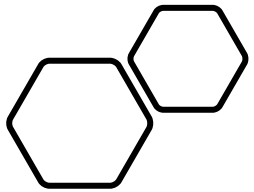
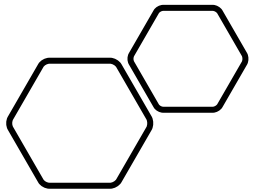


# Lab 13
# Flask and OOPs concept

# Application Programming Interface (API)

- SOAP APIs (Simple Object Access Protocol)
  - A protocol
  - Need more bandwidth, mostly consists of XML data
  - WSDL file contains all the information of the services

- REST APIs (Representational State Transfer)
  - An architectural style
  - Doesn’t need much bandwidth, mostly consists of JSON data
  - Uses Uniform Service Locators



# REST API using Flask

# Flask

- A micro web application framework
- Based on Werkzeug WSGI (Web Server Gateway Interface) toolkit
- Contains server and debugger
- RESTful request dispatching

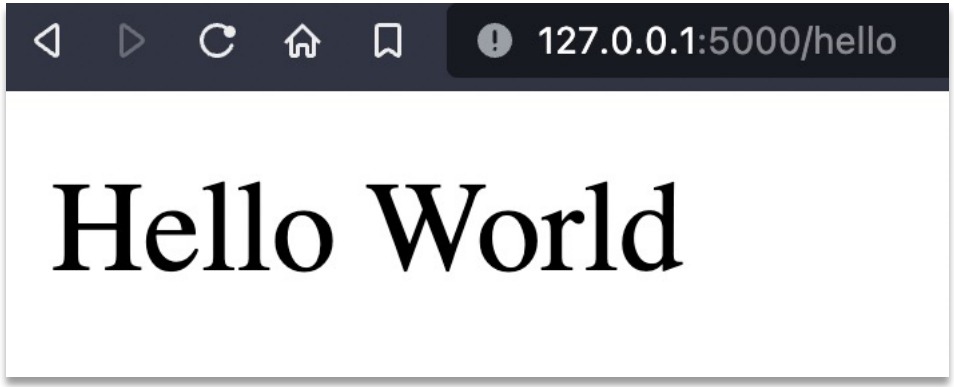
- Installation –

  **pip install Flask**

# Flask – Hello World (Basic Example)

```python
from flask import Flask

app = Flask(__name__)

@app.route("/hello")
def get_hello():
    return "Hello World"


if "__main__" == __name__:
    app.run(host="127.0.0.1", port="5000", debug=True)
```

Output:



# Flask – Database Management

- **Flask-SQLAlchemy** is a powerful **ORM**
- It provides efficient and high-performing database access for relational databases

- Installation –

```
pip install flask-sqlalchemy
```

```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///user.db'

db = SQLAlchemy(app)

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)


db.create_all()
```

# Flask – Session Management

- Flask-Login provides user session management for Flask
- handles the common tasks of
  - logging in,
  - logging out, and
  - remembering your users' sessions

- Installation –

  **pip install flask-login**

```python
# Importing required modules
from flask_login import (LoginManager, login_manager,
    login_user, logout_user, login_required, UserMixin)

# Configuring the application
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///user.db'
app.config['SECRET_KEY'] = 'secretkey'

db = SQLAlchemy(app)
login_manager = LoginManager()

# Initializing using SECRET KEY
login_manager.init_app(app)

# User Model
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password = db.Column(db.String(80), nullable=False)

# Load the user object from the user ID stored in the session
@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

# Flask – HTTP Methods

- POST methods
  - To extract JSON from request body

    req_body = request.get_json()

**server.py**

```
@app.route('/do_signin', methods = ['POST'])
def do_signin():
    if(request.method=='POST'):
        req = request.get_json()
        username = req['username']
        password = req['password']
        check_user = User.query.filter_by(username=username).first()
        if(check_user is not None):
            if(check_user.password == password):
                login_user(check_user)
                return "LOGGED in successfully"
            else:
                return "Incorrect Password"
        else:
            return "No such User exists"
```

**client.py**

```
def signin():
    username = input("Enter username: ")
    password = input("Enter password: ")

    payload = {
        "username": username,
        "password": password
    }
    resp = requests.post("http://127.0.0.1:8000/do_signin", json=payload).content.decode()

    print(resp)
```

**output**

```
Enter option:
1. SignUp
2. SignIn
3. SignOut
->      2
Enter username: qwerty
Enter password: 123456
Response from server =  LOGGED in successfully
```

# Flask – HTTP Methods

- GET methods
  - Default:

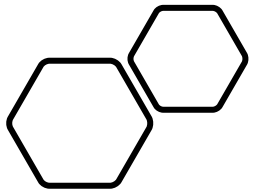
```
@app.route("/hello")
def get_hello():
    return "Hello World"
```

  - Path variable:

```
@app.route('/user/<id>')
def user_id(id):
    return "The user ID is " + id
```

  - Query params:

```
@app.route('/params')
def get_params():
    params = request.args
    return "The parameter passed is " + params.get('username')
```



# SOAP API using Flask-Spyne

# SOAP APIs

- Python library required – **Flask-Spyne**

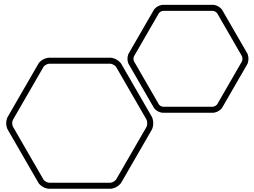
[pip install Flask-Spyne]

*server.py*

```python
class HelloWorldService(ServiceBase):
    @rpc(Unicode, Integer, _returns=Iterable(Unicode))
    def say_hello(ctx, name, times):
        for i in range(times):
            yield u'Hello, %s' % name


application = Application([HelloWorldService], 'examples.hello.soap',
                          in_protocol=Soap11(validator='lxml'),
                          out_protocol=Soap11())

wsgi_application = WsgiApplication(application)


if __name__ == '__main__':
    from wsgiref.simple_server import make_server

    server = make_server('127.0.0.1', 8000, wsgi_application)
    server.serve_forever()
```

*client.py*

```python
from suds.client import Client

c = Client('http://localhost:8000/soap?wsdl')
print(c.service.say_hello("Willem", 10))
```



# OOPs Concept

# Classes and Objects

- A class is a collection of objects.
- It is a logical entity that contains some attributes and methods.

- The object is an entity that has a state and behaviour associated with it.
- An object consists of :
  - *State* (attributes)
  - *Behaviour* (methods)
  - *Identity* (name)

```python
class Person:
    name:str
    age:int

    def __init__(self, name, age) -> None:
        self.name = name
        self.age = age

    def details(self) -> None:
        print(f'Person Data: [ Name: {self.name}, Age: {self.age} ]')
        return


if "__main__"==__name__:
    p = Person("Ashish", 26)

    # Output-> Person Data: [ Name: Ashish, Age: 26 ]
    p.details()
```

# Encapsulation

- Encapsulation describes the idea of **wrapping data and the methods** that work on data within **one unit**.
- Puts restrictions on accessing variables and methods directly and can **prevent** the **accidental modification of data**.
- An object's variable can only be changed by an object's method.

```python
class Sensor():
    def __init__(self, name):
        self.name = name
        self.__version = '1.0' # Encapsulated private variable

    # a getter function
    def get_version(self):
        print(f'The sensor version is {self.__version}')

    # a setter function
    def set_version(self, version):
        self.__version = version


sensor1 = Sensor('Acc')

print(sensor1.name)
print(sensor1.get_version())
```

Output :

```
Acc
The sensor version is 1.0
```

# Abstraction

- Abstraction is defined as a process of **handling complexity** by **hiding unnecessary information** from the user.

- We need to import the abc module, which provides the base for defining **Abstract Base classes (ABC)**.

```
from abc import ABC, abstractmethod
class Absclass(ABC):
    def print(self,x):
        print("Passed value: ", x)
    @abstractmethod
    def task(self):
        print("We are inside Absclass task")

class test_class(Absclass):
    def task(self):
        print("We are inside test_class task")

class example_class(Absclass):
    def task(self):
        print("We are inside example_class task")

#object of test_class created
test_obj = test_class()
test_obj.task()
test_obj.print(100)

#object of example_class created
example_obj = example_class()
example_obj.task()
example_obj.print(200)

print("test_obj is instance of Absclass? ", isinstance(test_obj, Absclass))
print("example_obj is instance of Absclass? ", isinstance(example_obj, Absclass))
```

**Output**

```
We are inside test_class task
Passed value:  100
We are inside example_class task
Passed value:  200
test_obj is instance of Absclass?  True
example_obj is instance of Absclass?  True
```

# Access Modifiers

```
class Person:
    def __init__(self, name, age, email) -> None:
        self.name = name
        self._email = email
        self.__age = age
```

Public variable

Protected variable

Private variable

- **Public Member**: Accessible anywhere from otside oclass.
- **Private Member**: Accessible within the class
- **Protected Member**: Accessible within the class and its sub-classes

# Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Base class** is the class being inherited from.
- **Derived class** is the class that inherits from another class.
- It provides the reusability of a code.

**Base Class**

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)

x = Person("John", "Doe")
x.printname()
```

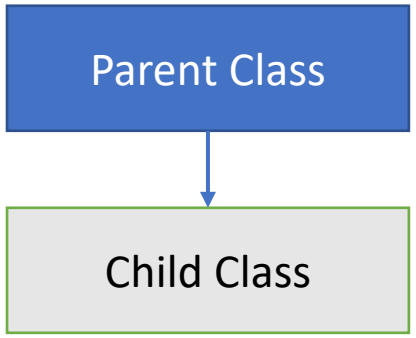
**Derived Class**

```
class Student(Person):
  pass

x = Student("Mike", "Olsen")
x.printname()
```

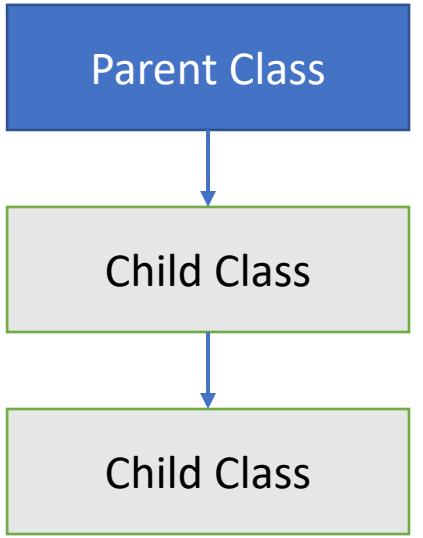
**Output**

```
Mike Olsen
```

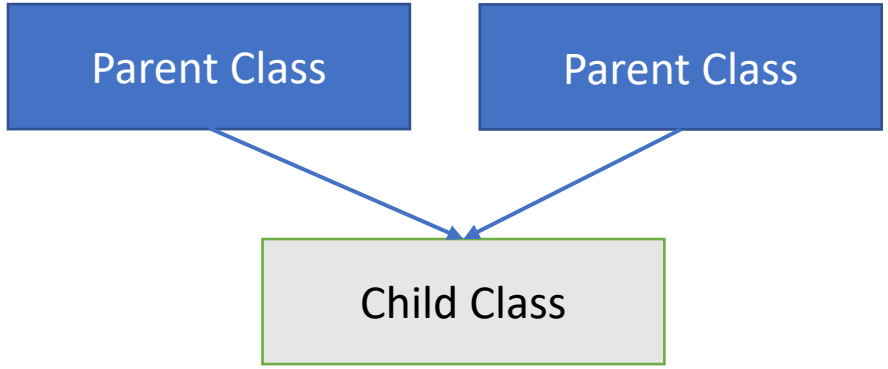
# Type of Inheritance



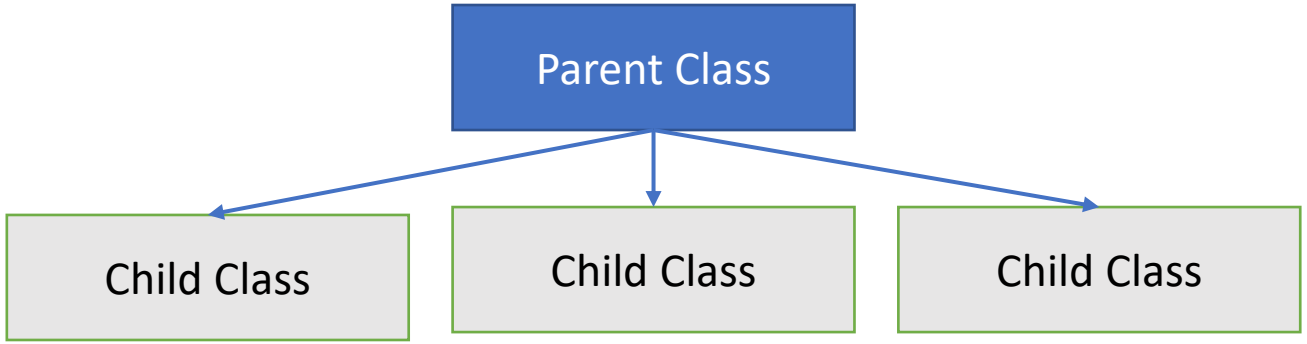

**Single Inheritance**




**Multi-level Inheritance**




**Multiple Inheritance**




**Hierarchical Inheritance**

# Polymorphism

- The word polymorphism means **having many forms**.
- In programming, polymorphism means the **same function** name (but **different signatures**) being used for different types.
- Example – Method Overloading, Method Overriding

```
# A simple Python function to demonstrate
# Polymorphism

def add(x, y, z = 0):
    return x + y + z

# Driver code
print(add(2, 3))
print(add(2, 3, 4))
```

**Output**

```
5
9
```