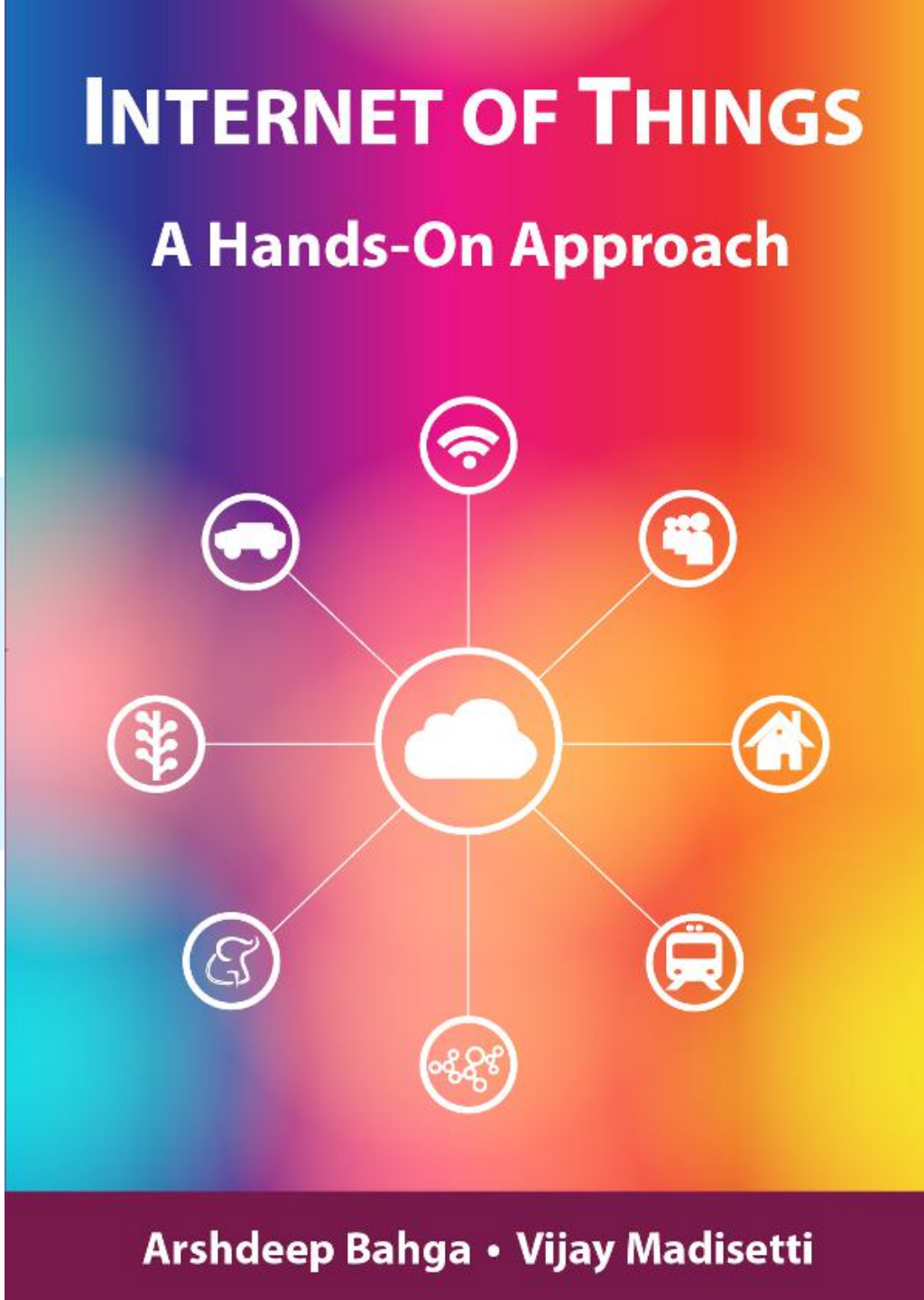
Chapter 6

IoT Systems – Logical Design using Python



Outline

- Introduction to Python
- Installing Python
- Python Data Types & Data Structures
- Control Flow
- Functions
- Modules
- Packages
- File Input/Output
- Date/Time Operations
- Classes

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.

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
```

Numbers

- Numbers
  - 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=9898878787676L
>>>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
```

Strings

- Strings
  - A string is simply a list of characters in order. There are no limits to the number of characters you can have in a string.

```
#Create 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'
```

Lists

- 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','r
adish']
>>>vegetables
['potato', 'carrot', 'onion', 'beans', 'radish']

>>>eatables=fruits+vegetables
>>>eatables
['appl
e',
'mang
o',
'orang
e',
'banan
a',
'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']]
```

Tuples

- 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')
```

Dictionaries

- 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
```

Type Conversions

- Type conversion examples

```
#Convert to string
>>>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'])
```

Control Flow – if statement

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

```
>>>a = 25**5
>>>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!
```

```
>>>student={'name':'Mary','id':'8776'}
>>>if not student.has_key('major'):
        student['major']='CS'

>>>student
{'major': 'CS', 'name': 'Mary', 'id': '8776'}
```

Control Flow – 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]
```

Control Flow – 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
```

Control Flow – 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]
```

Control Flow – 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
4
32
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
```

Control Flow – 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.

```
>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='EC
E')
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 subpackages.
- 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 line by line

```
>>>fp = open('file1.txt','r')
>>>print "Line-1: " + fp.readline()
Line-1: Python supports more than one programming paradigms.
>>>print "Line-2: " + fp.readline()
Line-2: Python is an interpreted language.
>>>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.
```

File Handling

 # Example of reading a certain number of bytes

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>fp.close()
```

 # Example of seeking to a certain position

```
>>>fp = open('file.txt','r')
>>>fp.seek(10,0)
>>>content = fp.read(10)
>>>print content
ports more
>>>fp.close()
```

 # Example of getting the current position of read

```
>>>fp = open('file.txt','r')
>>>fp.read(10)
'Python sup'
>>>currentpos = fp.tell
>>>print currentpos
<built-in method tell of file object at 0x0000000002391390>
>>>fp.close()
```

 # Example of writing to a file

```
>>>fo = open('file1.txt','w')
>>>content='This is an example of writing to a file in
Python.'
>>>fo.write(content)
>>>fo.close()
```

Date/Time Operations

- Python provides several functions for date and time access and conversions.
- The datetime module allows manipulating date and time in several ways.
- The time module in Python provides various time-related functions.

 # Examples of manipulating with date

```
>>>from datetime import date

>>>now = date.today()
>>>print "Date: " + now.strftime("%m-%d-%y")
Date: 07-24-13

>>>print "Day of Week: " + now.strftime("%A")
Day of Week: Wednesday

>>>print "Month: " + now.strftime("%B")
Month: July

>>>then = date(2013, 6, 7)
>>>timediff = now - then
>>>timediff.days
47
```

 # Examples of manipulating with time

```
>>>import time
>>>nowtime = time.time()
>>>time.localtime(nowtime)
time.struct_time(tm_year=2013, tm_mon=7, tm_mday=24, tm_ec=51, tm_wday=2, tm_yday=205,
tm_isdst=0)

>>>time.asctime(time.localtime(nowtime))
'Wed Jul 24 16:14:51 2013'

>>>time.strftime("The date is %d-%m-%y. Today is a %A. It is %H hours, %M minutes and %S seconds now.")
'The date is 24-07-13. Today is a Wednesday. It is 16 hours, 15 minutes and 14 seconds now.'
```

Classes

- Python is an Object-Oriented Programming (OOP) language. Python provides all the standard features of Object Oriented Programming such as classes, class variables, class methods, inheritance, function overloading, and operator overloading.
- Class
  - A class is simply a representation of a type of object and user-defined prototype for an object that is composed of three things: a name, attributes, and operations/methods.
- Instance/Object
  - Object is an instance of the data structure defined by a class.
- Inheritance
  - Inheritance is the process of forming a new class from an existing class or base class.
- Function overloading
  - Function overloading is a form of polymorphism that allows a function to have different meanings, depending on its context.
- Operator overloading
  - Operator overloading is a form of polymorphism that allows assignment of more than one function to a particular operator.
- Function overriding
  - Function overriding allows a child class to provide a specific implementation of a function that is already provided by the base class. Child class implementation of the overridden function has the same name, parameters and return type as the function in the base class.

Class Example

- The variable *studentCount* is a class variable that is shared by all instances of the class Student and is accessed by *Student.studentCount*.
- The variables *name, id* and *grades* are instance variables which are specific to each instance of the class.
- There is a special method by the name __init__() which is the class constructor.
- The class constructor initializes a new instance when it is created. The function __del__() is the class destructor

```
# Examples of a class
class Student:
        studentCount = 0

        def __init__(self, name, id):
                print "Constructor called"
                self.name = name
                self.id = id
                Student.studentCount = Student.studentCount + 1
                self.grades={}

        def __del__(self):
                print "Destructor called"

        def getStudentCount(self):
                return Student.studentCount

        def addGrade(self,key,value):
                self.grades[key]=value
                def getGrade(self,key):
                return self.grades[key]

        def printGrades(self):
                for key in self.grades:
                        print key + ": " + self.grades[key]
```

```
>>>s = Student('Steve','98928')
Constructor called

>>>s.addGrade('Math','90')
>>>s.addGrade('Physics','85')
>>>s.printGrades()
Physics: 85
Math: 90

>>>mathgrade = s.getGrade('Math')
>>>print mathgrade
90

>>>count = s.getStudentCount()
>>>print count
1

>>>del s
Destructor called
```

Class Inheritance

- In this example Shape is the base class and Circle is the derived class. The class Circle inherits the attributes of the Shape class.
- The child class Circle overrides the methods and attributes of the base class (eg. draw() function defined in the base class Shape is overridden in child class Circle).

```
# Examples of class inheritance
class Shape:
        def __init__(self):
                print "Base class constructor"
                self.color = 'Green'
                self.lineWeight = 10.0

        def draw(self):
                print "Draw - to be implemented"
                def setColor(self, c):
                self.color = c
                def getColor(self):
                return self.color

        def setLineWeight(self,lwt):
                self.lineWeight = lwt

        def getLineWeight(self):
                return self.lineWeight
```

```
class Circle(Shape):
        def __init__(self, c,r):
                print "Child class constructor"
                self.center = c
                self.radius = r
                self.color = 'Green'
                self.lineWeight = 10.0
                self.__label = 'Hidden circle label'

        def setCenter(self,c):
                self.center = c
                def getCenter(self):
                return self.center

        def setRadius(self,r):
                self.radius = r

        def getRadius(self):
                return self.radius

        def draw(self):
                print "Draw Circle (overridden function)"
```

```
class Point:
        def __init__(self, x, y):
                self.xCoordinate = x
                self.yCoordinate = y

        def setXCoordinate(self,x):
                self.xCoordinate = x

        def getXCoordinate(self):
                return self.xCoordinate

        def setYCoordinate(self,y):
                self.yCoordinate = y

        def getYCoordinate(self):
                return self.yCoordinate
```

```
>>>p = Point(2,4)
>>>circ = Circle(p,7)
Child class constructor
>>>circ.getColor()
'Green'
>>>circ.setColor('Red')
>>>circ.getColor()
'Red'
>>>circ.getLineWeight()
10.0
>>>circ.getCenter().getXCoordinate()
2
>>>circ.getCenter().getYCoordinate()
4
>>>circ.draw()
Draw Circle (overridden function)
>>>circ.radius
7
```

Further Reading

- Code Academy Python Tutorial, http://www.codecademy.com/tracks/python
- Google's Python Class, https://developers.google.com/edu/python/
- Python Quick Reference Cheat Sheet, http://www.addedbytes.com/cheat-sheets/python-cheat-sheet/
- PyCharm Python IDE, http://www.jetbrains.com/pycharm/