Key: CW = code writing, CT = code tracing, SA = short answer, MC = multiple choice,

List methods and recursion

1. **CW:** Write the **recursive** method sumOfPositiveEvens(L) that takes in a list L and returns a sum of all the even and positive integers in the list. For example, sumOfPositiveEvens([1,2,3,4,5]) should return 6 and sumOfPositiveEvens([-2,-4,-6,-8]) should return 0.

```
def sumOfPositiveEvens(L):
    if len(L)==0:
        return 0
    else:
        if L[0]%2==0 and L[0]>=0:
            return L[0]+sumOfPositiveEvens(L[1:])
        else:
            return sumOfPositiveEvens(L[1:])

print(sumOfPositiveEvens([1,2,3,4,5]))
print(sumOfPositiveEvens([-2,-4,-6,-8]))
```

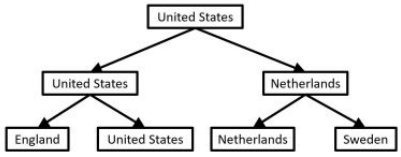
2. **CW:** Write the destructive function shuffleQueue(L, n) that takes in a list of songs represented as strings and an integer n, and shuffles the input list L **destructively** n times by choosing two random songs in the list and switching their locations. If the two songs randomly chosen on a given iteration are the same song, do not count the current iteration in the n shuffles.

```
import random

def shuffleQueue(L, n):
      shuffles = 0
      while shuffles < n:
            song1 = random.randint(0, len(L) - 1)
            song2 = random.randint(0, len(L) - 1)
            if song1 != song2:
                  tempSong = L[song2]
                  L[song2] = L[song1]
                  L[song1] = tempSong
                  shuffles += 1
      return
```

Trees and recursion

1. **CW:** We are back at it representing a tournament as a binary tree. Recall how the information from last time worked… ("United States" is misspelled "United Stated" on accident in a few)

For example, the following bracket represents the last two rounds of the Women's World Cup in 2019.



In our binary tree dictionary format, this would look like:

```
t1 = { "value" : "United Stated",
       "left" : { "value" : "United Stated",
           "left"  : { "value" : "England", "left" : None, "right" : None },
           "right" : { "value" : "United States", "left" : None, "right" : None } },
       "right" : { "value" : "Netherlands",
           "left"  : { "value" : "Netherlands", "left" : None, "right" : None },
           "right" : { "value" : "Sweden", "left" : None, "right" : None } }
     }
```

Write the function opponentsBeat(tree, team) which takes a tournament bracket in (as a tree) and a team from that bracket (as a string), then returns a result list of all the opponents that that team beat in the bracket. Note that it should return an empty [ ] if the team beat no one and should be done **recursively**. Observe that since this is a bracket, it means every node that isn't a leaf must have two children since it takes two teams to play a game.

Hint: Recommended to have 1 base case that checks for a leaf (figure out why that is); observe that if the node is not a leaf, then the children of the node are the team itself and the opponent they beat.

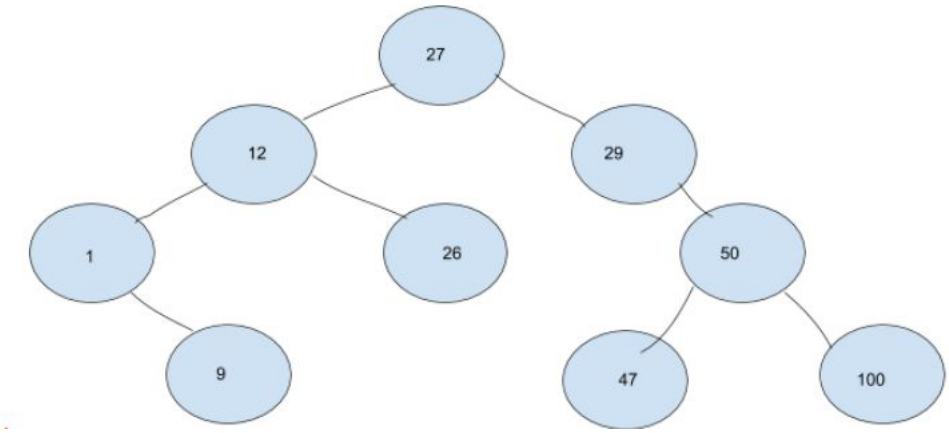
Exs:
opponentsBeat(t1, "United States") → ["Netherlands", "England"]
opponentsBeat(t1, "England") → [ ]

```
def opponentsBeat(tree, team):
      if tree["right"]==None and tree["left"]==None:
            return []
      if tree["value"]==team:
            if tree["right"]["value"] != team:
                  return [tree["right"]["value"]] +
                       opponentsBeat(tree["left"],team)
            else:
                   return [tree["left"]["value"]] +
                        opponentsBeat(tree["right"],team)
```

2. **CT:** Consider the following function:

```
def oddValuesTree(tree):
  if tree == None:
    return []
  else:
    if tree["value"] % 2 == 0:
      return oddValuesTree(tree["left"]) + oddValuesTree(tree["right"])
    else:
      return [tree["value"]] + oddValuesTree(tree["left"]) +
             oddValuesTree(tree["right"])
```

Passed the tree below, what should the return value be?



Answer: [27, 1, 9, 29, 47]

The function has now changed to this…

```
def oddValuesTree(tree):
  if tree == None: #base case
    return []
  else: #recursive case
    if tree["value"] % 2 == 0:
      return oddValuesTree(tree["right"]) + oddValuesTree(tree["left"])
    else:
      return oddValuesTree(tree["right"]) + oddValuesTree(tree["left"]) +
[tree["value"]]
```

Passed the same tree, what is the return value now?
Answer: [47, 29, 9, 1, 27]

Simulation - Randomness & Monte Carlo

1. **CW:**
   Write a simulation function runTrial() that takes in 0 parameters and simulates rolling an eight-sided die 3 times. The function should return True if the first dice roll is EVEN, the second roll is ODD, and that the third roll is LESS THAN 5. Otherwise, it should return False.

   Ex: if Die 1 rolls a 4, Die 2 rolls a 7, and Die 3 rolls a 3 → returns True
   Ex2: If Die 1 rolls 2, Die 2 rolls 5, and Die 3 rolls a 8 → returns False

   Make sure to import the necessary modules!

```
import random

def runTrials():
      rollOne = random.randint(1, 8)
      rollTwo = random.randint(1, 8)
      rollThree = random.randint(1, 8)
      if rollOne % 2 == 0 and rollTwo % 2 == 1 and rollThree <
5:
            return True
      return False
```

2. **MC:**
   Which of these statements is/are false (can be more than one)?
   A. Computers can produce truly random numbers using functions from the random module
      False, computers can only produce pseudo random numbers
   B. Law of large numbers says that if the number of trials is around 100,000, the exact expected value will be reached.
      False, an approximate expected value will be reached, but not the exact value
   C. Testing a simulation with 10 runs of it will give a bad estimate of the expected value
      True, 10 is not enough trials to converge on the approximate expected value
   D. Monte Carlo methods are used to repeat simulations over and over again
      True, this is the basic essence of Monte Carlo methods!

   Answer: A and B are false

Simulation - model/view/controller

1. **CW:** For this problem, you have two different circles, the entries in makeModel marked with a 1 correspond to the first circle and the entries marked with a 2 correspond to the second. Implement this MVC program by modifying makeModel and filling in keyPressed and mousePressed, assuming you have a 400x400 board:
   If the left half of the board is clicked, then only circle 1 is allowed to move
   If the right half is clicked, then only circle 2 is allowed to move
   If a half is clicked and it is the half of the circle currently allowed to move, then neither circle is allowed to move
   If the right or left arrow is pressed, then the circle that's currently allowed to move will move, if neither is allowed, then neither will move

```
def makeModel(data):
      data["cx1"] = 100
      data["cy1"] = 100
      data['cx2'] = 300
      data['cy2'] = 300
      data["size"] = 50
      data["color1"] = "cyan"
      data['color2'] = 'pink'
      data['turn'] = 0

def makeView(data, canvas):
        canvas.create_oval(data["cx1"] - data["size"], data["cy1"] -
data["size"], data["cx1"] + data["size"], data["cy1"] + data["size"],
fill=data["color1"])
        canvas.create_oval(data["cx2"] - data["size"], data["cy2"] -
data["size"], data["cx2"] + data["size"], data["cy2"] + data["size"],
fill=data["color2"])

def keyPressed(data, event):
  if data['turn'] != 0:
         if data['turn'] == 1:
                cx = 'cx1'
                cy = 'cy1'
         elif data['turn'] == 2:
                cx = 'cx2'
                cy = 'cy2'
          if event.keysym == "Left":
                data[cx] = data[cx] - 10
          if event.keysym == "Right":
                data[cx] = data[cx] + 10
```

```
def mousePressed(data, event):
      if event.x < 200:
          if data['turn'] == 1:
                  data['turn'] = 0
          else:
                   data['turn'] = 1
      else:
          if data['turn'] == 2:
                  data['turn'] = 0
          else:
                   data['turn'] = 2
```

2. **CW:** Given the setup below, write a function such that when you click on a random spot on the screen, a circle of size 50 is added to the board. Assume the board is 400x400
The circle is added at the spot clicked and needs to have a randomly selected color(you need at least two colors on the board). (update mouseClicked)
The circle will then start to float until it hits the top of the board(no specific bound, just be close to the top). (update runRules for this)
We will use a dictionary implementation for the circles with relevant entries being displayed in makeview. These circles should be stored in a list in data.

```
def makeModel(data):
      # Set up your simulation components here
      data['circleDicts'] = [ ]

def makeView(data, canvas):
      # The simulation view is written here, using the Tkinter canvas
      for circle in data['circleDicts']:
            cx = circle['cx']
            cy = circle['cy']
            size = circle['size']
            color = circle['color']
            canvas.create_oval(cx-size, cy-size, cx+size, cy+size,
      fill=color)

def runRules(data, call):
      for circle in data['circleDicts']:
            if circle['cy'] > circle['size']/2:
                  circle['cy'] -= 10

def mousePressed(data, event):
      colors = ['yellow', 'pink', 'orange', 'blue']
      color = random.choice(colors)
      circle = {'cx':event.x, 'cy':event.y, 'color':color, 'size':50}
      data['circleDicts'] += [circle]
```

Runtime/Big O

1. **SA** Determining tractability
    a. Consider a problem called 3COL: Given an undirected graph, can we color the vertices with 3 colors so that no two adjacent vertices share the same colors?
        i. Imagine we have an algorithm that solves the 3COL problem by trying all the possible combinations of colors for all the vertices. Is this algorithm tractable or intractable?
        Intractable because a brute force algorithm for this problem will not be in polynomial time.
        ii. Now we have an algorithm such that we go over each edge of the graph and check if both vertices connected by the edge have the same colors, if so we return False, else return True. Is this verifying process tractable or not?
        Tractable. To verify, we can simply go through all the edges in the graph and check each of them which is O(n) where n is the number of edges inside the graph.
        iii. Based on your answers to parts i and ii, what complexity class is the 3COL problem in?
        NP, it can be verified in polynomial time but cannot be solved in polynomial time.

    b. Label the following Big-O complexities as either tractable or intractable
        i. O((5^n)log(n))
        ii. O(n^247)
        iii. O(n!)
        Intractable, tractable, intractable

2. **SA:** What's the big O of the following function?

```
def mystery(n):
    result = 0
    start = 1
    while start <= n:
        result += start
        start *= 2
    for i in range(result * n):
        print("Hello")
```

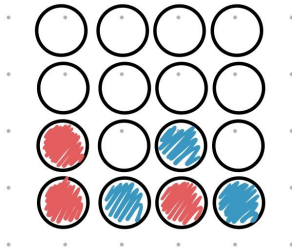
    a. What is the Big-O complexity of this function if we only count the **bolded multiplication step**?
    O(log(n)), as start is multiplied by 2 every iteration, only log(n) iterations of the while loop will occur.
    b. What is the Big-O complexity of this function if we only count the **bolded print statements**?
    O(n^2), the maximum value of result after the while loop is 2n - 1, as we are adding 0 + 1 + 2 + 4 + … + n / 4 + n / 2 + n = 2n - 1. As the for loop iterates through range result * n, the number of iterations is bounded by 2n * n = 2n^2 ⇒ n^2, and we print each iteration.

ML and AI

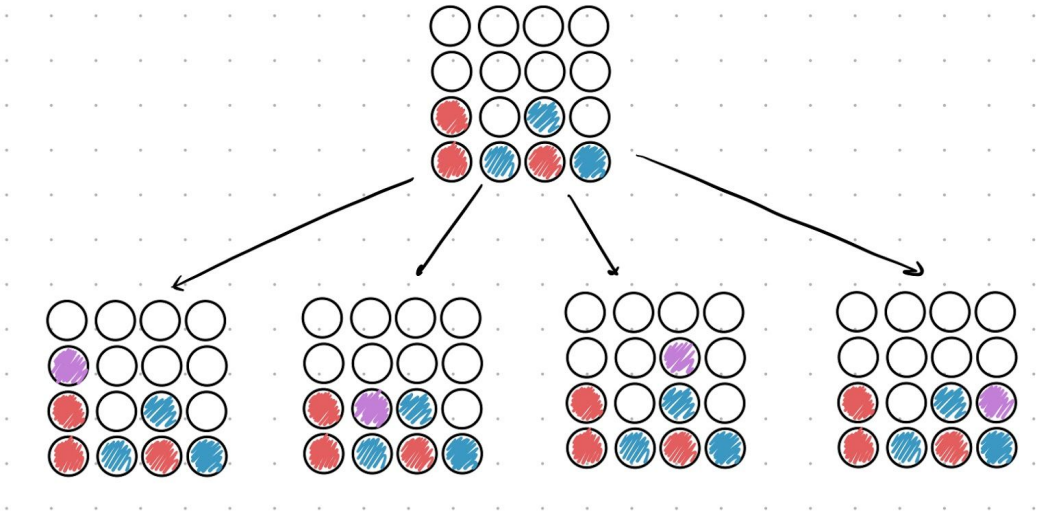
1. **MC:** Categorize each of the following as either a classification, regression, or clustering problem:
   a. Based on a set of symptoms, determine what illness a patient has.
   b. Group a set of pictures into three groups, with similar pictures being in the same group
   c. Using the number of people who show up to a movie premiere to predict how much money the movie will make

   a: classification
   b: clustering
   c: regression

2. **SA:** From this connect-four board, draw the next level of the game tree for the red player. (If you aren't familiar with connect-four, you can read about it here! http://www.ludoteka.com/connect-4.html#:~:text=The%20aim%20for%20both%20players,be%20vertical%2C%20horizontal%20or%20diagonal.&text=Before%20starting%2C%20players%20decide%20randomly,made%20alternatively%2C%20one%20by%20turn)

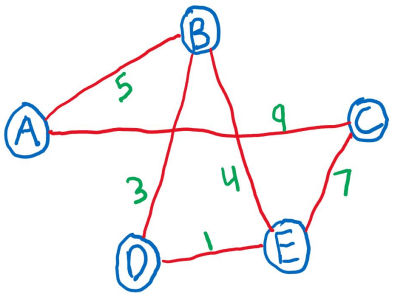


Answer: (purple is the potential next move)



Graphs and Search

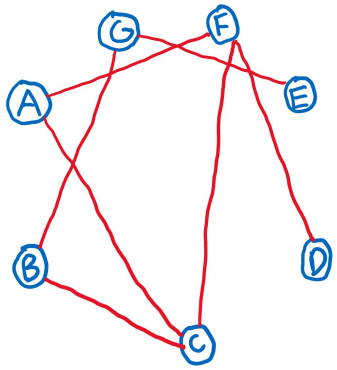
1. **CW:** Write a function maxEdgeWeight that takes in a graph g and returns a list of the maximum weight edge, represented as a 2 element list of the nodes the edge is connected to, connected to each node in the graph. If there are no edges connected to a given node, don't add anything to the result list for this node.



For ex., the output for the above graph would be [["A","C"], ["B","A"], ["C","A"], ["D","B"], ["E","C"]]
Remember that the above graph would be represented as follows:

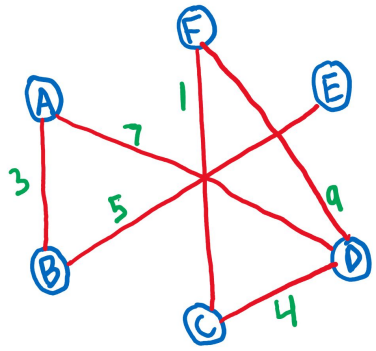
```
g =    {
              "A" : [["B", 5], ["C", 9]],
              "B" : [["A", 5], ["D", 3], ["E", 4]],
              "C" : [["A", 9], ["E", 7]],
              "D" : [["B", 3], ["E", 1]],
              "E" : [["B", 4], ["C", 7], ["D", 1]]
       }

def maxEdgeWeight(g):
       edges = []
       for node in g:
              maxNode = None
              maxWeight = 0
              for neighbor in g[node]:
                     if neighbor[1] > maxWeight:
                            maxWeight = neighbor[1]
                            maxNode = neighbor[0]
              edges.append([node, maxNode])
       return edges
```

2. **SA:** Perform a BFS and DFS starting at node B for the graph shown below:



Answer:
BFS: BCGAFED
DFS: BCAFDGE

Now perform a **weighted** BFS and DFS on the graph below starting from node B, where instead of visiting neighbors in lexicographic order, you visit neighbors based on their connecting edge weight, from greatest to least.



Answer:
BFS: BEADFC
DFS: BEADFC

Search/Sort, Hashing

1. **CT:** Given the following function, trace the

```
def someSort(lst):
     for i in range(len(lst)):
          min_index = i
          for j in range(i, len(lst)):
               if lst[ j ] < lst[ i ]:
                    min_index = j
          # this function destructively swaps the values at the two
indices if they are different indices
          swap(i, min_index)

traceList = [2, 8, 5, 3, 1]
someSort(traceList )
```

i	**list**
0 start (value of lst at the start of the loop iteration	[2, 8, 5, 3, 1]
0 end (value of lst at the end of the loop iteration)	[1, 8, 5, 3, 2]
1 start	[1, 8, 5, 3, 2]
1 end	[1, 2, 5, 3, 8]
2 start	[1, 2, 5, 3, 8]
2 end	[1, 2, 3, 5, 8]
3 start	[1, 2, 3, 5, 8]
3 end	[1, 2, 3, 5, 8]
4 start	[1, 2, 3, 5, 8]
4 end	[1, 2, 3, 5, 8]

2. **MC/SA**
   What is the runtime of mergesort and why?
   Mergesort has an O(nlogn) runtime.
   The logn comes from the number of levels of dividing the input list and then merging it back together, as we divide the list by half until we are left with 1 element sublists, which takes logn splits and then logn merges. The n comes from the number of comparisons we need to do to merge two lists at a given level. As two lists we will be merging at any point are both sorted, we only need to make n comparisons at most to merge them into one sorted list.

   ```
   =>              [1,3,2,4,5,6,7,8]
   =>        [1,2,3,4]          [5,6,7,8]
   =>    [1,2]      [3,4]     [5,6]      [7,8]
   => [1]  [2]    [3] [4]  [5] [6]  [7]   [8]
   ```

3. **MC:** You are given the following hash function which takes in a string elem

```
def hash(elem):
      lowercase = elem.lower()
      first = ord(lowercase[0])
      return first - ord("a")

lst = ["hello", "bazooka", "Christmas", "Santa", "present"]
```

   If we hash the elements in the list above using our hash function, and place them in a hash table of length 5 as shown below, what will be the final state of the hash table? Hash the items in the list from left to right and order them as such in the table below. As an example, “apple” would be hashed to index 0 of the table, as the hash function would return 0.

Index 0	Index 1	Index 2	Index 3	Index 4
“present”	“bazooka”	“hello” “Christmas”	“Santa”	

Authentication & Encryption

1. **SA:** What are the two types of authentication and briefly describe how they work. Give an example of both types.
   Authentication via password is used to verify a person’s identity through text, physical tokens, and other forms when logging into an online service. The password is encrypted and verified by matching the encryption with the server’s database. An example is logging into your andrew account with your ID and password.
   Authentication via certificate is a way for websites to confirm their identity associated with a legit organization. Certificates need to be updated by certificate authorities who issue certificates for websites before they expire. An example is making sure you do not get scammed when making an online purchase by checking if the certificate is valid.

2. **CW:** You and your friend want to send each other secret encrypted messages that only you two can encrypt and decrypt. Write an encryption and decryption function, encryptMessage(key, message) and decryptMessage(key, message), which are both given some key, a list of letters where the letter at the ith index corresponds to the ith+1 letter of the alphabet (i.e. index 0 of key is the letter corresponding to "a", the first letter of the alphabet), and a string message. When encrypting, the letter in the message will determine the index in the key, and the new letter will be letter to the left of the index in the key (if the letter in the message is "a", then the encrypted letter will be the letter corresponding to the index of "z", the last index of the key). Decryption will occur in the opposite way. This way, even if someone were to find the key, they will not be able to decrypt the message immediately.
   You may assume that the messages and keys will all be lowercase letters and that the length of key will always be 26. Leave spaces and punctuations unchanged. An example is shown below:

```
key = ["b", "z", "k", "m", "f", "l", "i", "a", "x", "u", "w", "e",
"q", "h", "j", "o", "p", "n", "c", "v", "t", "g", "s", "d", "r",
"y"]
encryptMessage(key, "hello world") = "imwwh ghpwk"
decryptMessage(key, "lhhkbdm") = "goodbye"
```

   Hint: For encryption, you may find the string function .islower(), which returns whether the string it is called on is all lowercase, useful. For decryption, you may find the list function .index(elem), which returns the first index elem appears in the list it is called on, useful!

```
def encryptMessage(key, message):
	result = ""
	for letter in message:
		if letter.islower():
			index = ord(letter) - ord("a")
			if index == 0:
				i = 25
			else:
				i = index - 1
			newLetter = key[i]
			result += newLetter
		else:
			result += letter
		return result

def decryptMessage(key, message):
	result = ""
	for letter in message:
		if letter.islower():
			index = key.index(letter)
			if index == 25:
				i = 0
			else:
				i = index+1
			newLetter = chr(i+ord("a"))
			result += newLetter
		else:
			result += letter
		return result
```

Parallelism, Pipelining, MapReduce

1. **CW:** The university wants to count the number of full time students for the semester using each student's course information. To be considered full-time, a student must have taken at least 36 units and received a final grade of at least 60 in every class they take. To parallelize this process, the university wants to use MapReduce.
   Course Information is represented as a dictionary with the key being the course name and the value being a two element list, where the first element is the number of units for the course and the second is the students grade in that course.
   Ex:
   ```
   courseDict =  {"Principles of Computing":[10,90], "Calculus 1":[10,92],
   "Principles of Microeconomics":[9,83],"Experimental
   Chemistry":[6,77],"Global Business": [9,96]}
   ```
   a. Write the map function that takes a student's course information for the semester and maps it to a boolean value representing the students full-time status.
   ```
   def map(courseDict):
         unitCount = 0
         for key in courseDict:
               if courseDict[key][1] < 60:
                     return False
               else:
                     unitCount = unitCount + courseDict[key][0]
         if unitCount < 36:
               return False
         else:
               return True
   ```

   b. Write the reduce function that takes the results from the map function and outputs the number of full time students
   ```
   def reduce(lst):
         count = 0
         for elem in lst:
               if elem == True:
                     count += 1
         return count
   ```

2. **MC/SA**
   Select which of the following are True:
   a) Deadlocks can be resolved by programs locking important resources that other programs need
      False, locking resources causes deadlocks and does not resolve it
   b) Pipelining is useful in scenarios where every sub-task is dependant on the previous sub-task and the task is performed only once
      False, if every task is performed only once, pipelining does not improve the time taken
   c) The length of time of a pipelined process depends on the length of the longest task
      True, the length of the longest task is the limiting factor on pipeline time
   d) The mapper passes on its output directly to the reducer which then combines the output
      False, the collector takes the output from each mapper, combines it, and sends it to the reducer

   Answer: Only c is True

Data Analysis/Visualization

1. **CW:** Write a function that randomly samples 100 choices from a list that contains 4 flavors of ice cream (Vanilla, Chocolate, Mint, and Caramel), and then creates a bar chart of the number of times each flavor was sampled. For example, if each flavor is sampled 25 times, then the corresponding bar chart should have 4 bars (1 for each flavor) each with height 25. Use the random.choice() function from the random library, and use the plt.bar() function from the matplotlib library. The plt.bar() function takes two parameters: the first is a list of labels for the bars on the x-axis, and the second is the list of heights corresponding to each bar.

   Answer

```
import matplotlib.pyplot as plt
import random

flavors = ["Vanilla", "Chocolate", "Mint", "Caramel"]
flavor_choices = {"Vanilla":0, "Chocolate":0, "Mint":0, "Caramel":0}

n_choice = 100
for i in range(n_choice):
      choice = random.choice(flavors)
      Flavor_choices[choice] += 1

choices = []
counts = []

for i in flavor_choices:
      choices.append(i)
      counts.append(flavor_choices[i])

plt.bar(choices,counts)
plt.show()
```

2. **SA:** Given the following types of data, name the best visualization method to display that data:

   A. Numerical × Ordinal
   B. Numerical
   C. Ordinal × Ordinal
   D. Categorical
   E. Numerical × Numerical × Numerical

   Answer

   F. Box-and-Whisker Plot
   G. Histogram
   H. Trick question! Use a table
   I. Pie Chart
   J. Bubble Plot / Scatter Plot Matrix

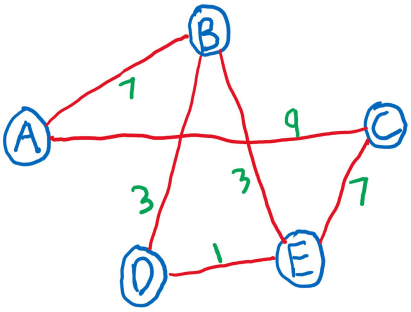
Dicts and Trees/Graphs

1. **CW:** Write a function combineCovidDicts that takes in two dictionaries, dict1 and dict2, each of which contains keys that represent cities (strings) with the corresponding value being the percentage of individuals in that city with Covid (float). This function should output a result dictionary which combines the two inputs as specified below:
    - If a city is key both dict1 and dict2, set that city as a key in your result dictionary with a value equal to the percentage for that city in dict2 - the percentage for that city in dict1
    - If a city is only in dict2, then just set that city as a key in your result dictionary with a value equal to the percentage for that city in dict2
    - If a city is only in dict1, then just set that city as a key in your result dictionary with a value equal to the percentage for that city in dict1

```
def combineCovidDicts(dict1, dict2):
      resultDict = {}
      for city in dict2:
            if city in dict1:
                  resultDict[city] = dict2[city] - dict1[city]
            else:
                  resultDict[city] = dict2[city]
      for city in dict1:
            if city not in dict2:
                  resultDict[city] = dict1[city]
      return resultDict
```

2. **CT:** Given the following mystery function, what would it return when given the graph below as input?

```
def f(g):
    result = {}
    for node in g:
        for neighbor in g[node]:
            if neighbor[1] not in result:
                result[neighbor[1]] = []
            result[neighbor[1]].append([node, neighbor[0]])
    return result
```



Remember that the above graph would be represented as follows:

```
g =   {   "A" : [["B", 7], ["C", 9]],
          "B" : [["A", 7], ["D", 3], ["E", 3]],
          "C" : [["A", 9], ["E", 7]],
          "D" : [["B", 3], ["E", 1]],
          "E" : [["B", 3], ["C", 7], ["D", 1]]
      }
```

Answer:

```
g =   {
          1 : [["D", "E"], ["E", "D"]],
          3 : [["B", "D"], ["B", "E"], ["D", "B"], ["E", "B"]],
          7 : [["A", "B"], ["B", "A"], ["C", "E"], ["E", "C"]],
          9 : [["A", "C"], ["C", "A"]]
      }
```

Concurrency, Internet

1. **SA:**
   You find that when you download a movie from movielovers.com, it takes much more time than other websites to download the document due to an extremely slow internet speed. The internet company claims that it happens because the movielovers.com did not pay them for the excessive internet access due to the huge number of movies downloaded. What term describes the company's behavior and what principle has the company violated?
   Answer: Throttling. Net neutrality.

2. **MC:** Match the following internet terms with their corresponding definitions:

Terms:	Definitions:
Packet	A computer that stores URL's and their associated IP addresses
DNS Server	A type of system that has failsafes and backups in place for when things go wrong
IP Address	Type of organization that connects a user's computer to the internet
Fault Tolerant	Information sent to an IP address
ISP	The set of numbers that uniquely identifies a computer hosting a website

   1 → 4, 2 → 1, 3 → 5, 4 → 2, 5 → 3

3. **SA**
   Draw out the concurrency tree for the following expression: [((a+b)**c)*(d-2*e)]/f
   How many levels are there?
   What's the number of steps?
   What's the number of time steps in the concurrency tree (time steps = non-leaf levels)?

   Levels: 5
   Steps: 6
   Time steps: 4