Learning Goals

Click to Continue

This session will help you to understand about,

- What is Java functional programming?
- Lambda functions.
- Functional interfaces.

Click to Continue

Assume we need a method to add two numbers. In java we do the following,

```
public class Calculator {
public int add(int a, int b) {
int c = a + b;
return c;
}
}
```

To invoke this method we need to create the instance of the object as below,

```
Calculator c = new Calculator();
c.add(10, 20);
```

This is the actual method. But to develop a small function in Java we need to develop a class.

Try it out - Functional Interface

Click to Continue

A quick recap: A interface can have one or more abstract methods.

Functional interfaces are interfaces with just one abstract methods and annotated as **@FunctionalInterface.**

NOTE: The annotation is optional, but if one provides it. Compiler will ensure that only one abstract method is added else a compilation error is thrown.

Assume there is a interface ICalculator with a method

```
public interface ICalculator
{

public void add();
}
```

To implement this one has to write a class which implements the ICalculator as below,

```
class CalculatorImpl implements
ICalculator {
public void add() {
        //logic goes here
}
}
```

Assume a class Demo need to use this should create instance of the CalculatorImpl and use it.

Try it out - Lambda Expression

Click to Continue

Assume there is a interface ICalculator with a method

```
@FunctionalInterface
public interface ICalculator
{

public void add();
}
```

Here the interface is marked as functional interface to ensure only one abstract method is specified.

Let us see how the method is implemented and invoked without a class.

```
class Demo {
public static void main(String[]
args) {
ICalculator calc = () -> {
System.out.println("Method
logic");
};
calc.add();
}
}
```

You can invoke the method like this.

```
ICalculator calc = () -> {
System.out.println("Method logic");
};
```

This is the lambda expression. Which can directly inject the method logic and assigned to a functional interface reference.

IMPORTANT: Lambda expressions can be used only with functional interfaces.

Try it out - Lambda Syntax

Click to Continue

Lambda expression syntax:

1. No argument lambda expression is as follows,

() -> { logic goes here};

2. Assume that there is a functional interface which accepts one int argument.

The parameter data type need not be specified. Java will consider it based on functional interface method

(n) -> { logic goes here};

This is how the implementation looks like.

```
@FunctionalInterface
public interface ICalculator
{

public void add(int a);
```

```
ICalculator calc = (n) -> {
System.out.println("n
value:"+n);
};
calc(10);
```

Implement this in the Demo Class main method as mentioned in the previous slide.

Try it out - Lambda Syntax

Click to Continue

3. Assume that there is a functional interface method which accepts two argument a double and a int argument.

(n,m) -> { logic goes here};

This is how the implementation looks like.

```
@FunctionalInterface
public interface ICalculator {

public void add(int a,double f);
}
```

```
ICalculator calc = (n,m) -> {
System.out.println("n
value:"+n*m);
};
calc(10,20.5);
```

Multiply two values and print.

Implement this in the Demo Class main method as mentioned in the previous slide.

Try it out - Lambda Syntax

Click to Continue

4. Functional interface method which returns a value. Lambda expression with multiple statements

This is how the implementation looks like.

```
@FunctionalInterface
public interface ICalculator {

public double add(int a,double f);
}
```

```
class Demo {
public static void main(String[] args) {
ICalculator calc = (n, m) -> {
double result = n * m;
return result;
};
double op = calc.add(10, 20.5);
System.out.println("result:" + op);
}
}
```

Lambda expression with multiple statements. It also returns a value

Return value can be accessed like this.

Try it out - Lambda expression as method parameter

Click to Continue

Java has introduced lots of methods for which a lambda expression can be passed.

Assume there is a list of fruits and we need to iterate. List provides API ***for each*** method let us see how it is implemented.

Here the lambda expression is passed as parameter to for each method.

```
List<String> list = new
ArrayList<>();
list.add("apple");
list.add("orange");
list.add("grapes");
list.add("Pineapple");
```

```
list.forEach((n) -> {
char c = n.charAt(0);
System.out.print(c);
});
```

The method would print the first character of each fruit.

Advantages of Lambda expression **Click to Continue**

- Easy to implement interfaces.
  - Lambda expression can be used to implement interfaces easily.
- Reduced lines of code.
  - Since no classes are required to be developed for functions , the lines of code is reduced.
- Cleaner and easy to understand code.
  - Less code makes it easy to develop and maintain.
- Passing behaviours into methods.
  - We can also pass lambda expressions as method parameters , thus telling the invoked method on what logic to be performed.