Student Name: Lester Hernandez Alfonso
Panther ID: 4017986
Course: CAP4630 – U01
Assignment#2

For each problem:

(a) Describe the two approaches you are using. What do the variables and values represent? What are the constraints? How many variables and constraints are there?

(b) What are the resulting outputs of program? What are the running times of the two approaches? Which one is more efficient? Would the same be true for larger instances?

Problem#1:

a)
1- Approach#1:
https://labix.org/python-constraint

I used the python-constraint library to model and solve the problem using CSP's general formulation.

Variables:
For this problem and approach, I decided to use 8 variables (4 students, 4 colors):
Ella, Henrietta, Omar, Valerie, black, blue, pink, silver.

All variables have the same domain: (15, 25, 35, 45)
Those are the values in feet that they can have.

In the python-constraint library variables are specified using the following syntax:
problem_name.addVariables([variable_names])

Constraints:
General Constraints:
All problems in this assignment have the same general constraints dictated by the characteristics of the puzzle. Those are:

Alldiff(Ella, Henrietta, Omar, Valerie)
Alldiff(black, blue, pink, silver)

These constraints are easy to specify using the python-constraint library. They are formulated with the following syntax:
problem_name.addConstraint(AllDifferentConstraint(),[variable_names])

Problem specific constraints:
In this particular problem there are 5 clues, and using the python constraint library we can add the 5 clues to the formulation using only 5 constraints.

These constraints are specified with the same syntax as the Alldiff() constraint above, except that Alldiff() is replaced by a custom function, specified using python's lambda functions' syntax. For example, the constraint corresponding to clue#3 would be specified as follows

problem.addConstraint(lambda x, y: x > y, ["Omar","silver"])

Where "Omar" and "silver" are input values that will be mapped to x and y in the lambda definition.
The rest of constraints for this problem are self-explanatory and are specified in the same manner.

2-Approach#2

https://pypi.python.org/pypi/PuLP
https://pythonhosted.org/PuLP/

I used the PuLP python library to model and solve this problem using a strict ILP formulation. ILP with no relaxations must express all constraints using only <= and known constants on the right side of the partial inequality.

That is, constraints must be of the form:
Ax <= b
Where x is a variable with a domain greater than or equal to zero, and b is a known constant.

I decided on using the ILP formulation because it was a challenge to express constraints without using ==, !=, or even logical operators like AND, OR.

Variables:
The first challenge when formulating this type of puzzle using ILP is that variables cannot be assigned the same domain as in the CSP approach, because we can't use != to specify the Alldiff() constraints and domains in ILP are either a specific value, or a continuous range.
A workaround suggested in https://pythonhosted.org/PuLP/CaseStudies/a_sudoku_problem.html to solve a problem with similar constraints is to define 4 variables for each variable one would normally define in CSP. These 4 variables would have a range (0-1), and they would each represent the possible values that a single entity can have.
For example, Valerie_15, Valerie_25, Valerie_35 and Valerie_45 would represent the values that Valerie can take on, and only one of those four variables would be equal to 1. The rest would be equal to 0, to comply with the puzzle constraints.

The variables can be defined using the PuLP syntax:

students_vars = LpVariable.dicts("choice",(students, distances),0,1,LpInteger)

That line will create a list of students and distances combinations variables all of which are named in the format:
choice_studentName_distanceNumber (e.g .choice_Valerie_15)

Constraints:
General Constraints:
We do not have Alldiff() so the constraints are specified using a function that restrains the sum of all variables representing a student (e.g. choice_Valerie_anydistance) to no more than1. Then the sum of variables for the same distance is restrained to be at most 1 for only a single student (e.g. choice_anyStudent_15) using the same syntax method:

```
prob += lpSum([students_vars[s][d] for d in distances]) <= 1, ""
 prob += -lpSum([students_vars[s][d] for d in distances]) <= -1, ""
```

One may wonder why two constraints in that format. The reason is that:
x = 1 is expressed using <= as:
x <= 1
x >= 1
which leads to:
x <= 1
-x <= -1

In ILP, an objective function is one which is to be minimized or maximized by the algorithm using constraints.
In our case, the objective function equals 0, we do not care if it is maximized or minimized since there's only one valid solution, and it is both minimal and maximal. Therefore, we would specify:

```
problem_name = LpProblem("anyName",LpMinimize)
prob += 0, "Arbitrary Objective Function"
```

Problem specific constraints:
A single clue could require many constraints in ILP, especially comparative clues since we did not define our variable's domains are integers, but rather as binary in nature.

For example, clue#2 would be defined in the following manner, since the only way for a binary variable to be equal in value to a second variable is for their subtraction always be equal to 0.

```
for d in distances:
  prob += students_vars["Henrietta"][d] - colors_vars["silver"][d] <= 0,""
  prob += -students_vars["Henrietta"][d] + colors_vars["silver"][d] <= 0,""
```

Some clues like #2 can be specified in a few constraints, but others like clue#3 require a substantial number of constraints, since in that specific instance the only way to compare distances in our ILP problem formulation is in a case by case basis.

```
prob += colors_vars["silver"][45] <= 0, ""
prob += students_vars["Omar"][15] <= 0, ""

prob += colors_vars["silver"][35] + students_vars["Omar"][45] <= 2, ""
prob += colors_vars["silver"][35] + students_vars["Omar"][35] <= 1, ""
prob += colors_vars["silver"][35] + students_vars["Omar"][25] <= 1, ""

prob += colors_vars["silver"][25] + students_vars["Omar"][45] <= 2, ""
prob += colors_vars["silver"][25] + students_vars["Omar"][35] <= 2, ""
prob += colors_vars["silver"][25] + students_vars["Omar"][25] <= 1, ""

prob += colors_vars["silver"][15] + students_vars["Omar"][45] <= 2, ""
prob += colors_vars["silver"][15] + students_vars["Omar"][35] <= 2, ""
prob += colors_vars["silver"][15] + students_vars["Omar"][25] <= 2, ""
```

Feel free to look at the code, and inspect how some clues were defined, the set of constraints can be very cleverly structured for some of them.

b)

1- Outputs:

The outputs for both problems show the solution in the following format:
[distance_value1, matching_variable1, matching_variable2]
[distance_value2, …….]

Where matching_variablei is the name of the variable that matched that distance, one of them should be a student, and another a color for problem 1. To see specific outputs to problems, execute the code. The format remains the same for all approaches and all problems.

The output might contain solution sets if the solver returns any, and problem status if the solver returns that field.

It will also list the running time of the algorithm.

2- Running Times:

Running time was calculated using ‘time()’ from the python ‘time’ library, which returns times in seconds with high precision.

The average running time for each approach after 10 executions is shown below rounded to 4 decimal figures:

CSP ----> 0.0013 seconds
ILP------> 0.0372 seconds

ILP is more than 10 times slower than CSP solving this particular problem, on average.

This makes sense since the average clue takes many more constraints to be specified in ILP than in CSP, and both algorithms efficiency is bound by the number of constraints they need to examine. In larger, more complicated problem instances, the difference in performance could be even greater since the more complicated the clues, the more constraints ILP would need per clue, while CSP's constraints to clues ratio is always 1:1.

I used the same approaches for every problem. The outputs are the same. Global, and problem specific constraints are specified in the same manner. The outputs follow the same format. The only things somewhat different are the variables and running time even though the procedure and conclusions are the same. Therefore, for every problem from 2-4, I will just specify the variables, and running time for each approach.

Problem#2

1-Variables:

CSP approach:

12 variables:
"Daily Ray", "Foxy Roxy", "Samantha", "Watery Pete", "Armstrong", "Jacobson", "Romero", "Yang", "Arno's Spit", "Betty Beach", "Rainbow Reef", and "Trey's Tunnel"

Their domain is the number of manatees that can be associated with them:
D = {3, 4, 5, 6}

ILP approach:

48 variables defined in the same format as in problem#1. 4 variables are created from each one of the variables in the CSP approach above. Domain = 0, 1 again.

2- Running Time:

Average:
CSP ----> 0.0016 seconds
ILP -----> 0.0548 seconds

The same conclusion as in problem#1

Problem#3 (Extra Credit)

1-Variables:

CSP approach:

12 variables:
"Cavalo", "Fierro", "Grandero", "Injitsu", "FRZ-192", "GGZ-007", "MRT-628", "YGA-441", "Alaska", "Colorado", "Hawaii", "Louisiana"

Their domain is the fine amount that can be associated with them:
D = {25, 50, 75, 100}

ILP approach:
48 variables defined in the same format as in problem#1. 4 variables are created from each one of the variables in the CSP approach above. Domain = 0, 1 again.

2- Running Time:

Average:
CSP ----> 0.0023 seconds
ILP -----> 0.0538 seconds

The same conclusion as in problem#1 and problem#2. Time did not increase for ILP, because for the ILP formulation problem#3 has the same complexity as problem#2. The same is true for problem#4. CSP got slightly slower due to constraints that involve more variables being slightly more complex.

Problem#4 (Extra Credit)

1-Variables:

CSP approach:
12 variables:
"Bizolam", "Damasol", "Favolin", "Gravon","dengue fever", "diabetes", "heart disease", "influenza", "beetle", "bromeliad","frog", "mushroom"

Their domain is the months that can be associated with them expressed as integer from 1 to 4:
D = {1, 2, 3, 4}

ILP approach:
48 variables defined in the same format as in problem#1. 4 variables are created from each one of the variables in the CSP approach above. Domain = 0, 1 again.

2- Running Time:

Average:
CSP ----> 0.0022 seconds
ILP -----> 0.0553 seconds

The same conclusion as in the previous problems.