



PYTHON

A Highly Expressive
Programming Language..

Computational Thinking with
Programming

@cse_bennett

@csebennett



Lecture Contents

- **String Processing:**
 - String
 - Basic String methods
 - String Traversal
 - Sequence Operation in String
 - Advanced String Processing

String Processing

- **String processing** refers to the operations performed on strings that allow them to be accessed, analyzed, and updated.
- We have already seen some operations on strings—for example, `str[k]`, for accessing individual characters, and `len(str)` for getting the length of a string.
- Python provides several other methods for string processing.

String

- String literals in python are surrounded by either single quotation marks, or double quotation marks. Ex. 'hello' is the same as "hello".
- You can display a string literal with the `print()` function:

Example:

```
print("Hello")  
print('Hello')
```

Output:

```
Hello  
Hello
```

- Assigning a string to a variable is done with the variable name followed by an equal sign and the string. For multi line string use three double quotes or three single quotes.

Single Word or single line string

```
a = "Hello"  
print(a)
```

Assigning Multi line string

```
a = """ Multi line string can be assigned  
using three double quotes, or  
three single quotes"""  
print(a)
```

Strings are Arrays

- Like many other popular programming languages, strings in Python are arrays of bytes representing Unicode characters.
- However, Python does not have a character data type, a single character is simply a string with a length of 1.
- Square brackets can be used to access elements of the string.

**Example: Get the character at position 1
(remember that the first character has the position 0):**

```
a = "Hello, World!"  
print(a[1])
```

Output: e

Example: String Slicing

```
b = "Hello, World!"  
print(b[2:5])
```

Output: llo

- **Slicing:** You can return a range of characters by using the slice syntax.
- Specify the **start** index and the **end** index (*excluded*), separated by a colon, to return a part of the string.

Strings are Arrays

- Use negative indexes to start the slice from the end of the string:

#Example

#Get the characters from position 5 to position 2 (not included), starting the count from the end of the string:

```
b = "Hello, World!"  
print(b[-5:-2])
```

Output:

orl

- **String Length:** To get the length of a string, use the `len()` function. This function returns the length of the string.

Example:

```
a = "Hello, World!"  
print(len(a))
```

Output:

13

String Traversal

- The characters in a string can be easily traversed, without the use of an explicit index variable, using the *for chr in string* form of the for statement.

Example1: Use of Index Variable

```
space = ' '  
num_spaces = 0  
  
line = input_file.readline()  
    for k in range(0, len(line)):  
        if line[k] == space:  
            num_spaces = num_spaces + 1
```

Example2: Without using Index Variable

```
for chr in line:  
    if chr == space:  
        num_spaces = num_spaces + 1
```

String-Applicable Sequence Operations

- Since strings (unlike lists) are ***immutable***, sequence-modifying operations are not applicable to strings.
- **For example:** one cannot add, delete, or replace characters of a string.
- All string operations that “**modify**” a string return a new string that is a modified version of the original string.

Sequences Operations Applicable to Strings			
Length	<code>len(str)</code>	Membership	<code>'h' in s</code>
Select	<code>s[index_val]</code>	Concatenation	<code>s + w</code>
Slice	<code>s[start:end]</code>	Minimum Value	<code>min(s)</code>
Count	<code>s.count(char)</code>	Maximum Value	<code>max(s)</code>
Index	<code>s.index(char)</code>	Comparison	<code>s == w</code>

Applicable Sequence Operations: Example

`s = 'Hello Goodbye!'`

```
>>> len(s)
14
```

```
>>> s[6]
'G'
```

```
>>> s[6:10]
'Good'
```

```
s.count('o')
3
```

```
>>> s.index('b')
10
```

```
>>> 'a' in s
False
```

```
>>> s + '!!!'
'Hello Goodbye!!!'
```

```
>>> min(s)
' '
```

```
>>> max(s)
'y'
```

Note: The find, replace, and strip methods in Python can be used to search and produce modified strings.

String Methods

Python provides a number of methods specific to strings, in addition to the general sequence operations.

- Checking the Contents of a String
- Searching and Modifying Strings:
 - Searching the contents of a String
 - Replacing the contents of a String
 - Removing the Contents of a String
- Splitting a String

Checking the Contents of a String

Checking the Contents of a String			
<code>str.isalpha()</code>	Returns True if <i>str</i> contains only letters.	<code>s = 'Hello'</code>	<code>s.isalpha()</code> → True
		<code>s = 'Hello!'</code>	<code>s.isalpha()</code> → False
<code>str.isdigit()</code>	Returns True if <i>str</i> contains only digits.	<code>s = '124'</code>	<code>s.isdigit()</code> → True
		<code>s = '124A'</code>	<code>s.isdigit()</code> → False
<code>str.islower()</code> <code>str.isupper()</code>	Returns True if <i>str</i> contains only lower (upper) case letters.	<code>s = 'hello'</code>	<code>s.islower()</code> → True
		<code>s = 'Hello'</code>	<code>s.isupper()</code> → False
<code>str.lower()</code> <code>str.upper()</code>	Return lower (upper) case version of <i>str</i> .	<code>s = 'Hello!'</code>	<code>s.lower()</code> → 'hello!'
		<code>s = 'hello!'</code>	<code>s.upper()</code> → 'HELLO!'

Searching, Modifying and Splitting Strings

Searching the Contents of a String			
<code>str.find(w)</code>	Returns the index of the first occurrence of <code>w</code> in <code>str</code> . Returns -1 if not found.	<code>s = 'Hello!'</code>	<code>s.find('l') → 2</code>
		<code>s = 'Goodbye'</code>	<code>s.find('l') → -1</code>
Replacing the Contents of a String			
<code>str.replace(w, t)</code>	Returns a copy of <code>str</code> with all occurrences of <code>w</code> replaced with <code>t</code> .	<code>s = 'Hello!'</code>	<code>s.replace('H', 'J') → 'Jello'</code>
		<code>s = 'Hello'</code>	<code>s.replace('ll', 'r') → 'Hero'</code>
Removing the Contents of a String			
<code>str.strip(w)</code>	Returns a copy of <code>str</code> with all leading and trailing characters that appear in <code>w</code> removed.	<code>s = ' Hello! '</code> <code>s = 'Hello\n'</code>	<code>s.strip(' !') → 'Hello'</code> <code>s.strip('\n') → 'Hello'</code>
Splitting a String			
<code>str.split(w)</code>	Returns a list containing all strings in <code>str</code> delimited by <code>w</code> .	<code>s = 'Lu, Chao'</code>	<code>s.split(',') → ['Lu', 'Chao']</code>

String Methods: Examples

- The **strip()** method: removes any whitespace from the beginning or the end:

Example:

```
a = " Hello, World! "  
print(a.strip())
```

Output:

```
Hello, World!
```

```
# Deleting a String using of del  
A = "hello"  
del a  
print(a)
```

```
NameError: name 'txt' is not defined
```

- The **lower()** and **upper()** methods returns the string in lower and upper case respectively:

Example:

```
a = "Hello, World!"  
print(a.lower())  
print(a.upper())
```

Output:

```
hello, world!  
HELLO, WORLD!
```

String Methods: Examples

- The `replace()` method replaces a string with another string:

Example:

```
a = "Hello, World!"  
print(a.replace("H", "J"))
```

Output:

```
Jello, World!
```

- The `split()` method splits the string into substrings if it finds instances of the separator:

Example:

```
a = "Hello, World!"  
print(a.split(","))
```

Output:

```
['Hello', ' World!']
```

Other Check String: Examples

- To check if a certain phrase or character is present in a string, we can use the keywords **in** or **not in**.

Example1:

#Check if the phrase "ain" is present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" in txt  
print(x)
```

Output:
True

Example2:

#Check if the phrase "ain" is NOT present in the following text:

```
txt = "The rain in Spain stays mainly in the plain"  
x = "ain" not in txt  
print(x)
```

Output:
False

String Concatenation

- To concatenate, or combine, two strings you can use the + operator.

Example 1:

#Merge variable a with variable b into variable c:

```
a = "Hello"  
b = "World"  
c = a + b  
print(c)
```

Output:

HelloWorld

Example 2:

#To add a space between them, add a " ":

```
a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)
```

Output:

Hello World

String Format

- We cannot combine strings and numbers.

Example:

```
age = 36
txt = "My name is John, I am " + age
print(txt)
```

Output:

```
Traceback (most recent call last):
  File "demo_string_format_error.py", line 2, in <module>
    txt = "My name is John, I am " + age
TypeError: must be str, not int
```

- Python provides `format()` method to combine strings and numbers.
- The `format()` method takes the passed arguments, formats them, and places them in the string where the placeholders `{}` are:

Example:

#Use the format() method to insert numbers into strings:

```
age = 36
txt = "My name is John, and I am {}"
print(txt.format(age))
```

Output:

```
My name is John, and I am 36
```

String Format: Named Indexes

- You can also use named indexes by entering a name inside the curly brackets `{carname}`.
- In this case, you must use names when you pass the parameter values `txt.format(carname = "Ford")`:

Example:

```
myorder = "I have a {carname}, it is a {model}."  
print(myorder.format(carname = "Ford", model = "Mustang"))
```

Output:

```
I have a Ford, it is a Mustang.
```

String Format: Example

- The format() method takes unlimited number of arguments, and are placed into the respective placeholders:

Example:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want {} pieces of item {} for {} dollars."
print(myorder.format(quantity, itemno, price))
```

Output:

```
I want 3 pieces of item 567 for
49.95 dollars.
```

- You can use index numbers {0} to be sure the arguments are placed in the correct placeholders:

Example:

```
quantity = 3
itemno = 567
price = 49.95
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."
print(myorder.format(quantity, itemno, price))
```

Output:

```
I want to pay 49.95 dollars for
3 pieces of item 567
```

Escape Character

- To insert characters that are illegal in a string, use an escape character.
- An escape character is a backslash \ followed by the character you want to insert.
- An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example:

#You will get an error if you use double quotes inside a string that is surrounded by double quotes:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Output:

[illegible]

Escape Character: Example

- To fix this problem, use the escape character `\`:
- The escape character allows you to use double quotes when you normally would not be allowed:

Example:

```
txt = "We are the so-called \"Vikings\" from the north."
```

Output:

```
We are the so-called "Vikings" from the north.
```

Other Escape Characters used in Python

Code	Result	Example	Output
\'	Single Quote	<code>print('It\'s alright.')</code>	It's alright.
\\	Backslash	<code>print("This will insert one \\ (backslash).")</code>	This will insert one \ (backslash).
\n	New Line	<code>print("Hello\nWorld!")</code>	Hello World!
\r	Carriage Return	<code>print("Hello\rWorld!")</code>	Hello World!
\t	Tab	<code>print("Hello\tWorld!")</code>	Hello World!
\b	Backspace	<code>print("Hello \bWorld!")</code>	HelloWorld!
\ooo	Octal value	#A backslash followed by three integers will result in a octal value: <code>print("\110\145\154\154\157")</code>	Hello
\xhh	Hex value	#A backslash followed by an 'x' and a hex number represents a hex value: <code>print("\x48\x65\x6c\x6c\x6f")</code>	Hello

MCQs

1. Some string methods alter the string they are called on, while others return a new altered version of the string.
 - a) TRUE
 - b) FALSE
2. The find method returns the number of occurrences of a character or substring within a given string.
 - a) TRUE
 - b) FALSE
3. Which of the results below does `s[2:4]` return for the string `s = 'abcdef'`.
 - a) 'cd' b) 'bcd' c) 'bc' d) 'cde'
4. Indicate which of the following is true.
 - a) String method `isdigit` returns true if the string applied to contains any digits.
 - b) String method `isdigit` returns true if the string applied to contains only digits.
5. Indicate which of the following `s.replace('c','e')` returns for `s = 'abcabc'`.
 - a) 'abeabc' b) 'abeabe'
4. Which of the results below does `s.strip('-')` return for the string `s = '---ERROR---`'.
 - a) '---ERROR' b) 'ERROR---' c) 'ERROR'

MCQs: Answers

1. Some string methods alter the string they are called on, while others return a new altered version of the string.
 - a) TRUE
 - b) FALSE**
2. The find method returns the number of occurrences of a character or substring within a given string.
 - a) TRUE
 - b) FALSE**
3. Which of the results below does `s[2:4]` return for the string `s = 'abcdef'`.
 - a) 'cd'**
 - b) 'bcd'
 - c) 'bc'
 - d) 'cde'
4. Indicate which of the following is true.
 - a) String method *isdigit* returns true if the string applied to contains any digits.
 - b) String method *isdigit* returns true if the string applied to contains only digits.**
5. Indicate which of the following `s.replace('c','e')` returns for `s = 'abcabc'`.
 - a) 'abeabc'
 - b) 'abeabe'**
4. Which of the results below does `s.strip('-')` return for the string `s = '---ERROR---`.
 - a) '---ERROR'
 - b) 'ERROR---
 - c) 'ERROR'**

Thank You

?