**Virtual Initio Programming: WS31 & Ex31 Sample Answers**

**NOTE:** Exercises have become sufficiently complex that quite wide variability in answers can be expected.

**WS31**

**Question 1:** `alice` is printed out.

**Question 2:** When the object is created the name field is set to “alice” and this is returned when getName() is called.

**Question 3:** `bob` is printed out.

**Question 4:** changeName(‘bob’) has changed the value of the name field to “bob”. This is returned when getName() is called.

**Question 5:** The virtual Initio moves forward and then stops.

**Question 6:** Because NameAgent sub-classes InitioAgent it can still use the methods and fields in InitioAgent.

**Question 7:** It creates a cognitive agent that reverses when an object is placed to the left of it, provided the reasoning cycle is running (i.e., the run\_agent() method has been called).

**Question 8:**

>>> bob = ReverseAgent()

>>> bob.run\_agent()

**Exercise 1:**

import bdi.initioagent as cognitive

import time

class ReverseAgent(cognitive.InitioAgent):

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

 cognitive.InitioAgent.\_\_init\_\_()

 self.add\_condition\_rule(self.B('obstacle\_left'), self.reverse\_rule)

 self.add\_condition\_rule(self.B('obstacle\_righ'), self.done)

 def reverse\_rule(self):

 self.robot.reverse(10)

 time.sleep(5)

 self.robot.stop()

**Exercise 2:**

import bdi.initioagent as cognitive

import time

class ForwardAgent(cognitive.InitioAgent):

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

 super().\_\_init\_\_()

 self.add\_condition\_rule(self.AND(self.NOT(self.B('obstacle\_left')), self.NOT(self.B(‘obstacle\_right’))), self.forward\_rule)

 self.add\_condition\_rule(self.B(‘line\_left’), self.done)

 def forward\_rule(self):

 self.robot.forward(10)

 time.sleep(5)

 self.robot.stop()

class ShortForwardAgent(ForwardAgent):

 def forward\_rule(self):

 self.robot.forward(10)

 time.sleep(1)

 self.robot.stop()

**Question 9:** The done() method and \_\_init()\_\_ method from the InitioAgent class.

**Question 10:** It creates a cognitive agent that, when running, will reverse if it encounters an obstacle on the left. If it detects an obstacle on the right it will stop running and print out “Exited the Reasoning Cycle”.

**Ex31**

**Exercise 1:** Note that this answer already includes proximity activation (expanded upon in next exercise) largely for convenience if porting the program to a physical Initio.

import bdi.initioagent as cognitive

import time

class LineFollower(cognitive.InitioAgent):

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

 super().\_\_init\_\_()

 start = self.AND(self.B('obstacle\_left'), self.NOT(self.B('started')))

 stop = self.AND(self.B('obstacle\_left'), self.B('started'))

 on\_line = self.AND(self.B('started'), self.NOT(self.OR(self.B('line\_left'), self.B('line\_right'))))

 line\_on\_left = self.AND(self.B('started'), self.B('line\_left'))

 line\_on\_right = self.AND(self.B('started'), self.B('line\_right'))

 self.add\_condition\_rule(start, self.start\_self)

 self.add\_condition\_rule(stop, self.stop\_self)

 self.add\_condition\_rule(on\_line, self.forward)

 self.add\_condition\_rule(line\_on\_left, self.left)

 self.add\_condition\_rule(line\_on\_right, self.right)

 self.add\_condition\_rule(self.B('stopping'), self.stop\_rule)

 def start\_self(self):

 self.add\_belief('started')

 time.sleep(5)

 return

 def stop\_self(self):

 self.drop\_belief('started')

 self.add\_belief('stopping')

 time.sleep(5)

 return

 def stop\_rule(self):

 self.robot.stop()

 self.done()

 self.drop\_belief('stopping')

 return

 def forward(self):

 self.robot.forward(10)

 return

 def left(self):

 self.robot.spinLeft(10)

 return

 def right(self):

 self.robot.forward(10)

 time.sleep(1)

 self.robot.spinRight(10)

 time.sleep(2)

 return

**Exercise 2:**

import bdi.initioagent as cognitive

import time

class ProximityActivatedAgent(cognitive.InitioAgent):

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

 super().\_\_init\_\_()

 start = self.AND(self.B('obstacle\_left'), self.NOT(self.B('started')))

 stop = self.AND(self.B('obstacle\_left'), self.B('started'))

 self.add\_condition\_rule(start, self.start\_self)

 self.add\_condition\_rule(stop, self.stop\_self)

 self.add\_condition\_rule(self.B('stopping'), self.stop\_rule)

 def start\_self(self):

 self.add\_belief('started')

 time.sleep(5)

 return

 def stop\_self(self):

 self.drop\_belief('started')

 self.add\_belief('stopping')

 time.sleep(5)

 return

 def stop\_rule(self):

 self.robot.stop()

 self.done()

 self.drop\_belief('stopping')

 return

**Exercise 3:**

import bdi.initioagent as cognitive

import time

class ProximityActivatedAgent(cognitive.InitioAgent):

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

 super().\_\_init\_\_()

 start = self.AND(self.B('obstacle\_left'), self.NOT(self.B('started')))

 stop = self.AND(self.B('obstacle\_left'), self.B('started'))

 self.add\_condition\_rule(start, self.start\_self)

 self.add\_condition\_rule(stop, self.stop\_self)

 self.add\_condition\_rule(self.B('stopping'), self.stop\_rule)

 def start\_self(self):

 self.add\_belief('started')

 time.sleep(5)

 return

 def stop\_self(self):

 self.drop\_belief('started')

 self.add\_belief('stopping')

 time.sleep(5)

 return

 def stop\_rule(self):

 self.robot.stop()

 self.done()

 self.drop\_belief('stopping')

 return

import bdi.initioagent as cognitive

import time

class LineFollower(ProximityActivatedAgent):

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

 super().\_\_init\_\_()

 on\_line = self.AND(self.B('started'), self.NOT(self.OR(self.B('line\_left'), self.B('line\_right'))))

 line\_on\_left = self.AND(self.B('started'), self.B('line\_left'))

 line\_on\_right = self.AND(self.B('started'), self.B('line\_right'))

 self.add\_condition\_rule(on\_line, self.forward)

 self.add\_condition\_rule(line\_on\_left, self.left)

 self.add\_condition\_rule(line\_on\_right, self.right)

 def forward(self):

 self.robot.forward(10)

 return

 def left(self):

 self.robot.spinLeft(10)

 return

 def right(self):

 self.robot.forward(10)

 time.sleep(1)

 self.robot.spinRight(10)

 time.sleep(2)

 return

**Exercise 4:**

import bdi.initioagent as cognitive

import time

class SwitchActivatedAgent(cognitive.Pi2GoAgent):

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

 super().\_\_init\_\_()

 start = self.AND(self.B('switch\_pressed'), self.NOT(self.B('started')))

 stop = self.AND(self.B('switch\_pressed'), self.B('started'))

 self.add\_condition\_rule(start, self.start\_self)

 self.add\_condition\_rule(stop, self.stop\_self)

 self.add\_condition\_rule(self.B('stopping'), self.stop\_rule)

 def start\_self(self):

 self.add\_belief('started')

 time.sleep(5)

 return

 def stop\_self(self):

 self.drop\_belief('started')

 self.add\_belief('stopping')

 time.sleep(5)

 return

 def stop\_rule(self):

 self.robot.stop()

 self.done()

 self.drop\_belief('stopping')

 return

import bdi.pi2goagent as cognitive

import time

class WallFollower(SwitchActivatedAgent):

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

 super().\_\_init\_\_()

 wall\_on\_right = self.AND(self.B('started'), self.AND(self.B('obstacle\_right'), self.NOT(self.b\_obstacle\_centre)))

 wall\_in\_front = self.AND(self.B('started'), self.b\_obstacle\_centre)

 lost\_wall = self.AND(self.B('started'), self.NOT(self.OR(self.b\_obstacle\_centre, self.B('obstacle\_right'))))

 floor = self.AND(self.B('started'), self.B('line\_left'))

 self.add\_condition\_rule(floor, self.stop\_rule)

 self.add\_condition\_rule(wall\_on\_right, self.forward)

 self.add\_condition\_rule(wall\_in\_front, self.left)

 self.add\_condition\_rule(lost\_wall, self.right)

 def forward(self):

 self.robot.forward(10)

 return

 def left(self):

 self.robot.spinLeft(10)

 return

 def right(self):

 self.robot.forward(10)

 time.sleep(1)

 self.robot.spinRight(10)

 time.sleep(2)

 return

 def b\_obstacle\_centre():

 if (agent.beliefbase['distance'] < 30):

 return True

 return False



 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/).