To run name server

Start name server by running 'pyro4-ns' in the terminal. Results illustrate localhost port.

```
NS running on localhost:9090 (127.0.0.1)
Warning: HMAC key not set. Anyone can connect to this server!
URI = PYRO:Pyro.NameServer@localhost:9090
```

Once name server is created, the code under 'server.py' in sublime can run, as highlighted below.

```
Last login: Tue Apr 18 07:29:33 on ttys001
Ffions-MacBook-Pro:~ ffionvrichards$ python '/Users/ffionvrichards/Downloads/cmt-202-master-f7ad547a0d90c26c51fd19e4e522784834a482a0/server.py'
Sales agency created.
Sales agency registered.
Properties available.
```

The code below highlights what is needed in order for the above code to run correctly as well as follow all tasks mentioned.

Server.py

```
1  from __future__ import print_function
2  import Pyro4
3  import uuid
4
5  #Gives remote access to class
6  @Pyro4.expose
7  class SalesAgency(object):
8      def __init__(self):
9          self._properties = {}
10
11     @property
12     def properties(self):
13         return self._properties
14
15
16 #Add a property
17     def add(self, owner, number, postcode, price):
18         prop = {
19             "Owner": owner,
20             "Number": number,
21             "Postcode": postcode,
22             "Price": price
23         }
24 #Give unique ID number
25 #ID is unique due to uuid
26         unique_id = uuid.uuid4()
27         if unique_id not in self._properties:
28             self._properties[unique_id] = prop
29 #Delete property
30     def delete(self, unique_id):
31         del self._properties[unique_id]
32
```

This code illustrates how to give remote access to the sales agency class, further including details on the attributes of the class. This allows a property to be added given the attributes of 'owner, number, postcode, and price'. In addition the unique ID has been created due to the uuid which can then be used in order to search for a property and then for it to be deleted.

Further, below illustrates the creation of a pyro server for the sales agency giving details of an example property.

```python
# Pyro server for the warehouse
def main():
    sale_agency = SalesAgency()
    sale_agency.add("David Bean", 16, "CF24 4QD", 250000)
    print("Sales agency created.")

    with Pyro4.Daemon() as daemon:
        sale_agency_uri = daemon.register(sale_agency)
        print("Sales agency registered.")

        with Pyro4.locateNS() as ns:
            ns.register("sales_agency", sale_agency_uri)

        print("Properties available.")
        daemon.requestLoop()


if __name__ == "__main__":
    main()

```

employee.py

```
1  from __future__ import print_function
2  import sys
3  import Pyro4
4
5  #runs in different versions of python
6  if sys.version_info < (3, 0):
7      input = raw_input
8
```

Raw input and input are very similar however it allows for the code to run on python version. Raw returns a string, yet Input runs as a python expression.

```
10 # Employee's details
11 class Employee(object):
12     def __init__(self, name):
13         self.name = name
14
15 # Search by Price Range
16     def search_by_price_range(self, sales_agency, minimum, maximum):
17         results = set()
18         for prop in sales_agency.properties:
19             if prop["Price"] > minimum and prop["Price"] < maximum:
20                 results.add(prop)
21         return results
22 #Search by postcode
23     def search_by_postcode(self, sales_agency, postcode):
24         results = set()
25         for prop in sales_agency.properties:
26             if prop["postcode"] == postcode:
27                 results.add(prop)
28         return results
```

Here the employee functions can be found. Above illustrates the possibility to search the sales agency by either price range or by postcode.

Below provides the code to adding a property to the sales agency warehouse. The employee must fill all entry fields in order for the property to be added successfully.

Further a property can be deleted from the system given its unique ID number.

```
#add a property - require details on owner, number, postcode, price
#If all details have been entered then property can be added
    def add(self, sales_agency):
        print("Properties: ", sales_agency.properties)
        print("Please give me details of a property to add.")
        owner = input("Who is the property owner:").strip()
        number = input("What is the property number:").strip()
        postcode = input("What is the property postcode:").strip()
        price = input("What is the property price:").strip()
        if owner and number and postcode and price:
            sales_agency.add(owner, number, postcode, price)
        print("Properties: ", sales_agency.properties)
#delete a property from the system given its unique ID
    def delete(self, sales_agency):
        print("Properties: ", sales_agency.properties)
        unique_id = input("Delete Property: '").strip()
        if unique_id:
            sales_agency.delete(unique_id)
        print("Properties: ", sales_agency.properties)
```

Employee_client.py

```python
import sys
import Pyro4
import Pyro4.util
from employee import Employee

#remote exception
sys.excepthook = Pyro4.util.excepthook
#retriving proxy for name server from within the network
#creates Proxy
#connects to pyro object
def find_sales_agency():
    sales_agency = None
    with Pyro4.locateNS() as ns:
        uri = ns.lookup("sales_agency")
        sales_agency = Pyro4.Proxy(uri)
    if not sales_agency:
        raise ValueError("no sales agency found!")
    return sales_agency


sales_agency = find_sales_agency()
#employee's actions
mark = Employee("Mark")
mark.add(sales_agency)
mark.remove(sales_agency)

#Employee- search by price range example
results = mark.search_by_price_range(1000, 251000)
for result in results:
    print result.owner
```

“sys.excepthook = Pyro4.util.excepthook” is a remote exception, allowing for clarity when/if an error occurs.

The code within ‘def find_sales_agency’ allows for the proxy to be created and retrieved from the name server within the sales agency network. This also allows for the pyro remote object to be connected.

In this case, the employee (Mark) searches properties by price range highlighting a minimum of 1000 and a maximum 251000. The results will be printed.

Customer_client.py

Suds had been implemented in order to link up to the soap sever in order for the customer / client to search the sales agency without being able to edit from a remote setting.

```
1  from __future__ import print_function
2  from suds.client import Client
3
4  #use of suds to test spyne soap server
5  if __name__ == '__main__':
6      url = "http://127.0.0.1:8000?WSDL"
7      client = Client(url)
8      print(client)
9
```

Soap_server.py

The soap server enables the pyroname to be located. It can remote call and define the parameters in order to create a web service. Once the soap parameters are set the protocol is then requested. WSGI is then coded in order to connect the application to the server.

```
1  from __future__ import print_function
2  from spyne.protocol.soap import Soap11
3  from spyne import Application, srpc, ServiceBase, Iterable, Unicode
4  from spyne.server.wsgi import WsgiApplication
5
6  #defining class attribute - sales agency
7
8  class SalesAgencyService(ServiceBase):
9      def find_sales_agency(self):
10         sales_agency = None
11         with Pyro4.locateNS() as ns:
12             uri = ns.lookup("sales_agency")
13             sales_agency = Pyro4.Proxy(uri)
14         if not sales_agency:
15             raise ValueError("no sales agency found!")
16         return sales_agency
17 # remote call and defines type/order of soap parameters
18     @srpc(_returns=Iterable(Unicode))
19     def get_properties():
20         sales_agency = self.find_sales_agency()
21         return sales_agency.properties
22 #request protocol
23 application  = Application([SalesAgencyService],
24     tns='salesAgency.http',
25     in_protocol=Soap11(validator='lxml'),
26     out_protocol=Soap11()
27 )
28 #wrap spyne app with Wsgi wrapper
29 wsgi_application = WsgiApplication(application)
30 #register WSGI app as handler to server
31 if __name__ == '__main__':
32     from wsgiref.simple_server import make_server
33     server = make_server('127.0.0.1', 8000, wsgi_application)
34     server.serve_forever()
35
```

Finally, to run the entire code after a name server has been created, the code can run through server.py. Below illustrates an outcome of this code, given the parameters under employee_client.py (to search a property within the price range 1000 to 251000