Python Functions

Prof. Parthasarathy P V (M) 8951523822

Function (Method)	Syntax & Parameter values	Example

input()
• The `input()` function allows user input.
	input(*prompt*)
	prompt - A String, representing a default message before the input.
		`x = input('Enter your name:')`
		`print('Hello, ' + x)`
		Output:

String.split()
• The `split()` method splits a string into a list.
• You can specify the separator, default separator is any whitespace.
	string.split(*separator, maxsplit*)
	• ***separator*** Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator
	• ***maxsplit*** Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"
		Split a string into a list where each word is a list item:
		`txt = "welcome to the jungle"`
		`x = txt.split()`
		`print(x)`
		Output:
		
		Split the string, using comma, followed by a space, as a separator:
		`txt = "hello, my name is Peter, I am 26 years old"`
		`x = txt.split(", ")`
		`print(x)`
		Output:
		
		Use a hash character as a separator:
		`txt = "apple#banana#cherry#orange"`
		`x = txt.split("#")`
		`print(x)`
		Output:
		
		Split the string into a list with max 2 items:
		`txt = "apple#banana#cherry#orange"`
		`# setting the maxsplit parameter to 1, will return a list with 2 elements!`
		`x = txt.split("#", 1)`
		`print(x)`
		Output:

len()
• The `len()` function returns the number of items in an object.
• When the object is a string, the `len()` function returns the number of characters in the string.
	len(object)
	• ***object*** Required. An object. Must be a sequence or a collection
		`mylist = "Hello"`
		`x = len(mylist)`
		Output:

string.isdigit()	***string.isdigit()***	Check if all the characters in the text are digits:

• The `isdigit()` method returns True if all the characters are digits, otherwise False.
• Exponents, like ², are also considered to be a digit.
	No parameters.	`a = "\u0030" #unicode for 0`
		`b = "\u00B2" #unicode for ²`
		`print(a.isdigit())`
		`print(b.isdigit())`
		Output:
		
		Check if all the characters in the text are digits:
		`txt = "50800"`
		`x = txt.isdigit()`
		`print(x)`
		Output:

string.isupper()
• The `isupper()` method returns True if all the characters are in upper case, otherwise False.
• Numbers, symbols and spaces are not checked, only alphabet characters.
	string.isupper()
	
	No parameters.
		Check if all the characters in the text are in upper case:
		`txt = "THIS IS NOW!"`
		`x = txt.isupper()`
		`print(x)`
		Output:
		
		Check if all the characters in the texts are in upper case:
		`a = "Hello World!"`
		`b = "hello 123"`
		`c = "MY NAME IS PETER"`
		`print(a.isupper())`
		`print(b.isupper())`
		`print(c.isupper())`
		Output:

string.islower()
• The `islower()` method returns True if all the characters are in lower case, otherwise False.
• Numbers, symbols and spaces are not checked, only alphabet characters.
	string.islower()
	
	No parameters.
		Check if all the characters in the texts are in lower case:
		`a = "Hello world!"`
		`b = "hello 123"`
		`c = "mynameisPeter"`
		`print(a.islower())`
		`print(b.islower())`
		`print(c.islower())`
		Output:
		
		Check if all the characters in the text are in lower case:
		`txt = "hello world!"`
		`x = txt.islower()`
		`print(x)`
		Output:

print()
• The `print()` function prints the specified message to the screen, or other standard output device.
• The message can be a string, or any other object, the object will be converted into a string before written to the screen.
	print*(object(s), sep=separator, end=end, file=file, flush=flush)*
	
	object(s): Any object, and as many as you like. Will be converted to string before printed
	sep='separator' Optional. Specify how to separate the objects, if there is more than one. Default is ' '
	
	end='end' Optional. Specify what to print at the end. Default is '\n' (line feed)
	
	file Optional. An object with a write method. Default is sys.stdout
	
	flush Optional. A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False
		Print a message onto the screen:
		`print("Hello World")`
		Output:
		
		Print more than one object:
		`print("Hello", "how are you?")`
		Output:
		
		Print a tuple:
		`x = ("apple", "banana", "cherry")`
		`print(x)`
		Output:
		
		Print two messages, and specify the separator:
		`print("Hello", "how are you?", sep="---")`
		Output:

***

Program (3.a): Write a Python program that accepts a sentence and find the number of words, digits, uppercase letters and lowercase letters.

Aim

Procedure

Program

```
s = input("Enter a sentence: ")
w, d, u, l = 0, 0, 0, 0
l_w = s.split()
w =  len(l_w)
for c in s:
    if c.isdigit():
        d = d + 1
    elif c.isupper():
        u = u + 1
    elif c.islower():
        l = l + 1

print ("No of Words: ", w)
print ("No of Digits: ", d)
print ("No of Uppercase letters: ", u)
print ("No of Lowercase letters: ", l)
```

Output

Result

***

Python Functions

Prof. Parthasarathy P V (M) 8951523822

Function (Method)	Syntax & Parameter values	Example
Python Conditions and If statements • Equals: a == b • Not Equals: a != b • Less than: a < b • Less than or equal to: a <= b • Greater than: a > b • Greater than or equal to: a >= b	**Indentation** Python relies on indentation (whitespace at the beginning of a line) to define scope in the code. Other programming languages often use curly-brackets for this purpose.	a = 33 b = 200 if b > a: print("b is greater than a") **Output:** --- a = 33 b = 200 if b > a: print("b is greater than a") # you will get an error

elif	The **elif** keyword is Python's way of saying "if the previous conditions were not true, then try this condition".	a = 33
		b = 33
		if b > a:
		  print("b is greater than a")
		elif a == b:
		  print("a and b are equal")
		Output:

else	The **else** keyword catches anything which isn't caught by the preceding conditions.	a = 200
		b = 33
		if b > a:
		  print("b is greater than a")
		elif a == b:
		  print("a and b are equal")
		else:
		  print("a is greater than b")
		Output:

Short Hand If	If you have only one statement to execute, you can put it on the same line as the if statement.	if a > b: print("a is greater than b") **Output:**
Short Hand If ... Else	If you have only one statement to execute, one for **if**, and one for **else**, you can put it all on the same line.	a = 2 b = 330 print("A") if a > b else print("B") **Output:**

Nested If	You can have **if** statements inside **if** statements, this is called *nested* **if** statements.	x = 41
		
		if x > 10:
		  print("Above ten,")
		  if x > 20:
		  print("and also above 20!")
		  else:
		  print("but not above 20.")
		Output:

The pass Statement	**if** statements cannot be empty, but if you for some reason have an **if** statement with no content, put in the **pass** statement to avoid getting an error.	a = 33 b = 200 if b > a: pass **Output:**
Python "for" Loops • A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). • This is less like the **for** keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages. • With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.		
Python "for" Loops	Print each fruit in a fruit list	fruits = ["apple", "banana", "cherry"] for x in fruits: print(x) **Output:**
Looping Through a String	Even strings are iterable objects, they contain a sequence of characters.	for x in "banana": print(x) **Output:**

The break Statement
With the **break** statement we can stop the loop
	before it has looped through all the items:
	Exit the loop when x is "banana":
		fruits = ["apple", "banana", "cherry"]
		for x in fruits:
		  print(x)
		  if x == "banana":
		  break
		Output:

	Exit the loop when x is "banana", but this time the break comes before the print	fruits = ["apple", "banana", "cherry"]
		for x in fruits:
		  if x == "banana":
		  break
		  print(x)
		Output:

<table>
<tr><td>The continue Statement</td><td>With the continue statement we can stop the current iteration of the loop, and continue with the next</td><td>fruits = ["apple", "banana", "cherry"]
for x in fruits:
  if x == "banana":
  continue
  print(x)
Output:</td></tr>
<tr><td rowspan="3">The range() Function
• To loop through a set of code a specified number of times, we can use the function.
• The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and ends at a specified number.</td><td>• Note that range(6) is not the values of 0 to 6, but the values 0 to 5.</td><td>for x in range(6):
  print(x)
Output:</td></tr>
<tr><td>• The range() function defaults to 0 as a starting value, however it is possible to specify the starting value by adding a parameter: range(2, 6), which means values from 2 to 6 (but not including 6)</td><td>for x in range(2, 6):
  print(x)
Output:</td></tr>
<tr><td>• The range() function defaults to increment the sequence by 1, however it is possible to specify the increment value by adding a third parameter: range(2, 30, 3)</td><td>for x in range(2, 30, 3):
  print(x)
Output:</td></tr>
<tr><td rowspan="2">Else in For Loop
The <code>else</code> keyword in a <code>for</code> loop specifies a block of code to be executed when the loop is finished</td><td>• Print all numbers from 0 to 5, and print a message when the loop has ended.
Note: The <code>else</code> block will NOT be executed if the loop is stopped by a <code>break</code> statement.</td><td>for x in range(6):
  print(x)
else:
  print("Finally finished!")
Output:</td></tr>
<tr><td>• Break the loop when <code>x</code> is 3, and see what happens with the <code>else</code> block</td><td>for x in range(6):
  if x == 3: break
  print(x)
else:
  print("Finally finished!")
Output:</td></tr>
<tr><td>Nested Loops</td><td>• A nested loop is a loop inside a loop.
• The "inner loop" will be executed one time for each iteration of the "outer loop":</td><td>adj = ["red", "big", "tasty"]
fruits = ["apple", "banana", "cherry"]

for x in adj:
  for y in fruits:
  print(x, y)</td></tr>
</table>

		Output:

The pass Statement	• `for` loops cannot be empty, but if you for some reason have a `for` loop with no content, put in the `pass` statement to avoid getting an error.	`for x in [0, 1, 2]:`
		`  pass`
		Output:

Program (3.b) : Write a Python program to find the string similarity between two given strings

Aim

Procedure

Program

```
str1 = input("Enter String 1 \n")
str2 = input("Enter String 2 \n")
if len(str2) < len(str1):
     short = len(str2)
     long = len(str1)
else:
     short = len(str1)
     long = len(str2)
     matchCnt = 0
for i in range(short):
     if str1[i] == str2[i]:
     matchCnt += 1
     print("Similarity between two said strings:")
print(matchCnt/long)
```

Output

Result

***