Python Notes

August 7, 2019

PS05EMCA22 Introduction to Data Science and Big Data (Python Part)

Dr. J. V. Smart

Syllabus

COURSE NO: PS05EMCA22

Introduction to Data Science and Big Data

w.e.f. June 2019

(3 Lectures & 1 Seminar/Tutorial per Week Total Marks: 100)

COURSE CONTENT:

1. **Introduction to Data Science Data Analytics**
   - Data Science Definition
   - Need and features
   - Importance of Data Science in Modern Business
   - Current Trends in Data Science
   - Analytical Techniques
2. **Introduction to Big Data and Big Data Analytics**
   - Types of Digital Data: Unstructured, Semi-structured and Structured
   - Working with Unstructured Data
   - Evolution and Definition of Big Data
   - Characteristics and Need of Big Data
   - Meaning and Characteristics of Big Data Analytics
   - Need of Big Data Analytics
   - Classification of Analytics
   - Importance of Big Data Analytics
3. **Introduction to the Python Programming Language**
   - Important characteristics of Python, key success factors, major application areas
   - Data types, syntax, control structures
   - Strings, input, output, basic file handling

- Lists and dictionaries
- Functions
- Classes and object-oriented programming
- Exception handing

4. **Data Analytics using Python and R**

   - Introduction to NumPy, SciPy
   - Introduction to pandas
   - Introduction to Matplotlib
   - Introduction to R
   - Introduction to R Studio
   - Developing data science applications using Python and R

MAIN REFERENCE BOOKS:

1. Davy Cielen, Arno D.B. Meysman, Mohamed Ali, Introducing Data Science: Big Data, Machine Learning and More, Using Python Tools
2. Seema Acharya, Subhashini Chellappan, Big Data and Analytics, Wiley
3. VigneshPrajapati, Big Data Analytics with R and Hadoop – Packrt
4. Mark Lutz, "Learning Python", 4 th Edition, O'Reilly, 2009
5. Wes McKinney, "Python for Data Analysis", O'Reilly, 2013
6. Robert I. Kabacoff, "R in Action: Data Analysis and Graphics with R", Manning, 2011

BOOKS FOR ADDITIONAL READING:

1. Akerkar R.A. and Sajja, P.S. "Intelligent techniques for data science", Springer International Publishing, Switzerland, august 2016
2. Minelli, Chambers, Dhiray, Big Data Big Analytics, Wiley
3. Bart Baesens, Analytics in a Big Data World , Wiley
4. Thomas Erl, Wajid Khattak, and Paul Buhler, Big data Fundamentals: Concepts, Drives, and Techniques, , Pearson India Education Services Pvt. Ltd., 2016
5. Roger D. Peng and Elizabeth Matsui, The Art of Data Science: A Guide for Anyone Who Works with Data, LeanPub, 2016
6. Brian Caffo, Roger D. Peng and Jeffrey Leek, Executive Data Science A Guide to Training and Managing the Best Data Scientists, LeanPub, 2016
7. Alex Holmes Hadoop in Practice – Dreamtech
8. Documentation of relevant software packages
9. Other web references

Introduction to the Python Programming Language

The Python Logo and Wordmark

The Python programming language was created by Guido van Rossum in 1991. It is named after the British comedy group Monty Python.

- **Key Characteristics**
  - Dynamically typed general purpose programming language
  - Usually interpreted

The Python Logo and Wordmark

  - Garbage collection
  - Cross platform
  - Supports multiple programming paradigms
  - Structured programming and object-oriented programming are fully supported. It also has functional programming features

- **Key Characteristics**
  - Use of white space indentation to mark blocks instead of curly braces or keywords. This is because incorrectly indented code can be understood by human reader differently than does a compiler or interpreter
  - Case sensitive
  - Two major versions currently in widespread use: 2.x and 3.x
  - Support for modules / packages
  - Large number of packages available
  - REPL (Read-Eval-Print Loop) - interactive shell

- **Why Do People Use Python?**
  - **Easy to Learn and to Get Started** Python has a simple syntax that is easy to learn. One can start coding in Python very quickly
  - **Concise Code** Python code is very compact. It takes less lines of code and less time to code algorithms in Python as compared to statically-typed languages
  - **Portability** Most Python programs run unchanged on major computing platforms. Python is supported on a wide range of hardware devices and operating systems ranging from small IoT devices and embedded systems to supercomputers
  - **Software Quality** The Python programming language is designed to be readable. One can understand Python code easily, even if it is written by someone else. Python also supports code reuse mechanisms like modules and object-oriented programming
  - **Extensibility** Python has a large collection of modules that are part of the standard library. The standard library covers all the frequently needed functionality. Fans of Python call this the *batteries included* approach. in additions, thousands of more modules are available in the official repository PyPI (Python Package Index)
  - **Free and Open Source** Python is free and open source. A large number of packages for Python are also free and open source
  - **Community Support** There is a large and devoted community of Python users who are often willing to help others and share their work with others
  - **Integration with Other Software Platforms** Python can integrate well with other leading software development platforms, including C, C++, Java, .NET, MATLAB, etc. It can also easily interface with other hardware through different types of ports

  - **Popularity among non-programmers** Because of the above benefits, Python has become popular among people working in non-IT fields who need to write computer programs to solve problems in their field

- **Major Application Areas for Python**
  - **Systems Programming**
  - **System Administration**
  - **GUI Software Development** using toolkits such as *Tkinter*, *wxPython*, etc.
  - **Server-side Web Development** using frameworks such as Django, Flask, etc.
  - **Component Integration** Python can be integrated with components developed using other software stacks
  - **Embedded Systems and IoT**
  - **Database Programming**
  - **Rapid Prototyping** Python development can be fast. As a result, Python id often used for developing first-cut prototypes of full-fledged applications
  - **Game Development** Simple games can be developed using Python quite easily using libraries like Pygame
  - **AI** Python has become very popular for coding artificial intelligence application
  - **Data Science** Python and various Python packages are heavily used for processing, analyzing and visualizing large amounts of data

Syntax

Comments

- **Single-line Comments** In any line, everything from the first # character up to the end of the line is a comment
- **Multi-line Comments** There are no multi-line comments

Data types

- **None** Like null in several other programming languages
- **Numbers**
  - **int** Virtually unlimited precision signed integer number
  - **float** Double precision floating point number
- **Strings**
  - **str**
- **Boolean** There is no boolean data type. But there is a class **bool**, which is a subclass of class **int**. There are only two objects of class **bool** :**True** and **False**
- **Collections**
  - **List**
  - **Dictionary**
- **Functions**
  - **Positional Parameters**

  - **Keyword Parameters**
  - **Default Values**

- **Classes**
- **Modules**

Naming Conventions

- PascalCase (UpperCamelCase) for class names (first letter of every word in a multi-word name is capital)
- snake_case for everything else i.e. module names, data member names, method names, function names and variable names (only lowercase letters, words in a multi-word name separated by _ (underscore)

Python Programming by Example

Input / Output

```
In [34]: name=input('Enter your name: ')
         print('Welcome ' + name)

Enter your name: Jignesh
Welcome Jignesh
```

```
In [35]: a = input("Enter a number: ")
         print(a)
         print(int(a)+20)

Enter a number: 15
15
35
```

```
In [5]: i=10
        print(i)
        # Python does not have ++ and -- operators
        # Using ++ and -- may or may not result in an error because they are parsed as +
        + and - - respectively
        # i++
        i += 1
        print(i)
        i = i + 1
        print(i)
        # i--
        i -= 1
        print(i)
        i = i - 1
        print(i)
```

```
10
11
12
11
10
```

Control Structures

```
In [27]: a = input("Enter a number: ")
         a = int(a)
         if a>0:
             print(str(a) + " is positive")
         elif a==0:
             print("0 is zero")
         else:
             print(str(a) + " is negative")
         a=10
         b=20
         print(str(a) + ' ' + str(b))
         b, a = a, b
         print(str(a) + ' ' + str(b))
```

```
Enter a number: 12
12 is positive
10 20
20 10
```

```
In [37]: i=0
         while i < 20:
             i=i+1
             if i%3 == 0:
                 continue
             print(i, end=' ')
```

```
1 2 4 5 7 8 10 11 13 14 16 17 19 20
```

```
In [38]: print("*" * 8)
         print('GDCST\n' * 3)
```

```
********
GDCST
GDCST
GDCST
```

```
In [2]: import re
```

```
        datePattern = re.compile(r'\d\d-\d\d-\d\d\d\d')
        commentPattern = re.compile(r'^#/')
        filename='../data-files/attendance-PS05EMCA22-2019.txt'
        outputFlag = False
        with open(filename) as file_obj:
            for line in file_obj:
        #         if re.search(r'\d\d-\d\d-\d\d\d\d', line):
                if datePattern.search(line):
                    outputFlag = True
                    currentDate = line.rstrip()
                elif commentPattern.match(line) and outputFlag:
                    print(currentDate + " " + line.lstrip('#').lstrip('/'), end='')
                else:
                    outputFlag = False
```

```
17-06-2019 Introduction to Python
24-06-2019 Introduction to Python
25-06-2019 Introduction to tools...
01-07-2019 ...Introduction to tools
01-07-2019 Data types
01-07-2019 control structures
01-07-2019 Basic input and output
01-07-2019 Examples
02-07-2019 Lists...
```

Lists

```
In [40]: list1 = [1, "abc", 5, False, 7.9]
         print(list1)
         for element in list1:
             print(element, end=' ')
         print()
```

```
[1, 'abc', 5, False, 7.9]
1 abc 5 False 7.9
```

```
In [41]: list1 = []
         list1.append(3)
         list1.append('abc')
         list1.append(True)
         list1.append(5.4)
         for element in list1:
             print(element, end=' ')
         print()
         print('list1[2]=' + str(list1[2]))
         print('len(list1)=' + str(len(list1)))
         list1.append('xyz')
```

```
        print('len(list1) after append()=' + str(len(list1)))
        list1.clear()
        print('len(list1) after clear()=' + str(len(list1)))
```

```
3 abc True 5.4
list1[2]=True
len(list1)=4
len(list1) after append()=5
len(list1) after clear()=0
```

```
In [8]: list1 = [3, 'abc', True, 5.4, 'xyz', 3]
        print('Before del: ', end=' ')
        for element in list1:
            print(element, end=' ')
        print()
        del list1[3]
        print('After del list1[3]: ', end=' ')
        for element in list1:
            print(element, end=' ')
        print()
        del list1[1:3]
        print('After del list1[1:3]: ', end=' ')
        for element in list1:
            print(element, end=' ')
        print()
        print("After list1.remove(3): ", end=' ')
        list1.remove(3)
        for element in list1:
            print(element, end=' ')
        print()
```

```
Before del:  3 abc True 5.4 xyz 3
After del list1[3]:  3 abc True xyz 3
After del list1[1:3]:  3 xyz 3
After list1.remove(3):  xyz 3
```

```
In [43]: list1 = [3, 'abc', True, 5.4, 'xyz']
         print('5.4 in list1=' + str(5.4 in list1))
         print("xyz' in list1=" + str('xyz' in list1))
         print("qwe' in list1=" + str('qwe' in list1))
         x=5
         if x in list1:
             print(str(x) + ' exists in the list')
         else:
             print(str(x) + " does not exist in the list")
         if x not in list1:
             print(str(x) + " does not exist in the list")
```

```
        else:
            print(str(x) + ' exists in the list')
```

```
5.4 in list1=True
xyz' in list1=True
qwe' in list1=False
5 does not exist in the list
5 does not exist in the list
```

```
In [44]: list1 = [0, 1, 3, 5]
         list2 = [0, 2, 4, 6]
         list3 = list1 + list2
         for element in list3:
             print(element, end=' ')
         print()
```

```
0 1 3 5 0 2 4 6
```

```
In [45]: list1 = [1, 2, 3]
         list2 = list1 * 3
         for element in list2:
             print(element, end=' ')
         print()
         list1[0] = "X"
         list1.append(4)
         for element in list1:
             print(element, end=' ')
         print()
         for element in list2:
             print(element, end=' ')
         print()
```

```
1 2 3 1 2 3 1 2 3
X 2 3 4
1 2 3 1 2 3 1 2 3
```

```
In [46]: # * creates shallow copies, nested structures are not copied
         list1 = [ [1, 2], [3, 4] ]
         for element in list1:
             print(element, end=' ')
         print()
         list2 = list1 * 3
         for element in list2:
             print(element, end=' ')
         print()
         list1[0].append('X')
```

```
        for element in list1:
            print(element, end=' ')
        print()
        for element in list2:
            print(element, end=' ')
        print()
```

```
[1, 2] [3, 4]
[1, 2] [3, 4] [1, 2] [3, 4] [1, 2] [3, 4]
[1, 2, 'X'] [3, 4]
[1, 2, 'X'] [3, 4] [1, 2, 'X'] [3, 4] [1, 2, 'X'] [3, 4]
```

```
In [1]: # * creates shallow copies, nested structures are not copied
        list1 = [ [1, 2], [3, 4] ]
        for element in list1:
            print(element, end=' ')
        print()
        list2 = list1 * 3
        for element in list2:
            print(element, end=' ')
        print()
        list1.append('X')
        for element in list1:
            print(element, end=' ')
        print()
        for element in list2:
            print(element, end=' ')
        print()
```

```
[1, 2] [3, 4]
[1, 2] [3, 4] [1, 2] [3, 4] [1, 2] [3, 4]
[1, 2] [3, 4] X
[1, 2] [3, 4] [1, 2] [3, 4] [1, 2] [3, 4]
```

```
In [10]: list1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
         print('list1=', end=' ')
         print(list1)
         list2 = list1[3:6] # Slicing a list
         print('list1[3:6]=', end=' ')
         print(list2)
         list2 = list1[3:9:2] # Slicing a list with step
         print('list1[3:9:2]=', end=' ')
         print(list2)
         #Copy list
         list3 = list1[:]
         print(list3)
```

```
list1= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
list1[3:6]= [4, 5, 6]
list1[3:9:2]= [4, 6, 8]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [3]: list2 = [3, 1, 4, 1, 5, 9, 2, 6, 5]
        print('len(list2)=' + str(len(list2)))
        print('min(list2)=' + str(min(list2)))
        print('max(list2)=' + str(max(list2)))
        print(list2)
        print('list2.count(5)=' + str(list2.count(5)))
        list2.sort()
        print('After sort():', end=' ')
        print(list2)
        list2.sort(reverse=True)
        print('After sort(reverse=True):', end=' ')
        print(list2)
```

```
len(list2)=9
min(list2)=1
max(list2)=9
[3, 1, 4, 1, 5, 9, 2, 6, 5]
list2.count(5)=2
After sort(): [1, 1, 2, 3, 4, 5, 5, 6, 9]
After sort(reverse=True): [9, 6, 5, 5, 4, 3, 2, 1, 1]
```

```
In [47]: list1 = [1, 3, 5, 7, 3, 7]
         for element in list1:
             print(element, end=' ')
         print()
         x=7
         if x in list1:
             print(str(x) + ' at position ' +
                   str(list1.index(x)) + ' in the list')
         else:
             print(str(x) + ' is not in the list')
         list2 = [ 'spu', 'gdcst', 'vvn']
         for element in list2:
             print(element, end=' ')
         print()
         x='vvnagar'
         if x in list2:
             print(str(x) + ' at position ' +
                   str(list2.index(x)) + ' in the list') # Raises ValueError if not found
         else:
             print(str(x) + ' is not in the list')
```

```
1 3 5 7 3 7
7 at position 3 in the list
spu gdcst vvn
vvnagar is not in the list
```

```
In [7]: # Like for (i = 0; i < 11; i++)
        list1 = range(1,11)
        for element in list1:
            print(element, end=' ')
        print()
        # Range returns an immutable list
        # list1[3] = 12
        # del list1[2]

        # Like for (i = 0; i < 11; i++)
        list2 = range(1,11)
        for element in list2:
            print(element, end=' ')
        print()
        # Like for (i = 0; i < 11; i+=3)
        list2 = range(1, 11, 3)
        for element in list2:
            print(element, end=' ')
        print()
```

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
1 4 7 10
```

List Comprehensions

```
In [31]: list1 = range(1,11)
         for element in list1:
             print(element, end=' ')
         print()
         squares = [x**2 for x in list1]
         print(squares)
         squares2 = [x**2 for x in list1 if x%2 == 0]
         print(squares2)
         names = ['angular', 'codeigniter', 'python', 'r']
         print(names)
         upper_names = [name.capitalize() for name in names]
         print(upper_names)
         # A list of list for multiplication table
         a = 13
         table = [[a, b, a * b] for b in range(1, 11)]
```

```
        for row in table:
            print(row)
```

```
1 2 3 4 5 6 7 8 9 10
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
[4, 16, 36, 64, 100]
['angular', 'codeigniter', 'python', 'r']
['Angular', 'Codeigniter', 'Python', 'R']
[13, 1, 13]
[13, 2, 26]
[13, 3, 39]
[13, 4, 52]
[13, 5, 65]
[13, 6, 78]
[13, 7, 91]
[13, 8, 104]
[13, 9, 117]
[13, 10, 130]
```

```
In [3]: fahrenheit = [102.56, 97.7, 99.14, 100]
        celsius = [ ((x-32.0) * float(5)/9) for x in fahrenheit]
        print(celsius)
        fahrenheit = [ ((float(9)/5)*x + 32) for x in celsius ]
        print(fahrenheit)
        # Pythagorean Triplets
        triplets = [(x,y,z) for x in range(1,20) for y in range(x,30) for z in
        range(y,30) if x**2 + y**2 == z**2]
        print(triplets)
```

```
[39.2, 36.5, 37.3, 37.77777777777778]
[102.56, 97.7, 99.14, 100.0]
[(3, 4, 5), (5, 12, 13), (6, 8, 10), (7, 24, 25), (8, 15, 17), (9, 12, 15), (10,
24, 26), (12, 16, 20), (15, 20, 25)]
```

Operation	Description
`list1 = [1, 2, 3]`	Create a list with three elements
`list1 = []`	Create an empty list
`list1.append(value)`	Append an element at the end of the list
`list1.clear()`	Remove all elements from the list
`del list1[i]`	Remove an element from the list by index
`del list1[i:j]`	Remove a range of elements from the list by index (slice)
`list1.remove(value)`	Remove an element from the list by value
`value in list1`	True if value exists in the list
`value not in list1`	True if value does exist in the list

Summary of operations on lists

Operation	Description
`list3 = list1 + list2`	Join two lists
`list2 = list1 * 3`	Repeat the list three times
`list2 = list1[i:j]`	Slice through the list (like substring)
`list2 = list1[i:j:k]`	Slice through the list with step
`list2 = list1[:]`	Copy list
`len(list1)`	Length of the list
`min(list1)`	Minimum value in the list
`max(list1)`	Maximum value in the list
`list1.sort()`	Sort the list, ascending by default
`list1.sort(reverse=True)`	Sort the list in descending order
`list1.index(value)`	Index of a value in the list, if it is present, exception otherwise

Summary of operations on lists

Dictionary

A Python dictionary (`dict`) is a collection of key-value pairs.Keys may be of almost any atomic type, but are typically integers or strings. Values may be of any type, including lists dictionaries.

```
In [2]: electives = {
            'PS05EMCA22' : 'Introduction to Data Science and Big Data',
            'PS05EMCA23' : 'Web Application Frameworks'
        }
        print(electives['PS05EMCA22'])
        print(electives['PS05EMCA23'])
        print('-------------')
        electives2 = {
            22 : 'Introduction to Data Science and Big Data',
            23 : 'Web Application Frameworks'
        }
        print(electives2[22])
        print(electives2[23])

Introduction to Data Science and Big Data
Web Application Frameworks
-------------
Introduction to Data Science and Big Data
Web Application Frameworks


In [165]: mobile = {
              'CPU' : 'Qualcomm Snapdragon MSM8953 2.0 GHz 8-core (8xA53)',
```

```
            'RAM' : 4,
            'Storage' : 64,
            'GPU' : 'Adreno 506 GPU @650 MHz',
            'Screen' : '5.5" FHD (1920x1080), 401 ppi, 16:9, IPS LCD, GG3',
            'Cameras' : {
                    'Rear': [ '13MP', '13MP' ],
                    'Front': [ '8MP' ]
                    },
            'Fingerprint Sensor' : True,
            'Infrared Blaster' : False,
            'NFC' : True,
            'OS' : 'Android 8.1 Oreo',
            'Battery' : 3000,
            'Slots' : ['4G', '4G+MicroSD']
            }
        print(mobile)
```

```
{'CPU': 'Qualcomm Snapdragon MSM8953 2.0 GHz 8-core (8xA53)', 'RAM': 4,
'Storage': 64, 'GPU': 'Adreno 506 GPU @650 MHz', 'Screen': '5.5" FHD
(1920x1080), 401 ppi, 16:9, IPS LCD, GG3', 'Cameras': {'Rear': ['13MP', '13MP'],
'Front': ['8MP']}, 'Fingerprint Sensor': True, 'Infrared Blaster': False, 'NFC':
True, 'OS': 'Android 8.1 Oreo', 'Battery': 3000, 'Slots': ['4G', '4G+MicroSD']}
```

```
In [167]: import pprint
          pp = pprint.PrettyPrinter(indent=4)
          pp.pprint(mobile)
```

```
{   'Battery': 3000,
    'CPU': 'Qualcomm Snapdragon MSM8953 2.0 GHz 8-core (8xA53)',
    'Cameras': {'Front': ['8MP'], 'Rear': ['13MP', '13MP']},
    'Fingerprint Sensor': True,
    'GPU': 'Adreno 506 GPU @650 MHz',
    'Infrared Blaster': False,
    'NFC': True,
    'OS': 'Android 8.1 Oreo',
    'RAM': 4,
    'Screen': '5.5" FHD (1920x1080), 401 ppi, 16:9, IPS LCD, GG3',
    'Slots': ['4G', '4G+MicroSD'],
    'Storage': 64}
```

```
In [168]: import json
          print(json.dumps(mobile, indent=4))
```

```
{
    "CPU": "Qualcomm Snapdragon MSM8953 2.0 GHz 8-core (8xA53)",
    "RAM": 4,
    "Storage": 64,
```

```
    "GPU": "Adreno 506 GPU @650 MHz",
    "Screen": "5.5\" FHD (1920x1080), 401 ppi, 16:9, IPS LCD, GG3",
    "Cameras": {
        "Rear": [
            "13MP",
            "13MP"
        ],
        "Front": [
            "8MP"
        ]
    },
    "Fingerprint Sensor": true,
    "Infrared Blaster": false,
    "NFC": true,
    "OS": "Android 8.1 Oreo",
    "Battery": 3000,
    "Slots": [
        "4G",
        "4G+MicroSD"
    ]
}
```

```
In [169]: print('CPU=' + mobile['CPU'])
          print('Rear camera type:', end=' ')
          if len(mobile['Cameras']['Rear']) > 1:
              print('Dual')
          else:
              print('Single')
          print('Front camera type:', end=' ')
          if len(mobile['Cameras']['Front']) > 1:
              print('Dual')
          else:
              print('Single')

CPU=Qualcomm Snapdragon MSM8953 2.0 GHz 8-core (8xA53)
Rear camera type: Dual
Front camera type: Single
```

```
In [9]: students = [
            {'pid': 'MG17001', 'elective_preference': 'PS05EMCA22'},
            {'pid': 'MG17002', 'elective_preference': 'PS05EMCA23'},
            {'pid': 'LG18101', 'elective_preference': 'PS05EMCA23'},
            {'pid': 'LG18103', 'elective_preference': 'PS05EMCA22'},
            {'pid': 'MS17041', 'elective_preference': 'PS05EMCA22'},
            {'pid': 'MS17042', 'elective_preference': 'PS05EMCA23'},
            {'pid': 'LS18201', 'elective_preference': 'PS05EMCA23'},
```

```
        {'pid': 'LS18203', 'elective_preference': 'PS05EMCA22'},
    ]
    student = students[0]
    print(student['elective_preference']) # PS05EMCA22
    print(electives['PS05EMCA22']) # Introduction to Data Science and Big Data
    print(electives[student['elective_preference']]) # Introduction to Data Science
    and Big Data
    print(students[0]['pid'] + ' ' + students[0]['elective_preference'])
    print(students[0]['pid'] + ' ' + electives[students[0]['elective_preference']])
```

```
PS05EMCA22
Introduction to Data Science and Big Data
Introduction to Data Science and Big Data
MG17001 PS05EMCA22
MG17001 Introduction to Data Science and Big Data
```

```
In [17]: for student in students:
             print(student['pid'] + ' ' + electives[student['elective_preference']])
         print('---------------')
         for i in range(0, len(students)):
             print(students[i]['pid'] + ' ' +
         electives[students[i]['elective_preference']])
```

```
MG17001 Introduction to Data Science and Big Data
MG17002 Web Application Frameworks
LG18101 Web Application Frameworks
LG18103 Introduction to Data Science and Big Data
MS17041 Introduction to Data Science and Big Data
MS17042 Web Application Frameworks
LS18201 Web Application Frameworks
LS18203 Introduction to Data Science and Big Data
---------------
MG17001 Introduction to Data Science and Big Data
MG17002 Web Application Frameworks
LG18101 Web Application Frameworks
LG18103 Introduction to Data Science and Big Data
MS17041 Introduction to Data Science and Big Data
MS17042 Web Application Frameworks
LS18201 Web Application Frameworks
LS18203 Introduction to Data Science and Big Data
```

```
In [18]: print(students[0]['pid'] + ' ' + students[0]['elective_preference'])
         print(students[0]['pid'] + ' ' + electives[students[0]['elective_preference']])
         students[0]['elective_preference'] = 'PS05EMCA23'
         print(students[0]['pid'] + ' ' + students[0]['elective_preference'])
         print(students[0]['pid'] + ' ' + electives[students[0]['elective_preference']])
```

```
MG17001 PS05EMCA22
MG17001 Introduction to Data Science and Big Data
MG17001 PS05EMCA23
MG17001 Web Application Frameworks
```

```
In [24]: book1 = {
             'book_code' : 'INP948',
             'title'     : 'PROFESSIONAL ANGULAR JS',
             'authors'   : ' KARPOV,VALERI',
         }
         print(book1)
         print('-----------------')
         book1['status'] = 'Issued'
         book1['issued_to'] = 'JVS'
         print(book1)
         print('-----------------')
         del book1['status']
         del book1['issued_to']
         print(book1)
```

```
{'book_code': 'INP948', 'title': 'PROFESSIONAL ANGULAR JS', 'authors': '
KARPOV,VALERI'}
-----------------
{'book_code': 'INP948', 'title': 'PROFESSIONAL ANGULAR JS', 'authors': '
KARPOV,VALERI', 'status': 'Issued', 'issued_to': 'JVS'}
-----------------
{'book_code': 'INP948', 'title': 'PROFESSIONAL ANGULAR JS', 'authors': '
KARPOV,VALERI'}
```

```
In [34]: books = [
             { 'book_code': 'UNX300', 'title': 'PYTHON IN A NUTSHELL ', 'authors':
         'MARTELLI ALEX, RAVENSCROFT ANNA , HOLDEN STEVE', 'issued_to': 'JVS',
         'issue_date': '30-03-19' },
             { 'book_code': 'INP973', 'title': 'DATA SCIENCE AND BIG DATA COMPUTING  ',
         'authors': 'ZAIGHAM MAHMOOD', 'issued_to': 'JVS', 'issue_date': '29-04-19' },
             { 'book_code': 'INP955', 'title': 'BOOTSTRAP', 'authors': 'SPURLOCK, JAKE',
         'issued_to': 'JVS', 'issue_date': '28-03-19' },
             { 'book_code': 'INP949', 'title': 'PROGRAMMING ENTITY FRAMEWORK', 'authors':
         'LEARMAN, JULIA', 'issued_to': 'JVS', 'issue_date': '29-04-19' },
             { 'book_code': 'INP948', 'title': 'PROFESSIONAL ANGULAR JS', 'authors':
         'KARPOV,VALERI', 'issued_to': 'JVS', 'issue_date': '28-03-19' },
             { 'book_code': 'INP916', 'title': 'ANALYTICS IN A BIG DATA WORLD :THE
         ESSENTIAL GUIDE TO DATA SCIENCE AND ITS APPL.', 'authors': 'BUESENS,BART',
         'issued_to': 'JVS', 'issue_date': '29-04-19' },
             { 'book_code': 'INP902', 'title': 'ANGULAR JS: UP AND RUNNING', 'authors':
         'SESHADRI , SHYAM AND GREEN,BRAD', 'issued_to': 'JVS', 'issue_date': '28-03-19'
```

```
        },
            { 'book_code': 'INP868', 'title': 'BEGINNING R : THE STATISTICAL PROGRAMMING
        LANGUAGE', 'authors': 'GARDENER , MARK', 'issued_to': 'JVS', 'issue_date':
        '29-04-19' },
            { 'book_code': 'INP861', 'title': 'R FOR DUMMIES :A WILEY BRAND', 'authors':
        'DEVRIES, ANDRIE AND MEYS, JORIS', 'issued_to': 'JVS', 'issue_date': '29-04-19'
        },
            { 'book_code': 'INP745', 'title': 'R IN A NUTSHELL', 'authors': 'ADLER,
        JOSEPH', 'issued_to': 'JVS', 'issue_date': '29-04-19' },
            { 'book_code': 'INP1050', 'title': 'MACHINE LEARNING AND DATA MINING FOR
        COMPUTER SECURITY ', 'authors': 'MALOOF MARCUS A. ', 'issued_to': 'JVS',
        'issue_date': '24-06-19' },
            { 'book_code': 'INP1028', 'title': 'AGILE DATA SCIENCE 2.0 ', 'authors':
        'JURNEY RUSSELL ', 'issued_to': 'JVS', 'issue_date': '29-04-19' },
            { 'book_code': 'INP1012', 'title': 'THE R BOOK ', 'authors': 'CRAWLEY
        MICHAEL J. ', 'issued_to': 'JVS', 'issue_date': '29-04-19' },
        ]
        for book in books:
            print(book, end='\n\n')
        print('Total number of books: ' + str(len(books)))
```

```
{'book_code': 'UNX300', 'title': 'PYTHON IN A NUTSHELL ', 'authors': 'MARTELLI
ALEX, RAVENSCROFT ANNA , HOLDEN STEVE', 'issued_to': 'JVS', 'issue_date':
'30-03-19'}

{'book_code': 'INP973', 'title': 'DATA SCIENCE AND BIG DATA COMPUTING  ',
'authors': 'ZAIGHAM MAHMOOD', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP955', 'title': 'BOOTSTRAP', 'authors': 'SPURLOCK, JAKE',
'issued_to': 'JVS', 'issue_date': '28-03-19'}

{'book_code': 'INP949', 'title': 'PROGRAMMING ENTITY FRAMEWORK', 'authors':
'LEARMAN, JULIA', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP948', 'title': 'PROFESSIONAL ANGULAR JS', 'authors':
'KARPOV,VALERI', 'issued_to': 'JVS', 'issue_date': '28-03-19'}

{'book_code': 'INP916', 'title': 'ANALYTICS IN A BIG DATA WORLD :THE ESSENTIAL
GUIDE TO DATA SCIENCE AND ITS APPL.', 'authors': 'BUESENS,BART', 'issued_to':
'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP902', 'title': 'ANGULAR JS: UP AND RUNNING', 'authors':
'SESHADRI , SHYAM AND GREEN,BRAD', 'issued_to': 'JVS', 'issue_date': '28-03-19'}

{'book_code': 'INP868', 'title': 'BEGINNING R : THE STATISTICAL PROGRAMMING
LANGUAGE', 'authors': 'GARDENER , MARK', 'issued_to': 'JVS', 'issue_date':
'29-04-19'}
```

```
{'book_code': 'INP861', 'title': 'R FOR DUMMIES :A WILEY BRAND', 'authors':
'DEVRIES, ANDRIE AND MEYS, JORIS', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP745', 'title': 'R IN A NUTSHELL', 'authors': 'ADLER, JOSEPH',
'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP1050', 'title': 'MACHINE LEARNING AND DATA MINING FOR COMPUTER
SECURITY ', 'authors': 'MALOOF MARCUS A. ', 'issued_to': 'JVS', 'issue_date':
'24-06-19'}

{'book_code': 'INP1028', 'title': 'AGILE DATA SCIENCE 2.0 ', 'authors': 'JURNEY
RUSSELL ', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP1012', 'title': 'THE R BOOK ', 'authors': 'CRAWLEY MICHAEL J.
', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

Total number of books: 13


In [55]: book = books[0]
         print('Keys: ', end=' ')
         for key in book.keys():
             print(key, end=' ')
         print('\n')
         print('Sorted Keys: ', end=' ')
         for key in sorted(book.keys()):
             print(key, end=' ')
         print('\n')
         print('Values: ', end=' ')
         for key in book.values():
             print(key, end=' ')
         print('\n')
         for key, value in book.items():
             print(key + ': ' + value)

Keys:  book_code title authors issued_to issue_date

Sorted Keys:  authors book_code issue_date issued_to title

Values:  UNX300 PYTHON IN A NUTSHELL  MARTELLI ALEX, RAVENSCROFT ANNA , HOLDEN
STEVE JVS 30-03-19

book_code: UNX300
title: PYTHON IN A NUTSHELL
authors: MARTELLI ALEX, RAVENSCROFT ANNA , HOLDEN STEVE
issued_to: JVS
issue_date: 30-03-19
```

```
In [62]: # List comprehension on list of dictionaries
         data_books = [ book for book in books if 'DATA' in book['title']]
         print('Data Books:\n-----------')
         for data_book in data_books:
             print(data_book, end='\n\n')
         jvs_books = [ book for book in books if 'JVS' in book['issued_to']]
         print('Books Issued to JVS:\n--------------------')
         for jvs_book in jvs_books:
             print(jvs_book, end='\n\n')
         print('count=' + str(len(jvs_books)))
```

```
Data Books:
-----------
{'book_code': 'INP973', 'title': 'DATA SCIENCE AND BIG DATA COMPUTING  ',
'authors': 'ZAIGHAM MAHMOOD', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP916', 'title': 'ANALYTICS IN A BIG DATA WORLD :THE ESSENTIAL
GUIDE TO DATA SCIENCE AND ITS APPL.', 'authors': 'BUESENS,BART', 'issued_to':
'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP1050', 'title': 'MACHINE LEARNING AND DATA MINING FOR COMPUTER
SECURITY ', 'authors': 'MALOOF MARCUS A. ', 'issued_to': 'JVS', 'issue_date':
'24-06-19'}

{'book_code': 'INP1028', 'title': 'AGILE DATA SCIENCE 2.0 ', 'authors': 'JURNEY
RUSSELL ', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

Books Issued to JVS:
--------------------
{'book_code': 'UNX300', 'title': 'PYTHON IN A NUTSHELL ', 'authors': 'MARTELLI
ALEX, RAVENSCROFT ANNA , HOLDEN STEVE', 'issued_to': 'JVS', 'issue_date':
'30-03-19'}

{'book_code': 'INP973', 'title': 'DATA SCIENCE AND BIG DATA COMPUTING  ',
'authors': 'ZAIGHAM MAHMOOD', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP955', 'title': 'BOOTSTRAP', 'authors': 'SPURLOCK, JAKE',
'issued_to': 'JVS', 'issue_date': '28-03-19'}

{'book_code': 'INP949', 'title': 'PROGRAMMING ENTITY FRAMEWORK', 'authors':
'LEARMAN, JULIA', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP948', 'title': 'PROFESSIONAL ANGULAR JS', 'authors':
'KARPOV,VALERI', 'issued_to': 'JVS', 'issue_date': '28-03-19'}

{'book_code': 'INP916', 'title': 'ANALYTICS IN A BIG DATA WORLD :THE ESSENTIAL
GUIDE TO DATA SCIENCE AND ITS APPL.', 'authors': 'BUESENS,BART', 'issued_to':
'JVS', 'issue_date': '29-04-19'}
```

```
{'book_code': 'INP902', 'title': 'ANGULAR JS: UP AND RUNNING', 'authors':
'SESHADRI , SHYAM AND GREEN,BRAD', 'issued_to': 'JVS', 'issue_date': '28-03-19'}

{'book_code': 'INP868', 'title': 'BEGINNING R : THE STATISTICAL PROGRAMMING
LANGUAGE', 'authors': 'GARDENER , MARK', 'issued_to': 'JVS', 'issue_date':
'29-04-19'}

{'book_code': 'INP861', 'title': 'R FOR DUMMIES :A WILEY BRAND', 'authors':
'DEVRIES, ANDRIE AND MEYS, JORIS', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP745', 'title': 'R IN A NUTSHELL', 'authors': 'ADLER, JOSEPH',
'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP1050', 'title': 'MACHINE LEARNING AND DATA MINING FOR COMPUTER
SECURITY ', 'authors': 'MALOOF MARCUS A. ', 'issued_to': 'JVS', 'issue_date':
'24-06-19'}

{'book_code': 'INP1028', 'title': 'AGILE DATA SCIENCE 2.0 ', 'authors': 'JURNEY
RUSSELL ', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

{'book_code': 'INP1012', 'title': 'THE R BOOK ', 'authors': 'CRAWLEY MICHAEL J.
', 'issued_to': 'JVS', 'issue_date': '29-04-19'}

count=13
```

Strings (str)

- String literals in Python can be enclosed in `' '` (single quotes) or `" "` (double quotes
- Triple quoted string literals using double quotes (`""" """`) or single quotes (`''' '''`) can span multiple lines. The embedded newlines and indentation are retained. Triple quoted strings using `""" """` (double quotes) are used for inline documentation. they are called docstrings. Triple quoted strings using `''' '''` (single quotes) can be used to enclose embedded code in other languages like SQL or HTML
- Triple quoted strings feature can also be used as a form of multiline comment - to comment-out several lines of code, enclose them in `''' '''`
- a prefix r indicates *raw* strings - the \ (backslash) character and escape sequences like `\n`, `\t`, etc. have no special meaning in them. They are often used for specifying regular expressions
- Python does not have a separate character type - a character is simply a string of length 1
- Strings are sequences of characters. They can be indexed to extract characters
- Python strings are *immutable*, a string object cannot be modified after it is created. Operations on strings that may require modification return new string objects
- Operations on strings are case-sensitive by default

```
In [1]: s1 = "GDCST"
        s2 = 'SPU'
        s3 = s1 + ' ' + s2
```

```
        print(s3)
        s4 = """    aaa
            bbb
            ccc   """
        print('XXX', end='')
        print(s4, end='')
        print('YYY')
        print("XXX\tYYY\nZZZ")
        print(r"XXX\tYYY\nZZZ")
        s5='abcd' "efg" # Consecutive string literals are joined automatically
        print(s5)
```

```
GDCST SPU
XXX    aaa
    bbb
    ccc   YYY
XXX     YYY
ZZZ
XXX\tYYY\nZZZ
abcdefg
```

```
In [102]: s1 = "GDCST"
          s2 = 'SPU'
          print(s1[2]) # Indexing a string
          print(s1[1:4]) # Slicing a string (a.k.a. substring)
          print('d' in s1)
          print('D' in s1)
          print('DCS' in s1)
          print(len(s1))
          ch = 'P'
          if ch in s2:
              print(s2.index(ch))
```

```
C
DCS
False
True
True
5
1
```

```
In [3]: s1 = 'GDCST'
        s2 = s1*4
        print(s2)
        s2 = (s1+' ') * 4
        print(s2)
        s3 = 'automata'
```

```
        print(s3.count('a'))
        print(s2.count('GDCST'))
        print(s2.startswith('GDCT'))
        print(s3.endswith('ata'))
        print(s2)
        index_substr = s2.find('DCS') # Return -1 if not found
        print(index_substr)
        index_substr = s2.index('DCS') # Raise ValueError if not found
        print(index_substr)
        index_substr = s2.rfind('DCS') # Reverse find, Return -1 if not found
        print(index_substr)
        index_substr = s2.rindex('DCS') # Reverse index, Raise ValueError if not found
        print(index_substr)
```

```
GDCSTGDCSTGDCSTGDCST
GDCST GDCST GDCST GDCST
3
4
False
True
GDCST GDCST GDCST GDCST
1
1
19
19
```

```
In [7]: s1 = 'GDCST'
        print(s1)
        print(s1.isupper())
        s2 = s1.lower()
        print(s2)
        print(s2.islower())
        s2 = s2.upper()
        print(s2)
        s2 = s1.capitalize()
        print(s2)
        s1 = 'abc dEf GHI'
        s2 = s1.capitalize()
        print(s2)
```

```
GDCST
True
gdcst
True
GDCST
Gdcst
Abc def ghi
```

```
In [53]: s1 = 'GDCST'
         s2 = s1.rjust(15)
         print('XXX' + s2 + 'YYY')
         print('XXX' + s2.lstrip() + 'YYY')
         s2 = s1.ljust(15)
         print('XXX' + s2 + 'YYY')
         print('XXX' + s2.rstrip() + 'YYY')
         s2 = s1.center(15)
         print('XXX' + s2 + 'YYY')
         print('XXX' + s2.strip() + 'YYY')
```

```
XXX          GDCSTYYY
XXXGDCSTYYY
XXXGDCST          YYY
XXXGDCSTYYY
XXX     GDCST     YYY
XXXGDCSTYYY
```

```
In [124]: print("'a'.isalpha() " + str('a'.isalpha()))
          print("'abc'.isalpha() " + str('abc'.isalpha()))
          print("'abc1'.isalpha() " + str('abc1'.isalpha()))
          print("'abc1'.isalnum() " + str('abc1'.isalnum()))
          print("'7'.isdigit() " + str('7'.isdigit()))
          print("'789'.isdigit() " + str('789'.isdigit()))
          print("'789.12'.isdigit() " + str('789.12'.isdigit()))
          s1 = " \t\n \t \n "
          print(s1.isspace())
          s1 = " \t\n \t \n a"
          print(s1.isspace())
```

```
'a'.isalpha() True
'abc'.isalpha() True
'abc1'.isalpha() False
'abc1'.isalnum() True
'7'.isdigit() True
'789'.isdigit() True
'789.12'.isdigit() False
True
False
```

```
In [4]: s1 = 'GDCST GDCST GDCST'
        print(s1.replace('GDCST', 'Department of Computer Science and Technology'))
        print(s1.replace('GDCST', 'Department of Computer Science and Technology', 1))
        print(s1.replace('GDCST', 'Department of Computer Science and Technology', 2))
        s2 = 'This is a line of text containing many words'
        s3 = s2.split(' ')
        print(s3)
```

```
        s2 = 'This is a line of text   containing      multiple consecutive delimiters'
        s3 = s2.split(' ')
        print(s3)
        s2 = '1,abc,67,56,73,23,ATKT'
        s3 = s2.split(',')
        print(s3)
        s2 = "The heights by great men reached and kept\nWere not attained by sudden
        flight,\nBut they, while their companions slept,\nWere toiling upward in the
        night.\n         - Henry Wadsworth Longfellow"
        s3 = s2.splitlines()
        for line in s3:
            print(line)
```

```
Department of Computer Science and Technology Department of Computer Science and
Technology Department of Computer Science and Technology
Department of Computer Science and Technology GDCST GDCST
Department of Computer Science and Technology Department of Computer Science and
Technology GDCST
['This', 'is', 'a', 'line', 'of', 'text', 'containing', 'many', 'words']
['This', 'is', 'a', 'line', 'of', 'text', '', '', 'containing', '', '', '', '',
'', 'multiple', 'consecutive', 'delimiters']
['1', 'abc', '67', '56', '73', '23', 'ATKT']
The heights by great men reached and kept
Were not attained by sudden flight,
But they, while their companions slept,
Were toiling upward in the night.
         - Henry Wadsworth Longfellow
```

```
In [135]: # String interpolation (old-style / printf-style formatting)
          # The right hand side operand of the % operator is a tuple
          print('| %3d | %s | %15s | %05.2f%% |' % (1, 'M14014', 'ABC', 81.45))
          print('| %-3d | %s | %-15s | %05.2f%% |' % (2, 'M14015', 'DEF', 7.6))
          print('| %03d | %s | %-15s | %05.2f%% |' % (3, 'M14016', 'GHI', 77.6789))
          s5 = '| %3d | %s | %-15s | %05.2f%% |' % (4, 'M14017', 'JKL', 84.16)
          print('substring (slice) of s5: ' + s5[15:35])
```

```
|   1 | M14014 |             ABC | 81.45% |
| 2   | M14015 | DEF             | 07.60% |
| 003 | M14016 | GHI             | 77.68% |
substring (slice) of s5: | JKL             |
```

```
In [5]: # New-style formatting using str.format()
        print("| {0} | {1} | {2} | {3}% |".format(1, 'M14014', 'ABC', 8.4567))
        print("| {} | {} | {} | {}% |".format(1, 'M14014', 'ABC', 8.4567))
        print("| {:03d} | {} | {:>15s} | {:05.2f}% |".format(1, 'M14014', 'ABC',
        8.4567))
```

```
        s5 = "| {:03d} | {} | {:>15s} | {:05.2f}% |".format(1, 'M14014', 'ABC', 8.4567);
        print('substring (slice) of s5: ' + s5[15:35])
```

```
| 1 | M14014 | ABC | 8.4567% |
| 1 | M14014 | ABC | 8.4567% |
| 001 | M14014 |             ABC | 08.46% |
substring (slice) of s5: |             ABC |
```

```
In [162]: student1 = {'seat_no' : 5, 'name' : 'XYZ', 'programme' : 'MCA', 'semester' : 5}
          result1 = {'seat_no' : 5, 'total_marks' : 256, 'result': 'Pass',
                         'percentage' : 56.50, 'class' : 'First'}
          student2 = {'seat_no' : 12, 'name' : 'QWERTY', 'programme' : 'MSc (IT)',
          'semester' : 3}
          result2 = {'seat_no' : 12, 'total_marks' : 226, 'result': 'Pass',
                         'percentage' : 56.50, 'class' : 'Second'}
          print('| {0[seat_no]:03d} | {0[name]:>15s} | {0[programme]:10} |'
                ' {0[semester]:d} | {1[total_marks]:3d} | {1[result]:4s} |'
                ' {1[percentage]:05.2f}% | {1[class]:s} |'.format(student1, result1))
          print('| {0[seat_no]:03d} | {0[name]:>15s} | {0[programme]:10} |'
                ' {0[semester]:d} | {1[total_marks]:3d} | {1[result]:4s} |'
                ' {1[percentage]:05.2f}% | {1[class]:s} |'.format(student2, result2))
```

```
| 005 |             XYZ | MCA        | 5 | 256 | Pass | 56.50% | First |
| 012 |          QWERTY | MSc (IT)   | 3 | 226 | Pass | 56.50% | Second |
```

Operation	Description
`s1 + s2`	String concatenation
`s1[2])`	Indexing a string to access characters from it
`s1[1:4]`	Slicing a string (substring)
'd' in s1	Check whether the given character exists in the string
`'DCS' in s1`	Check whether the given substring exists in the string
`len(s1)`	Length of the string
s1.index(ch))	Return the index of character ch in the string. Raise ValueError if not found
s1.find(ch))	Return the index of character ch in the string. Return -1 if not found
s1.rindex(ch))	Return the index of character ch in the string, searching from the end. Raise ValueError if not found
s1.rfind(ch))	Return the index of character ch in the string, searching from the end. Return -1 if not found

Summary of Operations on Strings

Operation	Description
`s1 * 4`	Repeat string 4 times
`s1.count('a')`	Count the number of times the given character appears in the string
`s1.count('xyz')`	Count the number of times the given substring appears in the string
`s1.startswith(s2)`	Return true if string `s1` starts with substring with `s2`
`s1.endswith(s2)`	Return true if string `s1` ends with substring with `s2`
`s1.isupper()`	Return true if every character in the string is an uppercase character
`s1.islower()`	Return true if every character in the string is a lowercase character
`s1.upper()`	Return an uppercase version of the entire string
`s1.lower()`	Return a lowercase version of the entire string
`s1.capitalize()`	Return a lowercase version of the entire string, with the first character in uppercase
`s2 = s1.rjust(15)`	Return a version of the string right-aligned in length 15
`s2 = s1.ljust(15)`	Return a version of the string left-aligned in length 15
`s2 = s1.center(15)`	Return a version of the string centered in length 15

Summary of Operations on Strings

Operation	Description
`s1.isalpha()`	Return true if all the characters in the string are alphabetic characters
`s1.isalnum()`	Return true if all the characters in the string are alphabetic characters or digits
`s1.isdigit()`	Return true if all the characters in the string are digits
`s1.isspace()`	Return true if all the characters in the string are white space characters
`s1.replace(str1, str2)`	Return string with all occurrences of `str1` replaced by `str2`
`s1.replace(str1, str2, 2)`	Return string with the first two occurrences of `str1` replaced by `str2`

Operation	Description
`s1.split(ch)`	Return an array containing parts of the string split using ch as the delimiter
`s1.splitlines()`	Return an array containing the lines from the string

Summary of Operations on Strings

Regular Expressions

```
In [23]: import re
         s1='13-07-2019'
         s2='Date: 13-07-2019'
         pattern = re.compile(r'\d\d-\d\d-\d\d\d\d')
         print(re.match(pattern, s1)) # re.match() always matches starting from the
         beginning of the string
         print(re.match(pattern, s2))
         if re.match(pattern, s1):
             print(s1 + ' matches the pattern')
         else:
             print(s1 + ' does not match the pattern')
         if re.match(pattern, s2):
             print(s2 + ' matches the pattern')
         else:
             print(s2 + ' does not match the pattern')
```

```
<re.Match object; span=(0, 10), match='13-07-2019'>
None
13-07-2019 matches the pattern
Date: 13-07-2019 does not match the pattern
```

```
In [36]: import re
         s1='13-07-2019'
         s2='Date: 13-07-2019'
         print(re.match(r'\d\d-\d\d-\d\d\d\d', s1))  # Same result as compile followed by
         match
                                                     # but less efficient for repeated
         matches
         #print(re.match('\\d\\d-\\d\\d-\\d\\d\\d\\d', s2))
         print(re.match(r'\d\d-\d\d-\d\d\d\d', s2))  # re.match() always starts matching
         from the beginning of the string
         print(re.search(r'\d\d-\d\d-\d\d\d\d', s2)) # re.search() searches anywhere in
         the string
         s3 = 'Sardar Patel University'
         print(re.match(r'[a-z]* [a-z]* [a-z]*', s3))
         print(re.match(r'[a-z]* [a-z]* [a-z]*', s3, re.I)) # The re.I flag stands for
         IGNORECASE
```

```
<re.Match object; span=(0, 10), match='13-07-2019'>
None
<re.Match object; span=(6, 16), match='13-07-2019'>
None
<re.Match object; span=(0, 23), match='Sardar Patel University'>
```

```
In [52]: s1 = 'Betty bought a bit of butter\nBut the bit of butter that Betty bought was
         bitter\nSo Betty bought a bit of better butter\nTo make the bitter butter
         better'
         print(s1)
         l1 = re.findall('b.*tt', s1, re.I) # Return all occurrences of matching parts of
         the string in a list of strings
         print(l1)
         l1 = re.split('\W+', s1)           # Split the string at the occurrences of the
         pattern. \W - non-word character
         print(l1)
         for word in l1:
             if re.match('b.*tt', word, re.I):
                 print(word)
         s2 = '13/07/2019'
         s3 = re.sub(r'(\d\d)/(\d\d)/(\d\d\d\d)', r'\2-\1-\3', s2)
         print(s3)
         s3 = re.sub(r'(\d\d)/(\d\d)/(\d\d\d\d)', r'\3-\2-\1', s2)
         print(s3)
```

```
Betty bought a bit of butter
But the bit of butter that Betty bought was bitter
So Betty bought a bit of better butter
To make the bitter butter better
['Betty bought a bit of butt', 'But the bit of butter that Betty bought was
bitt', 'Betty bought a bit of better butt', 'bitter butter bett']
['Betty', 'bought', 'a', 'bit', 'of', 'butter', 'But', 'the', 'bit', 'of',
'butter', 'that', 'Betty', 'bought', 'was', 'bitter', 'So', 'Betty', 'bought',
'a', 'bit', 'of', 'better', 'butter', 'To', 'make', 'the', 'bitter', 'butter',
'better']
Betty
butter
butter
Betty
bitter
Betty
better
butter
bitter
butter
better
07-13-2019
```

```
2019-07-13
```

```
In [59]: s1 = 'Username=JVS, Password=JVS123, LoginDate=12-07-2019'
         m1 = re.match(r'Username=([a-zA-Z]*), Password=([a-zA-Z0-9]*),
         LoginDate=([0-9-]*)', s1)
         print(m1)
         print('Username: ' + m1.group(1))
         print('Password: ' + m1.group(2))
         print('Login Date: ' + m1.group(3))
```

```
<re.Match object; span=(0, 51), match='Username=JVS, Password=JVS123,
LoginDate=12-07-20>
Username: JVS
Password: JVS123
Login Date: 12-07-2019
```

File Handling

```
In [33]: filename='../data-files/stock-data.csv'
         with open(filename) as file_obj:
             for line in file_obj:
                 print(line.rstrip('\n'))
```

```
Date,Open,High,Low,Close,Adj Close,Volume
2017-09-11,934.250000,938.380005,926.919983,929.080017,929.080017,1267000
2017-09-12,932.590027,933.479980,923.861023,932.070007,932.070007,1134400
2017-09-13,930.659973,937.250000,929.859985,935.090027,935.090027,1102600
2017-09-14,931.250000,932.770020,924.000000,925.109985,925.109985,1397600
2017-09-15,924.659973,926.489990,916.359985,920.289978,920.289978,2505400
2017-09-18,920.010010,922.080017,910.599976,915.000000,915.000000,1306900
2017-09-19,917.419983,922.419983,912.549988,921.809998,921.809998,936700
2017-09-20,922.979980,933.880005,922.000000,931.580017,931.580017,1669800
2017-09-21,933.000000,936.530029,923.830017,932.450012,932.450012,1290600
2017-09-22,927.750000,934.729980,926.479980,928.530029,928.530029,1052700
2017-09-25,925.450012,926.400024,909.700012,920.969971,920.969971,1856800
2017-09-26,923.719971,930.820007,921.140015,924.859985,924.859985,1666900
2017-09-27,927.739990,949.900024,927.739990,944.489990,944.489990,2239400
2017-09-28,941.359985,950.690002,940.549988,949.500000,949.500000,1020300
2017-09-29,952.000000,959.786011,951.510010,959.109985,959.109985,1581000
2017-10-02,959.979980,962.539978,947.840027,953.270020,953.270020,1283400
2017-10-03,954.000000,958.000000,949.140015,957.789978,957.789978,888300
2017-10-04,957.000000,960.390015,950.690002,951.679993,951.679993,952400
2017-10-05,955.489990,970.909973,955.179993,969.960022,969.960022,1213800
2017-10-06,966.700012,979.460022,963.359985,978.890015,978.890015,1173900
2017-10-09,980.000000,985.424988,976.109985,977.000000,977.000000,891400
2017-10-10,980.000000,981.570007,966.080017,972.599976,972.599976,957000
```

```
2017-10-11,973.719971,990.710022,972.250000,989.250000,989.250000,1540934
```

```
In [32]: import sys
         filename='../data-files/stock-data.csv'
         # Highest and lowest prices
         with open(filename) as file_obj:
             lineList = file_obj.readlines()
         lineList = lineList[1:]
         highest_values = [ x.split(',')[2] for x in lineList ]
         print('Highest Value: ' + max(highest_values))
         lowest_values = [ x.split(',')[3] for x in lineList ]
         print('Lowest Value: ' + min(lowest_values))
```

```
Highest Value: 990.710022
Lowest Value: 909.700012
```

```
In [1]: import sys
        filename='../data-files/stock-data.csv'
        # Highest and lowest prices
        highest_price = 0
        highest_price_date = ''
        lowest_price = sys.maxsize
        lowest_price_date=''
        # print(lowest_price)
        with open(filename) as file_obj:
            for line in file_obj:
                fields = line.split(',')
                try:
                    current_value = float(fields[2])
                    if current_value > highest_price:
                        highest_price = current_value
                        highest_price_date = fields[0]
                except ValueError:
                    pass
                try:
                    current_value = float(fields[3])
                    if current_value < lowest_price:
                        lowest_price = current_value
                        lowest_price_date = fields[0]
                except ValueError:
                    pass
        print('Highest Price:' + str(highest_price) +
                  ' on ' + highest_price_date +
                  ' Lowest Price:' + str(lowest_price) +
                  ' on ' + lowest_price_date)
```

```
Highest Price:990.710022 on 2017-10-11 Lowest Price:909.700012 on 2017-09-25
```

```
In [25]: import sys
         filename='../data-files/stock-data.csv'
         # Average prices
         with open(filename) as file_obj:
             for line in file_obj:
                 fields = line.split(',')
                 try:
                     highest_price = float(fields[2])
                     lowest_price = float(fields[3])
                     average_price = (highest_price + lowest_price) / 2;
                     print(fields[0] + ':\t' + str(average_price))
                 except ValueError:
                     pass
```

```
2017-09-11:     932.649994
2017-09-12:     928.6705015
2017-09-13:     933.5549925
2017-09-14:     928.38501
2017-09-15:     921.4249875
2017-09-18:     916.3399965
2017-09-19:     917.4849855
2017-09-20:     927.9400025
2017-09-21:     930.180023
2017-09-22:     930.60498
2017-09-25:     918.050018
2017-09-26:     925.980011
2017-09-27:     938.820007
2017-09-28:     945.619995
2017-09-29:     955.6480105
2017-10-02:     955.1900025
2017-10-03:     953.5700075
2017-10-04:     955.5400085
2017-10-05:     963.044983
2017-10-06:     971.4100035
2017-10-09:     980.7674865
2017-10-10:     973.825012
2017-10-11:     981.480011
```

```
In [22]: import sys
         filename='../data-files/stock-data.csv'
         # Average of average prices
         sum = 0.0
         count = 0.0
         with open(filename) as file_obj:
             for line in file_obj:
                 fields = line.split(',')
                 try:
```

```
                    highest_price = float(fields[2])
                    lowest_price = float(fields[3])
                    average_price = (highest_price + lowest_price) / 2;
                    sum = sum + average_price
                    count = count + 1
                except ValueError:
                    pass
        print(sum / count)
```

```
942.8774360434784
```

```
In [4]: import re
        filename='../data-files/gdcst-faculty-page.html'
        all_emails=[]
        with open(filename) as file_obj:
            for line in file_obj:
                emails=re.findall(r'([a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+)',
        line)
                if len(emails) > 0:
        # list.append() appends the argument to the list
        # list.extend() iterates over the argument and appends all its members to the
        list
                    all_emails.extend(emails)
        for email in all_emails:
            print(email)
        print('--------------------')
        for email in set(all_emails):
            print(email)
```

```
dbshah66@yahoo.com
dbshah66@yahoo.com
dbchoksi@yahoo.com
dbchoksi@yahoo.com
pvvirparia@yahoo.com
pvvirparia@yahoo.com
priti@pritisajja.info
priti@pritisajja.info
bbpdoc@yahoo.com
bb_patel@spuvvn.edu
prashantppittalia@yahoo.com
prashantppittalia@yahoo.com
jv_smart@spuvvn.edu
jv_smart@spuvvn.edu
sshardikpandit@yahoo.co.in
sshardikpandit@yahoo.co.in
birajpatel_4@yahoo.co.in
birajpatel_4@yahoo.co.in
```

```
jeegar.trivedi@yahoo.com
jeegar.trivedi@yahoo.com
raina_gaharwar@yahoo.co.in
raina_gaharwar@yahoo.co.in
sohnee.ss14@gmail.com
sohnee.ss14@gmail.com
--------------------
jeegar.trivedi@yahoo.com
sshardikpandit@yahoo.co.in
sohnee.ss14@gmail.com
pvvirparia@yahoo.com
raina_gaharwar@yahoo.co.in
bbpdoc@yahoo.com
dbshah66@yahoo.com
dbchoksi@yahoo.com
priti@pritisajja.info
bb_patel@spuvvn.edu
jv_smart@spuvvn.edu
birajpatel_4@yahoo.co.in
prashantppittalia@yahoo.com
```

Functions

```
In [22]: def greet(name):
             print('Hello ' + name + '!')

         greet('Jignesh')
```

```
Hello Jignesh!
```

```
In [1]: def greet(name):
            from datetime import datetime
            currentTime = datetime.now().time()
            currentHour = currentTime.hour
            if currentHour < 12:
                return 'Good morning ' + name + '!';
            else:
                return 'Good afternoon ' + name + '!';

        greeting = greet('Jignesh')
        print(greeting)
```

```
Good morning Jignesh!
```

```
In [20]: def expr(op1, operator, op2):
         # No switch in Python
```

```
            if operator == '+':
                return op1 + op2;
            elif operator == '-':
                return op1 - op2;
            elif operator == '*':
                return op1 * op2;
            elif operator == '/':
                return op1 / op2;
            elif operator == '%':
                return op1 % op2;
            elif operator == '**':
                return op1 ** op2;
            else:
                return 'Error: Unknown operator ' + operator;

        print(expr(10, '+', 20))
        print(expr(10, '-', 20))
        print(expr(10, '*', 20))
        print(expr(10, '/', 20))
        print(expr(10.0, '/', 20))
        print(expr(23, '%', 7))
        print(expr(2, '**', 8))
```

```
30
-10
200
0.5
0.5
2
256
```

```
In [4]: def fun1(a, b, c, d, e, f):
            print('a=' + str(a) + ' b=' + str(b) + ' c=' + str(c) +
                  ' d=' + str(d) + ' e=' + str(e) + ' f=' + str(f))
        # Positional parameters; must be passed in order
        fun1(1, 2, 3, 4, 5, 6)
        fun1(6, 5, 4, 3, 2, 1)
        # Keyword parameters; these may be passed in any order
        fun1(a=1, b=2, c=3, d=4, e=5, f=6)
        fun1(f=6, e=5, d=4, c=3, b=2, a=1)
        # Mixed parameters, all the keyword parameters MUST BE AFTER
        #                   all the positional parameters
        fun1(1, 2, 3, f=4, d=5, e=6)
```

```
a=1 b=2 c=3 d=4 e=5 f=6
a=6 b=5 c=4 d=3 e=2 f=1
a=1 b=2 c=3 d=4 e=5 f=6
```

```
a=1 b=2 c=3 d=4 e=5 f=6
a=1 b=2 c=3 d=5 e=6 f=4
```

```
In [6]: # default values for parameters
        # Parameters a and b do not have default values; hence it is mandatory to pass
        them
        # Parameters c, d, e and f have default values; hence they are optional
        parameters
        # If a parameter having default value is not passsed, it assumes its default
        value
        def fun1(a, b, c=40, d=50, e=60, f=70):
            print('a=' + str(a) + ' b=' + str(b) + ' c=' + str(c) +
                  ' d=' + str(d) + ' e=' + str(e) + ' f=' + str(f))
        # TypeError: fun1() missing 1 required positional argument: 'b'
        # fun1(1)
        fun1(1, 2)
        fun1(1, 2, 3, 4, 5, 6)
        fun1(1, 2, 333, 444, f=555, e=666)
        fun1(a=1, b=2, f=333, d=444, e=555, c=666)
        fun1(a=1, b=2, f=333, d=444)
```

```
a=1 b=2 c=40 d=50 e=60 f=70
a=1 b=2 c=3 d=4 e=5 f=6
a=1 b=2 c=333 d=444 e=666 f=555
a=1 b=2 c=666 d=444 e=555 f=333
a=1 b=2 c=40 d=444 e=60 f=333
```

```
In [6]: # Variable number of positional parameters
        def sum(*args):
            # print(type(args)) --> tuple
            total=0
            for arg in args:
                total = total + arg
            return total
        print(sum())
        print(sum(15))
        print(sum(1, 2, 3))
        print(sum(1, 2, 3, 4, 5, 6))
        # Variable number of positional and fixed number of keyword parameters
        def sum(*args, tax_rate, surcharge_rate):
            total=0
            for arg in args:
                total = total + arg
            total = total + total * tax_rate / 100;
            total = total + total * surcharge_rate / 100;
            return total
```

```
        print(sum(100, 200, 300, tax_rate=10, surcharge_rate=5))
        print(sum(100, 200, 300, 400, 500, 600,
                  tax_rate=10, surcharge_rate=5))
```

```
0
15
6
21
693.0
2425.5
```

```
In [1]: # Variable number of positional and variable number of keyword paramneters
        def fun2(*args, **kwargs):
            # print(type(args)) --> tuple
            for arg in args:
                print(arg, end=' ')
            # print(type(kwargs)) --> dictionary
            for kwarg, value in kwargs.items():
                print(kwarg + '=' + str(value), end=' ')
            print()
        fun2(1, 2, 3, a=111, b=222, c=333, d=444)
        fun2(1, 2, 3, 4, 5)
        fun2(a=111, b=222, c=333)
        fun2()
```

```
1 2 3 a=111 b=222 c=333 d=444
1 2 3 4 5
a=111 b=222 c=333
```

```
In [4]: # Mandatory positional arguments followed by
        # variable number of positional and variable number of keyword paramneters
        def fun2(a1, a2, *args, **kwargs):
            print('a1=' + str(a1), end=' ')
            print('a2=' + str(a2), end=' ')
            for arg in args:
                print(arg, end=' ')
            for kwarg, value in kwargs.items():
                print(kwarg + '=' + str(value), end=' ')
            print()
        # a1='X', a2='Y', *args=[1, 2, 3], **kwargs={a=111, b=222, c=333}
        fun2('X', 'Y', 1, 2, 3, a=111, b=222, c=333, d=444)
        fun2('X', 'Y', 1, 2, 3, 4, 5)
        fun2('X', 'Y', a=111, b=222, c=333)
        # a1=1, a2=2, *args=[3], **kwargs={a=111, b=222, c=333}
        fun2(1, 2, 3, a=111, b=222, c=333, d=444)
        # a1=1 a2=???
```

```
        # TypeError: fun2() missing 1 required positional argument: 'a2'
        # fun2(1, a=111, b=222, c=333, d=444)

a1=X a2=Y 1 2 3 a=111 b=222 c=333 d=444
a1=X a2=Y 1 2 3 4 5
a1=X a2=Y a=111 b=222 c=333
a1=1 a2=2 3 a=111 b=222 c=333 d=444


In [28]: # Returning multiple values
         def result(m1, m2, m3, m4):
             total = m1 + m2 + m3 + m4
             percentage = total / 4
             fail_count = 0
             if m1 < 40:
                 fail_count = fail_count + 1
             if m2 < 40:
                 fail_count = fail_count + 1
             if m3 < 40:
                 fail_count = fail_count + 1
             if m4 < 40:
                 fail_count = fail_count + 1
             if fail_count == 0:
                 result='Pass'
                 if percentage > 70:
                     result_class='First Class with Distinction'
                 elif percentage > 60:
                     result_class='First Class'
                 elif percentage > 50:
                     result_class='Second Class'
                 elif percentage > 40:
                     result_class='Pass Class'
             elif fail_count <= 2:
                 result='ATKT'
                 result_class=''
             else:
                 result='Fail'
                 result_class=''
             return result, result_class
         r1, c1 = result(76, 65, 70, 73)
         print('Result=' + r1 + ' Class=' + c1)
         r1, c1 = result(76, 65, 70, 63)
         print('Result=' + r1 + ' Class=' + c1)
         r1, c1 = result(56, 61, 62, 41)
         print('Result=' + r1 + ' Class=' + c1)
         r1, c1 = result(46, 42, 40, 47)
         print('Result=' + r1 + ' Class=' + c1)
         r1, c1 = result(36, 45, 50, 29)
```

```
print('Result=' + r1 + ' Class=' + c1)
r1, c1 = result(19, 15, 40, 33)
print('Result=' + r1 + ' Class=' + c1)
```

```
Result=Pass Class=First Class with Distinction
Result=Pass Class=First Class
Result=Pass Class=Second Class
Result=Pass Class=Pass Class
Result=ATKT Class=
Result=Fail Class=
```

Classes and Object-oriented Programming

```
In [1]: # Class name should be in upper camel case (e.g. PythonExampleClass)
        class Rectangle:
            """A class representing a rectangle"""
            # The above string is the "docstring" for the class
            # It serves as documentation if it is the first thing in the class
            # It can be accessed as name.__doc__

            # A data member created at the class level becomes a class (static) member
            # It can be accessed as classname.member_name (or self.member_name as well)
            # It is shared between all instances (objects) of the class
            no_sides = 4

            # A class (static) method does not have the self argument
            # It can be invoked as classname.method_name
            def no_of_sides():
                return Rectangle.no_sides;

            # All instance (non-static) methods must be declared with self as the first
        argument
            # "self" is like "this" in C++ and Java
            # It refers to the object on which the method is invoked
            def my_sides(self):
            # Class (static) members can also be accessed as self.member_name
                return 'I have ' + str(self.no_sides) + ' sides.';

            def area(self):
                """Returns the area of the rectangle"""
            # The above string is the docstring for the method
            # (must be the first thing in the method)

            # Instance (non-static) members can be created in any method by assigning a
        value
            # These members can only be accessed as self.member_name
            # These members may have different values in different instances (objects)
```

```
            return self.length * self.breadth;

        def perimeter(self):
            return 2 * (self.length + self.breadth)

        # Constructor
        # There can be only one constructor, it cannot be overloaded
        def __init__(self):
            self.length = 0
            self.breadth = 0

        # "toString()" method; returns a string representation of the object
        def __str__(self):
            return 'Rectangle(' + str(self.length) + ',' + str(self.breadth) + ')'

    # Print the docstring for the class
    print(Rectangle.__doc__)
    # Print the docstring for the area() method
    print(Rectangle.area.__doc__)
    print(Rectangle.no_of_sides())
    r1 = Rectangle()
    print(r1)
    # Members are public by default
    r1.length = 5
    r1.breadth = 3
    print(str(r1.length) + ' ' + str(r1.breadth))
    print(r1)
    print(r1.my_sides())
    print(r1.perimeter())
    print(r1.area())
    r1.breadth = 10
    print(r1.area())
    Rectangle.no_sides = 33
    print(r1.my_sides())
    r2 = Rectangle()
    print(r1.my_sides())
    help(Rectangle)
    # pydoc modulename/classname/methodname at the command prompt works like the man
    command in Unix
```

```
A class representing a rectangle
Returns the area of the rectangle
4
Rectangle(0,0)
5 3
Rectangle(5,3)
I have 4 sides.
16
```

```
15
50
I have 33 sides.
I have 33 sides.
Help on class Rectangle in module __main__:

class Rectangle(builtins.object)
 |  A class representing a rectangle
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __str__(self)
 |      Return str(self).
 |
 |  area(self)
 |      Returns the area of the rectangle
 |
 |  my_sides(self)
 |      # All instance (non-static) methods must be declared with self as the
first argument
 |      # "self" is like "this" in C++ and Java
 |      # It refers to the object on which the method is invoked
 |
 |  no_of_sides()
 |      # A class (static) method does not have the self argument
 |      # It can be invoked as classname.method_name
 |
 |  perimeter(self)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  no_sides = 33
```

```
In [37]: class Person:
             """Represents a person class"""
             def __init__(self, name, number):
                 self.name = name
                 self.number = number
             def __str__(self):
                 return 'Person(' + self.name + ')'
         p1 = Person('Abc', '9876543210')
         print(p1)
         p2 = Person('Xyz', '1234567890')
         print(p2)
```

```
Person(Abc)
Person(Xyz)
```

```
In [71]: # Inheritance

         class Person:
             """Represents a person class"""
             def __init__(self, name, number):
                 self.name = name
                 self.number = number
             def __str__(self):
                 return 'Person(' + self.name + ')'

         # Class employee inherits from class Person
         class Employee(Person):
             def __init__(self, name, number, salary):
                 # There are two ways of calling super class constructor
                 # Person.__init__(self, name, number)
                 super().__init__(name, number)
                 self.salary = salary
             def __str__(self):
                 return 'Employee(' + self.name + ',' + str(self.salary) + ')'
         e1 = Employee('Qwe', '1122334455', 35000)
         print(e1)
```

```
Employee(Qwe,35000)
```

```
In [76]: # Overriding a method

         class Person:
             def introduce(self):
                 print('I am a person. My name is ' + self.name + '.')
             def __init__(self, name):
                 self.name = name
```

```
        class Teacher:
            def introduce(self):
                print('I am a teacher. My name is ' + self.name + '. I teach ' +
        self.subject + '.')
            def __init__(self, name, subject):
                Person.__init__(self, name)
                self.subject = subject

        p1 = Person('Rishad')
        p1.introduce()
        t1 = Teacher('Jignesh', 'Python')
        t1.introduce()
```

```
I am a person. My name is Rishad.
I am a teacher. My name is Jignesh. I teach Python.
```

In [106]:
```
# Multiple inheritance

        class AquaticVehicle:
            def capabilities():
                return ['can swim']
            def swimming(self):
                print ('can swim at ' + self.swimming_speed)
            def __init__(self, speed):
                self.swimming_speed = speed

        class TerrestrialVehicle:
            def capabilities():
                return ['has wheels']
            def on_road(self):
                print ('on road speed: ' + self.on_road_speed)
            def __init__(self, speed):
                self.on_road_speed = speed

        class AerialVehicle:
            def capabilities():
                return ['can fly']
            def flying(self):
                print ('can fly at ' + self.flying_speed)
            def __init__(self, speed):
                self.flying_speed = speed

        class Seaplane(AquaticVehicle, AerialVehicle):
            def capabilities():
                capability_list = []
                capability_list.extend(AquaticVehicle.capabilities())
                capability_list.extend(AerialVehicle.capabilities())
```

```
            return capability_list;
        def __init__(self, swimming_speed, flying_speed):
            AquaticVehicle.__init__(self, swimming_speed)
            AerialVehicle.__init__(self, flying_speed)

    class AmphibiousAircraft(AquaticVehicle, TerrestrialVehicle, AerialVehicle):
        def capabilities():
            capability_list = []
            capability_list.extend(AquaticVehicle.capabilities())
            capability_list.extend(TerrestrialVehicle.capabilities())
            capability_list.extend(AerialVehicle.capabilities())
            return capability_list;
        def __init__(self, swimming_speed, on_road_speed, flying_speed):
            AquaticVehicle.__init__(self, swimming_speed)
            TerrestrialVehicle.__init__(self, on_road_speed)
            AerialVehicle.__init__(self, flying_speed)

    print('Aquatic vehicle capabilities: ', AquaticVehicle.capabilities())
    print('Terrestrial vehicle capabilities: ', TerrestrialVehicle.capabilities())
    print('Aerial vehicle capabilities: ', AerialVehicle.capabilities())
    print('Seaplane capabilities: ', SeaPlane.capabilities())
    print('AmphibiousAircraft capabilities: ', AmphibiousAircraft.capabilities())

    v1 = AquaticVehicle('25 mph')
    print('AquaticVehicle:')
    v1.swimming()
    v2 = TerrestrialVehicle('120 kmph')
    print('TerrestrialVehicle:')
    v2.on_road()
    v3 = AerialVehicle('750 kmph')
    print('AerialVehicle:')
    v3.flying()
    v4 = Seaplane('10 mph', '500 kmph')
    print('Seaplane:')
    v4.swimming()
    v4.flying()
    v5 = AmphibiousAircraft('10 mph', '80 kmph', '500 kmph')
    print('AmphibiousAircraft:')
    v5.swimming()
    v5.on_road()
    v5.flying()
```

```
Aquatic vehicle capabilities:  ['can swim']
Terrestrial vehicle capabilities:  ['has wheels']
Aerial vehicle capabilities:  ['can fly']
Seaplane capabilities:  ['can swim', 'can fly']
AmphibiousAircraft capabilities:  ['can swim', 'has wheels', 'can fly']
AquaticVehicle:
```

```
can swim at 25 mph
TerrestrialVehicle:
on road speed: 120 kmph
AerialVehicle:
can fly at 750 kmph
Seaplane:
can swim at 10 mph
can fly at 500 kmph
AmphibiousAircraft:
can swim at 10 mph
on road speed: 80 kmph
can fly at 500 kmph
```

Exception Handling

```
In [7]: print('Enter an integer:', end=' ')
        x=input()
        z=int(x) * 2
        print('x * 2 = ' + str(z))

Enter an integer: a


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-7-f3c50a9d7a0d> in <module>
          1 print('Enter an integer:', end=' ')
          2 x=input()
    ----> 3 z=int(x) * 2
          4 print('x * 2 = ' + str(z))


        ValueError: invalid literal for int() with base 10: 'a'


In [9]: while True:
            try:
                print('Enter an integer:', end=' ')
                x = input()
                x = int(x)
                break
            except ValueError:
                print('Invalid integer ' + str(x) + '\nTry and try and you will
        succeed')
        print('x * 2 = ' + str(z))
```

```
Enter an integer: a
Invalid integer a
Try and try and you will succeed
Enter an integer: 123.45
Invalid integer 123.45
Try and try and you will succeed
Enter an integer: 10
x * 2 = 10
```

```
In [22]: import sys
         x=5
         y=0
         try:
             z = x / y
             print(z)
         except ZeroDivisionError:
             print('Attempt to divide by zero detected')
             # raise
         f_name = 'non_existant.file'
         try:
             with open(f_name) as f_obj:
                 lines = f_obj.readlines()
         except FileNotFoundError as ex:
             print(ex)
             print('ex.errno=' + str(ex.errno))
             print('ex.args=' + str(ex.args))
             msg = "Can't find file {0}.".format(f_name)
             print(msg)
```

```
Attempt to divide by zero detected
[Errno 2] No such file or directory: 'non_existant.file'
ex.errno=2
ex.args=(2, 'No such file or directory')
Can't find file non_existant.file.
```

```
In [23]: x='abc'
         y=10
         try:
             z = int(x) / y
             print(y)
         except ZeroDivisionError:
             print('Attempt to divide by zero detected')
         except FileNotFoundError:
             print('Impossible to reach here...')
         # Default exception handler
         # Catches all exceptions, must be the last except clause
```

```
        except:
            print("Unexpected error:", sys.exc_info())
```

```
Unexpected error: (<class 'ValueError'>, ValueError("invalid literal for int()
with base 10: 'abc'"), <traceback object at 0x7fcd9c2501c8>)
```

```
In [54]: import sys
         f_obj = open('../data-files/attendance-PS05EMCA22-2019.txt')
         print('file opened')
         try:
             lines = f_obj.readlines()
             print(''.join(lines[0:5]))
             print(1/0)
             f_obj.close()
             print('file closed')
         except:
             print(sys.exc_info())
         print('f_obj.closed=' + str(f_obj.closed))
```

```
file opened
##subjectcode=PS05EMCA22
##subjectname=Introduction to Data Science and Big Data
##academicyear=2019-20
##batch=1&2
##start1=1

(<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero'), <traceback
object at 0x7fcd9c2bbd88>)
f_obj.closed=False
```

```
In [47]: import sys
         f_obj = open('../data-files/attendance-PS05EMCA22-2019.txt')
         print('file opened')
         try:
             lines = f_obj.readlines()
             print(''.join(lines[0:5]))
             print(1/0)
         except:
             print(sys.exc_info())
         finally:
             f_obj.close()
             print('file closed')
```

```
file opened
##subjectcode=PS05EMCA22
##subjectname=Introduction to Data Science and Big Data
##academicyear=2019-20
```

```
##batch=1&2
##start1=1

(<class 'ZeroDivisionError'>, ZeroDivisionError('division by zero'), <traceback
object at 0x7fcd9c2955c8>)
file closed
```

```
In [55]: with open('../data-files/attendance-PS05EMCA22-2019.txt') as f_obj:
             lines = f_obj.readlines()
             print(''.join(lines[0:5]))
             print(1/0)
         # f_obj is out of scope here, but it has been closed

##subjectcode=PS05EMCA22
##subjectname=Introduction to Data Science and Big Data
##academicyear=2019-20
##batch=1&2
##start1=1
```

```
---------------------------------------------------------------------------

ZeroDivisionError                         Traceback (most recent call last)

<ipython-input-55-e231d68e1bfc> in <module>
  2     lines = f_obj.readlines()
  3     print(''.join(lines[0:5]))
----> 4     print(1/0)
  5 # f_obj is out of scope here, but it has been closed

ZeroDivisionError: division by zero
```