School of Computer Science

COMP47360

Tutorial
Setting up the Environment

Students:	NA
Supervisor(s):	Eimear Galligan, Conor Lawlor, Muireann MacCarthy
Date:	June 6, 2019
Total Number of Pages:	10

Executive Summary

This document aims at helping you to set up a minimal environment for this summer practicum. As you have already been told, the application should follow a three-tier model, i.e., you will have three elements that will interact in your application: data management, presentation/view and logic/"business intelligence". This tutorial will not give you direct answers to the challenges you will be facing and the different decisions you will have to make. It will just show you a panel of technologies you could use.

There is a good chance that you are already familiar with some of the concepts of this tutorial - so please skip these and focus on what you want to learn/what's new to you.

In this tutorial you will go through the following:

- set up a (virtual) environment (3 different solutions: VMs, IaaS or Paas)
- launch a data management system and a web application server
- define a schema for the data management system, run a few queries and ingest new data
- create a simple data processing algorithm
- create a simple user interface to display some of the information you collected during the previous steps

This tutorial also introduces unit testing.

1 Setting up the "Infrastructure"

There are 3 main options you can follow here: your own VM (on the School server), a IaaS solution or a PaaS solution.

1.1 Connecting to a Linux VM

You have received credentials (username and password) for an account on the School's server: csserver.ucd.ie (this is the hostname of the server). In this section we will see how to connect to this Linux account.

Mac OSX This is very straightforward, we simply open up the **Terminal** application and type:

```
$ ssh login@hostName
```

That's it!

Windows For windows, using ssh is slightly trickier, we need to download a piece of software that will allow us to connect. For a tutorial with visuals take a look at this: `https://mediatemple.net/community/products/dv/204404604/using-ssh-in-putty-`

- Download Putty from: `http://www.putty.org/`
- When you have downloaded it, find the putty.exe (the executable file) and double-click.
- Once it is installed, run the application.
- Under 'Host Name' enter the host name that was given to you.
- For 'Port', insert '22'
- The connection type will be SSH
- Hit open. (Note: You can actually save configurations with Putty which may be useful for you going forward.)
- Enter the password for the VM when prompted and your connected!

Alternatively if you have Git Bash installed:
Login to the remote server with the following command through the Git Bash Terminal:

```
$ ssh login@hostName
```

1.2 IaaS

Infrastructure as a Service (IaaS) is one of the ways you can consume the Cloud: cloud vendors let you use some VMs in their data centres. If you've never tried it, set up a VM with Amazon EC2 or Microsoft Azure (or any other vendor really).

- Amazon EC2: `https://aws.amazon.com/ec2`
- Azure: `https://azure.microsoft.com/en-us/free/`

They both have web interfaces and command line interfaces (CLI) to deploy, start, etc. VMs – don't hesitate to try both.

1.3 PaaS

Platform as a Service (Paas) is another way to consume cloud services: the Cloud is seen as a framework (lots of libraries, APIs, applications that you can use) on which you can deploy your code and your data.

Create an account on Heroku and go through the following tutorial: `https://devcenter.heroku.com/articles/getting-started-with-python#introduction`

2 Setting up the Data Management System

MySQL is a relational database management system. I believe you are familiar with this concept. If not, again, have a look online.

2.1 In your own VM

To install a MySQL service/server on your VM, run the following command:
`sudo apt-get install mysql-server`. (note: there might already be one running - just try to connect to the server to see whether something is already present).

To check that your set up worked, try the following command:
```
mysql -u root -p
```
and give the password you set up during the install.

2.2 In a platform

PaaS platforms generally come with data management systems already set up. PostgreSQL is the classical solution for Heroku: `https://devcenter.heroku.com/categories/reference#data-management`.

Exercise Given the conceptual model below (Figure 1), create a relational database (containing multiple tables). Remember that each entity has to become a relation (=table), with the identifier(s) as primary key, each relationship 1:N sets a key to the N relation, each relationship N:M creates a new relation with the keys from both sides. The 1:1 relationship is a little more tricky to manipulate. Try to remember or figure out what to do with it.

3 Setting up the App and Web Servers

I'll assume you'll be using Python. There are many options for the web/app servers in Python, the most popular being Django and Flask. Check the following pages for a good description of the context (web/app servers and frameworks for Python): `http://python-guide-pt-br.readthedocs.io/pt_BR/latest/scenarios/web.html` `https://docs.python-guide.org/scenarios/web/`

Install or use some of the frameworks in your own (local) VM, your IaaS or PaaS solutions.

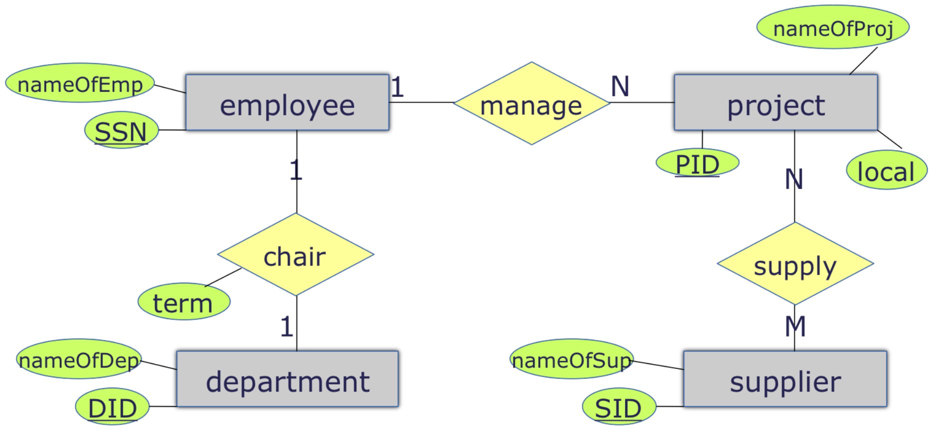


Figure 1: Entity/Relationship conceptual model

4 Django Tutorial

Django is an MVC (Model View Controller) web framework. Models represent the data in the database, views control how the site will be displayed to the user and controllers manage the interactions between models and views, and contain any logic needed for the site. The advantage of this is that it allows the different components of the website to be modular and separate. While there may be an initial learning curve to getting used to Django - it has the benefits of providing many out-of-the-box features such as authentication and object relational mapping which is very useful when dealing with data retrieved from your database. Additionally, it works well with Heroku if you choose to use that method for hosting.

The following tutorial goes through how to set up a simple blog website using Django: `https://tutorial.djangogirls.org/en/`
There will be some sections in the above tutorial which you can certainly skip as you will know it already. When you get to the deployment stage of the tutorial - also try getting your application working with Heroku as you did with the sample application in the Heroku tutorial.

Exercise Write a simple web application that queries the database you created before and displays something on a webpage.

5 Unit Testing in Python

Software Testing is one of the main methods to ensure software quality[1]. The aim of Software Testing is to investigate whether software artefacts meet the (funtional and non-functional) requirements that they are supposed to meet.

One particular way of testing software artefacts is through **unit testing**: a method to test individual components of a software program. In short, unit tests consider one single element of the programme (typically, a method or an instruction or a class) and aims at testing thoroughly its behaviour, with an emphasis on the acceptable behaviours.

[1]For more details, see [1] and the online version here: `http://se.ethz.ch/~meyer/publications/testing/principles.pdf`.

Take the example of the Triangle class below:

```python
class Triangle:

    def __init__(self, a=1, b=1, c=1):
        self.__a = a
        self.__b = b
        self.__c = c

    def classify(self):

        if ((self.__a <= 0) or (self.__b <= 0) or (self.__c <= 0)):
            return "INVALID"
        trian = 0
        if self.__a == self.__b:
            trian += 1
        if self.__a == self.__c:
            trian += 2
        if self.__b == self.__c:
            trian += 3

        if trian == 0:
            if(((self.__a + self.__b) < self.__c) or ((self.__a + self.__c) < self.__b) or ((self.__b + self.__c) < self.__a)):
                return "INVALID"
            else:
                return "SCALENE"

        if trian > 3:
            return "EQUILATERAL"

        if ((trian == 1) and ((self.__a + self.__b) > self.__c)):
            return "ISOCELES"
        else:
            if ((trian == 2) and ((self.__a + self.__c) > self.__b)):
                return "ISOCELES"
            else:
                if ((trian == 3) and ((self.__b + self.__c) > self.__a)):
                    return "ISOCELES"
        return "INVALID"
```

This class defines a Triangle given the length of its three sides (a, b and c). Depending on the values of a, b andc, a triangle can be classified as scalene, equilateral, isosceles – or invalid if given a, b and c the object cannot be a triangle.

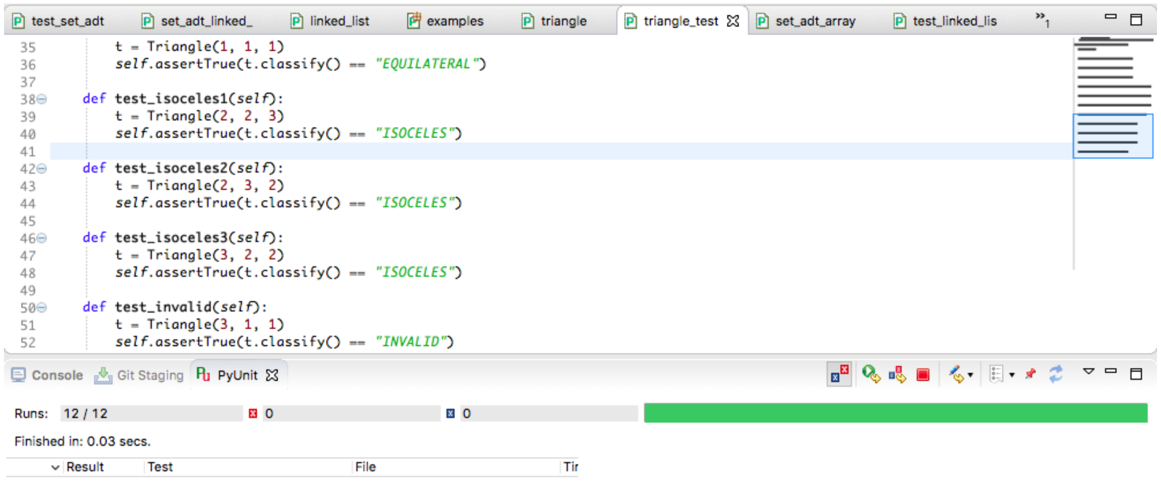
To make sure this implementation is correct (really does what it's suppose to do), we will use unit tests. See below an example of a class with tests for individual functions of the Triangle class:

```python
import unittest
from examples.triangle import Triangle

class TestTriangle(unittest.TestCase):

    def test1(self):
        t = Triangle(1, 2, 3)
        self.assertTrue(t.classify() == "SCALENE")

    def test_invalid1(self):
        t = Triangle(1, 2, 4)
        self.assertTrue(t.classify() == "INVALID")

    def test_invalid2(self):
        t = Triangle(1, 4, 2)
        self.assertTrue(t.classify() == "INVALID")

    def test_invalid3(self):
        t = Triangle(4, 1, 2)
        self.assertTrue(t.classify() == "INVALID")

    def test_invalid_neg1(self):
        t = Triangle(-1, 1, 1)
        self.assertTrue(t.classify() == "INVALID")

    def test_invalid_neg2(self):
        t = Triangle(1, -1, 1)
        self.assertTrue(t.classify() == "INVALID")

    def test_invalid_neg3(self):
        t = Triangle(1, 1, -1)
        self.assertTrue(t.classify() == "INVALID")

    def test_equilateral(self):
        t = Triangle(1, 1, 1)
        self.assertTrue(t.classify() == "EQUILATERAL")

    def test_isoceles1(self):
        t = Triangle(2, 2, 3)
        self.assertTrue(t.classify() == "ISOCELES")

    def test_isoceles2(self):
        t = Triangle(2, 3, 2)
        self.assertTrue(t.classify() == "ISOCELES")

    def test_isoceles3(self):
        t = Triangle(3, 2, 2)
        self.assertTrue(t.classify() == "ISOCELES")

    def test_invalid(self):
        t = Triangle(3, 1, 1)
        self.assertTrue(t.classify() == "INVALID")
```

The important thing to notice (from a "technical" point of view) is the assert instructions (e.g., `assertTrue()` here). There are many more possible assert functions in unittest: see `https://docs.python.org/3.3/library/unittest.html` In general, you don't need to have only one assert per function in a testing class – and often testing functions test one particular perspective/function of the original source code.

You will find in Appendix A a test class for the Set ADT (implemented using arrays or linked lists). You will see that the test methods (unit tests) can be more complicated and in particular contain more than one assert.

Eclipse is equipped with a unit testing view that can be useful (see Figure 2). To access the view go to `Window > Show View > PyUnit`. Now when you run the test class, you can decide to run it as Python unit-test.



```
        t = Triangle(1, 1, 1)
        self.assertTrue(t.classify() == "EQUILATERAL")

    def test_isoceles1(self):
        t = Triangle(2, 2, 3)
        self.assertTrue(t.classify() == "ISOCELES")

    def test_isoceles2(self):
        t = Triangle(2, 3, 2)
        self.assertTrue(t.classify() == "ISOCELES")

    def test_isoceles3(self):
        t = Triangle(3, 2, 2)
        self.assertTrue(t.classify() == "ISOCELES")

    def test_invalid(self):
        t = Triangle(3, 1, 1)
        self.assertTrue(t.classify() == "INVALID")
```

```
Runs:  12 / 12     0     0
Finished in: 0.03 secs.
Result   Test   File   Tir
```

Figure 2: The testing view in Eclipse

Exercise
In this exercise, you are asked to implement in Python the array-based set ADT as you probably designed it last semester. See Appendix B for a skeleton of the code you should come up with.

References

[1] Meyer, B. Seven principles of software testing. *Computer 41*, 8 (2008).

Appendix A: Test Class for Set ADT

```
import unittest
from data_structures.set_adt_array import Set

class TestSetAdt(unittest.TestCase):

    def test_add(self):
        a = Set()
        a.add(24)
        a.add(24)
        self.assertTrue(a.size()==1)

    def test_remove(self):
        a = Set()
        a.add(24)
        a.add("toto")
        self.assertTrue(a.size()==2)
        a.remove("titi")
        self.assertTrue(a.size()==2)
        a.remove("toto")
        self.assertTrue(a.size()==1)

    def test_empty(self):
        a = Set()
        self.assertTrue(a.is_empty())
        a.add(24)
        self.assertFalse(a.is_empty())
        a.remove(24)
        self.assertTrue(a.is_empty())
        a.add(24)
        a.remove(23)
        self.assertFalse(a.is_empty())

    def test_union(self):
        a = Set()
        b = Set()
        a.add(24)
        a.add(25)
        b.add(4)
        b.add(6)
        b.add(24)
        a.union(b)
        self.assertEqual(a.size(), 4)

    def test_intereection(self):
        a = Set()
        b = Set()
        a.add(25)
        b.add(4)
        b.add(6)
        b.add(24)
        a.intersection(b)
        self.assertEqual(a.size(), 0)

        c = Set()
        c.add(4)
        c.add(6)
        c.intersection(b)
        self.assertEqual(c.size(), 2)

    def test_difference(self):
        a = Set()
        b = Set()
        a.add(24)
        a.add(25)
        b.add(4)
        b.add(6)
        b.add(24)
        a.difference(b)
        self.assertEqual(a.size(), 1)
        b.difference(a)
        self.assertEqual(b.size(), 3)


if __name__ == '__main__':
    unittest.main()
```

Appendix B: Skeleton of Array-based Implementation of Set ADT

```python
import sys


class Set:
    __my_array = []
    __size_my_array = 0

    def __init__(self):
        self.__my_array = []
        self.__size_my_array = 0

    def size(self):
        return ?

    def is_empty(self):
        return ?

    def contains(self, elem):
        ?
        return ?

    def add(self, elem):
        ?

    def remove(self, elem):
        ?

    def union(self, b):
        ?

    def intersection(self, b):
        ?

    def difference(self, b):
       ?

    def to_string(self):
        ?
```