PYTHON-FILES

FILE SYSTEM

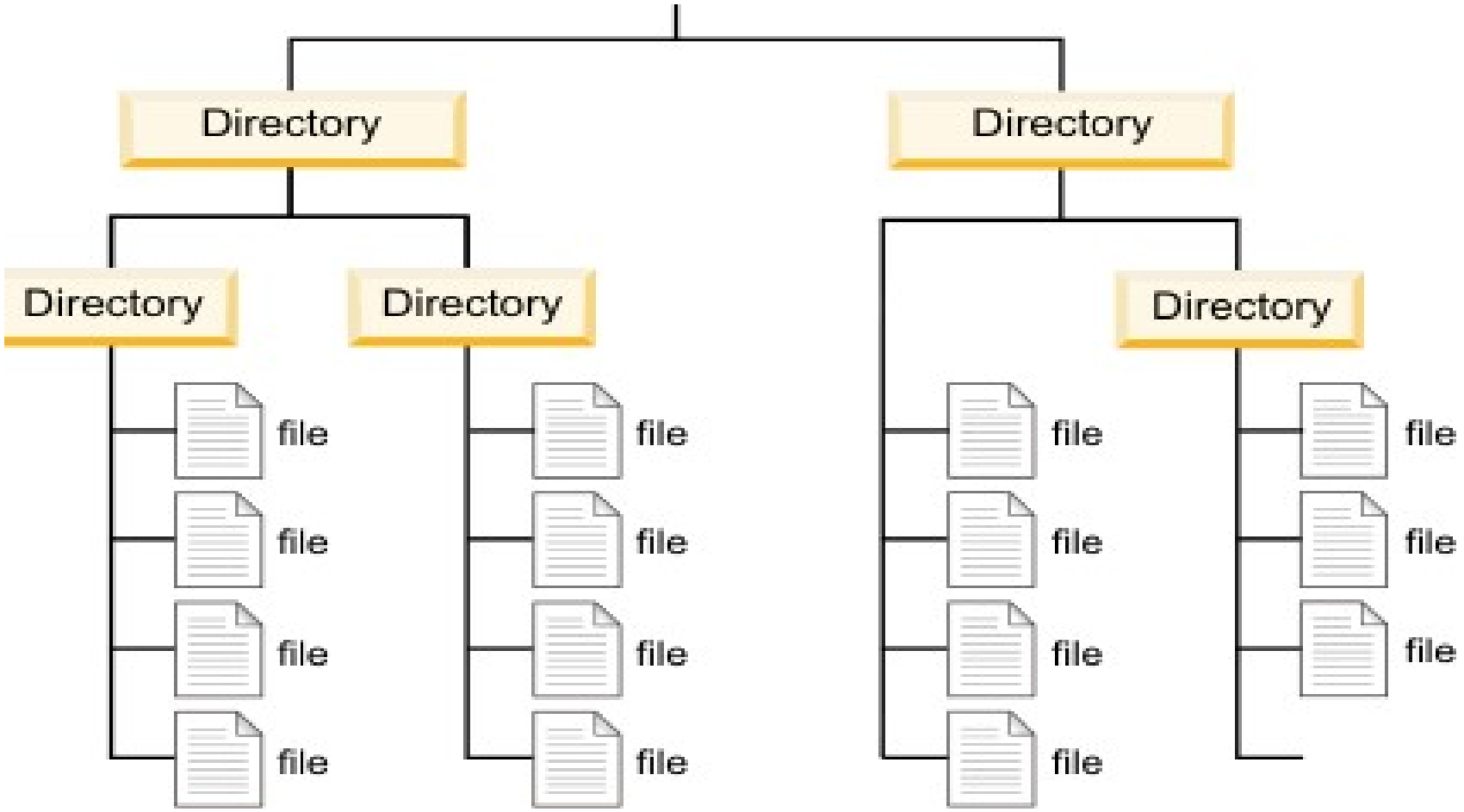
A file system is the structure in which files are given name and placed in the hard disk for storage and retrieval.

- The data used in the Python programs specified in the previous units will not be stored in the data disk. They will be lost once the programs are executed.

However the programs would be available if it is saved as a Python file.

APEC

STRUCTURE OF FILE SYSTEM



FILE

A file stores related data, information, settings, or commands in secondary storage device like magnetic disks, magnetic tapes and optical disks.

- A file can be a sequence of bits, bytes, lines or records depending of the application/software used to create it.
- For example a text file is organized as a sequence of lines.

APC

FILE TYPES

- A file is a collection of contiguous data. It might be a picture, video, text or even set of records.
  - The following are some of the types of files and the applications that support those files. The extension specifies the type of file. For example .txt, .doc, .jpg etc.

APEC

FILE TYPES



FILE TYPES

Type	Application//Software
Text	Notepad
Audio	Media Player
Video	Media Player
Presentations	PowerPoint
DOC	Word
Pictures	Ms Paint
HTML	Browsers
PDF	Acrobat Reader
ZIP	WinZip

Executables	Run Programs

APEC

WAYS TO ACCESS FILES

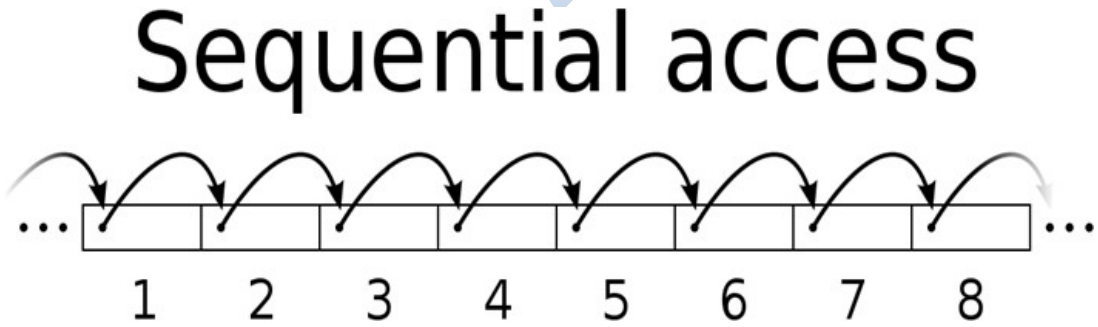
- Sequential access
- Direct/Random access
- Indexed sequential access

APEC

SEQUENTIAL ACCESS

Sequential access is the most basic approach. The contents of the file are accessed in a sequential order.

- For example in a text file the lines are read one after another. Most of the compilers access the file/program to be complied in a sequential manner.



APEC

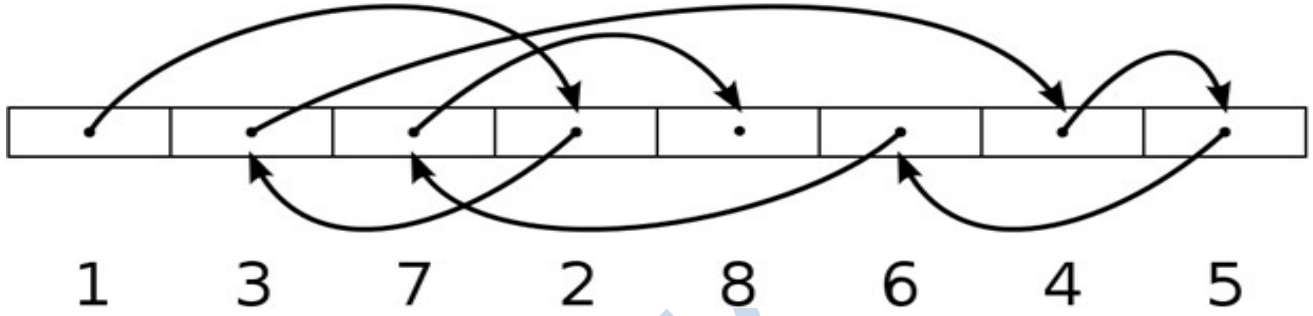
DIRECT/RANDOM ACCESS

Random access mechanism reads the contents of the file from any point of the file.

- It is also called direct access because the desired contents are read accessed directly by specifying the address.

APEC

Random access



APEC

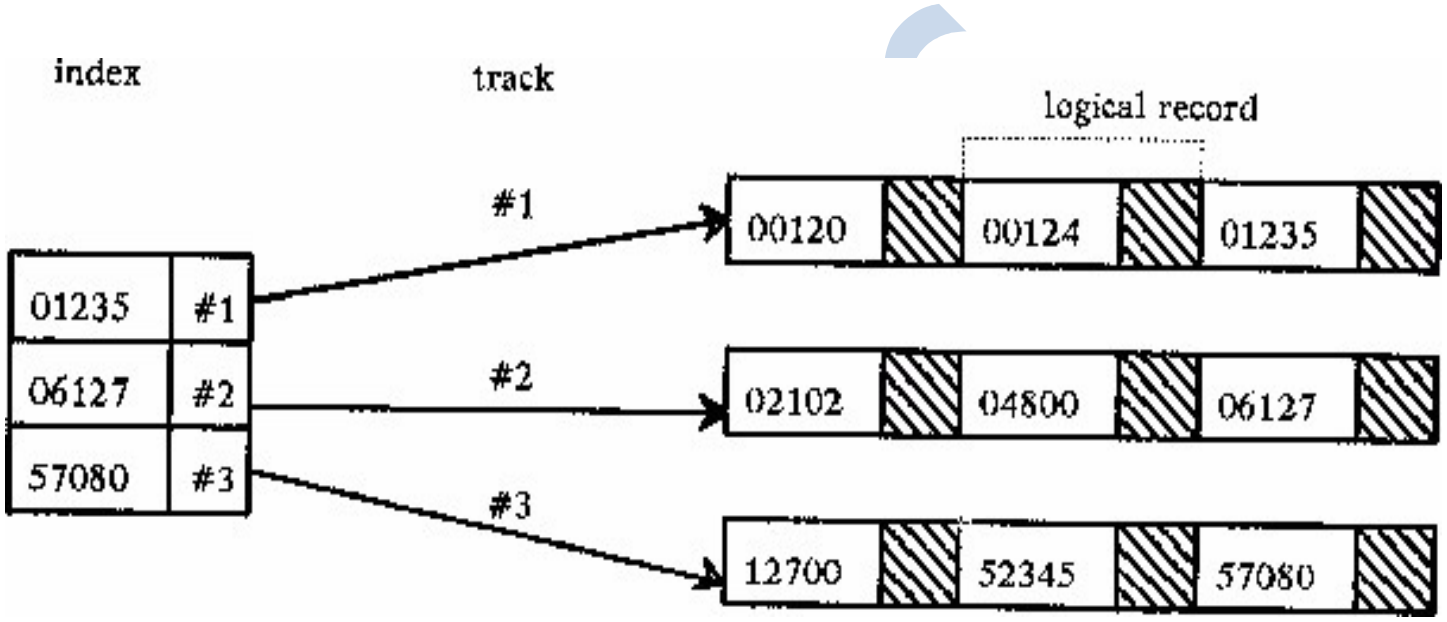
INDEXED SEQUENTIAL ACCESS

This mechanism is combination of both sequential and random access. An index is created for the contents of the file.

- The index can be accessed or searched for desired data in a sequential manner. Once the desired index value is fetched the file contents corresponding to the index is fetched in random.

AEC

INDEXED SEQUENTIAL ACCESS



FILES IN PYTHON

```
1. # File Open Example in Python
2. F_Read = open("Readme.txt", "r")
3. F_Write = open("Writeme.txt", "w")
4. print F_Read
5. print F_Write
```

Execution 1:

```
<open file 'Readme.txt', mode 'r' at 0x01FE50D0>
<open file 'Writeme.txt', mode 'w' at 0x020B4860>
```

APEC

FILE OPENING MODES

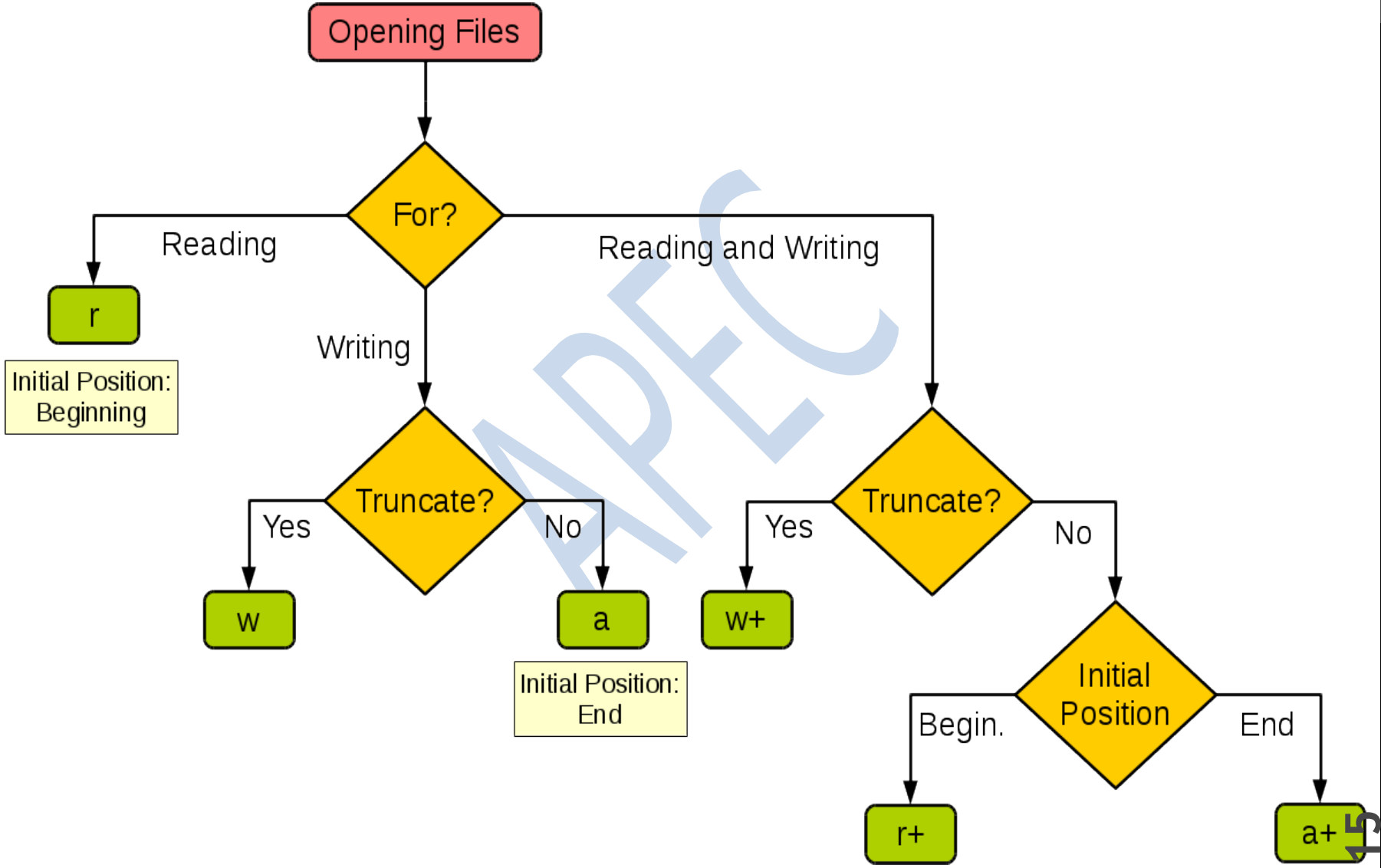
Syntax for opening a file :

<FileVariableName> = **open**(“FilePath”, <Mode>)

Or

<FileVariableName> = **open**(“FilePath”)

APEC


Opening Files
For?
Reading
Writing
Reading and Writing
r
Initial Position: Beginning
Truncate?
Yes
No
w
a
Initial Position: End
Truncate?
Yes
No
w+
Initial Position
Begin.
End
r+
a+

File Modes

Modes	Description
r	Opens a file for reading only.
r+	Opens a file for both reading and writing.
w	Opens a file for writing only. Overwrites the file if the file exists.
w+	Opens a file for both writing and reading.
a	Opens a file for appending. File pointer is at the end of the file.
a+	Opens a file for both appending and reading.

APEC

Attributes of File Object

Attribute	Description
file.closed	If file is closed returns true else false
file.mode	Returns one of the file modes
file.name	Returns name of the file.
file.softspace	Returns false if space explicitly required with print, true otherwise.

APEC

ATTRIBUTES OF FILE OBJECT – EXAMPLE PROGRAM

```
1. fileRead = open("text.txt", "r+") 2.
3. print "Name of the file: ", fileRead.name
4. print "Closed or not : ", fileRead.closed
5. print "Opening mode : ", fileRead.mode
6. print "Softspace flag : ", fileRead.softspace
7.
8. #Most of the above details can be fetched by the below statement
9. print "Printing file object : ", fileRead
```

Execution:

```
Name of the file:  text.txt
Closed or not :  False
Opening mode :  r+
Softspace flag :  0

Printing file object :  <open file 'text.txt', mode 'r+' at 0x01E050D0>
```

APEC

METHODS IN FILE OBJECT

- Opening a File
- Closing a file
- Writing to a file
- Reading from a file
- Appending to a file
- Seek to the location
- Tell the current location

APEC

OPENING A FILE

Syntax for opening a file :

<FileVariableName> = open(“FilePath”, <Mode>)

Or

<FileVariableName> = open(“FilePath”)

- **File is** opened using the open() function

- Example

```
# Open the file "MyFile1.txt"  in append mode
file1 = open("MyFile.txt","a")
```

APEC

CLOSING A FILE

- close() function closes the file and frees the memory space acquired by that file.

Syntax:

File_object.close()

Example:

```
# Opening and Closing a file "MyFile.txt"
file1 = open("MyFile.txt","a") file1.close()
```

APEC

WRITING TO A FILE

- There are two ways to write in a file.
- **write() :** Inserts the string str1 in a single line in the text file.

  Syntax: **File_object.write(str1)**

- **writelines() :**Used to insert multiple strings at a single time.

  Syntax: **File_object.writelines(L)**

  where, L = [str1, str2, str3]

APEC

READING FROM A FILE

- There are three ways to read data from a text file.
- **read() :** Reads n bytes, if n is not specified, reads the entire file.

  Syntax: File_object.read([n])

- **readline() :** For specified n, reads at most n bytes. However, does not reads more than one line, even if n exceeds the length of the line.

  Syntax: File_object.readline([n])

- **readlines() :** Reads all the lines.

APEC

Syntax: File_object.readlines()

APEC

```
# Program to show various ways to read and write data in a file.
file1 = open("newfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]

# \n is placed to indicate EOL (End of Line)
file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes

file1 = open("myfile.txt","r+")

print "Output of Read function is "
print file1.read()

# seek(n) moves to the nth byte
file1.seek(0)

print "Output of Readline function is "
print file1.readline()

file1.seek(0)

# Difference between read and readline
print "Output of Read(9) function is "
print file1.read(9)

file1.seek(0)

print "Output of Readline(9) function is "
print file1.readline(9)

file1.seek(0)
# readlines function
print "Output of Readlines function is "
print file1.readlines()

file1.close()
```

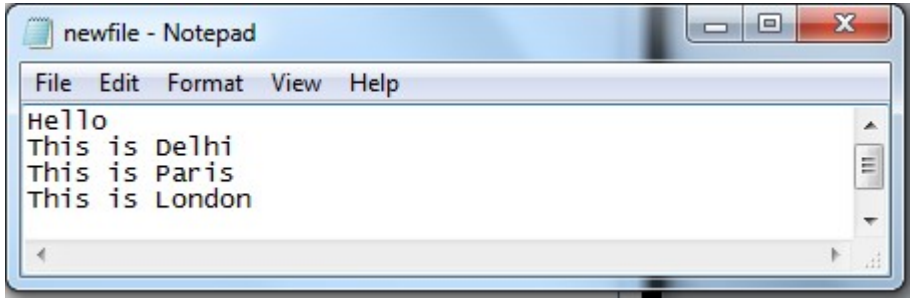
OUTPUT

```
========================= RESTART: C:/Python27/p12.py ========================
Output of Read function is
Hello
This is Delhi
This is Paris
This is London

Output of Readline function is
Hello

Output of Read(9) function is
Hello
Th
Output of Readline(9) function is
Hello

Output of Readlines function is
['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']
>>>
```



File Edit Format Run Options Window Help

```
# Program to show various ways to read and write data in a file.
file1 = open("newfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]

# \n is placed to indicate EOL (End of Line)
file1.write("Hello \n")
file1.writelines(L)
file1.close() #to change file access modes

file1 = open("myfile.txt","r+")

print "Output of Read function is "
print file1.read()

# seek(n) moves to the nth byte
file1.seek(0)

print "Output of Readline function is "
print file1.readline()

file1.seek(0)

# Difference between read and readline
print "Output of Read(9) function is "
print file1.read(9)

file1.seek(0)

print "Output of Readline(9) function is "
print file1.readline(9)

file1.seek(0)
# readlines function
print "Output of Readlines function is "
print file1.readlines()

file1.close()
```

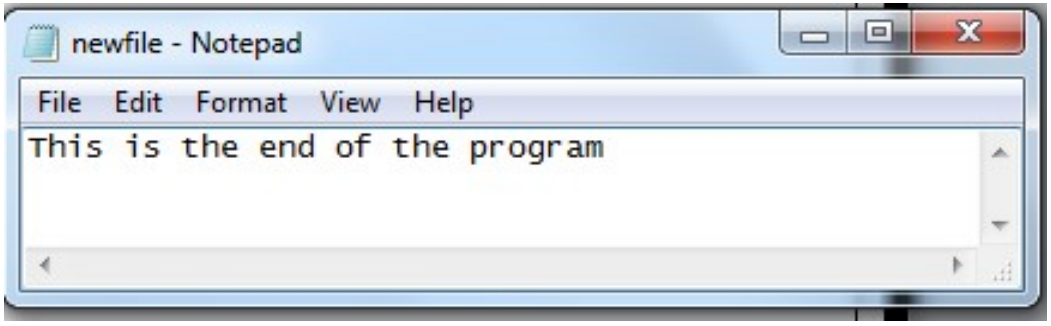
OUTPUT

```
========================= RESTART: C:/Python27/p12.py =========================
Output of Read function is
Hello
This is Delhi
This is Paris
This is London

Output of Readline function is
Hello

Output of Read(9) function is
Hello
Th
Output of Readline(9) function is
Hello

Output of Readlines function is
['Hello \n', 'This is Delhi \n', 'This is Paris \n', 'This is London \n']
>>>
```



APPENDING TO A FILE

File Edit Format Run Options Window Help

```
# Python program to illustrate
# Append vs write mode
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]
file1.writelines(L)
file1.close()

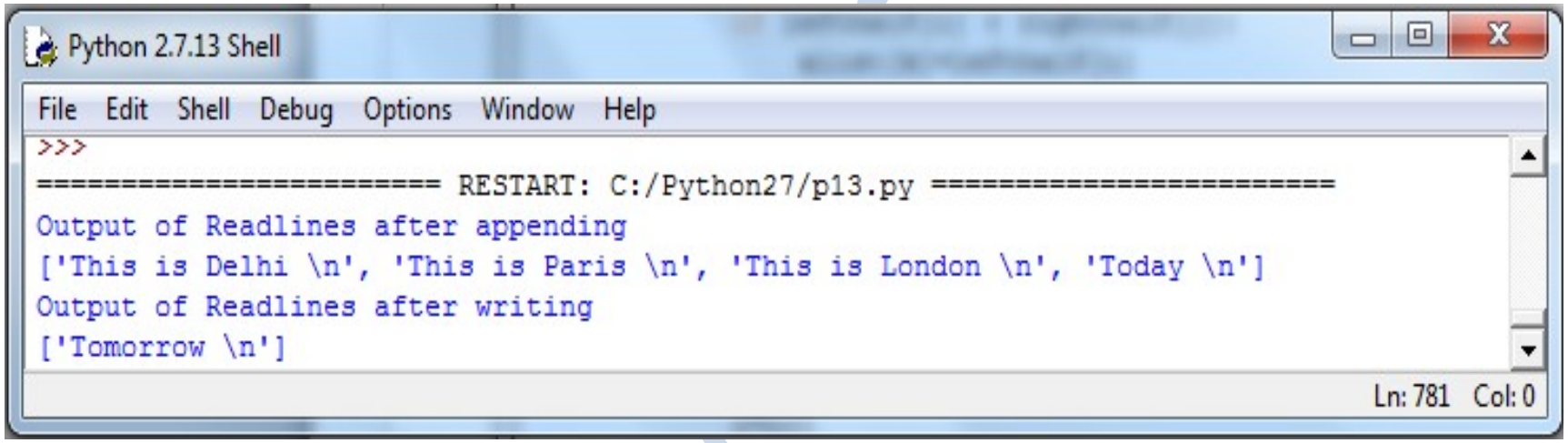
# Append-adds at last
file1 = open("myfile.txt","a+")#append mode
file1.write("Today \n")
file1.close()

file1 = open("myfile.txt","r")
print "Output of Readlines after appending"
print file1.readlines()
file1.close()

# Write-Overwrites
file1 = open("myfile.txt","w")#write mode
file1.write("Tomorrow \n")
file1.close()

file1 = open("myfile.txt","r")
print "Output of Readlines after writing"
print file1.readlines()
```

OUTPUT


Python 2.7.13 Shell
File Edit Shell Debug Options Window Help
>>>
========== RESTART: C:/Python27/p13.py ==========
Output of Readlines after appending
['This is Delhi \n', 'This is Paris \n', 'This is London \n', 'Today \n']
Output of Readlines after writing
['Tomorrow \n']
Ln: 781 Col: 0

SEEK() FUNCTION

Syntax: **filepointer.seek(offset, from_where]**

- **Offset** ➔ Position of the read/write pointer within the file.
- **from_where** ➔ defines the point of reference
  - 0 ➔ means your reference point is the beginning of the file
  - 1 ➔ means your reference point is the current file position
  - 2 ➔ means your reference point is the end of the file.

APEC

TELL()

- Returns the current file location.
- Syntax: **filepointer.tell()**

APEC

File Edit Format Run Options Window Help

```
# Python program to illustrate
# seek and tell function in Python
file1 = open("myfile.txt","w")
L = ["This is Delhi \n","This is Paris \n","This is London \n"]
file1.writelines(L)
file1.close()

# Opening the file in reading mode
file1 = open("myfile.txt","r")
print "Output "
print file1.readline()
print file1.tell()

# Again set the pointer to the beginning
file1.seek(0, 1)
print file1.tell()
line = file1.readline()
print "Read Line:" + line

file1.close()
```

File Edit Shell Debug Options Window Help

```
========================= RESTART: C:\Python27\p13.py =========================
Output
This is Delhi

16
16
Read Line:This is Paris

>>>
```

FORMATTING

FORMAT OPERATOR

One of Python's very interesting features is the format operator.

The general syntax for using the format operator is

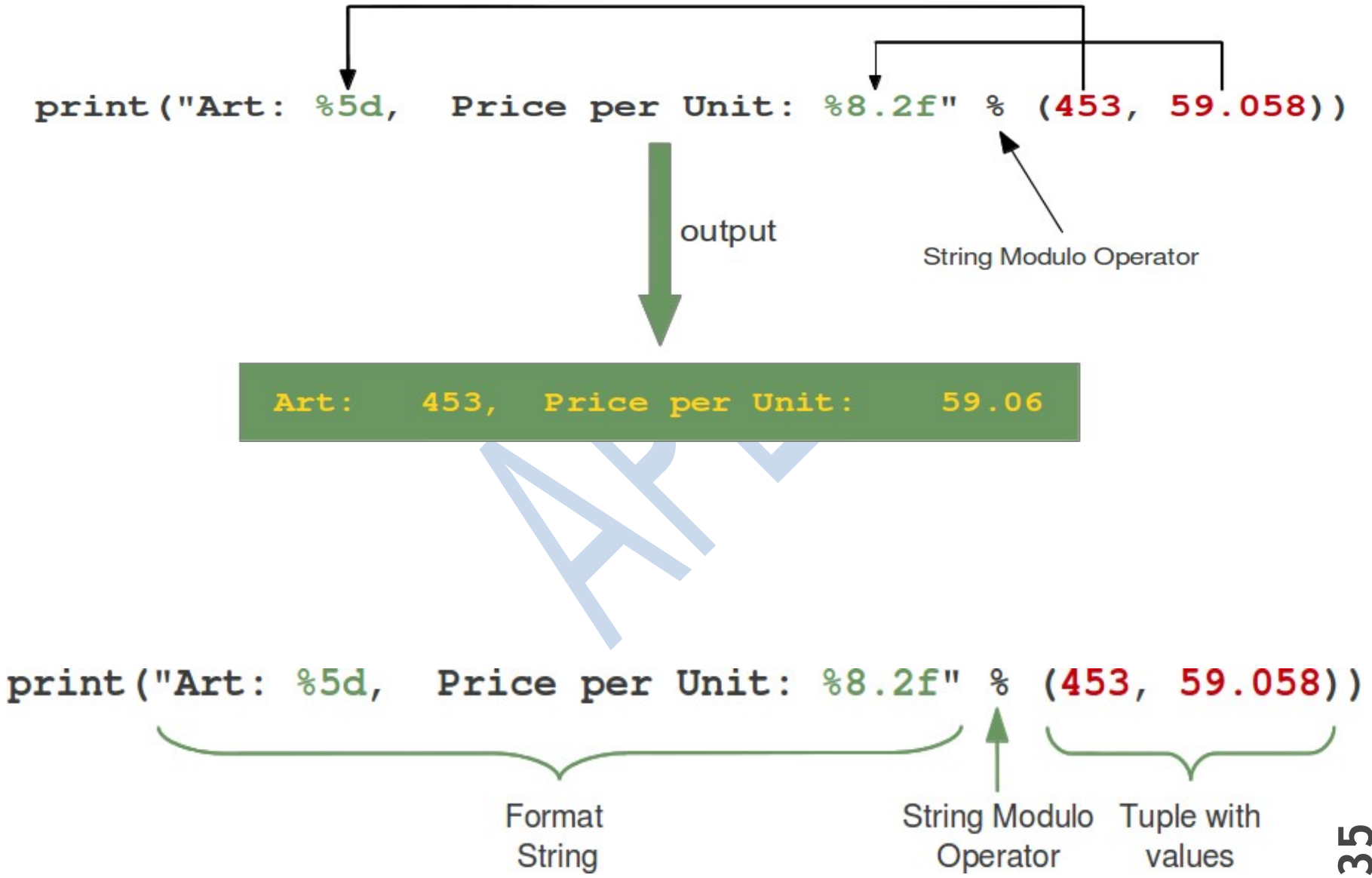
<format expression> % (v1, v2, v3...vn)

Or

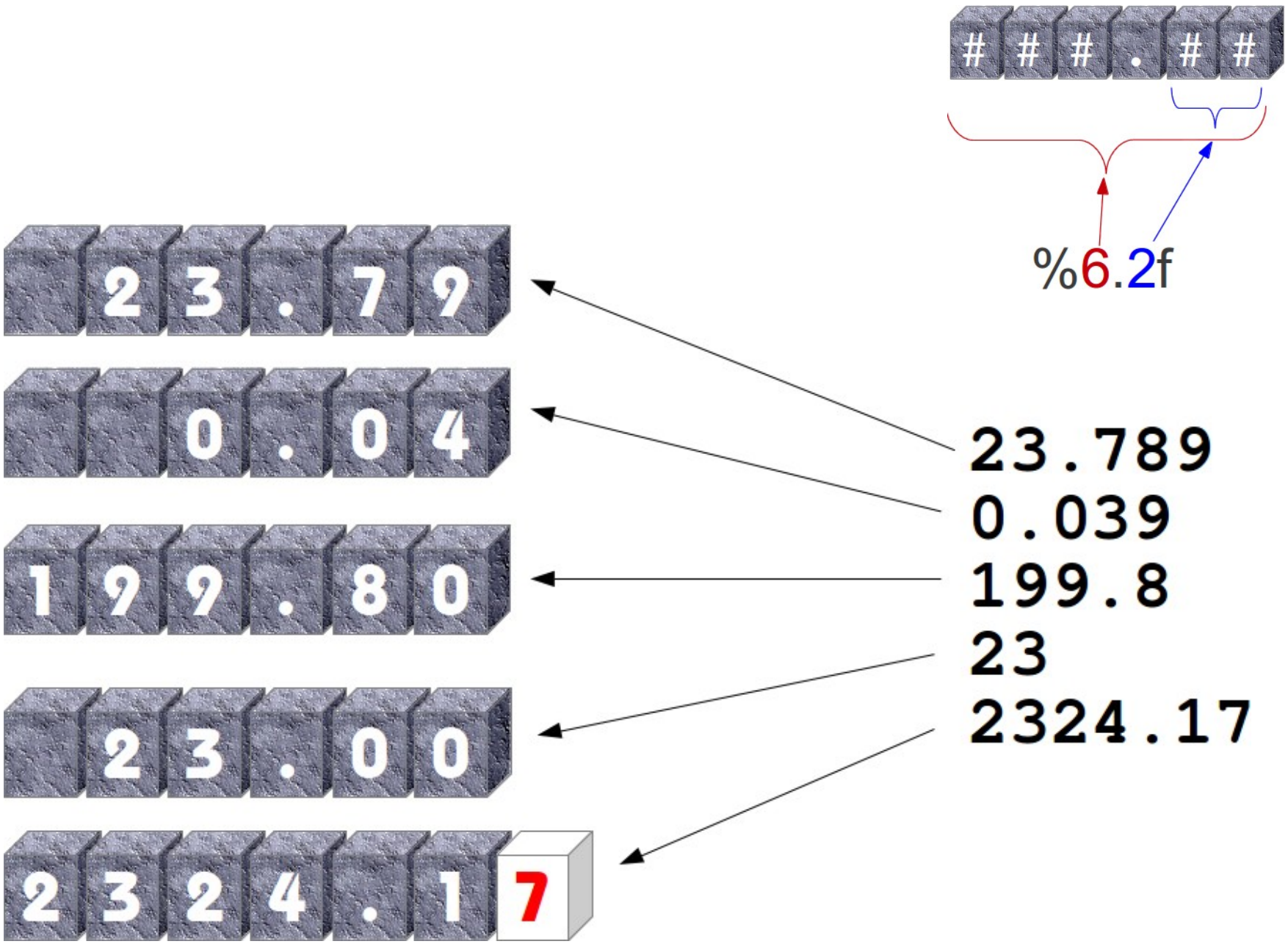
<format expression> % values

Where v1 to vn are variables that are formatted

APEC


print("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))
output
String Modulo Operator
Art: 453, Price per Unit: 59.06
print("Art: %5d, Price per Unit: %8.2f" % (453, 59.058))
Format String
String Modulo Operator
Tuple with values

APEC


###.##
%6.2f
23.789
0.039
199.8
23
2324.17
23.79
0.04
199.80
23.00
2324.17

SYNTAX - FORMAT

[key][flags][width][.precision][length type]conversion type

Key :

Mapping key

Flags:

Conversion flags, which affect the conversion types.

Width:

Minimum field width.

Precision:

Precision, given as a '.' (dot) followed by the precision.

Length type:

Length modifier.

Conversion type:

APEC

Conversion type.

APEC

Flags in Format Expression	
Flag	Use
#	The value conversion will use the ``alternate form"
0	The conversion will be zero padded for numeric values.
-	The converted value is left adjusted
	(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.
+	A sign character ("+" or "-") will precede the conversion
*****	argument specifies width or precision
(var)	mapping variable (dictionary arguments)
m.n.	m is the total width and n is the number of digits to display after the decimal point

Conversion Types	
Conversion	Meaning
d	Signed integer decimal.
i	Signed integer decimal.
o	Unsigned octal.
u	Unsigned decimal.
x	Unsigned hexadecimal (lowercase).
X	Unsigned hexadecimal (uppercase).
e	Floating point exponential format (lowercase).
E	Floating point exponential format (uppercase).
f	Floating point decimal format.
F	Floating point decimal format.
g	Same as "e" if exponent is greater than -4 or less than precision, "f" otherwise.
G	Same as "E" if exponent is greater than -4 or less than precision, "F" otherwise.
c	Single character (accepts integer or single character string).
r	String (converts any python object using repr()).
s	String (converts any python object using str()).
%	No argument is converted, results in a "%" character in the result.

ARC

FORMATTING STRINGS - 5.3.1 IN BOOK

File Edit Format Run Options Window Help

```
#Format strings
print '%s %s' % ('one', 'two')

#Format strings with quotes
print '%s %r' % ('hi', 'bye')

#Padding and aligning strings
print '%30s' % ('I ll have space before')

#Aligning strings - Left
print '%-20s see the space' % ('After me')

#Truncating long strings
print 'Truncate 5 characters of %r : %.5s' % ('1234567910', '1234567910')

#Combining truncating and padding
print 'Truncate and pad of %r : %10.5s ' % ('Python is good','Python is good')

#print a single character
print 'Covert single digit no: %r to char %c' % ('4', '4')
```

```
========================== RESTART: C:/Python27/21.py ==========================
one two
hi 'bye'
        I ll have space before
After me              see the space
Truncate 5 characters of '1234567910' : 12345
Truncate and pad of 'Python is good' :      Pytho
Covert single digit no: '4' to char 4
```

FORMATTING NUMBERS - 5.3.2 IN BOOK

```
#Format numbers
print 'Simple integer formatting : %d' % (42)

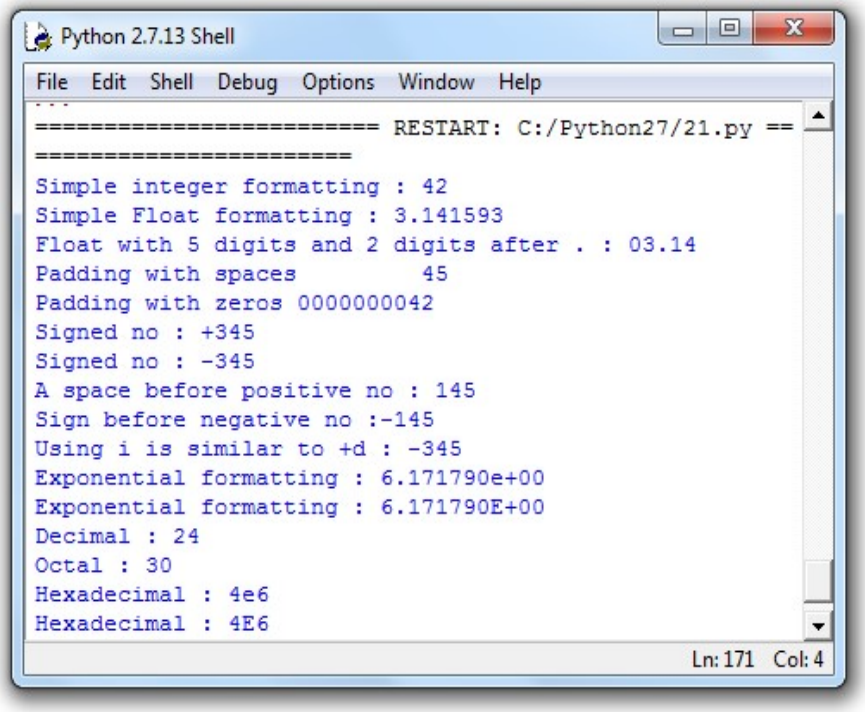
#Format float
print 'Simple Float formatting : %f' % (3.141592653589793)
print 'Float with 5 digits and 2 digits after . : %05.2f' % (3.141592653589793)

#Padding numbers
print 'Padding with spaces %10d' % (45)
print 'Padding with zeros %010d' % (42)

#Signed numbers
print 'Signed no : %+d' % (345)
print 'Signed no : %+d' % (-345)
print 'A space before positive no :% d' % (145)
print 'Sign before negative no :% d' % (-145)
print 'Using i is similar to +d : %+i' % (-345)

#Format Exponential
print 'Exponential formatting : %e' % (6.171790134278593)
print 'Exponential formatting : %E' % (6.171790134278593)

#Format Decimal
print 'Decimal : %u' % (24)
print 'Octal : %o' % (24)
print 'Hexadecimal : %x' % (1254)
print 'Hexadecimal : %X' % (1254)
```



COMMAND LINE ARGUMENTS

- The command line arguments are arguments send to the program being executed.
- **sys.argv** is a variable that contains the command-line arguments passed to the script.

APEC

EXAMPLE

```
1.  # Command Line Arguments Example
2.  import sys
3.   # Get the total number of args passed to main.py
4.  No_Args = len(sys.argv)
5.   # Get the arguments list
6.  List_Args = str(sys.argv)
7.
8.  print (“Numbers of args passed to the script: %d " % No_Args)
9.  print ("Arguments list: %s " % List_Args)
```

Execution:

```
sh-4.3$ python main.py input.txt 678 var3
Numbers of args passed to the script: 1
```

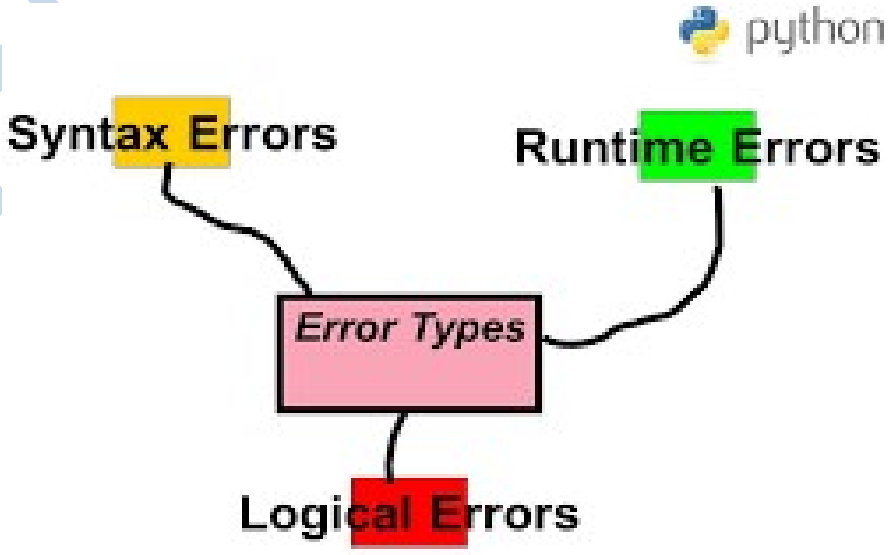
APEC

Args list: ['main.py', 'input.txt', '678', 'var3']

APEC

ERRORS

- Errors are caused by the mistakes in the program.
- There are three types of errors;
  - Syntax errors
  - Runtime Errors
  - Logical Errors.



ADEC

- **Syntax errors** are the grammatical errors
- It prevents the program from running

```
if x<10
print X
```

Execution:

SyntaxError: invalid syntax

```
for  i in range (4) :
print 'hi‘
```

Execution:

```
IndentationError: expected an indented block
```

APEC

- **Runtime errors** occur when we do something illegal

Exception Errors:

X = '678' + 70

Execution:

TypeError: cannot concatenate 'str' and 'int' objects

LOGICAL ERRORS

- **Logical errors** give the wrong output

The following code prints 16 instead of 30.

The line sum_sq+sq must come inside the for loop.

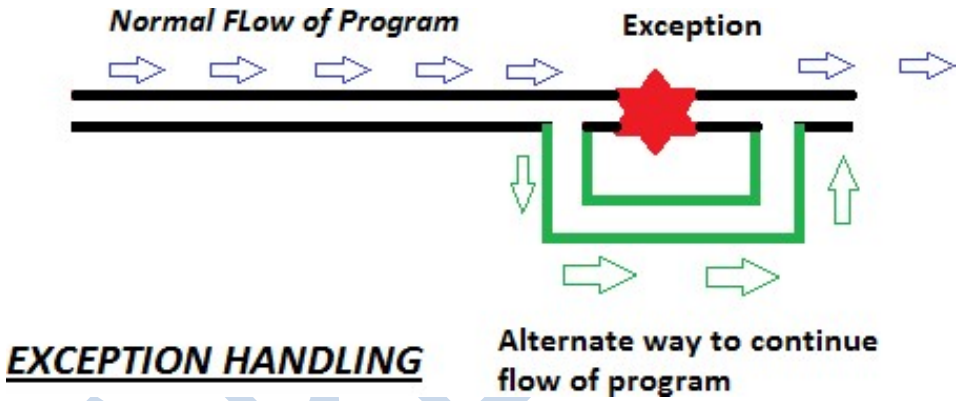
```
sum_sq = 0
for i in range(5):
    sq = i**2
sum_sq += sq
print sum_sq
```

Execution:

16

APEC

EXCEPTION HANDLING



- **Exception** is the unexpected event which disturbs the normal flow of the program.
- An alternate way to continue the program is called **Exception Handling**.

Standard Exceptions	
EXCEPTION	**DESCRIPTION**
Exception	Base class for all exceptions
StopIteration	Raised when the next() method of an iterator points to null
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division by zero takes place
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when Ctrl+c is pressed to interrupt program execution
LookupError	Base class for all lookup errors.
IndexError	Raised when an index is not found in a sequence
KeyError	Raised when specified key is not found in the dictionary.
NameError	Raised when an identifier is not found in the local or global namespace.
UnboundLocalError	Raised when a local variable has no value assigned to it.

ARC

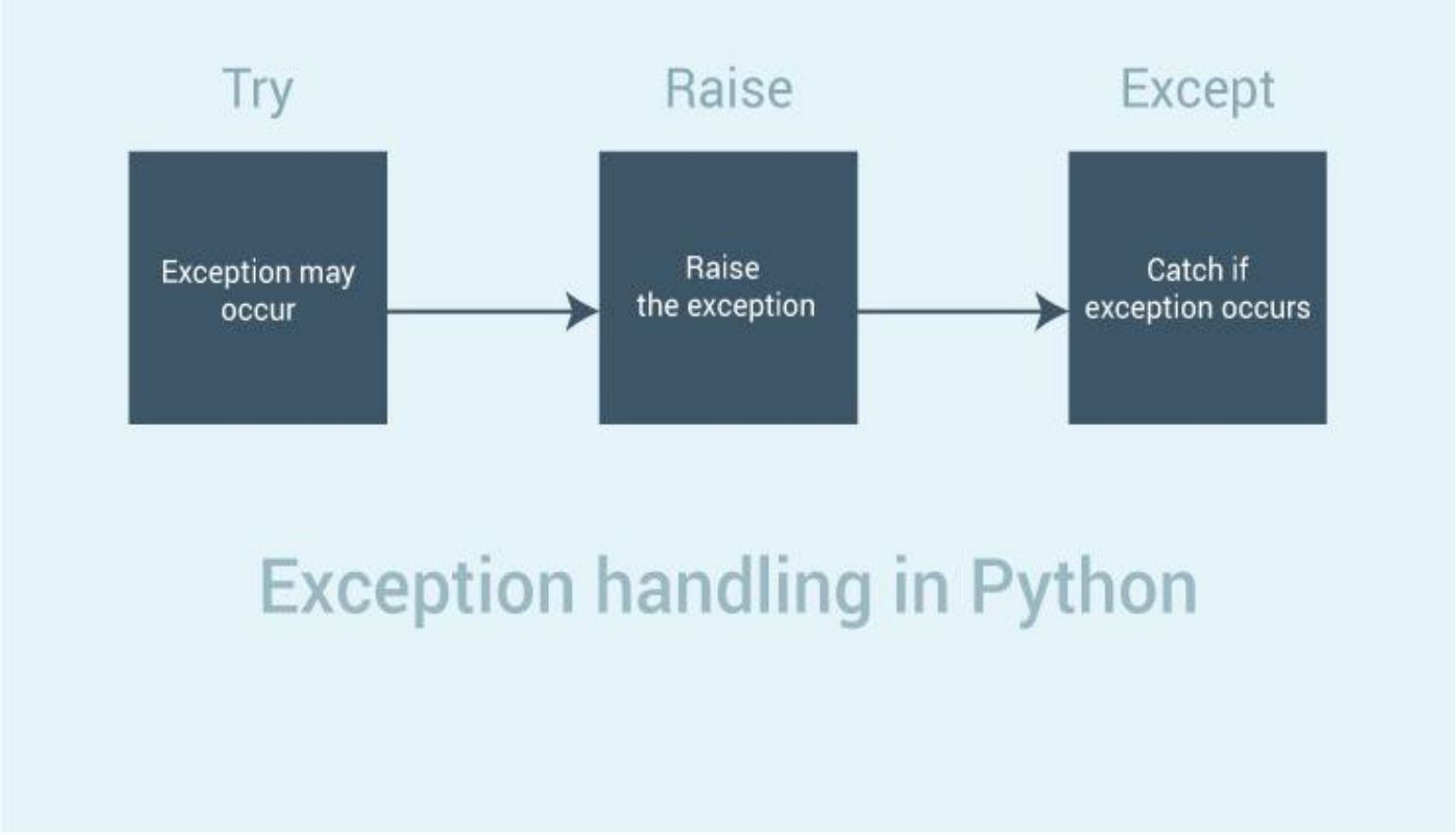
EnvironmentError	Base class for all exceptions that occur outside the Python environment.

APEC

Table 5.6 Standard Exceptions	
EXCEPTION	**DESCRIPTION**
IOError	Raised for operating system-related errors.
SyntaxError	Raised when there is an error in Python syntax.
IndentationError	Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

APC

HANDLING EXCEPTIONS



HANDLING EXCEPTIONS

The try and except statements are used to handle the runtime errors.

try:

    # lines of code that might encounter runtime error

except:

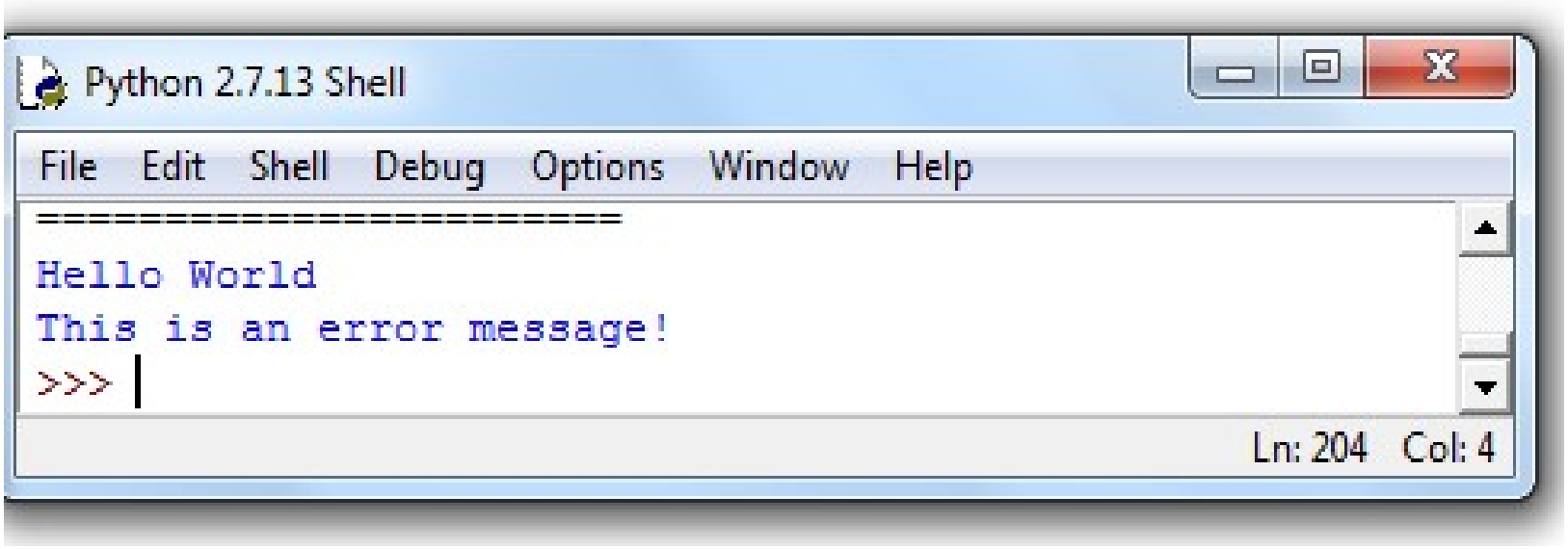
  # lines of code that will be executed when runtime error occurs

APEC

PROGRAM

File Edit Format Run Options Window Help

```
# Divide by Zero exception
try:
    print "Hello World"
    x= 10/0
    print "Never Executed"
except:
    print "This is an error message!"
```



APEC

```
# Multiple except statements
try:
    A = int(input("Please enter a no: "))
    B = int(input("Please enter a no: "))
    print ("Sum of %d and %d = %d" % (A, B, A+B))
    print ("Divison of %d by %d = %d" % (A, B, A/B))

# Action for Name Error
except NameError:
    print("The inputs must be numbers!")

# Action for Arithmetic Error
except ArithmeticError:
    print("Divide by Zero")
```

```
>>>
========================= RESTART: C:/Python27/21.py ==================
Please enter a no: 7
Please enter a no: p
The inputs must be numbers!
>>>
========================= RESTART: C:/Python27/21.py ==================
Please enter a no: 8
Please enter a no: 0
Sum of 8 and 0 = 8
Divide by Zero
>>>
```

TRY-EXCEPT-ELSE;

In the *try-except-else*; else block is executed when there is no exception but before the finally-clause.

```
try:
    # Code that might throw exceptions of type I or II
except EXCEPTION I:
    # handle_the_exception I
except EXCEPTION II:
    # handle_the_exception II
else:
    # Code that has to be executed before ending
finally:
    # Code that has to be executed always
```

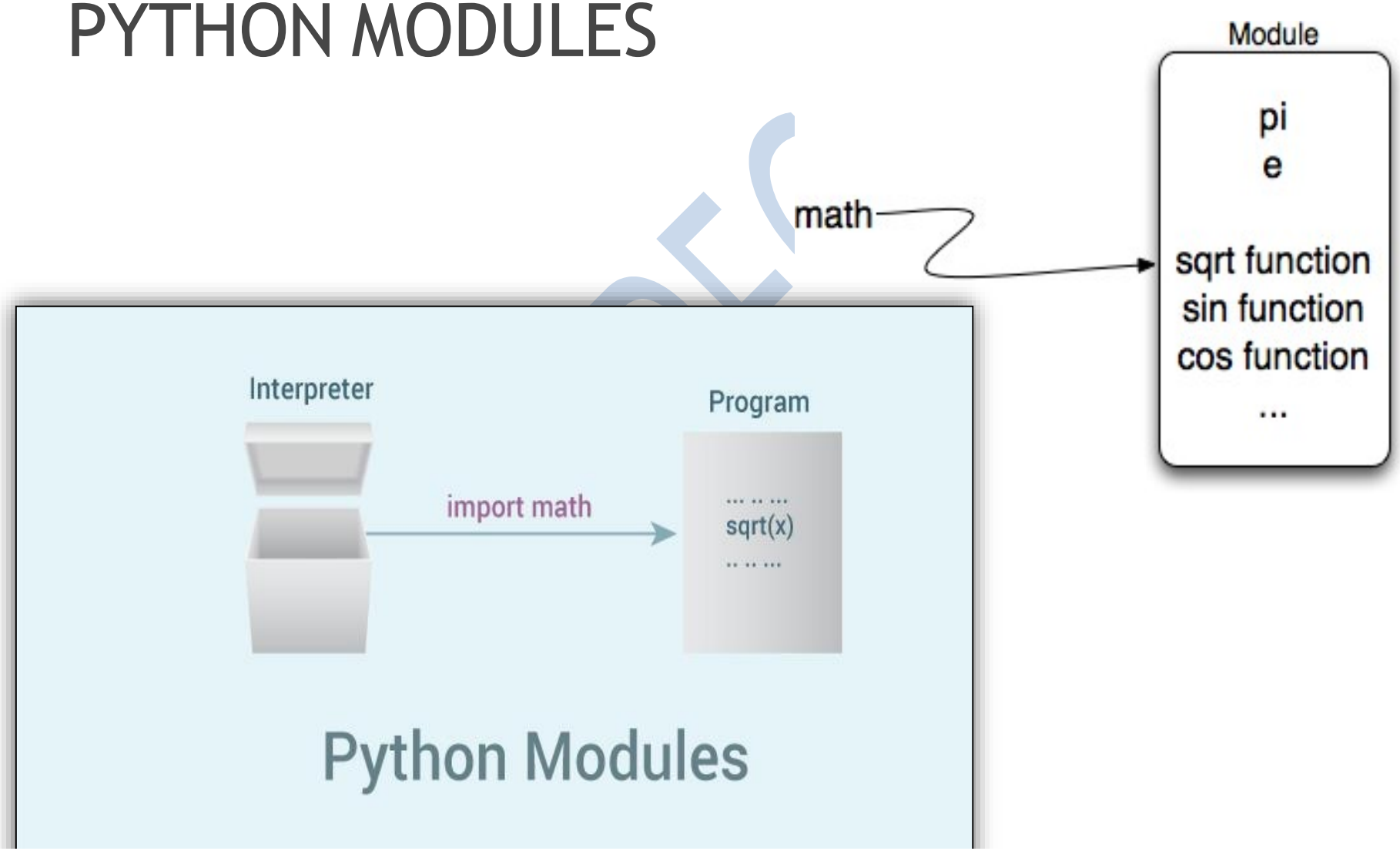
APEC

```
try:
   file_read= open("myfile.txt", "r")
   X = file_read.read()
   X = X+8
except IOError:
        print "Error: can\'t find file or read data"
except :
        print "Some other error"
else:
        print "Content of file", X
finally:
       file_read.close()
       print "Finally Close the file"
```

```
---------- RESTART: C
Some other error
Finally Close the file
>>>
```

APEC

PYTHON MODULES



PYTHON MODULES

- Module is a file containing Python definitions and statements.
- It defines functions, classes and variables and includes runnable code also.
- Modules are groups of functions.

APEC

PYTHON MODULE - STEPS

- Create a python file with one or more functions and save it with .py extension
- Include the import statement.
  - The import has the following syntax:

    □ □ 1[,□ 2[,...□□□□□□□]

  - When the interpreter comes across an import statement, it imports the module if the module is already present in the search path.
    - A search path is a list of directories that the interpreter searches before importing a module.
    - Module is loaded only once, even if multiple imports occur.
- Using the module name the function is accessed using dot (.) operation

APEC

EXAMPLE PROGRAM

Program : *example.py*

```
def add(a, b):
    """This program adds two
    numbers and return the result"""
    result = a + b
    return "Addition of two numbers", result
```

Execution

```
>>> import example
>>>example.add(89,90)
('Addition of two numbers', 179)
```

APEC

THE IMPORT STATEMENT

✓For a module involving hundreds of functions, Statement is recommended to save loading time.

Syntax: from python_file import function_name

✓Multiple functions can be imported by separating their names with commas.

Syntax: *from python_file import funtion_name1, Function_name2...*

✓Use the asterisk symbol(*) to import everything

APEC

Syntax: from python_file import*

APEC

EXAMPLE PROGRAM

Program : *example.py*

```
def add(a, b):
"""This program adds two
   numbers and return the result"""
   result = a + b
   return "Addition of two numbers", result
 def sub(a, b):
 def sub(a, b):
"""This program Subtracts two
   numbers and return the result"""
   result = a - b
   return "subtraction of two numbers", result
```

Execution

```
>>> from example import add,sub
>>> add(9,2)
('Addition of two numbers', 11)
>>> sub(9,2)
('subtraction of two numbers', 7)
```

APEC

THE IMPORT STATEMENT

- Any function in a python program can be reused in another python program by using import statement. The import has the following syntax:
- *import module1[, module2[,... moduleN]*

APEC

LOCATING MODULES

- The modules are searched in the following manner
  - First - > The current directory.
  - Second -> Each directory in the shell variable PYTHONPATH.
  - Third -> Checks the default path. For example in UNIX, this default path is normally /usr/local/lib/python/.
- The module search path is stored in the system module in the sys.path variable. The sys.path variable contains the current directory, PYTHONPATH, and the installation-dependent default. The PYTHONPATH is an environment variable, consisting of a list of directories.
- Here is a typical PYTHONPATH from a Windows system:
- *set PYTHONPATH=c:\python20\lib;*
- And here is a typical PYTHONPATH from a UNIX system:
- *set PYTHONPATH=/usr/local/lib/python*

APEC

PACKAGES IN PYTHON

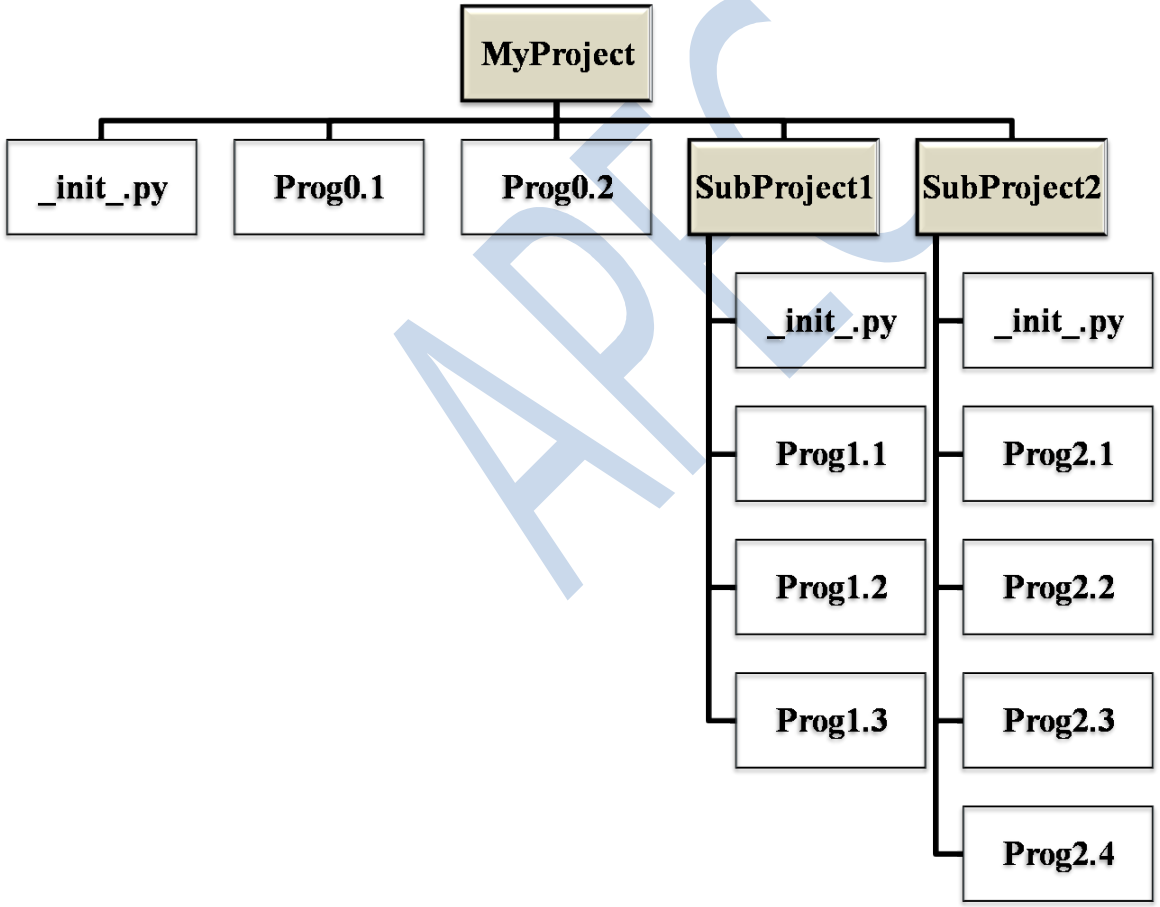
A package is a collection of modules in directories that give a package hierarchy.

- When a complex application/program is created it is better to be organized.

package gives the hierarchical file directory structure of the Python application environment that consists of modules and subpackages and sub-subpackages, and so on.

APEC

STRUCTURE OF PACKAGE



APEC

PICKLING

The write methods of the file object in Python accepts only string datatype . Any other datatype has to be converted to string before writing in the file.

All the datatypes are returned back as string only.

APEC

PROGRAM

```
1.  # Without pickle
2.  File1 = open("test.txt", 'w')
3.  #Write String
4.  File1.write('hi')
5.  #Write float
6.  File1.write(str(56.789))
7.  #Write dictionary
8.  File1.write(str({'Age':23, 'Status': 'Single'}))
9.  File1.close()
10.
11. print ("Contents of file : test.txt")
12. File1 = open("test.txt", 'r')
13. print(File1.read())
14. File1.close()
```

Execution:

```
sh-4.3$ python main.py
Contents of file :test.txt
```

APEC

hi56.789{'Status': 'Single', 'Age': 23}

APEC

CONTD…

Python consists of a standard module; pickle that preserves the data structures in the file. The files are opened as usual. However when writing and reading the following syntax is used;

For Write :pickle.dump(<datatype variable>, <filename>)

For Read :pickle.load(<filename>)

APEC

EXAMPLE

```
1.  # Use of pickle
2.  import pickle
3.  File1 = open("test.txt", 'w')
4.  #Write String
5.  pickle.dump('hi', File1)
6.  #Write float
7.  pickle.dump(56.789, File1)
8.  #Write dictionary
9.  pickle.dump({'Age':23, 'Status': 'Single'}, File1)
10. File1.close()
11.
12. print ("Contents of file : test.txt")
13. File1 = open("test.txt", 'r')
14. print(pickle.load(File1))
15. File1.close()
```

```
sh-4.3$ python main.py
Contents of file : test.txt
hi
```

APEC

56.789

{'Status': 'Single', 'Age': 23}

APEC

THANK YOU

APEC