Week8 - Arrays and Strings

Python Arrays

- An array is defined as a collection of items that are stored at contiguous memory locations.
- It is a container which can hold multiple number of data items and these items should be of the same type.
- An array is popular in most programming languages like C/C++, JavaScript, etc.
- Array is an idea of storing multiple items of the same type together and it makes easier to calculate the position of each element by simply adding an offset to the base value.
- The Array can be created in Python by importing the **array** module to the python program.
- Syntax for creating arrays in python

```
import arrays as arr
arrayName = arr.array(typecode, [initializers])
```

where, typecode-> code that specifies type of data

Initializers-> list of values to be stored in array

typecode	Python Type
B,B,i,I,l,L,h,H	int
f	float
d	double

- Example
  import array as arr
  a = arr.array('d', [1.1, 3.5, 4.5])
  print(a)

Accessing array elements:

We can access the array elements using the respective indices of those elements.

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])
print("First element:", a[0])
print("Second element:", a[1])
print("Last element:", a[-1])
```

Output

First element: 2

Second element: 4

Last element: 8

Slicing Python Arrays

We can access a range of items in an array by using the slicing operator :.

Example

```
import array as arr
numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
```

```
numbers_array = arr.array('i', numbers_list)
print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:])  # 6th to end
print(numbers_array[:])   # beginning to end
```

Output:

```
 array('i', [62, 5, 42])
array('i', [2, 5, 62])
array('i', [52, 48, 5])
array('i', [2, 5, 62, 5, 42, 52, 48, 5])
```

Built in functions and methods on arrays:

All the built-in functions and methods which can be applied on list can also be applied on arrays (Refer the Week6 material)

Example: Program to illustrate the use of all the built-in functions and methods on arrays

```
import array as ar
numbers = arr.array('i', [1, 2, 3, 5, 7, 10])
# changing first element
numbers[0] = 0
print(numbers)                    # Output: array('i', [0, 2, 3, 5, 7, 10])
# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)                    # Output: array('i', [0, 2, 4, 6, 8, 10])
```

```
# append() function single data item to the end of array
numbers.append(40)
print(numbers)                    # Output: array('i', [0, 2, 4, 6, 8, 10,40])
# extend() appends iterable to the end of the array
numbers.extend([5, 6, 7])
print(numbers)            # Output: array('i', [0, 2, 4, 6, 8, 10,40,5,6,7])
del number[2]             # removing third element
print(number)             # Output: array('i', [0, 2,6, 8, 10,40,5,6,7])

del number                # deleting entire array
print(number)              # NameError: array number is not defined
#removing data items of array using remove() and pop() method
numbers.remove(40)
print(numbers)   # Output: array('i', [0, 2,6, 8, 10,5,6,7])
print(numbers.pop(2))   # Output: 6
print(numbers)   # Output: array('i', [0, 2,6, 10,5,6,7])
```

Finding the length of an array

The length of an array is defined as the number of elements present in an array. The len() returns an integer value that is equal to the total number of the elements present in that array.

Syntax

len(array_name)

Array Concatenation

We can easily concatenate any two arrays using the + symbol.

Example

```
a=arr.array('d',[1.1 , 2.1 ,3.1,2.6,7.8])
b=arr.array('d',[3.7,8.6])
c=arr.array('d')      # creates an empty array
c=a+b
print("Array c = ",c)
```

Output:
Array c= array('d', [1.1, 2.1, 3.1, 2.6, 7.8, 3.7, 8.6])

Strings:

The usage of Strings is found in almost all types of applications. A string consists of a sequence of characters, which includes letters, numbers, punctuation marks and spaces. To represent strings, you can use a single quote, double quotes or triple quotes.

Creating and Storing Strings

Strings are another basic data type available in Python. They consist of one or more char- acters surrounded by matching quotation marks.

For example,

```
single_quote = 'This is a single message'
double_quote = "Hey it is my book"
single_char_string = "A"
empty_string = ""
empty_string = ''
```

```
single_within_double_quote = "Opportunities don't happen. You create them."
double_within_single_quote = "Why did she call the man 'smart'?"
same_quotes = 'I\'ve an idea'
triple_quote_string = '''This
… is
… triple
… quote'''
```

The str() Function

The str() function returns a string. The syntax for str() function is,

str(object)

It returns a string version of the object. If the object is not provided, then it returns an empty string.

Example:

```
print(str(10))                    # Output: '10'
create_string = str()             # Creates an empty string
print(type(create_string))        #<class 'str'>
```

Basic String Operations:

- **Concatenation (+):** strings can also be concatenated using + sign

Example:

1)

```
string_1 = "face"
string_2 = "book"
```

```
concatenated_string = string_1 + string_2
print(concatenated_string)                # Output: 'facebook'
```

2)
```
concatenated_string_with_space = "Hi " + "There"
print(concatenated_string_with_space)     # Output: 'Hi There'
```

3)
```
string = 50 + " percent"
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

- Concatenation cannot be applied between two different type. Therefore the above error occurs. To Continue with the concatenation convert int data type into string using str()

```
string=str(50)+" percent"          # Output: 50 percent
```

- **Repetition (*):** * operator is used to get the repeated sequence
  Example:
  ```
  repetition_of_string = "wow"
  print(repetition_of_string *5)    #Output: 'wowwowwowwowwow'
  ```
-
- **Membership operators:** We can check the existence of particular substring within the string using in and not in operators
  Example:
  ```
  fruit_string = "apple is a fruit"
  fruit_sub_string = "apple"
  print(fruit_sub_string in fruit_string)          #Output: True
  another_fruit_string = "orange"
  print(another_fruit_string not in fruit_string)  #Output: True
  ```

- **String Comparison**

  You can use (>, <, <=, >=, ==, !=) to compare two strings resulting in either Boolean True or False value. Python compares strings using ASCII value of the characters.

  For example,

```
print("january" == "jane")        # Output: False
print("january" != "jane" )       #True
print("january" < "jane")         #False
print("january" > "jane" )        #True
print("january" <= "jane" )       #False
print("january" >= "jane")        #True
```

Built-In Functions Used on Strings

There are many built-in functions for which a string can be passed as an argument

len()-> The len() function calculates the number of characters in a string. The white space characters are also counted.

max()-> The max() function returns a character having highest ASCII value.

min()-> The min() function returns character having lowest ASCII value.

For example,

```
count_characters = len("gpt athani")
print(count_characters)   # output:10
print(max("axis"))        # output:'x'
print(min("bad") )        # output:'a'
```

Accessing Characters in String by Index Number

- Each character in the string occupies a position in the string.
- Each of the string's character corresponds to an index number. The first character is at index 0; the next character is at index 1, and so on.
- The length of a string is the number of characters in it. You can access each character in a string using a subscript operator i.e., a square bracket. Square brackets are used to perform indexing in a string to get the value at a specific index or position.
- Syntax

```
string_name[index]
```

  - ☞ where index is usually in the range of 0 to one less than the length of the string.
  - ☞ The value of index should always be an integer(Positive or negative)

String Slicing:

The "slicing is used to extract sub-parts of sequence of characters within an original string.

The syntax for string slicing is,

```
string_name[start:end[:step]
```

For Example:

```
healthy_drink = "green tea"
print(healthy_drink[0:3])          # 'gre'
print(healthy_drink[:5])           #'green'
print(healthy_drink[6:])           #'tea'
print(healthy_drink[:])            #'green tea'
print(healthy_drink[4:4])          #"n"
print(healthy_drink[6:20])         #'tea'
print(healthy_drink[-3:-1])        #'te'
print(healthy_drink[6:-1])         #'te'
```

Specifying Steps in Slice Operation

- In the slice operation, a third argument called step which is an optional can be specified along with the start and end index numbers.
- This step refers to the number of characters that can be skipped after the start indexing character in the string.
- The default value of step is one.
- For example,

```
newspaper = "new york times"
print(newspaper[0:12:4])        # 'ny'
print(newspaper[::4])           # 'ny e'
```

** Write Python Code to Determine Whether the Given String Is a Palindrome or Not, Using Slicing**

```
user_string = input("Enter string: ")
if user_string == user_string[::-1]:
    print(f"User entered string is palindrome")
else:
    print(f"User entered string is not a palindrome")
```

OUTPUT

Case 1:

Enter string: madam

User entered string is palindrome

Case 2:

Enter string: cat

User entered string is not a palindrome

Joining Strings Using join() Method

- Strings can be joined with the join() string.
- The join() method provides a flexible way to concatenate strings.
- **The syntax of join() method is,**

```
string_name.join(sequence)
```

- Here sequence can be string or list.
- If the sequence is a string, then join() function inserts string_name between each character of the string sequence and returns the concatenated string.
- If the sequence is a list, then join() function inserts string_name between each item of list sequence and returns the concatenated string. It should be noted that all the items in the list should be of string type.

For Example,

```
date_of_birth = ["15", "05", "1987"]
print(":".join(date_of_birth))        #Output:  '15:05:1987'
social_app = ["instagram", "is", "an", "photo", "sharing", "application"]
print(" ".join(social_app))
#'Output: instagram is an photo sharing application'
numbers = "179"
characters = "gpt"
print(numbers.join(characters))        # Output :'g179p179t'
```

Split Strings Using split() Method

- The split() method returns a list of string items by breaking up the string using the delimiter string.
- **The syntax of split() method is,**

```
string_name.split([separator [, maxsplit]])
```

  - ☞ Here separator is the delimiter string and is optional.
  - ☞ A given string is split into list of strings based on the specified separator.
  - ☞ If the separator is not specified then whitespace is considered as the delimiter string to separate the strings.
  - ☞ If maxsplit is given, at most max-split splits are done
  - ☞ If maxsplit is not specified or −1, then there is no limit on the number of splits.

For Example:

```
inventors = "edison, tesla, marconi, newton"
print(inventors.split(","))#O/p: ['edison', ' tesla', ' marconi', ' newton']
watches = "rolex hublot titan sonata"
print(watches.split() # O/p:['rolex', 'hublot', 'titan', 'sonata']
```

Strings Are Immutable

As strings are immutable, it cannot be modified. The characters in a string cannot be changed once a string value is assigned to string variable. However, you can assign different string values to the same string variable.

For Example,

```
immutable = "dollar"
immutable[0] = "c"
```

```
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

```
string_immutable = "c" + immutable[1:]
print(string_immutable)                    #output: 'collar'
immutable = "rollar"
print(immutable)                           #output: 'rollar'
```

String Traversing

Since the string is a sequence of characters, each of these characters can be traversed using the for loop.

Program to Demonstrate String Traversing Using the for Loop

```
alphabet = "google"
index = 0
print(f"In the string '{alphabet}'")
for each_character in alphabet:
    print(f"Character '{each_character}' has an index value of {index}")
    index += 1
```

OUTPUT

```
In the string 'google'
Character 'g' has an index value of 0
Character 'o' has an index value of 1
Character 'o' has an index value of 2
Character 'g' has an index value of 3
Character 'l' has an index value of 4
Character 'e' has an index value of 5
```

Program to Print the Characters Which Are Common in Two Strings

```
string_1=”rose”
string_2=”goose”
for letter in string_1:
    if letter in string_2:
        print(f"Character '{letter}' is found in both the strings")
```

Output:

Character 'o' is found in both the strings

Character 's' is found in both the strings

Character 'e' is found in both the strings

** Write Python Program to Count the Total Number of Vowels, Consonants and Blanks in a String**

```
user_string = input("Enter a string: ")
vowels = 0
consonants = 0
blanks = 0
for each_character in user_string:
  if  each_character in "aeiouAEIOU":
    vowels += 1
  elif each_character==' ':
```

```
        blanks += 1
    else:
        consonants += 1
print(f"Total number of Vowels in user entered string is {vowels}")
print(f"Total number of Consonants in user entered string is {consonants}")
print(f"Total number of Blanks in user entered string is {blanks}")
```

Output:

Enter a string: Gpt Athani

Total number of Vowels in user entered string is 3

Total number of Consonants in user entered string is 6

Total number of Blanks in user entered string is 1

** Write Python Program to Calculate the Length of a String Without Using Built-In len() Function**

```
user_string = input("Enter a string: ")
count_character = 0
for each_character in user_string:
    count_character += 1
print(f"The length of user entered string is {count_character} ")
```

Output:

Enter a string: 179 Government Polytechnic Athani

The length of user entered string is 33

String Methods:

String Methods	Syntax	Description
capitalize()	string_name.capitalize()	The *capitalize()* method returns a copy of the string with its first character capitalized and the rest lowercased.
casefold()	string_name.casefold()	The *casefold()* method returns a casefolded copy of the string. Casefolded strings may be used for caseless matching.
center()	string_name.center(width[, fillchar])	The method *center()* makes string_name centered by taking width parameter into account. Padding is specified by parameter fillchar. Default filler is a space.
count()	string_name.count(substring [, start [, end]])	The method *count()*, returns the number of non-overlapping occurrences of substring in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.
endswith()	string_name.endswith(suffix[, start[, end]])	This method *endswith()*, returns *True* if the string_name ends with the specified suffix substring, otherwise returns *False*. With optional start, test beginning at that position. With optional end, stop comparing at that position.
find()	string_name. find(substring[, start[, end]])	Checks if substring appears in string_name or if substring appears in string_name specified by starting index *start* and ending index *end*. Return position of the first character of the first instance of string substring in string_name, otherwise return –1 if substring not found in string_name.
isalnum()	string_name.isalnum()	The method *isalnum()* returns Boolean *True* if all characters in the string are alphanumeric and there is at least one character, else it returns Boolean *False*.
isalpha()	string_name.isalpha()	The method *isalpha()*, returns Boolean *True* if all characters in the string are alphabetic and there is at least one character, else it returns Boolean *False*.
isdecimal()	string_name.isdecimal()	The method *isdecimal()*, returns Boolean *True* if all characters in the string are decimal characters and there is at least one character, else it returns Boolean *False*.

String Methods	Syntax	Description
istitle()	string_name.istitle()	The method *istitle()* returns Boolean *True* if the string is a title cased string and there is at least one character, else it returns Boolean *False*.
isupper()	string_name.isupper()	The method *isupper()* returns Boolean *True* if all cased characters in the string are uppercase and there is at least one cased character, else it returns Boolean *False*.
upper()	string_name.upper()	The method *upper()* converts lowercase letters in string to uppercase.
lower()	string_name.lower()	The method *lower()* converts uppercase letters in string to lowercase.
ljust()	string_name.ljust(width[, fillchar])	In the method *ljust()*, when you provide the string to the method *ljust()*, it returns the string left justified. Total length of string is defined in first parameter of method width. Padding is done as defined in second parameter fillchar. (default is space).
rjust()	string_name.rjust(width[, fillchar])	In the method *rjust()*, when you provide the string to the method *rjust()*, it returns the string right justified. The total length of string is defined in the first parameter of the method, width. Padding is done as defined in second parameter fillchar. (default is space).
title()	string_name.title()	The method *title()* returns "titlecased" versions of string, that is, all words begin with uppercase characters and the rest are lowercase.
swapcase()	string_name.swapcase()	The method *swapcase()* returns a copy of the string with uppercase characters converted to lowercase and vice versa.
splitlines()	string_name. splitlines([keepends])	The method *splitlines()* returns a list of the lines in the string, breaking at line boundaries. Line breaks are not included in the resulting list unless keepends is given and true.

startswith()	string_name.startswith(prefix[, start[, end]])	The method *startswith()* returns Boolean *True* if the string starts with the prefix, otherwise return *False*. With optional start, test string_name beginning at that position. With optional end, stop comparing string_name at that position.
strip()	string_name.strip([chars])	The method *lstrip()* returns a copy of the string_name in which specified *chars* have been stripped from both side of the string. If char is not specified then space is taken as default.
rstrip()	string_name.rstrip([chars])	The method *rstrip()* removes all trailing whitespace of string_name.
lstrip()	string_name.lstrip([chars])	The method *lstrip()* removes all leading whitespace in string_name.
replace()	string_name. replace(old, new[, max])	The method *replace()* replaces all occurrences of old in string_name with new. If the optional argument max is given, then only the first max occurrences are replaced.
zfill()	string_name.zfill(width)	The method *zfill()* pads the string_name on the left with zeros to fill width.
isdigit()	string_name.isdigit()	The method *isdigit()* returns Boolean *True* if all characters in the string are digits and there is at least one character, else it returns Boolean *False*.
isidentifier()	string_name.isidentifier()	The method *isidentifier()* returns Boolean *True* if the string is a valid identifier, else it returns Boolean *False*.
islower()	string_name.islower()	The method *islower()* returns Boolean *True* if all characters in the string are lowercase, else it returns Boolean *False*.
isspace()	string_name.isspace()	The method *isspace()* returns Boolean *True* if there are only whitespace characters in the string and there is at least one character, else it returns Boolean *False*.
isnumeric()	string_name.isnumeric()	The method *isnumeric()*, returns Boolean *True* if all characters in the string_name are numeric characters, and there is at least one character, else it returns Boolean *False*. Numeric characters include digit characters and all characters that have the Unicode numeric value property.

** Write Python Program That Accepts a Sentence and Calculate the Number of Words, Digits, Uppercase Letters and Lowercase Letters**

```
user_string=input("Enter the string")
word_count = 0
digit_count = 0
upper_case_count = 0
lower_case_count = 0
for each_char in user_string:
    if each_char.isdigit():
        digit_count += 1
    elif each_char.isspace():
        word_count += 1
    elif each_char.isupper():
        upper_case_count += 1
    elif each_char.islower():
        lower_case_count += 1
    else:
        pass
print(f"Number of digits in sentence is {digit_count}")
print(f"Number of words in sentence is {word_count + 1}")
print(f"Number of upper case letters in sentence is {upper_case_count}")
print(f"Number of lower case letters in sentence is {lower_case_count}")
```

Output:

Enter the string: Python is an object oriented language which has 4 data collection types

Number of digits in sentence is 1

Number of words in sentence is 12

Number of upper case letters in sentence is 1

Number of lower case letters in sentence is 58

Write Python Program to Convert Uppercase Letters to Lowercase and Vice Versa without using built-in methods

```
user_string=input("Enter string")
convert_case=str()
for char in user_string:
    if char.islower():
        convert_case+=char.upper()
    else:
        convert_case+=char.lower()
print(f"{convert_case}")
```

Output:

Enter stringGpt Athani

gPT aTHANI

**Write Python Program to Replace Comma-Separated Words with Hyphens and Print Hyphen-Separated Words in Ascending Order **

```
comma_separated_words = input("Enter comma separated words ")
split_words = comma_separated_words.split(",")
split_words.sort()
hyphen_separated_words = "-".join(split_words)
print(f"Hyphen separated words in ascending order are '{hyphen_separated_words}'")
```

Output:

Enter comma separated words python,program

Hyphen separated words in ascending order are 'program-python'

**Write Python Program to Count the Occurrence of User-Entered Words in a Sentence **

```
input_string = input("Enter a string ")
user_word = input("Enter a word to count its occurrence ")
word_count = 0
for each_word in input_string.split():
    if each_word == user_word:
        word_count += 1
print(f"The word '{user_word}' has occurred {word_count} times")
```

Output:

Enter a string This is a book a book has no of pages and a book is always prefixed with preface

Enter a word to count its occurrence book

The word 'book' has occurred 3 times