**Python Software Engineer Questions**

Please put your responses in line below in red.

#1

Pretend that an API exists at <https://example.com/api/products> which returns:

[{"product": "Shoes", "price": 35, "rating": 4.2},  
{"product": "White Hat", "price": 21, "rating": 4.8},  
...]

* Write a function in Python to consume this API.

import requests

response = requests.get("<https://example.com/api/products>")

response.text

* Write another function for business logic to find all products rated 4 and above.

from bs4 import BeautifulSoup

import requests

response = requests.get("https://example.com/api/products")

data = response.text

soup = BeautifulSoup(data)

for link in soup.find\_all('rating' > 4):

print (link.get('product'))

* Write enough unit / integration tests to completely cover the code in both functions.

import unittest

from pyramid import testing

class MyTest(unittest.TestCase):

def setUp(self):

self.config = testing.setUp()

def tearDown(self):

testing.tearDown()

def test\_view\_fn\_forbidden(self):

from pyramid.httpexceptions import HTTPForbidden

from my.package import view\_fn

self.config.testing\_securitypolicy(userid='hank',

permissive=False)

request = testing.DummyRequest()

request.context = testing.DummyResource()

self.assertRaises(HTTPForbidden, view\_fn, request)

def test\_view\_fn\_allowed(self):

from my.package import view\_fn

self.config.testing\_securitypolicy(userid='hank',

permissive=True)

request = testing.DummyRequest()

request.context = testing.DummyResource()

response = view\_fn(request)

self.assertEqual(response, {'greeting':'hello'})

from pyramid.view import view\_config

@view\_config(route\_name='home', renderer='myproject:templates/mytemplate.jinja2')

def my\_view(request):

return {'project': 'myproject'}

@pytest.fixture

def testapp(app):

testapp = webtest.TestApp(app, extra\_environ={

'HTTP\_HOST': 'example.com',

})

return testapp

def test\_root(testapp):

res = testapp.get('/', status=200)

assert b'Pyramid' in res.body

def test\_notfound(testapp):

res = testapp.get('/badurl', status=404)

assert res.status\_code == 404

#2

Think of a recent time you worked on a performance problem or bug in production.

* Write a paragraph about it, in technical detail of how you figured out what the problem was and resolved it.

A circular module dependencies.

This is so trivial, but also frustratingly common: in one file, let’s say *bug.py*, you write *import anotherbug*. In another file, anotherbug.py, you write import bug.

This isn’t going to work — how should the computer know which file to include in which? You probably knew this but did it by accident. The solution is to rephrase the module names.

* Write another paragraph about it, as if you were explaining to your manager in less-technical but still meaningful terminology.

The problem was given because in the system there were several components with the same name and when we called these redundant components, the system did not know which of them we were referring to.

#3

Write a small python webapp using flask, bottle, or similar.  It should have an API endpoint to receive movie titles, descriptions, price and release date.  Save those records in the datastore of your choice.  Then add an RSS feed and an Atom feed to the webapp using this library: <https://pypi.org/project/feedgenerator/>  You will have to do some exploration to figure out how to use the library.  The data for the two feeds should be the movie records that were saved.

# 3 Escribe una pequeña aplicación web de Python usando un matraz, botella o similar.

Debe tener un punto final de API para recibir títulos de películas, descripciones, precio y fecha de lanzamiento.

Guarde esos registros en el almacén de datos de su elección.

Luego agregue una fuente RSS y una fuente Atom a la aplicación web usando esta biblioteca: <https://pypi.org/project/feedgenerator/>

Tendrá que explorar un poco para descubrir cómo usar la biblioteca.

Los datos de las dos fuentes deben ser los registros de películas que se guardaron.

import datetime

from feedgen.feed import FeedGenerator

import os

from flask import Flask, Response, request

from flask\_mongoengine import MongoEngine

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

app.config['MONGODB\_SETTINGS'] = {

'host': os.environ['MONGODB\_HOST'],

'username': os.environ['MONGODB\_USERNAME'],

'password': os.environ['MONGODB\_PASSWORD'],

'db': 'webapp'

}

db = MongoEngine()

db.init\_app(app)

class Movie(db.Document):

title = db.StringField(max\_length=60)

description = db.StringField()

price = db.FloatField(default=0)

pub\_date = db.DateTimeField(default=datetime.datetime.now)

@app.route("/api")

def index(title,description,price,pub\_date):

Movie.objects().delete()

Movie(title,description,price,pub\_date).save()

todos = Movie.objects().to\_json()

return Response(todos, mimetype="application/json", status=200)

@app.route("/feed\_rss")

def feed\_rss():

fg = FeedGenerator()

fg.title('https://example.com/api/')

fg.description('Some Testfeed')

fg.price,( {'price':'20.0'} )

fg.pub\_date('01-20-10')

atomfeed = fg.atom\_str(pretty=True) # Get the ATOM feed as string

rssfeed = fg.rss\_str(pretty=True) # Get the RSS feed as string

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

app.run(debug=True, port=5000)