Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: String

Table of Contents

1. String Introduction:

A string is a sequence of characters. This sequence can be a combination of letters, numbers and special characters. However, computers cannot understand any of these characters except 0s and 1s. Therefore, all of these characters in Python are converted using the popular Unicode encoding technique and hence it is called a sequence of Unicode characters.

1.1 String Creation:

- Strings can be created by enclosing the characters either inside a single quote ('Hello World!!!') or a double quote ("Hello World!!!").
- Mixture such as ("This is an error') or ('This is an error") cannot be used for string creation.
- To represent multiple lines, triple quotes are used. For example:

```
'''Welcome to Python.
Python is easy and fun.
Let's get started'''
```

- String is case sensitive thus 'A' or 'a' has different Unicode values and represents two different strings.
- Space inside " " or ' ' is also a string as space is part of the special characters.

2. Indexing:

All the characters of a string can be accessed using the indexing technique. Indexing **ALWAYS** starts at 0. Therefore, the indexing of a string is always within a finite range based on the length of the string. Trying to access a character out of the index range will raise an **IndexError**. The index must also be an integer. We cannot use floats or other types, this will result into **TypeError**. Interestingly python allows negative indexing for its sequences. The last character of a string is indexed as -1, the second last as -2 and so on. **(Important Note: Space is also a character and indexed)**

Example:

Characters	P	y	t	h	o	n		i	s		e	a	s	y
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Negative Index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> str = 'Python is easy'
>>> print(str[0])
Output: P
>>> print(str[8])
Output: s
>>> print(str[6])
Output:  <-------- Space is printed
>>> print(str[-1])
Output: y
>>> print(str[-4])
Output: e
>>> print(str[15])
Output: IndexError
>>> print(str[-15])
Output: IndexError
>>> print(str[8.0])
Output: TypeError
```

To access the last character of a string where the total numbers of characters are not known, there are two possible ways to do it. Either the negative indexing technique can be used where the last character has an index of -1 or the built in **len()** function can be used. The len() function takes a string and returns the total number of characters. Remember to subtract 1 from the len function as the indexing starts from 0 to get the last index. For example:

```
>>> str = 'Python is easy'
>>> print(len(str))
Output: 14 #The output is 14 because there are total 14 characters including spaces in String
```

```
#Solution 1: To get the last character
>>> str = 'Python is easy'
>>> print(str[-1])
Output: y
```

```
#Solution 2: To get the last character
>>> str = 'Python is easy'
>>> print(str[len(str) - 1])
Output: y
```

3. Mutability of String:

Strings are immutable. This means that individual elements of a string cannot be changed once they have been assigned. We can only reassign different strings to the same name. For example

```
#Cannot Do
>>> str = 'Python is easy'
>>> str[2] = 'H'
#This line will give you an error
TypeError: 'str' object does not support item assignment
```

```
#Can Do
>>> str = 'Python is easy'
>>> print(str)
Output: Python is easy
>>> str = 'Why python?'
>>> print(str)
# str no longer represents 'Python as easy' since we changed str.
Output: Why python?
```

4. Iterating through a String:

A while or for loop can be used to iterate over a string character by character. While loop should be used if index along with the character is required or else for loop should be used. For example:

```
#Solution 1: Use index variable to control the loop
>>> index = 0
>>> str = 'Python is easy'
>>> while index < len(str):
...         print(str[index])
...         index += 1
```

```
#Solution 2: No need to maintain indexing
>>> str = 'Python is easy'
>>> for char in str:
...       print(char)
```

```
Output:
P
y
t
h
o
n

i
s

e
a
s
y
```

5. String Operations:

5.1 Concatenation:

Joining of two or more strings into a single one is called concatenation (merging). We use the '+' operator to concat or merge. This operator does not add any space in between.

```
>>> str = 'Python is easy'
>>> str = str + 'and fun'
>>> print(str)
Output: Python is easyand fun
#No space was printed.
```

Remember, space is also a special character. Therefore space can be added as a separate string with a + operator to add space in between or space can be used as the start or last character of a string depending on the situation.

```
#Method 1
>>> str = 'Python is easy'
>>> str = str + ' ' + 'and fun'
>>> print(str)
Output: Python is easy and fun
```

```
#Method 2
>>> str = 'Python is easy'
>>> str = str + ' and fun'
>>> print(str)
Output: Python is easy and
fun
```

5.2 Deletion

Remember string is immutable, therefore we cannot delete or remove individual characters from a string. But deleting the string entirely is possible using the del keyword.

```
#Cannot delete individual
>>> str = 'Python is easy'
>>> del str[0]
Output: TypeError: 'str'
object doesn't support
item deletion
```

```
#Can delete the entire string
>>> str = 'Python is easy'
>>> print(str)
>>> del str[0]
>>> print(str)
Outpur: NameError: name 's'
is not defined
```

5.3 Repetition

Strings can be repeated using the * operator. It can be used to print the same thing multiple times without using any loop

```
>>> str = 'Python is easy.'
>>> print(str * 3)
Output: Python is easy.Python is easy. Python is easy.
```

```
#Using loop
>>> str = 'Python is easy.'
>>> for x in range(0,3):
…        print(str)
```

```
Output:
Python is easy.
Python is easy.
Python is easy.
```

```
#Using * operator
>>> str = 'Python is easy.\n'
>>> print(str * 3)
```

5.4 Slicing

Slicing is a technique to access a range of sequence of characters from a string. Slicing is done using the square bracket.
String[start: stop[: step]]
start: From which index to start, inclusive. If not set, default value is 0
stop: At which index to stop slicing, exclusive. If not set, default value is last index of the String
step: Optional. How many step to take. If not set, default value is 1.

Characters	P	y	t	h	o	n		i	s		e	a	s	y
Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Negative Index	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
>>> str = 'Python is easy'
>>> print(str[7:9])
Output: is
>>> print(str[:8])
Output: Python i
>>> print(str[8:])
Output: s easy
>>> print(str[:])
Output: Python is easy
>>> print(str[2:11:2])
Output: to se
>>> print(str[8:4:-1])
Output: si n
```

6. Checking String Membership:

The '**in**' keyword can be used to find whether a string is part of another string or not. If the string is found the keyword will return True or else it will return False. It is used in the if statement as a condition.

```
>>> str = 'Python is easy'
>>> another_str = 'easy'
>>> print(another_str in str)
Output: True
>>> another_str = 'fun'
>>> print(another_str in str)
Output: False
```

7. Escape Sequence:

As mentioned earlier, strings are represented either using a single quote ('Hello World!!!') or a double quote ("Hello World!!!"). What if a single or double quote needs to be printed.

```
>>> str = 'Python 'is' easy'
>>> print(str)
Output: SyntaxError: Invalid Syntax
```

This will result in an error because the interpreter will be confused where the quote ends. To solve this one of the basic solution is to use single quote for the substring and represent the string using double quote or vice versa.

```
>>> str = ''Python 'is' easy''
>>> print(str)
Output: Python 'is' easy
>>> str = 'Python ''is'' easy'
>>> print(str)
Output: 'Python ''is'' easy
```

Another solution and better compared to the previous method is using escape sequence represented using (\ ). There are many special characters that can confuse the interpreter so using the escape sequence the problem can be easily solved. For printing quote, use the escape character (\") or (\'). More escape characters can be found in this link.

```
>>> str = 'Python \'is\' easy'
>>> print(str)
Output: Python 'is' easy
>>> str = ''Python \''is\'' easy''
>>> print(str)
Output: 'Python ''is'' easy
```

8. Formatting:

8.1 String:

The format() function that is available with the string object is very versatile and powerful in formatting strings. Format strings contain curly braces {} as placeholders or replacement fields which get replaced. Positional arguments can be used which is a list of parameters that can be accessed with index of parameter inside curly braces {index}. Keyword arguments can also be used which is a list of parameters of type key=value, that can be accessed with key of parameter inside curly braces {key}.

```
# default(implicit) order
>>> str = "Hello {}, Welcome to {}.".format("User", "python")
>>> print(str)
Output: Hello User, Welcome to python.

# order using positional argument
>>> str = "Hello {0}, Welcome to {1}.".format("User", "python")
>>> print(str)
Output: Hello User, Welcome to python.

# order using positional argument
>>> str = " Welcome to {1}, hello{0}.".format("User", "python")
>>> print(str)
Output: Welccome to python, hello user.

# order using keyword argument
>>>  str  =  "Hello  {user},  welcome  to{lang}.".format(user="User",
lang="python")
>>> print(str)
Output: Hello User, welcome to python.
```

8.2 Number:

The format() method can also be used to format the numbers. They can be used to control the floating points, format a number into binary, octal and many other formats.

```
>>> str = "Hello {0}, can you lend me {1:d}$?".format("Bob", 100)
>>> print(str)
Output: Hello Bob, can you lend me 100$?
>>> str = "I am {0}, {1}. I have only {2:4.2f}$.".format("sorry",
"Alice", 50.95876)
>>> print(str)
Output: I am sorry, Alice. I have only 50.96$.
```

9. ASCII:

ASCII (American Standard Code for Information Interchange) is a code for representing any character typed using keyboards as numbers, assigned from 0 to 127. Machines do not understand any characters or decimals. Instead the machine only understands binary numbers (0s and 1s). Therefore, the characters are first converted into ASCII values which is then converted in their corresponding binary numbers. Any character can be converted to its corresponding ASCII value using ord(character) function and number can be converted to its corresponding character using chr(int) function. The ASCII code is available here.

10. String Functions:

Python comes with many built-in string functions that you do not have to create from scratch. These functions can solve many problems in only one line. Below are some of the important functions that can be used for string. You can also use this link for reference.

```
>>> str = "HeLLo FrIeNDs"
>>> print(str.lower())
Output: hello friends
```

1) The **lower()** function changes all the uppercase letters of a given string into lowercase. The function does not take any parameters. It only changes alphabets, rest characters stay as it is.

```
>>> str = "HeLLo FrIeNDs"
>>> print(str.upper())
Output: HELLO FRIENDS
```

2) The **upper()** function changes all the lowercase letters of a given string into uppercase. The function does not take any parameters. It only changes alphabets, rest characters stay as it is.

```
>>> str = "   HeLLo FrIeNDs    "
>>> print(str.strip())
Output: HeLLo FrIeNDs
>>> str = "   HeLLo FrIeNDs    "
>>> print(str.strip(" HeNs"))
Output: LLo FrIeND
```

3) The **strip([chars])** function returns a string by removing characters from the left and right of a given string. The function takes an optional parameter char. If nothing is provided only white spaces before and after is removed. However, if a set of characters including white space is provided, then only those characters are removed from the given string.

```
>>> str = "hi friend hi"
>>> print(str.count("hi"))
Output: 2
>>> str = "hi friend hi"
>>> print(str.count("hi", 0, 11))
Output: 1
```

4) The **count(substring[, start, end])** function returns the number of times the substring in the parameter occurs in the given string. The function takes in a mandatory parameter that is the substring. However, the function also takes in optional start and end index of a string where start represents the starting index inclusive and ending index exclusive.

```
>>> str = "hi friend hi"
>>> print(str.startswith("hi"))
Output: True
>>> str = "hi friend hi"
>>> print(str.startswith("fr", 3, 4))
Output: False
```

5) The **startswith(prefix[, start, end])** function returns true if the string starts with the given substring or else returns false. The function takes one mandatory parameter prefix that is the substring. It has other start and end optional parameters which specifies the starting index inclusive and ending index exclusive.

```
>>> str = "hi friend hi"
>>> print(str.endswith("hi"))
Output: True
>>> str = "hi friend hi"
>>> print(str.endswith("hi", 0, 10))
Output: False
```

6) Similar to startswith(), The **endswith(prefix[, start, end])** function returns true if the string ends with the given substring or else returns false. The function takes one mandatory parameter prefix that is the substring. It has other start and end optional parameters that specify the starting index inclusive and ending index exclusive.

```
>>> str = "Hi friend hi friend"
>>> print(str.find("hi"))
Output: 10
>>> str = "Hi friend hi friend"
>>> print(str.endswith("hi", 0, 11))
Output: -1
```

7) The **find(sub[, start, end])** function returns the index of the first occurrence of the substring from a given string or else returns -1. The function takes one mandatory parameter sub that is the substring. It has other start and end optional parameters that specify the starting index inclusive and ending index exclusive to be searched from.

```
>>> str = "Hi friend hi friend hi"
>>> print(str.replace("hi", "bye"))
Output: Hi friend bye friend bye
>>> str = "Hi friend hi friend hi"
>>> print(str.replace("hi", "bye", 1))
Output: Hi friend bye friend hi
```

8) The **replace(old, new[, count])** takes in two mandatory parameters and one optional parameter. The function replaces all the occurrence of the given old parameter substring into new parameter substring. If the count parameter is provided, it only replaces the old substring that number of times from the start into new substring.