# MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9

**Prepared By:**

Mr. Muhammad Ahsan Naeem

**YouTube Playlist**

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

# LAB 22: STRING CLO-4

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. We have been using strings since first lab session. We have also used string concatenation and repetition. Here we will see further detail and built-in functions for strings.

## Indexing and Slicing:

The content (characters) of a string can be individually accessed through indexing very much like list. Both forward and reverse indexing is possible as in case of list. String characters are accessed by specifying the index inside `[]`. For example:

```
msg="Hello Python"
```

| **List Elements** | H | e | l | l | o |  | P | y | t | h | o | n |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Positive Indices** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| **Negative Indices** | -12 | -11 | -10 | -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1 |

If we execute:

```
print(msg[8:11])
```

It will print:

```
tho
```

All slicing variants we have seen in list are applicable for strings as well. Just to refresh that, see the code and respective output:

```
s="HelloWorld"
t=s[:5]+" "+s[-5:-1:1]+s[-1]
print(t)
```

## Built-in Python Functions for String:

Some Python built-in functions for list are given here:

| Method | Description | Example | Output |
|---|---|---|---|
| `len(a)` | Returns the length of the string. | `s="Hello World"`<br>`print(len(s))` | 11 |
| `max(a)` | Returns the maximum character of the string. Maximum is determined on | `s="Hello World"`<br>`print(max(s))` | r |

| | | | |
|---|---|---|---|
| | the basis of ASCII code of the character. | | |
| `min(a)` | Returns the minimum character of the string. Minimum is determined on the basis of ASCII code of the character. | `s="Hello World"`<br>`print(min(s))`<br>`s="HelloWorld"`<br>`print(min(s))` | `H` |
| `sorted(a)` | **a** is the list which will get sorted in ascending order | `s="HelloWorld"`<br>`t=sorted(s)`<br>`print(t)` | `['H', 'W', 'd', 'e', 'l', 'l', 'l', 'o', 'o', 'r']` |
| | List a can also be sorted in reverse order. | `s="HelloWorld"`<br>`t=sorted(s,reverse=True)`<br>`print(t)` | `['r', 'o', 'o', 'l', 'l', 'l', 'e', 'd', 'W', 'H']` |

## Built-in String Methods:

There are many built-in very useful methods in string class for its processing and operations. We can view all these as:

```
print(dir(str))
```

And here is the output:

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__getnewargs__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__iter__', '__le__', '__len__', '__lt__',
'__mod__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__rmod__', '__rmul__', '__setattr__',
'__sizeof__', '__str__', '__subclasshook__', 'capitalize',
'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs',
'find', 'format', 'format_map', 'index', 'isalnum', 'isalpha',
'isascii', 'isdecimal', 'isdigit', 'isidentifier', 'islower',
'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join',
'ljust', 'lower', 'lstrip', 'maketrans', 'partition', 'removeprefix',
'removesuffix', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition',
'rsplit', 'rstrip', 'split', 'splitlines', 'startswith', 'strip',
'swapcase', 'title', 'translate', 'upper', 'zfill']
```

Other than the special methods (sandwiched between double underscores), there are 47 string methods. We will explore most of these step by step. The remaining you can verify by yourself and you can see the help on any string method as:

```
help(str.center)
```

The detail of some commonly used methods is here:

| Method | Description | Example | Output |
|---|---|---|---|
| `lower()` | Converts the string to lower characters | `s="Hello World"`<br>`t=s.lower()`<br>`print(t)` | `hello world` |
| `upper()` | Converts the string to upper characters | `s="Hello World"`<br>`t=s.upper()`<br>`print(t)` | `HELLO WORLD` |
| `count(x)` | Counts the occurrence of x in string | `s="Hello World"`<br>`t=s.count('o')`<br>`print(t)` | `2` |
| `replace(x,y)` | Replaces all x with y | `s="Fed"`<br>`t=s.replace('e','oo')`<br>`print(t)` | `Food` |
| `index(x)` | Gives the index of first occurrence if x. | `s="Hello World"`<br>`t=s.index('o')`<br>`print(t)` | `4` |
| `capitalize()` | Converts first character to Capital Letter | `a='hello world'`<br>`print(a.capitalize())` | `Hello world` |
| `isalpha(x)` | Returns True if all characters in a string are alphabets. | `s="HelloWorld"`<br>`t=s.isalpha()`<br>`print(t)`<br>`s="Hello World"`<br>`t=s.isalpha()`<br>`print(t)` | `True`<br>`False` |

Many other built-in string methods are listed here which you can verify yourself:

| Method | Description |
|---|---|
| `center()` | Pads string with specified character |
| `casefold()` | Converts to casefolded strings |
| `expandtabs()` | Replaces Tab character With Spaces |
| `encode()` | Returns encoded string of given string |
| `find()` | Returns the index of first occurrence of substring |
| `format()` | Formats string into nicer output |
| `isalnum()` | Checks Alphanumeric Character |
| `isdecimal()` | Checks Decimal Characters |

| | |
|---|---|
| `isdigit()` | Checks Digit Characters |
| `isidentifier()` | Checks for Valid Python Identifier |
| `islower()` | Checks if all Alphabets in a String are Lowercase |
| `isnumeric()` | Checks Numeric Characters |
| `isprintable()` | Checks Printable Character |
| `isspace()` | Checks Whitespace Characters |
| `istitle()` | Checks for Title cased String |
| `isupper()` | Returns if all characters are uppercase characters |
| `join()` | Returns a Concatenated String |
| `ljust()` | Returns left-justified string of given width |
| `rjust()` | Returns right-justified string of given width |
| `startswith()` | Returns True if the string starts with the specified value |
| `endswith()` | Returns True if the string ends with the specified value |
| `swapcase()` | Swap uppercase characters to lowercase; vice versa |
| `lstrip()` | Removes Leading/Leftmost Characters |
| `rstrip()` | Removes Trailing/Rightmost Characters |
| `strip()` | Removes Both Leading and Trailing Characters |
| `partition()` | Will return the partition of the string into three parts based on the separator provided as input. Returns a Tuple with three values; the string part before separator, the separator and the string part after separator. |
| `rpartition()` | Similar to partition() but will search for the separator from right side. |
| `maketrans()` | Returns a translation table |
| `translate()` | Returns mapped charactered string |
| `rfind()` | Returns the Highest Index of Substring |
| `rindex()` | Returns Highest Index of Substring |
| `split()` | Splits String from Left |
| `rsplit()` | Splits String From Right |
| `splitlines()` | Splits String at Line Boundaries |
| `title()` | Returns a Title Cased String |
| `zfill()` | Returns a Copy of The String Padded With Zeros |
| `format_map()` | Formats the String Using Dictionary |

## Using `in` to access string characters or substrings:

As we can use `in` to access a list element, we can use it with string to access string characters. An example is shown here:

```
s="Computer"
for i in s:
    print(i)
```

Here is the output of above code:

```
C
o
m
p
u
t
e
r
```

So, if we have a string and we want to print it in a way that each character is printed twice. We can do it in two ways:

Firstly, using **for** loop with loop variable as the index as shown here:

```
s="Computer"
for i in range(len(s)):
    print(str(s[i])*2,end="")
```

Secondly, we can use **in** to access string characters as:

```
s="Computer"
for i in s:
    print(str(i)*2,end="")
```

In above code the loop variable **i** is the character of string **s** starting with first element in first iteration till the last element in the last iteration.

Exactly the way we used **in** to check whether an element is in the list or not, we can use it with string to check a character present in string or not. If we want to check whether a string starts from a vowel character, we can do it like:

```
s="Computer"
if s[0].lower() in 'aeiou':
    print("string starts with a vowel")
else:
    print("string does not start with a vowel")
```

***Tasks:***

**[1]** Write a program that asks user to enter a message and will print the count of vowels in that message. The vowels can be in lower or upper format.

**[2]** Write a program that asks user to enter a message and will display the number of words in that message. Number of words in any message is simply the number of white spaces plus one.

## String to List and Vice Versa:

We can convert a string into list in a way that each character in string becomes an element of that list as shown here:

```
s="Computer Programming"
L=list(s)
print(L)
```

The output is shown:

```
['C', 'o', 'm', 'p', 'u', 't', 'e', 'r', ' ', 'P', 'r', 'o', 'g',
'r', 'a', 'm', 'm', 'i', 'n', 'g']
```

What if we want to create a list of all words inside a string? For that we must use `split` method where we specify a **delimiter** character. To have each word as element of list we will use white space as delimiter as shown here:

```
s="There is no Power on earth that can undo Pakistan"
L=s.split(" ")
print(L)
```

The output is shown below:

```
['There', 'is', 'no', 'Power', 'on', 'earth', 'that', 'can', 'undo',
'Pakistan']
```

Now we can access any word using index from above list.

The opposite is also possible i.e. converting the list into string. If we have a list containing string elements, we can merge them using `join` method. The `join` method is applied on a string and that is inserted in between the other strings provided as input argument:

```
a='ABC'.join(['lmn','pqr','xyz'])
print(a)
```

The output will be:

```
lmnABCpqrABCxyz
```

So, we can use this method to join different strings present inside a list. For that, we should apply this method on a list with one space so that the space is inserted in between when joining all strings.

```
L=["Computer","Programming","is","Fun"]
s=" ".join(L)
print(s)
```

The output will be:

```
Computer Programming is Fun
```

**What if the list includes numeric value?** The above code will not work. There can be ways to handle that. One method is to access individual element and convert to string and use string concatenation.

```
L=["Computer","Programming",2,"will","also","be","Fun"]
s=""
for element in L:
    s+=f'{element} '
print(s)
```

In above case, there will be one extra space at the end of the string. We can eliminate that using the **rstrip()** method.

```
L=["Computer","Programming",2,"will","also","be","Fun"]
s=""
for element in L:
    s+=f'{element} '
s=s.rstrip()
print(list(s))
```

However, the most appropriate way is to covert each element of the list into a string using the list comprehension and use that with **join** method as shown here:

```
L=["Computer","Programming",2,"will","also","be","Fun"]
s=" ".join([str(element) for element in L])
print(s)
```

## Tasks:

**[3]** Take a message from user and display how many 4-character words are in the message.

**[4]** In many word games a word is shown in shuffled way and player is asked to guess the word. This task is not about developing such game but just to implement the feature of generating a shuffled word.

**[5]** A strong password is always preferred. Generally, the agencies asks to set a password that must include small alphabet (a-z), large alphabet (A-Z), numeric value (0-9) and a special character: !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~. Write a program that asks user to set a password. If password meets above criterion, the program must display that password is set successfully else it must display that a valid password must be chosen.

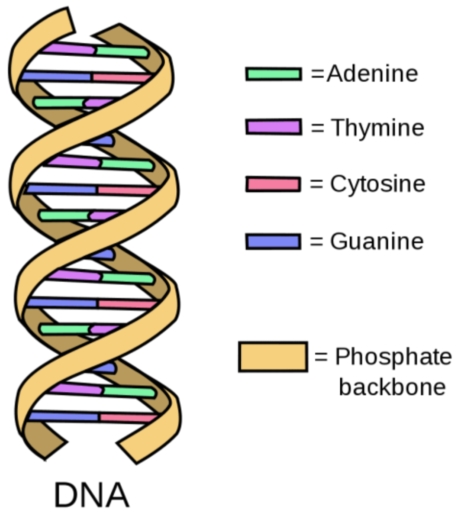
**[6]** Pakistan won the Cricket World Cup in 1992 beating England in the final. The bowling figures of Pakistani bowlers are presented as one string here:

```
bowling="""Wasim Akram   : 10-0-49-3
    Aaqib Javed: 10- 2-27-2
    Mushtaq Ahmed:10- 1-41-3
    Ijaz Ahmed:3-0-13-0
    Imran Khan:6.2-0-43- 1
    Aamer Sohail     : 10-0-49 -0"""
```

Note that there are many whitespaces (Tabs, Spaces and New Lines) in the provided string and those are not symmetric for each bowler. The format of Bowling figures is **Overs-Maiden-Runs-Wickets.**
Write a program that will process the above string to display:

- Name of the bowler, with best Economy (minimum runs per over).
- The total Maiden Overs bowled by Pakistan.
- The name of the bowler who took the most wickets.

**[7]** All living objects are made up of cells and one cell has a lot of components. DNA is one vital part of a cell. A DNA itself is made up of many components and one important component is Nucleotide. There are four types of Nucleotides:
Adenine (A), Thymine (T), Guanine (G), and Cytosine (C).
(Aydaneen GuaNeen)
There can be around 100 Million Nucleotides on one DNA strand (also known as Phosphate Backbone). The order of the Nucleotides on the strand is known as **DNA Sequence**. One DNA molecule has two such strands forming a double Helix as shown below:



If you see carefully, the two strands of DNA are also connected with each other by forming bonds between the nucleotides on each of those. The important thing is that Adenine can get bonded with Guanine only and vice versa and so as the Thymine and Cytosine. These pairs (Adenine-Guanine and Thymine-Cytosine) are also called as Complement of each other.

So, the different Nucleotides are attached on a DNA strands in a way that they have the complement Nucleotide on the other strand to get connected. Hence, we call that one strand is **Reverse Compliment** of the other strand.

Write a program that will:

- Create one DNA Sequence (one strand) with 20 random Nucleotides attached on it.

- The percentage ratio of Adenine and Thymine content on a sequence is known as **AT Content.** Write a function named as `ATContent()` that will take one sequence as input argument and will return the AT Content of that Sequence.
- Write a function named as `reverseComplement()` that will take one sequence as input argument and will return the **Reverse Compliment** of that sequence.