

Module2

- 1. Iteration**
- 2. Strings**
- 3. Files**

Iteration

- Repeating the execution of statements for a given number of times is called ***iteration***. The variable which changes or updates its values each time through a loop is called ***iteration variable***.
- A **loop** is a sequence of instructions that is continually repeated until a certain condition is reached.
- Updating a variable by adding 1 is called an ***increment***;
 - $x = x + 1$
- Subtracting *a variable by 1* is called a ***decrement***.
 - $x = x - 1$

Types of Looping Statements

1. While Loop
2. For Loop

1. While Loop

- Loops are used in programming to repeat a specific block of code.
- The while loop in Python is used to iterate over a block of code as long as the test expression (condition) is true.
- We generally use this loop when we don't know beforehand, the number of times to iterate.

- Example

Syntax of while Loop in Python

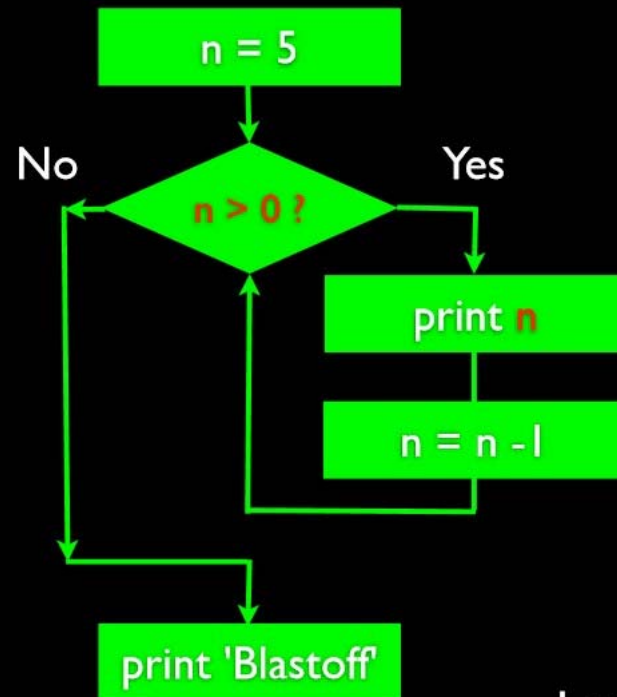
Syntax :

initialization

```
while test_expression:  
    # initialize sum and  
    # Body of while  
    counter sum = 0  
    statement_1  
    i = 1 statement_2  
    ....
```

Next Stmtnt

- while i <= n:
 • sum = sum + i
 • i = i + 1 # update counter



Repeated Steps

Program:

```
n = 5
while n > 0 :
    print n
    n = n - 1
print 'Blastoff!'
print n
```

Output:

5
4
3
2
1
Blastoff!
0

Loops (repeated steps) have **iteration variables** that change each time through a loop. Often these **iteration variables** go through a sequence of numbers.

The flow of execution for a while statement:

1. Evaluate the condition, yielding True or False.
2. If the condition is false, exit the while statement and continue execution at the next statement.
3. If the condition is true, execute the body and then go back to step 1.

This type of flow is called a ***loop because the third step loops back around to the top.*** We call each time we execute the body of the loop an *iteration*.

```
n = 10
while True:
    print(n, end=' ')
    n = n - 1
print('Done!')
```

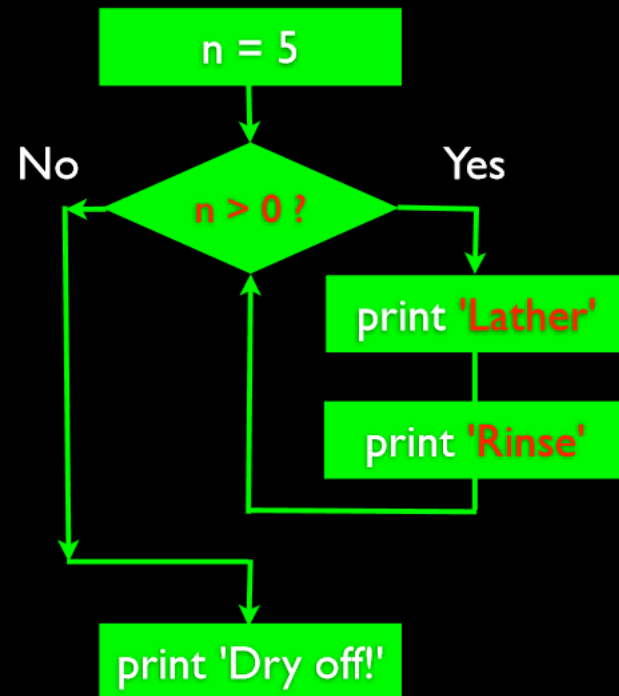
Infinite loops
An infinite loop (sometimes called an endless **loop**) is a piece of coding that lacks a functional exit so that it repeats indefinitely.

An Infinite Loop

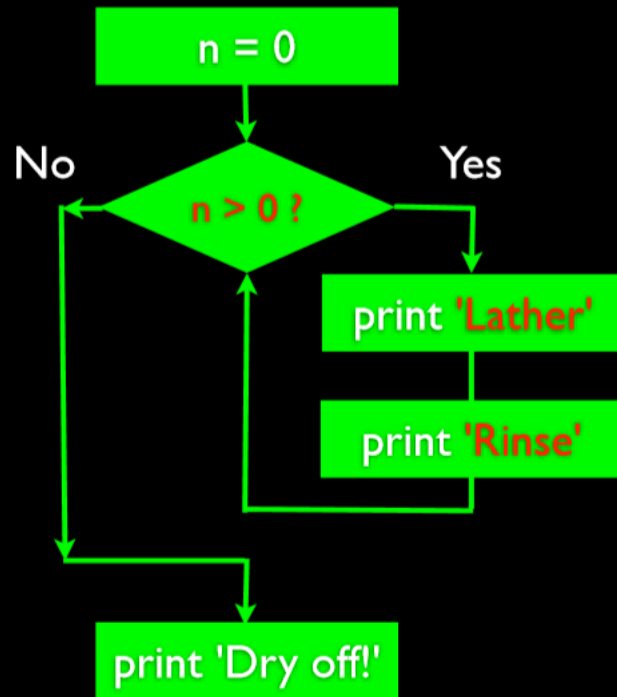
```
n = 5
while n > 0 :
    print 'Lather'
    print 'Rinse'

print 'Dry off!'
```

What is wrong with this loop?



Another Loop



```
n = 0
while n > 0 :
    print 'Lather'
    print 'Rinse'
```


```
print 'Dry off!'
```

What does this loop do?

Breaking out of a Loop

- The break statement ends the current loop and jumps to the statement immediately following the loop
- It is like a loop test that can happen anywhere in the body of the loop

```
while True:
    line = raw_input('> ')
    if line == 'done':
        break
    print line
print 'Done!'
```



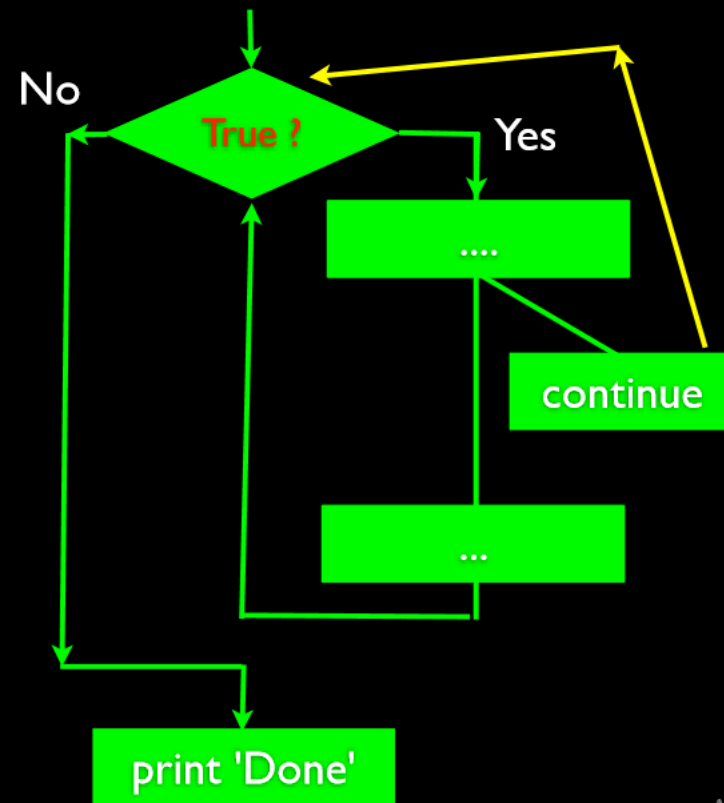
Finishing a Iteration with

- The continue statement ends the current iteration and jumps to the top of the loop and starts the next iteration

continue

```
while True:
    line = raw_input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print line
print 'Done!'
```

```
while True:
    line = raw_input('> ')
    if line[0] == '#':
        continue
    if line == 'done':
        break
    print line
print 'Done!'
```



Exercise 1

- Print the Fibonacci Sequence 0 1 1 2 3 -----

```
# Program to display the Fibonacci sequence up to n-th term where

nterms = 10
n1 = 0
n2 = 1
count = 0

if nterms <= 0:
    print("Please enter a positive integer")
elif nterms == 1:
    print("Fibonacci sequence upto",nterms,":")
    print(n1)
else:
    print("Fibonacci sequence upto",nterms,":")
    while count < nterms:
        print(n1,end=' , ')
        nth = n1 + n2
        # update values
        n1 = n2
        n2 = nth
        count += 1
```

Definite and Indefinite Loop

- Conceptually, we distinguish two types of loops, which differ in the way in which the number of *iterations* (i.e., repetitions of the body of the loop) is determined:
- In **definite loops**, the number of iterations is known before we start the execution of the body of the loop
- **Example: repeat for 10 times printing out a *.**
- In **indefinite loops**, the number of iterations is not known before we start to execute the body of the loop, but depends on when a certain condition becomes true (and this depends on what happens in the body of the loop)
- **Example: while the user does not decide it is time to stop, print out a * and ask the user whether he wants to stop.**

1. Indefinite Loops

- While loops are called "indefinite loops" because they keep going until a logical condition becomes False

2. Definite Loops

- Quite often we have a list of items of the lines in a file - effectively a finite set of things
- We can write a loop to run the loop once for each of the items in a set using the Python for construct
- These loops are called "*definite loops*" because they execute an exact number of times
- We say that "*definite loops iterate through the members of a set*"

A Simple Definite Loop

```
for i in [5, 4, 3, 2, 1]:  
    print i
```

```
print 'Blastoff!'
```

5
4
3
2
1
Blastoff!

A Simple Definite Loop

```
friends = ['Joseph', 'Glenn', 'Sally']
```

```
for friend in friends :
```

```
    print 'Happy New Year:', friend
```

```
print 'Done!'
```

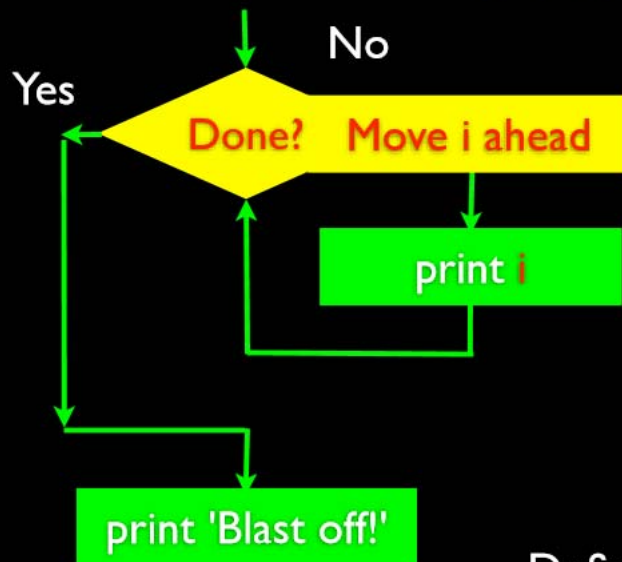
Happy New Year: Joseph

Happy New Year: Glenn

Happy New Year: Sally

Done!

A Simple Definite Loop



```
for i in [5, 4, 3, 2, 1]:  
    print i
```

```
print 'Blastoff!'
```

5
4
3
2
1
Blastoff!

Definite loops (for loops) have explicit **iteration variables** that change each time through a loop. These **iteration variables** move through the sequence or set.

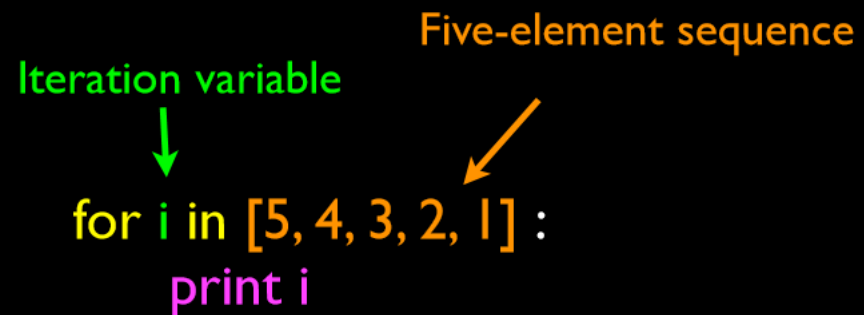
Looking at **in**...

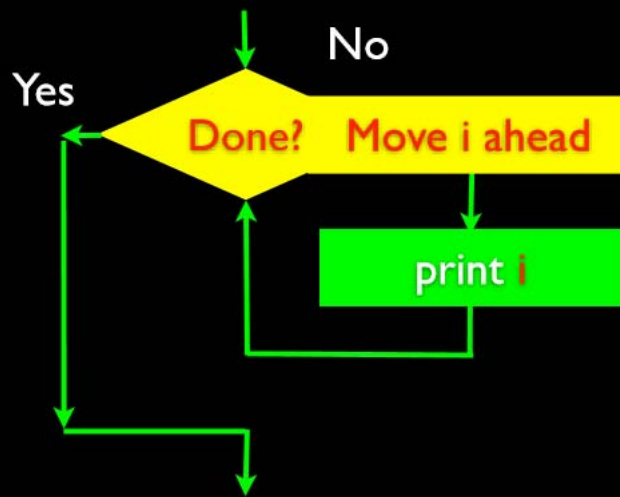
- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**

Iteration variable

Five-element sequence

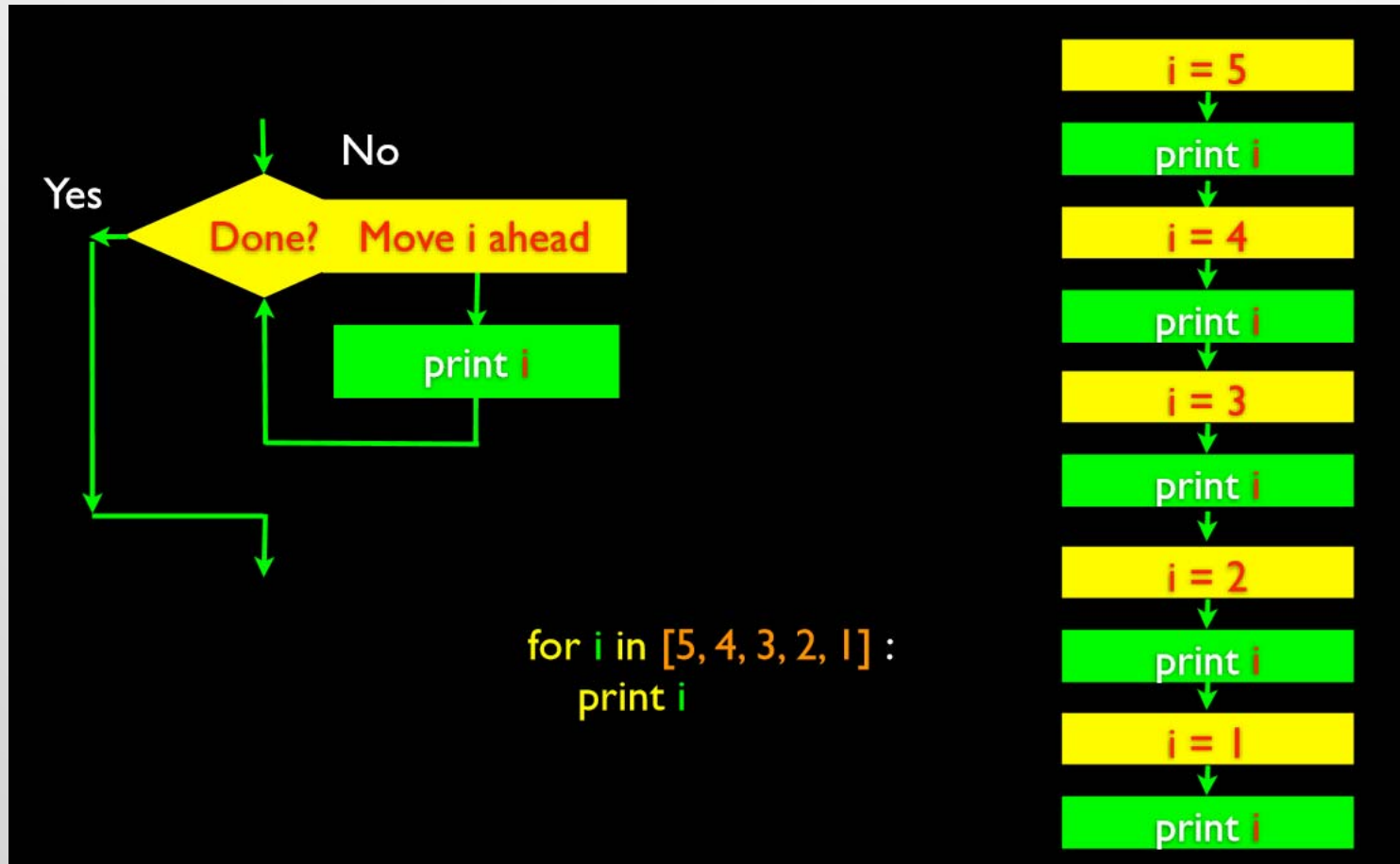
```
for i in [5, 4, 3, 2, 1]:  
    print i
```





```
for i in [5, 4, 3, 2, 1]:  
    print i
```

- The **iteration variable** “iterates” through the **sequence** (ordered set)
- The **block (body)** of code is executed once for each value **in** the **sequence**
- The **iteration variable** moves through all of the values **in** the **sequence**



Example : Loop Pattern1

```
count = 0
```

```
for itervar in [3, 41, 12, 9, 74, 15]:
```

```
    count = count + 1
```

```
print('Count: ', count)
```

Example : Loop Pattern2

```
total = 0
```

```
for itervar in [3, 41, 12, 9, 74, 15]:
```

```
    total = total + itervar
```

```
print('Total: ', total)
```

Maximum loops

To find the largest value in a list or sequence, we construct the following loop:

```
largest = None
```

```
print('Before:', largest)
```

```
for itervar in [3, 41, 12, 9, 74, 15]:
```

```
    if largest is None or itervar > largest :
```

```
        largest = itervar
```

```
    print('Loop:', itervar, largest)
```

```
print('Largest:', largest)
```

To compute the smallest
number,

```
smallest = None
```

```
print('Before:', smallest)
```

```
for itervar in [3, 41, 12, 9, 74, 15]:
```

```
    if smallest is None or itervar < smallest:
```

```
        smallest = itervar
```

```
    print('Loop:', itervar, smallest)
```

```
print('Smallest:', smallest)
```

simple version of the Python built-in min() function

```
def min(values):  
    smallest = None  
    for value in values:  
        if smallest is None or value < smallest:  
            smallest = value  
    return smallest
```

Exercise on For Loop

- To check whether a given number is Prime.

```
# Python program to check if the input number is prime or not
```

```
num = 5
```

```
if num > 1:
```

```
    for i in range(2, num):
```

```
        if (num % i) == 0:
```

```
            print(num, "is not a prime number")
```

```
            break
```

```
    else:
```

```
        print(num, "is a prime number")
```

```
else:
```

```
    print(num, "is not a prime number")
```

```
5 is a prime number
```

Exercise 2

- To print a factorial of a given number


```
num = 7
# uncomment to take input from the user
#num = int(input("Enter a number: "))

factorial = 1

# check if the number is negative, positive or zero
if num < 0:
    print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
    print("The factorial of 0 is 1")
else:
    for i in range(1, num + 1):
        factorial = factorial*i
    print("The factorial of", num, "is", factorial)
```

Exercises

1. Write a program which repeatedly reads numbers until the user enters “done”. Once “done” is entered, print out the total, count, and average of the numbers. If the user enters anything other than a number, detect their mistake using try and except and print an error message and skip to the next number.
2. Write another program that prompts for a list of numbers as above and at the end print out both the maximum and minimum of the numbers instead of the average.

Strings

- **A string is a sequence of characters.** We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes.
- Creating strings is as simple as assigning a value to a variable

Example :

var1 = 'Hello World!'

var2 = "Python Programming"

fruit = 'apple'

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.

Example : Creating a string

all of the following are equivalent

```
my_string = 'Hello'
print(my_string)
my_string = "Hello"
print(my_string)
my_string = """Hello"""
print(my_string)
```

triple quotes string can extend multiple lines

```
my_string = """Hello, welcome to
                the world of Python"""
print(my_string)
```

Accessing the characters in a string

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring. You can access the characters one at a time with the bracket operator:
- In Python, the index is an offset from the beginning of the string, and the offset of the first letter is zero.
- You can use any expression, including variables and operators, as an index, but the value of the index has to be an integer. Otherwise you get:
- **Example :**
`letter = fruit[1]`
`print(letter)`
`P`
- **Example :**
`letter = fruit[1.5]`
TypeError: string indices must be integers

Example

```
var1 = 'Hello World!'  
var2 = "Python Programming"  
print ("var1[0]: ", var1[0])  
print ("var2[1:5]: ", var2[1:5])
```

Output

```
var1[0]: H  
var2[1:5]: ytho
```

String


```
str = 'programiz'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])

#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2])
```



b	a	n	a	n	a
[0]	[1]	[2]	[3]	[4]	[5]

Figure 6.1: String Indexes

Getting the length of a string using

- len is a built-in function that returns the number of characters in a string:

```
>>> fruit = 'apple'
```

```
>>> len(fruit)
```

```
5
```

- To get the last letter of a string, :

```
>>> length = len(fruit)
```

```
>>> last = fruit[length]
```

```
IndexError: string index out of range
```

```
>>> last = fruit[length-1]
```

```
>>> print(last)
```

```
e
```

- Alternatively, you can use negative indices, which count backward from the end of the string. The expression **fruit[-1]** yields the last letter, **fruit[-2]** yields the second to last, and so on.

Traversal through a string with a loop

- The process of traversal includes select each character in turn, do something to it, and continue until the end.

```
fruit = 'STRAWBERRY'
```

```
index = 0
```

```
while index < len(fruit):
```

```
    letter = fruit[index]
```

```
    print(letter)
```

```
    index = index + 1
```

Example

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

```
for ch in fruit:  
    print(ch)
```

Exercise 1:

- Write a while loop that starts at the last character in the string and works its way backwards to the first character in the string, printing each letter on a separate line, except backwards.

```
fruit="strawberry"  
i=-1  
c=len(fruit)  
while (c>0):  
    print(fruit[i])  
    i=i-1  
    c=c-1
```

y
r
r
e
b
w
a
r
t
s

String slices

- A segment of a string is called a **slice**. Selecting a slice is similar to selecting a character:

```
s = 'Monty Python'
```

```
print(s[0:5])
```

```
print(s[6:12])
```

- **Monty**
- **Python**
- 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**.

String slices

- 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:
 - `fruit = 'banana'`
 - `fruit[:3]` gives `'ban'`
 - `fruit[3:]` gives `'ana'`
- If the first index is greater than or equal to the second the result is an empty string, represented by two quotation marks:
- `fruit = 'banana'`
- `fruit*3:3+` gives `''`
- An empty string contains no characters and has **length 0**, but other than that, it is the same as any other string.

Strings are immutable

- It is tempting to use the operator on the left side of an assignment, with the intention of changing a character in a string.
- For example:
- `>>> greeting = 'Hello, world!'`
- `>>> greeting[0] = 'J'`
- **TypeError:** 'str' object does not support item assignment

Example : Substituting using Concatenation

- `>>> greeting = 'Hello, world!'`
- `>>> new_greeting = 'J' + greeting[1:]`
- `>>> print(new_greeting)` Jello, world!

Looping and counting

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

```
word = 'strawberry'
```

```
count = 0
```

```
for letter in word:
```

```
    if letter == 'r':
```

```
        count = count + 1
```

```
print(count)
```

The in operator

- The word **in** is a boolean operator that takes two strings and returns **True** if the first appears as a substring in the second:
- `>>> 'straw' in 'strawberry'`
- `True`
- `>>> 'apple' in 'strawberry'`
- `False`

String comparison

- To see if two strings are equal:
if word == 'berry':
print('All right, bananas.')
- Other comparison operations are useful for putting words in alphabetical order:
if word < 'berry':
print('Your word,' + word + ', comes before berry.')
elif word > 'berry':
print('Your word,' + word + ', comes after berry.')
else:
print('All right, bananas.')

string methods

- **Strings are an example of Python objects.** An object contains *both data (the actual string itself) and methods*, which are effectively functions that are built into the object and are available to any instance of the object.
- 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.

Listing string methods using `dir()`

```
stuff = 'Hello world'
```

```
type(stuff) <class 'str'>
```

```
dir(stuff)
```

```
['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']
```

string methods

- Methods : 'capitalize', 'count', 'endswith', 'find', 'format', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle', 'isupper', 'join', 'lower', 'lstrip', 'replace', 'rsplit', 'rstrip', 'split', 'startswith', 'strip', 'upper'.
- Example1 :
word = 'strawberry'
new_word = word.upper()
print(new_word)
STRAWBERRY
- Example 2:
word = 'banana'
index = word.find('a')
print(index)
1

Example : The *find* method can find substrings as well as characters:

- `word.find('na')`

2

- `word.find('na', 3)`

4

Python String Operators

Operator	Operation	Description	Example Code
+	Concatenation	Combining two strings into one.	<pre>var1 = 'Python' var2 = 'String' print (var1+var2) O/P: PythonString</pre>
*	Repetition	Creates new String by repeating the String given number of times.	<pre>var1 = 'Python' print (var1*3) O/P: PythonPythonPython</pre>
[]	Slicing	Prints the character at given index.	<pre>var1 = 'Python' print (var1[2]) O/P : t</pre>
[:]	Range Slicing	Prints the characters present at the given range .	<pre>var1 = 'Python' print (var1[2:5]) O/P : tho</pre>

Python String Operators

Operator	Operation	Description	Example Code
in	Membership	Returns 'True' value if character is present in the given String.	var1 = 'Python' print ('n' in var1) O/P: True
not in	Membership	Returns 'True' value if character is not present in given String.	var1 = 'Python' print ('N' not in var1) O/P: True
for	Iterating	Using for we can iterate through all the characters of the String.	var1 = 'Python' for var in var1: print (var) O/P:
r/R	Raw String	Used to ignore the actual meaning of Escape characters inside a string. For this we add 'r' or 'R' in front of the String.	print (r'\n') O/P: \n print (R'\n') O/P: \n

Built-in String Functions in Python

Function Name	Description	Example Code
capitalize()	Returns the String with first character capitalized and rest of the characters in lower case.	var = 'PYTHON' print (var.capitalize()) output: Python
lower()	Converts all the characters of the String to lowercase.	var = 'TechBeamers' print (var.lower()) output: techbeamers
upper()	Converts all the characters of the String to uppercase.	var = 'TechBeamers' print (var.upper()) output: TECHBEAMERS
swapcase()	Swaps the case of every character in the String means that lowercase characters are changed to uppercase and vice-versa.	var = 'TechBeamers' print (var.swapcase()) output: tECHbEAMERS

Built-in String Functions in Python

Function Name	Description	Example Code
<code>title()</code>	Returns the 'title cased' version of String which means that all words start with uppercase and rest of the characters in the words are in lowercase.	<pre>var = 'welcome to Python programming' print (var.title())</pre> O/P: <i>Welcome To Python Programming</i>
<code>count(str[,beg [,end]])</code>	Returns the number of times substring 'str' occurs in range [beg, end] if beg and end index are given. If it is not given then substring is searched in whole String. Search is case-sensitive.	<pre>var='TechBeamers' str='e' print (var.count(str))</pre> O/P:3 <pre>var1='Eagle Eyes' print (var1.count('e'))</pre> O/P: 2 <pre>var2='Eagle Eyes' print (var2.count('E',0,5))</pre> O/P:1

strip method

Using strip method one can remove white space (*spaces, tabs, or newlines*) from the beginning and end of a string using the strip method:

- `line = ' YES I CAN DO '`
- `line.strip()`
- `'YES I CAN DO'`



```
line = 'Have a nice day'
```

startswith

Example1:

```
line.startswith('Have')
```

True

Example2:

```
line.startswith('h')
```

False

lower

- `line = 'Have a nice day'`
- `line.lower()` gives `'have a nice day'`
- `line.lower().startswith('h')`
- `True`

Example

```
word = 'banana'  
index = word.find('a')  
print(index)
```

1

Parsing strings using *find*

```
data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
```

```
atpos = data.find('@')
```

```
print(atpos)
```

```
21
```

```
sppos = data.find(' ',pos)
```

```
print(sppos)
```

```
31
```

```
host = data[atpos+1:sppos]
```

```
print(host)
```

```
uct.ac.za
```

Example

```
data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'  
atpos = data.find('@')  
print(atpos)  
sppos = data.find(' ', atpos)  
print(sppos)
```

21


31

Example1 – String methods

```
var = 'welcome to Python programming'  
print(var.count('w'))  
print(var.upper())  
print(var.lower())  
print(var.title())  
print(var.swapcase())  
print(var.capitalize())
```

1

```
WELCOME TO PYTHON PROGRAMMING  
welcome to python programming  
Welcome To Python Programming  
WELCOME TO pYTHON PROGRAMMING  
Welcome to python programming
```



```
var = 'welcome to Python programming'
var1= 'Python'
var2= '22'
var3='1075tgs'
line = '          YES I CAN DO AND I WILL DO          '
print(var.islower())
print(var.isupper())
print(var2.isdigit())
print(var1.isalpha())
print(var3.isalnum())
```

```
False
False
True
True
True
```

Example2 : String Methods

```
line = '      YES I CAN DO AND I WILL DO '
line2='YES I CAN DO IT'
print(line.strip())
print(line.lstrip())
print(line.rstrip())
print(line2.startswith('YES'))
```

YES I CAN DO AND I WILL DO

YES I CAN DO AND I WILL DO

YES I CAN DO AND I WILL DO

True

Format (%/format) operators

- There are two format operators :

- 1. % and 2. ***format***

- ***Syntax of % :***

`print('%s1 %s2 ---' %(arg1,arg2----argn))`

Here

- **s1,s2** are conversion specifiers like d,f,g ,s
 - **arg1,arg2---** are variables /values

- ***Syntax of format :***

`print('{0} {1}....' .format(arg1,agr2,--argn))`

Here

- **0,1** are position specifiers |
 - **arg1,arg2---** are variables /values



1. Usage of % operator

Example1

```
camels = 42  
'%d' % camels
```

```
'42'
```


Example2

```
camels = 42  
print('I have spotted %d camels.' % camels)
```

I have spotted 42 camels.



```
camels = 42
print('In %d years I have spotted %g %s.' % (3, 0.1, 'camels'))
```

In 3 years I have spotted 0.1 camels.

```
print('%s %s' % ('one', 'two'))  
print('%d %d' % (1, 2))  
x= 20  
y= 3.15  
z= "abcde"  
print('The values of x=%d,y=%fz=%s'% (x,y,z))
```


one two

1 2

The values of x=20,y=3.150000z=abcde

Warning

- 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:
- `>>> '%d %d %d' % (1, 2)`
- **TypeError:** not enough arguments for format string
- `>>> '%d' % 'dollars'`
- **TypeError:** %d format: a number is required, not str



2.Usage of *format*

```
print('{} {}'.format('one', 'two'))
print("First argument: {0}, second one: {1}".format(47,11))
print("Second argument: {1}, first one: {0}".format(47,11))
print("First argument: {}, second one: {}".format(47,11))
x= 20
y= 3.15
z= "abcde"
print('The values of x={0},y={1}z={2}'.format(x,y,z))
```

```
one two
First argument: 47, second one: 11
Second argument: 11, first one: 47
First argument: 47, second one: 11
The values of x=20,y=3.15z=abcde
```

Exercise

- Write a program to find the simple interest making use of
 - 1. % in print statement and
 - 2. format in print statement

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.



FILES

What are Files ?

- A file is some information or data which stays in the computer storage devices.
- Different kinds of file : Music files, Video files, text files, Multimedia Files , etc.
- Generally files are divided into two categories: text file and binary file.
- Text files are simple text where as the binary files contain binary data which is only readable by computer.

Text Files and Lines

- A text file can be thought of as a sequence of lines, much like a Python string can be thought of as a sequence of characters.
- 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.

Ex: `stuff = 'Hello \n Word'`

Ananth G S

Files are stored in Secondary Memory

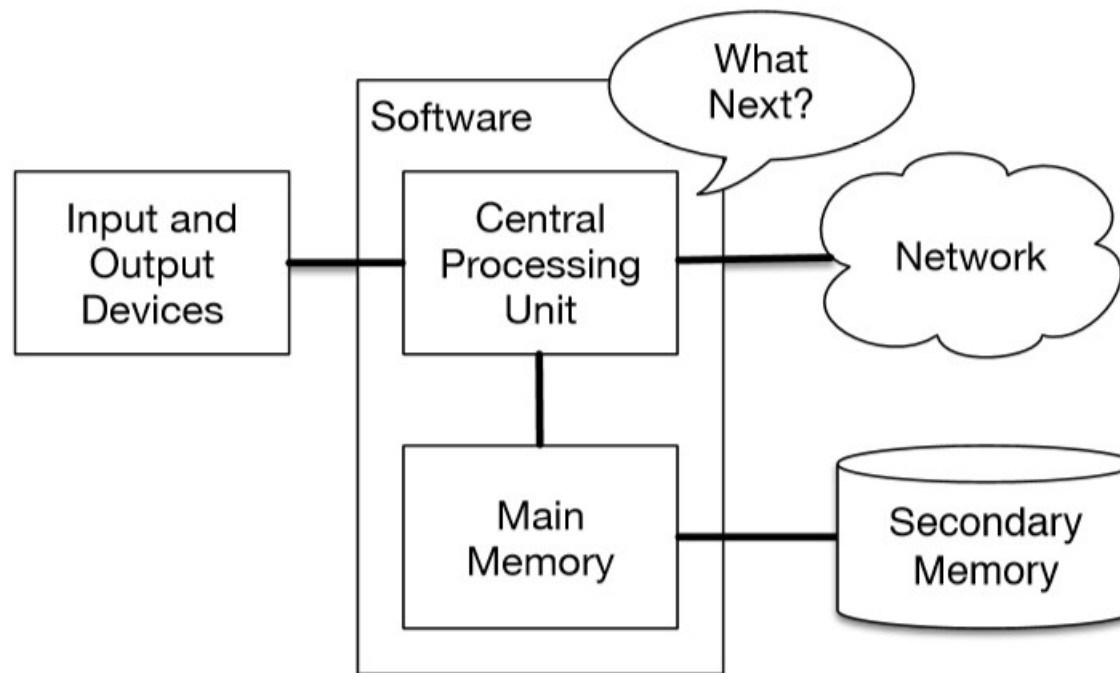


Figure 7.1: Secondary Memory

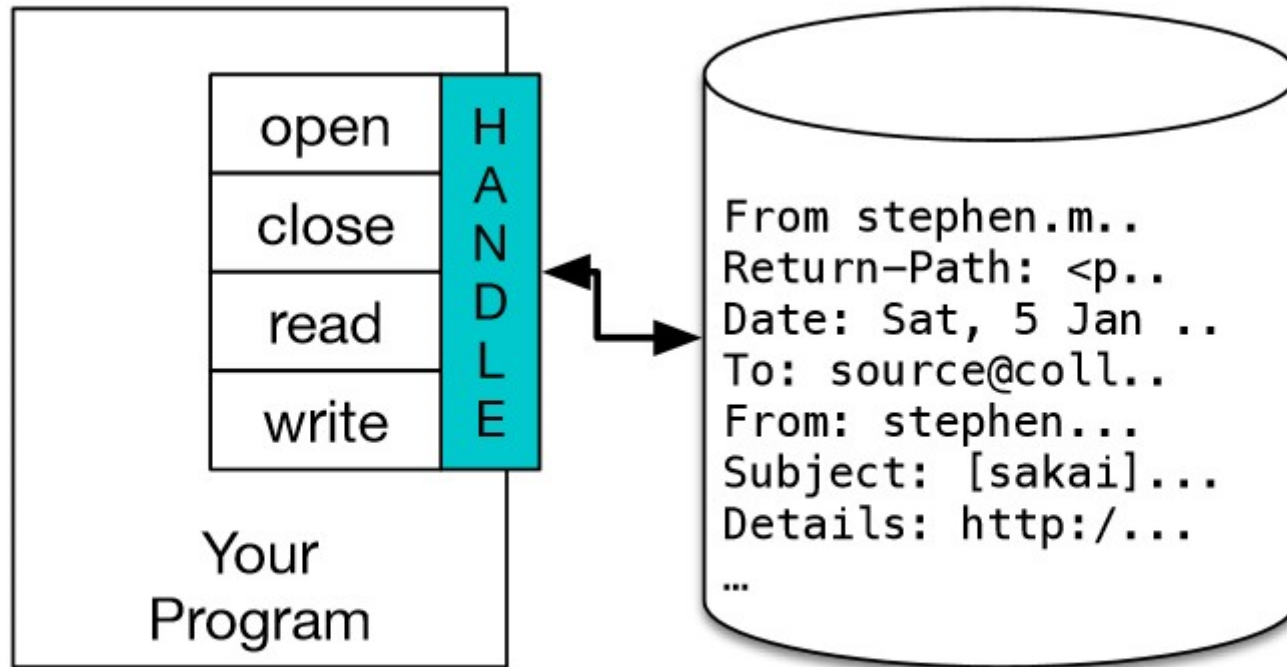


Figure . : A File Handle

File Operations

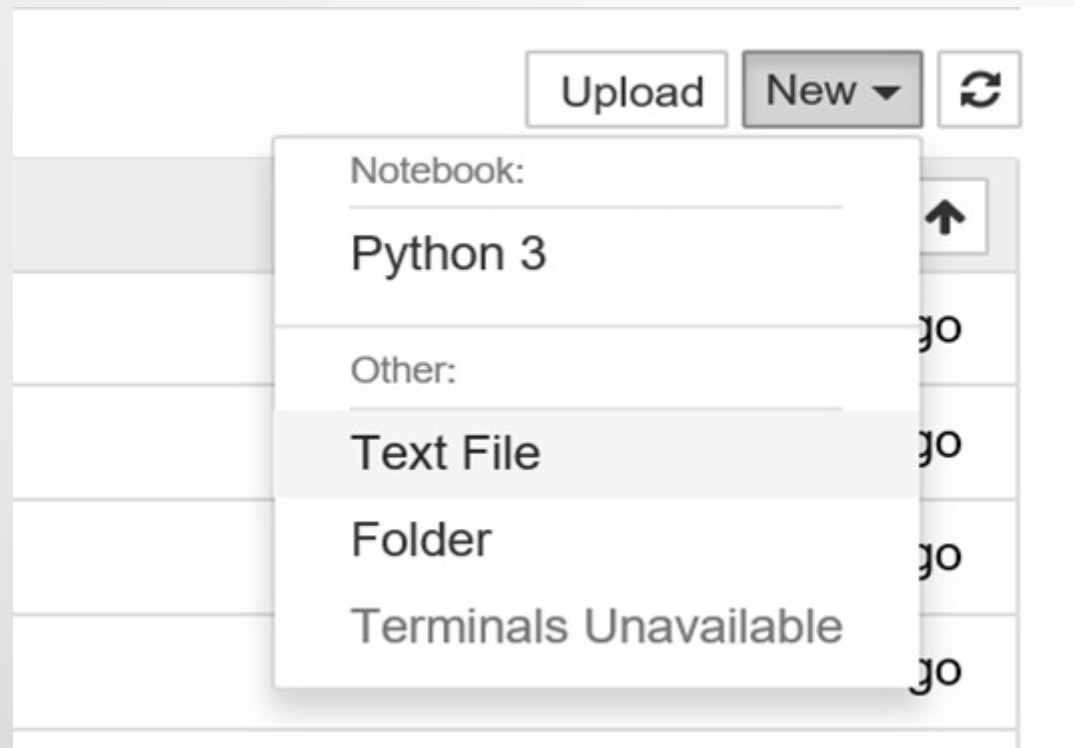
1. Opening a File
2. Closing a File
3. Reading a File
4. Writing to a File
5. Telling the current position in a File
6. Seek to the required position to a file.



Activity

Create a file

- In Jupiter note book open a text file and write half page documents related to your profile and save it with name say *profile.txt*





untitled.txt ✓ a few seconds ago

File

Edit

View

Language

1

```
1 Dr Thyagaraju G S is currently working as
2 Associate Professor and HOD
3 in the department of Computer Science Engineering of SDMI
4 Ujire His Educational Qualification is
5 He has more than 18 years of teaching and research experience.
6 He has published more than 40 research publications in national and international
7 journals and conference proceedings
8 He has been recognized as an active reviewer
9 for Elsevier journal of Advanced Soft Computing since May 2014
10 He has been nominated as associative editor for the international journal
11 of advanced Pervasive and Ubiquitous Computing published by IGI publishing house
12
```

1. File opening

- To open a file we use *open()* function. It requires two arguments, first the file path or file name, second which mode it should open. Modes are like
 - “r” -> open read only, you can read the file but can not edit / delete anything inside
 - “w” -> open with write power, means if the file exists then delete all content and open it to write
 - “a” -> open in append mode
- The default mode is read only, ie if you do not provide any mode it will open the file as read only. Let us open a file

```
f1 = open("bio.txt")  
f1
```

```
<_io.TextIOWrapper name='bio.txt' mode='r' encoding='cp1252'>
```

Open a file

```
f1 = open("bio.txt")  
print(f1)
```

```
<_io.TextIOWrapper name='bio.txt' mode='r' encoding='cp1252'>
```

File Not Found Error

```
f1 = open("bi.txt")  
print(f1)
```

```
-----  
---  
FileNotFoundError                                Traceback (most recent call last)  
<ipython-input-30-0c80fe80dd2b> in <module>()  
----> 1 f1 = open("bi.txt")  
      2 print(f1)  
  
FileNotFoundError: [Errno 2] No such file or directory: 'bi.txt'
```

2.Closing a file

- After opening a file one should always close the
- opened file. We use method *close()* for this.
- **f1.close()**
- Always make sure you *explicitly* close each open 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.


```
f1 = open("bio.txt")  
print(f1)  
f1.close()
```

```
<_io.TextIOWrapper name='bio.txt' mode='r' encoding='cp1252'>
```

3. Reading a File

```
f1 = open("bio.txt")
print(f1)
print(f1.read())
f1.close()
```

```
<_io.TextIOWrapper name='bio.txt' mode='r' encoding='cp1252'>
```

Dr Thyagaraju G S is currently working as

Associate Professor and HOD

in the department of Computer Science Engineering of SDMI

Ujire His Educational Qualification is

He has more than 18 years of teaching and research experience.

He has published more than 40 research publications in national and international

journals and conference proceedings

He has been recognized as an active reviewer

for Elsevier journal of Advanced Soft Computing since May 2014

He has been nominated as associative editor for the international journal

of advanced Pervasive and Ubiquitous Computing published by IGI publishing

- **file.readline()** can help you ~~file.readline()~~ read ~~each~~ each time from the file.

```
f1 = open("bio.txt")  
f1.readline()
```

```
'Dr Thyagaraju G S is currently working as \n'
```

```
f1.readline()
```

```
'Associate Professor and HOD\n'
```

```
f1.readline()
```

```
'in the department of Computer Science Engineering of SDMI\n'
```

- To read all the lines in a list we use *readlines()* method.

```
f1 = open("bio.txt")  
f1.readlines()
```

```
['Dr Thyagaraju G S is currently working as \n',  
  'Associate Professor and HOD\n',  
  'in the department of Computer Science Engineering of SDMI\n',  
  'Ujire His Educational Qualification is \n',  
  'He has more than 18 years of teaching and research experience. \n',  
  'He has published more than 40 research publications in national and in  
ternational \n',  
  'journals and conference proceedings\n',  
  'He has been recognized as an active reviewer \n',  
  'for Elsevier journal of Advanced Soft Computing since May 2014 \n',  
  'He has been nominated as associative editor for the international jour  
nal \n',  
  'of advanced Pervasive and Ubiquitous Computing published by IGI publis  
hing house \n']
```

You can even loop through the lines in a file object

```
f1 = open("bio.txt")
for x in f1:
    print(x, end=' ')
```

Dr Thyagaraju G S is currently working as Associate Professor and HOD in the department of Computer Science Engineering of SDMI Ujire His Educational Qualification is He has more than 18 years of teaching and research experience. He has published more than 40 research publications in national and international journals and conference proceedings He has been recognized as an active reviewer for Elsevier journal of Advanced Soft Computing since May 2014 He has been nominated as associative editor for the international journal of advanced Pervasive and Ubiquitous Computing published by IGI publishing house

Program to read a file whose name is

- Let us write a program which will take the file name as the input from the user and show the content of the file in the console.

```
name = input("Enter the file name: ")  
fobj = open(name)  
print(fobj.read())  
fobj.close()
```



```
name = input("Enter the file name: ")
fobj = open(name)
print(fobj.read())
fobj.close()
```

Enter the file name: bio.txt

Dr Thyagaraju G S is currently working as

Associate Professor and HOD

in the department of Computer Science Engineering of SDMI

Ujire His Educational Qualification is

He has more than 18 years of teaching and research experience.

He has published more than 40 research publications in national and international

journals and conference proceedings

He has been recognized as an active reviewer


for Elsevier journal of Advanced Soft Computing since May 2014

He has been nominated as associative editor for the international journal

1

Searching through a file

- When you are searching through data in a file, it is a very common pattern to read through a file, ignoring most of the lines and only processing lines which meet a particular condition.



```
fhand = open('bio.txt')
count = 0
for line in fhand:
    if line.startswith('He'):
        print(line)
```

He has more than 18 years of teaching and research experience.

He has published more than 40 research publications in national and international

He has been recognized as an active reviewer

He has been nominated as associative editor for the international journal

4. Writing in a File

- Let us open a file then we will write some random text into it by using the write() method.

Now read the file we just

```
fobj = open('me.txt')  
s = fobj.read()  
print(s)
```

MIPI Solutions

Ajith Nagar

Plot No: A7

Staff QuartersUJIRE


Program to copyfile

- In this example we will copy a given text file to another file

```
import sys

f1 = open("me.txt")
s = f1.read()
f1.close()
f2 = open("new.txt", 'w')
f2.write(s)
f2= open("new.txt", 'r')
print(f2.read())
f2.close()
```

MIPI Solutions
Ajith Nagar
Plot No: A7
Staff QuartersUJIRE



Program to count the number of lines (use try and except.)

```
fname = input('Enter the file name: ')

try:
    fhand = open(fname)

except:
    print('File cannot be opened:', fname)
    exit()

count = 0

for line in fhand:
    count = count + 1

print('There were', count, 'lines in', fname)
```

Enter the file name: me.txt
There were 4 lines in me.txt

Program to count the words

Problem Solution

1. Take the file name from the user.
2. Read each line from the file and split the line to form a list of words.
3. Find the length of items in the list and print it.
4. Exit.

Source Program

```
fname = input("Enter file name:")  
  
num_words = 0  
  
with open(fname, 'r') as f:  
    for line in f:  
        words = line.split()  
        num_words += len(words)  
print("Number of words:")  
print(num_words)
```


```
Enter file name: me.txt  
Number of words:  
9
```

Python Program to Read a String from the User and Append it into a File

- Problem Solution
- 1. Take the file name from the user.
- 2. Open the file in append mode.
- 2. Take in a string from the user, append it to the existing file and close the file.
- 3. Open the file again in read mode and display the contents of the file.
- 4. Exit.

Source Program

```
fname = input("Enter file name: ");
file3=open(fname,"a")
c=input("Enter string to append: \n");
file3.write("\n")
file3.write(c)
file3.close()
print("Contents of appended file:");
file4=open(fname,'r')
line1=file4.readline()
while(line1!=""):
    print(line1)
    line1=file4.readline()
file4.close()
```



```
Enter file name: me.txt
Enter string to append:
karnataka
Contents of appended file:
MIPI Solutions
```

```
Ajith Nagar
```

```
Plot No: A7
```

```
Staff QuartersUJIRE
```

```
karnataka
```

Python program to count a number of spaces

Algorithm :

1. Take the file name from the user.
2. Open the file in read mode.
2. Create one variable to store the *count of blank spaces* and initialize it as '0'.
3. Open the file and read *line one by one*.
4. For each line, read words one by one.
5. For each word, read *character one by one*.
6. Check each character if it is *space or not*.
7. If the character is space, *increment* the *count* variable.
8. Finally print out the *count*.

Source Program

```
fname = input("Enter file name: ")
fp=open(fname,"r")
space_count = 0
for line in fp:
    words = line.split()
    for word in words:
        for char in word:
            if(char.isspace):
                space_count = space_count + 1

print("Total blank space found : ",space_count)
```

```
Enter file name: bio.txt
Total blank space found : 518
```

5.ftell() and 6.fseek()

- **ftell()** is used to get to know the current position in the file
- **fseek()** is used to change the current position in a file

Input File : me.txt

MIPI Solutions
Ajith Nagar
Plot No: A7
Staff QuartersUJIRE

```
#Open a file
f = open('me.txt', 'r+')
data = f.read(19);
print('Read String is : ', data)

#Check current position
position = f.tell();
print('Current file position : ', position)

#Reposition pointer at the 5th position
position = f.seek(5, 0);
data = f.read(10);
print('Again read String is : ', data)
#Reposition pointer at the beginning once again
position = f.seek(10, 0);
data = f.read(5);
print('Again read String is : ', data)
#Reposition pointer at the beginning once again
position = f.seek(0, 0);
data = f.read(10);
print('Again read String is : ', data)
```

Output

```
Read String is :  MIPI Solutions  
Ajit  
Current file position :  20  
Again read String is :  Solutions  
  
Again read String is :  ions  
  
Again read String is :  MIPI Solut
```

Exercises

- Write a program to read through a file and print the contents of the file (line by line) all in upper case.
- Write a program to prompt for a file name, and then read through the file and look for lines of the form: X-DSPAM-Confidence:0.8475



End of Module2