THE NEW COLLEGE

(AN AUTONOUMOUS INSTITUTION AFFILIATED TO THE UNIVERSITY OF MADRAS
& ACCREDITED BY NAAC WITH 'A++' GRADE IN THE 4TH CYCLE)
SPONSORED BY : THE MUSLIM EDUCATIONAL ASSOCIATION OF SOUTHERN INDIA
(MEASI)

YA ALLAH
INCREASE MY KNOWLEDGE

DEPARTMENT OF COMPUTER APPLICATIONS

(B.C.A)

E-Content

Subject	**: INTERNET OF THINGS**
Class	**: III B.C.A.,**
Unit	**: III**

UNIT-3: IoT Design Methodology, IoT Systems -Logical Design using Python,Python Data Types and Data Structures, Control Flow, Functions, Modules, Packages, File Handling, Data/Time Operations. Classes. Python Packages for IoT: JSON, XML, HTTPLib and URLLib, SMTPLibIoT Systems Management with NETCONF- YANG.

3.1 IoT Design Methodology

Designing IoT systems can be a complex and challenging task as these systems involve interactions between various components. A wide range of choices are available for each component. IoT designers often tend to design the system keeping specific products in mind.

IoT Design Methodology that includes:

- Purpose & Requirements Specification
- Process Specification
- Domain Model Specification
- Information Model Specification
- Service Specifications
- IoT Level Specification
- Functional View Specification
- Operational View Specification
- Device & Component Integration
- Application Development

Step 1: Purpose & Requirements Specification
The first step in IoT system design methodology is to define the purpose and requirements of the system. In this step, the system purpose, behavior and requirements (such as data collection requirements, data analysis requirements, system management requirements, data privacy and security requirements, user interface requirements,) are captured.

Step 2: Process Specification
The second step in the IoT design methodology is to define the process specification. In this step, the use cases of the IoT system are formally described based on and derived from the purpose and requirement specifications.

Step 3: Domain Model Specification
The third step in the IoT design methodology is to define the Domain Model. The domain model describes the main concepts, entities and objects in the domain of IoT system to be designed. Domain model defines the attributes of the objects and relationships between

objects. Domain model provides an abstract representation of the concepts, objects and entities in the IoT domain, independent of any specific technology or platform. With the domain model, the IoT system designers can get an understanding of the IoT domain for which the system is to be designed.

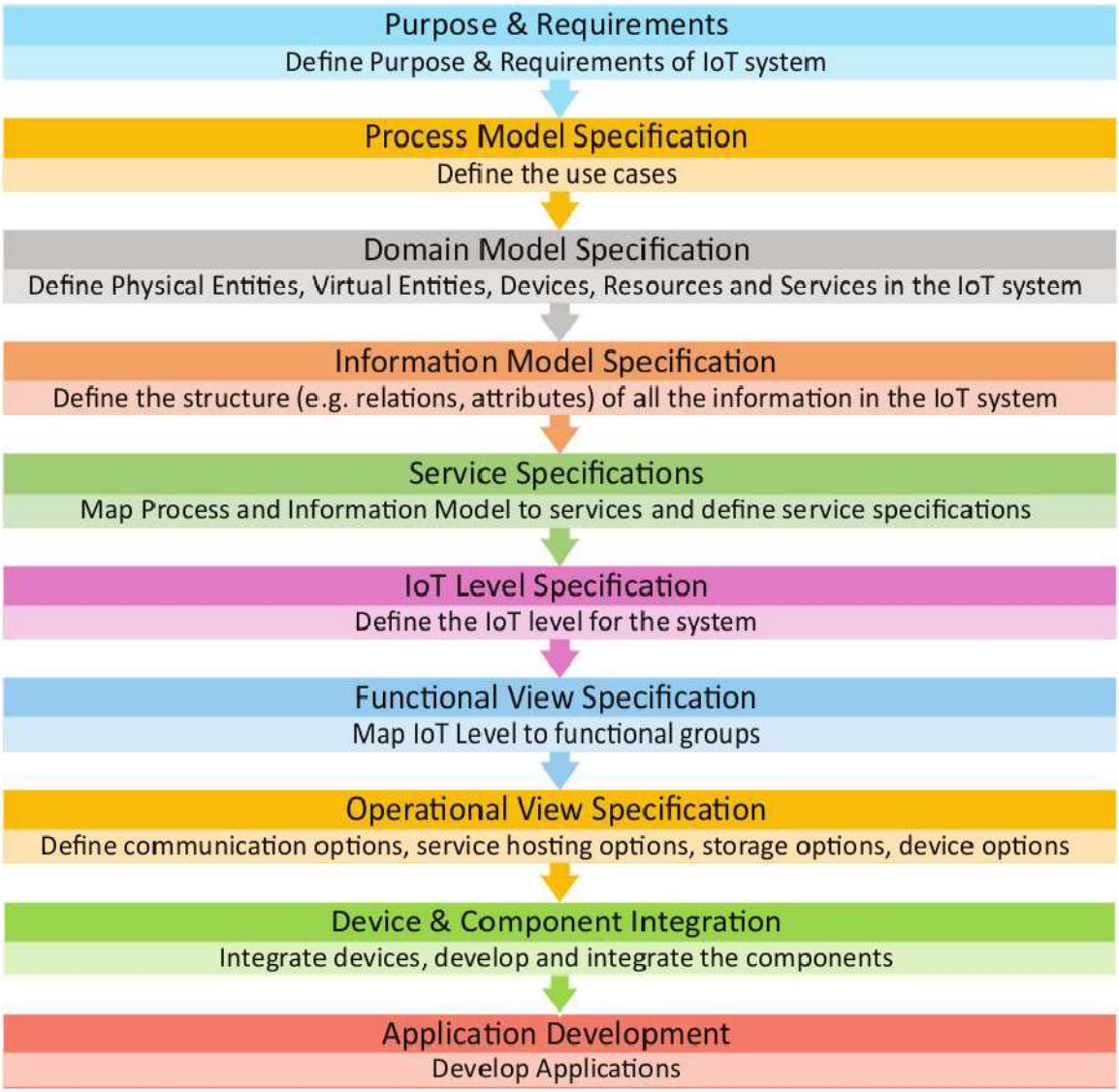

Fig 3.1: Steps involved in IoT System Design Methodology

Step 4: Information Model Specification

The fourth step in the IoT design methodology is to define the Information Model. Information Model defines the structure of all the information in the IoT system, for example, attributes of Virtual Entities, relations, etc. Information model does not describe the specifics of how the information is represented or stored. To define the information model, first listing the Virtual Entities defined in the Domain Model. Information model adds more details to the Virtual entities by defining their attributes and relations.

Step 5: Service Specifications
The fifth step in the IoT design methodology is to define the service specifications. Service specifications define the services in the IoT system, service types, service inputs/output, service endpoints, service schedules, service preconditions and service effects.

Step 6: IoT Level Specification
The sixth step in the IoT design methodology is to define the IoT level for the system.

Step 7: Functional View Specification
The seventh step in the IoT design methodology is to define the Functional View. Functional View (FV) defines the functions of the IoT systems grouped into various Functional Groups (FGs). Each Functional Group either provides functionalities for interacting with instances of concepts defined in the Domain Model or provides information related to these concepts.

Step 8: Operational View Specification
The eighth step in the IoT design methodology is to define the Operational View Specifications. In this step, various options pertaining to the IoT system deployment and operation are defined, such as, service hosting options, storage options, device options, application hosting options, etc

Step 9: Device & Component Integration
The ninth step in the IoT design methodology is the integration of the devices and components.

Step 10: Application Development
The final step in the IoT design methodology is to develop the IoT application.

3.2 IoT Systems – Logical Design using Python

3.2.1 Introduction to 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:

Multi-paradigm programming language
Python supports more than one programming paradigms including object-oriented programming and structured programming

Interpreted Language
Python is an interpreted language and does not require an explicit compilation step. The Python interpreter executes the program source code directly, statement by statement, as a processor or scripting engine does.

Interactive Language
Python provides an interactive mode in which the user can submit commands at the Python prompt and interact with the interpreter directly.

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 speeding 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.

3.2.2 Python - Setup

Windows

- Python binaries for Windows can be downloaded from http://www.python.org/getit

- 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

3.3 Python Datatypes and Data Structures

3.3.1Numbers

Number data type is used to store numeric values. Numbers are immutable data types, therefore changing the value of a number data type results in a newly allocated object.

#Integer

```
>>>a=5
>>>type(a)
<type 'int'>
```

#Floating Point

```
>>>b=2.5
>>>type(b)
<type 'float'>
```

#Long

```
>>>x=98988787876 76L
>>>type(x)
<type 'long'>
```

#Complex

```
>>>y=2+5j
```

```
>>>y
(2+5j)
>>>type(y)
<type 'complex'>

>>>y.real
2
 >>>y.imag
5

#Addition
>>>c=a+b
>>>c
7.5
>>>type(c)
<type 'float'>

#Subtraction
>>>d=a-b
>>>d
2.5
>>>type(d)
<type 'float'>

#Multiplication
>>>e=a*b
>>>e
12.5
>>>type(e)
<type 'float'>

#Division
>>>f=b/a
>>>f
0.5
>>>type(f)
<type float'>
```

#Power

```
>>>g=a**2
>>>g
25
```

3.3.2 Strings

A string is simply a list of characters in order. There are no limits to the number of characters in a string.

```
>>>s="Hello World!"
>>>type(s)
<type 'str'>
```

#String concatenation

```
>>>t="This is sample program."
>>>r = s+t
>>>r
'Hello World!This is sample program.'
```

#Get length of string

```
>>>len(s)
12
```

#Convert string to integer

```
>>>x="100"
>>>type(s)
<type 'str'>
>>>y=int(x)
>>>y
100
```

#Print string

```
>>>print s
Hello World!
```

#Formatting output

```
>>>print "The string (The string (Hello World!)
has 12 characters
```

#Convert to upper/lower case

```
>>>s.upper()
'HELLO WORLD!'
>>>s.lower()
'hello world!'
```

#Accessing sub-strings

```
>>>s[0]
'H'
>>>s[6:]
'World!'
>>>s[6:-1]
'World'
```

#strip: Returns a copy of the string with the #leading and trailing characters removed.

```
>>>s.strip("!")
'Hello World'
```

3.3.3 Lists

List a compound data type used to group together other values. List items need not all have the same type. A list contains items separated by commas and enclosed within square brackets.

#Create List

```
>>>fruits=['apple','orange','banana','mango']
>>>type(fruits)
<type 'list'>
```

#Get Length of List

```
>>>len(fruits)
4
```

#Access List Elements

```
>>>fruits[1]
'orange'
>>>fruits[1:3]
['orange', 'banana']
>>>fruits[1:]
['orange', 'banana', 'mango']
```

#Appending an item to a list

```
>>>fruits.append('pear')
>>>fruits
['apple', 'orange', 'banana', 'mango', 'pear']
```

#Removing an item from a list

```
>>>fruits.remove('mango')
>>>fruits
['apple', 'orange', 'banana', 'pear']
```

#Inserting an item to a list

```
>>>fruits.insert(1,'mango')
>>>fruits
['apple', 'mango', 'orange', 'banana', 'pear']
```

#Combining lists

```
>>>vegetables=['potato','carrot','onion','beans','radish']
>>>vegetables
['potato', 'carrot', 'onion', 'beans', 'radish']
>>>eatables=fruits+vegetables
>>>eatables
['apple','mango','orange','banana','pear', 'potato', 'carrot', 'onion', 'beans', 'radish']
```

#Mixed data types in a list

```
>>>mixed=['data',5,100.1,8287398L]
>>>type(mixed)
<type 'list'>
>>>type(mixed[0])
<type 'str'>
>>>type(mixed[1])
<type 'int'>
>>>type(mixed[2])
<type 'float'>
>>>type(mixed[3])
<type 'long'>
```

#Change individual elements of a list

```
>>>mixed[0]=mixed[0]+" items"
>>>mixed[1]=mixed[1]+1
```

```
>>>mixed[2]=mixed[2]+0.05
>>>mixed
['data items', 6, 100.14999999999999, 8287398L]

#Lists can be nested
>>>nested=[fruits,vegetables]
>>>nested
[['apple', 'mango', 'orange', 'banana', 'pear'],
['potato', 'carrot', 'onion', 'beans', 'radish']]
```

3.3.4 Tuples

A tuple is a sequence data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed within parentheses. Unlike lists, the elements of tuples cannot be changed, so tuples can be thought of as read-only lists.

```
#Create a Tuple
>>>fruits=("apple","mango","banana","pineapple")
>>>fruits
('apple', 'mango', 'banana', 'pineapple')
>>>type(fruits)
<type 'tuple'>

#Get length of tuple
>>>len(fruits)
4

#Get an element from a tuple
>>>fruits[0]
'apple'
>>>fruits[:2]
('apple', 'mango')
#Combining tuples
>>>vegetables=('potato','carrot','onion','radish')
>>>eatables=fruits+vegetables
>>>eatables
('apple', 'mango', 'banana', 'pineapple', 'potato', 'carrot', 'onion', 'radish')
```

3.3.5 Dictionaries

Dictionary is a mapping data type or a kind of hash table that maps keys to values. Keys in a dictionary can be of any data type, though numbers and strings are commonly used for keys. Values in a dictionary can be any data type or object.

#Create a dictionary

```
>>>student={'name':'Mary','id':'8776','major':'CS'}
>>>student
{'major': 'CS', 'name': 'Mary', 'id': '8776'}
>>>type(student)
<type 'dict'>
```

#Get length of a dictionary

```
>>>len(student)
3
```

#Get the value of a key in dictionary

```
>>>student['name']
'Mary'
```

#Get all items in a dictionary

```
>>>student.items()
[('gender', 'female'), ('major', 'CS'), ('name', 'Mary'),
('id', '8776')]
```

#Get all keys in a dictionary

```
>>>student.keys()
['gender', 'major', 'name', 'id']
```

#Get all values in a dictionary

```
>>>student.values()
['female', 'CS', 'Mary', '8776']
```

#Add new key-value pair

```
>>>student['gender']='female'
>>>student
{'gende
r': 'female', 'major': 'CS', 'name': 'Mary', 'id': '8776'}
```

#A value in a dictionary can be another dictionary

```
>>>student1={'name':'David','id':'9876','major':'ECE'}
>>>students={'1': student,'2':student1}
>>>students
{'1':
{'gende
r': 'female', 'major': 'CS', 'name': 'Mary', 'id': '8776'}, '2':
{' major': 'ECE', 'name': 'David', 'id': '9876'}}
```

#Check if dictionary has a key

```
>>>student.has_key('name')
True
>>>student.has_key('grade')
False
```

3.3.6 Type Conversions

Type conversion examples

```
>>>a=10000
>>>str(a)
'10000'
```

#Convert to int

```
>>>b="2013"
>>>int(b)
2013
```

#Convert to float

```
>>>float(b)
2013.0
```

#Convert to long

```
>>>long(b)
2013L
```

#Convert to list

```
>>>s="aeiou"
>>>list(s)
['a', 'e', 'i', 'o', 'u']
```

#Convert to set

```
>>>x=['mango','apple','banana','mango','banana']
```

```
>>>set(x)
set([’mango’, ’apple’, ’banana’])
```

3.4 Control Flow – if statement

The *if* statement in Python is similar to the *if* statement in other languages.

```
>>>if a>10000:
print "More"
else:
print "Less"
More

>>>if a>10000:
if a<1000000:
print "Between 10k and 100k"
else:
print "More than 100k"
elif a==10000:
print "Equal to 10k"
else:
print "Less than 10k"

More than 100k

>>>s="Hello World"
>>>if "World" in s:
s=s+"!"
print s
Hello World!
```

3.5 for statement

• The *for* statement in Python iterates over items of any sequence (list, string, etc.) in the order in which they appear in the sequence.

• This behavior is different from the *for* statement in other languages such as C in which an initialization, incrementing and stopping criteria are provided.

#Looping over characters in a string

```
helloString = "Hello World"
for c in helloString:
print c
```

#Looping over items in a list

```
fruits=[’apple’,’orange’,’banana’,’mango’]
i=0
for item in fruits:
print "Fruit-%d: %s" % (i,item)
i=i+1
```

#Looping over keys in a dictionary

```
student
=
’nam
e’:
’Mar
y’, ’id’: ’8776’,’gender’: ’female’, ’major’: ’CS’
for key in student:
print "%s: %s" % (key,student[key]
```

while statement

The while statement in Python executes the statements within the while loop as long as the while condition is true.

#Prints even numbers upto 100

```
>>> i = 0
>>> while i<=100:
if i%2 == 0:
print i
i = i+1
```

range statement

The range statement in Python generates a list of numbers in arithmetic progression.

#Generate a list of numbers from 0 – 9

```
>>>range (10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

#Generate a list of numbers from 10 - 100 with increments of 10

```
>>>range(10,110,10)
[10, 20, 30, 40, 50, 60, 70, 80, 90,100]
```

break/continue statements

The *break* and *continue* statements in Python are similar to the statements in C.

break

break statement breaks out of the for/while loop

#Break statement example

```
>>>y=1
>>>for x in range(4,256,4):
        y = y * x
        if y > 512:
            break
            print y
43
2
384
```

Continue

Continue statement continues with the next iteration.

#Continue statement example

```
>>>fruits=['apple','orange','banana','mango']
>>>for item in fruits:
        if item == "banana":
            continue
        else:
            print item
apple
orange
mango
```

pass statement

The *pass* statement in Python is a null operation. The *pass* statement is used when a statement is required syntactically but you do not want any command or code to execute.

#pass statement example

```
>fruits=[’apple’,’orange’,’banana’,’mango’]
>for item in fruits:
        if item == "banana":
            pass
        else:
            print item
```

apple
orange
mango

Functions

A function is a block of code that takes information in (in the form of parameters), does some computation, and returns a new piece of information based on the parameter information.

• A function in Python is a block of code that begins with the keyword *def* followed by the function name and parentheses. The function parameters are enclosed within the parenthesis.

• The code block within a function begins after a colon that comes after the parenthesis enclosing the parameters.

• The first statement of the function body can optionally be a documentation string or docstring.

```
students = { '1': {'name': 'Bob', 'grade': 2.5},
'2': {'name': 'Mary', 'grade': 3.5},
'3': {'name': 'David', 'grade': 4.2},
'4': {'name': 'John', 'grade': 4.1},
'5': {'name': 'Alex', 'grade': 3.8}}
```

```
def averageGrade(students):
        “This function computes the average grade”
        sum = 0.0
        for key in students:
            sum = sum + students[key]['grade']
            average = sum/len(students)
        return average
```

```
avg = averageGrade(students)
print "The average garde is: %0.2f" % (avg)
```

Functions - Default Arguments

Functions can have default values of the parameters. If a function with default values is called with fewer parameters or without any parameter, the default values of the parameters are used

```
>>>def displayFruits(fruits=['apple','orange']):
        print "There are %d fruits in the list" % (len(fruits))
        for item in fruits:
            print item
```

#Using default arguments

```
>>>displayFruits()
apple
orange

>>>fruits = ['banana', 'pear', 'mango']
>>>displayFruits(fruits)
banana
pear
mango
```

Functions - Passing by Reference

All parameters in the Python functions are passed by reference. If a parameter is changed within a function the change also reflected back in the calling function.

```
>>>def displayFruits(fruits):
        print "There are %d fruits in the list" % (len(fruits))
        for item in fruits:
            print item
            print "Adding one more fruit"
            fruits.append('mango')

>>>fruits = ['banana', 'pear', 'apple']
>>>displayFruits(fruits)
```

```
There are 3 fruits in the list
banana
pear
apple

#Adding one more fruit
>>>print "There are %d fruits in the list" % (len(fruits))
There are 4 fruits in the list
```

Functions - Keyword Arguments

Functions can also be called using keyword arguments that identifies the arguments by the parameter name when the function is called.

```
>>>def printStudentRecords(name,age=20,major='CS'):
        print "Name: " + name
        print "Age: " + str(age)
        print "Major: " + major

#This will give error as name is required argument
>>>printStudentRecords()
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: printStudentRecords() takes at least 1
argument (0 given)

#Correct use
>>>printStudentRecords(name='Alex')
Name: Alex
Age: 20
Major: CS
>>>printStudentRecords(name='Bob',age=22,major='ECE')
Name: Bob
Age: 22
Major: ECE
>>>printStudentRecords(name='Alan',major='ECE')
Name: Alan
Age: 20
Major: ECE
```

#name is a formal argument.
#kwargs is a keyword argument that receives all arguments except the formal argument as a dictionary.**

```
>>>def student(name, **kwargs):
        print "Student Name: " + name
            for key in kwargs:
            print key + ': ' + kwargs[key]

>>>student(name='Bob', age='20', major = 'CS')
Student Name: Bob
age: 20
major: CS
```

Functions - Variable Length Arguments

Python functions can have variable length arguments. The variable length arguments are passed to as a tuple to the function with an argument prefixed with asterix (*)

```
>>>def student(name, *varargs):
        print "Student Name: " + name
        for item in varargs:
            print item

>>>student('Nav')
Student Name: Nav
>>>student('Amy', 'Age: 24')
Student Name: Amy
Age: 24
>>>student('Bob', 'Age: 20', 'Major: CS')
Student Name: Bob
Age: 20
Major: CS
```

Modules

Python allows organizing the program code into different modules which improves the code readability and management.

A module is a Python file that defines some functionality in the form of functions or classes.
Modules can be imported using the import keyword.
Modules to be imported must be present in the search path.

```
#student module - saved as student.py
def averageGrade(students):
        sum = 0.0
        for key in students:
            sum = sum + students[key]['grade']
            average = sum/len(students)
        return average

def printRecords(students):
        print "There are %d students" %(len(students))
        i=1
        for key in students:
            print "Student-%d: " % (i)
            print "Name: " + students[key]['name']
            print "Grade: " + str(students[key]['grade'])
            i = i+1

# Importing a specific function from a module
>>>from student import averageGrade

# Listing all names defines in a module
>>>dir(student)

#Using student module
>>>import student
>>>students = '1': 'name': 'Bob', 'grade': 2.5,
'2': 'name': 'Mary', 'grade': 3.5,
'3': 'name': 'David', 'grade': 4.2,
'4': 'name': 'John', 'grade': 4.1,
'5': 'name': 'Alex', 'grade': 3.8

>>>student.printRecords(students)
There are 5 students
Student-1:
Name: Bob
Grade: 2.5
Student-2:
Name: David
Grade: 4.2
```

Student-3:
Name: Mary
Grade: 3.5
Student-4:
Name: Alex
Grade: 3.8
Student-5:
Name: John
Grade: 4.1
>>>avg = student. averageGrade(students)
>>>print "The average garde is: %0.2f" % (avg)
3.62

Packages

Python package is hierarchical file structure that consists of modules and sub packages. Packages allow better organization of modules related to a single application environment.

 # skimage package listing
skimage/ **Top level package**
__init__.py **Treat directory as a package**

color/ color **color subpackage**
__init__.py
colorconv.py
colorlabel.py
rgb_colors.py

draw/ draw **draw subpackage**
__init__.py
draw.py
setup.py
exposure/ **exposure subpackage**
__init__.py
_adapthist.py
exposure.py
feature/ **feature subpackage**
__init__.py
_brief.py
_daisy.py

...

File Handling

Python allows reading and writing to files using the file object.

• The open(filename, mode) function is used to get a file object.

• The mode can be read (r), write (w), append (a), read and write (r+ or w+), read-binary (rb), write-binary (wb), etc.

• After the file contents have been read the close function is called which closes the file object.

 # Example of reading an entire file

```
>>>fp = open('file.txt','r')
>>>content = fp.read()
>>>print content
This is a test file.
>>>fp.close()
```

 # Example of reading lines in a loop

```
>>>fp = open('file1.txt','r')
>>>lines = fp.readlines()
>>>for line in lines:
        print(line)
```

Python supports more than one programming paradigms.

Python is an interpreted language.