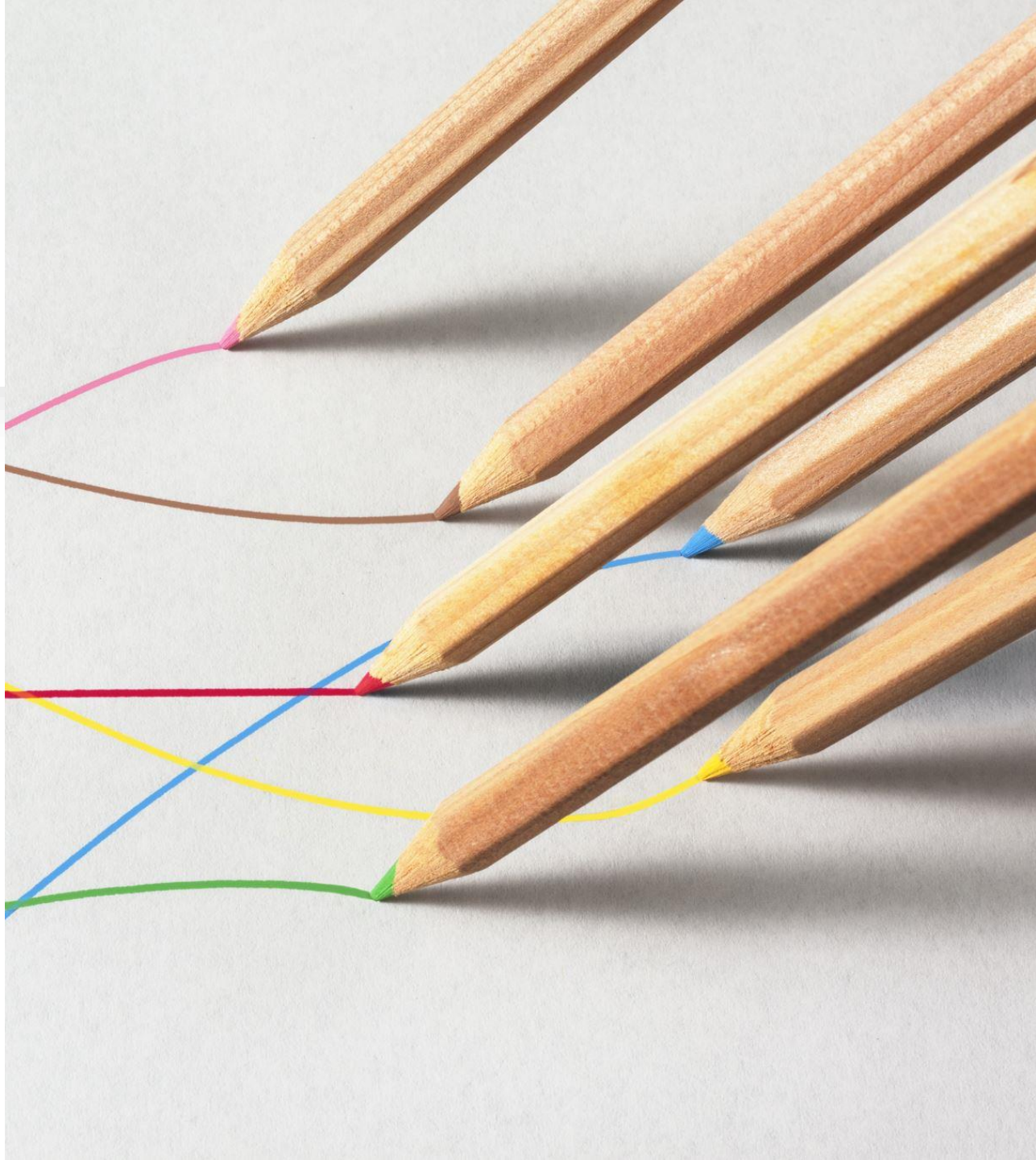

String

Outline

- 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	`len(str)`	Membership	`'h' in s`
Select	`s[index_val]`	Concatenation	`s + w`
Slice	`s[start:end]`	Minimum Value	`min(s)`
Count	`s.count(char)`	Maximum Value	`max(s)`
Index	`s.index(char)`	Comparison	`s == w`

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 several 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			
`str.isalpha()`	Returns True if *str* contains only letters.	`s = 'Hello'`	`s.isalpha()` → `True`
		`s = 'Hello!'`	`s.isalpha()` → `False`
`str.isdigit()`	Returns True if *str* contains only digits.	`s = '124'`	`s.isdigit()` → `True`
		`s = '124A'`	`s.isdigit()` → `False`
`str.islower()` `str.isupper()`	Returns True if *str* contains only lower (upper) case letters.	`s = 'hello'`	`s.islower()` → `True`
		`s = 'Hello'`	`s.isupper()` → `False`
`str.lower()` `str.upper()`	Return lower (upper) case version of *str*.	`s = 'Hello!'`	`s.lower()` → `'hello!'`
		`s = 'hello!'`	`s.upper()` → `'HELLO!'`

Searching, Splitting and Modifying Strings

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

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!']
```

String Methods: 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 Concatanation

- 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 Index

- 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:

```
  File "demo_string_escape_error.py", line 1
    txt = "We are the so-called "Vikings" from the north."
                                ^
SyntaxError: invalid syntax
```

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 Escae Characters used in Python

Code	Result	Example	Output
\'	Single Quote	print('It\'s alright.')	It's alright.
\\	Backslash	print("This will insert one \\ (backslash).“)	This will insert one \ (backslash).
\n	New Line	print("Hello\nWorld!“)	Hello World!
\r	Carriage Return	print("Hello\rWorld!“)	Hello World!
\t	Tab	print("Hello\tWorld!“)	Hello World!
\b	Backspace	print("Hello \bWorld!“)	HelloWorld!

\ooo	Octal value	#A backslash followed by three integers will result in a octal value:
		print("\110\145\154\154\157“)
			Hello

\xhh	Hex value	#A backslash followed by an 'x' and a hex number represents a hex value:
		print("\x48\x65\x6c\x6c\x6f“)
			Hello

MCQs

Some string methods alter the string they are called on, while others return a new altered version of the string.

a) TRUE

b) FALSE

The find method returns the number of occurrences of a character or substring within a given string.

a) TRUE

b) FALSE

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

Some string methods alter the string they are called on, while others return a new altered version of the string.

a) TRUE

b) FALSE

The find method returns the number of occurrences of a character or substring within a given string.

a) TRUE

b) FALSE

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'**