Module 2

Module -2 Building Python Programs

Strings and text files – Accessing characters, substrings, Data encryption, Strings and number system, String methods, Text files, A case study on text analysis.

Design with Functions – Functions as Abstraction Mechanisms, Problem solving with top-down design, Design with recursive functions, Managing a program's namespace, Higher-Order Functions.

Lists - Basic list Operations and functions, List of lists, Slicing, Searching and sorting list, List comprehension. Work with tuples. Sets. Work with dates and times, A case study with lists.

Dictionaries - Dictionary functions, dictionary literals, adding and removing keys, accessing and replacing values, traversing dictionaries, reverse lookup. Case Study – Data Structure Selection.

Strings and text files

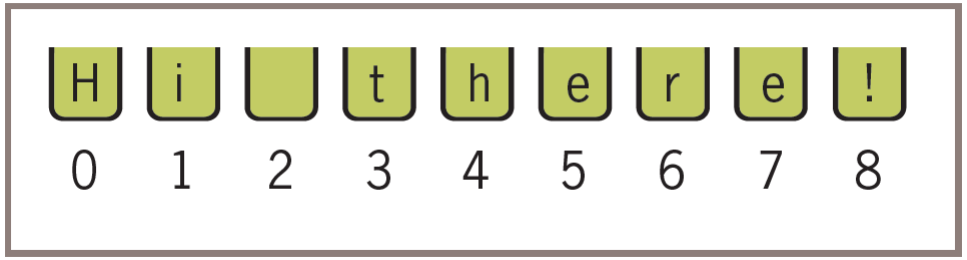
Strings and text files

- After completing this session, you will be able to
  - Access individual characters in a string
  - Retrieve a substring from a string
  - Search for a substring in a string
  - Convert a string representation of a number from one base to another base
  - Use string methods to manipulate strings
  - Open a text file for output and write strings or numbers to the file
  - Open a text file for input and read strings or numbers from the file
  - Use library functions to access and navigate a file system

Accessing Characters and Substrings in Strings

- A string's length is the number of characters it contains.
- Python's **len** function returns this value when it is passed a string

```
>>> len("Hi there!")
9
>>> len("")
0
```



The string is an **immutable data structure.**

The Subscript Operator

- Although a simple **for** loop can access any of the characters in a string, sometimes you just want to inspect one character at a given position without visiting them all.
- The **subscript operator []** makes this possible.

```
<a string>[<an integer expression>]
```

The Subscript Operator

```
>>> name = "Alan Turing"
>>> name[0]                 # Examine the first character
'A'
>>> name[3]                 # Examine the fourth character
'n'
>>> name[len(name)]         # Oops! An index error!
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
>>> name[len(name) - 1]     # Examine the last character
'g'
>>> name[-1]                # Shorthand for the last character
'g'
>>> name[-2]                # Shorthand for next to last character
'n'
```

The Subscript Operator

```
>>> data = "Hi there!"
>>> for index in range(len(data)):
        print(index, data[index])
0 H
1 i
2
3 t
4 h
5 e
6 r
7 e
8 !
```

Slicing for Substrings

- Python string supports slicing to create substring.
- Note that Python string is immutable, slicing creates a new substring from the source string and original string remains unchanged.
- Python slice string syntax is:

```
str_object[start_pos:end_pos:step]
```

  - The slicing starts with the **start_pos index (included) and ends at end_pos index (excluded)**.

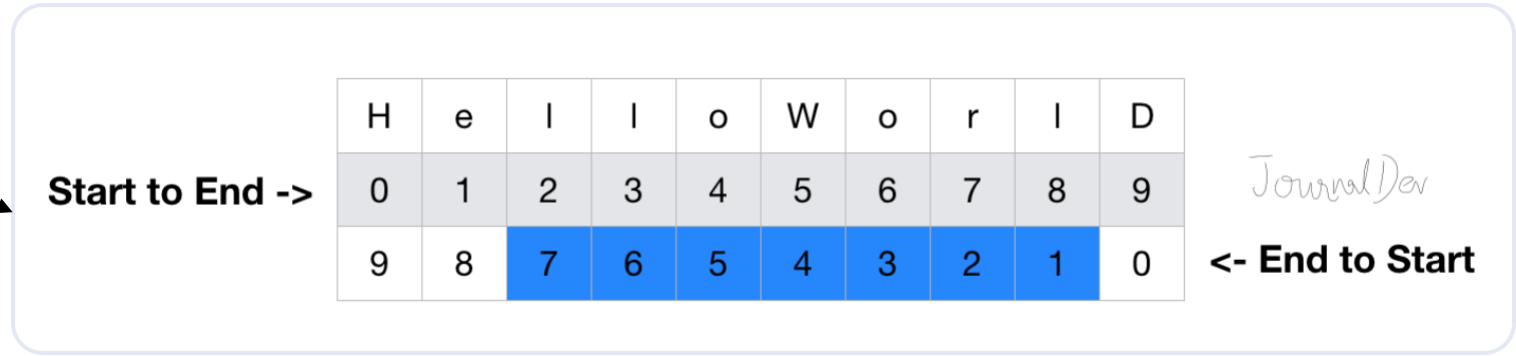
```
s = 'HelloWorld'
first_five_chars = s[:5]
print(first_five_chars)

third_to_fifth_chars = s[2:5]
print(third_to_fifth_chars)
```

Slicing for Substrings

- Reverse a String using Slicing
- We can reverse a string using slicing by providing the step value as -1.

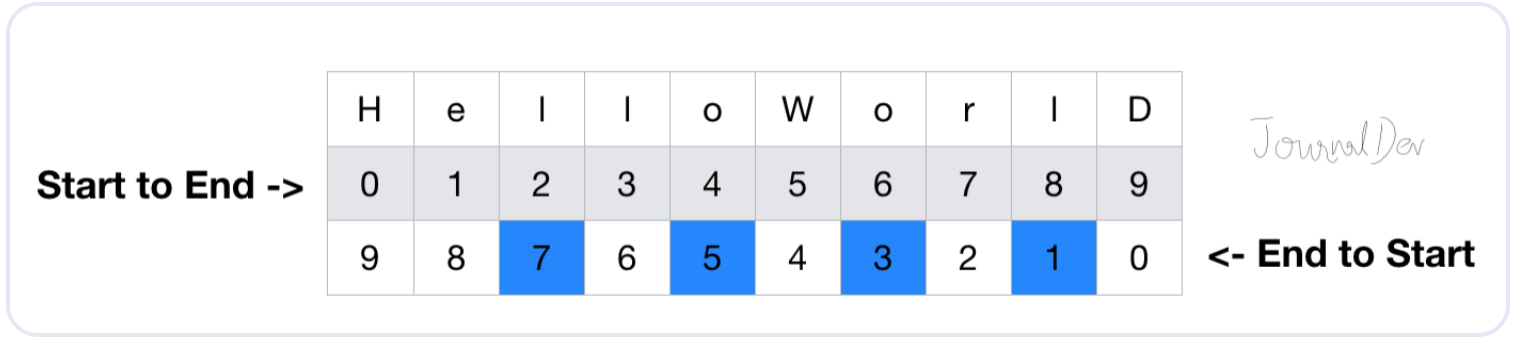
```
s = 'HelloWorld'
reverse_str = s[::-1]
print(reverse_str)
```

```
s1 = s[8:1:-1]
print(s1)
```



Slicing for Substrings

```
s1 = s[8:1:-2]
print(s1)
```



Slicing for Substrings

- Python slice works with negative indexes too, in that case, the start_pos is excluded and end_pos is included in the substring.

```
>>>s = 'Python'
>>>s[100:]
''
>>>s[2:50]
'thon'
```

Testing for a Substring with the in Operator

- For example, you might want to pick out filenames with a .txt extension.

```
>>> fileList = ["myfile.txt", "myprogram.exe", "yourfile.txt"]
>>> for fileName in fileList:
            if ".txt" in fileName:
                print(fileName)

myfile.txt
yourfile.txt
```

Exercises

- Assume that the variable **data** refers to the string **"myprogram.exe".** Write the values of the following expressions:
  a) data[2]
  b) data[-1]
  c) len(data)
  d) data[0:8]

Exercises

- Assume that the variable **data** refers to the string **"myprogram.exe".** Write the expressions that perform the following tasks:

a. Extract the substring "gram" from data.
b. Truncate the extension ".exe" from data.
c. Extract the character at the middle position from data.

Exercises

- Assume that the variable **myString** refers to a string. Write a code segment that uses a loop to print the characters of the string in reverse order.

- Assume that the variable **myString** refers to a string, and the variable **reversedString** refers to an empty string. Write a loop that adds the characters from **myString** to **reversedString** in reverse order.

Strings and text files

Converting Binary to Decimal

$$1100111_2 =$$

$$1 * 2^6 + 1 * 2^5 + 0 * 2^4 + 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 1 * 2^0 =$$

$$1 * 64 + 1 * 32 + 0 * 16 + 0 * 8 + 1 * 4 + 1 * 2 + 1 * 1 =$$

$$64 + 32 + 4 + 2 + 1 = 103$$

Converting Binary to Decimal

```
"""
Program: binarytodecimal.py
"""
binaryNumber = input("Enter the Binary Number:")
decimalNumber=0
exponent = len(binaryNumber)-1
for digit in binaryNumber:
    decimalNumber = decimalNumber + int(digit) *
2**exponent
    exponent-=1
print("The decimal equivalent is: ",decimalNumber)
```

```
Enter the Binary Number:1010
The decimal equivalent is:  10
```

Converting Decimal to Binary

Division	Remainder (R)
112 / 2 = 56	0
56 / 2 = 28	0
28 / 2 = 14	0
14 / 2 = 7	0
7 / 2 = 3	1
3 / 2 = 1	1
1 / 2 = 0	1

Now, write remainder from bottom to up (in reverse order), this will be 1110000 which is equivalent binary number of decimal integer 112.

Converting Binary to Decimal

```
decimalNumber=int(input("Enter a decimal number:"))
if decimalNumber==0:
    print(0)
else:
    binaryNumber=""
    while decimalNumber>0:
        remainder = decimalNumber%2
        decimalNumber = decimalNumber//2
        binaryNumber = str(remainder)+binaryNumber
    print(binaryNumber)
```

```
Enter the decimal number:10
10
```

Exercises

- Translate each of the following numbers to decimal numbers:
  - 11001_2
  - 100000_2
  - 11111_2
- Translate each of the following numbers to binary numbers:
  - 47_{10}
  - 127_{10}
  - 64_{10}
- Translate each of the following numbers to binary numbers:
  - 47_8
  - 127_8
  - 64_8

Exercises

- Translate each of the following numbers to decimal numbers:
  - 47_8
  - 127_8
  - 64_8
- Translate each of the following numbers to decimal numbers:
  - 47_{16}
  - 127_{16}
  - AA_{16}

String Methods

Case Conversion

- These are the case conversion methods
  - `str.capitalize()`
  - `str.lower()`
  - `str.swapcase()`
  - `str.title()`
  - `str.upper()`

Case Conversion

- `str.capitalize()` - Converts first character to Capital Letter

```
sentence = "i love PYTHON"

# converts first character to uppercase and others to lowercase
capitalized_string = sentence.capitalize()

print(capitalized_string)

# Output: I love python
```

Case Conversion

- `str.lower()` - Converts all uppercase characters in a string into lowercase characters and returns it.

```
sentence = "i love PYTHON"

capitalized_string = sentence.lower()

print(capitalized_string)

# Output: i love python
```

Case Conversion

- `str.swapcase()-` method returns the string by converting all the characters to their opposite letter case(uppercase to lowercase and vice versa).

```
name = "JoHn CeNa"

# converts lowercase to uppercase and vice versa
print(name.swapcase())

# Output: jOhN cEnA
```

Case Conversion

- `str.title()` - method returns a string with first letter of each word capitalized; a title cased string.

```
text = 'My favorite number is 25.'
print(text.title())

text = '234 k3l2 *43 fun'
print(text.title())

# Output: My Favorite Number Is 25.
          234 K3L2 *43 Fun
```

Case Conversion

- `str.upper()-` converts all lowercase characters in a string into uppercase characters and returns it.

```
message = 'python is fun'

# convert message to uppercase
print(message.upper())

# Output: PYTHON IS FUN
```

Find and Seek

- These are find and seek methods:
  - `str.count(<sub>[, <start>[, <end>]])`
  - `str.endswith()`
  - `str.startswith()`
  - `str.find()`
  - `str.rfind()`
  - `str.index()`
  - `str.rindex()`

Find and Seek

- `str.count()-` method returns the number of occurrences of a substring in the given string.

```
message = 'python is popular programming language'

# number of occurrence of 'p'
print('Number of occurrence of p:', message.count('p'))

# Output: Number of occurrence of p: 4
```

Find and Seek

- `str.count(<sub>[, <start>[, <end>]])-` method returns the number of occurrences of a substring in the given string.

```
# define string
string = "Python is awesome, isn't it?"
substring = "i"

# count after first 'i' and before the last 'i'
count = string.count(substring, 8, 25)

# print count
print("The count is:", count)

# Output: The count is: 1
```

Find and Seek

- `str.endswith()` - method returns **True** if a string ends with the specified suffix. If not, it returns **False**.

```
message = 'Python is fun'

# check if the message ends with fun
print(message.endswith('fun'))

# Output: True
```

Find and Seek

- `str.startswith()` - method returns **True** if a string starts with the specified prefix(string). If not, it returns **False**.

```
message = 'Python is fun'

# check if the message ends with fun
print(message.startswith('Python'))

# Output: True
```

Find and Seek

- `str.find()-` method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

```
message = 'Python is a fun programming language'

# check the index of 'fun'
print(message.find('fun'))

# Output: 12
```

Find and Seek

- `str.rfind()-` method returns the highest index of the substring (if found). If not found, it returns -1.

```
quote = 'Let it be, let it be, let it be'

result = quote.rfind('let it')
print("Substring 'let it':", result)

# Output: Substring 'let it': 22
```

Find and Seek

- `str.index()-` method returns the index of a substring inside the string (if found). If the substring is not found, it raises an exception.

```
sentence = 'Python programming is fun.'

result = sentence.index('is fun')
print("Substring 'is fun':", result)

result = sentence.index('Java')
print("Substring 'Java':", result)
```

```
Substring 'is fun': 19

Traceback (most recent call last):
  File "<string>", line 6, in
    result = sentence.index('Java')
ValueError: substring not found
```

Character Classification

- Here are character classification methods:
  - `str.isalnum()`
  - `str.isalpha()`
  - `str.isdigit()`
  - `str.isidentifier()` *
  - `iskeyword(<str>)` *
  - `str.islower()`
  - `str.isupper()`
  - `str.isascii()` *

Find and Seek

- `str.center(<width>[, <fill>])`

```
>>> s = 'spam'
>>> s.center(10)
'   spam   '
>>> s.center(10, '-')
'---spam---'
```

Exercises

- Write a program to check Whether a String is Palindrome or Not

```
inputString=input("Enter the string to check whether it is palindrome or not:")
inputString=inputString.upper()
if inputString==inputString[::-1]:
    print("The String is palindrome")
else:
    print("The String is not palindrome")
```

Text Files

Create A Empty Text File

```
open('file_Path', 'access_mode')
```

File Mode	Meaning
`w`	Create a new file for writing. If a file already exists, it truncates the file first. Use to create and write content into a new file.
`x`	Open a file only for exclusive creation. If the file already exists, this operation fails.
`a`	Open a file in the append mode and add new content at the end of the file.
`b`	Create a binary file
`t`	Create and open a file in a text mode

File access mode

Create File

Example: Create a new empty text file named ‘sales.txt’

```
# create a empty text file
# in current directory
fp = open('sales.txt', 'x')
fp.close()
```

Example: Create and write content into a file.

```
# create a empty text file and
write
fp = open('sales.txt', 'w')
fp.write('first line')
fp.close()
```

Python Lists

A List is a Kind of Collection

- A **collection** allows us to put many values in a single “variable”
- A **collection** is nice because we can carry all many values around in one convenient package.

```
friends = [ 'Joseph', 'Glenn', 'Sally' ]

carryon = [ 'socks', 'shirt', 'perfume' ]
```

List Constants

- **List** constants are surrounded by square brackets and the elements in the list are separated by commas
- A **list** element can be any Python object - even **another list**
- A **list** can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow', 'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

We Already Use Lists!

```
for i in [5, 4, 3, 2, 1] :
    print(i)
print('Blastoff!')
```

```
5
4
3
2
1
Blastoff!
```

Lists and Definite Loops - Best Pals

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends :
    print('Happy New Year:',  friend)
print('Done!')
```

```
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

```
z = ['Joseph', 'Glenn', 'Sally']
for x in z :
    print('Happy New Year:’,  x)
print('Done!')
```

Looking Inside Lists

- Just like strings, we can get at any single element in a list using an index specified in **square brackets**

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
```

Lists are Mutable

- **Strings** are "**immutable**" - we cannot change the contents of a string - we must make a **new string** to make any change

- **Lists** are "**mutable**" - we can **change** an element of a list using the **index** operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print(x)
banana
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

How Long is a List?

- The **len()** function takes a **list** as a parameter and returns the number of **elements** in the **list**
- Actually **len()** tells us the number of elements of any set or sequence (such as a string...)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
>>>
```

Concatenating Lists Using +

- We can create a new list by adding two existing lists together

```
>>> a = [1, 2, 3]
>>> b = [4, 5, 6]
>>> c = a + b
>>> print(c)
[1, 2, 3, 4, 5, 6]
>>> print(a)
[1, 2, 3]
```

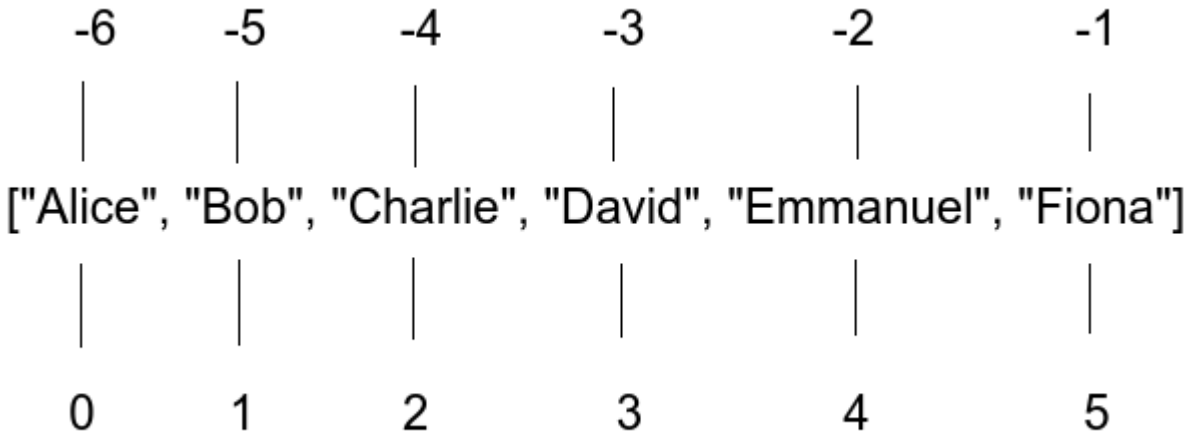
Lists Can Be Sliced Using :

```
>>> t = [9, 41, 12, 3, 74,
15]
>>> t[1:3]
[41,12]
>>> t[:4]
[9, 41, 12, 3]
>>> t[3:]
[3, 74, 15]
>>> t[:]
[9, 41, 12, 3, 74, 15]
```

- **Remember**: Just like in strings, the second number is *"up to but not including"*

Lists Can Be Sliced Using :

```
names = ["Alice", "Bob",
"Charlie", "David", "Emmanuel"
"Fiona"]

last_three = names[-3:]

print(last_three)
```



List Methods

```
>>> x = list()
>>> type(x)
<type 'list'>
>>> dir(x)
['append', 'count', 'extend', 'index', 'insert',
'pop', 'remove', 'reverse', 'sort']
>>>
```

List Methods

List Method	What It Does
`L.append(element)`	Adds `element` to the end of `L`.
`L.extend(aList)`	Adds the elements of `aList` to the end of `L`.
`L.insert(index, element)`	Inserts `element` at `index` if `index` is less than the length of `L`. Otherwise, inserts `element` at the end of `L`.
`L.pop()`	Removes and returns the element at the end of `L`.
`L.pop(index)`	Removes and returns the element at `index`.

Building a List from Scratch

- We can create an empty **list** and then add elements using the **append** method

- The **list** stays in order and new elements are **added** at the end of the **list**

```
>>> stuff = list()
>>> stuff.append('book')
>>> stuff.append(99)
>>> print(stuff)
['book', 99]
>>> stuff.append('cookie')
>>> print(stuff)
['book', 99, 'cookie']
```

Building a List from Scratch

```
>>> example = [1, 2]
>>> example
[1, 2]
>>> example.insert(1, 10)
>>> example
[1, 10, 2]
>>> example.insert(3, 25)
>>> example
[1, 10, 2, 25]
```

```
>>> example = [1, 2]
>>> example
[1, 2]
>>> example.append(3)
>>> example
[1, 2, 3]
>>> example.extend([11, 12, 13])
>>> example
[1, 2, 3, 11, 12, 13]
>>> example + [14, 15]
[1, 2, 3, 11, 12, 13, 14, 15]
>>> example
[1, 2, 3, 11, 12, 13]
```

Building a List from Scratch

```
>>> example
[1, 2, 10, 11, 12, 13]
>>> example.pop() # Remove the last element
13
>>> example
[1, 2, 10, 11, 12]
>>> example.pop(0) # Remove the first element
1
>>> example
[2, 10, 11, 12]
```

Is Something in a List?

- Python provides two **operators** that let you check if an item is in a list
- These are logical operators that return **True** or **False**
- They do not modify the list

```
>>> some = [1, 9, 21, 10, 16]
>>> 9 in some
True
>>> 15 in some
False
>>> 20 not in some
True
>>>
```

Lists are in Order

- A list can hold many items and keeps those items in the order until we do something to change the order

- A list can be **sorted** (i.e., change its order)

```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> friends.sort()
>>> print(friends)
['Glenn', 'Joseph', 'Sally']
>>> print(friends[1])
Joseph
>>>
```

Built-in Functions and Lists

- There are a number of **functions** built into Python that take lists as parameters

```
>>> nums = [3, 41, 12, 9, 74, 15]
>>> print(len(nums))
6
>>> print(max(nums))
74
>>> print(min(nums))
3
>>> print(sum(nums))
154
>>> print(sum(nums)/len(nums))
25.6
```

Best Friends: Strings and Lists

```
>>> abc = 'With three words'
>>> stuff = abc.split()
>>> print(stuff)
['With', 'three', 'words']
>>> print(len(stuff))
3
>>> print(stuff[0])
With
```

```
>>> print(stuff)
['With', 'three', 'words']
>>> for w in stuff :
...     print(w)
...
With
Three
Words
>>>
```

Split breaks a string into parts and produces a list of strings. We think of these as words. We can **access** a particular word or **loop** through all the words.

Excercises

- Write a Python program to read n integers into a list and separate the positive and negative numbers into two different lists.

From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> line = 'From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008'
>>> words = line.split()
>>> print(words)
['From', 'stephen.marquard@uct.ac.za', 'Sat', 'Jan', '5', '09:14:16', '2008']
>>>
```

```
fhand = open('mbox-short.txt')
for line in fhand:
    line = line.rstrip()
    if not line.startswith('From ') : continue
    words = line.split()
    print(words[2])
```

```
Sat
Fri
Fri
Fri
  …
```

The Double Split Pattern

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
print pieces[1]
```

stephen.marquard@uct.ac.za

The Double Split Pattern

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
print pieces[1]
```

stephen.marquard@uct.ac.za

```
pieces = email.split('@')
```

['stephen.marquard', 'uct.ac.za']

The Double Split Pattern

```
From stephen.marquard@uct.ac.za Sat Jan  5 09:14:16 2008
```

```
words = line.split()
email = words[1]
print pieces[1]
```

stephen.marquard@uct.ac.za

```
pieces = email.split('@')
```

['stephen.marquard', 'uct.ac.za']

```
print(pieces[1])
```

'uct.ac.za'

Python Dictionary

Python Dictionary

- Python dictionary is an ordered collection of items.
- It stores elements in **key/value** pairs. Here, **keys** are unique identifiers that are associated with each **value**.
- If we want to store information about countries and their capitals, we can create a dictionary with country names as keys and capitals as values.

```
capital_city = {"Nepal": "Kathmandu", "Italy": "Rome",
"England": "London"}
```

- **Keys** are "Nepal", "Italy", "England“
- **Values** are "Kathmandu", "Rome", "London"

Add Elements to a Python Dictionary

- We can add elements to a dictionary using the name of the dictionary with **[]**

```
capital_city = {"Nepal": "Kathmandu", "England": "London"}
print("Initial Dictionary: ",capital_city)

capital_city["Japan"] = "Tokyo"

print("Updated Dictionary: ",capital_city)
```

```
Initial Dictionary:  {'Nepal': 'Kathmandu', 'England': 'London'}
Updated Dictionary:  {'Nepal': 'Kathmandu', 'England': 'London',
'Japan': 'Tokyo'}
```

Change Value of Dictionary

- We can also use **[]** to change the value associated with a particular key.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print("Initial Dictionary: ", student_id)

student_id[112] = "Stan"

print("Updated Dictionary: ", student_id)
```

```
Initial Dictionary:  {111: 'Eric', 112: 'Kyle', 113: 'Butters'}
Updated Dictionary:  {111: 'Eric', 112: 'Stan', 113: 'Butters'}
```

Accessing Elements from Dictionary

- In Python, we use the keys to access their corresponding values

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

print(student_id[111])  # prints Eric
print(student_id[113])  # prints Butters
```

- If we try to access the value of a key that doesn't exist, we'll get an error.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print(student_id[211])

# Output: KeyError: 211
```

Removing elements from Dictionary

- We use the **del** statement to remove an element from the dictionary.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
print("Initial Dictionary: ", student_id)
del student_id[111]
print("Updated Dictionary ", student_id)
```

```
Initial Dictionary:  {111: 'Eric', 112: 'Kyle', 113: 'Butters'}
Updated Dictionary  {112: 'Kyle', 113: 'Butters'}
```

Removing elements from Dictionary

- We can also delete the whole dictionary using the **del** statement,

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}
# delete student_id dictionary
del student_id
print(student_id)

# Output: NameError: name 'student_id' is not defined
```

Python Dictionary Methods

len()	Return the length (the number of items) in the dictionary.
sorted()	Return a new sorted list of keys in the dictionary.
clear()	Removes all items from the dictionary.
keys()	Returns a new object of the dictionary's keys.
values()	Returns a new object of the dictionary's values

Dictionary Membership Test

- We can test if a key is in a dictionary or not using the keyword in.

```
# Membership Test for Dictionary Keys
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True
print(1 in squares) # prints True

print(2 not in squares) # prints True

# membership tests for key only not value
print(49 in squares) # prints false
```

Python Program to Count the Frequency of Each Word in a String using Dictionary

```
inputString=input("Enter the string:")
li=inputString.split() #converts the string into the list of words
freq = {} #dictionary to store word and its count
for item in li:
    if (item in freq):
        freq[item] += 1 #item already in the dictionary ,
                                    increment the count
    else:
        freq[item] = 1
print(freq)
```

Sort the dictionary by keys

```
myDict = {'ravi': 10, 'rajnish': 9,
        'sanjeev': 15, 'yash': 2, 'suraj': 32}

myKeys = list(myDict.keys())
myKeys.sort()
sorted_dict = {i: myDict[i] for i in myKeys}

print(sorted_dict)
```

Traversing a dictionary

```
statesAndCapitals = {
    'Gujarat': 'Gandhinagar',
    'Maharashtra': 'Mumbai',
    'Rajasthan': 'Jaipur',
    'Bihar': 'Patna'
}
print('List Of given capitals:\n')

for capital in statesAndCapitals.values():
    print(capital)
```

```
OUTPUT:

List Of given capitals:
Gandhinagar
Mumbai
Jaipur
Patna
```

```
statesAndCapitals = {
    'Gujarat': 'Gandhinagar',
   'Maharashtra': 'Mumbai',
   'Rajasthan': 'Jaipur',
   'Bihar': 'Patna'
}
 for key, value in statesAndCapitals.items():
    print(f"{key}: {value}")
```

```
Output:
Gujarat: Gandhinagar
Maharashtra: Mumbai
Rajasthan: Jaipur
Bihar: Patna
```

Reversed Look-up

```
test_dict = {‘You' : 4, ‘are' : 2, 'best' : 5}

print("The original dictionary : " + str(test_dict))
res = dict(reversed(list(test_dict.items())))
 print("The reversed order dictionary : " + str(res))
```

Output:
The original dictionary : {‘You': 4, ‘are’: 2, 'best': 5}
The reversed order dictionary : {'best': 5, ‘are': 2, ‘You': 4}

Tuples

Tuples

- A tuple is a type of sequence that resembles a list, except that, unlike a list, a **tuple is immutable**.
- You indicate a tuple literal in Python by enclosing its elements in **parentheses instead of square brackets**.

```
>>> fruits = ("apple", "banana")
>>> fruits
('apple', 'banana’)
>>> meats = ("fish", "poultry")
>>> meats
('fish', 'poultry')
>>> food = meats + fruits
>>> food
('fish', 'poultry', 'apple', 'banana')
>>> veggies = ["celery", "beans"]
>>> tuple(veggies)
('celery', 'beans')
```

Defining Simple Functions

Creating a Function

- In Python a function is defined using the **def** keyword:
- The syntax to declare a function is:

```
def function_name(arguments):
    # function body
    return
```

```
def my_function():
  print("Hello from a function")
```

Calling a Function

- To call a function, use the function name followed by parenthesis:

```
def my_function():
  print("Hello from a function")

my_function()
```

Arguments

- Information can be passed into functions as arguments.

- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

```
# function with two arguments
def add_numbers(num1, num2):
    sum = num1 + num2
    print('Sum: ',sum)

# function call with two values
add_numbers(5, 4)
```

Arbitrary Arguments, *args

- If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.
- This way the function **will receive a tuple of arguments**, and can access the items accordingly:

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function("Emil", "Tobias", "Linus")
```

```
The youngest kid is Linus
```

Keyword Arguments

- You can also send arguments with the key = value syntax.
- This way the order of the arguments does not matter.

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")
```

Arbitrary Keyword Arguments, **kwargs

- If you do not know how many keyword arguments that will be passed into your function, add two asterisk: ** before the parameter name in the function definition.

- This way the function **will receive a dictionary of arguments**, and can access the items accordingly:

```
def my_function(**kid):
  print("His last name is " + kid["lname"])

my_function(fname = "Tobias", lname = "Refenes")
```

Default Parameter Value

- The following example shows how to use a default parameter value.

- If we call the function without argument, it uses the default value:

```
def my_function(country = "Norway"):
  print("I am from " + country)

my_function("Sweden")
my_function("India")
my_function()
my_function("Brazil")
```

Passing a List as an Argument

- You can send any data types of argument to a function (string, number, list, dictionary etc.), and it will be treated as the same data type inside the function.

- E.g. if you send a List as an argument, it will still be a List when it reaches the function:

```
def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

Return Values

- To let a function return a value, use the **return** statement:

```
def square(x):
"""Returns the square of x."""
      return x * x
```

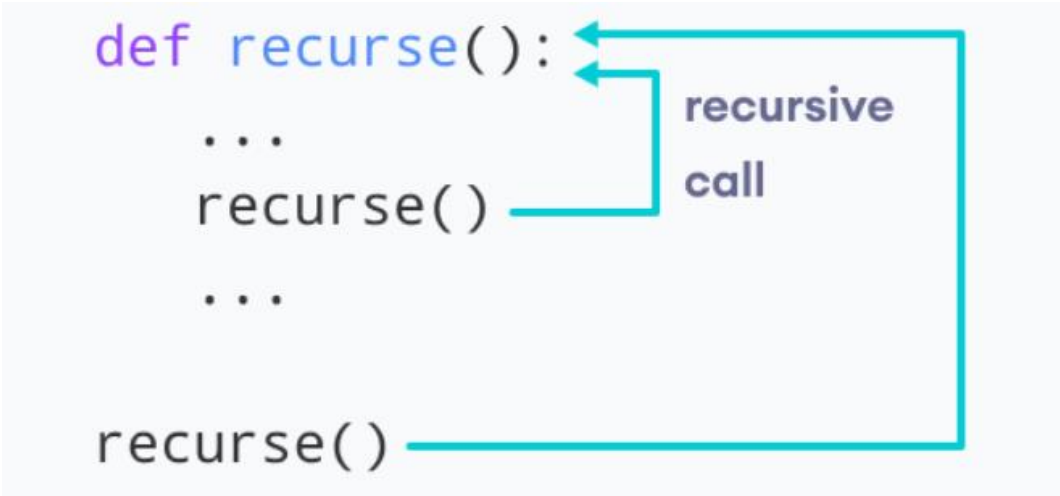
```
>>> square(2)
4
>>> square(6)
36
>>> square(2.5)
6.25
```

```
def average(lyst):
"""Returns the average of the
numbers in lyst."""
      theSum = 0
      for number in lyst:
            theSum += number
      return theSum / len(lyst)
```

```
>>> average([1, 3, 5, 7])
4.0
```

Python Recursive Function

- In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of construct are termed as recursive functions.
- The following image shows the working of a recursive function called recurse



Example of a recursive function

```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Boolean Functions

```
def odd(x):
"""Returns True if x is odd or False otherwise."""
      if x % 2 == 1:
            return True
      else:
            return False
```

```
>>> odd(5)
True
>>> odd(6)
False
```

Tutorial Questions

- Define a function named **even**. This function expects a number as an argument and returns **True** if the number is divisible by 2, or it returns **False** otherwise. (Hint: A number is evenly divisible by 2 if the remainder is 0.)
- Define a function named **summation**. This function expects two numbers, named **low** and **high**, as arguments. The function computes and returns the sum of the numbers between **low** and **high**, inclusive.
- Write a Python function to find the maximum of three numbers.
- Write a Python program to reverse a string.
  - Sample String : "1234abcd"
  - Expected Output : "dcba4321"

Tutorial Questions

- Write a Python function that accepts a string and counts the number of upper and lower case letters.
  - Sample String : 'The quick Brow Fox'
  - Expected Output :
    - No. of Upper case characters : 3
    - No. of Lower case Characters : 12
- Write a Python function that takes a list and returns a new list with distinct elements from the first list.
  - Sample List : [1,2,3,3,3,3,4,5]
  - Unique List : [1, 2, 3, 4, 5]
- Write a program to create a function show_employee() using the following conditions.
  - It should accept the employee's name and salary and display both.
  - If the salary is missing in the function call then assign default value 9000 to salary

Tutorial Questions

- Write a program to create a recursive function to calculate the sum of numbers from 0 to 10.

- Define a function which counts vowels and consonant in a word.
  - Input:
  - Enter a word = pythonlobby
  - Expected output
  - Count of vowel is = 2
  - Count of consonant is = 9

Tutorial Questions

- Define a function that accepts radius and returns the area of a circle.
  - Input:
    - Enter radius 4
  - Expected output
    - Area of a circle is 50.24
- Create a function min_max() that takes n numbers as list argument and return the smallest and largest numbers.

- Write a Python program to read n integers into a list and separate the positive and negative numbers into two different lists.
- Create a dictionary of names and birthdays. Write a Python program that asks the user to enter a name, and the program display the birthday of that person.

Tutorial Questions

- Write a Python program to count how many times each character appears in a given string and store the count in a dictionary with key as the character.

Python Sets

Python Sets

- A set is a collection of unique data. That is, elements of a set cannot be duplicate. For example,
- Suppose we want to store information about student IDs. Since student IDs cannot be duplicate, we can use a set.

```
# create a set of integer type
student_id = {112, 114, 116, 118, 115}
print('Student ID:', student_id)
# create a set of string type
vowel_letters = {'a', 'e', 'i', 'o', 'u'}
print('Vowel Letters:', vowel_letters)
# create a set of mixed data types
mixed_set = {'Hello', 101, -2, 'Bye'}
print('Set of mixed data types:', mixed_set)
```

Python Sets

```
# typecasting list to set
myset = set(["a", "b", "c"])
print(myset)

# Adding element to the set
myset.add("d")
print(myset)
```

Python set is an unordered datatype, ie, we cannot know in which order the elements of the set are stored.

Thus Output of above code can be

{'c', 'b', 'a'} {'d', 'c', 'b', 'a'}

```
# values of a set cannot be changed
myset = {“One", “Two", “Three"}
print(myset)
myset[1] = "Hello"
print(myset)
```

Adding elements to set

```
people = {"Jay", "Idrish", "Archi"}

print("People:", end = " ")
print(people)

people.add("Daxit")
print("\nSet after adding element:", end = " ")
print(people)
```

Output:
People: {'Idrish', 'Archi', 'Jay’}
Set after adding element: {'Idrish', 'Archi', 'Jay', 'Daxit'}

Set Operations- Union

```
town = {"Jay", "Idrish", "Archil"}
village = {"Karan", "Arjun"}
city = {"Deepanshu", "Raju"}

population = town.union(villege)
print(population)

# Union using "|“ operator
population = town|city

print(population)
```

Set Operations - Intersection

```
set1 = set()
set2 = set()
 for i in range(5):
    set1.add(i)
 for i in range(3,9):
    set2.add(i)

# Intersection using intersection() function
set3 = set1.intersection(set2)
 print(set3)

# Intersection using "&" operator
set3 = set1 & set2
 print(set3)
```

Program to perform different set operations Union, Intersection, Difference and Symmetric Difference

```
A = {0, 2, 4, 6, 8};
B = {1, 2, 3, 4, 5};

print("Union :", A | B)
 print("Intersection :", A & B)
 print("Difference :", A - B)
 print("Symmetric difference :", A ^ B)
```