string.rstrip([chars])`	**Returns a copy** of the ***string*** with trailing characters removed. If used without any argument, it removes the leading whitespaces. The **chars** ***argument*** is a ***string*** specifying the set of characters to be removed. The **chars** argument is not a suffix; rather, all combinations of its values are stripped.

Examples

```
>>> string2 = 'There'
'There'
>>> string2.lstrip( 'The' )
're'
>>> "saregamapadhanisa".lstrip( "tears" )
'gamapadhanisa'
>>> string2.rstrip('care')
'Th'
>>> "saregamapadhanisa".rstrip( "tears" )
'saregamapadhani'
```

'The', 'Th', 'he', 'Te' , 'T', 'h', 'e" and their reversed strings are matched, if any of these found, is removed from the left of the string 'The' found , hence removed

020 LEFT STRIP AND RIGHT STRIP

```
a="Viva Technologies"
print(a.lstrip("Viv"))
print(a.rstrip("gies"))
```

String Functions

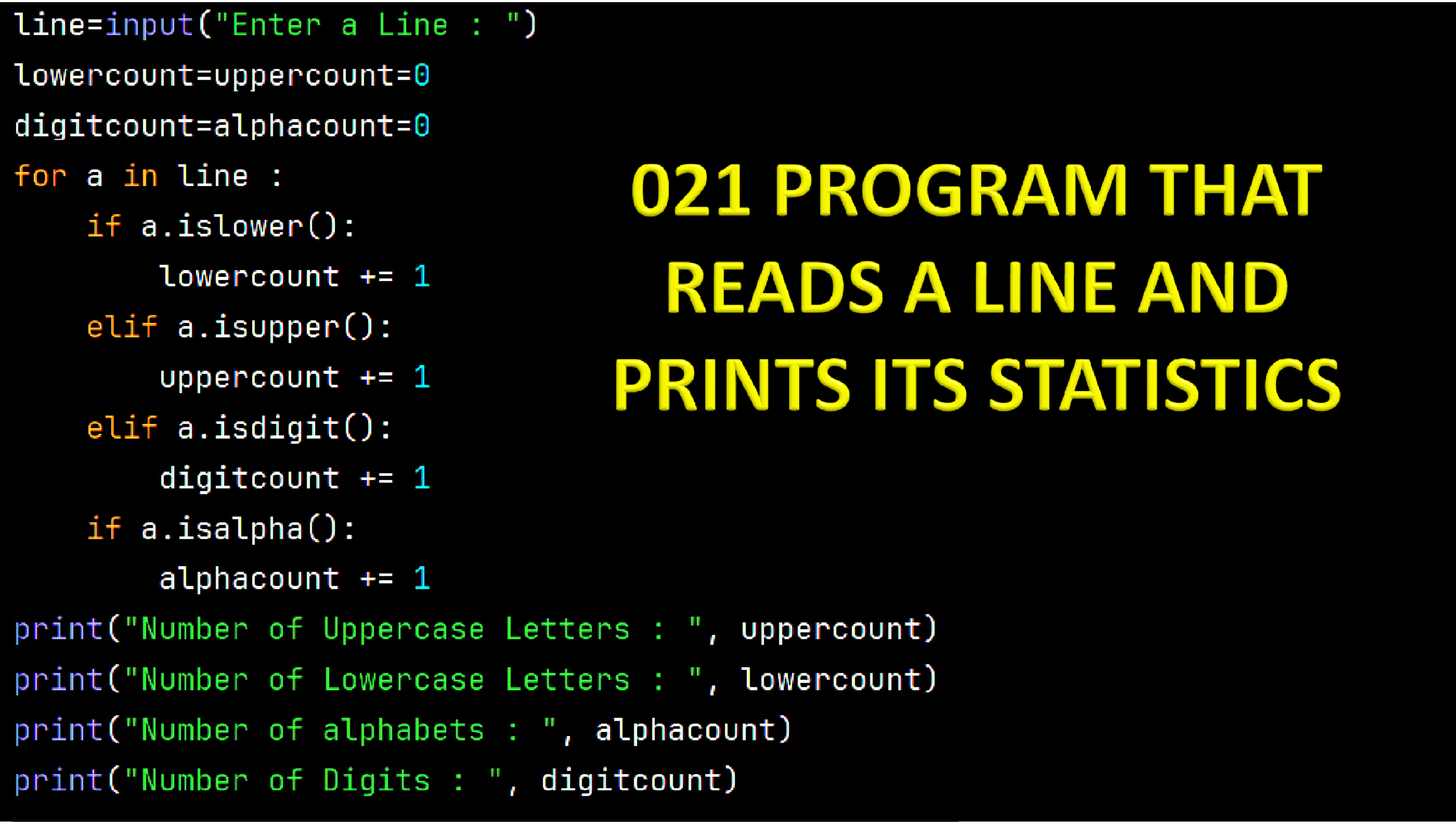
- Program that reads a line and prints its statistics like :

  Number of uppercase letters :

  Number of lowercase letters :

  Number of alphabets :

  Number of digits :



021 PROGRAM THAT READS A LINE AND PRINTS ITS STATISTICS

```python
line=input("Enter a Line : ")
lowercount=uppercount=0
digitcount=alphacount=0
for a in line :
    if a.islower():
        lowercount += 1
    elif a.isupper():
        uppercount += 1
    elif a.isdigit():
        digitcount += 1
    if a.isalpha():
        alphacount += 1
print("Number of Uppercase Letters : ", uppercount)
print("Number of Lowercase Letters : ", lowercount)
print("Number of alphabets : ", alphacount)
print("Number of Digits : ", digitcount)
```