**Virtual Initio Programming: WS27 and Ex27 Sample Answers**

**WS27**

**Question 1:** It prints an empty list.

**Question 2:** It created an empy list/array called readings.

**Question 3:** It prints an empty list (same as when data1.print\_readings() was called).

**Question 4:** take\_readings()

**Question 5:** print\_readings()

**Question 6:** The answer will depend upon where boxes have been placed and how many readings have been taken, but it will be a list of floats.

**Question 7:** data2 = DataLogger()

**Question 8:** Each data logger only stores a reading when its own take\_readings() method is called. Therefore the two data loggers have different lists of readings because the objects were moved between taking readings with different loggers.

**Exercise 1:**

import simclient.simrobot as initio

class DataLogger:

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

self.right\_readings = []

self.left\_readings = []

def take\_irRight\_reading(self):

self.right\_readings.append(initio.irRight())

def take\_irLeft\_reading(self):

self.left\_readings.append(initio.irLeft())

def take\_reading(self):

self.right\_readings.append(initio.irRight())

self.left\_readings.append(initio.irLeft())

def print\_irRight\_log(self):

print(self.right\_readings)

def print\_irLeft\_log(self):

print(self.left\_readings)

def print\_readings(self):

print("Right Sensor:" + str(self.right\_readings))

print("Left Sensor:" + str(self.left\_readings))

**Exercise 2:**

import simclient.simrobot as initio

class DataLogger:

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

self.readings = []

self.logger\_name = name

def take\_reading(self):

self.readings.append(initio.getDistance())

def print\_readings(self):

print(self.readings)

def get\_name(self):

return self.logger\_name

def set\_name(self, name):

self.logger\_name = name

**Question 9:** The first logger prints its name as logger1 and the second as logger2. This is because they were given different names when initialised.

**Ex27**

**NOTE:** Exercises are reaching sufficient complexity that it is unreasonable to expect answers to be largely similar – particularly in the way functions, methods, fields and classes may be constructed. Sample answers have to be regarded as examples only, not definitive answers.

**Exercise 1:**

import simclient.simrobot as initio

import time

class WallFollower:

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

print("Initialising")

def drive\_to\_wall(self):

while (not initio.getDistance() < 20 and not initio.irLeft() and not initio.irRight()):

initio.forward(10)

initio.stop()

def spin(self, direction):

if (direction == 'right'):

initio.spinRight(10)

else:

initio.spinLeft(10)

def obstacle\_to(self, direction):

if (direction == 'right'):

return initio.irRight()

else:

return initio.irLeft()

def opposite\_direction(self, direction):

if (direction == 'right'):

return 'left'

else:

return 'right'

def follow\_wall(self, direction):

while (not initio.getSwitch() == True):

if (initio.getDistance() < 20):

while(initio.getDistance() < 20 or self.obstacle\_to(direction)):

self.spin(self.opposite\_direction(direction))

elif (self.obstacle\_to(direction)):

initio.forward(10)

time.sleep(2)

elif (not self.obstacle\_to(direction)):

self.spin(direction)

time.sleep(1)

initio.stop()

**Exercise 2:**

import simclient.simrobot as initio

import time

class WallFollower:

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

self.direction = side

def drive\_to\_wall(self):

while (not initio.getDistance() < 20 and not initio.irLeft() and not initio.irRight()):

initio.forward(10)

initio.stop()

def spin(self, direction):

if (direction == 'right'):

initio.spinRight(10)

else:

initio.spinLeft(10)

def obstacle\_to(self):

if (self.direction == 'right'):

return initio.irRight()

else:

return initio.irLeft()

def opposite\_direction(self):

if (self.direction == 'right'):

return 'left'

else:

return 'right'

def follow\_wall(self):

while (not initio.irLeftLine()):

if (initio.getDistance() < 20)

while(initio.getDistance() < 20 or self.obstacle\_to()):

self.spin(self.opposite\_direction())

elif (self.obstacle\_to()):

initio.forward(10)

time.sleep(2)

elif (not self.obstacle\_to()):

self.spin(self.direction)

time.sleep(1)

initio.stop()

initio.init()

right\_wall = WallFollower('right')

left\_wall = WallFollower('left')

right\_wall.follow\_wall()

initio.reverse(10)

time.sleep(5)

initio.spinRight(10)

time.sleep(5)

left\_wall.follow\_wall()

**Exercise 3:** Note we have a call to the ultrasonic sensor in follow\_policy to allow a user to stop the robot by placing something in front of it.

import simclient.simrobot as initio

import random, time

class MachineLearner():

def \_\_init\_\_(self, learning\_rate):

self.learning\_rate = learning\_rate

self.epsilon = 1

self.actions = ['forward', 'backward', 'left', 'right']

self.dictionary = self.action\_reward(self.actions, 1)

self.rewards = {(1, 1):1, (1, 0):2, (0, 1):1, (0, 0):0}

def action\_reward(self, action\_list, default):

action\_rewards = {}

for i in range(0, 2):

for j in range(0, 2):

for k in (action\_list):

action\_rewards[((i, j), k)] = default

return action\_rewards

def get\_max(self, state):

max\_reward = 0

action = 'forward'

for act in self.actions:

if (self.dictionary[(state, act)] > max\_reward):

action = act

max\_reward = self.dictionary[(state, act)]

return action

def action\_execute(self, action):

if (action == "forward"):

initio.forward(10)

elif (action == "backward"):

initio.reverse(10)

elif (action == "left"):

initio.spinLeft(10)

elif (action == "right"):

initio.spinRight(10)

else:

initio.stop()

def learn(self):

while (self.epsilon > 0):

explore = random.random();

state = (initio.irLeftLine(), initio.irRightLine())

if (explore < self.epsilon):

print("Exploring")

action = random.choice(self.actions)

else:

action = self.get\_max(state)

self.action\_execute(action)

time.sleep(3)

reward = self.rewards[(initio.irLeftLine(), initio.irRightLine())]

if (reward == 2):

self.epsilon = self.epsilon - 0.05

print("updating epsilon: " + str(self.epsilon))

self.dictionary[(state, action)] = self.dictionary[(state, action)] + ( reward - self.dictionary[(state, action)]) \* self.learning\_rate

def follow\_policy(self):

while (not initio.getDistance() < 20):

state = (initio.irLeftLine(), initio.irRightLine())

action = self.get\_max(state)

self.action\_execute(action)

time.sleep(3)

initio.stop();

initio.init()

learner = MachineLearner(0.5)

learner.learn()

learner.follow\_policy()



University of Liverpool, 2020

This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).