

PYTHON

Lecture – 07



Recap

- Control Statements (If-elif-else)
- Loop Statements (while & for)

Contents

- **String**
- **Collection/ Sequence Data Types:** *Built-in data types in Python used to store collections of data*
 - List
 - Tuple
 - Set
 - Dictionary
- **Array**

All Types at a glance

- **List**

- A list is a mutable, ordered collection of elements.
- Defined using square brackets `[]`
- Example: `my_list=[10, 20, 30, 23]`

- **Tuple**

- A tuple is an immutable, ordered collection of elements.
- Defined using parentheses `()`
- Example: `my_tuple = (1, 2, 3, 4)`

- **SET**

- A set is a **mutable, unordered** collection of **unique** elements.
- Defined using curly braces `{}` or the `set()` function
- Example: `my_set={10, 20, 30, 23}`

All Types at a glance

- **Dictionary**

- A dictionary is a ***mutable, unordered*** collection of key-value pairs.
- Defined using curly braces `{}` with key-value pairs separated by a colon : (e.g., `{key: value}`).
- Example: `my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}`

- **Array**

- An array is a mutable, ordered collection of elements, where all elements are of the same data type (unlike a list, which can contain different types).
- Need to import the array module, and arrays are created using `array.array()`
- Example:

```
import array
my_array = array.array('i', [1, 2, 3, 4])
```

String

- A **string** in Python is a *sequence of characters* enclosed in quotes.
- Strings are used to represent *text data* and can include **letters**, **digits**, **symbols**, and even **whitespace**.
- Strings can be created using *single quotes (')*, *double quotes (")*, or *triple quotes (''' or """)* for multi-line strings.

Example:

```
txt='hello' or txt ="hello" #with variable
print(txt)
print("hello") #directly with print function
```

Multiline String:

```
a = ''' This is a multiline
        string in python with single quotes '''
b = """ This is a multiline
        string in python with double quotes """
print(a)
print(b)
```

String (Cont...)

- 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.
- The **len()** function returns the length of a string:

```
a = "Hello, Python!"  
print(a[1])  
print(len(a)) #length function len()
```

Looping through a string:

```
for x in "banana":  
    print(x)
```

String Operators

- **Concatenation:** Strings can be concatenated using the **+** operator.

```
txt1 = "Hello"  
txt2 = "Python"  
full_text = txt1 + " " + txt2 # Output: "Hello World"
```

- **Repetition:** Strings can be repeated using the ***** operator.

```
text = "Hi! "  
repeated_text = text * 3 # Output: "Hi! Hi! Hi! "
```


String Slicing

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

```
text = "Hello, Python"
```

```
substring1 = text[0:5]      # Output: 'Hello' // from index 0 to 5
```

```
substring2 = text[:5]       # Output: 'Hello' // from start to 5
```

```
substring3 = text [2:5]     # Output: 'llo'    // from 2 to 5 (excluded)
```

```
substring4 = text[7:]       # Output: 'Python' // from 7 to end
```

```
substring5 = text[-7:-1]    # Output: 'Python' // from end 1 to 6
```

(excluded)

String Functions

Method	Description	Example: <code>a = 'hello'</code>
<code>lower()</code>	Converts the string to lowercase.	<code>a.lower()</code> → <code>'hello'</code>
<code>upper()</code>	Converts the string to uppercase.	<code>a.upper()</code> → <code>'HELLO'</code>
<code>capitalize()</code>	Capitalizes the first character.	<code>a.capitalize()</code> → <code>'Hello'</code>
<code>title()</code>	Capitalizes the first letter of every word.	<code>a.title()</code> → <code>'Hello'</code>
<code>strip()</code>	Removes leading/trailing whitespace.	<code>' Hello '.strip()</code> → <code>'hello'</code>
<code>replace(old, new)</code>	Replaces occurrences of a substring.	<code>a.replace('h', 'A')</code> → <code>'Aello'</code>
<code>split(delimiter)</code>	Splits the string into a list.	<code>'a,b,c'.split(',')</code> → <code>['a', 'b', 'c']</code>
<code>join(list)</code>	Joins a list of strings into one string.	<code>','.join(['a', 'b', 'c'])</code> → <code>'a, b, c'</code>
<code>count(substring)</code>	Counts the occurrences of a substring.	<code>a.count('l')</code> → <code>2</code>

Examples

```
a = "Hello bangladesh"
print(a.lower())
print(a.upper())
print(a.capitalize())
print(a.title())
print(a.count("l"))
print(a.replace("Hello", "My"))
print(a.split()) #default is whitespace
```

```
b = " Hello bangladesh "
print(b.strip())
```

```
c=['My', 'Bangladesh']
print(" ".join(c))
```

```
hello bangladesh
HELLO BANGLADESH
Hello bangladesh
Hello Bangladesh
3
My bangladesh
['Hello', 'bangladesh']
Hello bangladesh
My Bangladesh
```

Exercises on String

- 8.1 Write a python code that take your department name as user input and print this department in upper case, lower case, capitalize and sentence case.
- 8.2 Write a code that find the length of a string and print each character using loop.
- 8.3 Write a Python program that takes two strings as input and concatenates them with a space in between. Print the result.

List

- A **list** in Python is a versatile, ordered collection of items (elements) that allows storage of multiple data types.
- **Ordered**: The elements have a **defined order**, meaning that when you create a list, its items are stored in the sequence you define.
- **Mutable**: Lists can be modified after creation, allowing for operations like adding, removing, or updating items.
- **Duplicates**: A list can contain multiple instances of the same item.
- **Heterogeneous**: Lists can store items of different data types (e.g., strings, numbers, other lists).
- Lists are defined by enclosing the elements in **square brackets []**.
- Store each element in **indexing** pattern where it starts with index **0**
- Example:

```
my_list = [1, 2, 3, 4]
```

Creating List

```
# Empty list
my_list = []

# List of integers
my_list = [1, 2, 3, 4]

# List of strings
my_list = ["apple", "banana", "cherry"]

# Mixed data types
my_list = [1, "apple", 3.5, True]

# List within a list (nested list)
my_list = [1, 2, [3, 4], 5]
```

Accessing List Items

- List items are indexed, and these can access them by referring to the index number.

```
thislist = ["apple", "banana", "orange", "cherry"]  
print(thislist[1]). # Output: banana  
print(thislist[-1]) # Output: cherry  
print(thislist[1:3]) # Output: ["banana", "orange"] //slicing  
  
for x in thislist:  
    print(x) # print all elements
```

List Operations

```
mylist=[1,2, 3, 4, 5]
#modyfing list
mylist[1]=9
print(mylist) #output: [1, 9, 3, 4, 5]
#adding elements
mylist.append(10) # adding 10 at the end of list
print(mylist) # output: [1, 9, 3, 4, 5, 10]
#adding using insert function
mylist.insert(2,8) #adding 8 in the 2 index
print(mylist) # Output: [1, 9, 8, 3, 4, 5, 10]
mylist2=[11, 12, 13]
mylist.extend(mylist2) #adding mylist2 with mylist
print(mylist)
```


List methods

```
#sort a list
numbers = [4, 1, 3, 2]
numbers.sort()
print(numbers) # Output: [1, 2, 3, 4]
#reverse a list
numbers = [1, 2, 3, 4]
numbers.reverse()
print(numbers) # Output: [4, 3, 2, 1]
#finding length of list
print(len(numbers)) # Output: 4
#counting occurrences of any item
numbers2 = [1, 2, 3, 2, 2]
print(numbers2.count(2)) # Output: 3
#return the first occurrence of an element
numbers3 = [1, 2, 3, 2]
print(numbers3.index(3)) # Output: 2
```

List sort

- A list can be ordered in ascending and descending order.
- By default, it sorts a list in ascending order.
- `sort()` function is used to order a list.
- Sort list alphabetically
 - `Sort()`
- Sort list in descending order
 - `sort(reverse = True)`

- Example:

```
thislist1 = [100, 50, 65, 82, 23]
thislist1.sort()
print(thislist1)
thislist2 = [100, 50, 65, 82, 23]
thislist2.sort(reverse=True)
print(thislist2)
```

Looping in List

```
thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
    print(x)
```

- Find the sum of all number in a list.
 - Numbers=[20, 30, 50, 42, 56]

User Input in List

- Take n numbers in a list and print the list.

```
n=int(input("Enter how many number you want to input: "))
numbers=list(map(int,input(f"Enter{n} numbers seperated by
space:").split()))
print(numbers)
```

- Take n numbers in a list and print the list using loop.

```
#using loop
n=int(input("Enter the value of n: "))
numbers=[]
for i in range(n):
    num=int(input(f"Enter number{i+1}: "))
    numbers.append(num)
print(numbers)
```

Example

- Take a list of n numbers and find the maximum number among them.
- Sort list in ascending and descending order.

Exercises

- 9.1 Write a python program that takes a list of n numbers and find the maximum number among them.
- 9.2 Find the sum of all numbers in given list.
- 9.3 Count the odd, even numbers in a given list.