String Processing & File Handling

UNIT - V

Unit-5: String Processing and File Handling

CO5 -Perform string manipulation and file operations to solve a given problem.

- Introduction to String
- Access String elements using index operator
- String functions
- Basic functions: len, max, min
- Testing functions: isalnum, isalpha, isdigit, isidentifier, islower, isupper, and isspace
- Searching functions: endswith, startswith, find, rfind, count
- Manipulation functions: capitalize, lower, upper, title, swapcase, replace, lstrip, rstrip, strip
- Formatting functions: format, center, ljust, rjust
- Introduction to Text files
- File Handling functions:
- Basic functions: open, close
- Reading file: read, readline, readlines
- Writing file: write, append, writelines

String

- Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.
- Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

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

```
>>> a = """ABCDEFGHIJK"""
>>> print(a)
```

- Using triple quotes allows us to make a multiline string or also allows us to use Single and Double quotes as part of string without having to use an escape sequence.

Strings are arrays

- 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.

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

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

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

Loop through the letters in the word "college"

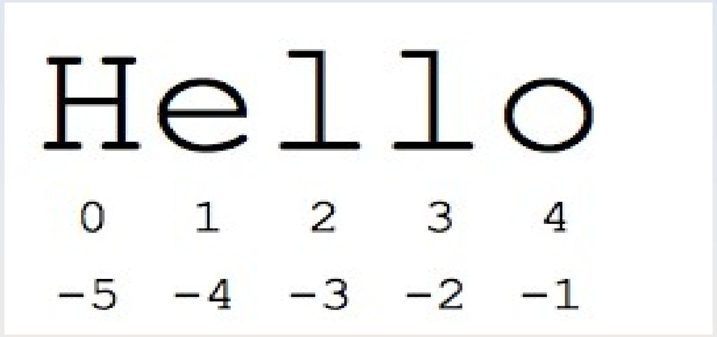
Strings

```
>>> s = 'hi’
>>> print s[1]              # i
>>> print len(s)            # 2
>>> print s + ' there'   # hi there
```

```
>>> pi = 3.14
>>> text = 'The value of pi is ' + pi # NO, does not work
>>> text = 'The value of pi is '  + str(pi)  # yes
```

String Slicing

Let s be,



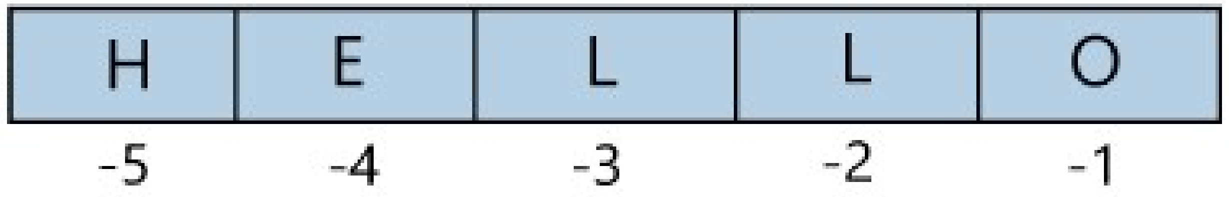
a) **s[1:4] is 'ell'** -- chars starting at index 1 and extending up to but not including index 4

b) **s[1:] is 'ello'** -- omitting either index defaults to the start or end of the string

c) **s[:] is 'Hello'** -- omitting both always gives us a copy of the whole thing (this is the pythonic way to copy a sequence like a string or list)

d) **s[1:100] is 'ello'** -- an index that is too big is truncated down to the string length

e) **s[1:5:2] is 'el'** -- Here the step value is considered to be 2 str = “HELLO”



```
str[-1] = ‘O’        str[-3:-1] = ‘LL’

str[-2] = ‘L’        str[-4:-1] = ‘ELL’

str[-3] = ‘L’        str[-5:-3] = ‘HE’

str[-4] = ‘E’        str[-4:] = ‘ELLO’

str[-5] = ‘H’        str[::-1] = ‘OLLEH’
```

String assignment

```python
str = "HELLO"
str[0] = "h"
print(str)
```

```
Traceback (most recent call last):
  File “demo.py", line 2, in <module>
    str[0] = "h";
TypeError: 'str' object does not support item
assignment
```

```python
str = "HELLO"
print(str)
str = "hello"
print(str)
```

```
HELLO
hello
```

String Operators

Operator	Description
+	It is known as concatenation operator used to join the strings given either side of the operator.
*	It is known as repetition operator. It concatenates the multiple copies of the same string.
[ ]	It is known as slice operator. It is used to access the sub-strings of a particular string.
[: :]	It is known as range slice operator. It is used to access the characters from the specified range.
in	It is known as membership operator. It returns if a particular sub-string is present in the specified string.
not in	It is also a membership operator and does the exact reverse of in. It returns true if a particular substring is not present in the specified string.
r/R	It is used to specify the raw string. Raw strings are used in the cases where we need to print the actual meaning of escape characters such as "C://python". To define any string as a raw string, the character r or R is followed by the string.
%	It is used to perform string formatting. It makes use of the format specifiers used in C programming like %d or %f to map their values in python.

Example

```
>>> str = "Hello"
>>> str1 = " world"
>>> print(str*3) # prints HelloHelloHello
>>> print(str+str1) # prints Hello world
>>> print(str[4]) # prints o
>>> print(str[2:4]) # prints ll
>>> print('w' in str) # prints false as w is not present in str
>>> print('wo' not in str1) #prints false as wo is present in str1
>>> print(r'C://python37') # prints C://python37 as it is written
>>> print("The string str : %s"%(str)) #prints The string str : Hello
```

String Basic Functions

len()

- It returns the length of a string.
- `a = "Hello, World!"`
- `print(len(a)) #returns 13`

min()

- Return the name with the lowest value, ordered alphabetically
- `x = min("Mahesh", "Suresh", "Ramesh")`
- `print(x) #returns Mahesh`

max()

- It will return the largest element (ordered alphabetically).
- `my_strings = ["Apple","Mango","Papaya","Orange"]`
- `large_str = max(my_strings)  #returns Papaya`

Testing functions

isalnum	isalpha	isdigit	isidentifier
islower	isupper	isspace	isascii
isdecimal	isnumeric	isprintable	istitle

Testing functions

isalnum

- Returns True if all characters in the string are alphanumeric
- **>>> s="A bite of python"**
- **>>> s.isalnum()**
- False

isalpha

- Returns True if all characters in the string are in the alphabet
- >>> **"zyz".isalpha()**
- True

isdigit

- Returns True if all characters in the string are digits
- "**101.129".isdigit()** # Returns False
- **"101".isdigit()** # Returns True

Testing functions

isidentifier

- Returns True if the string is an identifier
- >>>"**2bring".isidentifier()**
- False

islower

- Returns True if all characters in the string are lower case
- **>>> "abc".islower()**
- True

isupper

- Returns True if all characters in the string are upper case
- **>>> "aBC".isupper()**
- False

Testing functions

isspace

- Returns True if all characters in the string are whitespaces
- **>>>" \n\t".isspace()**
- True

isascii

- Returns True if all characters in the string are ascii characters
- **>>>"Company123".isascii()**
- True

isdecimal

- Returns True if all characters in the string are decimals.
- **>>>"\u0033".isdecimal() #Unicode of 3**
- True

Testing functions

isnumeric

- Returns True if all characters in the string are numeric
- >>>"565".isnumeric()
- True

isprintable

- Returns True if all characters in the string are printable
- >>>"Hello! Are you #1?".isprintable()
- True
- >>>"Hello!\n Fine".isprintable()
- False

istitle

- returns True if all words in a text start with a upper case letter, AND the rest of the word are lower case letters, otherwise False. Numbers and Symbols are ignored.
- >>>"Hello123".istitle()
- True

Searching functions

endswith

startswith

find

rfind

index

rindex

count

Searching functions

endswith

- The method returns True if the string ends with the specified value, otherwise False.
- *string*.endswith(*value*, *start*, *end*)

-
  ```
  txt = "Hello, welcome to my world."
  x = txt.endswith("my world.")
  print(x)
  ```
- True

startswith

- The method returns True if the string starts with the specified value, otherwise False.
- *string*.startswith(*value*, *start*, *end*)

-
  ```
  txt = "Hello, welcome to my world."
  x =txt.startswith("wel", 7, 20)
  print(x)
  ```
- True

Searching functions

find

- The find() method finds the first occurrence of the specified value.
- The find() method returns -1 if the value is not found.
- The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found.
- `string.find(value, start, end)`
-
```
txt = "Hello, welcome to my world."
x = txt.find("e")
print(x)  # 1
```

rfind

- The rfind() method finds the last occurrence of the specified value.
- The rfind() method returns -1 if the value is not found.
- The rfind() method is almost the same as the rindex() method.
- `string.rfind(value, start, end)`
-
```
txt = "Hello, welcome to my world."
x = txt.rfind("e")
print(x)
```
- 13

nctions

index

- The index() method finds the first occurrence of the specified value.
- The index() method raises an exception if the value is not found.
- The index() method is almost the same as the [find()]
- `string.index(value, start, end)`

-
  ```
  txt = "Hello, welcome to my world."
  x = txt.index("e", 5, 10)
  ```

rindex

- The rindex() method finds the last occurrence of the specified value.
- The rindex() method raises an exception if the value is not found.
- `string.rindex(value, start, end)`

-
  ```
  txt = "Hello, welcome to my world."
  ```

count

- The method returns the number of times a specified value appears in the string
- `string.count(value, start, end)`

-
  ```
  txt = "I love apples, apple are my favorite fruit"
  x =
  txt.count("apple", 10, 24
  )
  ```

Manipulation functions

- capitalize
- casefold
- center
- ljust
- rjust
- lower
- upper
- title
- swapcase
- replace
- lstrip
- rstrip
- strip

Manipulation functions

capitalize

- Returns a copy of the original string and converts the first character of the string to a capital **(uppercase)** letter, while making all other characters in the string **lowercase** letters.
- `“live laugh”.capitalize()`
- `Live Laugh`

casefold

- Returns a string where all the characters are lower case.
- `“Hello world”.casefold()`
- `hello world`

Manipulation functions

lower

upper

swapcase

replace

Manipulation functions

lstrip

rstrip

strip

- The method removes any leading characters (space is the default leading character to remove).
- "   geeksforgeeks“.

Formatting functions

format

center

ljust

rjust

Formatting functions

- format()
- The format() method formats the specified value(s) and insert them inside the string's placeholder.
- The placeholder is defined using curly brackets: {}.

```python
txt = "For only {price:.2f} Rs"
print(txt.format(price = 49))
```

```
For only 49.00 Rs
```

Formatting functions

- center()
- The center() method will center align the string, using a specified character (space is default) as the fill character.
- *string*.center(*length, character*)

Parameter	Description
length	Required. The length of the returned string
character	Optional. The character to fill the missing space on each side. Default is " " (space)

```
txt = "banana"
x = txt.center(20, "O")
print(x)
```

```
OOOOOOObananaOOOOOOO
```

Formatting functions

- ljust()
- The ljust() method will left align the string, using a specified character (space is default) as the fill character.
- *string*.ljust(*length*, *character*)

```
>>> txt = "banana"
>>> x = txt.ljust(20, "O")
>>> print(x)
```

```
bananaOOOOOOOOOOOOOO
```

Formatting functions

- rjust()
- The rjust() method will right align the string, using a specified character (space is default) as the fill character.
- *string*.rjust(*length*, *character*)

```
>>> txt = "banana"
>>> x = txt.rjust(20, "O")
>>> print(x)
```

```
OOOOOOOOOOOOOObanana
```

File Handling with Python

- Introduction to Text files
- File Handling functions:
- Basic functions: open, close
- Reading file: read, readline, readlines
- Writing file: write, append, writelines

Introduction to Text files

- Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).
- There are **two types** of files that can be handled in python, normal text files and binary files (written in binary language, 0s, and 1s).
- **Text files:** In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character (‘\n’) in python by default.
- **Binary files:** In this type of file, there is no terminator for a line, and the data is stored after converting it into machine-understandable binary language.
- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

File Modes

Read Only ('r') :	• Open text file for reading. The handle is positioned at the beginning of the file. If the file does not exists, raises the I/O error. This is also the default mode in which a file is opened.
Read and Write ('r+'):	• Open the file for reading and writing. The handle is positioned at the beginning of the file. Raises I/O error if the file does not exist.
Write Only ('w') :	• Open the file for writing. For the existing files, the data is truncated and over-written. The handle is positioned at the beginning of the file. Creates the file if the file does not exist.
Write and Read ('w+') :	• Open the file for reading and writing. For an existing file, data is truncated and over-written. The handle is positioned at the beginning of the file.
Append Only ('a'):	• Open the file for writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.
Append and Read ('a+') :	• Open the file for reading and writing. The file is created if it does not exist. The handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

File Handling functions : Basic functions: open, close

- **open()**
- Python has a built-in open() function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.
- `>>> f = open("test.txt")    # open file in current directory`
- `>>> f = open("C:/Python38/README.txt")  # specifying full path`

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists
- `>>>f = open("demofile.txt", "rt")`

File Handling functions : Basic functions: open, close

- `close()`
- close() function closes the file and frees the memory space acquired by that file.
- It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.
- `>>>file1 = open("MyFile.txt","a")`
- `>>>file1.close()`

Reading file: read, readline, readlines

- Reading file: read, readline, readlines

```
f=open("myFile.txt","w")
f.write("Yesha\n")
f.write("Shanvi\n")
f.write("Vidhi\n")
f.close()
print("File Created")
print("Reading entire File")
f=open("myFile.txt","r")
print(f.read())
f.close()
print("Reading First Five Characters”)
f = open("myFile.txt","r")
print (f.read(5))
f.close()
```

```
#Output
File Created
Reading entire File
Yesha
Shanvi
Vidhi

Reading First Five Characters
Yesha
```

Reading file: read, readline, readlines

- Writing file: readline

```
f=open("myFile.txt","w")
f.write("Yesha\n")
f.write("Shanvi\n")
f.write("Vidhi\n")
f.close()
print("File Created")
print("Reading File")
f=open("myFile.txt","r")
print(f.readline())
print(f.readline())
print(f.readline())
f.close()
```

```
#Output
File Created
Reading File
Yesha

Shanvi

Vidhi
```

Reading file: read, readline, readlines

- Writing file: readlines

```
f=open("myFile.txt","w")
FileContent=["Yesha\n“,"Shanvi\n“,"Vidhi\n”]
f.writelines(FileContent)
f.close()
print("File Created")
print("ReadingFile")
f=open("myFile.txt","r")
ReadLines=f.readlines()
for x in ReadLines:
   print(x)
f.close()
```

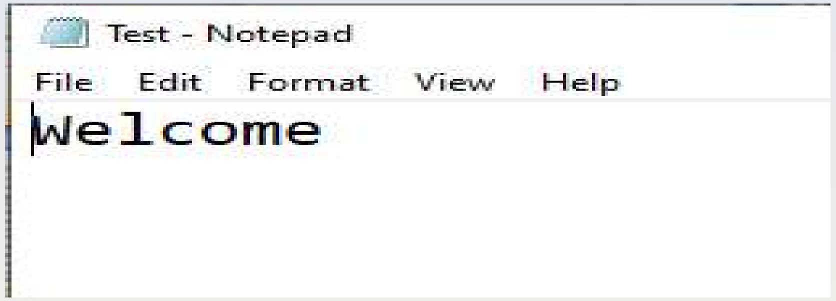
```
#Output
File Created
Reading File
Yesha

Shanvi

Vidhi
```

Writing file: write, append, writelines

- Writing file: write

```
fp = open("Test.txt","w")
fp.write("Welcome\n")
fp.close()
```

#Output

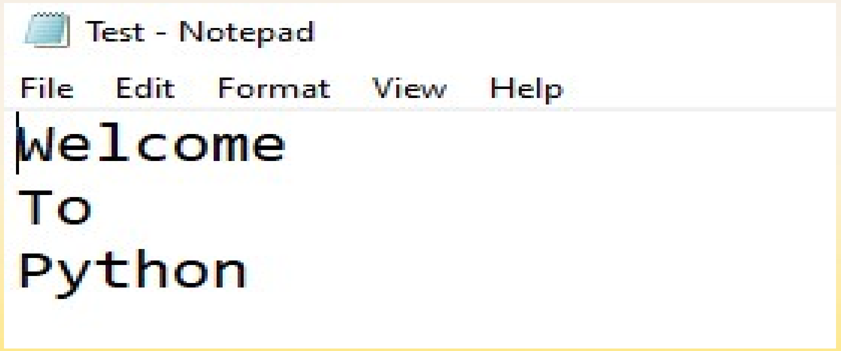


```
Welcome
```

- Writing file : Writelines

```
String_List=["Welcome\n",
  "To\n","Python\n"]
fp=open("Test.txt","w")
fp.writelines (String_List)
```

#Output



```
Welcome
To
Python
```

Random access in file: tell () ,seek()

- tell():
  - Returns current position of file pointer in file.
  - *Syntax:position=File_Object.tell()*
- seek():
  - Moves file pointer to specific position in file.
  - *Syntax:File_Object.seek(offset[,from])*
  - Offset→ indicates number of bytes to be moved.
  - From→ indicates reference position.
  - It means from which position the file pointer will moves.It can have one of following three values:
    - 0: Beginning of File
    - 1: Current Position
    - 2: End of File

Random access in file: tell () ,seek()

```
f=open("myFile.txt","w")
f.write("Linel\nLine2\nLine3\n")
f.close()
print("File Created")
f=open("myFile.txt","r")
position=f.tell()
print("FilePointer Position:",position)
print (f.read(5))
position=f.tell()
print("File Pointer Position:", position)
f.seek (14,0)
print(f.read(5))
position=f.tell()
print("File Pointer Position:",position)
f.seek (7,0)
print (f.read(5))
position=f.tell ()
print("File Pointer Position:",position)
f.close()
```

```
#OUTPUT
File Created
FilePointer Position: 0
Linel
File Pointer Position: 5
Line3
File Pointer Position: 19
Line2
File Pointer Position: 12
```