Module-3

Semester-7th Faculty-A.K.Rout Branch-CSE

Python

Python is a general-purpose high level programming language and suitable for providing a solid foundation to the reader in the area of cloud computing.
The main characteristics of Python are:

1) Multi-paradigm programminglanguage.

2) Python supports more than one programming paradigms including object- oriented programming and structured programming.

3) InterpretedLanguage.

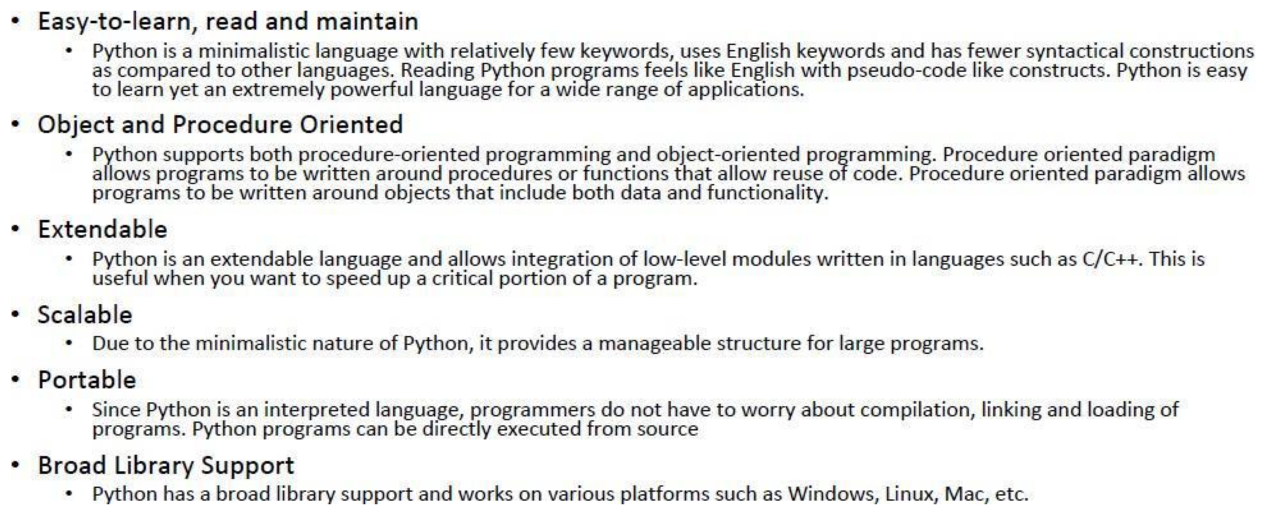
4) Python is an interpreted language and does not require an explicit compilationstep.

5) The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.

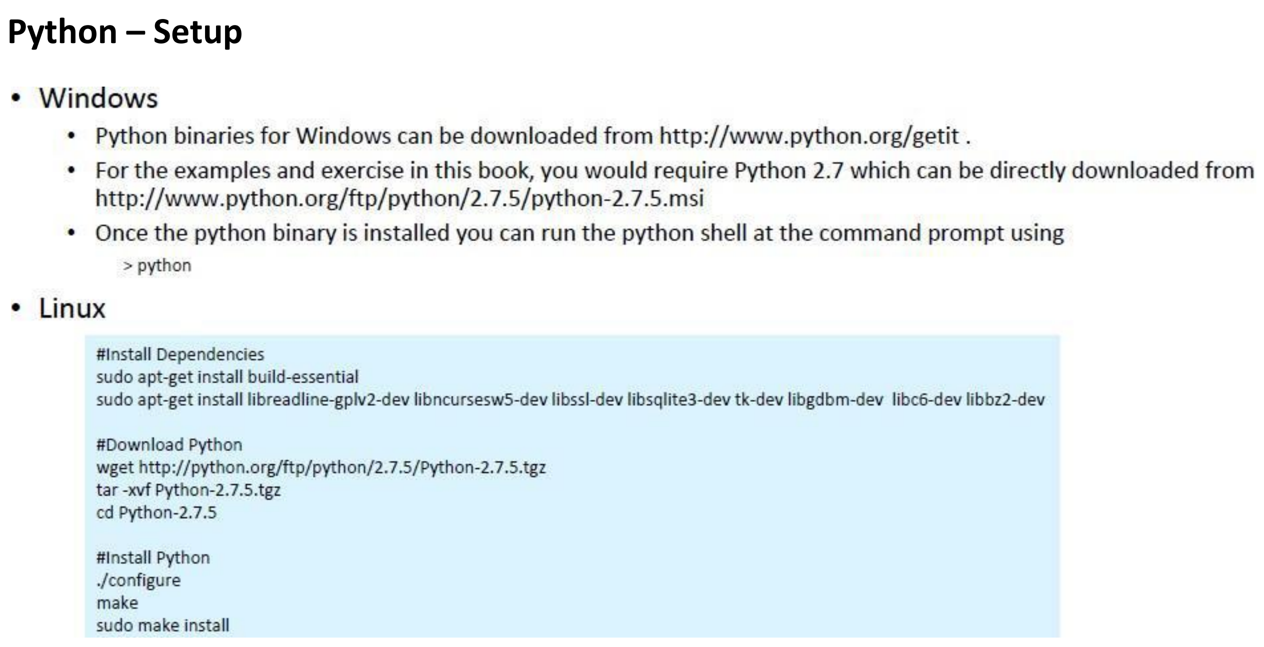
6) Interactive Language

7) Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreterdirectly.

Python Benefits



- **Easy-to-learn, read and maintain**
  - Python is a minimalistic language with relatively few keywords, uses English keywords and has fewer syntactical constructions as compared to other languages. Reading Python programs feels like English with pseudo-code like constructs. Python is easy to learn yet an extremely powerful language for a wide range of applications.
- **Object and Procedure Oriented**
  - Python supports both procedure-oriented programming and object-oriented programming. Procedure oriented paradigm allows programs to be written around procedures or functions that allow reuse of code. Procedure oriented paradigm allows programs to be written around objects that include both data and functionality.
- **Extendable**
  - Python is an extendable language and allows integration of low-level modules written in languages such as C/C++. This is useful when you want to speed up a critical portion of a program.
- **Scalable**
  - Due to the minimalistic nature of Python, it provides a manageable structure for large programs.
- **Portable**
  - Since Python is an interpreted language, programmers do not have to worry about compilation, linking and loading of programs. Python programs can be directly executed from source
- **Broad Library Support**
  - Python has a broad library support and works on various platforms such as Windows, Linux, Mac, etc.



Python – Setup

- Windows
  - Python binaries for Windows can be downloaded from http://www.python.org/getit .
  - For the examples and exercise in this book, you would require Python 2.7 which can be directly downloaded from http://www.python.org/ftp/python/2.7.5/python-2.7.5.msi
  - Once the python binary is installed you can run the python shell at the command prompt using

```
> python
```

- Linux

```
#Install Dependencies
sudo apt-get install build-essential
sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev  libc6-dev libbz2-dev

#Download Python
wget http://python.org/ftp/python/2.7.5/Python-2.7.5.tgz
tar -xvf Python-2.7.5.tgz
cd Python-2.7.5

#Install Python
./configure
make
sudo make install
```

Datatypes
Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.
There are various data types in Python. Some of the important types are listed below.
Python Numbers
Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python. We can use the type() function to know which class a variable or a value belongs to and the isinstance() function to check if an object belongs to a particular class.

Script.py
```
a = 5
print(a, "is of type", type(a))
a = 2.0
print(a, "is of type", type(a))
a = 1+2j
print(a, "is complex number?", isinstance(1+2j,complex))
```

Integers can be of any length, it is only limited by the memory available. A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. 1 is integer, 1.0 is floating point number. Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part. Here are someexamples.

```
>>> a = 1234567890123456789
>>> a 1234567890123456789
>>> b = 0.1234567890123456789
>>> b 0.12345678901234568

>>> c = 1+2j
>>> c (1+2j)
```

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [].

```
>>> a = [1, 2.2, 'python']
```

We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Script.py

```
1. a = [5,10,15,20,25,30,35,40]
2. # a[2] = 15
3. print("a[2] = ", a[2])
4. # a[0:3] = [5, 10, 15]
5. print("a[0:3] = ", a[0:3])
6. # a[5:] = [30, 35, 40]
7. print("a[5:] = ", a[5:])
```

Lists are mutable, meaning; value of elements of a list can be altered.

```
>>> a = [1,2,3]
>>> a[2]=4
>>> a [1, 2, 4]
```

Python Tuple

Tuple is an ordered sequences of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated bycommas.

```
>>> t = (5,'program', 1+3j)
```

Script.py

```
t = (5,'program', 1+3j) # t[1] = 'program' print("t[1] = ", t[1])
# t[0:3] = (5, 'program', (1+3j))
print("t[0:3] = ", t[0:3]) # Generates error
# Tuples are immutable t[0] = 10
```

Python Strings

String is sequence of Unicode characters. We can use single quotes or double quotes to represent strings. Multi-line strings can be denoted using triple quotes, ''' or """.

```
>>> s = "This is a string"
>>> s = '''a multiline
```

Like list and tuple, slicing operator [ ] can be used with string. Strings are immutable.

Script.py

```
a ={5,2,3,1,4}
# printing setvariable print("a = ", a)
# data type of variable a print(type(a))
```

We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates. Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work. It is generally used when we have a huge amount of data.

Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of anytype.

```
>>> d = {1:'value','key':2}
>>> type(d)
<class 'dict'>
```

We use key to retrieve the respective value. But not the other way around.

Script.py

```
d ={1:'value','key':2} print(type(d)) print("d[1] = ",d[1]);
print("d['key'] = ", d['key']); # Generates error print("d[2] = ",d[2]);
```

Python if...else Statement

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes. Decision making is required when we want to execute a code only if a certain condition is satisfied.

The if…elif…else statement is used in Python for decision making.

Python if Statement Syntax

if test expression: statement(s)

Here, the program evaluates the test expression and will execute statement(s) only if the text expression is True.

If the text expression is False, the statement(s) is not executed. In Python, the body of the if statement is indicated by the indentation. Body starts with an indentation and the first unindented line marks the end. Python interprets non-zero values as True. None and 0 are interpreted as False.

Python if Statement Flowchart

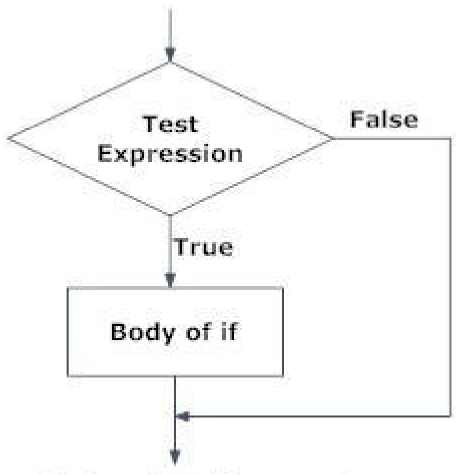


Fig: Operation of if statement

Example: Python if Statement

```
# If the number is positive, we print an appropriate message num = 3
if num > 0:
print(num, "is a positive number.") print("This is always printed.")
```

num = -1 if num >0:
print(num, "is a positive number.") print("This is also always printed.")
When you run the program, the output willbe: 3 is a positivenumber
This is alwaysprinted
This is also always printed.
In the above example, num > 0 is the test expression. The body of if is executed only if this evaluates to True.
When variable num is equal to 3, test expression is true and body inside body of if is executed. If variable num is equal to -1, test expression is false and body inside body of if is skipped. The print() statement falls outside of the if block (unindented). Hence, it is executed regardless of the testexpression.
Python if...else Statement Syntax
if test expression:
Body of if

else:
Body of else
The if..else statement evaluates test expression and will execute body of if only when test condition is True.
If the condition is False, body of else is executed. Indentation is used to separate the blocks.
Python if..else Flowchart
Example of

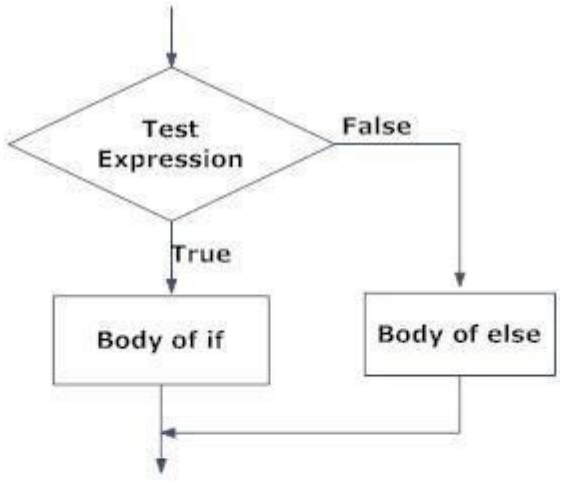


Fig: Operation of if...else statement

Example of if...else
# Program checks if the number is positive or negative # And displays an appropriate message
num = 3
# Try these two variations as well. # num = -5
# num = 0
if num >= 0: print("Positive or Zero")
else:

print("Negative number")
In the above example, when num is equal to 3, the test expression is true and body of if is executed and body of else is skipped.
If num is equal to -5, the test expression is false and body of else is executed and body of if is skipped.
If num is equal to 0, the test expression is true and body of if is executed and body of else is skipped.

Python if...elif...else Statement Syntax
if test expression:
Body of if
elif test expression:
Body of elif else:
Body of else
The elif is short for else if. It allows us to check for multiple expressions. If the condition for if is False, it checks the condition of the next elif block and so on. If all the conditions are False, body of else is executed. Only one block among the several if...elif...else blocks is executed according to the condition. The if block can have only one else block. But it can have multiple elif blocks.

Flowchart of if...elif...else

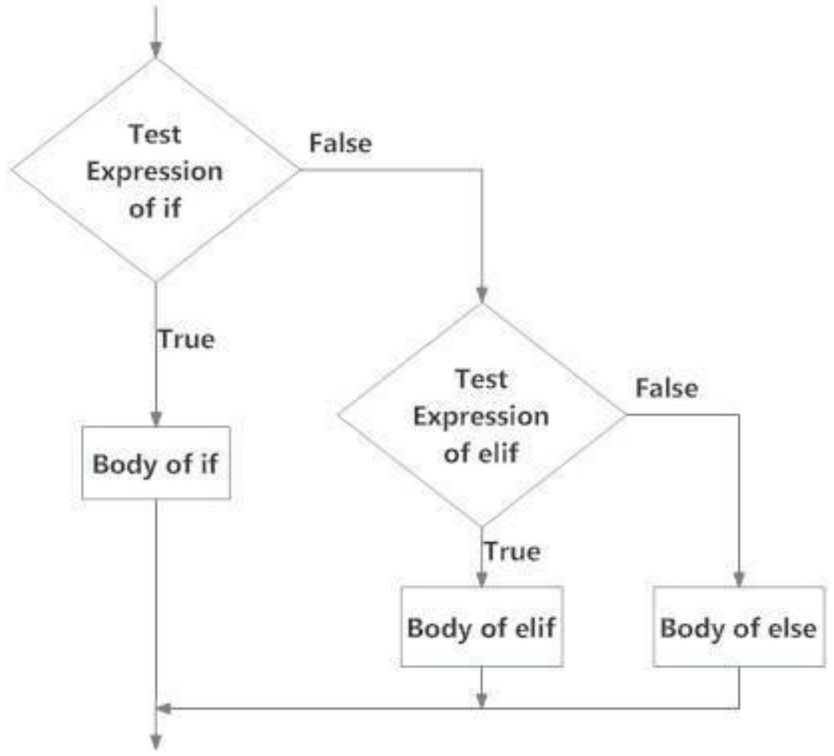


Fig: Operation of if...elif...else statement

Example of if...elif...else
# In this program,
# we check if the number is positive or # negative or zero and
# display an appropriate message num = 3.4

# Try these two variations as well: # num = 0
# num = -4.5 if num > 0:
print("Positive number") elif num == 0:
print("Zero") else:
print("Negative number")
When variable num is positive, Positive number is printed. If num is equal to 0, Zero is printed.
If num is negative, Negative number is printed

Python Nested if statements

We can have a if...elif...else statement inside another if...elif...else statement. This is called nesting in computer programming. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting. This can get confusing, so must be avoided if we can.

Python Nested if Example
# In this program, we input a number # check if the number is positive or # negative or zero anddisplay
# an appropriate message
# This time we use nested if
num = float(input("Enter a number: ")) if num >= 0:
if num == 0: print("Zero")
else:
print("Positive number")
else:
print("Negative number")

Output 1

Enter a number: 5 Positive number Output 2
Enter a number: -1

Negative number Output 3
Enter a number: 0 Zero

Python for Loop

The for loop in Python is used to iterate over a sequence (list, tuple, string) or other iterable objects. Iterating over a sequence is called traversal.
Syntax of for Loop for val in sequence:
Body of for
Here, val 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.
Flowchart of for Loop

Syntax

# Program to find the sum of all numbers stored in a list # List of numbers
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]
# variable to store the sum sum = 0
# iterate over the list for val in numbers:
sum = sum+val # Output: The sum is 48 print("The sum is", sum)
when you run the program, the output will be: The sum is 48

The range() function

We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided. This function does not store all the values in emory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on thego.
To force this function to output all the items, we can use the function list(). The following example will clarify this.

```
# 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)))
```

We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing. Here is an example.

```
# Program to iterate through a list using indexing genre = ['pop', 'rock', 'jazz']
# iterate over the list using index for i in range(len(genre)):
print("I like", genre[i])
```

When you run the program, the output will be: I likepop
I likerock I likejazz

What is while loop in Python?

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.

Syntax of while Loop in Python

while test_expression: Body of while

In while loop, test expression is checked first. The body of the loop is entered only if the test_expression evaluates to True. After one iteration, the test expression is checked again. This process continues until the test_expression evaluates to False. In Python, the body of the while loop is determined through indentation. Body starts with indentation and the first unindented line marks the end. Python interprets any non-zero value as True. None and 0 are interpreted asFalse.

Flowchart of while Loop

```
# Program to add natural # numbers upto
# sum = 1+2+3+...+n
# To take input from the user, # n = int(input("Enter n: "))
n = 10
# initialize sum and counter sum = 0
i = 1
while i <= n: sum = sum + i

i=i+1 # updatecounter # print thesum
print("The sum is", sum)
```

When you run the program, the output will be: Enter n: 10
The sum is 55

In the above program, the test expression will be True as long as our counter variable i is less than or equal to n (10 in ourprogram).
We need to increase the value of counter variable in the body of the loop. This is very important (and mostly forgotten). Failing to do so will result in an infinite loop (never ending loop).
Finally the result is displayed.

Python Modules
A file containing a set of functions you want to include in the application is called Module.
Create a Module
To create a module just save the code you want in a file with the file extension .py:
Example
Save this code in a file named mymodule.py def greeting(name):
print("Hello, " + name)
Use a Module
Now we can use the module we just created, by using the import statement:
Example
Import the module named mymodule, and call the greeting function: import mymodule
mymodule.greeting("Jonathan")
Note: When using a function from a module, use the syntax: module_name.function_name.
Variables in Module
The module can contain functions, as already described, but also variables of all types(arrays, dictionaries, objects etc):
Example
Save this code in the file mymodule.py

person1 = {"name": "John","age": 36,"country": "Norway"}
Example
Import the module named mymodule, and access the person1 dictionary: import mymodule
a = mymodule.person1["age"]
print(a)
Naming a Module
You can name the module file whatever you like, but it must have the file extension .py
Re-naming a Module
You can create an alias when you import a module, by using the as keyword:
Example
Create an alias for mymodule called mx: import mymodule as mx
a = mx.person1["age"] print(a)
Built-in Modules
There are several built-in modules in Python, which you can import whenever you like.
Example
Import and use the platform module: import platform
x = platform.system() print(x)
Using the dir() Function
There is a built-in function to list all the function names (or variable names) in a module. The dir() function:
Example
List all the defined names belonging to the platform module: import platform

x = dir(platform) print(x)
Note: The dir() function can be used on all modules, also the ones you create yourself.
Import from Module
You can choose to import only parts from a module, by using the from keyword.
Example
The module named mymodule has one function and one dictionary: def greeting(name):
print("Hello, " + name)

person1 = {"name": "John", "age": 36, "country": "Norway"}
Example
Import only the person1 dictionary from the module: from mymodule import person1
print (person1["age"])
Note: When importing using the from keyword, do not use the module name when referring to elements in the module. Example: person1["age"], not mymodule.person1["age"].
Packages
We don't usually store all of our files in our computer in the same location. We use a well- organized hierarchy of directories for easier access. Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and modules for files. As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptuallyclear.
Similar, as a directory can contain sub-directories and files, a Python package can have sub-packages and modules. A directory must contain a file namedinit.py in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file. Here is an example. Suppose we are developing a game, one possible organization of packages and modules could be as shown in the figure below.
Package Module Structure in Python Programming Importing module from a package
We can import modules from packages using the dot (.) operator. For example, if want to import the start module in the above example, it is done as follows.

import Game.Level.start
Now if this module contains a function named select_difficulty(), we must use the full name to reference it.
Game.Level.start.select_difficulty(2)
If this construct seems lengthy, we can import the module without the package prefix as follows.
from Game.Level import start
We can now call the function simply as follows. start.select_difficulty(2)
Yet another way of importing just the required function (or class or variable) form a module within a package would be as follows.
from Game.Level.start import select_difficulty Now we can directly call this function.
select_difficulty(2)
Although easier, this method is not recommended. Using the full namespace avoids confusion and prevents two same identifier names from colliding. While importing packages, Python looks in the list of directories defined in sys.path, similar as for module search path.
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. When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed. Hence, in Python, a file operation takes place in the followingorder.

1. Open afile

2. Read or write (perform operation)

3. Close thefile

How to open a file?

Python has a built-in function open() to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
>>> f=open("test.txt") # open file in currentdirectory
>>> f = open("C:/Python33/README.txt") # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read 'r', write 'w' or append 'a' to the file. We also specify if we want to open the file in text mode or binary mode. The default is reading in text mode. In this mode, we get strings when reading from the file. On the other hand, binary mode returns bytes and this is the mode to be used when dealing with non-text files like image or exe files.

Python File Modes

Mode	**Description**
'r'	Open a file for reading. (default)
'w'	Open a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
'x'	Open a file for exclusive creation. If the file already exists, the operation fails.
'a'	Open for appending at the end of the file without truncating it. Creates a new file if it does not exist.
't'	Open in text mode. (default)
'b'	Open in binary mode.
'+'	Open a file for updating (reading and writing)

# perform file operations f.close()

This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file.

A safer way is to use a try...finally block. try:

f = open("test.txt",encoding = 'utf-8') # perform file operations

finally: f.close()

This way, we are guaranteed that the file is properly closed even if an exception is raised, causing program flow to stop. The best way to do this is using the with statement. This ensures that the file is closed when the block inside with is exited. We don't need to explicitly call the close() method. It is doneinternally.

with open("test.txt",encoding = 'utf-8') as f: # perform file operations

How to write to File Using Python?

In order to write into a file in Python, we need to open it in write 'w', append 'a' or exclusive creation 'x' mode. We need to be careful with the 'w' mode as it will overwrite into the file if it already exists. All previous data are erased. Writing a string or sequence of bytes (for binary files) is done using write() method. This method returns the number of characters written to the file.

with open("test.txt",'w',encoding = 'utf-8') as f: f.write("my first file\n")
f.write("This file\n\n") f.write("contains three lines\n")

This program will create a new file named 'test.txt' if it does not exist. If it does exist, it is overwritten. We must include the newline characters ourselves to distinguish different lines.

How to read files in Python?

To read a file in Python, we must open the file in reading mode. There are various methods available for this purpose. We can use the read(size) method to read in size number of data. If size parameter is not specified, it reads and returns up to the end of the file.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4) # read the first 4 data

'This'
>>>f.read(4) # read the next 4 data ' is'
>>>f.read() # read in the rest till end of file 'my first file\nThis file\ncontains threelines\n'
>>> f.read() # further reading returns empty sting "
```

We can see that, the read() method returns newline as '\n'. Once the end of file is reached, we get empty string on further reading. We can change our current file cursor (position) using the seek() method. Similarly, the tell() method returns our current position (in number of bytes).

```
>>>f.tell() # get the current file position 56
>>> f.seek(0) # bring file cursor to initial position 0
>>> print(f.read()) # read the entire file This is my first file
This file
contains three lines
```

We can read a file line-by-line using a for loop. This is both efficient and fast.

```
>>> for line in f:
... print(line, end = ")
...
This is my first file This file
contains three lines
```

The lines in file itself has a newline character '\n'.

Moreover, the print() end parameter to avoid two newlines when printing. Alternately, we can use readline() method to read individual lines of a file. This method reads a file till the newline, including the newlinecharacter.

```
>>> f.readline()
'This is my first file\n'
>>> f.readline() 'This file\n'

>>> f.readline() 'contains three lines\n'
>>> f.readline() "
```

Lastly, the readlines() method returns a list of remaining lines of the entire file. All these reading method return empty values when end of file (EOF) is reached.

```
>>> f.readlines()
['This is my first file\n', 'This file\n', 'contains three lines\n']
```

Python File Methods

There are various methods available with the file object. Some of them have been used in above examples. Here is the complete list of methods in text mode with a brief description.

Python File Methods

Method	**Description**
close()	Close an open file. It has no effect if the file is already closed.
detach()	Separate the underlying binary buffer from the

	TextIOBase and return it.
fileno()	Return an integer number (file descriptor) of the file.
flush()	Flush the write buffer of the file stream.
isatty()	Return True if the file stream is interactive.
read(n)	Read at most n characters form the file. Reads till end of file if it is negative or None.
readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
seekable()	Returns True if the file stream supports random access.
tell()	Returns the current file location.
truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to current location.
writable()	Returns True if the file stream can be written to.
write(s)	Write string s to the file and return the number of characters written.
writelines(lines)	Write a list of lines to the file.

readable() Returns True if the file stream can be readfrom.
readline(n=-1) Read and return one line from the file. Reads in at most n bytes if specified.
readlines(n=-1) Read and return a list of lines from the file. Reads in at most n bytes/characters ifspecified.
seek(offset,from=SEEK_SET) Change the file position to offset bytes, in reference to from (start, current,end).
seekable() Returns True if the file stream supports randomaccess. tell() Returns the current filelocation.
truncate(size=None) Resize the file stream to size bytes. If size is not specified, resize to currentlocation.
writable() Returns True if the file stream can be writtento.
write(s) Write string s to the file and return the number of characterswritten. writelines(lines) Write a list of lines to thefile.