Faculty of Engineering, University Of Jaffna

Department Of Computer Engineering

EC9640 – Artificial Intelligence

Lab 01

Date: 05 June 2022 Duration: 3 Hours

Instructions:

- Any plagiarized work will be given 0 marks.
- Submit your lab work according to the instructions given below **on/before given deadline** via teams.
- Failure to adhere to any of the above instructions may also result in zero marks.

Introduction to Python (ref: www.kaggle.com) [10 minutes - Reading]

You may try in out the commands in Jupiter notebook (Start>Anaconda>Anaconda Navigator>Jupiter Notebook)

Note intent decides whether you are in a class, definition, loop, or if condition thus if indent thus the code should ALWAYS be formatted correctly.

Comments

the symbol '#' is used for inline comments in python

Print (similar to 'disp' in MATLAB).

```
print(5 / 2) # prints the value of  5/2
```

Numbers and arithmetic in Python

Operator	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b, removing fractional parts
a % b	Modulus	Integer remainder after division of a by b
a ** b	Exponentiation	a raised to the power of b
-a	Negation	The negative of a

Built-in functions for working with numbers

```
min(1, 2, 3) #gives minimum of all inputs
max(1, 2, 3) #gives maximum of all inputs
abs(32) # gives absolute value of input
```

Getting Help

```
help(round) #here round is the name of a function
```

Functions (this is analogous to functions in MATLAB)

```
def least_difference(a, b, c): # def function_name(input_1, input_2, … , input_n)
 diff1 = abs(a - b)
 diff2 = abs(b - c)
 diff3 = abs(a - c)
 return min(diff1, diff2, diff3)
```

Lists

```
primes = [2, 3, 5, 7] # list of numbers
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune'] # list of strings
my_favourite_things = [32, 'raindrops on roses', help] # list can contain different items
hands = [['J', 'Q', 'K'], ['2', '2', '2'], ['6', 'A', 'K']] # list of lists
```

Indexing

```
planets[0] # get zeroth term
planets[-1] # last term
```

Slicing

```
planets[0:3] # extract elements zero to three
planets[:3] # again first 3
planets[3:] # 3rd to last
planets[1:-1] # All except the first and last
planets[-3:] # The last 3 elements
planets[3] = 'Malacandra' # change the last element
planets[:3] = ['Mur', 'Vee', 'Ur'] # change 3 elements at once
len(planets) # length of planets
sorted(planets) # sort in alphabetical order
sum(primes) # sum of all elements
max(primes) # maximum of elements
planets.append('Pluto') # add element to end of list
planets.pop() # remove from end of list
planets.index('Earth') # the index location of ‘Earth’
"Earth" in planets # returns Boolean output on whether “Earth” is in the list planets
```

Tuples

```
t = (1, 2, 3) # initializing a tuple
t = 1, 2, 3 # equivalent to above
# t[0] = 100 is not valid as values in a tuple cannot be modified
a, b = b, a # swapping variables a and b
```

Loops

Eg.1:

```
planets = ['Mercury', 'Venus', 'Earth', 'Mars', 'Jupiter', 'Saturn', 'Uranus', 'Neptune']
for planet in planets:
    print(planet, end=' ') # print all on same line
```

Out:

Mercury Venus Earth Mars Jupiter Saturn Uranus Neptune

Eg. 2

```
product = 1
for mult in multiplicands:
product = product * mult
product
```

Out:

360

Eg. 3

```
s = 'steganograpHy is the practicE of conceaLing a file, message, image, or video within ano
ther fiLe, message, image, Or video.'
msg = ''
# print all the uppercase letters in s, one at a time
for char in s:
    if char.isupper():
       print(char, end='')
```

Out:

HELLO

Eg. 4

```
for i in range(5):
    print("Doing important work. i =", i)
```

Out:

```
Doing important work. i = 0
Doing important work. i = 1
Doing important work. i = 2
Doing important work. i = 3
Doing important work. i = 4
```

Eg. 5

```
i = 0
while i < 10:
print(i, end=' ')
i += 1
```

Out:

0 1 2 3 4 5 6 7 8 9

Class

A class in python is an object with properties and methods. A constructor is a special method in the class which runs immediately as the class is created

Analogy to MATLAB (MATLAB also has classes but this is just for simpler understanding):

-Class: an instance of MATLAB with its own work space
-properties: variables in the work space assigned different values
-methods: functions
-constructor: a start-up script you run to load some initial values into workspace variables

Constructor

Eg.

```
class transportationProblem
def __init__(self, N):
    self.N =N+8
```

Here,
When you run a command like,
Prob **=** transportationProblem(N=10)
You get,
-An instance of transportationProblem stored in Prob
-With the class property N set to 10
-self refers to that instance of class (as in C++ or Java)

Exercise 1: Water Jug problem using BFS(Breadth-first Search):

Problem Definition:

You are given an **m** liter jug and a **n** liter jug. Both jugs are initially empty. The jugs don't have markings to allow measuring smaller quantities. you have to use the jugs to measure **d** liters of water where **d** is less than **n.**

(X, Y) Corresponds to a state where X refers to the amount of water in Jug 1 and Y refers to the amount of water in Jug 2 Determine the path from the initial state (Xi, Yi) to the final state (Xf, Yf), where (Xi, Yi) is (0, 0) which is indicate both jugs are initially empty and (Xf, Yf) indicates a state which could be (0,d) or (d,0).

The operation you can perform are:
1.Empty a jug, (X,Y) -> (0,Y) Empty Jug 1
2.Fill a jug, (0,0) -> (X,0) fill jug 1
3.Pour water from one jug to the other until one of the jugs is either empty or full, (X,Y) -> (X-d, Y+d)

To Do:

You are to define the following and submit them before the deadline (submit your answer as a pdf)

- A state representation for the above problem
- Define a start state
- Define a goal test corresponding to your state representation
- State a list of possible **actions** and a function to define **successor** state and a **function**
- State tests to check if the successor state is valid

Table 1: States of jugs

Cases	Jug 1	Jug 2	Is valid
Cases 1	Fill it	Empty it	✓
Cases 2	Empty it	Fill it	✓
Cases 3	Fill it	Fill it	Redundant case
Cases 4	Empty it	Empty it	Already visited
Cases 5	Unchanged	Fill it	✓
Cases 6	Fill it	Unchanged	✓
Cases 7	Unchanged	Empty	✓
Cases 8	Empty	Unchanged	✓
Cases 9	Transfer water into this	Transfer water into this	✓
Cases 10	Transfer water into this	Transfer water into this	✓

Exercise 2: Backtracking search:

Definition:
Backtracking search explores nodes until it reaches the bottom like depth first search but it remembers the cost so if multiple solutions are found it chooses the optimum one.

The code for backtracking search for the problem in exercise 1 is given below. you are to modify where marked in red to get the expected output.

```
# uncomment the lines below if you need to increase
the recursion limit
#import sys
#sys.setrecursionlimit(10000)

# This function is used to initialize the
# dictionary elements with a default value.
from collections import defaultdict

# jug1 and jug2 contain the value
# for max capacity in respective jugs
# and aim is the amount of water to be measured.
jug1, jug2, aim = 4, 3, 2

# Initialize dictionary with
# default value as false.
visited = defaultdict(lambda: False)

# Recursive function which prints the
```

```
# intermediate steps to reach the final
# solution and return boolean value
# (True if solution is possible, otherwise False).
# amt1 and amt2 are the amount of water present
# in both jugs at a certain point of time.
def waterJugSolver(amt1, amt2):

# Checks for our goal and
# returns true if achieved.
if (amt1 == aim and amt2 == 0) or (amt2 == aim and
amt1 == 0):
        print(amt1, amt2)
        return True

# Checks if we have already visited the
# combination or not. If not, then it proceeds further.
if visited[(amt1, amt2)] == False:
        print(amt1, amt2)

        # Changes the boolean value of
        # the combination as it is visited.
        visited[(amt1, amt2)] = True

        # Check for all the 6 possibilities and
        # see if a solution is found in any one of
them.
        return (waterJugSolver(0, amt2) or
                        waterJugSolver(amt1, 0) or
                        waterJugSolver(jug1,
amt2) or
                        waterJugSolver(amt1,
jug2) or
                        waterJugSolver(amt1 +
min(amt2, (jug1-amt1)),
                        amt2 - min(amt2, (jug1-
amt1))) or
                        waterJugSolver(amt1 -
min(amt1, (jug2-amt2)),
                        amt2 + min(amt1, (jug2-
amt2))))

# Return False if the combination is
# already visited to avoid repetition otherwise
# recursion will enter an infinite loop.
else:
        return False

print("Steps: ")

# Call the function and pass the
# initial amount of water present in both jugs.
waterJugSolver(0, 0)
```

Exercise 3: Heuristic search:

Problem:
Solve the above problem using uniform cost search

```
def gcd(a, b):
if b==0:
return a
return gcd(b, a%b)


''' fromCap -- Capacity of jug from which
            water is poured
toCap -- Capacity of jug to which
            water is poured
d            -- Amount to be measured '''
def Pour(toJugCap, fromJugCap, d):

fromJug = fromJugCap
toJug = 0

step = 1
while ((fromJug is not d) and (toJug is not d)):


temp = min(fromJug, toJugCap-toJug)

toJug = toJug + temp
fromJug = fromJug - temp

step = step + 1
if ((fromJug == d) or (toJug == d)):
            break
if fromJug == 0:
            fromJug = fromJugCap
            step = step + 1

if toJug == toJugCap:
            toJug = 0
            step = step + 1

return step

def minSteps(n, m, d):
if m> n:
temp = m
m = n
n = temp

if (d%(gcd(n,m)) is not 0):
return -1
```

```
return(min(Pour(n,m,d), Pour(m,n,d)))

if __name__ == '__main__':

n = 3
m = 5
d = 4

print('Minimum number of steps required is',
minSteps(n, m, d))
```

You have to add the comments and explain this code.