come up with hypothesis questions in terms of:

“Autonomy - each system is free and independent with its own purpose of operation;

Belonging - systems function collaboratively to meet a common higher purpose;

Connectivity - synergism is enabled by the highly dynamic distributed network;

Diversity - the constituents are heterogeneous, self-sufficient systems that are open for enhancement by evolution and adaptation;

Emerging - the cumulative actions and interactions between the constituents of an SoS give rise to the behaviours that can be attributed to the SoS as a whole.”

**Assignment Brief: Part 1 (group submission)**

**600 words** is for a required Discussion in **your README file only**.

You **should each INDIVIDUALLY create a question or hypothesis about one of the ABCDE characteristics** (as a function of security) of the system you modelled in part 1. *For example, your question may be “is it more efficient and secure to encrypt individual messages rather than encrypting the entire message stack/ protocol?”*

**Once you have your question, discuss it with your team members and as a group decide on which is the most interesting question.**

Your team should then **design and build a prototype system to investigate the selected problem, i.e.,** –

* create a number of experiments to test the hypothesis,
* analyse your results and produce a summary paper that discusses the question, model, experiments, analysis and the conclusion.

*For example, with the question above the conclusion should include an answer to whether it is more efficient/ secure to encrypt individual messages or the protocol and how you came to this conclusion.*

As a minimum, you **should create a prototype that demonstrates the operation of, and interaction between, a simulated device and a simulated controller**.

**Both should be modelled in Python** and **should demonstrate code and techniques that address both the challenges of distributed computing, mitigate (some of) the vulnerabilities identified in your AD trees produced** in part 1 of the assessment, as well as **answer the question posed by your hypothesis**.

**Checklist for this section of the assignment:**

1. Individually create a question/ hypothesis that explores one of the ABCDE characteristics/ security of your selected system – you will use this question in your reflection as well.
2. Agree as a team on the most appropriate question/ hypothesis from all those created by team members.
3. Design a model specification that will be used to test the hypothesis.
4. Produce a prototype in Python that models/ simulates a client device and controller/ hub (as a minimum) that can be used to investigate your chosen hypothesis.
5. Demonstrate interactions between the two simulated systems, illustrating how your model investigates your question and addresses some of the common problems of distributed systems (such as latency, power consumption, lost messages, reliability and security).
6. Provide evidence of the above (via demos and/or test outputs).
7. Carry out comprehensive testing (of style, code security and functionality/ operation) to prove the model meets its specification.
8. Carry out experiments that evaluate your question and produce results for analysis.
9. Produce comprehensive documentation including code comments/ markup and a supplementary readme file that must discuss your chosen question, the data and analysis and conclusions/ answers. (Hint: could use of the e-portfolio make this easier?)

**File Submission**

1. **The source code and applicable documentation** (comments explaining the code, as well as a README file containing a description of the question, the model implemented, data analysis, conclusions/ answers AND instructions on how to execute the code).
2. **Output from demonstrations, testing tools and suites**.
3. **Discussions of vulnerabilities identified, mitigations deployed and discussion of any omissions/ lack of mitigations**. **(600 words maximum).**
4. **Practical Aspect**
5. **Regular demonstrations of code and executables**, as well as **testing, and demonstrating development over the lifetime of the module** (these may happen in the seminar sessions or in one-to-one sessions with your tutor).
6. Note that the associated grading criteria are highlighted in the requirements below, to be reviewed alongside the grading criteria (Module Resources).

**Python code, covering:**

1. **Code Quality –** is the appropriate approach (techniques, algorithms etc) used? and
2. **Simulation** (Knowledge and Understanding weighted at 30%)
3. **Application of object-oriented programming features** and **Mitigating Security Vulnerability, including testing**. (Application of Knowledge and understanding weighted a 30%)
4. **Structure and Presentation** (weighted at 30%) focuses on:
   1. **Comments on the code** explaining it,
   2. **your README file** containing a description of the implementation and instructions on how to execute the code, as well as the discussion requested above, and
   3. **how well you have organised your code**
5. Please note that **academic integrity applies to the demonstrations** as mentioned above, and **your codes where all sources can be placed in the README file and/or as part of the code commentary.** (Academic Integrity weighted at 10%)
6. **Remember! The Python files should be provided, along with a README explaining their use**. Good programming principles should be applied throughout the code. Also log your submissions in your e-portfolio for future reference.

**My Submission**

Nmkjk

Here's an example of how you could create a simulation in Python to demonstrate the operation of and interaction between a device and a controller for a light control system, as described in the article by Kodali et al. (2016).

python

Copy code

class Light:

def \_\_init\_\_(self):

self.is\_on = False

def turn\_on(self):

print("Light turned on")

self.is\_on = True

def turn\_off(self):

print("Light turned off")

self.is\_on = False

class Controller:

def \_\_init\_\_(self, light):

self.light = light

def receive\_sensor\_input(self, is\_motion\_detected):

if is\_motion\_detected:

self.light.turn\_on()

else:

self.light.turn\_off()

light = Light()

controller = Controller(light)

controller.receive\_sensor\_input(True)

controller.receive\_sensor\_input(False)

controller.receive\_sensor\_input(True)

This code simulates a light control system with a light that can be turned on or off, and a controller that can receive sensor inputs and control the light. The simulation prints the state of the light (on or off) whenever it is changed by the controller, allowing you to visualize the interaction between the device (light) and the controller.

This is a simple example, and you could expand on it to create a more complex and realistic simulation, depending on the requirements of your system.

Here's an example of how you could code and implement a light control system in Python, as described in the article by Kodali et al. (2016).

python

Copy code

class Light:

def \_\_init\_\_(self):

self.is\_on = False

def turn\_on(self):

print("Light turned on")

self.is\_on = True

def turn\_off(self):

print("Light turned off")

self.is\_on = False

class Controller:

def \_\_init\_\_(self, light):

self.light = light

def receive\_sensor\_input(self, is\_motion\_detected):

if is\_motion\_detected:

self.light.turn\_on()

else:

self.light.turn\_off()

light = Light()

controller = Controller(light)

controller.receive\_sensor\_input(True)

controller.receive\_sensor\_input(False)

controller.receive\_sensor\_input(True)

This code defines two classes: Light and Controller. The Light class represents the light device and has methods for turning the light on or off. The Controller class represents the controller and has a method for receiving sensor inputs and controlling the light accordingly.

To implement this system, you would create instances of the Light and Controller classes, associate the light instance with the controller instance, and then call the receive\_sensor\_input method of the controller instance, passing in the sensor input (i.e., whether motion was detected or not). The simulation prints the state of the light (on or off) whenever it is changed by the controller, allowing you to visualize the interaction between the device (light) and the controller.

To run the code for the light control system in Python, you need to have Python installed on your computer. You can then open a terminal or command prompt, navigate to the directory where the code is saved, and run the following command:

Copy code

python filename.py

where filename.py is the name of the file that contains the code. When you run the code, you should see output similar to the following:

vbnetCopy code

Light turned on Light turned off Light turned on

This output shows that the light was turned on, then turned off, and then turned on again based on the sensor inputs received by the controller.

Note: This is just an example simulation and does not actually control a real light. To control a real light, you would need to use physical components such as sensors and actuators, and interface them with the controller, which could be a microcontroller or a computer running the Python code.

Top of Form

To integrate the light control system into a website, you can use a web framework such as Flask or Django to build a web interface for the system. Here's an example of how you could implement the same light control system in Flask:

python

Copy code

from flask import Flask, request

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

class Light:

def \_\_init\_\_(self):

self.is\_on = False

def turn\_on(self):

self.is\_on = True

def turn\_off(self):

self.is\_on = False

class Controller:

def \_\_init\_\_(self, light):

self.light = light

def receive\_sensor\_input(self, is\_motion\_detected):

if is\_motion\_detected:

self.light.turn\_on()

else:

self.light.turn\_off()

light = Light()

controller = Controller(light)

@app.route("/")

def index():

return "Light is currently {}".format("ON" if light.is\_on else "OFF")

@app.route("/receive\_sensor\_input", methods=["POST"])

def receive\_sensor\_input():

is\_motion\_detected = request.form.get("is\_motion\_detected") == "True"

controller.receive\_sensor\_input(is\_motion\_detected)

return "Sensor input received: motion\_detected={}".format(is\_motion\_detected)

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

This code uses the Flask web framework to create a web server that serves the light control system. The index function returns a string that represents the current state of the light (on or off), and the receive\_sensor\_input function receives sensor inputs via HTTP POST requests and updates the state of the light accordingly.

To run this code, you can save it to a file with the extension .py and run the following command in a terminal or command prompt:

javascript

Copy code

export FLASK\_APP=filename.py

flask run

where filename.py is the name of the file that contains the code. When you run the code, Flask will start a web server on your local machine and you can access the light control system in a web browser by visiting <http://localhost:5000/>.

To integrate the light control system into a Django web application, you can use the Django framework to build a web interface for the system. Here's an example of how you could implement the same light control system in Django:

Start by creating a new Django project and a new app within the project:

bash

Copy code

django-admin startproject myproject

cd myproject

python manage.py startapp myapp

In myapp/models.py, create the following classes to represent the light and controller:

python

Copy code

from django.db import models

class Light(models.Model):

is\_on = models.BooleanField(default=False)

def turn\_on(self):

self.is\_on = True

self.save()

def turn\_off(self):

self.is\_on = False

self.save()

class Controller(models.Model):

light = models.OneToOneField(Light, on\_delete=models.CASCADE)

def receive\_sensor\_input(self, is\_motion\_detected):

if is\_motion\_detected:

self.light.turn\_on()

else:

self.light.turn\_off()

In myapp/views.py, create the following view to show the state of the light:

python

Copy code

from django.shortcuts import render

from .models import Light

def index(request):

light = Light.objects.first()

return render(request, "index.html", {"light": light})

In myapp/urls.py, create the following URL pattern to map to the index view:

python

Copy code

from django.urls import path

from . import views

urlpatterns = [

path("", views.index, name="index"),

]

Create a template file myapp/templates/index.html with the following content to display the state of the light:

html

Copy code

<h1>Light is currently {{ light.is\_on|yesno:"ON,OFF" }}</h1>

In myproject/settings.py, add myapp to the INSTALLED\_APPS list:

python

Copy code

INSTALLED\_APPS = [

# ...

'myapp',

]

In myproject/urls.py, include the myapp.urls module:

python

Copy code

from django.urls import include, path

urlpatterns = [

path("", include("myapp.urls")),

]

Run the following commands to create the database tables and start the Django development server:

Copy code

python manage.py makemigrations

python manage.py migrate

python manage.py runserver

When you run the code, Django will start a web server on your local machine and you can access the light control system in a web browser by visiting http://localhost:8000/.

Note: This is just an example and does not represent a complete or secure implementation of a light control system for a web application. You would need to consider issues such as security, scalability, and