**PYTHON:**

Introduction to Python(Why, Where, How)

Installation (Mac, Windows)

Getting Friendly with Python Syntax

Indentation and Comments

Data types in Python

Changing data types in python

Conditionals in Python(if else, nested if else)

Loops in Python(for, while)

Strings and its functions in Python

Lists and its functions in Python

Tuple and its functions in Python

Dictionary and its functions in Python

Functions in Python and Exception Handling

**PYTHON**

* **STRING:**

# Initialize a string

str = 'python programming is fun'

str2= "python is easy"

# multi line string

str3 =''' python is fundamentally simple

and easy to learn'''

print('str = ', str)

print('str2 = ', str2)

print('str3 = ', str3)

# extract characters using index

#first character

print('str[0] = ', str[0])

#print('str[25] = ' , str[25]) # will raise index out of range

#error

#last character

print('str[-1] = ', str[-1])

# extract a part of the string using a range

#slicing 2nd to 5th character

print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character

print('str[5:-2] = ', str[5:-2])

#Generates a reversed string

print(‘str[::-1]=’,str[::-1])

# modifying a character

#str[3]='A' # will raise error.

# String does not support reassignment of an item

print('modified str = ', str)

str ="PYTHON PROGRAMMING IS FUN" # redefine the entire string

print('reassigned str = ', str)

#Concatenate strings

print('str + str2 = ', str + str2)

# To repeat the string for a given number of times.

print('str2 \* 3 =', str2 \* 3)

* **LISTS :**

• A set of elements. A vector of elements

identified by the position.

• List is Mutable.

# creating lists

# empty list

my\_list1 = []

or

my\_list1=list()

print("Empty List :", my\_list1)

range()

print(list(range(10)))

print(list(range(10)))

print(list(range(2,8)))

print(list(range(2,20,2)))

print(list(range(1,50,3)))

print(list(range(20,10,-1)))

print(list(range(20,10,-2)))

# list of integers

my\_list2 = [1, 2, 3]

print("Integer List :", my\_list2)

# list with mixed datatypes

my\_list3 = [1, "Hello", 3.4]

print("Mixed type List :", my\_list3)

# list with duplicate data items

my\_list4 = [1, "Hello", 1, 2, 3.4]

print("Duplicate values in List :", my\_list4)

# nested list

my\_list5 = ["mouse", [8, 4, 6], ['a'], 21, 3.14]

print("Nesting of List :", my\_list5)

# use indexing to access list elements.

# integer index starting from 0

my\_list =

['apple','banana','orange','grape','lichi','strawberry','mango']

# Output: apple

print(my\_list[0])

# Output: orange

print(my\_list[2])

# Output: lichi

print(my\_list[4])

# Output: mango

print(my\_list[-1]) # -1 index refers to the last element

# Will raise Error! Only integer can be used for indexing

# print(my\_list[4.0])

# Nested List

n\_list = ["Happy", [2,0,1,7] ,['aa','bb','cc','dd']]

# Lists are mutable. We can change the list elements directly

odd = [2, 4, 6, 8]

print("Original List: ", odd)

# change the 1st item

odd[0] = 1

print("changed the first item List: ", odd)

# modifying the element

list[2] = [2011,2015,2016];

print ("List with New value at index 2 : ", list)

**Python List Methods :**

1. append() - Add an element to the end of the list
2. extend() - Add all elements of a list to the another list
3. insert() - Insert an item at the defined index
4. remove() - Removes an item from the list
5. pop() - Removes and returns an element at the given index
6. clear() - Removes all items from the list
7. index() - Returns the index of the first matched item
8. count() - Returns the count of number of items passed as an argument
9. sort() - Sort items in a list in ascending order
10. reverse() - Reverse the order of items in the list
11. copy() - Returns a shallow copy of the list

# append example

odd = [2, 4, 6, 8]

print("original list : ",odd)

# append will add one value at the end

odd.append(7)

print("after append : ",odd)

# extend will add multiple values at the end

odd.extend([9, 11, 13])

print("After extend :",odd)

# + operator will concatenate lists

odd += [100,200,300]

print("concatenated list: ", odd)

# insertion at a specific place

odd1 = [1, 9]

print("before insert into odd1 ",odd1)

# insert at position 1 , value 3

odd1.insert(1,3)

print(" after insert into odd1: ",odd1)

my\_list = ['p','r','o','b','l','e','m']

# delete one item

del my\_list[2]

# Output: ['p', 'r', 'b', 'l', 'e', 'm']

print(my\_list)

# delete multiple items

del my\_list[1:5]

# Output: ['p', 'm']

print(my\_list)

# delete entire list

del my\_list

my\_list = [3, 8, 1, 6, 0, 8, 4]

# Output: 1 the index of element 8

print(my\_list.index(8))

# Output: 2 the count of occurances of 8

print(my\_list.count(8))

# prints None. Indicates return value is None

# only sorts

print(my\_list.sort())

# Output: [0, 1, 3, 4, 6, 8, 8]

print(my\_list)

my\_list.reverse()

# Output: [8, 8, 6, 4, 3, 1, 0]

print(my\_list)

# remove() removes the first matching value

# del removes the item at the specific index

# pop removes the specific indexed item and returns the value

# Stack like operations on List

my\_list = ['p','r','o','b','l','e','m']

my\_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']

print("After remove : ",my\_list)

# Output: ‘o’

print(my\_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']

print("After pop(1) .. ",my\_list)

# Output: 'm'

print("after pop() .. : ", my\_list.pop()) # pop() removes the top item of the list

my\_list.clear() # Output: []

* **TUPLES :**

# empty tuple

# Output: ()

my\_tuple = ()

print(my\_tuple)

# tuple having integers

# Output: (1, 2, 3)

my\_tuple = (1, 2, 3)

print(my\_tuple)

# tuple with mixed datatypes

# Output: (1, "Hello", 3.4)

my\_tuple = (1, "Hello", 3.4)

print(my\_tuple)

# nested tuple

# Output: ("mouse", [8, 4, 6], (1, 2, 3))

my\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

print(my\_tuple)

# tuple can be created without parentheses

# also called tuple packing

# Output: 3, 4.6, "cat"

my\_tuple = 3, 4.6, "cat"

print(my\_tuple)

# tuple unpacking is also possible

a, b, c = my\_tuple

print(a)

print(b)

print(c)

# convert a tuple into a list

my\_list = [a,b,c] or my\_list=list(my\_tuple)

print(my\_list)

# tuple with only one element needs a comma within ()

# this is treated as string

my\_tuple = ("hello")

print(type(my\_tuple))

# need a comma at the end

# Output: <class 'tuple'>

my\_tuple = ("hello",)

print(type(my\_tuple))

# nested tuple

n\_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index

print(n\_tuple[0][3])

# slicing is same as in lists

print(my\_tuple[1:4])

#Deleting a tuple

del my\_tuple

# tuple elements are non mutable. They cannot be changed.

my\_tuple = (4, 2, 3, [6, 5])

* **DICTIONARY :**

**Dictionary is an unordered collection of items that has a key: value pair. Optimised to retrieve values when a key is known.**

1.Creating and adding elements to dictionary

# empty dictionary

my\_dict = {}

# dictionary with integer keys

my\_dict = {1: 'apple', 2: 'ball'}

print (my\_dict[1]) # print value with key =1

print (my\_dict[2]) # print value with key =2

Computer Engg. Department, SAKEC 2

# dictionary with mixed keys

my\_dict = { 'name' : 'John', 1: 25, 'a':'apple'}

print(my\_dict['a'])

#print (my\_dict[2]) #KeyError

my\_dict = {'name': 'John', 1: [2, 4, 3]}

print (my\_dict['name'])

# initializing using dict()

my\_dict = dict({1:'apple', 2:'ball'})

print (my\_dict[1]) # print value with key =1

# how to change elements in a dictionary

employee ={'name': ‘ABC', 'age': 30 }

print("Original :",employee)

# update value

employee['age']=27

print("Updated :" ,employee)

# add a new attribute

employee['address'] = 'Mulund'

print("New details of Employee :", employee)

2. Removing elements from dictionary

squares = {1:1, 2:4, 3:9, 4:16, 5:25, 6:36}

print("Original Dict is : ",squares)

# remove a particular item using pop()

# provide key value to delete

print("Value popped is : ",squares.pop(4))

# Output: {1: 1, 2: 4, 3: 9, 5: 25, 6:36}

print("After pop :", squares)

# delete a particular item given by key

del squares[5]

# Output: {2: 4, 3: 9, 6:36}

print(squares)

# remove all items

squares.clear()

print(squares)

del squares

# display items in the dictionary

print ("items of the dictionary are :" , squares.items())

Operations on a dictionary

Computer Engg. Department, SAKEC 7

# display all keys

print("keys are :", squares.keys())

# display all values

print("keys are :", squares.values())

**CONDITIONAL STATEMENTS :**

* **if statement**

Syntax:

if expression:

statement(s)

* **if...else statement**

Syntax:

if expression:

statement(s)

else:

statement(s)

* **if...elif...else statement**

Syntax:

if expression1:

statement(s)

elif expression2:

statement(s)

elif expression3:

statement(s)

else:

statement(s)

**LOOPS:**

* **for loop:**

**It has the ability to iterate over the items of any sequence, such as a list or a string.**

**Syntax:**

for iterating\_var in sequence:

statements(s)

#examples

1.

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

**for x in fruits:**

**print(x)**

2.

**for x in range(2, 6):**

**print(x)**

* **while loop**

**A while loop statement in Python programming language repeatedly executes a target**

**statement as long as a given condition is true.**

**Syntax:**

while expression:

statement(s)

#examples

1.

i = 1

while i < 6:

print(i)

i += 1

2.

i = 0

while i < 6:

i += 1

if i == 3:

continue

print(i)

**FUNCTION:**

def functionname( parameters ):

   "function\_docstring"

   function\_suite

   return [expression]

Example:

def printme( str ):

   "This prints a passed string into this function"

   print str

   return

**FLASK:**

* **Introduction to flask(Why, Where, How)**
  + Flask is a microframework for Python based on Werkzeug, Jinja 2**.**
  + Flask is also easy to get started with as a beginner because there is little boilerplate code for getting a simple app up and running.
  + Flask is also easy to get started with as a beginner. You only need this below code for running your classy Hello World Webpage. Obviously with Python and flask installed on you machine.
* **Environment Setup**
  + Install python
  + Install PIP
  + Install Flask
* **Basic app**

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/')**

**def hello\_world():**

**return 'Hello, World!'**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run()**

* **Routing in flask**

Modern web frameworks use the routing technique to help a user remember application URLs. It is useful to access the desired page directly without having to navigate from the home page.

The route() decorator in Flask is used to bind URL to a function

**@app.route(‘/hello’)**

**def hello\_world():**

**return ‘hello world’**

Here, URL ‘/hello’ rule is bound to the hello\_world() function.

* **Variables in flask**

1 int accepts integer

2 float For floating point value

3 path accepts slashes used as directory separator character

**from flask import Flask**

**app = Flask(\_\_name\_\_)**

**@app.route('/blog/<int:postID>')**

**def show\_blog(postID):**

**return 'Blog Number %d' % postID**

**@app.route('/rev/<float:revNo>')**

**def revision(revNo):**

**return 'Revision Number %f' % revNo**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run()**

Mapping URL in flask

* **HTTP methods in flask**

Http protocol is the foundation of data communication in world wide web. Different methods of data retrieval from specified URL are defined in this protocol.

1 **GET** Sends data in unencrypted form to the server. Most common method.

2 **POST** Used to send HTML form data to server. Data received by POST method is not cached by server.

By default, the Flask route responds to the **GET** requests. However, this preference can be altered by providing methods argument to **route()** decorator.

In order to demonstrate the use of **POST** method in URL routing, first let us create an HTML form and use the **POST** method to send form data to a URL.

**<html>**

**<body>**

**<form action = "/login" method = "post">**

**<p>Enter Name:</p>**

**<p><input type = "text" name = "nm" /></p>**

**<p><input type = "submit" value = "submit" /></p>**

**</form>**

**</body>**

**</html>**

**FLASK CODE**

**from flask import Flask, redirect, url\_for, request**

**app = Flask(\_\_name\_\_)**

**@app.route('/success/<name>')**

**def success(name):**

**return 'welcome %s' % name**

**@app.route('/login',methods = ['POST', 'GET'])**

**def login():**

**if request.method == 'POST':**

**user = request.form['nm']**

**return redirect(url\_for('success',name = user))**

**else:**

**user = request.args.get('nm')**

**return redirect(url\_for('success',name = user))**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug = True)**

* **HTML templates in flask**

**Flask will try to find the HTML file in the templates folder, in the same folder in which this script is present.**

**Application folder**

* + **Hello.py**
  + **templates**
    - **Hello.html**

Flask uses jinga2 template engine. A web template contains HTML syntax interspersed placeholders for variables and expressions (in these case Python expressions) which are replaced values when the template is rendered.

The **Jinga2** template engine uses the following delimiters for escaping from HTML.

* {% ... %} for Statements
* {{ ... }} for Expressions to print to the template output
* {# ... #} for Comments not included in the template output
* **Static flies in flask**

A web application often requires a static file such as a javascript file or a CSS file supporting the display of a web page. Usually, the web server is configured to serve them for you, but during the development, these files are served from *static* folder in your package or next to your module and it will be available at */static* on the application.

A special endpoint ‘static’ is used to generate URL for static files.

In the following example, a javascript function defined in hello.js is called on OnClick event of HTML button in index.html, which is rendered on ‘/’ URL of the Flask application.

**<html>**

**<head>**

**<script type = "text/javascript"**

**src = "{{ url\_for('static', filename = 'hello.js') }}" ></script>**

**</head>**

**<body>**

**<input type = "button" onclick = "sayHello()" value = "Say Hello" />**

**</body>**

**</html>**

# **Sending Form Data to Template**

The **results()** function collects form data present in **request.form** in a dictionary object and sends it for rendering to **result.html**.

The template dynamically renders an HTML table of **form** data.

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route('/')

def student():

return render\_template('student.html')

@app.route('/result',methods = ['POST', 'GET'])

def result():

if request.method == 'POST':

result = request.form

return render\_template("result.html",result = result)

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)

<html>

<body>

<form action = "http://localhost:5000/result" method = "POST">

<p>Name <input type = "text" name = "Name" /></p>

<p>Physics <input type = "text" name = "Physics" /></p>

<p>Chemistry <input type = "text" name = "chemistry" /></p>

<p>Maths <input type ="text" name = "Mathematics" /></p>

<p><input type = "submit" value = "submit" /></p>

</form>

</body>

</html>

<!doctype html>

<html>

<body>

<table border = 1>

{% for key, value in result.items() %}

<tr>

<th> {{ key }} </th>

<td> {{ value }} </td>

</tr>

{% endfor %}

</table>

</body>

</html>

* **Passing Object in flask**
* **Cookies & Session in flask**

Session data is stored on server. Session is the time interval when a client logs into a server and logs out of it. The data, which is needed to be held across this session, is stored in a temporary directory on the server.

A session with each client is assigned a Session ID. The Session data is stored on top of cookies and the server signs them cryptographically. For this encryption, a Flask application needs a defined SECRET\_KEY.

Session object is also a dictionary object containing key-value pairs of session variables and associated values.

**@app.route('/login', methods = ['GET', 'POST'])**

**def login():**

**if request.method == 'POST':**

**session['username'] = request.form['username']**

**return redirect(url\_for('index'))**

**return '''**

**@app.route('/logout')**

**def logout():**

**# remove the username from the session if it is there**

**session.pop('username', None)**

**return redirect(url\_for('index'))**

* **Sending Mail**

**from flask import Flask**

**from flask\_mail import Mail, Message**

**app =Flask(\_\_name\_\_)**

**mail=Mail(app)**

**app.config['MAIL\_SERVER']='smtp.gmail.com'**

**app.config['MAIL\_PORT'] = 465**

**app.config['MAIL\_USERNAME'] = 'yourId@gmail.com'**

**app.config['MAIL\_PASSWORD'] = '\*\*\*\*\*'**

**app.config['MAIL\_USE\_TLS'] = False**

**app.config['MAIL\_USE\_SSL'] = True**

**mail = Mail(app)**

**@app.route("/")**

**def index():**

**msg = Message('Hello', sender = 'yourId@gmail.com', recipients = ['id1@gmail.com'])**

**msg.body = "Hello Flask message sent from Flask-Mail"**

**mail.send(msg)**

**return "Sent"**

**if \_\_name\_\_ == '\_\_main\_\_':**

**app.run(debug = True)**