# Lambda Expression

→ One-line or inline functions → code will be a oneline anonymous function
↳ no name, no modifier, no return type, no datatypes of parameter

→ Benefits :→ (1) reduce code lines, makes code readable/maintainable/concise
(2) Sequential & parallel execution support → passing functions as parameters
(3) Calls API very efficiently

→ Inbuilt Functions using Lambda Expression ⇒ forEach method

→ Custom Lamba Expression
- → Priority Queue → Comparable / Comparator
- → Arrays.sort or Collections.sort → Reverse Order / Custom order (Pairs)

```
public String concatenate(String a, String b) {
    return a+b;
}
```

⇓ Type Infer at Compile time

```
(a, b) → {
    return a+b;
}
```

```java
Run | Debug
public static void main(String[] args) {
    int[] arr = { 10, 20, 30, 40, 50 };

    // FOR EACH LOOP (Iterable)
    for (int val : arr) {
        System.out.print(val + " ");
    }
    System.out.println();

    // FOR EACH METHOD
    ArrayList<Integer> al = new ArrayList<>();
    al.add(e: 10);
    al.add(e: 20);
    al.add(e: 30);
    al.add(e: 40);

    al.forEach((val) -> System.out.print(val + " "));
    System.out.println();
}
```

anonymous function or arrow function = Lambda Expression

Syntax ②

Curly braces can be omitted only if one line instruction is there

Syntax ①

```java
al.forEach((val) -> {
    System.out.print(val + " ");
});
System.out.println();
```

```java
al.forEach((val) -> {
    if (val % 2 == 0)
        System.out.print(s: "Even ");
    else
        System.out.print(s: "Odd ");
});
```

Comparator
↳ many

Same class whose objects to be compared

By default Sorting

Comparable
↳ unique

```
public int compare(Object a, Object b){
    ① return a.val - b.val;
    ② return b.val - a.val
}
```

① a-b = -ve → Smaller will be placed first
⇒ Increasing order or min Heap

② b-a = +ve → Larger will be placed first
⇒ Decreasing order or max Heap

```
public int compareTo(Object other){
    ① return this.val - other.val;   → Increasing order
    ② return other.val - this.val;   → Decreasing order
}
```

# Arrays.sort

## Approach (1) Custom Lambda Expression

Marks of PCM (GFG)

```java
public static void customLambdaExpression() {
    int[] arr = { 50, 30, 80, 90, 10, 20, 70, 40, 100, 60 };
    Arrays.sort(arr);
    // Arrays.sort(arr, comparator); INVALID FOR PRIMITIVES

    // Increasing Order : Default
    // System.out.println(arr); // HashCode

    for (int val : arr)
        System.out.print(val + " ");
    System.out.println();

    Integer[] copy = new Integer[arr.length];
    for (int idx = 0; idx < arr.length; idx++)
        copy[idx] = arr[idx];

    Arrays.sort(copy, (a, b) -> a - b); // Increasing Order
    for (int val : copy)
        System.out.print(val + " ");
    System.out.println();

    Arrays.sort(copy, (a, b) -> b - a); // Decreasing Order
    for (int val : copy)
        System.out.print(val + " ");
    System.out.println();
}
```

Integer

Integer

```java
class Student{
    int phy, chem, maths;
    Student(int phy, int chem, int maths){
        this.phy = phy;
        this.chem = chem;
        this.maths = maths;
    }
}
public void customSort (int phy[], int chem[], int math[], int N)
{
    Student[] stud = new Student[N];
    for(int idx = 0; idx < N; idx++){
        stud[idx] = new Student(phy[idx], chem[idx], math[idx]);
    }

    Arrays.sort(stud, (a, b) -> {
        if(a.phy != b.phy) return a.phy - b.phy;
        // Increasing order of physics
        if(a.chem != b.chem) return b.chem - a.chem;
        // Same in Physics, Decreasing Order of Chemisty
        return a.maths - b.maths;
        // Same in Phy & Chem, Increasing order in maths
    });

    for(int idx = 0; idx < N; idx++){
        phy[idx] = stud[idx].phy;
        chem[idx] = stud[idx].chem;
        math[idx] = stud[idx].maths;
    }
}
```

Students Type

```java
String[] names = { "Guneet", "Vrushabh", "Chinmay", "Raghav", "Hardik", "Archit" };
Arrays.sort(names);
for (String val : names)
    System.out.print(val + " ");
System.out.println();
```

Archit < Chinmay < Guneet < Hardik < Raghav < Vrushabh → Lexicographically increasing order

Lexicographical:

- Chinmay < Chirag
- Chin < Chinmay

"g21211111" < "g2129"

"g212" < "g2120000"

# Largest No from Array

52 90 6 92 5 97 50 9 59 927

"9 97 92 927 90 6 595 52 50"

92 97

59 5

9 + 97 > 97 + 9

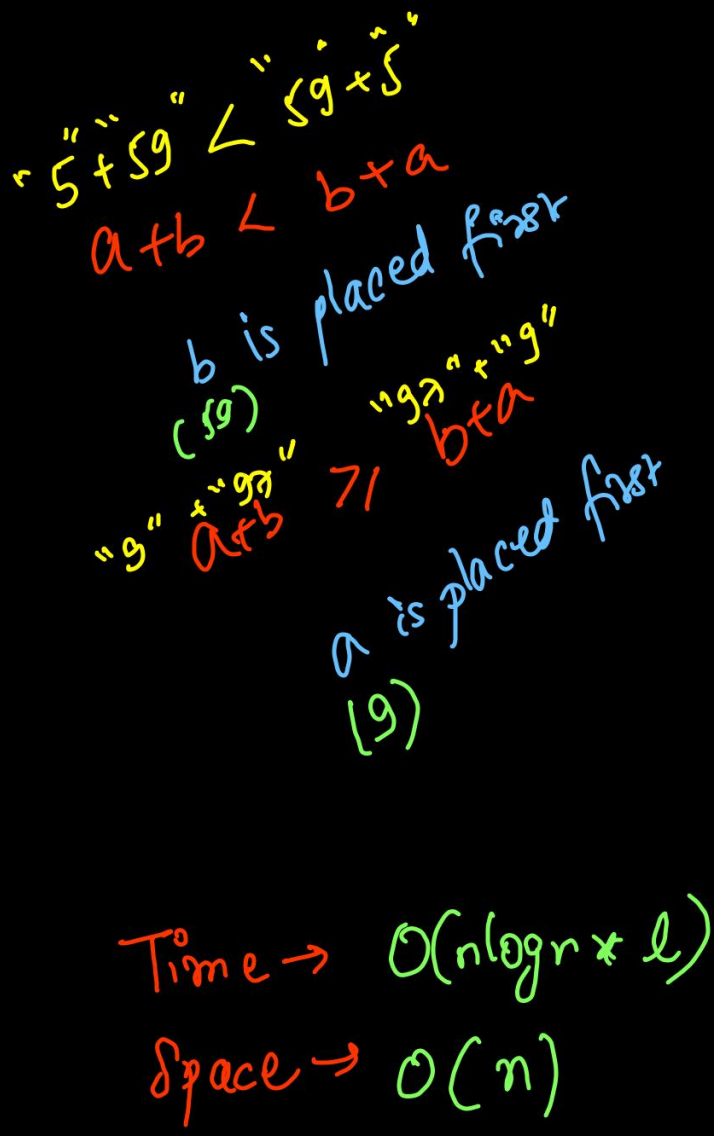
59 + 5 > 5 + 59

```java
public String largestNumber(int[] arr) {
    String[] copy = new String[arr.length];
    boolean allZero = true;
    for(int idx = 0; idx < arr.length; idx++)
    {
        if(arr[idx] != 0) allZero = false;
        copy[idx] = Integer.toString(arr[idx]);
    }

    if(allZero == true) return "0";

    Arrays.sort(copy, (a, b) -> {
        if((a + b).compareTo(b + a) < 0) return +1;
        return -1;
    });

    StringBuilder res = new StringBuilder("");
    for(String val: copy) res.append(val);
    return res.toString();
}
```




Time → O(n log n * l)

Space → O(n)

```java
class Movie{
    int duration;
    double ratings;
    String name;

    public Movie(int duration, double ratings, String name) {
        this.duration = duration;
        this.ratings = ratings;
        this.name = name;
    }
}
```

Not possible

```java
public static void comparableVSComparator() {
    Movie[] arr = new Movie[5];

    arr[0] = new Movie(duration: 180, ratings: 4.5, name: "Avengers");
    arr[1] = new Movie(duration: 150, ratings: 5.0, name: "Titanic");
    arr[2] = new Movie(duration: 100, ratings: 3.0, name: "Spiderman");
    arr[3] = new Movie(duration: 200, ratings: 5.0, name: "Avatar");
    arr[4] = new Movie(duration: 50, ratings: 1.0, name: "Thor");

    Arrays.sort(arr);
}
```

Runtime Error : How to compare 2 movie objects?

# Approach #2: Using Comparable

```java
class Movie implements Comparable<Movie> {
    int duration;
    double ratings;
    String name;

    public Movie(int duration, double ratings, String name) {
        this.duration = duration;
        this.ratings = ratings;
        this.name = name;
    }

    public int compareTo(Movie other) {
        return this.duration - other.duration;
        // Default Sorting: Based on Increasing Order of Duration
    }

    public String toString() {
        return "Name : " + this.name + " of " + this.duration
                + " Minutes " + " with " + ratings + " ratings";
    }
}
```

```java
Movie[] arr = new Movie[5];

arr[0] = new Movie(duration: 180, ratings: 4.5, name: "Avengers");
arr[1] = new Movie(duration: 150, ratings: 5.0, name: "Titanic");
arr[2] = new Movie(duration: 100, ratings: 3.0, name: "Spiderman");
arr[3] = new Movie(duration: 200, ratings: 5.0, name: "Avatar");
arr[4] = new Movie(duration: 50, ratings: 1.0, name: "Thor");

Arrays.sort(arr);

for (Movie val : arr)
    System.out.println(val);
```

```
architaggarwal@Archits-MacBook-Air Java OOPS % java OOPS_Codes.LambdaExpression
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
```

```java
class MovieDurationIncreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return a.duration - b.duration;
    }
}

class MovieDurationDecreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return b.duration - a.duration;
    }
}

class MovieLexicographicalComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        return a.name.compareTo(b.name);
    }
}

class MovieRatingIncreasingComparator implements Comparator<Movie> {
    public int compare(Movie a, Movie b) {
        if (a.ratings - b.ratings < 0)
            return -1;
        return +1;
    }
}
```

```java
Arrays.sort(arr, new MovieDurationIncreasingComparator());

for (Movie val : arr)
    System.out.println(val);

System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieDurationDecreasingComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieRatingIncreasingComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");

Arrays.sort(arr, new MovieLexicographicalComparator());

for (Movie val : arr)
    System.out.println(val);
System.out.println(x: " ----------  ");
```

Approach ③ Using Comparator

```
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
 ----------
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Thor of 50 Minutes  with 1.0 ratings
 ----------
```

```
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
 ----------
Name : Avatar of 200 Minutes  with 5.0 ratings
Name : Avengers of 180 Minutes  with 4.5 ratings
Name : Spiderman of 100 Minutes  with 3.0 ratings
Name : Thor of 50 Minutes  with 1.0 ratings
Name : Titanic of 150 Minutes  with 5.0 ratings
 ----------
```

→ functional interfaces in Java

eg Runnable, Comparable, Comparator, etc

↳ Anonymous Classes / Two name

Interface with only one abstract function

```
interface Operation {
    int operation(int a, int b);
}
```

```
Operation sum = (a, b) -> a + b;
Operation prod = (a, b) -> a * b;
Operation sub = (a, b) -> a - b;
```

} Implementing Interface or overriding function in interface

```
private int operate(int a, int b, Operation op) {
    return op.operation(a, b);
}
```

```
LambdaFunctions myCalculator = new LambdaFunctions();
System.out.println(myCalculator.operate( a: 5,  b: 3, sum));
System.out.println(myCalculator.operate( a: 5,  b: 3, prod));
System.out.println(myCalculator.operate( a: 5,  b: 3, sub));
```

@ Functional Interface

```java
interface Operation {
    public int applyOp(int a, int b);
}

class Sum implements Operation {
    public int applyOp(int a, int b) {
        return a + b;
    }
}

class Difference implements Operation {
    public int applyOp(int a, int b) {
        return a - b;
    }
}

class Product implements Operation {
    public int applyOp(int a, int b) {
        return a * b;
    }
}

class Division implements Operation {
    public int applyOp(int a, int b) {
        return a / b;
    }
}
```

```java
Operation obj = new Sum();
System.out.println(obj.applyOp(a: 15, b: 10));

Operation obj2 = new Difference();
System.out.println(obj2.applyOp(a: 15, b: 10));

Operation obj3 = new Product();
System.out.println(obj3.applyOp(a: 15, b: 10));

Operation obj4 = new Division();
System.out.println(obj4.applyOp(a: 15, b: 10));
```

→ 25
→ 5
→ 150
→ 1

OR

```java
Operation sum = (a, b) -> a + b;
Operation diff = (a, b) -> a - b;
Operation prod = (a, b) -> a * b;
Operation div = (a, b) -> a / b;

System.out.println(sum.applyOp(a: 15, b: 10));
System.out.println(diff.applyOp(a: 15, b: 10));
System.out.println(prod.applyOp(a: 15, b: 10));
System.out.println(div.applyOp(a: 15, b: 10));
```