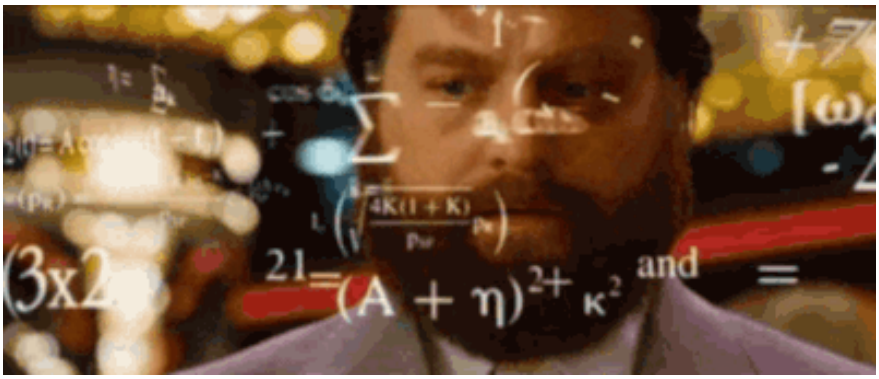
Applause from Cory Althoff, Tariq Ellis, and 6 others

Adrian D. Finlay
@thewiprogrammer. Writer @hackernoon. Code, LOTS of it. Mangos, LOVE THEM! Barbering. Health. Travel. Business. & more! Network w/ me @ adriandavid.me/network
Nov 12, 2017 · 4 min read

Lambda Expressions in Java, Python, C#, C++



GfyCat

No, Lambda Expressions are not as complicated as Church's Lambda Calculus. No need to worry.

> *Lambda Expressions are simply an expression of an anonymous function. This is a very important concept in Functional Programming and is something like a literal expression for a function (such as how 1020L may be interpreted as a long integer literal in certain languages). This is a widespread, cornerstone concept in Functional Programming that has been added to OOP/Procedural centered languages like Java to reap some of the benefits of functional programming, which is useful for things like processing data. It also provides neat, concise syntax for implementing*

> *functions on the fly. They are based of off functional programming concepts used in the construction of expression trees.*

In this article, I will demonstrate basic use among some common, popular languages. Instead of engaging in a tutorial of lambda expressions in each language, I will instead simply show the same example in various languages. Note also that I will not cover a language's full coverage of Lambda Expressions (or the equivalent) rather, some basic use to get your feet wet.

I decided to piggyback off of Quincy Larson's recommendation to use Medium's **code blocks** but because they are visually unappealing to me and **an utter nuisance to indent**, I will not be using them again. This is, of course, not the fault of Quincy Larson. I have other small issues with this format but it is an OK start for small snippets. I will perhaps, as Larson also recommends, resort to GitHub gists.

Let's start with Java.

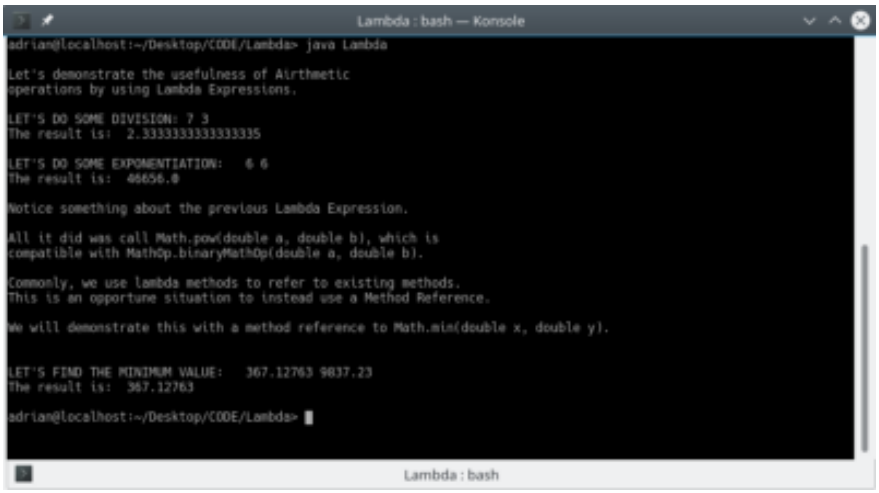
Java

Java provides support for Lambda Expressions via Functional Interfaces. Let's take a peek. Do note that because of my qualms with the format, I have not included the full source below. Look towards the end of the article for the full source.

```
import java.util.Scanner; //For Scanner class
import java.lang.Math; //For pow(), min()
import java.util.InputMismatchException;
//InputMismatchException class
import static java.lang.System.out; //For println(),
print()


//Defines Structure for Lambda Expression
@FunctionalInterface
interface MathOp {
//Must contain one and only one abstract method
       double binaryMathOp(double a, double b);
}


//Main Class
public class Lambda {
```

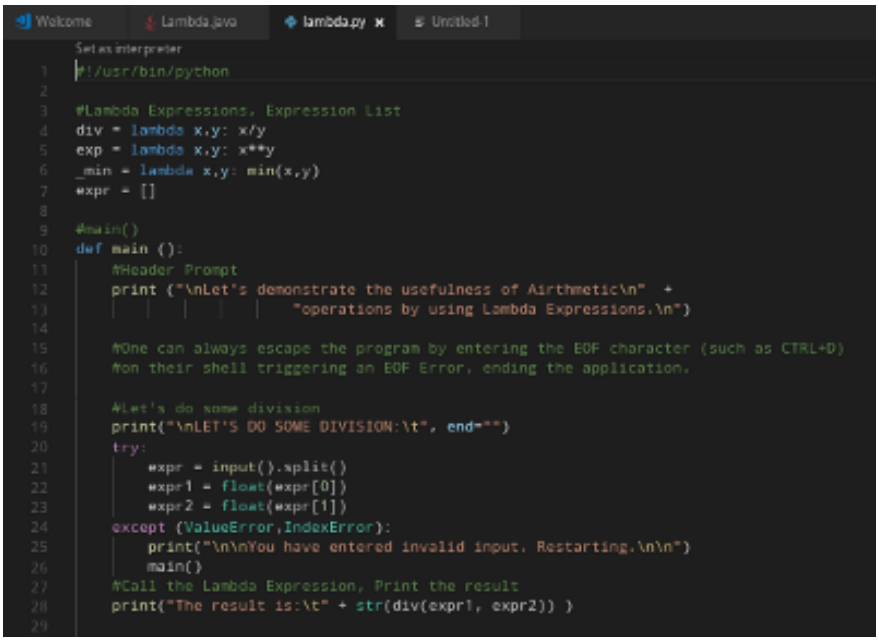
```
    //Main method
    public static void main (String[] args) {


    //Input Stream
    Scanner in = new Scanner(System.in);


    //Lambda Reference, Arithmetic Expressions
    MathOp doMath;
    double expr1=0, expr2 =0;


    //Header
    out.println("\nLet's demonstrate the usefulness of
Airthmetic\n"  +
                  "operations by using Lambda
Expressions.\n");


    //Prompt for Division
    out.print("LET'S DO SOME DIVISION:\t");
    try{
         expr1 = in.nextDouble();
         expr2 = in.nextDouble();
    } catch (InputMismatchException e) {
         out.println("\n\nYou have entered invalid
input." +
                        "Restarting.\n\n");
         main(args);
    }
    //Lambda Expression for Division
    doMath = (x,y) -> {  return x / y; };

    //Print Result
    out.println("The result is:\t" +
doMath.binaryMathOp(expr1, expr2));



    //Method References


    // .... <Omitted>


    //Prompt to find minimum value
    out.print("\nLET'S FIND THE MINIMUM VALUE:\t");


    try{
          expr1 = in.nextDouble();
          expr2 = in.nextDouble();
    } catch (InputMismatchException e) {
          out.println("\n\nYou have entered invalid
input." +
                       "Restarting.\n\n");
          main(args);
    }
    //Method Reference (Lambda Expression)
    doMath = Math::min;
```

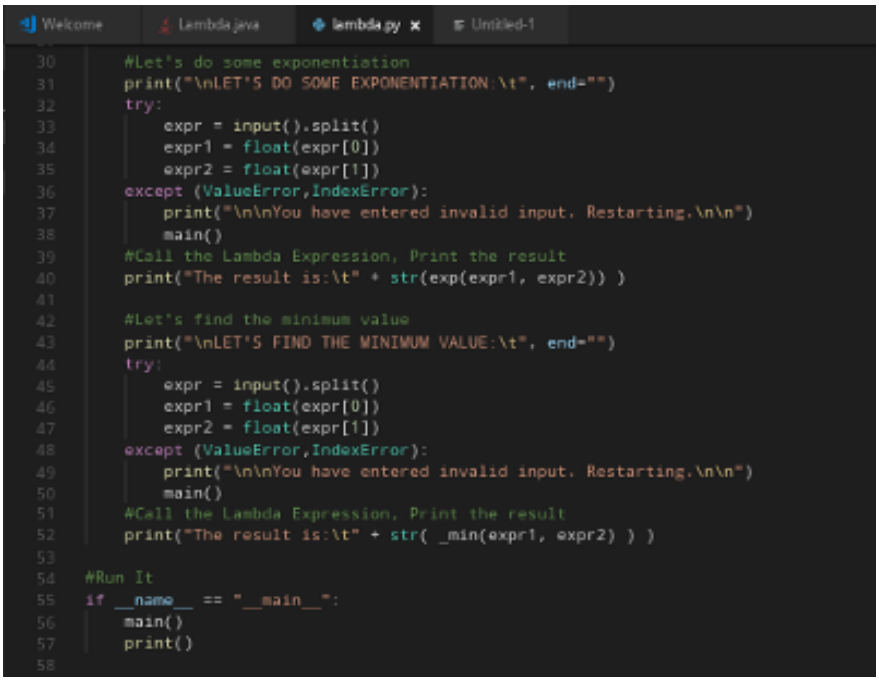
```
      //Print Result
      out.println("The result is:\t" +
  doMath.binaryMathOp(expr1, expr2) + "\n");


     //Close Input Stream
     in.close();


    }
  }
```

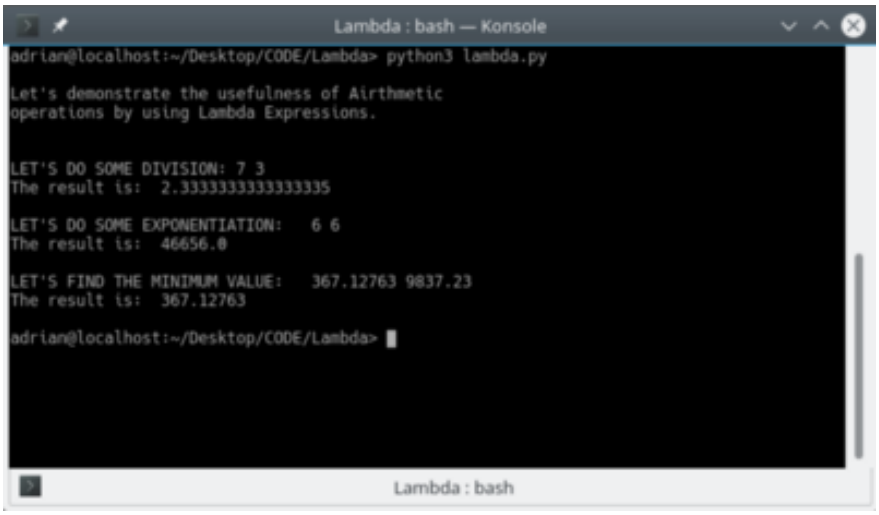


Python

Lambda Expressions work a little differently in python. In python, Lambda Expressions are essentially anonymous functions. Also, note that unlike Java, we can define it's behavior on the fly in the same way.

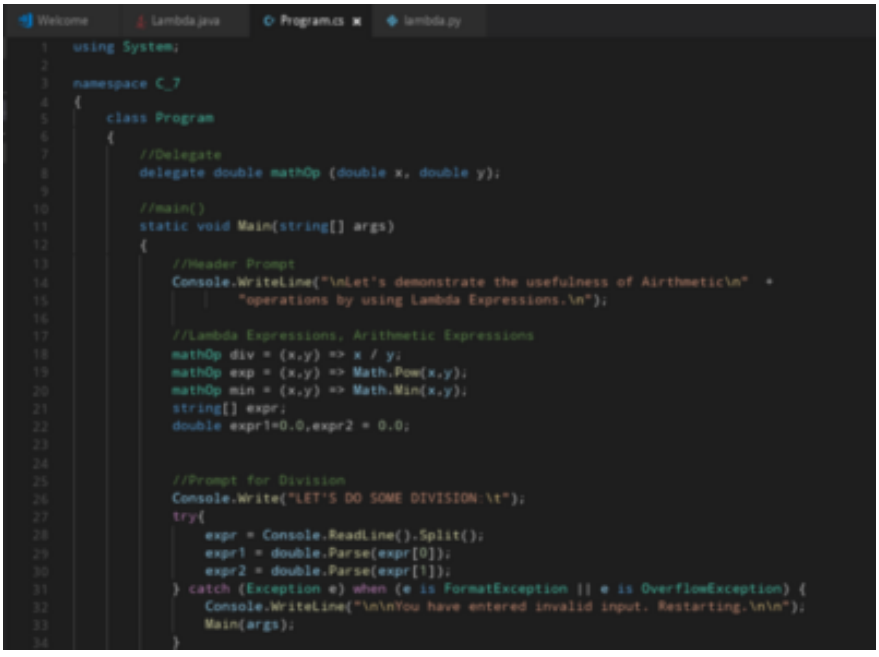

Welcome
Lambda.java
lambda.py
Untitled-1
Set as interpreter
#!/usr/bin/python
#Lambda Expressions, Expression List
div = lambda x,y: x/y
exp = lambda x,y: x**y
_min = lambda x,y: min(x,y)
expr = []
#main()
def main ():
#Header Prompt
print ("\nLet's demonstrate the usefulness of Airthmetic\n" +
"operations by using Lambda Expressions.\n")
#One can always escape the program by entering the EOF character (such as CTRL+D)
#on their shell triggering an EOF Error, ending the application.
#Let's do some division
print("\nLET'S DO SOME DIVISION:\t", end="")
try:
expr = input().split()
expr1 = float(expr[0])
expr2 = float(expr[1])
except (ValueError,IndexError):
print("\n\nYou have entered invalid input. Restarting.\n\n")
main()
#Call the Lambda Expression, Print the result
print("The result is:\t" + str(div(expr1, expr2)))

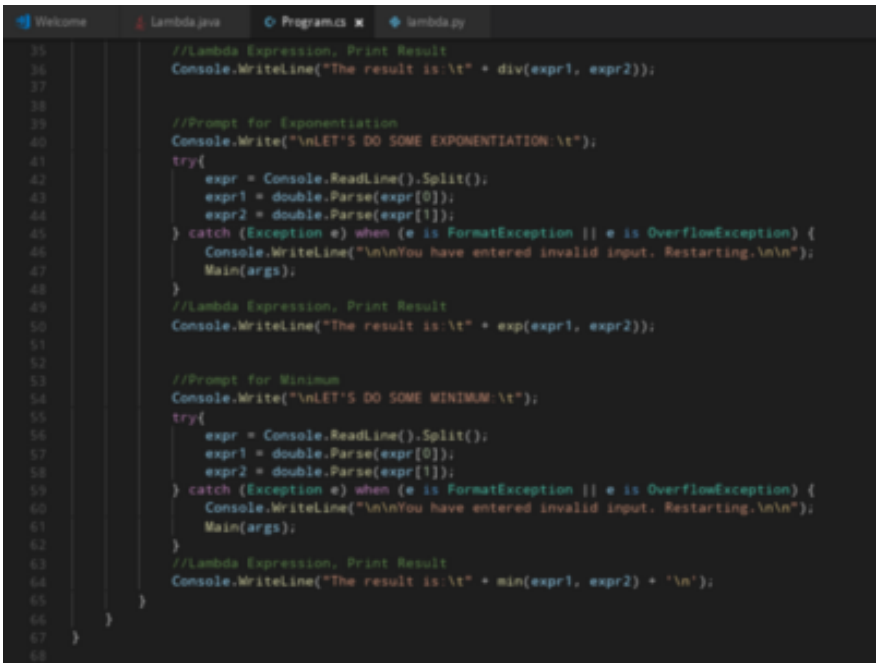

Welcome
Lambda.java
lambda.py
Untitled-1
#Let's do some exponentiation
print("\nLET'S DO SOME EXPONENTIATION:\t", end="")
try:
expr = input().split()
expr1 = float(expr[0])
expr2 = float(expr[1])
except (ValueError,IndexError):
print("\n\nYou have entered invalid input. Restarting.\n\n")
main()
#Call the Lambda Expression, Print the result
print("The result is:\t" + str(exp(expr1, expr2)))
#Let's find the minimum value
print("\nLET'S FIND THE MINIMUM VALUE:\t", end="")
try:
expr = input().split()
expr1 = float(expr[0])
expr2 = float(expr[1])
except (ValueError,IndexError):
print("\n\nYou have entered invalid input. Restarting.\n\n")
main()
#Call the Lambda Expression, Print the result
print("The result is:\t" + str(_min(expr1, expr2)))
#Run It
if __name__ == "__main__":
main()
print()

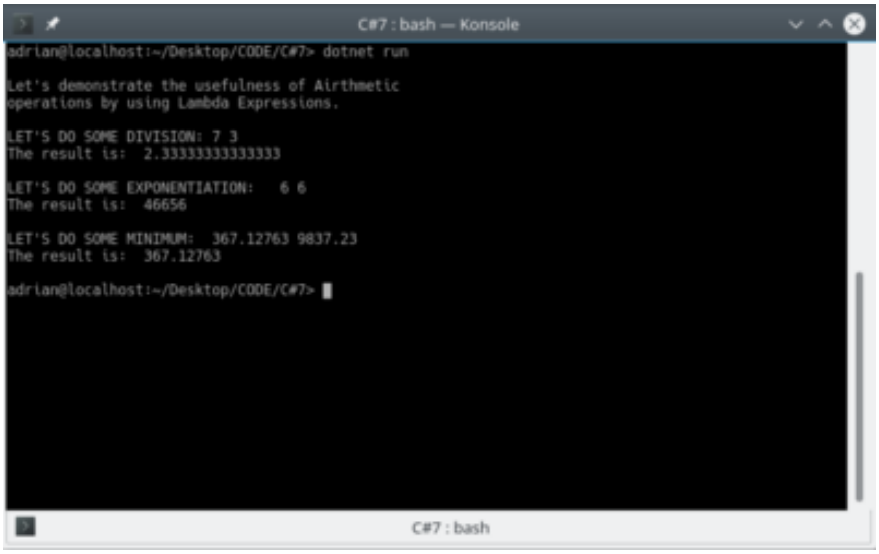


C#

In C#, Lambda Expressions are anonymous functions that can be used to create expression trees or delegates. We will discuss creating a delegate, which is something like a function pointer. This is much closer to the nature of a Lambda Expression than that of Java and Python.

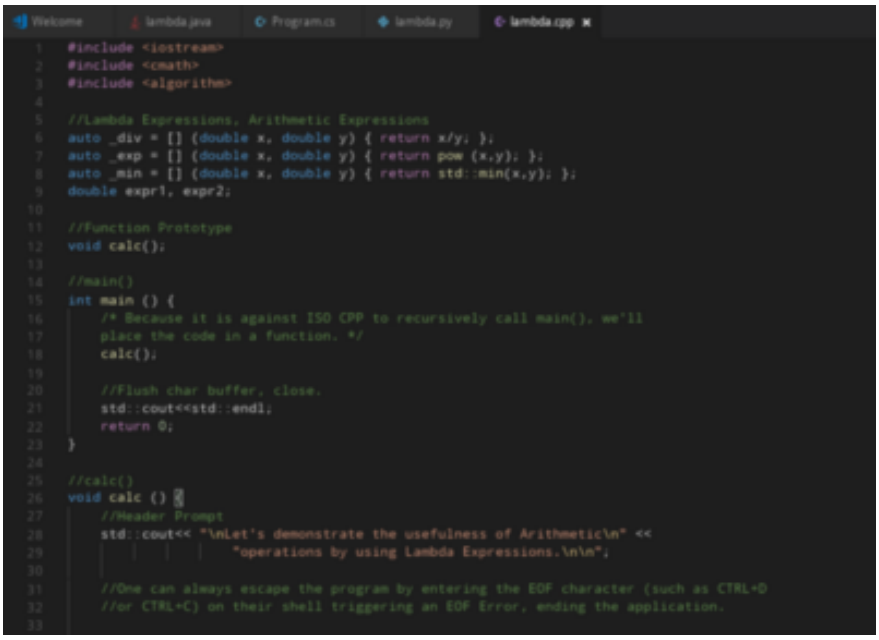




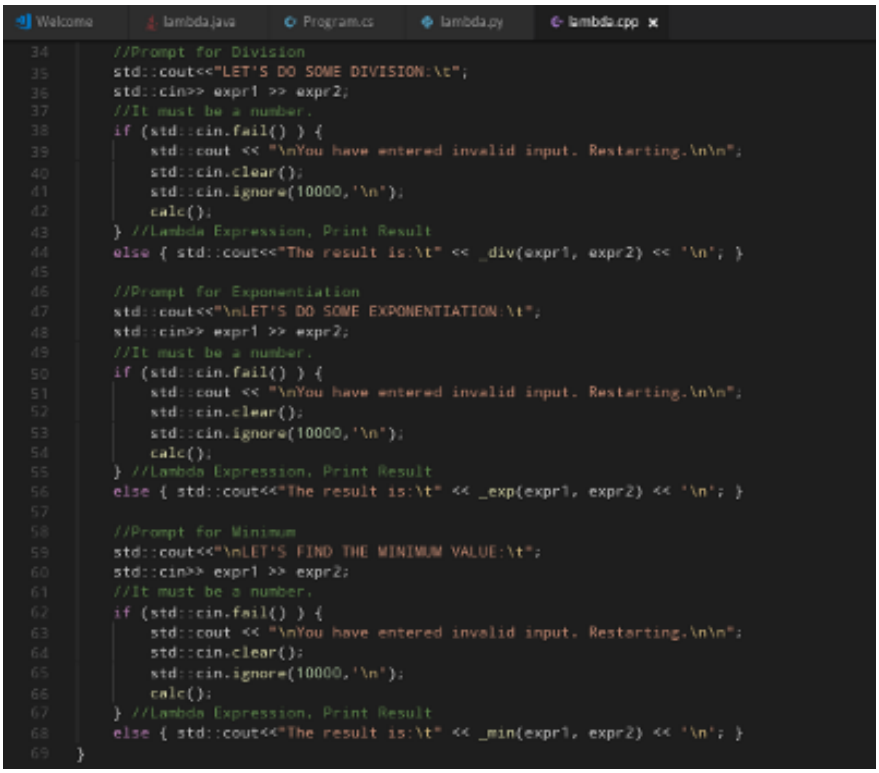


C++

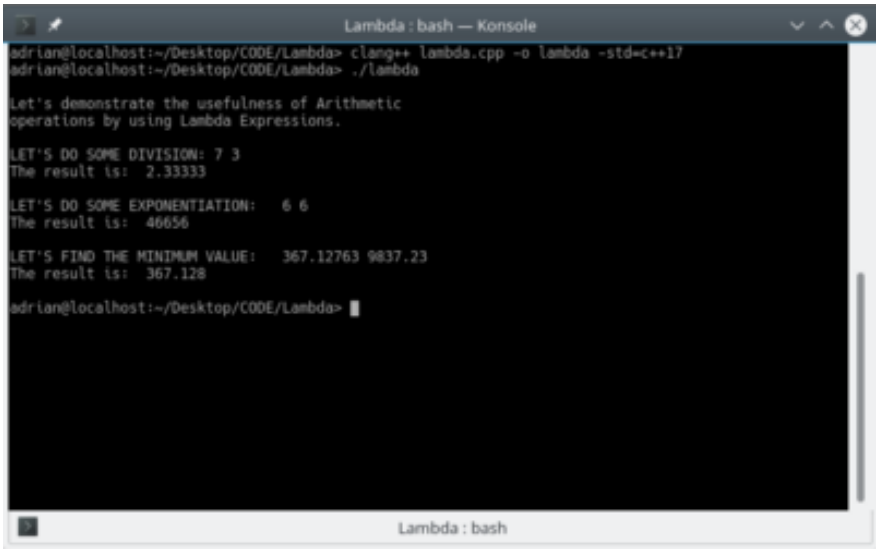
In C++, Lambda Expressions are **closures**, that is, they are an unnamed function object. You can use this function on the fly.



```cpp
#include <iostream>
#include <cmath>
#include <algorithm>

//Lambda Expressions, Arithmetic Expressions
auto _div = [] (double x, double y) { return x/y; };
auto _exp = [] (double x, double y) { return pow (x,y); };
auto _min = [] (double x, double y) { return std::min(x,y); };
double expr1, expr2;

//Function Prototype
void calc();

//main()
int main () {
    /* Because it is against ISO CPP to recursively call main(), we'll
    place the code in a function. */
    calc();

    //Flush char buffer, close.
    std::cout<<std::endl;
    return 0;
}

//calc()
void calc () {
    //Header Prompt
    std::cout<< "\nLet's demonstrate the usefulness of Arithmetic\n" <<
                "operations by using Lambda Expressions.\n\n";

    //One can always escape the program by entering the EOF character (such as CTRL+D
    //or CTRL+C) on their shell triggering an EOF Error, ending the application.
```



```cpp
    //Prompt for Division
    std::cout<<"LET'S DO SOME DIVISION:\t";
    std::cin>> expr1 >> expr2;
    //It must be a number.
    if (std::cin.fail() ) {
        std::cout << "\nYou have entered invalid input. Restarting.\n\n";
        std::cin.clear();
        std::cin.ignore(10000,'\n');
        calc();
    } //Lambda Expression, Print Result
    else { std::cout<<"The result is:\t" << _div(expr1, expr2) << '\n'; }

    //Prompt for Exponentiation
    std::cout<<"\nLET'S DO SOME EXPONENTIATION:\t";
    std::cin>> expr1 >> expr2;
    //It must be a number.
    if (std::cin.fail() ) {
        std::cout << "\nYou have entered invalid input. Restarting.\n\n";
        std::cin.clear();
        std::cin.ignore(10000,'\n');
        calc();
    } //Lambda Expression, Print Result
    else { std::cout<<"The result is:\t" << _exp(expr1, expr2) << '\n'; }

    //Prompt for Minimum
    std::cout<<"\nLET'S FIND THE MINIMUM VALUE:\t";
    std::cin>> expr1 >> expr2;
    //It must be a number.
    if (std::cin.fail() ) {
        std::cout << "\nYou have entered invalid input. Restarting.\n\n";
        std::cin.clear();
        std::cin.ignore(10000,'\n');
        calc();
    } //Lambda Expression, Print Result
    else { std::cout<<"The result is:\t" << _min(expr1, expr2) << '\n'; }
}
```



Want the source? Grab it here.

afinlay5/LambdaExpr

Gradle source code repository for C++, Java, C#, Python source code examples posted on persona...

github.com

Like Lambdas? Let me know in the comments below :)

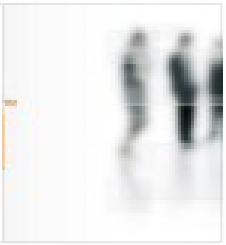
Looney Tunes Ending [4]

Interested in Java? Join my Java group on Facebook:

Join My Java Facebook Group

Interested in Java? Check out my Facebook Group: Java Software Development Group!

medium.com



Like my Content? Subscribe to my mailing list:

Don’t forget to give it a…. ;)

IEmoji.com