MODULE 2

1 ITERATION

1.1 UPDATING VARIABLE

- Consider the following assignment statement:

  x = x + 1

  This means that get the current value of x, add 1 and the update x with the new value.
- If you try to update a variable that doesn't exist, you get an error, because Python evaluates the right side before it assigns a value to x:

  >>> x = x + 1

  NameError: name 'x' is not defined
- We have to initialize the variable before updating it using assignment statement.

  For example:

  >>> x = 0

  >>> x = x + 1
- Updating a variable by 1 is called **increment**, subtracting by 1 is called **decrement**.

1.2 THE WHILE STATEMENT (LOOPING STATEMENT)

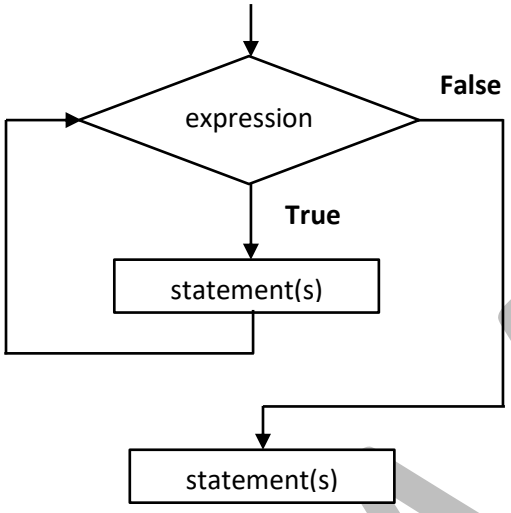
- **A set of statements that are executed repeatedly until a condition is satisfied is called a loop. When a condition is not satisfied then the loop is terminated. It useful in performing repetitive tasks.**
- **Whenever we want to do some task repeatedly we use while statement.**
- The syntax of while statement is:

```
while expression :
    Statement 1
    Statement 2
    …..
    Statement n
Statement(s)
```

- The **flow of execution of while statement** is:
  - i. **Evaluate the condition, which results in either True or False.**
  - ii. **If the condition is False, exit the while statement and continue execution at the next statement.**
  - iii. **If the condition is True, then execute the body and then goto step 1.**
    - ✓ This type of flow is called **loop**, since the third step loops back to the top.
    - ✓ Each time we execute the body of the loop we call it an **iteration**.
    - ✓ The body of the loop should change the value of one or more variables so that eventually the condition becomes false and the loop terminates. We call the variable that changes each time the loop executes and controls when the loop finishes the ***iteration variable***.

- ✓ If there is no iteration variable, the loop will repeat forever, resulting in an ***infinite loop***.

➢ The flowchart of while statement is:



➢ Consider a simple program that counts from 5 to 1.

```
n = 5
while n > 0 :
    print(n)
    n = n - 1
print("Good Bye!!!")
```

Output:

```
5
4
3
2
1
Good Bye!!!
```

While n is greater than 0, display the value of n and then decrement the value of n by 1. When you get to 0, exit the while statement and display "Good Bye!!!".

1.3 THE FOR STATEMENT (LOOPING STATEMENT)

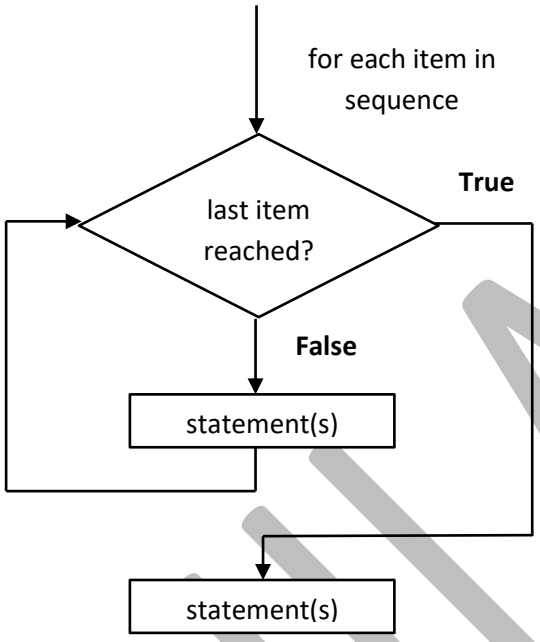
➢ **The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects.**

➢ The syntax of for loop is:

```
for variable_name in sequence:
    Statement 1
    Statement 2
    …..
    Statement n
Statement(s)
```

- Here, `variable_name` is the variable that takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence.
- The body of for loop is separated from the rest of the code using indentation.

➢ The flowchart of while statement is:



➢ Consider a program to print each character in a word.

```
for letter in 'python':
   print(letter)
```

Output:

```
p
y
t
h
o
n
```

➢ We can **generate a sequence of numbers using `range()` function**.

- **`Syntax: range(number)`**
  `For example: range(10)` will generate numbers from 0 to 9 (10 numbers).
- We can also define the start, stop and step size as
  Syntax: range(start, stop, step_size)
  or
  range(lower limit, upper limit, Increment/Decrement by).
  step_size defaults to 1 if not provided.
  This function **does not store all the values in memory**, it would be inefficient. So **it remembers the start, stop, step size and generates the next number on the go**.
- **To force this function to output all the items, we can use the function `list()`.**

For Example:

- **# Output: range(0, 10)**
  print(range(10))
- **# Output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]**
  print(list(range(10)))
- **# Output: [2, 3, 4, 5, 6, 7]**
  print(list(range(2, 8)))
- **# Output: [2, 5, 8, 11, 14, 17]**
  print(list(range(2, 20, 3)))
- **# To print numbers from 5 to 0.**
  for i in range(5,-1,-1):
  print(i)
  Output:
  5
  4
  3
  2
  1
  0
- **#To print sum of first 10 natural numbers.**
  sum = 0
  for i in range(11):
  sum = sum + i
  print(sum)
  Output:
  55
- **# To print numbers from 1 to 5.**
  for i in range(1,6):
  print(i)
  Output:
  1
  2
  3
  4
  5

1.4 BREAK STATEMENT (LOOP CONTROL/JUMP STATEMENT)

- Loops iterate over a block of code until test expression is false, but sometimes we wish to terminate the current iteration or even the whole loop without checking test expression.
- Break statement is a **jump statement which is used to transfer execution control**.
- It **terminates the current loop** and **resumes execution at the next statement**.
- The general syntax of break statement is:

  break
- Consider the following examples:
  - **for var in 'python':**

```
    if var == 't':
        break
    print(var)
print('End of program!!!')
```

Output:

```
p
y
End of program!!!
```

- ```
  for i in range(10):
      if i == 5:
          break
      print(i)
  print('End of program!!!')
  ```

  Output:

  ```
  0
  1
  2
  3
  4
  End of program!!!
  ```

- ```
  #To check if 11 is prime or not.
  n=11
  flag = 0
  for i in range(2,n):
      if n%i == 0:
          flag = 1
          break
  if flag == 1:
      print('Not Prime number')
  else:
      print('Prime number')
  ```

  Output:

  ```
  Prime number
  ```

1.5 CONTINUE STATEMENT (LOOP CONTROL/JUMP STATEMENT)

- The **continue statement is used to skip the rest of the code inside a loop for the current iteration only**. **Loop does not terminate but continues on with the next iteration**.
- The general syntax of continue statement is:

```
continue
```

- Consider the following examples:
  - ```
    for var in 'python':
        if var == 't':
            continue
        print(var)
    print('End of program!!!')
    ```

    Output:

    ```
    p
    y
    h
    o
    n
    End of program!!!
    ```
  - ```
    for i in range(10):
        if i == 5:
            continue
        print(i)
    print('End of program!!!')
    ```

    Output:

    ```
    0
    1
    2
    3
    4
    6
    7
    8
    9
    End of program!!!
    ```

Note: Loop control/Jump statements can be used only within the loops.

1.6 EXERCISES

1. Write a python program to print natural numbers from 1 to n.
2. Write a python program to print natural numbers from n to 1.
3. Write a python program to print odd numbers from 1 to n.
4. Write a python program to print even numbers from 1 to n.
5. Write a python program to check if a number is prime or not.
6. Write a python program to print all prime numbers between 1 to n.
7. Write a python program to find sum of first n natural numbers.

8. Write a python program to find sum of squares of natural numbers from 1 to n. ($1^2+2^2+3^2+4^2…+n^2$)
9. Write a python program to find the sum of $x+x^2/2+x^3/3+…..+x^n/n$.
10. Write a python program to find the sum of sin(x) using Taylor Series expansion.
11. Write a python program to find the sum of cos(x) using Taylor Series expansion.
12. Write a python program to find sum of all odd numbers from 1 to n.
13. Write a python program to find sum of all even numbers from 1 to n.
14. Write a python program to find factorial of a number.
15. Write a python program to find Fibonacci series.
16. Write a python program to generate multiplication table.
17. Write a python program to find HCF of two numbers.
18. Write a python program to find LCM of two numbers.
19. Write a python program to find the number of digits in an integer number.
20. Write a python program to calculate the sum of digits of a number.
21. Write a python program to reverse an integer number.
22. Write a python program to calculate power of a number.
23. Write a python program to check whether a given number is palindrome or not.
24. Write a python program to check whether a given number is Armstrong number of not. (Eg: 153, 371, 1634…)
25. Write a python program to implement a scientific calculator.

1.7 GLOSSARY

accumulator	A variable used in a loop to add up or accumulate a result.
counter	A variable used in a loop to count the number of times something happened. We initialize a counter to zero and then increment the counter each time we want to "count" something.
decrement	An update that decreases the value of a variable.
initialize	An assignment that gives an initial value to a variable that will be updated.
increment	An update that increases the value of a variable (often by one).
infinite loop	A loop in which the terminating condition is never satisfied or for which there is no terminating condition.
iteration	Repeated execution of a set of statements using either a function that calls itself or a loop.

2 STRINGS

2.1 A STRING IS A SEQUENCE

A string is a sequence of characters.

CREATE STRING IN PYTHON

- Strings can be created by enclosing characters inside a single quote or double quotes.
- Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.
- The general syntax of creating a string is:

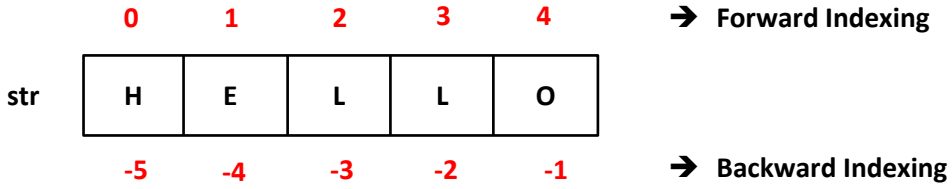
```
variable_name = 'string'
```

- **For Example:**
  - **my_string = 'Hello'**
    print(my_string)
    Output:
    Hello
  - **my_string = "Hello"**
    print(my_string)
    Output:
    Hello
  - **my_string = '''Hello'''**
    print(my_string)
    Output:
    Hello
  - **my_string = """Hello, welcome to the world of Python"""**
    print(my_string)
    Output:
    Hello, welcome to
    the world of Python

ACCESSING CHARACTER IN A STRING

- In Python, **Strings are stored as individual characters in a contiguous memory location**.
- The benefit of using String is that it can be **accessed from both the directions in forward and backward**.
  Forward indexing starts with 0,1,2,3,....
  Backward indexing starts with -1,-2,-3,-4,....
- We can **access the characters one at a time with the bracket operator ([])**.

- In the below figure str[0] gives the first letter, str[1] gives second letter and so on. Similarly str[-1] gives the last letter, str[-2] gives the last but one letter and so on. The below figure shows string indexes:



- **The expression in brackets is called an index. Index starts from 0.**
- **The index must be an integer**. We **can't use float or other types**, this will **result into `TypeError`**.
- **Trying to access a character out of index range will raise an `IndexError`.**
- We can **access individual characters using indexing and a range of characters using slicing**. **We can access a range of items in a string by using the slicing operator (colon i.e., ':').**
  For Example:
  - **movie = 'titanic'**
    letter = movie[2]
    print(letter)
    Output:
    t
  - **str = 'programming'**
    print('str = ', str)
    Output:
    str = programming
  - **#first character**
    str = 'programming'
    print('str [0] = ', str [0])
    Output:
    str [0] = p
  - **#last character**
    str = 'programming'
    print('str [-1] = ', str [-1])
    Output:
    str [-1] = g
  - **#slicing 2nd to 5th character**
    str = 'programming'
    print('str [1 : 5] = ', str [1 : 5])
    Output:
    str [1 : 5] = rogr
  - **#slicing 6th to 2nd last character**

```
str = 'programming'
print('str [ 5 : -2] = ', str [ 5 : -2 ])
```

Output:

```
str [ 5 : -2 ] = ammi
```

- #This will not work

```
print('str [ -2 : 5 ] = ', str [ -2 : 1 ])
```

Output:

```
str [ -2 : 1 ] =
```

- #This will not work

```
print('str [ -2 : -5 ] = ', str [ -2 : -5 ])
```

Output:

```
str [ -2 : -5 ] =
```

- print('str[1.5] = ', str[1.5])

Output:

```
Traceback (most recent call last):
  File "C:/Users/akhil/AppData/Local/Programs/Python/Python36-32/str.c", line 22, in <module>
    print('str[1.5] = ', str[1.5])
TypeError: string indices must be integers
```

- print('str[15] = ', str[15])

Output:

```
Traceback (most recent call last):
  File "C:/Users/akhil/AppData/Local/Programs/Python/Python36-32/str.c", line 24, in <module>
    print('str[15] = ', str[15])
IndexError: string index out of range
```

2.2 GETTING THE LENGTH OF A STRING USING LEN

- **len is a built-in function that returns the number of characters in a string.**
- **For Example:**

```
>>> player = 'virat'
>>> len(player)
5
```

- **To get the last letter of the string we have to do this:**

```
>>> player = 'Virat'
>>> length = len(player)
>>> last = player[length-1]
>>> print(last)
t
```

→ Since the index of a string starts with 0, the last character we have to subtract 1 from the length.

- We can also **use negative indexes to get the last character.**

```
>>> player = 'Virat'
>>> last = player[-1]   → -1 index gives you the last character in the string.
>>> print(last)
t
>>> last = player[-5]
>>> print(last)
v
```

2.3 TRAVERSAL OF A STRING WITH A LOOP

- Start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a traversal.
- The forward traversal of a string is:

```
player = input('Enter the player name: ')     # Read the string from the user
index = 0                                     # Initialize index to 0.
length = len(player)                          # Compute the length of the entered string.
while index < length:                         # Loop till the index is less than length of the string.
   letter = player[index]                     # Store the index^th character in letter.
   print(letter)                              # Display the letter.
   index = index + 1                          # Increment the index.
```

Output:

```
Enter the player name: virat
v
i
r
a
t
```

OR

Another way to write a traversal is with a for loop:

Each time through the loop, the next character in the string is assigned to the variable char. The loop continues until no characters are left.

```
player = input('Enter the player name: ')
for char in player:
   print(char)
```

Output:

```
Enter the player name: virat
v
i
r
a
t
```

OR

```
player = input('Enter the player name: ')
length = len(player)
for i in range(0,length):
   print(player[i])
```

Output:

```
Enter the player name: virat
v
i
r
a
t
```

- The backward traversal of a string is:

```
player = input('Enter the player name: ')
length = len(player)
i = length - 1
while i >= 0:
   print(player[i])
   i=i-1
```

→ Exercise 1

Output:

```
Enter the player name: virat
t
a
r
i
v
```

OR

```
player = input('Enter the player name: ')
length = len(player)
for i in range(length-1, -1, -1):
   print(player[i])
```

Output:

```
Enter the player name: virat
t
a
r
i
v
```

OR

```
player = input('Enter the player name: ')
length = len(player)
for i in range(-1, -(length+1), -1):
   print(player[i])
```

Output:

```
Enter the player name: virat
t
a
r
i
v
```

- To reverse a string

```
player = input('Enter the player name: ')
print(The reverse of the string is', player[ : : -1 ] )
```

Output:

```
Enter the player name: virat
The reverse of the string is tariv
```

2.4 STRING SLICES

- A segment of a string is called a slice.
- For Example:

```
>>> s = 'Leela Palace'
>>> print(s[0:4])
Leel
>>> print(s[0:5])
Leela
>>> print(s[0:6])
Leela
>>> print(s[0:7])
Leela P
>>> print(s[2:9])
ela Pal
>>> print(s[-1:-7])      → This won't work.
                         → It results in an empty string
>>> print(s[-1])
e
>>> print(s[-4])
l
```

The operator returns the part of the string from the "n-eth" character to the "m-eth" character, including the first but excluding the last.

- **If you omit the first index (before the colon), the slice starts at the beginning of the string. If you omit the second index, the slice goes to the end of the string.**

```
>>> veg = 'babycorn'
>>> print(veg[:4])
baby
>>> print(veg[4:])
corn
```

- **If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks.**

```
>>> fruit = 'apple'
>>> print(fruit[2:2])
                          → Empty String
>>> print(fruit[-1:0])
                          → Empty String
>>> print(fruit[-3:-1])
pl
>>> print(fruit[-5:-1])
appl
>>> print(fruit[-5:0])
                          → Empty String
>>>
>>> print(fruit[:])       → Exercise 2
Apple
```

Note: An empty string contains no characters and has length 0, but other than that, it is the same as any other string.

2.5 STRINGS ARE IMMUTABLE

- **Strings are immutable. This means that elements of a string cannot be changed once it has been assigned.** We can simply reassign different strings to the same name.
- **For Example:**

```
>>> msg = 'Hello World'
>>> msg[3] = 'z'
Traceback (most recent call last):
  File "<pyshell#25>", line 1, in <module>
    msg[3] = 'z'
TypeError: 'str' object does not support item assignment
```

The "object" in this case is the string and the "item" is the character you tried to assign. For now, an object is the same thing as a value. An item is one of the values in a sequence.

- We can only **reassign different strings to the same name or create a new string that is variation of the original**.

For Example:

```
>>> msg = 'Hello World'
>>> new_msg = 'Y' + msg[1:]
>>> print(new_msg)
Yello World
>>> msg = 'Z' + msg[:]
>>> print(msg)
ZHello World
>>> msg = 'Good Morning'
>>> print(msg)
Good Morning
```

2.6 LOOPING AND COUNTING

- The following program counts the number of times the letter a appears in a string:

```
#To count the number of a's in the given word.
word = input('Enter the word: ')
count = 0
for i in word:
   if i == 'a':
      count = count + 1
print('The no of a\'s in', word, 'is', count)
```

Output:

```
Enter the word: malayalam
The no of a's in malayalam is 4
```

This program demonstrates another pattern of computation called a counter. The variable count is initialized to 0 and then incremented each time ‘a’ is found. When the loop exits, count contains the result: the total number of a’s.

Exercise 3:

```
#To count the number of occurrences of a character in the given word using functions.
def count(word, letter):
   counter = 0
   for i in word:
      if i == letter:
         counter = counter + 1
   print('The no of’, letter, ‘in', word, 'is', counter)

word = input('Enter the word: ')
letter = input(‘Enter the character: ‘)
count(word,letter)
```

Output:

Enter the word: Malayalam
Enter the character: a
The no of a in malayalam is 4

2.7 STRING OPERATIONS

1. CONCATENATION OPERATOR (+) and REPLICATION OPERATOR (*)

- **Joining of two or more strings into a single one is called concatenation.**
- The **+** operator does this in Python. Simply writing two string literals together also concatenates them.
- The ***** operator can be used to **repeat the string for a given number of times**.

For Example:

- **s1 = 'Monty'**
  s2 = 'Python'

  print('s1 + s2 = ', s1+s2)

  print('s2 * 3 = ', s2 * 3)
  Output:
  s1 + s2 = MontyPython
  s2 * 3 = PythonPythonPython
- **>>> s1 = 'Monty'**
  >>> s2 = 'Python'
  >>> print(s1+3)
  Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
  print(s1+3)
  TypeError: must be str, not int
  >>> print(s1+'3')
  Monty3

- Writing two string literals together also concatenates them like **+** operator.
- If we want to concatenate strings in different lines, we can use parentheses.

For Example:

>>> 'Monty''Python'
'MontyPython'
>>> s = ('Hello'
'World')
>>> s
'HelloWorld'

2. in and not in OPERATOR (MEMBERSHIP OPERATOR)

- **in is a boolean operator that takes two strings and returns True if the first appears as a substring in the second.**

  For Example:

```
>>> 'a' in 'akhilaa'
True
>>> 'aa' in 'akhilaa'
True
>>> 'hl' in 'akhilaa'
False
>>> 'AA' in 'akhilaa'
False
>>> 'AA' not in 'akhilaa'
True
>>> 'y P' in 'Monty Python'
True
```

3. **RELATIONAL OPERATOR (<, >, <=, >=, ==, !=)**
   - The Strings are compared based on the ASCII value or Unicode(i.e., dictionary Order).
   - **All the uppercase letters come before all the lowercase letters.**
   - The **ASCII value of a is 97, b is 98, c is 99** and so on. The **ASCII value of A is 65,B is 66,C is 67** and so on. The comparison between strings are done on the basis on ASCII value.

     For Example:

```
>>> 'chicken' == 'fish'
False
>>> 'AKHILAA' == 'AKHILAA'
True
>>> 'kabab' != 'babycorn'
True
>>> 'abc' >= 'ABC'
True
>>> 'xyz' <= 'XYZ'
False
>>> 'abcd' < 'abce'
True
>>> 'abce' > 'abcd'
True
>>> 'abce' < 'abcd'
False
>>> '3' < 'a'
True
>>> '5' < '9'
True
>>> '9' < '3'
```

```
False
>>>
```

Note: To find the ASCII value, we have to use ord()

```
>>> ord('a')
97
>>> ord('z')
122
>>> ord('A')
65
>>> ord('Z')
90
>>> ord('3')
51
>>> ord(' ')
32
>>> ord('@')
64
>>> ord('<')
60
```

2.8 STRING METHODS

- Strings are an example of Python *objects*. An object contains both data (the actual string itself) and *methods.*
- **Python has a function called dir which lists the methods available for an object. The type function shows the type of an object and the dir function shows the available methods.**

In Interactive mode,

```
>>> string = 'python programming'
>>> type(string)
<class 'str'>
>>> dir(string)
['capitalize', 'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier',
'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'ljust', 'lower',
'lstrip', 'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper', 'zfill']
>>> help(str.capitalize)
Help on method_descriptor:

capitalize(...)
   S.capitalize() -> str
```

Return a capitalized version of S, i.e. make the first character have upper case and the rest lower case.

- While the dir function lists the methods, and you can use help to get some simple documentation on a method, a better source of documentation for string methods would be https://docs.python.org/3.5/library/stdtypes.html#string-methods.
- Calling a *method* is similar to calling a function (it takes arguments and returns a value) but the syntax is different. We call a method by appending the method name to the variable name using the period as a delimiter.
- Some of the functions are mentioned below:

1. **upper()**
   - **The upper() method returns the uppercased string from the given string. It converts all lowercase characters to uppercase.**
   - **If no lowercase characters exist, it returns the original string.**
   - **For example:**

```
>>> str = 'python application programming'
>>> print(str.upper())
PYTHON APPLICATION PROGRAMMING
```

2. **find()**
   - **Syntax:**

```
str.find(sub[, start[, end]] )
```

sub - It's the substring to be searched in the *str* string.
start (optional) - starting index, by default its 0.
end (optional) – ending index, by default its equal to the length of the string
Note: [] means optional.

   - **The find() method returns an integer value:**
     - ✓ **If substring exists inside the string, it returns the lowest index where substring is found.**
     - ✓ **If substring doesn't exist inside the string, it returns -1.**
   - **For example:**

I.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('old'))
14
>>> quote = 'it is not too Old and it is not too late'
>>> print(quote.find('old'))
-1
```

II.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('o',10))
11
>>> print(quote.find('o',15))
29
```

III.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('too',9,30))
10
>>> print(quote.find('too',10,30))
10
>>> print(quote.find('too',20,30))
-1
>>> print(quote.find('too',30,20))
-1
>>> print(quote.find('too',20,40))
32
```

IV.
```
>>> quote = 'it is not too old and it is not too late'
>>> print(quote.find('and',-40,-10))
18
>>> print(quote.find('and',-10,-40))
-1
```

3. **strip()**

- **Syntax:**

```
str.strip( [chars] )
```

chars (optional) - a string specifying the set of characters to be removed. If the *chars* argument is not provided, all leading and trailing whitespaces are removed from the string.

- The strip() returns a copy of the string with both leading and trailing characters stripped.
  - ✓ When the combination of characters in the *chars* argument mismatches the character of the string in the left, it stops removing the leading characters.
  - ✓ When the combination of characters in the *chars* argument mismatches the character of the string in the right, it stops removing the trailing characters.
- **For example:**

I.
```
>>> string = '   welcome to python programming   '
>>> print(string.strip())
welcome to python programming
```

II.
```
>>> string = 'welcome to python programming'
>>> print(string.strip())
welcome to python programming
```

III.
```
>>> string = '***welcome to python programming***'
>>> print(string.strip('*'))
welcome to python programming
```

IV.
```
>>> string = 'welcome to python programming'
>>> print(string.strip('welcome'))
 to python programming
```

V.
```
>>> string = 'welcome to python programming'
```

```
>>> print(string.strip('welcome to '))
python programming
```

VI.
```
>>> string = 'welcome to python programming'
>>> print(string.strip('gam'))
welcome to python programmin
```

4. **startswith()**

- **Syntax:**

```
str.startswith(prefix[, start[, end]])
```

- The startswith() method takes maximum of three parameters:
  - ✓ **prefix** - String or tuple of strings to be checked.
  - ✓ **start** (optional) - Beginning position where **prefix** is to be checked within the string.
  - ✓ **end** (optional) - Ending position where **prefix** is to be checked within the string.
- The startswith() method returns a boolean:
  - ✓ It returns *True* if the string starts with the specified prefix.
  - ✓ It returns *False* if the string doesn't start with the specified prefix.

- **For example:**

I.
```
>>> text = 'Python is easy to learn'
>>> res = text.startswith('python')
>>> print(res)
False
```

II.
```
>>> text = 'Python is easy to learn'
>>> res = text.startswith('Python is easy')
>>> print(res)
True
```

III.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('programming',7)
>>> print(res)
True
```

IV.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('programming',8)
>>> print(res)
False
```

V.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('programming is',7,18)
>>> print(res)
False
```

VI.
```
>>> text = 'Python programming is easy'
>>> res = text.startswith('program',7,18)
>>> print(res)
```

True

VII. >>> text = 'Python programming is easy'
>>> res = text.startswith('easy to',7,18)
>>> print(res)
False

5. lower()

- The lower() method returns the lowercased string from the given string. It converts all uppercase characters to lowercase.
- If no uppercase characters exist, it returns the original string.
- For example:

I. >>> string = 'THIS SHOULD BE IN LOWERCASE'
>>> print(string.lower())
this should be in lowercase

II. >>> string = 'th!s shouLd3 Be iN lOwer#case'
>>> print(string.lower())
th!s should3 be in lower#case

SOME OF THE STRING BUILT-IN FUNCTIONS

Syntax	Meaning	Return Value
`str.count(substring[, start[,end]])`	count() method only requires a single parameter for execution. However, it also has two optional parameters: • **substring** - string whose count is to be found. • **start (Optional)** - starting index within the string where search starts. • **end (Optional)** - ending index within the string where search ends.	count() method returns the number of occurrences of the substring in the given string.
`str.index(sub[, start[, end]])`	The index() method takes three parameters: • **sub** - substring to be searched in the string *str*. • **start** and **end**(optional) - substring is searched within **str[start:end]**	• If substring exists inside the string, it returns the lowest index in the string where substring is found. • If substring doesn't exist inside the string, it raises a **ValueError** exception. The index() method is similar to find() method for strings. The only difference is that *find()* method returns -1 if the substring is not found, whereas *index()* throws an exception.

str.isalnum()	-	The isalnum() returns: • **True** if all characters in the string are alphanumeric • **False** if at least one character is not alphanumeric
str.isalpha()	-	The isalpha() returns: • **True** if all characters in the string are alphabets (can be both lowercase and uppercase). • **False** if at least one character is not alphabet.
str.isdecimal()	-	The isdecimal() returns: • **True** if all characters in the string are decimal characters. • **False** if at least one character is not decimal character.
str.isdigit()	-	The isdigit() returns: • **True** if all characters in the string are digits. • **False** if at least one character is not a digit.
str.islower()	-	The islower() method returns: • *True* if all alphabets that exist in the string are lowercase alphabets. • *False* if the string contains at least one uppercase alphabet.
str.swapcase()	-	The swapcase() method returns the string where all uppercase characters are converted to lowercase, and lowercase characters are converted to uppercase.
str.replace(old, new [, count])	The replace() method can take maximum of 3 parameters: • **old** - old substring you want to replace • **new** - new substring which would replace the old substring • **count** (optional) - the number of times you want to replace the *old* substring with the *new* substring If *count* is not specified, replace() method replaces all occurrences of the *old* substring with the *new* substring.	The replace() method returns a copy of the string where *old* substring is replaced with the *new* substring. The original string is unchanged. If the *old* substring is not found, it returns the copy of the original string.

str.join(iterable)	The join() method takes an iterable - objects capable of returning its members one at a time
	Some of the example of iterables are:
	• **Native datatypes** - List, Tuple, String, Dictionary and Set
	• File objects and objects you define with an __iter__() or __getitem()__ method
		The join() method returns a string concatenated with the elements of an iterable.
		If the iterable contains any non-string values, it raises a **TypeError** exception.
		Examples:
		1. >>> numList = ['1','2','3','4']
		>>> separator = ','
		>>> print(separator.join(numList))
		1,2,3,4
		2. >>> numTuple = ('1', '2', '3', '4')
		>>> separator = ','
		>>> print(separator.join(numTuple))
		1,2,3,4
		3. >>> str1 = 'anushka'
		>>> str2 = 'virat'
		>>> print(str1.join(str2))
		vanushkaianushkaranushkaaanushkat
		>>> print(str2.join(str1))
		aviratnviratuviratsvirathviratkvirata

2.9 PARSING STRINGS

- Sometimes you want to find sub string in a string.

For Example:

Consider the following line:

From akhilaa@cmrit.ac.in Mon Jan 22 22:09:30 2018

Suppose we wanted to pull out only the second half of the address (i.e., uct.ac.za), we can do this by using the find method and string slicing.

First, we will find the position of the at-sign in the string. Then we will find the position of the first space *after* the at-sign. And then we will use string slicing to extract the portion of the string which we are looking for.

```
>>> data = 'From akhilaa@cmrit.ac.in Mon Mar 05 18:06:30 2018'
>>> atpos = data.find('@')
>>> print(atpos)
12
>>> spos = data.find(' ', atpos)      → Find the first space starting from atpos
>>> print(spos)
24
>>> host = data[atpos+1:spos]         → Slice the string from atpos+1 to spos
>>> print(host)
cmrit.ac.in
```

Exercise

Take the following Python code that stores a string:

str = 'X-DSPAM-Confidence:0.8475'

Use find and string slicing to extract the portion of the string after the colon character and then use the float function to convert the extracted string into a floating point number.

```
>>> str = 'X-DSPAM-Confidence:0.8475'
>>> pos = str.find(':')
>>> nstr = str[pos+1:]
>>> cstr = float(nstr)
>>> print(cstr)
0.8475
>>> type(cstr)
<class 'float'>
```

2.10 FORMAT OPERATOR

- The *format operator*, % allows us to construct strings, replacing parts of the strings with the data stored in variables. When applied to integers, % is the modulus operator. But when the first operand is a string, % is the format operator.
- The first operand is the *format string*, which contains one or more *format sequences* that specify how the second operand is formatted. The result is a string.

Format sequence	Meaning
%d or %i or %u	Integer/Decimal
%f or %F	Floating-point decimal format (up to 6 decimals)
%g or %G	Floating-point decimal format (up to 4 decimals)
%c	Single character
%s	Strings
%o	Octal
%x or %X	Hexadecimal
%e or %E	Floating-point exponential format

- For example:

I.
```
>>> age = 27
>>> print('The age is %d' %age)
The age is 27
```

II.
```
>>> pi = 3.1412
>>> print('The value of pi is %g' %pi)
The value of pi is 3.1412
>>> print('The value of pi is %f' %pi)
The value of pi is 3.141200
```

III.
```
>>> name = 'Akhilaa'
>>> print('My name is %s' %name)
```

```
My name is Akhilaa
```

IV.
```
>>> print('%x'%12)
c
```

- If there is more than one format sequence in the string, the second argument has to be a tuple. Each format sequence is matched with an element of the tuple, in order.
- The number of elements in the tuple must match the number of format sequences in the string. The types of the elements also must match the format sequences.

For example:

I.
```
>>> years = 27.3
>>> n = 250
>>> string = 'movies'
>>> print('In %g years I saw %d %s' %(years, n, string))
In 27.3 years I saw 250 movies
```

II.
```
>>> years = 27.3
>>> n = 250
>>> string = 'movies'
>>> print('In %f years I saw %d %s' %(years, n))
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: not enough arguments for format string
```

2.11 EXERCISES

1. Write a Python program to calculate the length of a string.
2. Write a python program to find the maximum character in a given string.
3. Write a python program to find the minimum character in a given string.
4. Write a Python program to check if a string is palindrome or not.
5. Write a Python program to count the number of characters (character frequency) in a string.
6. Write a Python program to count the occurrences of each word in a given sentence.
7. Write a Python program to sort a string lexicographically.
8. Write a Python program to count occurrences of a substring in a string.
9. Write a Python program to reverse a string.
10. Write a Python program to reverse words in a string.
11. Write a python program to count repeated characters in a string.
12. Write a Python program to count and display the vowels of a given text.

2.12 GLOSSARY

counter	A variable used to count something, usually initialized to zero and then incremented.
empty string	A string with no characters and length 0, represented by two quotation marks.
format operator	An operator, %, that takes a format string and a tuple and generates a string that includes the elements of the tuple formatted as specified by the format string.

format sequence	A sequence of characters in a format string, like %d, that specifies how a value should be formatted.
format string	A string, used with the format operator, that contains format sequences.
flag	A boolean variable used to indicate whether a condition is true or false.
invocation	A statement that calls a method.
immutable	The property of a sequence whose items cannot be assigned.
index	An integer value used to select an item in a sequence, such as a character in a string.
item	One of the values in a sequence.
method	A function that is associated with an object and called using dot notation.
object	Something a variable can refer to. For now, you can use “object” and “value” interchangeably.
search	A pattern of traversal that stops when it finds what it is looking for.
sequence	An ordered set; that is, a set of values where each value is identified by an integer index.
slice	A part of a string specified by a range of indices.
traverse	To iterate through the items in a sequence, performing a similar operation on each.

3 FILES

- Once the power is turned off, anything stored in either the CPU or main memory is erased. Secondary memory is not erased when the power is turned off.
- input() and print() functions uses keyboard as medium. It works fine as long as the data is small.
- Many real-time problems involve large volumes of data, in such situations the above input/output functions have two major issues:
  - Becomes cumbersome and time consuming to handle large volumes of data.
  - The entire data is lost when the program is terminated.
- Hence, a more flexible approach is required where data can be stored and used especially on disks and read whenever necessary, without destroying the data.
- Also it is easy to move data from one computer to another through files.
- **File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).**
- Since, random access memory (RAM) is volatile which loses its data when computer is turned off, we use files for future use of the data.

3.1 OPENING FILES

- When we want to read or write a file (say on your hard drive), we must first *open* the file.
- Opening the file communicates with your operating system, which knows where the data for each file is stored.
- When you open a file, you are asking the operating system to find the file by name and make sure the file exists.
- The files can be opened in the following modes:

Mode	Description
r	Opens a file for reading only. (It's a default mode.)
w	Opens a file for writing. (If a file doesn't exist already, then it creates a new file, otherwise, it's truncate a file.)
x	Opens a file for exclusive creation. (Operation fails if a file does not exist in the location.)
a	Opens a file for appending at the end of the file without truncating it. (Creates a new file if it does not exist in the location.)
t	Opens a file in text mode. (It's a default mode.)
b	Opens a file in binary mode.
+	Opens a file for updating (reading and writing.)

- **For example**
  - Create a file sample.txt (File→ New File→ sample.txt)

- We open a file sample.txt, which should be stored in the same folder that you are in when you start Python. By default it will open in read mode.

```
>>> fhand = open('sample.txt')
>>> print(fhand)
<_io.TextIOWrapper name='sample.txt' mode='r' encoding='cp1252'>
```

Note: Refer https://docs.python.org/3/library/io.html for more details regarding io.TextIOWrapper

- If the open is successful, the operating system returns us a *file handle*. The file handle is not the actual data contained in the file, but instead it is a "handle" that we can use to read the data. You are given a handle if the requested file exists and you have the proper permissions to read the file.

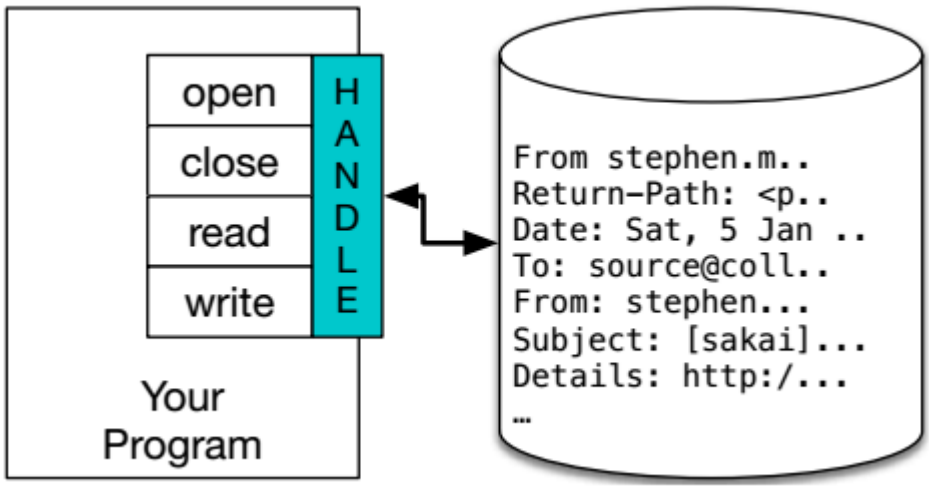


Fig: A file handle

- If the file does not exist, open will fail with a traceback and you will not get a handle to access the contents of the file:

```
>>> fhand = open('test.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

3.2 TEXT FILES AND LINES

- **A text file can be thought of as a sequence of lines.**
- To break the file into lines, there is a special character that represents the "end of the line" called the *newline* character.
- In Python, we represent the *newline* character as a backslash-n in string constants. Even though this looks like two characters, it is actually a single character. When we look at the variable by entering "stuff" in the interpreter, it shows us the \n in the string, but when we use print to show the string, we see the string broken into two lines by the newline character.

- **For example:**

```
>>> mess = 'hello\nworld'
>>> mess
'hello\nworld'
>>> print(mess)
hello
world
>>> len(mess)
11
```

- You can also see that the length of the string X\nY is three characters because the newline character is a single character.
- So when we look at the lines in a file, we need to imagine that there is a special invisible character called the newline at the end of each line that marks the end of the line.
- So the newline character separates the characters in the file into lines.

3.3 READING FILES

- **Create a file called 'poem.txt'. The contents of the file is as below:**

In poem.txt:

When to the session of sweet silent thought
I summon up remembrance of things past,
I sigh the lack of many a thing I sought,
And with old woes new wail my dear time's waste:
Then can I drown an eye, unused to flow,
For precious friends hid in death's dateless night,
And weep afresh love's long since cancelled woe,
And moan the expense of many a vanish'd sight:
Then can I grieve at grievances foregone,
And heavily from woe to woe tell o'er
The sad account of fore-bemoaned moan,
Which I new pay as if not paid before.
But if the while I think on thee, dear friend,
All losses are restored and sorrows end.

- **Create another file called 'file.py'. Note 'poem.txt' is in the same folder as file.py. In this file we will write a small python code to count the number of lines in poem.txt.**

```
# To count the number of lines in file poem.txt

fhand = open('poem.txt')
count = 0
for line in fhand:
```

```
    count = count + 1
print('The total Line count is: ', count)
```

Output:

```
The total Line count is:  14
```

- The rough translation of the above for loop is, "for each line in the file represented by the file handle, add one to the count variable."
- The reason that the open function does not read the entire file is that the file might be quite large with many gigabytes of data. The open statement takes the same amount of time regardless of the size of the file. The for loop actually causes the data to be read from the file.
- When the file is read using a for loop in this manner, Python takes care of splitting the data in the file into separate lines using the newline character. Python reads each line through the newline and includes the newline as the last character in the line variable for each iteration of the for loop.
- Because the for loop reads the data one line at a time, it can efficiently read and count the lines in very large files without running out of main memory to store the data. The above program can count the lines in any size file using very little memory since each line is read, counted, and then discarded.
- If you know the file is relatively small compared to the size of your main memory, you can read the whole file into one string using the read method on the file handle.

```
>>> fhand = open('poem.txt')
>>> string = fhand.read()
>>> print(len(string))
611
>>> print(string[:30])
When to the session of sweet s
```

- Remember that this form of the open function should only be used if the file data will fit comfortably in the main memory of your computer. If the file is too large to fit in main memory, you should write your program to read the file in chunks using a for or while loop.

3.4 SEARCHING THROUGH A FILE

- If we wanted to read a file and only print out lines which started with the prefix "And", we could use the string method *startswith* to select only those lines with the desired prefix.
- **For example:**

```
# To search and print all the lines starting with 'And' in poem.txt.

fhand = open('poem.txt')

for line in fhand:
  if line.startswith('And') :
    print(line)
```

Output:

And with old woes new wail my dear time's waste:

And weep afresh love's long since cancelled woe,

And moan the expense of many a vanish'd sight:

And heavily from woe to woe tell o'er

- Each of the lines ends with a newline, so the print statement prints the string in the variable *line* which includes a newline and then print adds *another* newline, resulting in the double spacing effect we see.
- We could use line slicing to print all but the last character, but a simpler approach is to use the *rstrip* method which strips whitespace from the right side of a string as follows:

```
# To search and print all the lines starting with 'And' in poem.txt without double spacing.

fhand = open('poem.txt')

for line in fhand:
  line = line.rstrip()
  if line.startswith('And') :
    print(line)
```

Output:

And with old woes new wail my dear time's waste:
And weep afresh love's long since cancelled woe,
And moan the expense of many a vanish'd sight:
And heavily from woe to woe tell o'er

- We can structure the loop to follow the pattern of skipping uninteresting lines as follows:

```
fhand = open('poem.txt')

for line in fhand:
  line = line.rstrip()
  if not line.startswith('And') :    #Skips uninteresting lines.
    continue
  print(line)
```

- We can use the find string method to simulate a text editor search that finds lines where the search string is anywhere in the line. Since find looks for an occurrence of a string within another string and either returns the position of the string or -1 if the string was not found, we can write the following loop to show lines which contain the string “of many”:

```
# To search and print all the lines containing string 'of many'.

fhand = open('poem.txt')

for line in fhand:
   line = line.rstrip()
   if line.find('of many') == -1 :
      continue
   print(line)
```

Output:

```
I sigh the lack of many a thing I sought,
And moan the expense of many a vanish'd sight:
```

3.5 LETTING THE USER CHOOSE THE FILENAME

- We read the file name from the user and place it in a variable named fname and open that file. Now we can run the program repeatedly on different files.
- **For example:**

```
#To read filename from the user.

fname = input('Enter the filename: ')
fhand = open(fname)
count = 0
for line in fhand:
   if line.startswith('Then'):
      count = count + 1
print('The number of The in', fname, 'are: ', count)
```

Output:

I. Enter the filename: poem.txt
   The number of The in poem.txt are: 2

II. Enter the filename: sample.txt
   The number of The in sample.txt are: 0

3.6 USING TRY, EXCEPT AND OPEN

- When you try to open a file that doesn't exist, it throws an error as below:

```
Enter the filename: samfile.txt
Traceback (most recent call last):
  File "C:\Users\akhil\AppData\Local\Programs\Python\Python36-32\search.py", line 52, in <module>
    fhand = open(fname)
FileNotFoundError: [Errno 2] No such file or directory: 'samfile.txt'
```

- When we see the flaw in the program, we can fix it using the try/except structure as below:

```
#To demonstrate file program using try and except.

fname = input('Enter the filename: ')

try:
   fhand = open(fname)
except:
   print('Error in opening file',fname)
   exit()

count = 0
for line in fhand:
   if line.startswith('Then'):
      count = count + 1
print('The number of The in', fname, 'are: ', count)
```

Output:

I. Enter the filename: haha.txt
   Error in opening file haha.txt
II. Enter the filename: poem.txt
   The number of The in poem.txt are: 2

Note: The exit function terminates the program.

3.7 WRITING FILES

- To write a file, you have to open it with mode "w" as a second parameter:

```
>>> fout = open('wfile.txt','w')
>>> print(fout)
<_io.TextIOWrapper name='wfile.txt' mode='w' encoding='cp1252'>
```

- **If the file already exists, opening it in write mode clears out the old data and starts fresh. If the file doesn't exist, a new one is created.**
- The write method of the file handle object puts data into the file, returning the number of characters written. The default write mode is text for writing (and reading) strings.

```
>>> mess = 'Hi. Good Morning All!!!'
>>> fout.write(mess)
23
```

- We must make sure to manage the ends of lines as we write to the file by explicitly inserting the newline character when we want to end a line. The print statement automatically appends a newline, but the write method does not add the newline automatically.

3.8 CLOSING A FILE

- After opening a file one should always close the opened file. We use method *close()* for this.
- **For example:**

```
>>> fhand = open('sample.txt')
>>> fhand.readlines()
['Welcome to Python Application Programming.\n', 'This is Akhilaa your course instructor.\n']
>>> fhand.close()
```

- Always make sure you *explicitly* close each opened file, once its job is done and you have no reason to keep it open. Because - There is an upper limit to the number of files a program can open. If you exceed that limit, there is no reliable way of recovery, so the program could crash. – Each open file consumes some main-memory for the data-structures associated with it, like file descriptor/handle or file locks etc. So you could essentially end-up wasting lots of memory if you have more files open that are not useful or usable. - Open files always stand a chance of corruption and data loss.

3.9 READING AND WRITING FILES IN PYTHON

- **Python program to read the entire contents of a file.**

In file rfile.txt:

```
10
20
```

In fsample.txt:

```
fr = open('rfile.txt')

a = fr.read()
print(a)
print(type(a))
fr.close()
```

Output:

```
10
20

<class 'str'>
```

- **Python program to read few bytes from a file.**

In file rfile.txt:

```
Hi Everyone.
My name is Akhilaa.
I am working in CMRIT as Assistant Professor.
```

In fsample.txt:

```
fr = open('rfile.txt')

str = fr.read(40)
print(str)

fr.close()
```

Output:

```
Hi Everyone.
My name is Akhilaa.
I am wo
```

- Python program to read line by line from a file.

1. In rfile.txt:

```
10
20
```

In fsample.txt:

```
fr = open('rfile.txt')

a = fr.readline()
print('The type of a is ',type(a))

b = fr.readline()
print('The type of b is ',type(b))

fr.close()
```

Output:

```
The type of a is <class 'str'>
The type of b is <class 'str'>
```

2. In rfile.txt:

```
10
20
```

In fsample.txt:

```
fr = open('rfile.txt')

a = int(fr.readline())
print('The value of a is',a)
print('The type of a is ',type(a))
```

```
b = int(fr.readline())
print('The value of b is',b)
print('The type of b is ',type(b))

print('The sum is ',a+b)

fr.close()
```

Output:

```
The value of a is 10
The type of a is <class 'int'>
The value of b is 20
The type of b is <class 'int'>
The sum is 30
```

- **Python program to read two numbers from a file and compute the sum and append it to the same file.**

In rfile.txt:

```
10
20
```

In fsample.txt:

```
fr = open('rfile.txt')

a = int(fr.readline())
b = int(fr.readline())

sum = str(a+b)

fw = open('wfile.txt', 'a')
fw.write('The sum is ')
fw.write(sum)

fr.close()
fw.close()
```

Output:

```
The contents of wfile.txt:
The sum is 30
```

- **Python program to read two numbers from a file and compute the sum and append it to the same file.**

In rfile.txt:

```
10
20
```

In fsample.txt:

```
fr = open('rfile.txt')

a = int(fr.readline())
b = int(fr.readline())

fr.close()

sum = str(a+b)

fa = open('rfile.txt', 'a')
fa.write('The sum is ')
fa.write(sum)

fa.close()
```

Output:

```
The contents of rfile.txt:
10
20
The sum is 30
```

3.10 EXERCISES

1. Write a program to read through a file and print the contents of the file (line by line) all in upper case.

```
filename = input('Enter the file name: ')
fr = open(filename)

for line in fr:
   print(line.upper().rstrip())
```

2. Write a python program to copy contents of one file to another file.

```
#Python program to copy contents of one file to another.

fr = open('poem.txt')
fw = open('cfile.txt','w')

for line in fr:
   fw.write(line)
```

```
fr.close()
fw.close()
```

OR

```
#Python program to copy contents of one file to another.

fr = open('poem.txt')
fw = open('cfile.txt','w')

line = fr.read()
fw.write(line)

fr.close()
fw.close()
```

3. Write a python program to append a line to a file.

```
#Python program to append a line to a file.

fa = open('poem.txt','a')

line = 'This is written by me'
fa.write(line)

fa.close()
```

4. Write a python program to find sum and average of 5 numbers and write the output to a file.
5. Write a python program to count the frequency of words in a file.

```
#Python program to find the frequency of a given word.

fhand = open('poem.txt')

word = input('Enter the word: ')

count=0
for line in fhand:
   line = line.rstrip()
   if line.find(word) == -1 :
      continue
   count = count + 1

print('The frequency of',word,'is',count)
```

Output:

1. **Enter the word: many**

The frequency of many is 2

2. **Enter the word: pay**
   The frequency of pay is 1
3. **Enter the word: cream**
   The frequency of cream is 0

3.11 GLOSSARY

catch	To prevent an exception from terminating a program using the try and except statements.
newline	A special character used in files and strings to indicate the end of a line.
Pythonic	A technique that works elegantly in Python. "Using try and except is the Pythonic way to recover from missing files".
Quality Assurance	A person or team focused on insuring the overall quality of a software product. QA is often involved in testing a product and identifying problems before the product is released.
text file	A sequence of characters stored in permanent storage like a hard drive.

QUESTIONS

1. What is a loop? Explain the syntax of while and for statement with example.
2. Explain break and continue statements with example.

   OR

   Explain jump statements/loop control statements with example.
3. What is string? Explain how to create and access strings.
4. Explain max(), min() and len() functions with examples.
5. Explain string slicing with examples.
6. Are strings immutable or mutable? Explain.
7. Explain the various operations that can be performed on a string with example. (+, *, [], [:], in, not in, Relational operators)
8. Explain any 5 built-in string functions with examples.
9. Explain with an example how string parsing is done.
10. Explain the format operator(%) with example.
11. What is a file? Explain different modes of opening a file.
12. Explain the following functions with example:
    a. open()
    b. read()
    c. write()
    d. readlines()
    e. close()