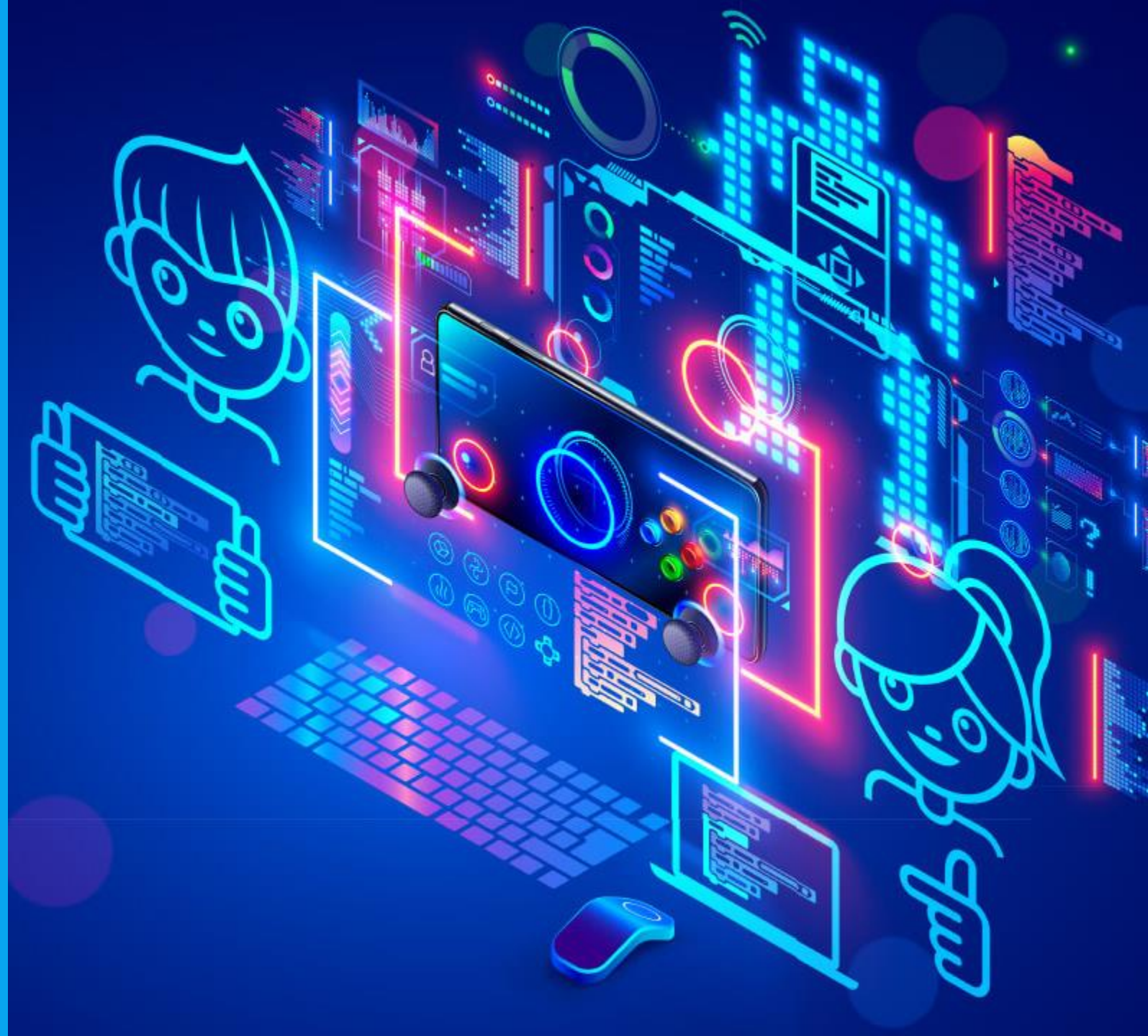
CS 24 AP Computer Science A Review

Week 10: FRQ and Algorithm II

DR. ERIC CHOU
IEEE SENIOR MEMBER

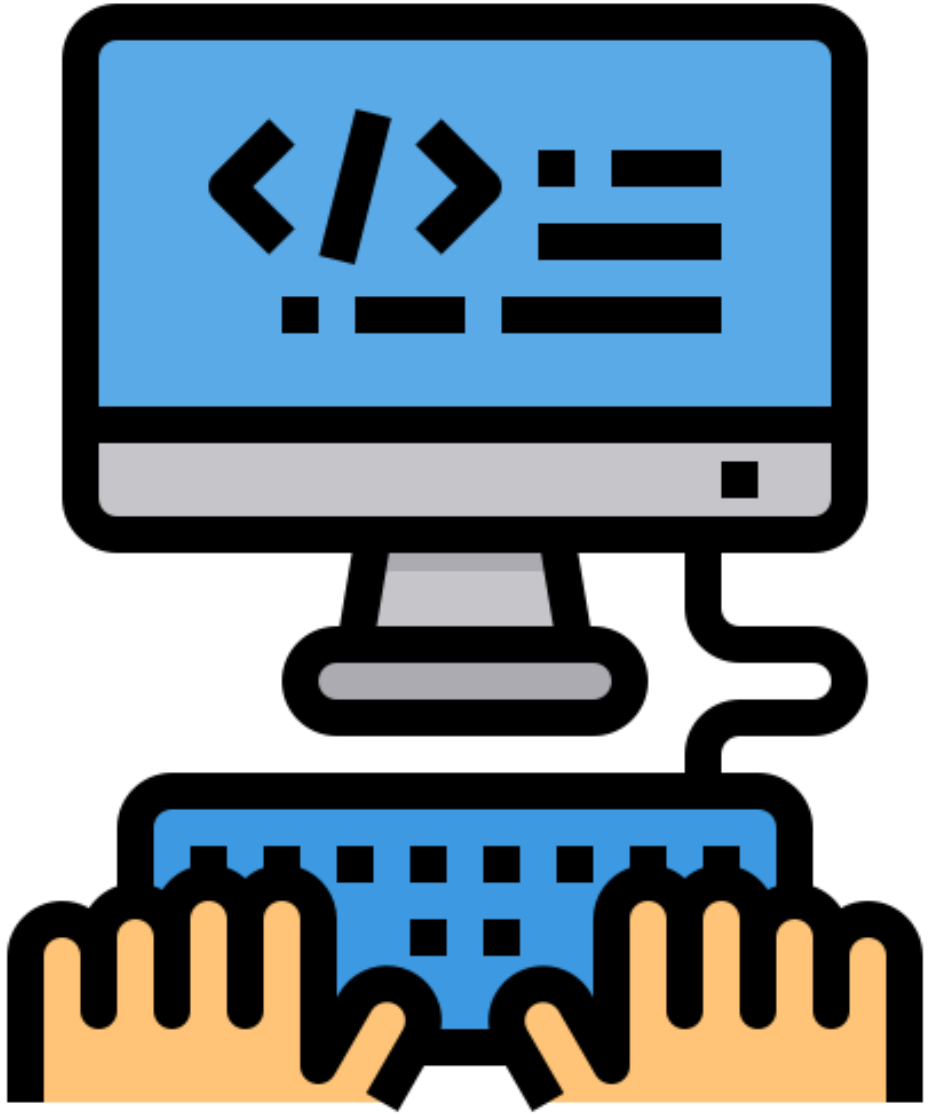


Topics

- String traversal using indexOf()
- Print a list with delimiters
- String alignment (Centered, Left, Right and Justified)
- pos Pattern Search
- 2D to 1-D Serialization (Using Stacked Index)
- Transcopy (1-D to 2D, 2D to 1-D, Partial Array Copy)
- Use of flag (skip)
- Intervals and cuts
- Filtering using ArrayList
- Removal of Element from a List

Topics

- Histogram
- 2-D Iterator, 2D Traversal using Iterator
- Path-Creation (Back-up from a list)
- 2D Column-Major Traversal, Zig-Zag Traversal
- 2D Mirror, Flop and Move
- 2D Matrix (Running Index, Reduced number of Rows CB 2021 Q4b)
- Balance of Parenthesis



String Problems

Section 1

Traversal by IndexOf

- Use of indexOf(pattern) and IndexOf(pattern, from) functions
- **pos**-method for traversal.

- Problem: How many “ab” patterns in a string?

```java
public class UseIndexOf
{
    static String s = "aasbabdfsaabbababewriuwbaavbabba";

    public static int countAB(String s, String pat){
        int c= 0;
        int pos = s.indexOf(pat);
        while (pos >=0){
            c++;
            pos = s.indexOf(pat, pos+1);
        }
        return c;
    }

    public static void main(String[] args){
       System.out.printf("How many ab in s? %d\n", countAB(s, "ab"));
    }
}
```

Overlapping and Non-overlapping

- Overlapping allows a pattern to be counted multiple times with shared characters.
- Non-overlapping does not allow any sharing of characters in a string for counting of the patterns.

Overlapping allowed

```
public static int countAB(String s, String pat){
    int c= 0;
    int pos = s.indexOf(pat);
    while (pos >=0){
        c++;
        pos = s.indexOf(pat, pos+1);
    }
    return c;
}
```

Overlapping not Allowed

```
public static int countABNonOverlapping(String s, String pat){
    int c= 0;
    int pos = s.indexOf(pat);
    while (pos >=0){
        c++;
        pos = s.indexOf(pat, pos+pat.length());
    }
    return c;
}
```

Print a list with delimiters

There are two ways to add delimiters:

```
1. a     +b     +c    +d
2. a+    b+     c+    d
```

a +b +c +d

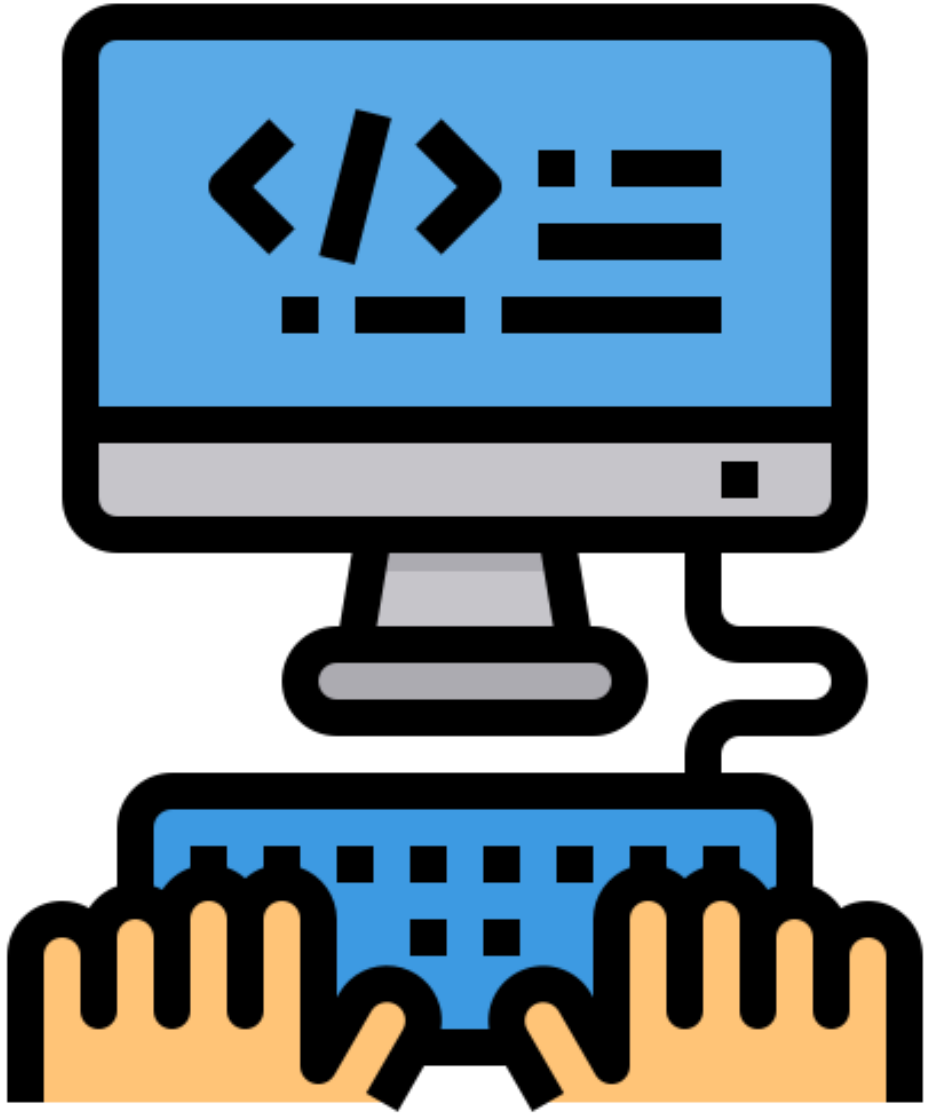
```
public static String addDelimiters(int[] a, String del){
    String s = "";
    for (int i=0; i< a.length; i++){
        if (i==0) s += a[i];
        else s += del+a[i];
     }
    return s;
}
```

a+ b+ c+ d

```
public static String addDelimiters2(int[] a, String del){
    String s = "";
    for (int i=0; i< a.length; i++){
        if (i==a.length-1) s += a[i];
        else s += a[i]+del;
     }
    return s;
}
```

String alignment (Centered, Left, Right and Justified)

- Advanced version of adding delimiters.
- Delimiters
- Spacing
- Extra-space,
- Counting of words.



Transcopy

Section 2

2D to 1-D Serialization (Using Stacked Index)

- Using running index
- Using stacked index

```java
public static int[] TwoToOne(int[][] m){
    int p = 0;
    int[] a = new int[m.length*m[0].length];

    for (int r=0; r<m.length; r++){
        for (int c=0; c<m[0].length; c++){
            a[p++] = m[r][c];
         }
     }
    return a;
}
```

Using Running Index

```java
public static int[] TwoToOne2(int[][] m){
    int p = 0;
    int[] a = new int[m.length*m[0].length];

    for (int r=0; r<m.length; r++){
        for (int c=0; c<m[0].length; c++){
            a[r*m[0].length+c] = m[r][c];
         }
     }
    return a;
}
```

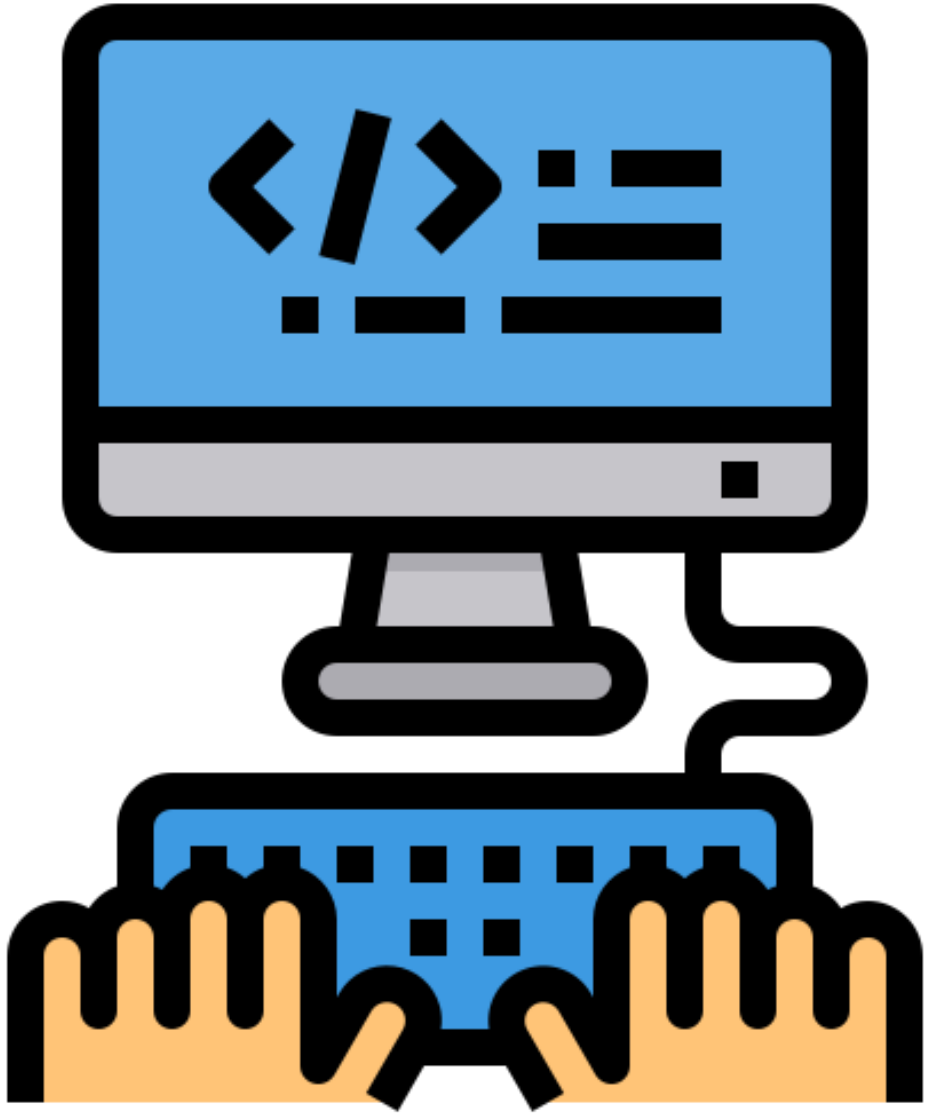
Using Stacked Index

Transcopy (1-D to 2D, 2D to 1-D, Partial Array Copy)

- Using running index on the 1-d Array
- 2-D array use the matrix standard traversal
- Must use p<a.length for the index qualification

Transcopy (1-D to 2D, 2D to 1-D, Partial Array Copy)

```
public static void fillMatrix(int[][] m, int[] a){
    int p=0;
    for (int i=0; i<m.length; i++){
        for (int j=0; j<m[0].length; j++){
            if (p<a.length) m[i][j] = a[p++];
        }
    }
}
```



Use of Flags, Interval and Run-length

Section 3

Use of flag (skip)

- Return the sum of the numbers in the array, except ignore sections of numbers starting with a 6 and extending to the next 7 (every 6 will be followed by at least one 7). Return 0 for no numbers. (Coding Bat Array-2)

```java
static int[] a = { 1, 2, 6, 3, 4, 7, 5, 6, 7, 8, 9};

public static int sum67(int[] a){
  int s = 0;
  boolean skip = false;

  for (int i=0; i<a.length; i++){
      if (a[i]==6) skip = true;
      if (!skip) s += a[i];
      if (a[i]==7) skip = false;
    }

  return s;
}
```

Intervals and cuts

- A series of the same numbers are named a run. Please return the length of the longest run.

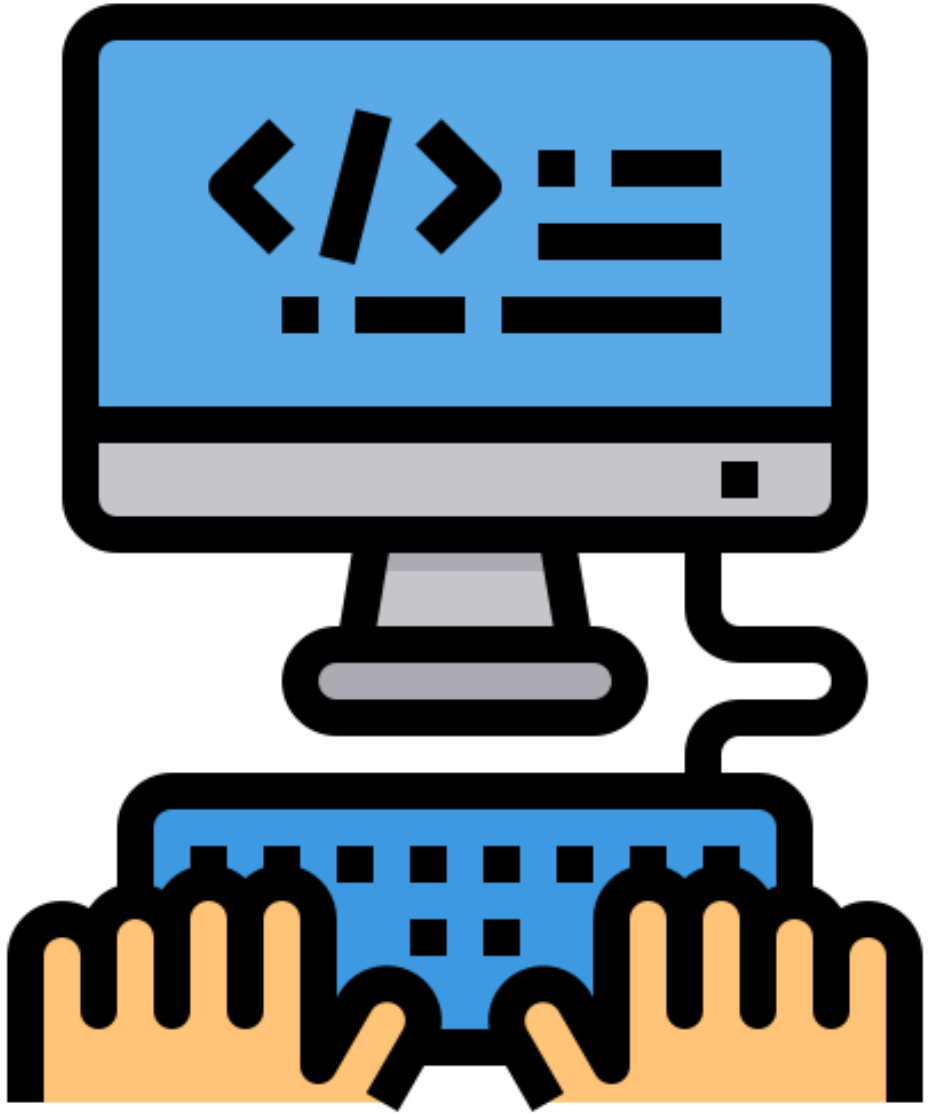
```java
public static int run(int[] a){
   int m=0;
   int r=1;
   for (int i=1; i<=a.length; i++){
        if (i==a.length || a[i]!=a[i-1]){
            if (r>m) m = r;
            r=1;
         }
        else r++;
     }
   return m;
}
```

Evaluate on Change

```java
public static int run2(int[] a){
  ArrayList<Integer> cuts = new ArrayList<Integer>();
  cuts.add(0);
  for (int i=1; i<a.length; i++){
      if (a[i-1] != a[i]) cuts.add(i);
    }
  cuts.add(a.length);

  int m=0;
  for (int i=0;i<cuts.size()-1; i++){
      int step = cuts.get(i+1)-cuts.get(i);
      if (step>m) m = step;
    }
  return m;
}
```

Use of Interval List



ArrayList Algorithms

Section 4

Filtering using ArrayList

- Select only the qualified numbers.
- Use of ArrayList

```java
public class KeepEvens
{
    static Integer[] a= {1, 2, 3, 4, 5, 6, 7, 8, 9};

    public static ArrayList<Integer> getEvens(Integer[] a){
        ArrayList<Integer> evens = new ArrayList<Integer>();

        for (int i=0; i<a.length; i++){
            if (a[i]%2==0) evens.add(a[i]);
          }
        return evens;
     }

    public static void main(String[] args){
        ArrayList<Integer> evens = getEvens(a);
        System.out.println(evens);
     }
}
```

Keep Even Numbers

Removal of Element from a List

- Remove the even Numbers
- Backward Traversal or Forward without advancing index on removals.

Histogram

- Fixed Size Histogram (Using Array)
- Flexible Size Histogram (Using ArrayList)

Fixed Size Histogram (Using Array)

- How many times the letters from a to z occurs in the Gettsburg address by Abraham Lincoln?
- Ignore case
- Ignore symbols and numbers

```java
public static int[] getHistogram(String s){
    int[] h = new int[26];
    s = s.toLowerCase();
    for (int i=0; i<s.length(); i++){
        char c = s.charAt(i);
        if (Character.isLetter(c)){
            h[c-'a']++;
          }
     }
    return h;
}
```

```
public static void printHistogram(int[] h){
    for (char c='a'; c<='z'; c++){
        System.out.printf("%c, %d\n", c, h[c-'a']);
     }
}

public static void main(String[] args){
    int[] h = getHistogram(s);
    printHistogram(h);
}
```

Flexible Size Histogram (Using ArrayList)

- Use Frequency List and Non-Recurring Set

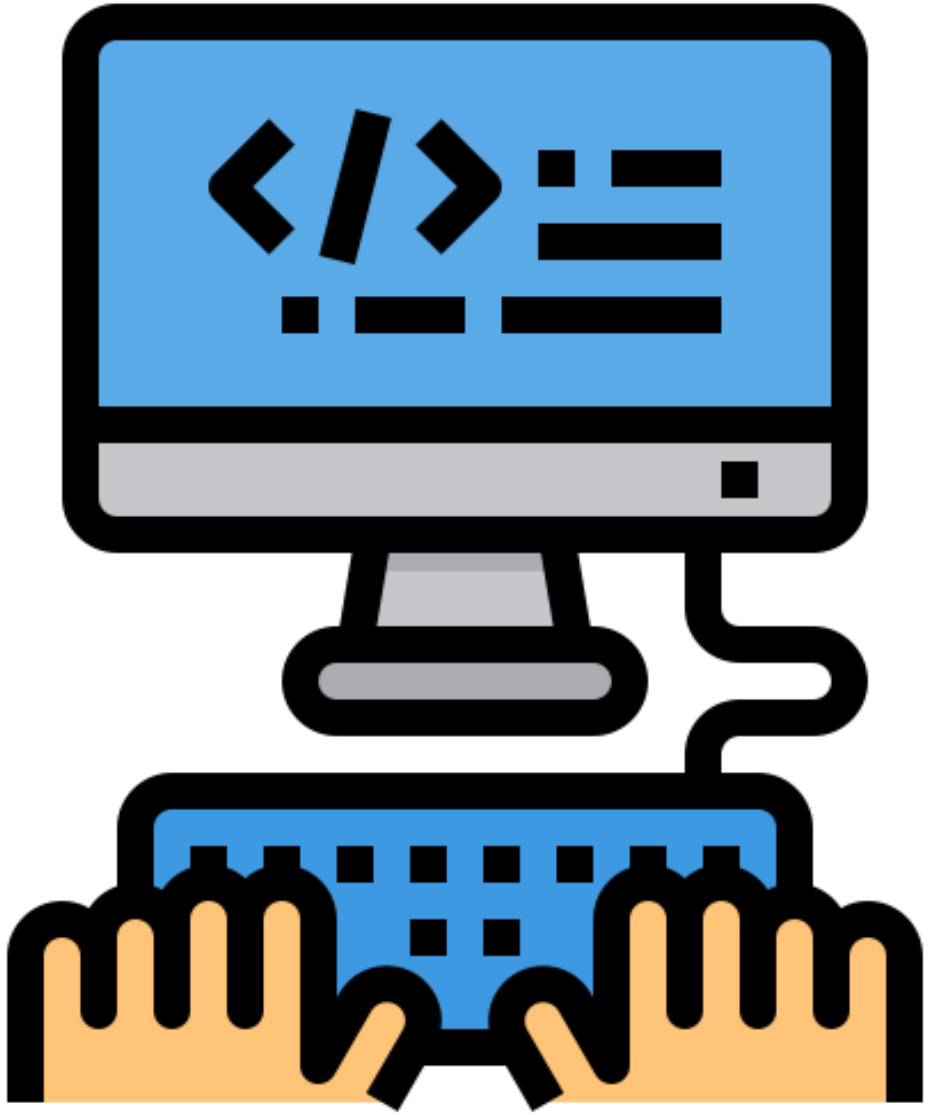
Flexible Size Histogram (Using ArrayList)

- How many times the letters from a to z occurs in the Gettsburg address by Abraham Lincoln?
- Ignore case
- Ignore symbols and numbers

```
static ArrayList<Integer> h = new ArrayList<Integer>();
static ArrayList<String> w  = new ArrayList<String>();
public static ArrayList<Integer> getHistogram(String s){
    s = s.toLowerCase();
    h.clear();
    w.clear();
    for (char x: s.toCharArray()){
       if (Character.isLetter(x)){
           String ch = ""+x;
           int idx = w.indexOf(ch);
           if (idx<0){
               w.add(ch);
               h.add(0);
               idx = w.size()-1;
            }
           h.set(idx, h.get(idx)+1);
        }
    }
    return h;
}
```

```java
public static void printHistogram(ArrayList<String> w, ArrayList<Integer> h){
    for (int i=0; i<w.size(); i++){
        System.out.printf("%s, %d\n", w.get(i), h.get(i));
     }
}

public static void main(String[] args){
    h = getHistogram(s);
    printHistogram(w, h);
}
```



Two-Dimensional Traversal

Section 5

2-D Iterator, 2D Traversal using Iterator

- The concept of Iterator. Iterator is a point which will perform serialized traversal over a data structure.

- Here we demonstrate a 2-D iterator. Using an iterator class.

```
public class TwoDIterator
{
    int M; // number of rows
    int N; // number of columns
    int itr =0;

    TwoDIterator(int m, int n){ M=m; N=n;  itr=0; }

    public void reset(){ itr=0; }

    public boolean hasNext(){ return itr >=0 && itr < M*N; }

    public int getRow(){ return itr/N; }
    public int getColumn(){ return itr%N; }

    public int next(){
        int x = itr;
        itr++;
        if (x >= M*N) return -1;
        return x;
    }
}
```

```java
public class TwoDimensionalTraversal
{
    static int[][] m = {
       {1, 2, 3, 4},
       {5, 6, 7, 8},
       {9, 10, 11, 12}
     };

    public static void traversal(int[][] m){
        TwoDIterator td = new TwoDIterator(m.length, m[0].length);
        td.reset();

        while(td.hasNext()){
            System.out.println(m[td.getRow()][td.getColumn()]);
            td.next();
         }
     }

    public static void main(String[] args){
        traversal(m);
     }
}
```

Main Program

Path-Creation (Back-up from a list)

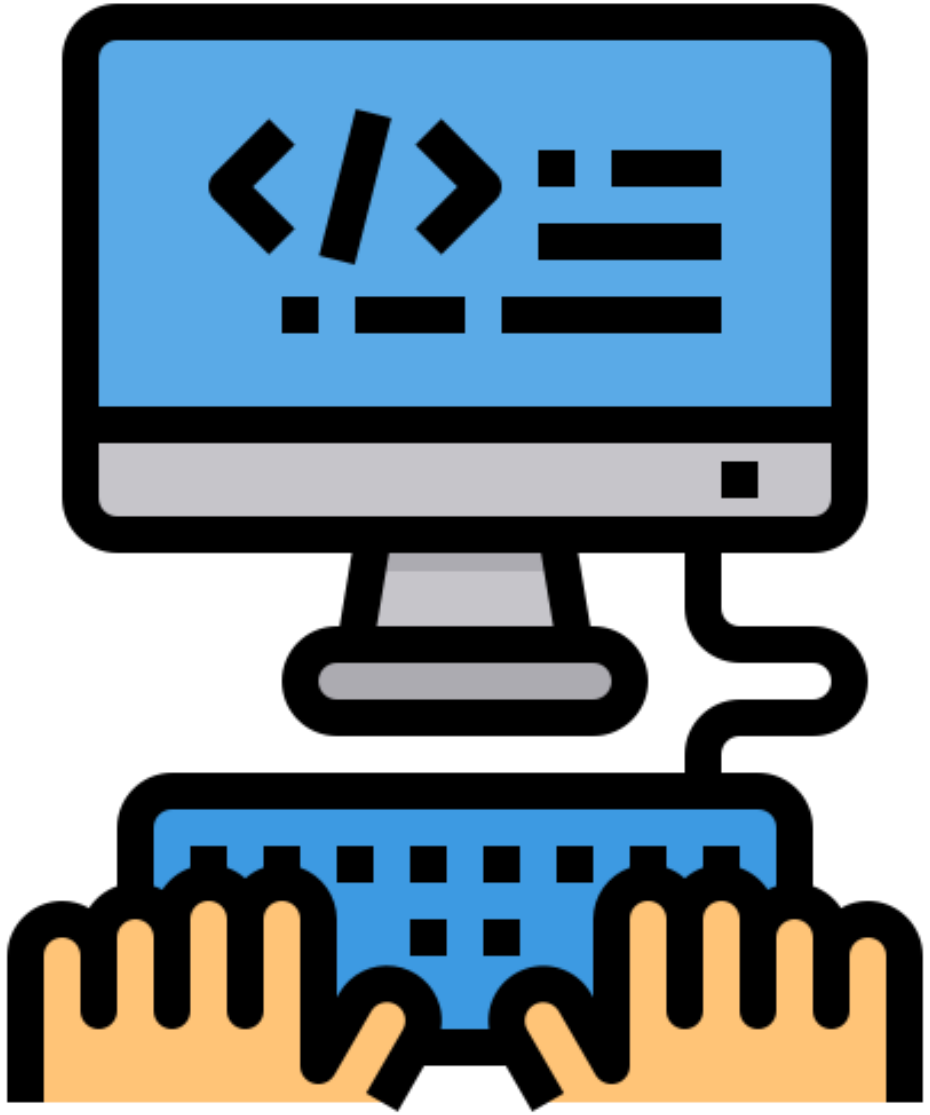
- Store the path into a stack.
- Push and pop the path nodes into the stack.

```java
public class Location
{
    int x;
    int y;
    Location(int x, int y){ this.x = x; this.y =y; }

    public int getRow(){ return x;}
    public int getColumn(){ return y;}
    public Location next(){ return new Location((int)(Math.random()*3), (int)(Math.random()*4)); }

    public String toString(){
       return "<"+x+", "+y+">";
    }
}
```

```
10      public static void main(String[] args){
11          Location a = new Location(3, 4);
12
13          ArrayList<Location> alist = new ArrayList<Location>();
14
15          for (int i=0; i<10; i++){
16              alist.add(a.next());
17           }
18          System.out.println("Forward: ");
19          for (Location x: alist){
20               int row = x.getRow();
21               int column = x.getColumn();
22               System.out.printf("m[%d][%d] = %d\n", row, column, m[row][column]);
23           }
24          System.out.println();
25          System.out.println("Backup: ");
26          while (alist.size()>0){
27              Location x = alist.remove(alist.size()-1);
28               int row = x.getRow();
29               int column = x.getColumn();
30               System.out.printf("m[%d][%d] = %d\n", row, column, m[row][column]);
31           }
32       }
33  }
```



Matrix Algorithms

Section 6

2D Column-Major Traversal, Zig-Zag Traversal

- Use proper indexing.

```
11  public static void traversal(int[][] m){
12      for (int i=0; i<m.length; i++){
13          if (i%2==0){
14              for (int j=0; j<m[i].length; j++){
15                  System.out.printf("%3d", m[i][j]);
16               }
17              System.out.println();
18            }
19          else{
20            for (int j=m[i].length-1; j>=0; j--){
21                  System.out.printf("%3d", m[i][j]);
22               }
23              System.out.println();
24            }
25       }
26   }
```

Zig-Zag

```
28  public static void columnMajor(int[][] m){
29      for (int c=0; c<m[0].length; c++){
30          for (int r=0; r<m.length; r++){
31              System.out.printf("%3d", m[r][c]);
32           }
33          System.out.println();
34       }
35   }
```

Column Major

2D Mirror

- Performing Mirror or Swap Operation

```
20  public static void mirrorHorizontal(int[][] m){
21      for (int i=0; i<m.length; i++){
22          for (int j=0; j<m[i].length/2; j++){
23              int tmp = m[i][j];
24              m[i][j] = m[i][m[i].length-1-j];
25              m[i][m[i].length-1-j] = tmp;
26           }
27       }
28   }
```

Mirror Horizontally

```
30  public static void mirrorVertical(int[][] m){
31      for (int i=0; i<m.length/2; i++){
32          for (int j=0; j<m[i].length; j++){
33              int tmp = m[i][j];
34              m[i][j] = m[m.length-1-i][j];
35              m[m.length-1-i][j] = tmp;
36           }
37       }
38   }
```

Mirror Vertically

2D Matrix (Running Index, Reduced number of Rows CB 2021 Q4b)

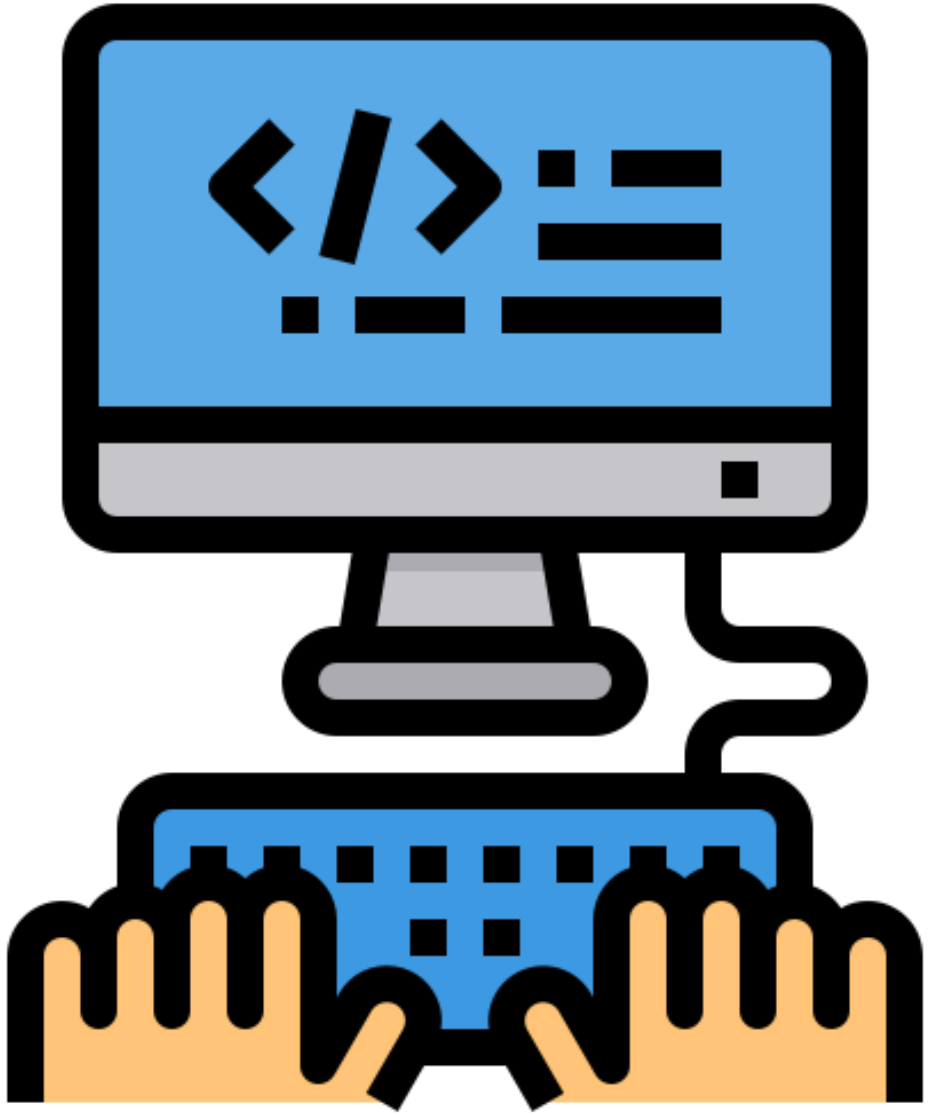
- Similar to 1-D deletion of an element. Must perform shifting.

```java
public static int[][] reduce(int[][] m, int row){
  if (row <0 || row >=m.length) return m; // no change

  int[][] n = new int[m.length-1][m[0].length];

  for (int r=0; r<row; r++){
      for (int c=0; c<m[0].length; c++){
         n[r][c] = m[r][c];
       }
    }

  for (int r=row; r<m.length-1; r++){
      for (int c=0; c<m[0].length; c++){
         n[r][c] = m[r+1][c];
        }
    }
  return n;
}
```



Balance of Parenthesis

Section 7

Balance of Parenthesis

Check if parenthesis are balanced?

E.g.

```
)() unbalanced
())  unbalanced
()(()()) balanced.
```

Balance of Parenthesis

1. Each (will have a) on the right to balance
2. A) without a proceeding (is considered unbalanced.

```java
public class Balanced
{
    static String[] patterns = {
     ")()", "()()", "())", "(()(()))"
    };
    public static boolean balanced(String s){
        int level = 0;
        for (int i=0; i<s.length(); i++){
            if (s.charAt(i)=='(') level++;
            if (s.charAt(i)==')') level--;
            if (level<0) return false;
         }

        return level ==0;
    }

    public static void main(String[] args){
        for (int i=0; i<patterns.length; i++){
            System.out.printf("Pattern %s is balanced is %b\n",
            patterns[i], balanced(patterns[i]));
         }
    }
}
```