CHAPTER 10

STRINGS

Learning Objectives

After completing this chapter, the reader will be familiar with the usage related to:

- Know how to access individual characters or values in a string
- Acquire a fundamental understanding of how to manipulate string appearance, order, and contents
- Retrieve, search for a substring from a string

- Understand how to use string methods to manipulate string using string operators, methods, and functions
- Acquire a fundamental understanding of methods of Strings, like count, split, replace, find, and index, which searches strings for substrings
- Learn to combine strings via concatenation to form new strings
- Know how to format a string and visit each of its characters with a for loop
- Ability to make use of regular expressions to match patterns in strings

10.1 Introduction to Strings

A string is a *data type* (numbers - an integer and floating-point text) that comprises a set of characters that can also contain spaces and numbers. *Strings* are *ordered sets of characters* used to denote non-numerical data, such as works of literature, genetic sequences, etc. In strings, the order is vital, and this *ordering feature differentiates* strings from sets of characters. Strings are *convenient for modelling sequential behaviour,* and computers are sequential machines. String management is an essential operation in many algorithms; it aids in data validation, text parsing, file conversions, etc.; Python Strings are a series of Unicode characters that cannot be changed (*immutable property*). Strings come in handy for passing data back and forth between the application and its users. To store data for the computer, they're less valuable.

10.2. Creating a String

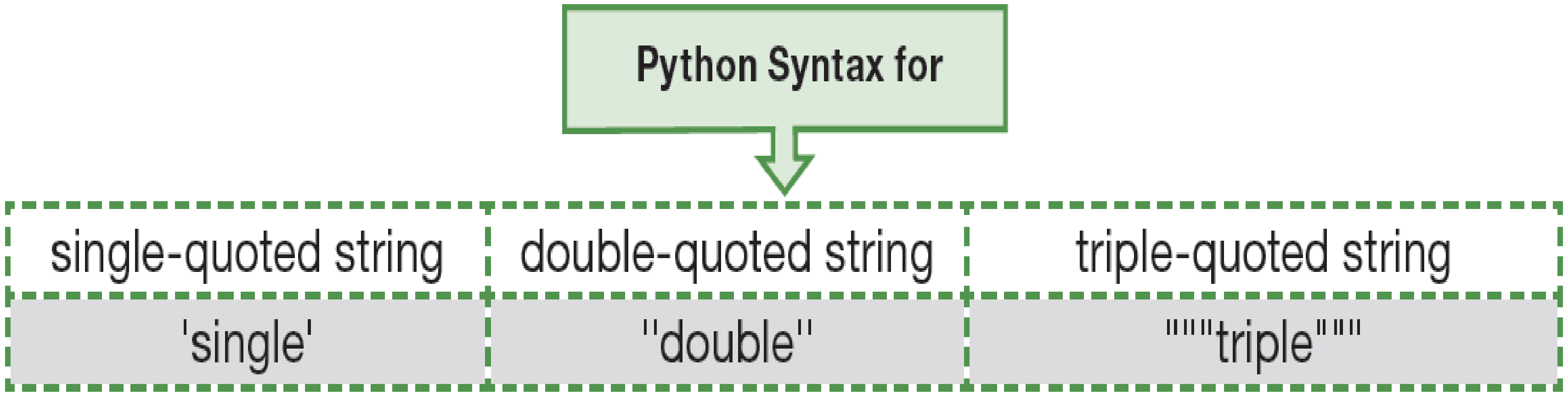
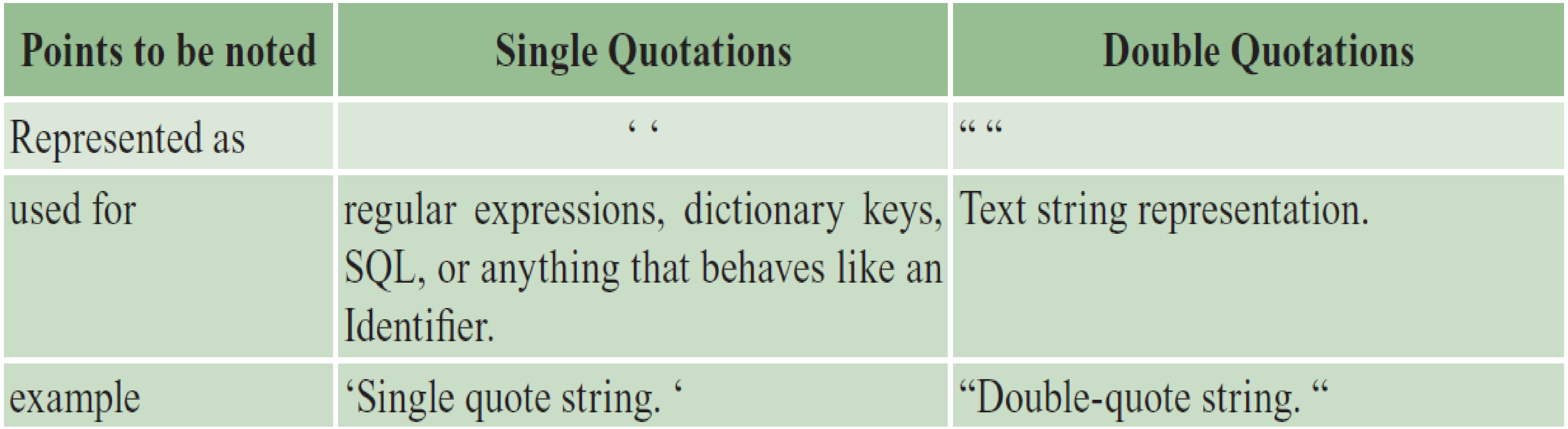


Table 10.1: Key Differences between the Single and Double Quotes in Python



Points to be noted	Single Quotations	Double Quotations
Represented as	‘ ‘	“ “
used for	regular expressions, dictionary keys, SQL, or anything that behaves like an Identifier.	Text string representation.
example	‘Single quote string. ‘	“Double-quote string. “

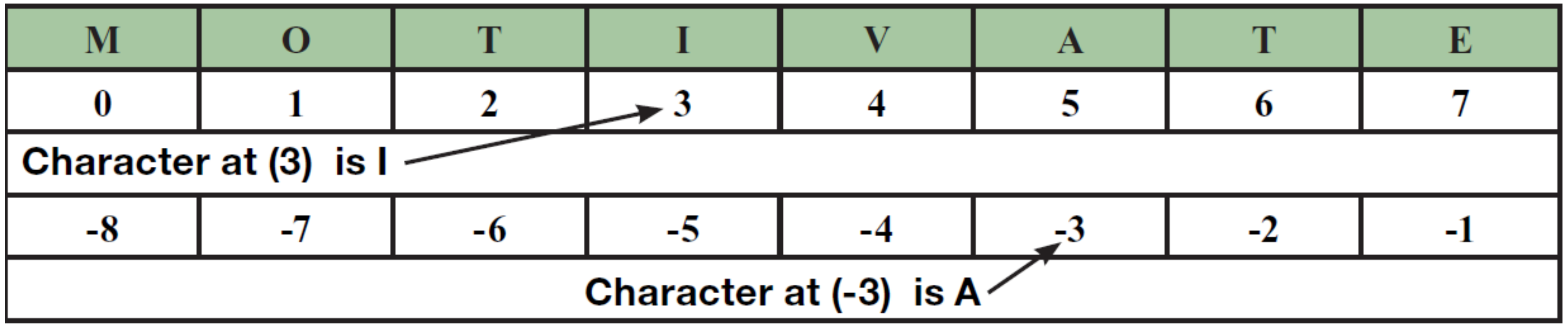
10.3 Operations in Python

The operations on the string are broadly classified into the following types.

- Indexing and Slicing
- Concatenation
- Replication
- Membership Operations

1. Indexing and Slicing:

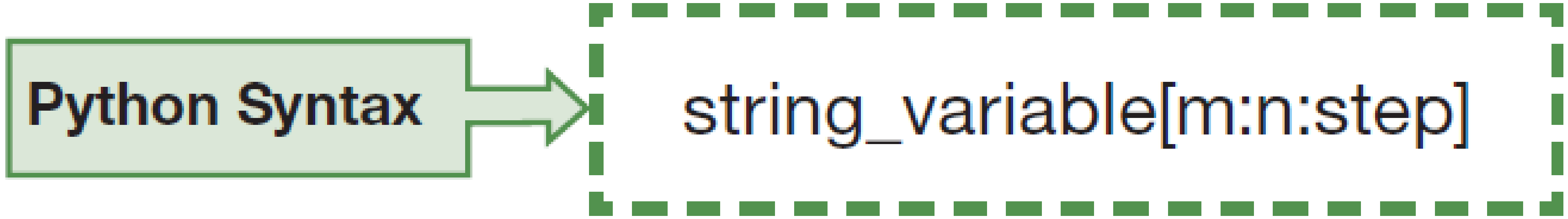
The positive indexing of the string starts from 0 (from the first character of the string), whereas the negative indexing starts from -1 (from the last character of the string).



M	O	T	I	V	A	T	E
0	1	2	3	4	5	6	7
Character at (3) is I							
-8	-7	-6	-5	-4	-3	-2	-1
			Character at (-3) is A				

Figure 10.1: Indexing for the string "MOTIVATE" stored in *var1*.

Slicing: In strings, slicing refers to extracting the substring from a string according to the indexing and step size. The syntax for the slicing operation is as follows.



```
string_variable[m:n:step]
```

Strings slicing has three values

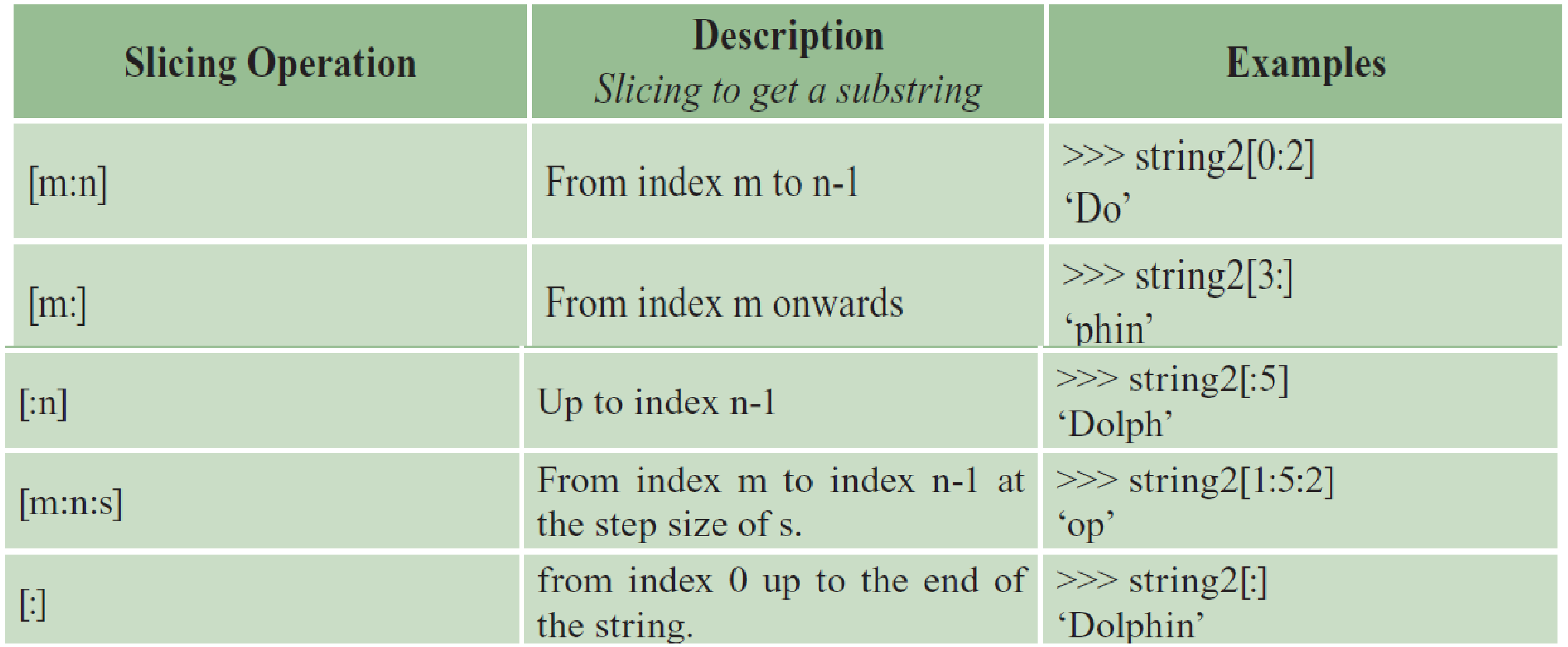
- *start [optional]* - An integer number shows the start of slicing.

  default setting is 0.

- *end* [optional]- An integer denoting the endpoint of the slicing.
- *step[optional]*-. An integer number denotes the slicing step.

  The default setting is 1.

Table 10.2: Slicing Operations



Slicing Operation	Description *Slicing to get a substring*	Examples
[m:n]	From index m to n-1	>>> string2[0:2] 'Do'
[m:]	From index m onwards	>>> string2[3:] 'phin'
[:n]	Up to index n-1	>>> string2[:5] 'Dolph'
[m:n:s]	From index m to index n-1 at the step size of s.	>>> string2[1:5:2] 'op'
[:]	from index 0 up to the end of the string.	>>> string2[:] 'Dolphin'

Concatenation Operation:

Concatenation refers to combining any Two or more strings into a single string.

ways of string concatenation in Python.

- Using “+” operator
- Using % Operator
- Using F-String
- Using *join()* method

Using % Operator: Strings can be concatenated by placing “%s” inside the quotes and providing the strings inside the parenthesis in order after placing the “%” between them. The syntax for performing concatenation using “%” is as follows.

Python Syntax

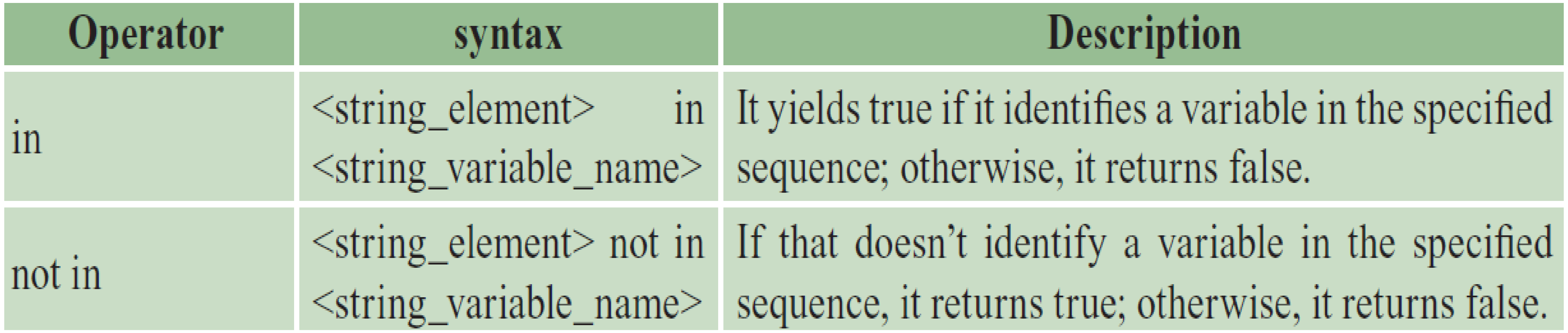
```
String_variable = “%s %s …. “%(str1, str2, str3, …..)
```

Replication Operation

Replication operation is performed by “*” operator. There are two ways of operating: n * *string_ variable* and *string_variable * n*, where n denotes the number of times replicated.

Membership Operations:

Table 10.3: Syntax of Membership Operators in Python



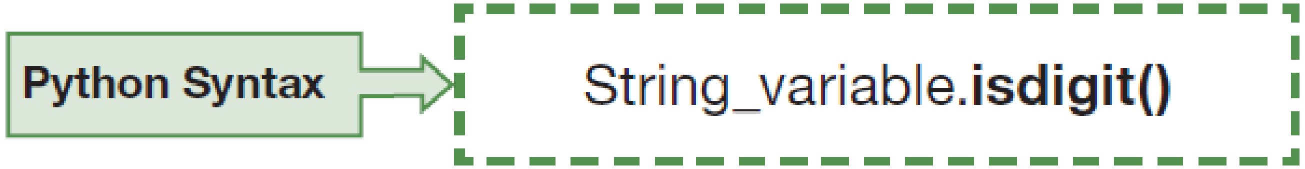
Operator	syntax	Description
in	<string_element> in <string_variable_name>	It yields true if it identifies a variable in the specified sequence; otherwise, it returns false.
not in	<string_element> not in <string_variable_name>	If that doesn’t identify a variable in the specified sequence, it returns true; otherwise, it returns false.

Input Parameters are:

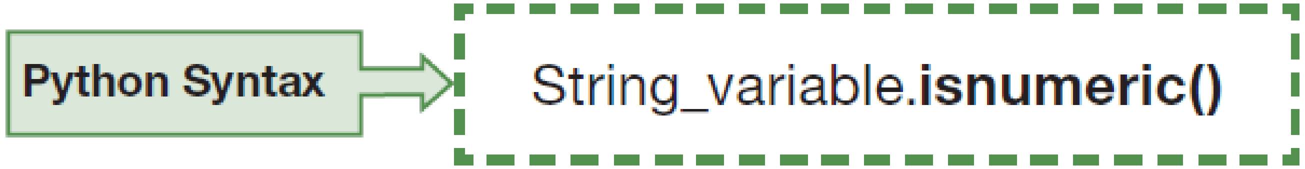
- **string_element:** Check to see if the string searched for is linked to another string.
- **string_variable_name:** This is the original string variable, which must be verified to lie in the string as a substring.
- **in & not in:** Is the logical membership operators.

10.4. Built-in Methods for Strings

isdigit() and isnumeric() - For each string passed to Python's isdigit() method or isnumeric() method, it determines whether a string contains only numeric characters (0-9). If all the characters are numbers, it returns True; otherwise, it returns False as a Boolean value. Digits are characters for which the Unicode property Numeric Type=Digit or Numeric Type=Decimal exists. The syntax for isdigit() and isnumeric() methods are as follows.



```
String_variable.isdigit()
```



```
String_variable.isnumeric()
```

isalnum()-For each string passed to Python’s isalnum() method, it checks if a string is all alphanumeric (i.e., letters and numbers only, no special characters or spaces). This method returns True if the input characters are alphabetic or decimal digits,and False otherwise.

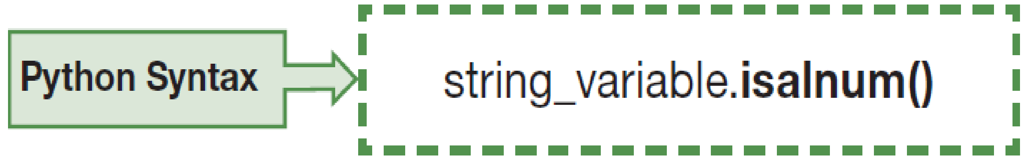
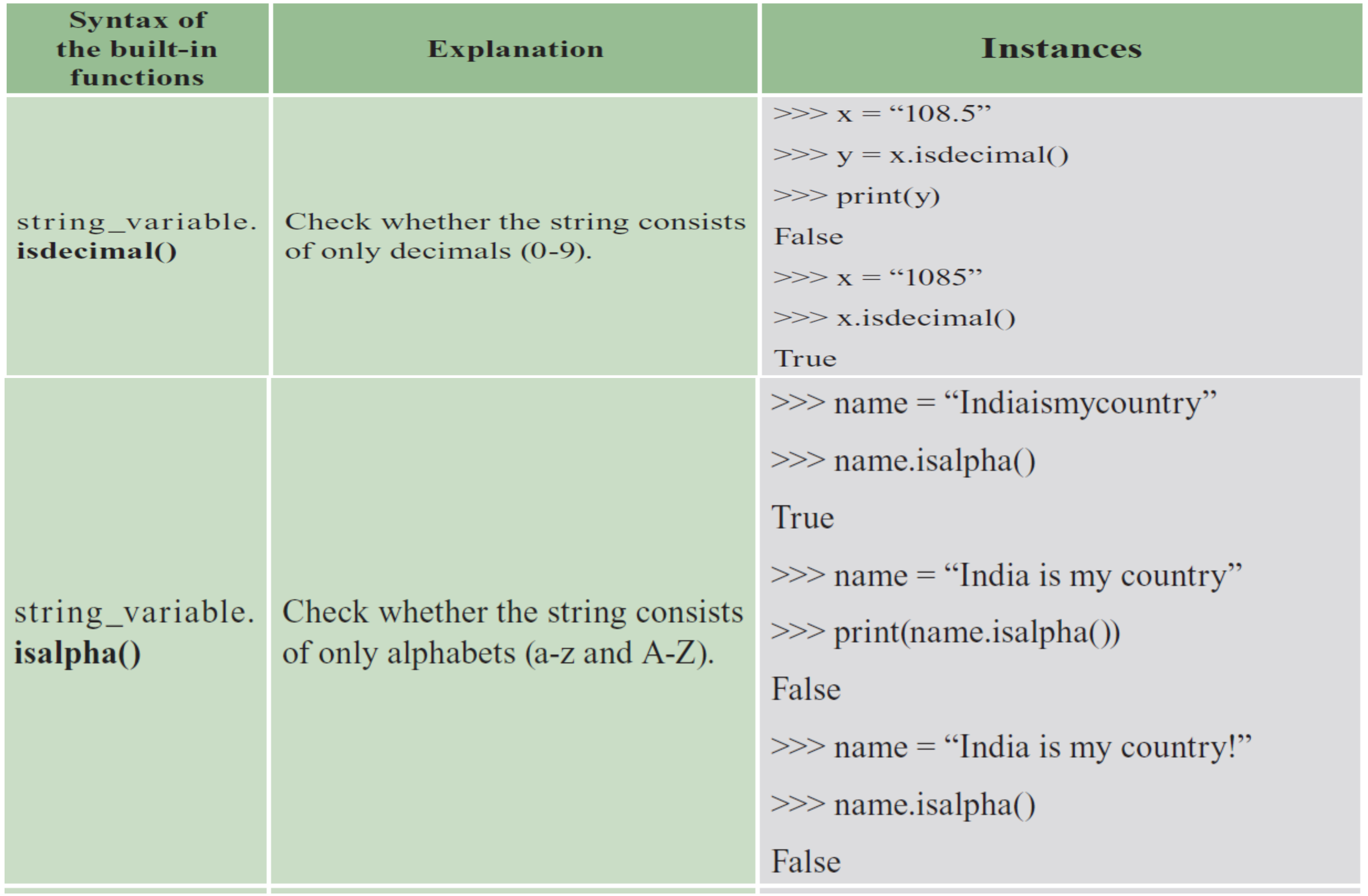


Table 10.4: Important Built-in methods



Syntax of the built-in functions	Explanation	Instances

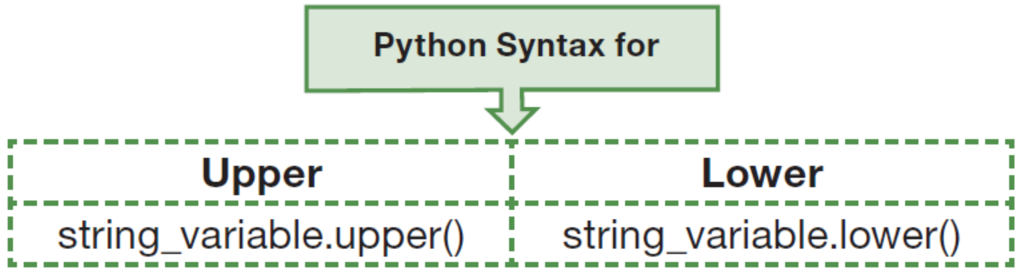
string_variable.**isdecimal()**	Check whether the string consists of only decimals (0-9).	>>> x = “108.5”
		>>> y = x.isdecimal()
		>>> print(y)
		False
		>>> x = “1085”
		>>> x.isdecimal()
		True

string_variable.**isalpha()**	Check whether the string consists of only alphabets (a-z and A-Z).	>>> name = “Indiaismycountry”
		>>> name.isalpha()
		True
		>>> name = “India is my country”
		>>> print(name.isalpha())
		False
		>>> name = “India is my country!”
		>>> name.isalpha()
		False

string_variable.
islower()
	Check whether the string consists of only lowercase letters.	>>> string1 = 'own less. live more.'
		>>> print(string1.islower())
		True
		>>> string1 = 'own less. Live more.'
		>>> string1.islower()
		False

string_variable.
isupper()
	Check whether the string consists of only uppercase letters.	>>> string1 = 'own less. live more.'
		>>> print(string1.isupper())
		False
		>>> string1 = 'OWN LESS. LIVE MORE'
		>>> string1.isupper()
		True

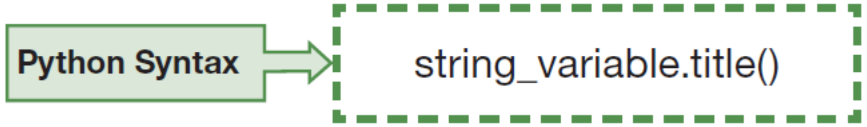
- **upper() -** It generates a new string with all the original characters capitalized.
- **lower() -** It generates a new string with all the original string’s characters converted to a lowercase.



Python Syntax for

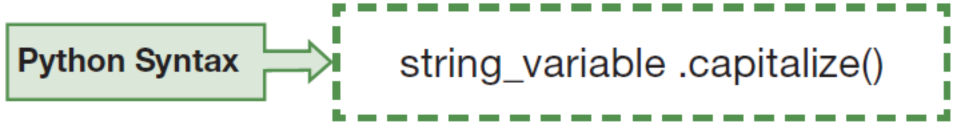
Upper	Lower
string_variable.upper()	string_variable.lower()

- **title() –** Each string passed to title() returns a new string with the first character of each word capitalized and the leftover characters lowercase.



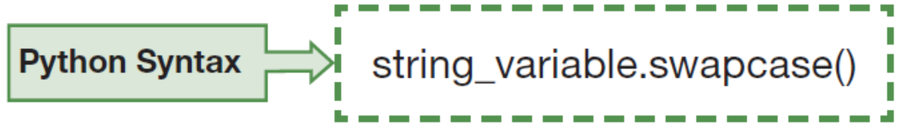
```
string_variable.title()
```

- **capitalise()-** Each string passed to capitalise() will return a new string with the first character capitalized and the leftover characters lowercase.



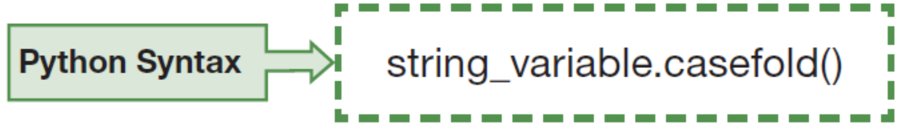
```
string_variable .capitalize()
```

- **swapcase()-**Each string passed to swapcase() returns a new string with uppercase characters converted to lowercase and vice versa.



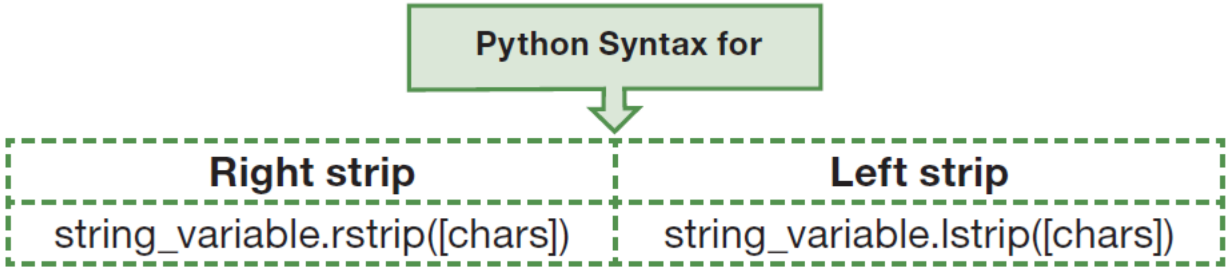
```
string_variable.swapcase()
```

- **casefold() -** For each string passed to casefold() it returns it. This function is used to compare case-insensitive strings. Case folding is an extreme lower-casing that accommodates script letter variations.



```
string_variable.casefold()
```

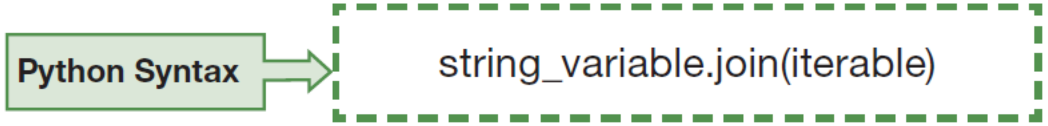
- **rstrip() –** The rstrip() method is used to eliminate the specified character string end (default is a space) or right of the string.



Python Syntax for

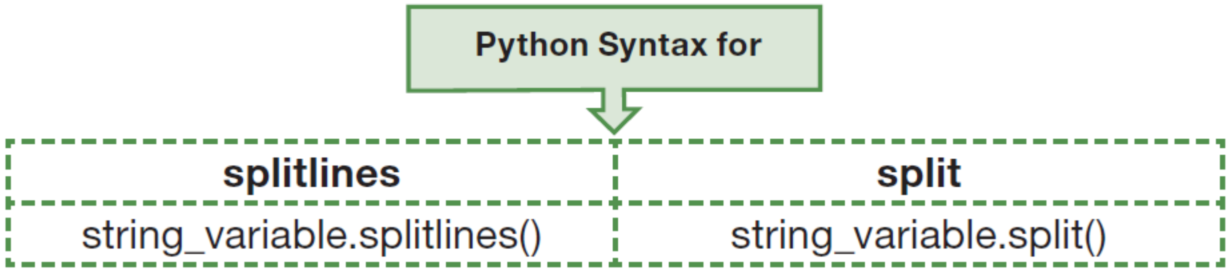
Right strip	Left strip
string_variable.rstrip([chars])	string_variable.lstrip([chars])

- **join(seq) or join() -**This function returns a new string that is the concatenation of the strings in iterable with a string object as a delimiter. The separator is used to demarcate individual components of the output string.



```
string_variable.join(iterable)
```

- **splitlines(num)-** used to split the content of the invoked string at each occurrence of a newline character, i.e., Splitlines() returns a list of split values. Python’s splitlines() method returns a list of strings for each line in a string. The newline character (‘n’) is the default line delimiter.

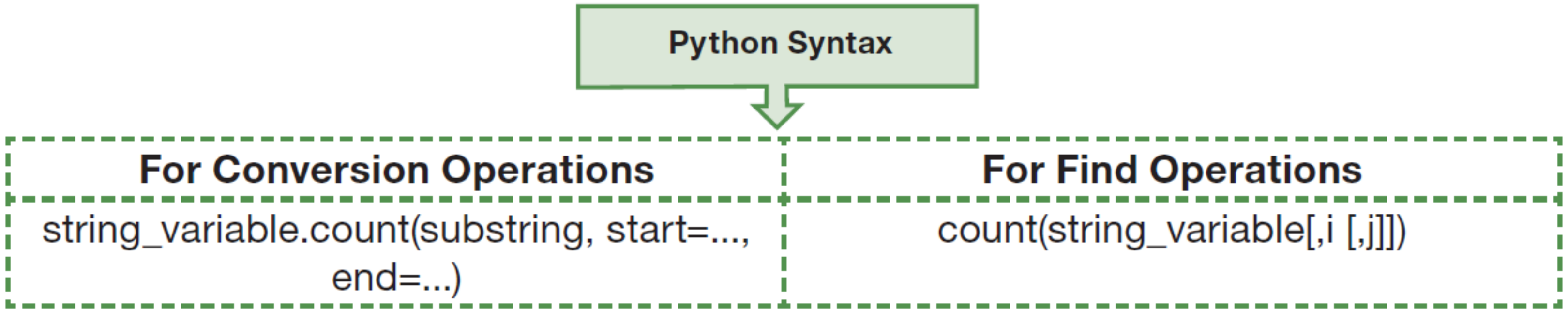


Python Syntax for

splitlines	split
string_variable.splitlines()	string_variable.split()

10.5. Built-in functions for Strings

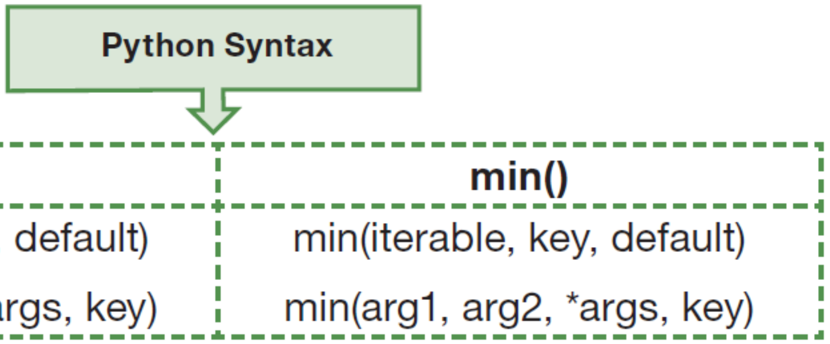
- **len() -** For each string passed to len(), it will return the number of items (length) of the string object.
- **count() –**used to calculate the number of occurrences of a substring in a string. This feature is handy for conversion and finding operations.



Python Syntax

For Conversion Operations	For Find Operations
string_variable.count(substring, start=..., end=...)	count(string_variable[,i [,j]])

- **max()-**Returns the largest value from the stated iterable or multiple arguments.



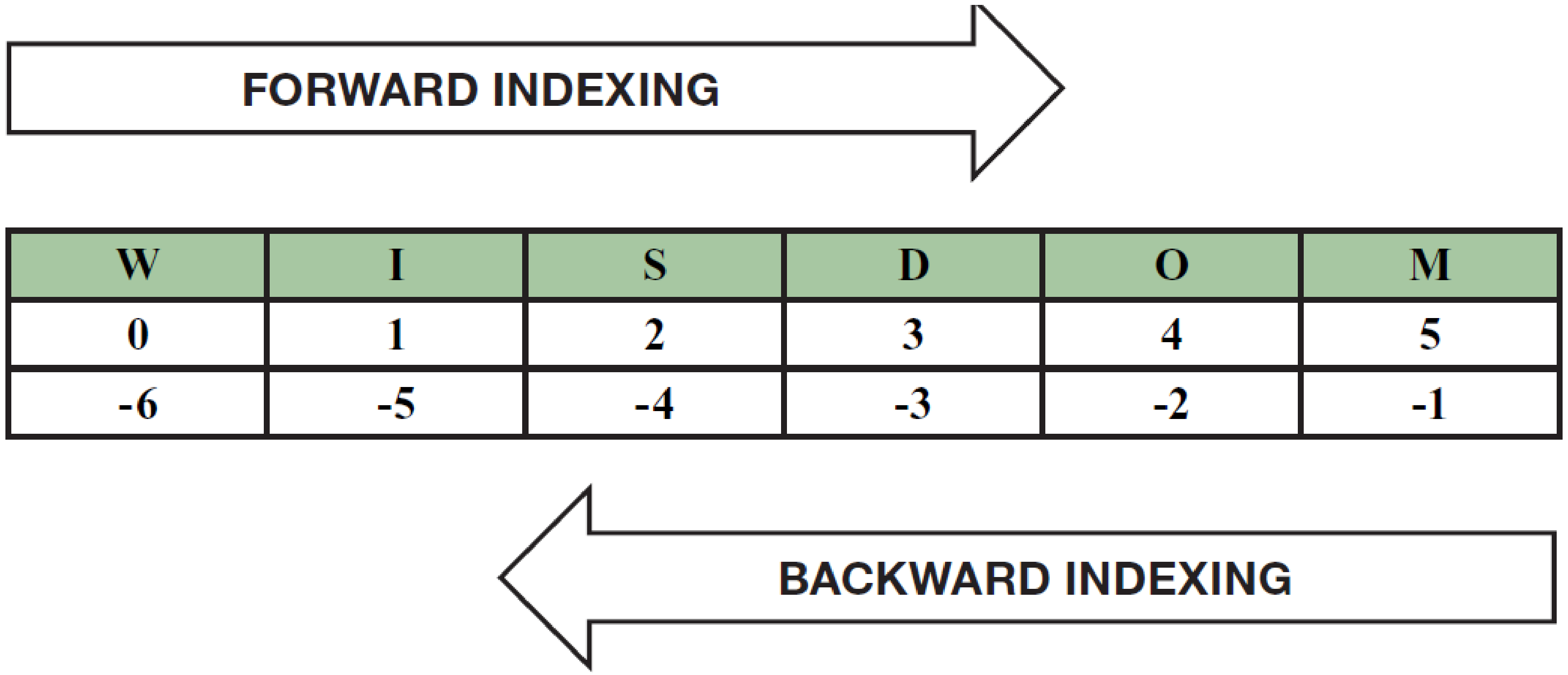
Python Syntax

max()	min()
max(iterable, key, default)	min(iterable, key, default)
max(arg1, arg2, *args, key)	min(arg1, arg2, *args, key)

It has three parameters:

- **iterable:** The iterable can be a list, tuple, set, dict, or string.
- **key: (Optional)** The built-in or user-defined function to be used for contrast.
- **default: (Optional)** The value to be refunded if the iterable is vacant

10.6. Looping with Strings

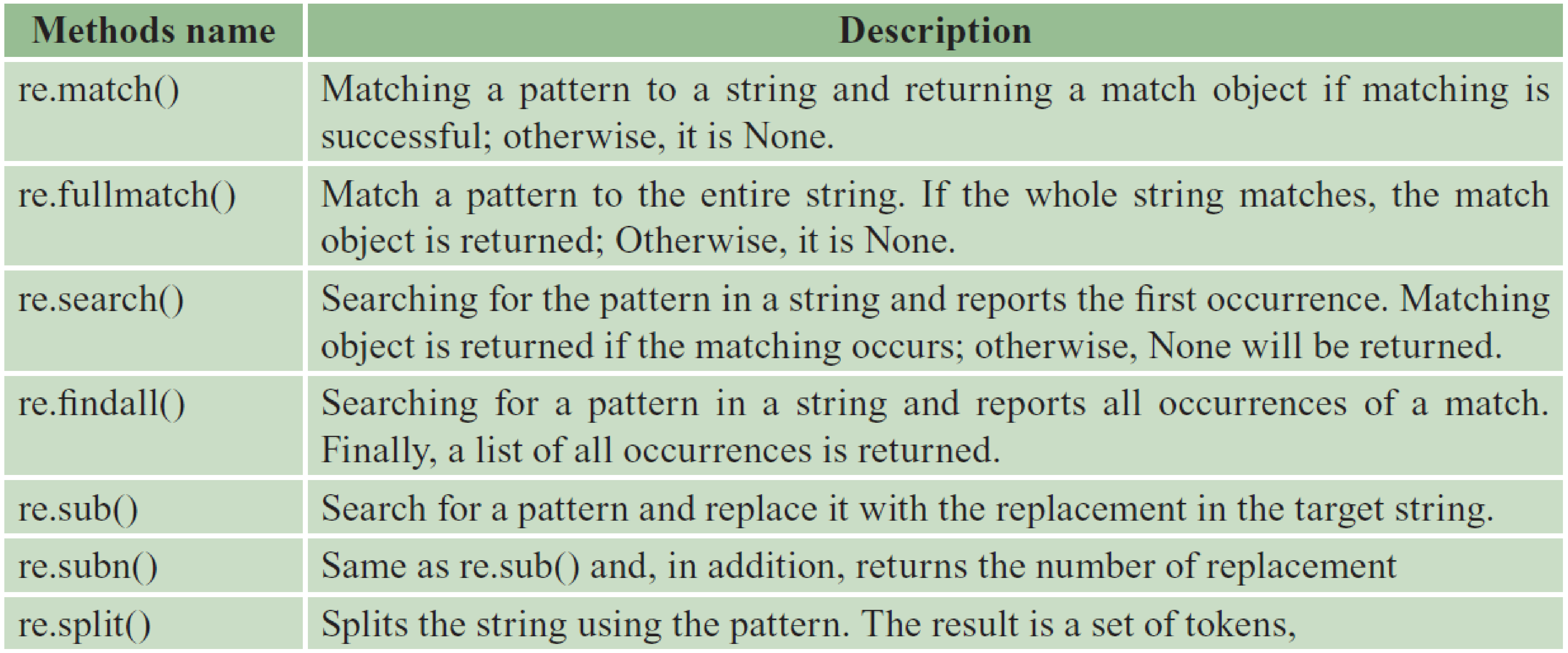


W	I	S	D	O	M
0	1	2	3	4	5
-6	-5	-4	-3	-2	-1

Figure 10.2: Indexing Operations

10.6. Regular Expressions

Table 10.5: re Package methods



Methods name	Description
re.match()	Matching a pattern to a string and returning a match object if matching is successful; otherwise, it is None.
re.fullmatch()	Match a pattern to the entire string. If the whole string matches, the match object is returned; Otherwise, it is None.
re.search()	Searching for the pattern in a string and reports the first occurrence. Matching object is returned if the matching occurs; otherwise, None will be returned.
re.findall()	Searching for a pattern in a string and reports all occurrences of a match. Finally, a list of all occurrences is returned.
re.sub()	Search for a pattern and replace it with the replacement in the target string.
re.subn()	Same as re.sub() and, in addition, returns the number of replacement
re.split()	Splits the string using the pattern. The result is a set of tokens,

Table 10.6: Character Class

Regular expression	Description
.	Any character, including a special character
\d	Any digit (0-9)
\D	Any character except the digits
\s	Space character
\S	Any character except the space character
\w	Any word character (a-z, A-Z, 0-9)
\W	Any character except the word character

Table 10.7: Regular expression with description

Regular expression	Description
[abc]	One of those three characters, a or b or c
[^ab]	Except for a and b
[a-z]	Any character in lowercase
[a-zA-Z0-9]	Any alphanumeric character
[^a-zA-Z0-9]	Except for the alphanumeric character
[0-9]	Any digits from 0 to 9
*	a* implies many a's like a, aa, aaa,aaa, etc., including 0.
+	a+ implies at least one a
A	Exactly one number of a
a?	At most or 1
a{k}	a{3} implies aaa

Matching of Full String

The re.fullmatch() method matches the regex pattern to the entire target string. If the full match of the pattern is present, then the match object is returned. Otherwise, it is None. The **syntax** of re.fullmatch**()** is given as :

Python Syntax

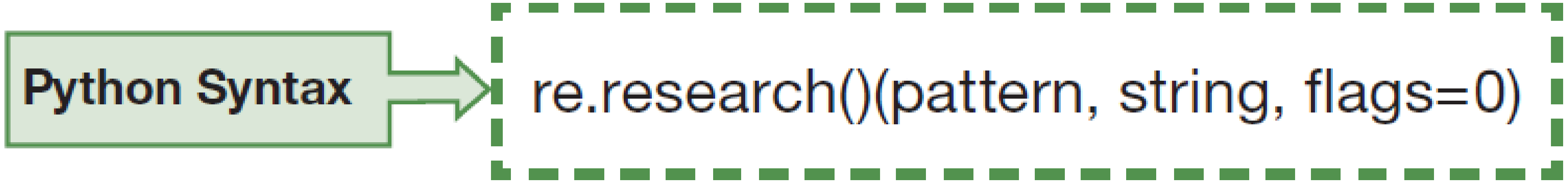
```
re.fullmatch(pattern, string, flags=0)
```

Has three parameters, namely, pattern, string, and flags.

- **pattern** -This is the regular expression used to match.
- **string** - This is the string that would be investigated to match the pattern at the start of the string.
- **flags – flags are modifiers. It is specified using** a bitwise OR.

On success, there. match function returns a match object; on failure, it returns None.

Search Function



```
re.research()(pattern, string, flags=0)
```

- **pattern -**This is the regular expression used to match.
- **string -** This is the string that would be investigated to match the pattern at the start of the string.
- **flags– flags are modifiers. It is specified using** a bitwise OR.

Substitute or Replacement function

Using this method, one can replace the matched regular expression in the target string. Here the word sub indicates the substitution or replacement. The method returns a string in which any occurrences that match are replaced with the value of the replaced variable. The syntax for re.sub() is:

Python Syntax

```
re.sub(pattern, replace, string)
```