St. Aloysius Sr.Sec.School, Cantt., Jabalpur

Class 12 (IP)

Unit 1 : (week 2)

Review of Python

Dated – 05/04/2020

Welcome children –

In class XI , you had studied the concepts of Python and many more concepts in it. So why not take a quick review of the past. At the end of the learning there will be questions for you to examine yourself.

Introduction –

Python is an interpreted language and we use interpreter to execute the statements in Python. We always worked in script mode to write our statements. A Python program constitutes several elements such as statements, expressions, functions, comments, etc., which looks like this

Programs --------modules--------statements------expressions------ $\searrow$

Objects

Also – a Hash (#) or Triple Double Quotation marks (“””) which are used for writing single line comments that do not span multiple lines and the second one mentioned is used to write multiple line comments.

Also - values are always a number or a letter or a string. Use of = equal operator helps in assigning values to variable.

Example : A = 20

Also – Identity refers to the address of the variable in memory which does not change once created.

Example : print id(A)

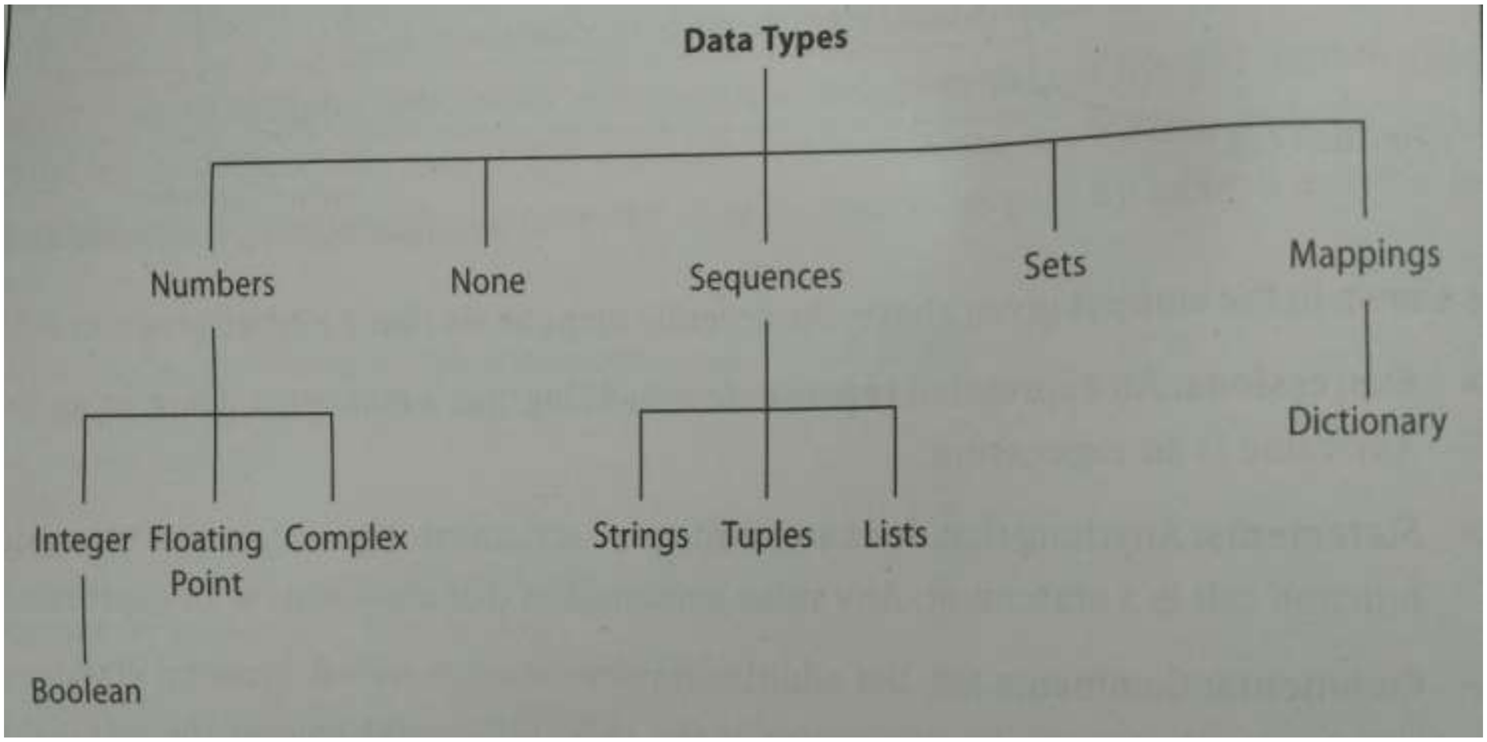
Also – In Python we do not declare a variable explicitly with its type. Python automatically allocates the related data type. Hence the type of the variable is same as the type of value assigned to it.

Example :   X = 45

This makes the variable X as integer type

<u>Data Types</u> -

Here have a look at the classification of Data Type in Python.



1. Numeric Data Type : This stores numeric values and are these types: Integer or Long, Float point, Complex numbers, Boolean.
   Example : BOOL1 = 6 == 6 * 1

   ```
                print  BOOL1
   OUTPUT : True
   ```

2. None : This is a special data type with an unidentified value. It indicates the absence of the value.
3. Sequence : It is an ordered collection of items, indexed by integers which can be positive or negative. The three type are Strings, lists and tuples.

4. Sets : It is an unordered collection of values of any type with no duplicate entry. It is immutable.

5. Mappings : This data is unordered and mutable. Dictionary in Python is one of this kind.

Dynamic Typing -

Python has a unique feature known as dynamic typing. A single variable can be used for storing different types of values. It allows the variable to redefine with different types such as float, integer, strings, list, tuple, dictionary etc.

Example :
```
A = 25

print (A)

A = 55.6

print (A)

A = ‘Lockdown’

print (A)

A = [5,6,7,8]

print (A)
```

All values assigned are different but the variable is the same.

Redefining the same for various values. Can you think of what is the advantage of this dynamic typing?

Keywords -

The reserve words are the keywords in Python with specific meaning for the interpreter.

Examples : None, True, False , and , or , not, break, continue, del, def, except, for, from, try, return, pass, with, in, else, elif, is, as, class, finally, while, etc.

Operators and Operands –

Data can be manipulated in Python through operators. The commonly used operators are –

Arithmetic operators - + , - , * , / ,// ,% , **

Assignment operators - = , +=, -=, *=, /=, //= , %=, **=

Relational operators - > , >= , < , <= , !=, ==

Logical operators – and, or, not

Membership operators – in

Input and output -

Python provides the two built-in library functions :

1. **input ()** – This function helps in accepting the user's input as a string and stores it in the variable which is assigned with the = equal operator. This always fetches a string value from the keyboard. The input() function takes one string argument (called prompt). During execution, this shows the prompt to the user and waits for

the user to input a value from the keyboard. The value received from the keyboard is stored in a variable.

Example 1 :
```
name = input("Enter your name : ")
print("You are ", name)
```

Output :
```
Enter your name : Yogesh
You are Yogesh
```

Example 2 :
```
me = int(input("Enter your age : "))
print("Your age is  ", me)
```

Output :
```
Enter your age : 14
Your age is 14
```

2. **eval () -** This method takes a string as an argument, evaluates it as a number and returns the numeric result (int or float as the case may be). If it is not a string or if it cannot be evaluated as a number then eval() returns an error.
3. **print () –** This is a function which helps to display the content on the screen. The content also called argument is specified within the (). Important – a print without a value or argument will simply jump to the next line.

   Example :
   ```
   A = "Situation"
   B = "During Lockdown"
   print(A,B)
   ```

   Output :
   ```
   Situation During Lockdown
   ```

<u>Type Casting</u> –

We can change the data type of a variable in Python from one type to another. There are two ways to do this -

- **explicit conversion –**

  This method is known as type casting, when the change is made deliberately i.e., the programmer forces it in the program.

  Syntax :

  (new_data_type) (expression)

  (there is a risk of data loss)

  Example :

  ```
  A =  30.75
  print(int(A))
  ```

  Output : 30

  - The integer part of 30.75

  See the table here –

Function	Description
int(A)	Converts A into an integer
float(A)	Converts A into a floating point value
chr(A)	Converts A into a character
str(A)	Converts A into a string

- **implicit conversion –**

  This method is known as coercion means automatically the data is converted and is not instructed by the programmer instructions.

  Example :

  ```
  A = 5
  B = 10.2
  C = A + B
  print C
  ```

  Output : 15.2

This is also due to type promotion which allows conversion of data into a wider sized data type without any loss of information.

Flow of execution –

The way in which the statements are executed defines the flow of execution in a Python program, which is categorized as –

1. Sequential 2. Selection/Conditional 3. Iteration/Looping

Sequential statements – The statements in a python program are executed one after the other i.e., from the beginning to the end.

Selection/conditional statements – The execution of the statements is based on whether a condition evaluated is true or false on the basis of the operators used basically – relational and logical operators.

The if statement increases the utilization in the program.

Example of if :
```
A = 20
if A == 20 :
    print(“the value is twenty”)
if A == 21 :
    print(“the value is twenty one”)
```
Output :
```
the value is twenty
```
Example of if.. else :
```
A = 21
if A == 20 :
    print(“the value is twenty”)
else :
    print(“the value is twenty one”)
```
Output :
```
the value is twenty one
```

<u>**Looping constructs**</u> **–**

The capability to execute a set of statements in a program repetitively based on a condition is called a looping construct.
There are two types of looping contructs –
for loop   while loop

- **for loop –** The for loop is used to repeat itself over a range of values or a sequence one by one. It is executed for each of these items in the range. When all the items in the range are exhausted the body of the loop is not executed any more.

  Syntax of for loop :

  ```
  for <Control_variable> in <sequence/items in range> :
          <statements in body of loop>
  else : # optional else
          <statements>
  ```

  The most commonly used built-in library function used with for loop is the range()

  range () – It helps in creating a list of numbers starting with the start and ending with one less than the stop

  Syntax :

  range(start,stop,step)

  the first and the last parameters are optional.

  Example :

	Output
range(10)	……………….
range(1,10)	……………….
range(1,10,2)	……………….
range(10,1,-1)	……………….
range(10,1,-2)	……………….

  Can you give the output ??????

  Example :

  ```
  for x in range(5):
          print ((x * 5)+2)
  ```

  Can you give the output ??????

- **while loop** –
  This also helps in executing the set of statements till the defined condition is true and as soon as the condition evaluates to false, control passes to the first line written after the loop.

Syntax of while loop :

```
while <test_expression> :
        Body of the while loop
else : # else is optional
        Body of else
```

Example :

```
a = 5
b = 10
while a <= b :
        print (a)
        a = a + 2
```

Output :

```
5
7
9
```

Can you find out how this output was derived ???????

Example :

```
a = 5
b = 10
while b > a :
        print (b)
        b = b - 2
```

Output :

```
………..

………..
```

<u>Strings</u> –

In Python a string is a sequence of characters within quotes. A single character in a string is accessed using an index (positive or negative integer).

Forward indexing means integers from 0 onwards

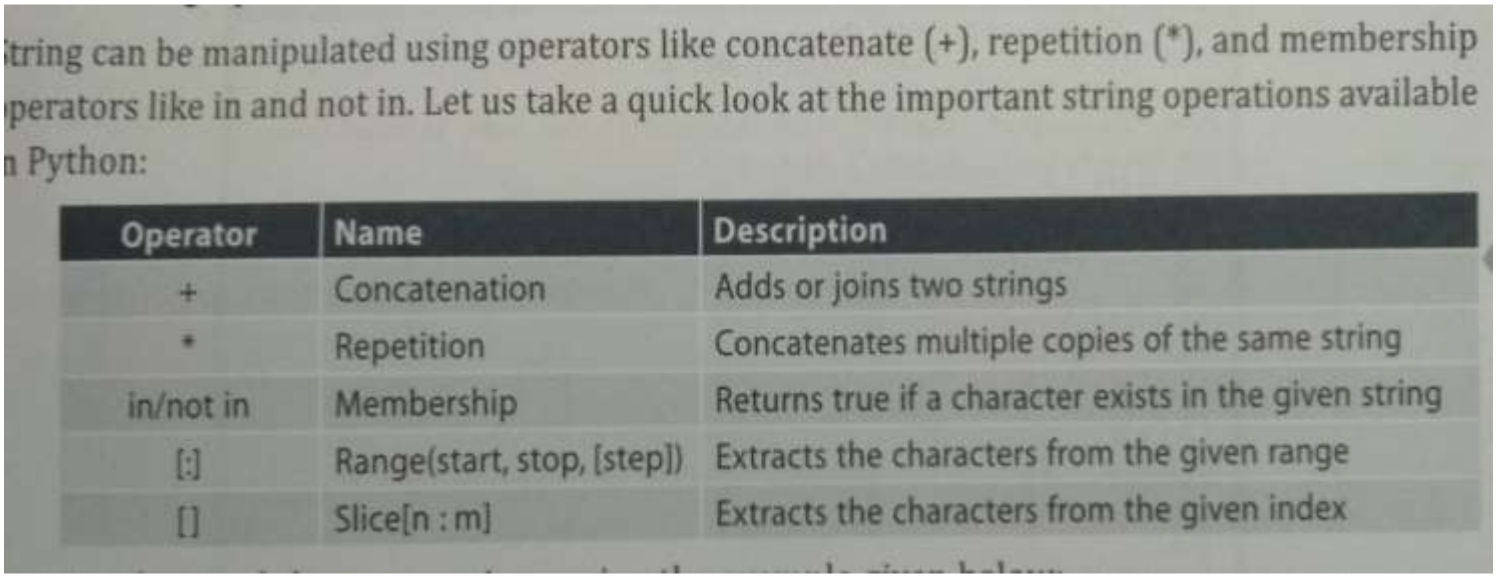
0 1 2 3 4 5 6 7 8 9 10 ……..

Backward indexing means integers from -1 to lower integer values

-10 -9 -8 -7 -6 -5 -4 -3 -2 -1

Remember strings are immutable.

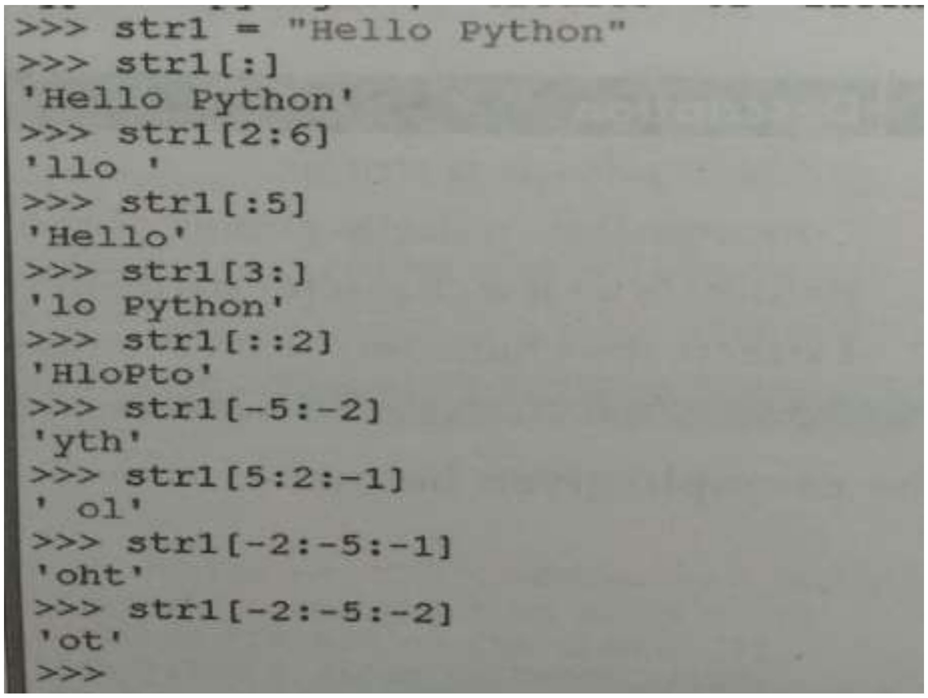
- **String operations** -



tring can be manipulated using operators like concatenate (+), repetition (*), and membership perators like in and not in. Let us take a quick look at the important string operations available n Python:

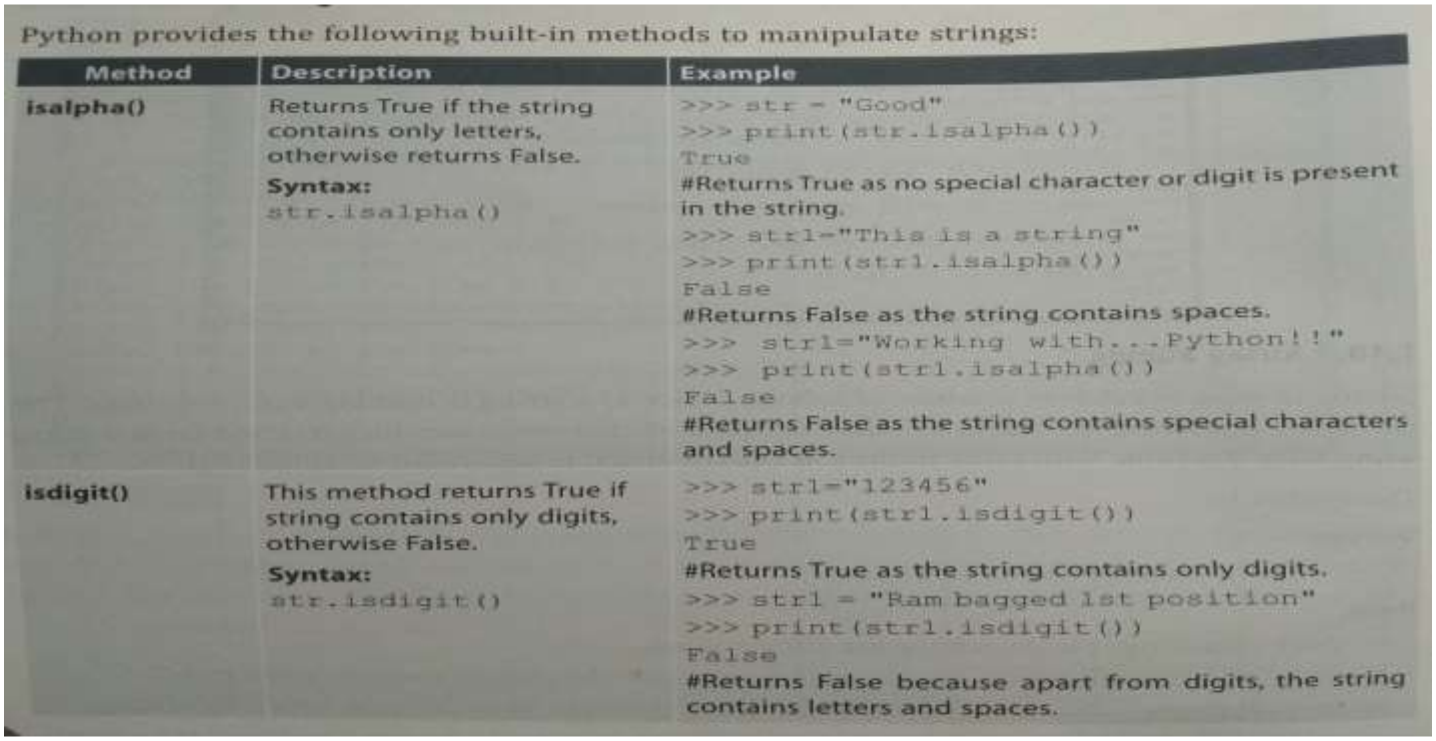
Operator	Name	Description
+	Concatenation	Adds or joins two strings
*	Repetition	Concatenates multiple copies of the same string
in/not in	Membership	Returns true if a character exists in the given string
[:]	Range(start, stop, [step])	Extracts the characters from the given range
[]	Slice[n : m]	Extracts the characters from the given index

- **Slicing of strings -**



```
>>> str1 = "Hello Python"
>>> str1[:]
'Hello Python'
>>> str1[2:6]
'llo '
>>> str1[:5]
'Hello'
>>> str1[3:]
'lo Python'
>>> str1[::2]
'HloPto'
>>> str1[-5:-2]
'yth'
>>> str1[5:2:-1]
' ol'
>>> str1[-2:-5:-1]
'oht'
>>> str1[-2:-5:-2]
'ot'
>>>
```

- **String Functions in Python -**



Python provides the following built-in methods to manipulate strings:

Method	Description	Example

isalpha()	Returns True if the string contains only letters, otherwise returns False.
	Syntax:
	`str.isalpha()`
		`>>> str = "Good"`
		`>>> print(str.isalpha())`
		`True`
		#Returns True as no special character or digit is present in the string.
		`>>> str1="This is a string"`
		`>>> print(str1.isalpha())`
		`False`
		#Returns False as the string contains spaces.
		`>>> str1="Working with...Python!!"`
		`>>> print(str1.isalpha())`
		`False`
		#Returns False as the string contains special characters and spaces.

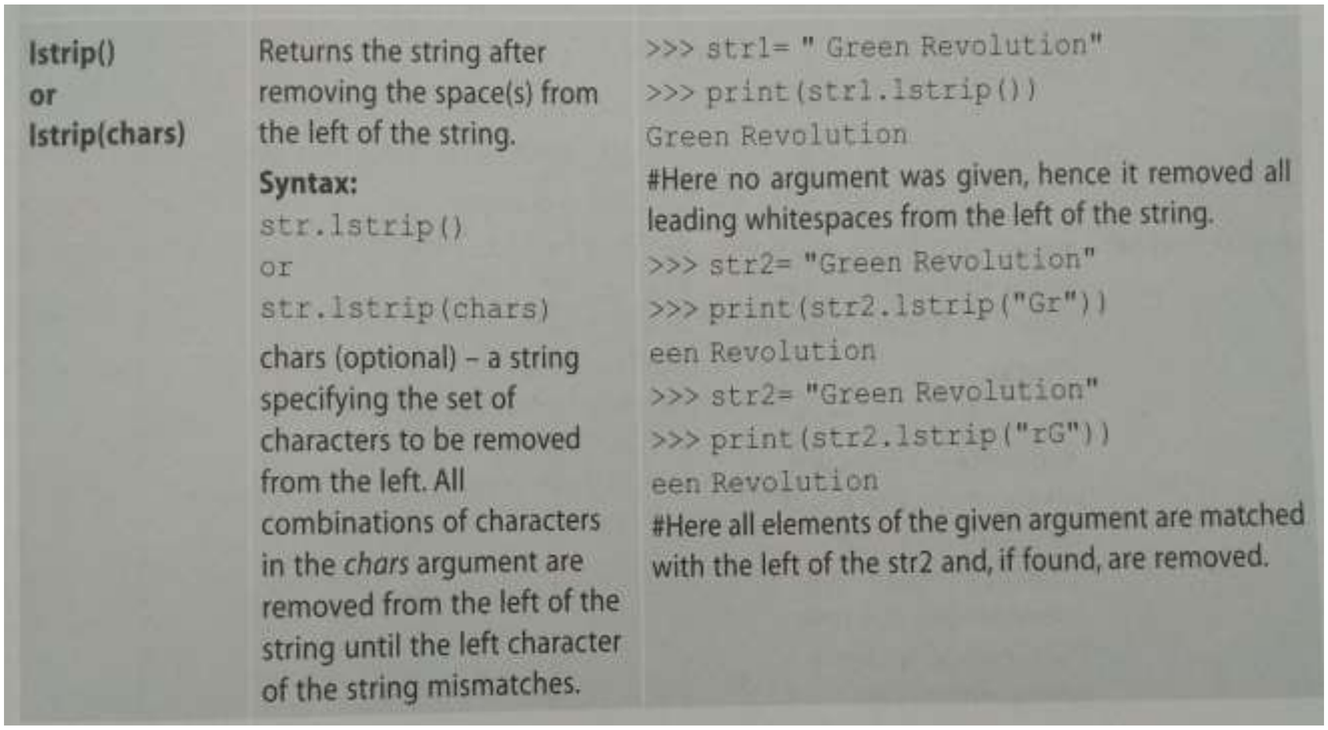
isdigit()	This method returns True if string contains only digits, otherwise False.
	Syntax:
	`str.isdigit()`
		`>>> str1="123456"`
		`>>> print(str1.isdigit())`
		`True`
		#Returns True as the string contains only digits.
		`>>> str1 = "Ram bagged 1st position"`
		`>>> print(str1.isdigit())`
		`False`
		#Returns False because apart from digits, the string contains letters and spaces.

lower()	Converts all the uppercase letters in the string to lowercase.
	Syntax:
	`str.lower()`
		`>>> str1= "Learning PYTHON"`
		`>>> print(str1.lower())`
		`learning python`
		#Converts uppercase letters only to lowercase.
		`>>> str1= "learning python"`
		`>>> print(str1.lower())`
		`learning python`
		#if already in lowercase, then it will simply return the string.

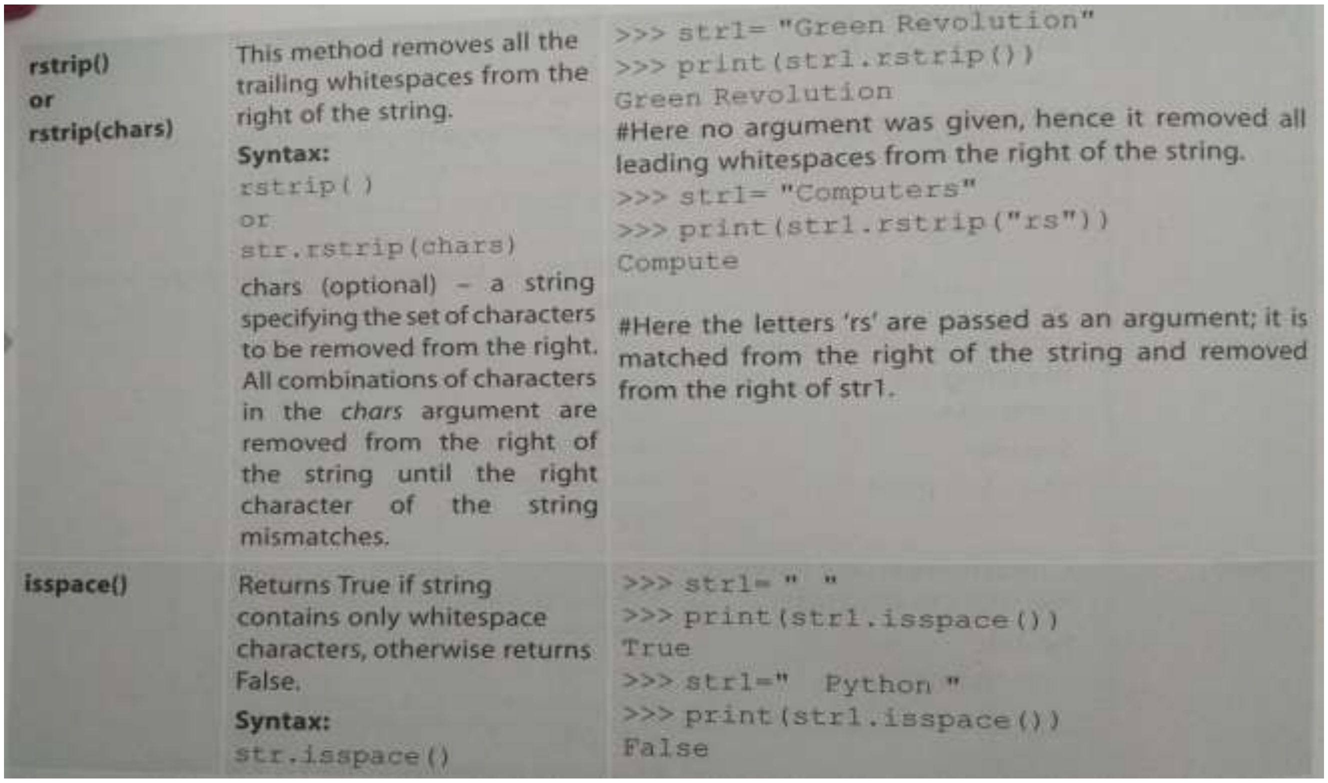
islower()	Returns True if all letters in the string are in lowercase.
	Syntax:
	`str.islower()`
		`>>> str1 ="python"`
		`>>> print(str1.islower())`
		`True`
		`>>> str1 = "Python"`
		`>>> print(str1.islower())`
		`False`

upper()	Converts lowercase letters in the string to uppercase.
	Syntax:
	`str.upper()`
		`>>> var1= "Welcome"`
		`>>> print(var1.upper())`
		`WELCOME`
		`>>> var1= "WELCOME"`
		`>>> print(var1.upper())`
		`WELCOME`
		#if already in uppercase, then it will simply return the string.

isupper()	Returns True if the string is in uppercase.
	Syntax:
	`str.isupper()`
		`>>> str1= "PYTHON"`
		`>>> print(str1.isupper())`
		`True`
		`>>> str1= "PythOn"`
		`>>> print(str1.isupper())`

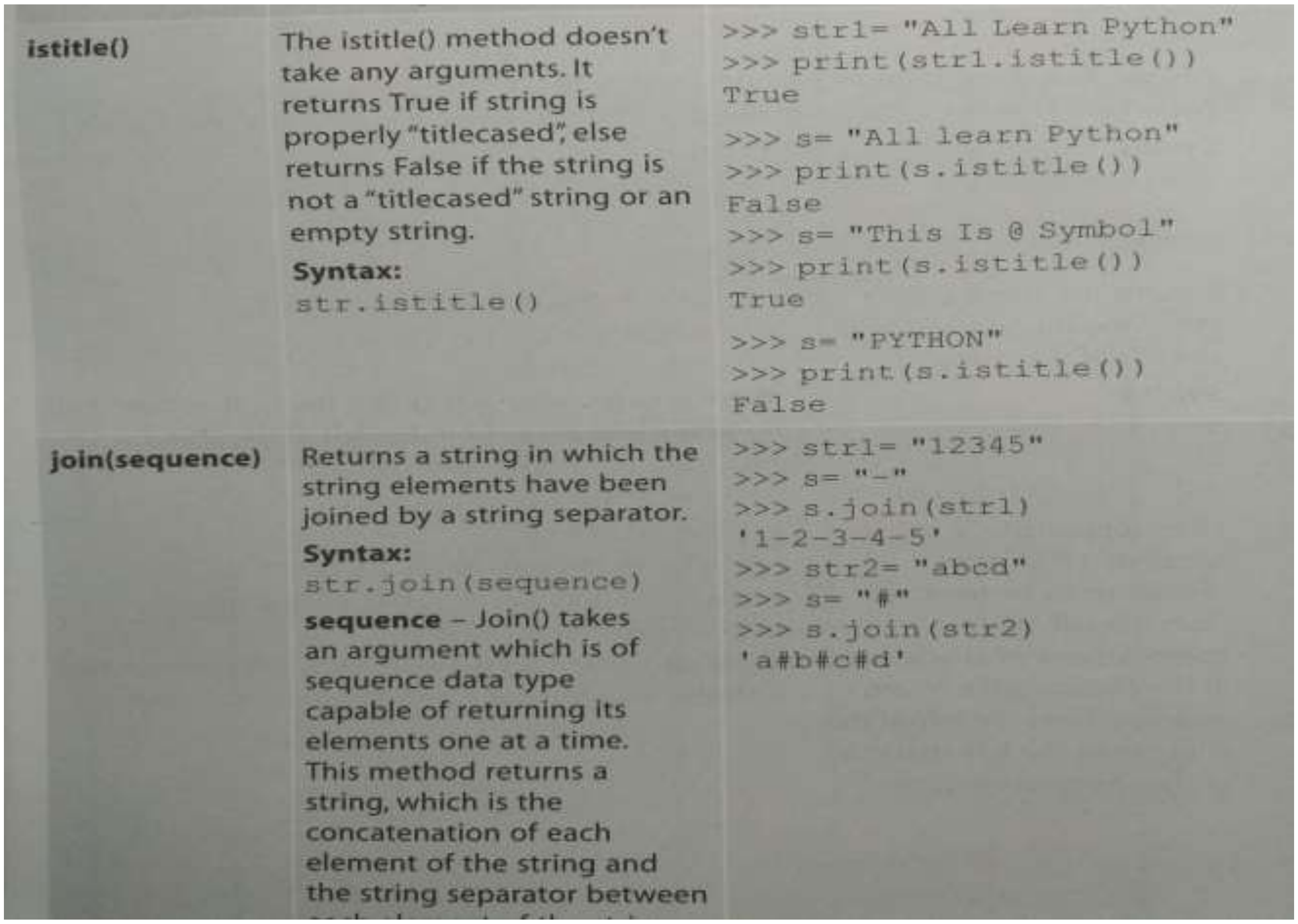


lstrip()
or
lstrip(chars)
	Returns the string after removing the space(s) from the left of the string.
	Syntax:
	`str.lstrip()`
	`or`
	`str.lstrip(chars)`
	chars (optional) – a string specifying the set of characters to be removed from the left. All combinations of characters in the *chars* argument are removed from the left of the string until the left character of the string mismatches.
		`>>> str1= " Green Revolution"`
		`>>> print(str1.lstrip())`
		`Green Revolution`
		#Here no argument was given, hence it removed all leading whitespaces from the left of the string.
		`>>> str2= "Green Revolution"`
		`>>> print(str2.lstrip("Gr"))`
		`een Revolution`
		`>>> str2= "Green Revolution"`
		`>>> print(str2.lstrip("rG"))`
		`een Revolution`
		#Here all elements of the given argument are matched with the left of the str2 and, if found, are removed.



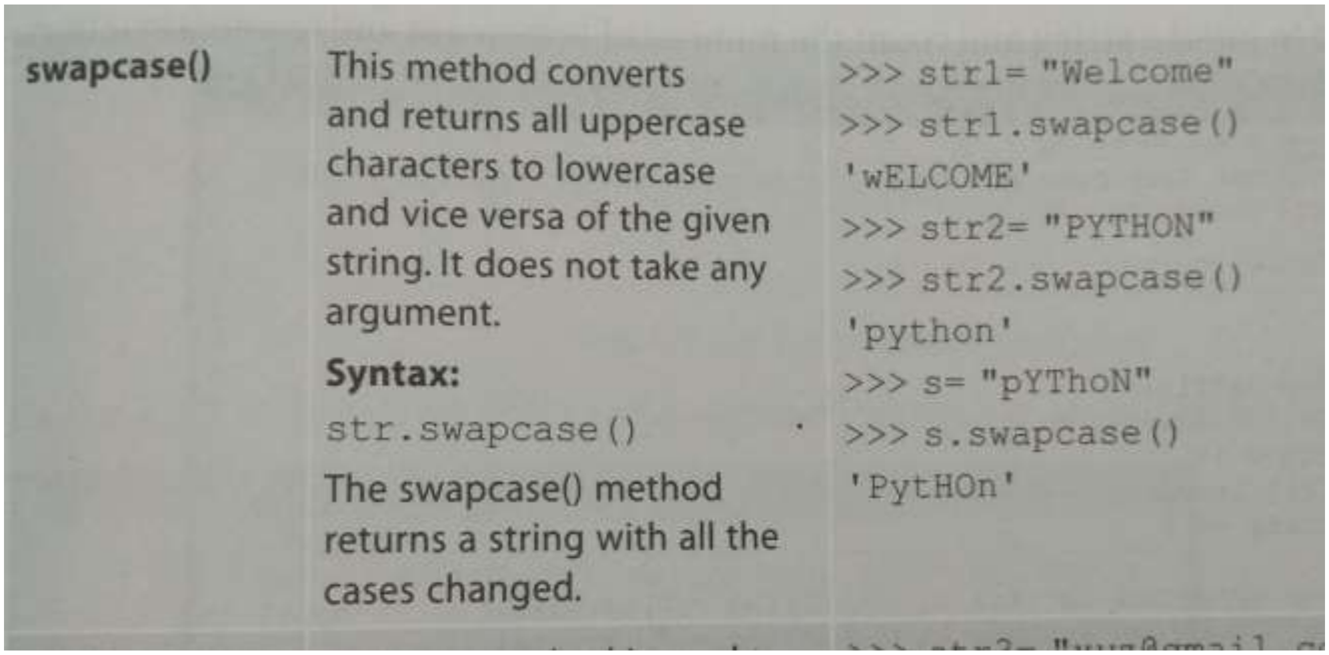
rstrip()
or
rstrip(chars)
	This method removes all the trailing whitespaces from the right of the string.
	Syntax:
	`rstrip( )`
	`or`
	`str.rstrip(chars)`
	chars (optional) – a string specifying the set of characters to be removed from the right. All combinations of characters in the *chars* argument are removed from the right of the string until the right character of the string mismatches.
		`>>> str1= "Green Revolution"`
		`>>> print(str1.rstrip())`
		`Green Revolution`
		#Here no argument was given, hence it removed all leading whitespaces from the right of the string.
		`>>> str1= "Computers"`
		`>>> print(str1.rstrip("rs"))`
		`Compute`
		#Here the letters 'rs' are passed as an argument; it is matched from the right of the string and removed from the right of str1.

isspace()	Returns True if string contains only whitespace characters, otherwise returns False.
	Syntax:
	`str.isspace()`
		`>>> str1= "  "`
		`>>> print(str1.isspace())`
		`True`
		`>>> str1="  Python "`
		`>>> print(str1.isspace())`
		`False`

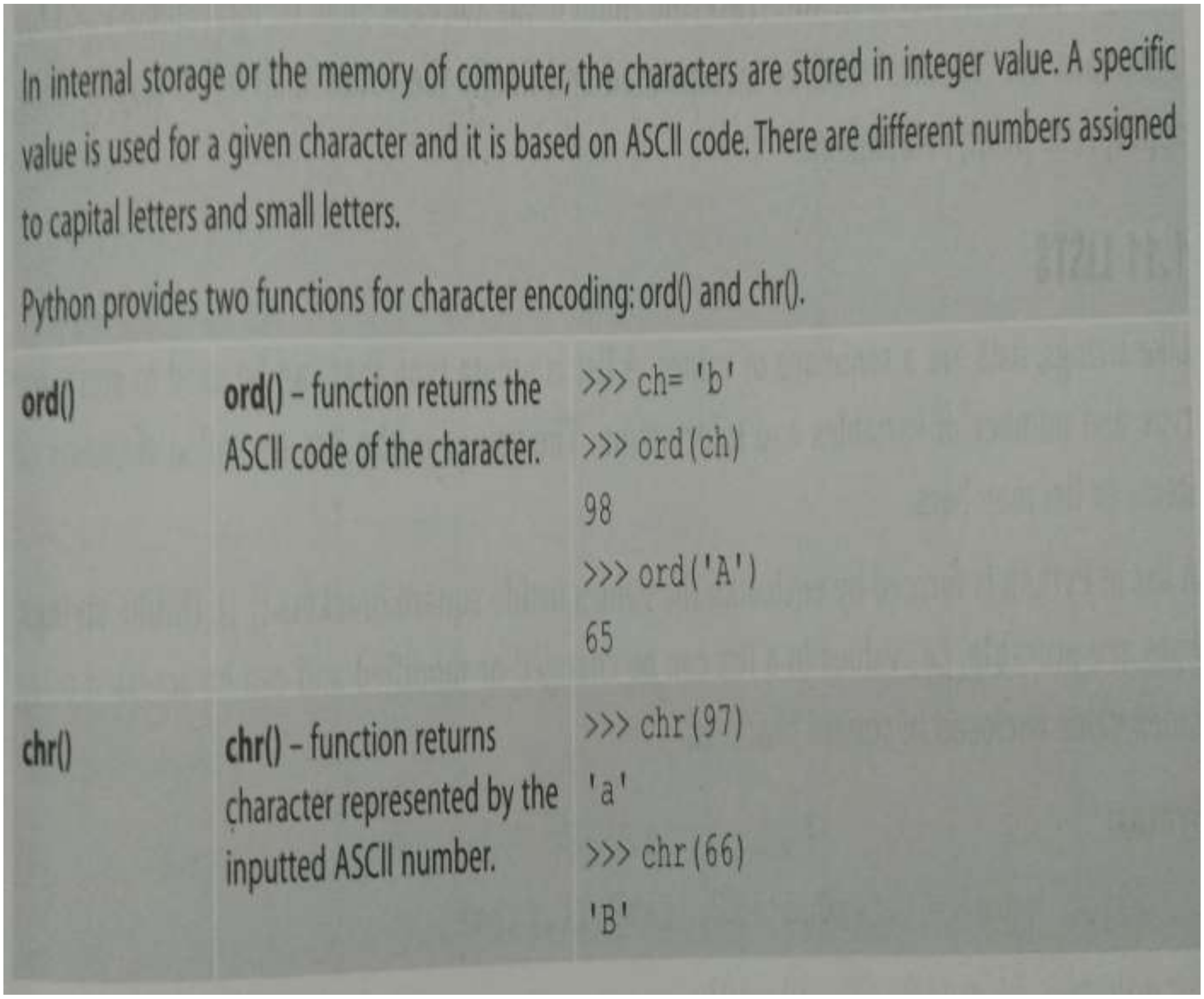


istitle()	The istitle() method doesn't take any arguments. It returns True if string is properly "titlecased", else returns False if the string is not a "titlecased" string or an empty string.
	Syntax:
	`str.istitle()`
		`>>> str1= "All Learn Python"`
		`>>> print(str1.istitle())`
		`True`
		`>>> s= "All learn Python"`
		`>>> print(s.istitle())`
		`False`
		`>>> s= "This Is @ Symbol"`
		`>>> print(s.istitle())`
		`True`
		`>>> s= "PYTHON"`
		`>>> print(s.istitle())`
		`False`

join(sequence)	Returns a string in which the string elements have been joined by a string separator.
	Syntax:
	`str.join(sequence)`
	sequence – Join() takes an argument which is of sequence data type capable of returning its elements one at a time. This method returns a string, which is the concatenation of each element of the string and the string separator between
		`>>> str1= "12345"`
		`>>> s= "-"`
		`>>> s.join(str1)`
		`'1-2-3-4-5'`
		`>>> str2= "abcd"`
		`>>> s= "#"`
		`>>> s.join(str2)`
		`'a#b#c#d'`



swapcase()	This method converts and returns all uppercase characters to lowercase and vice versa of the given string. It does not take any argument.
	Syntax:
	`str.swapcase()`
	The swapcase() method returns a string with all the cases changed.
		`>>> str1= "Welcome"`
		`>>> str1.swapcase()`
		`'wELCOME'`
		`>>> str2= "PYTHON"`
		`>>> str2.swapcase()`
		`'python'`
		`>>> s= "pYThoN"`
		`>>> s.swapcase()`
		`'PytHOn'`



In internal storage or the memory of computer, the characters are stored in integer value. A specific value is used for a given character and it is based on ASCII code. There are different numbers assigned to capital letters and small letters.

Python provides two functions for character encoding: ord() and chr().

ord()	**ord()** – function returns the ASCII code of the character.	`>>> ch= 'b'`
		`>>> ord(ch)`
		`98`
		`>>> ord('A')`
		`65`

chr()	**chr()** – function returns character represented by the inputted ASCII number.	`>>> chr(97)`
		`'a'`
		`>>> chr(66)`
		`'B'`

<u>These strings functions you need to memorize thoroughly.</u>

*************End *****************