# Java Programming
## Lab FP 2

# Function arguments

## 1. Lab objectives

This initial lab is designed to help get started with the basics of using functions as parameters to other functions.

## 2. Lab Setup

You can reuse the last lab you did or start a new project from scratch the same way that you did in the last lab by defining a class with a main() method.

## 3. Lab Solutions

Since the demos are based on the labs, if you are having problems getting started, refer to the demo code in the repository to give you some hints.

## 4. Create Some Functions.

Create some functions to pass as parameters. Typically, the sorts of functions that tend to pass are brief snippets of logic. We create more complex functions by assembling smaller bits of functional code. The bit of functional code are normally Lambda function which can be written in one line or two. This is often called a UNIX architecture because this is how the UNIX OS was designed, complex operations are created by using small single function utilities, like grep, which are chained together using pipes.

Create four variables to hold functions of different types:

1. A variable called **square** that holds a function that takes and Integer argument and returns an integer
2. A variable called **isEven** that holds a predicate that takes an Integer argument.
3. A variable called **sum** that takes two Integers and arguments and returns an Integer.
4. A variable called **dsquare** that takes an Integer and returns the square but as a Double.

```
// Define a few functions
public static Function<Integer,Integer> identity = x -> x;
public static Function<Integer,Integer> square = x -> x * x;
public static Function<Integer,Integer> cube = x -> x * x * x;
// Note that this has a different type than the preceding three functions
public static Function<Integer,Double> dsquare = x ->  Double.valueOf(x * x);
```

## 5. Create a meta function

A meta-function is a function that can take on different forms at run time. This is a bit different that the process that Java uses for generics to create something like a generic function. For generics, the generic is expanded at compile time, not at run time.

Meta-functions typically modify their code during execution. Many languages, like LISP, have very sophisticated met-programming systems, but be can emulate some of this functionality using functional programming.

Our meta-function is called **applyFunc(..)** and take a data item, in this case an integer, and performs some transformation to it by applying some function to that data and then returning the data. What transformation is applied depends on what function is passed when the code is executing.

```
// returns the value of applying function f to the integer x
private static int applyFunc(int x, Function<Integer,Integer> f) {
    return f.apply(x);
}
```

## 6. Test the function

Now call the meta-function using the different functions you have defined earlier. You can also experiment with passing a Lambda function directly.

```
public static void main(String[] args) {
    System.out.println("Identity for 3 is " + applyFunc(3,identity));
    System.out.println("Square of 3 is " + applyFunc(3,square));
    System.out.println("Cube of 3 is " + applyFunc(3,cube));
    // Passing a Lambda function
    System.out.println("Adding 1 to 78 is " + applyFunc(78,(x) -> x + 1));
    // Note that the following will not compile
    // Type mismatch in the argument
    // System.out.println("Square of 3 is " + applyFunc(3,dsquare));

}
```

# End of Lab