1. Doggie Day Care

I’ve given you the TestDog class code

Do This: Write a Dog class that will pass the tests contained in our test class. Do NOT change

anything in our test class!

Your Dog class must provide the following methods to adequately pass our tests:

\* \_\_init\_\_ (takes a string name of dog, int age and Boolean indicating a purebred or not),

\* is\_purebred (returns True or False depending on if the dog is a purebred)

\* clone (makes a non-mutated copy of the current object)

\* \_\_str\_\_ returns a string representation of the object

\* \_\_eq\_\_ returns True or False when comparing two instances of the Dog class

import unittest

from Dog import Dog

class TestDog(unittest.TestCase):

''' class to test all the methods of the Dog class

imports from TestCase

methods:

\* \_\_init\_\_,

\* is\_purebred,

\* clone (makes a non-mutated copy of the current object)

\* \_\_str\_\_, represent the Dog as a string

\* \_\_eq\_\_, compare current object with another Dog. Are they equal?

'''

def test\_init(self):

d = Dog("Fido", 10, True)

self.assertEqual(d.name, "Fido")

self.assertEqual(d.age, 10)

self.assertTrue(d.is\_purebred)

d2 = Dog("Fifi", 11, False)

self.assertEqual(d2.name, "Fifi")

self.assertEqual(d2.age, 11)

self.assertFalse(d2.is\_purebred())

def test\_bad\_init(self):

with self.assertRaises(ValueError):

d = Dog("Fido", -10, True)

def test\_is\_purebred(self):

d = Dog("Fido", 10, True)

d2 = Dog("Fifi", 11, False)

self.assertTrue(d.is\_purebred())

self.assertFalse(d2.is\_purebred())

def test\_eq(self):

d = Dog("Fido", 10, True)

d2 = Dog("Fifi", 11, False)

d3 = Dog("Fifi", 11, False)

self.assertTrue(d2.\_\_eq\_\_(d3))

self.assertFalse(d.\_\_eq\_\_(d2))

self.assertTrue(d2 == d3)

self.assertFalse(d == d2)

def test\_clone(self):

d2 = Dog("Fifi", 11, False)

d3 = d2.clone()

self.assertIsNot(d2,d3) # should NOT be same object!

self.assertEqual(d2, d3) # but SHOULD be equal!

def test\_str(self):

d = Dog("Fido", 10, True)

d2 = Dog("Fifi", 11, False)

msg = "Purebred: Fido 10 years"

self.assertEqual(d.\_\_str\_\_(), msg)

msg = "NOT Purebred: Fifi 11 years"

self.assertEqual(d2.\_\_str\_\_(), msg)

def main():

unittest.main(verbosity = 3)

main()

2. Equal Stacks

You may not modify the Stack class.

Write a function named equal\_stacks that accepts two stacks of Dog objects as parameters and

determines if the two stacks are equal to each other. Two stacks are equal if (1) they contain the same

number of objects, and (2) the corresponding objects in each stack are identical, according to your

\_\_eq\_\_ method. Your function should return a Boolean.

Note: If you were unable to get your Dog class working in problem 8, you may assume we’ll use a

Stack of integers for this question.

For example, these stacks are equal...

s1 = Stack()

s1.push(Dog(“Fido”, 10, True))

s1.push(Dog(“Fifi”, 11, False))

s2 = Stack()

s1.push(Dog(“Fido”, 10, True))

s1.push(Dog(“Fifi”, 11, False))

... these stacks are not:

s3 = Stack()

s1.push(Dog(“Fido”, 10, True))

s1.push(Dog(“Fifi”, 11, False))

s4 = Stack()

s1.push(Dog(“Fido”, 10, True))

s1.push(Dog(“Fifi”, 5, True))

... and these stacks are not:

s3 = Stack()

s1.push(Dog(“Fido”, 10, True))

s4 = Stack()

s1.push(Dog(“Fido”, 10, True))

s1.push(Dog(“Fifi”, 11, False))

3. Predicate & Filter

(a) Write a predicate function (named predicate) that takes as input a single integer. It uses the

constant MAX\_VALUE given below as the basis of its test. If the integer is less than or equal to

MAX\_VALUE, your predicate function should return True. Otherwise return False.

● Name: predicate

● Parameters: an integer

● Returns: True or False, depending on if the parameter is less-than-or-equal-to

MAX\_VALUE, or not

MAX\_VALUE = 10

(b) Write a filter function called filter\_at. Your function must take a list of integers as its only

parameter and return a new list that contains the integers for which your predicate() written in

part (a) returned True. Your filter\_at() function MUST use your predicate() function for

its implementation.

Example (given MAX\_VALUE == 10 as before):

x = [1, 2, 30, 4, 55, 6, 10, 20, 5, 100]

print(filter\_at(x))

Gives this output:

[1, 2, 4, 6, 10, 5]