Introduction to Programming and Problem Solving

Week 8: Strings

Lecturer: Lemi Agrey Oliver

Department of Information Technology

Kumi University

Last week recap

- Last week we learned about functions in python which we defined as reusable block of codes that performs a specific task or a set of related tasks
- First and foremost, we categorically distinguished between the two major types of functions: custom or user-defined functions, which we create to suit our specific needs, and Python's built-in functions, a treasure trove of pre-existing tools at our disposal. Armed with this knowledge, we jumped into the art of creating and invoking functions, a foundational skill that empowers us to streamline and organize our code effectively.

Last week recap+

- To deepen our understanding, we unraveled the difference between parameters and arguments, crucial elements in function usage. We explored the various categories of arguments and honed our ability to harness their versatility to our advantage.
- We delved into the intriguing world of variables, distinguishing between local variables, whose scope is confined to within a function, and global variables, which transcend the confines of a single function and are visible throughout the entire application.

Last week recap++

- Our journey didn't stop there; we embraced the elegance of lambda functions and the power of recursion in Python, equipping ourselves with these advanced tools for more sophisticated problem-solving and elegant code design.
- Our exploration was comprehensive, touching upon a plethora of other captivating topics, forming a robust foundation for our continued journey into the world of Python programming.
- This week we shall concentrate on Strings in python

Content

- Introduction String
- The str class
- Basic Inbuilt Python Functions for String
- The index[] Operator
- Traversing String with for and while Loop
- Immutable Strings
- The String Operators
- String Operations

Introduction to Strings

- Characters are building blocks of Python. A program is composed of a sequence of characters. When a sequence of characters is grouped together, a meaningful string is created. [1]
- Therefore, a string in python is a sequence of characters enclosed in either single quotes (') or double quotes ("). Strings are one of the fundamental data types in Python and are used to represent textual data.
- String belongs to the compound datatypes.
- Compound datatypes are datatypes that comprises of smaller pieces.

The str class

- The **str** class (short for "string") is a built-in class that represents and manipulates text as sequences of characters.
- It is one of the most commonly used classes in Python because it allows you to work with textual data and perform various operations on strings.
- The **str** class provides a wide range of methods for string manipulation and formatting

String Creation:

- You can create a string in Python by enclosing a sequence of characters within single (' '), double (" "), or triple (''' ''' or """ """) quotes. For example:
- Single_quoted = 'This is a string. '
- double_quoted = "This is also a string."
- triple_quoted = '''This is a multi-line string.'''

Basic Inbuilt Python Functions for String

- Python provides a variety of built-in functions that can be used to manipulate and work with strings. Here are some of the basic inbuilt Python functions for string manipulation:
- len(): This function returns the length (the number of characters) of a string.
- text = "Hello, World!" length = len(text) # length is 13
- str(): This function converts a value to a string. It's often used to convert non-string values into strings for concatenation or printing.
- num = 42
- text = str(num) # text is now the string "42"

Basic Inbuilt Python Functions for String+

- upper() and lower(): These methods are used to convert a string to all uppercase or all lowercase characters, respectively.

```
1.   info = “Welcome to Handong University"
2.   uppercase_info = info.upper()
3.   # uppercase_info contains “WELCOME TO HANDONG UNIVERSITY"
4.   lowercase_info = info.lower()
5.   # lowercase_info contains “welcome to handong university"
```

Basic Inbuilt Python Functions for String+

- strip(): This method removes leading and trailing whitespace (spaces, tabs, newlines) from a string.

```
text = "  Some text with spaces  "
stripped_text = text.strip()
# stripped_text contains "Some text with spaces"
```

- split(): This method splits a string into a list of substrings based on a specified delimiter. By default, it splits on spaces.

```
text = "apple,banana,cherry"
items = text.split(",")
```

items is a list containing ['apple', 'banana', 'cherry']

Basic Inbuilt Python Functions for String++

- **join()**: This method is used to join a list of strings into a single string using a specified separator.

```
items = ['apple', 'banana', 'cherry’]
text = ",".join(items)
```

text is "apple,banana,cherry"

- **find() and index()**: These methods are used to find the index of a substring within a string.

- The difference is that **find()** returns -1 if the substring is not found, while **index()** raises an exception.

```
text = "Python is a powerful language."
position1 = text.find("is") # position1 is 7
position2 = text.index("not")
# raises a ValueError since "not" is not in the string
```

Basic Inbuilt Python Functions for String+++

replace(): This method replaces all occurrences of a substring with another substring in a given string.

```
text = "I like cats. Cats are cool."

modified_text = text.replace("cats", "dogs")

# modified_text is "I like dogs. Dogs are cool."
```

count(): This method returns the number of non-overlapping occurrences of a substring in a string.

text = "She sells seashells by the seashore." count = text.count("se") # count is 3

Basic Inbuilt Python Functions for String++++

startswith() and endswith():

These methods check if a string starts or ends with a specified substring and return a boolean value.

```
text = "Hello, World!"

starts_with_hello = text.startswith("Hello")

# True

ends_with_world = text.endswith("world")

# False
```

The index[] Operator

- The index operator **[]** in Python is used for indexing and slicing strings, lists, tuples, and other iterable objects.
- When applied to strings, it allows you to access individual characters within the string or extract substrings using slicing. Here's how the index operator works with strings
- **Accessing Individual Characters**:
- You can use the index operator to access individual characters in a string. In Python, string indexing starts at 0 for the first character. For example:

```
text = "Python"
first_char = text[0]
# first_char contains "P"
second_char = text[1]
# second_char contains "y"
```

The index[] Operator

- You can also use negative indexing to access characters from the end of the string. -1 represents the last character, -2 the second-to-last, and so on.
- text = "Python" last_char = text[-1] # last_char contains "n"
- second_last_char = text[-2] # second_last_char contains "o"

Immutable Strings

- strings are immutable. This means that once a string is created, its content cannot be changed.
- Any operation that appears to modify a string actually creates a new string with the desired changes.
- This immutability has some important implications for how strings work in Python:

Creating New Strings: When you perform an operation that seems to modify a string, you are, in fact, creating a new string object. The original string remains unchanged. For example:

```
1. original_string = "Hello"
2. # This creates a new string
3. modified_string = original_string + ", World!"
4. print(id(original_string)) #ID_OUTPUT: 2650522383664
5. print(id(modified_string)) #ID_OUTPUT: 2650517248112
```

The String Operators+

In Python, you can use various operators to perform operations on strings. These operators allow you to manipulate, compare, and concatenate strings. Here are some of the most commonly used string operators in Python:

Concatenation Operator (+):The + operator is used to concatenate (combine) two or more strings.

Example:

```
string1 = "Hello, “

 string2 = "World!"

result = string1 + string2

 # result contains "Hello, World!"
```

The String Operators++

Repetition Operator (*):The ***** operator is used to repeat a string a specified number of times.

Example: text = “LOVE YOU "

repeated_text = text * 4

repeated_text contains “LOVE YOU LOVE YOU LOVE YOU LOVE YOU "

Augmented Assignment Operators (+= and *=): Augmented assignment operators allow you to modify a string in place by adding or repeating content.

Example:

```
1.   text = "Python"
2.   text += " is great!"
3.   # text is now "Python is great!"
```

The String Operators++

- **identity Operators (is and is not)**:
- Identity operators are used to compare the memory addresses of two string objects.
- Example:

```
1.string1 = "Hello"
2.string2 = "Hello"
3.are_identical = string1 is string2
```

- # True (for small strings, Python optimizes and reuses memory)

String Operations

String operations in Python refer to various tasks and manipulations that you can perform on strings to process, modify, or analyze text data. Here are some common string operations in Python:

Concatenation:You can concatenate (combine) two or more strings using the **+** operator or string interpolation.

Example:

string1 = "Hello, " string2 = "World!"

result = string1 + string2

result contains "Hello, World!"

"

String Operations+

String Repetition: You can repeat a string a specified number of times using the ***** operator..

Example:

```
text = "Python "

repeated_text = text * 3

#repeated_text contains "Python Python Python
```

String Length:

You can find the length (number of characters) of a string using the **len()** function.

Example:

```
text = "Hello, World!"

length = len(text) # length is 13
```

String Operations++

String Slicing and Indexing:

String slicing and indexing are fundamental operations in Python for accessing and manipulating individual characters or substrings within a string. In Python, strings are treated as sequences of characters, and you can use indexing and slicing to work with them. Here's an in-depth discussion with examples:

String Indexing:

String indexing allows you to access individual characters within a string. In Python, string indexing is zero-based, meaning the first character is at index 0, the second at index 1, and so on.

```
1. text = "Python" first_char = text[0] # Access the first character 'P’
2. second_char = text[1] # Access the second character 'y'
```

String Operations++

- **String Slicing:**
- String slicing allows you to extract substrings from a string by specifying a range of indices. Slicing is performed using the colon : operator, with the format [start:stop], where start is inclusive and stop is exclusive. You can also specify a step value to control the interval between characters.

```
text = "Hello, World!"
sliced = text[7:12] # Extracts the substring "World"
```

- If you omit **start**, **stop**, or the step in the slice, Python assumes default values.
- If **start** is omitted, it defaults to the beginning of the string.
- If **stop** is omitted, it defaults to the end of the string.
- If the step is omitted, it defaults to 1.

String Operations+++

- **Slicing with Step:**
- You can control the interval between characters by specifying a step value. This allows you to skip characters when extracting substrings.
- text = "Python Programming"
- every_second_char = text[::2]
- # Extracts every second character "Pto rgamn"
- **You can also reverse a string by using a step of -1.**
- text = "Hello, World!"
- reversed_text = text[::-1]
- # Reverses the string "dlroW ,olleH"

String Formatting:

- String formatting allows you to create strings with placeholders for inserting variables, values, or expressions.
- There are several ways to format strings in Python, and I'll discuss three commonly used methods:
- f-strings, the **str.format()** method, and the **%** operator.
- **F-strings (Formatted String Literals):**
- F-strings are a convenient and readable way to format strings. You can create an f-string by prefixing a string literal with an 'f' or 'F'. Inside an f-string, you can include variables or expressions within curly braces **{}**. Python evaluates the expressions and substitutes them into the string.

String Formatting+

- name = "Alice" age = 30
- formatted_string = f"My name is {name} and I am {age} years old."
- print(formatted_string)
- **Output**:
- My name is Alice and I am 30 years old
- F-strings provide a concise and expressive way to format strings, making your code more readable and maintainable

String Formatting++

- **str.format() Method:**

- The **str.format()** method is an older, more versatile way to format strings. You create a string with placeholders (using curly braces **{}**), and then call the **format()** method on the string to fill in those placeholders.

```
name = "Bob" age = 25
formatted_string = "My name is {} and I am {} years old.".format(name, age)
print(formatted_string)
```

- **Output**:

- My name is Bob and I am 25 years old.

String Formatting++++

- **The % Operator (String Interpolation):**
- The **%** operator allows you to interpolate values into a string using format specifiers. This method is similar to the way C language handles string formatting. You specify placeholders using **%** followed by a format specifier and provide the values in a tuple.
- name = "Charlie" age = 28
- formatted_string = "My name is %s and I am %d years old." % (name, age)
- print(formatted_string)

Traversing String with for and while Loop

- **Using a for Loop**:

- A **for** loop is the most common and convenient way to traverse a string in Python. You can use the **for** loop to iterate through each character in the string:

```
text = "Hello, World!"
for char in text:
    print(char)
```

- In the above example, the **for** loop iterates through each character in the string and prints it to the console.

Traversing String with for and while Loop+

- **Using a while Loop**:

- You can also use a **while** loop to traverse a string. In this case, you'll need to keep track of the current position (index) in the string and increment it in each iteration until you reach the end of the string:

- text = "Hello, World!"

- index = 0

```
1. while index < len(text):
2.     print(text[index])
3.     index += 1
```

Traversing String with for and while Loop++

- In this example, we initialize an index variable to 0 and use a **while** loop to print characters at each position until we reach the end of the string.
- We increment the index in each iteration.
- Using a **for** loop is generally more Pythonic and easier for string traversal because it doesn't require explicit index management unlike while loop.

A program to print every third character

```
1. strs="Codingisthesweetestthingever"
2. for i in range(1, len(strs), 3):
3.     print(strs[i], end=" ")
```

```
Output: o n s e e e t n v
```

Explanation: The for loop traverse a string and print every third character, starting with the character at the 1st index position

Program to print words that are both in country_a, country_b and country_c

```
1. country_a="South Sudan"
2. country_b="South Africa"
3. country_c="South America"
4. for word in country_a:
5.     if word in country_b and word in country_c:
6.         print(word, end='')
```

- The provided code compares three strings, country_a, country_b, and country_c, to find common characters shared among them. Here's an explanation of the code:
- country_a, country_b, and country_c are three string variables, each containing a country name: "South Sudan," "South Africa," and "South America," respectively.

Program to print words that are both in country_a, country_b and country_c+

- The code uses a for loop to iterate through the characters in the string stored in country_a.
- For each character (represented by the variable word) in country_a, the code checks if the same character is present in both country_b and country_c using the if statement:
- if word in country_b and word in country_c:
- This condition checks whether the character (word) is found in both country_b and country_c.
- If a character is found in all three strings (i.e., it's common to all of them), it prints that character without a newline by using end=''. This means the characters are printed on the same line, separated by a space.
- So, for the given input values, "South Sudan," "South Africa," and "South America," the code would output the letters **South Sua** because those are the characters that are common to all three country names.

Summary of week 8

- As we conclude our Week 8 lecture, we have delved into the fascinating world of strings. In our exploration, we have uncovered the foundational concepts that form the bedrock of string manipulation in Python.
- Our journey began with an introduction to strings as sequences of characters. We learned how to create strings, those versatile containers of text that are omnipresent in the world of programming. Then, we delved deeper into the intricacies of the str class, understanding how it encapsulates these sequences and provides a multitude of methods for string manipulation.

Summary of week 8+

- We further expanded our horizons by uncovering the basic inbuilt Python functions for strings. These functions, such as len(), str(), and count(), opened new avenues for us to extract information, manipulate text, and gain insights from strings.
- The power of string indexing and slicing became evident as we harnessed these techniques to access and manipulate substrings within our strings, with the versatile for and while loops aiding us in traversing these textual terrains.
- Additionally, we discovered the concept of immutable strings, where each string is a fixed entity, and any operation results in a new string, preserving the integrity of the original.

Summary of week 8++

- We also explored the myriad operators and operations that enable us to concatenate, format, and modify strings, making them an invaluable tool in a programmer's arsenal.
- As we bid adieu to this week's journey through the world of strings, we have gained a profound understanding of their significance in Python and their pivotal role in text processing and data manipulation. We look forward to applying these newfound skills in our programming endeavors.

References

[1] Kamthane, A. N., & Kamthane , A. A. (2018). *PROGRAMMING AND PROBLEM SOLVING WITH PYTHON* (p. 165). McGraw Hill Education (India) Private Limited.

[2]Downey, A. B. (2015). *Think Python*. O'Reilly Media, Inc, 1005 Gravenstein highway North, Sebastopol, CA 95472.