Functions in Programming

- a *subroutine* associates a set of statements with a name
  - may have one or more *parameters*
  - may have a *return value*
- subroutines with a return value are called *functions*

  though this terminology is used sloppily – often any subroutine is called a function, and in object-oriented languages like Java, any operation defined on an object is called a *method*, whether or not it returns a value

- the *header* (or *prototype*) of a function defines its parameters and return value, including types

```
int square ( int n ) {
  return n*n;
}
```

square: *int* → *int*

```
int mult ( int a, int b ) {
  return a*b;
}
```

mult: *int* × *int* → *int*

Functions in Programming

- functions in programming are often not true mathematical functions
  - many functions in programming are really *partial functions* which map a subset of *A* to *B*

```
// n >= 1
int square ( int n ) {
  return n*n;
}
```

square: *int* → *int*

  - functions in programming don't always return only a single value for particular parameter values

```
int random ( int n ) {
  …
}
```

random: *int* → *int*

First-Class Functions

- in mathematics, the elements in a set can be anything – including other sets, ordered pairs, and functions

- some programming languages support *first-class functions* where
  - a function can be passed as a parameter to a function
  - a function can be returned from a function
  - a function can be assigned to a variable and used later

  just like any other type

Applications of First-Class Functions

- e.g. define a function corresponding to the Σ operator

```
sum ( f, a, b ) {
  total = 0;
  for ( i = a ; i <= b ; i++ ) {
    total += f(i);
  }
  return total;
}
```

$$\sum_{i=a}^{b} f(i)$$

then

```
sum( function(n) { return n*n; }, 1, 100 )
```

```
square = function(n) { return n*n; }
```

```
sum(square,1,100)
```

First-Class Functions in JavaScript

```
// Functions as values of a variable
var cube = function (x) {
  return Math.pow(x, 3);
};
var cuberoot = function (x) {
  return Math.pow(x, 1 / 3);
};

// Higher order function
var compose = function (f, g) {
  return function (x) {
    return f(g(x));
  };
};

// Storing functions in a array
var fun = [Math.sin, Math.cos, cube];
var inv = [Math.asin, Math.acos, cuberoot];

for (var i = 0; i < 3; i++) {
  // Applying the composition to 0.5
  console.log(compose(inv[i], fun[i])(0.5));
}
```

First-Class Functions (More or Less) in Java

```
public class Test {
    public static void main(String[] args) {
        Comparator<String> comparator = new StringLengthComparator();
        PriorityQueue<String> queue = new PriorityQueue<String>(10, comparator);
        queue.add("short");
        queue.add("very long indeed");
        queue.add("medium");
        while (queue.size() != 0) {
            System.out.println(queue.remove());
        }
    }
}

// StringLengthComparator.java
import java.util.Comparator;

public class StringLengthComparator implements Comparator<String> {
    @Override
    public int compare(String x, String y) {
        // Assume neither string is null. Real code should
        // probably be more robust
        // You could also just return x.length() - y.length(),
        // which would be more efficient.
        if (x.length() < y.length()) {
            return -1;
        }
        if (x.length() > y.length()) {
            return 1;
        }
        return 0;
    }
}
```

can be approximated
through functional interfaces

First-Class Functions (More or Less) in Java

- newer versions of Java support lambda expressions
  - can be subroutines – they don’t have to be true functions

```
helloButton.setOnAction( evt -> message.setText("Hello World!") );
```

```
canvas.setOnMousePressed( evt -> {
    GraphicsContext g = canvas.getGraphicsContext2D();
    if ( evt.isShiftDown() ) {
        g.setFill( Color.BLUE );
        g.fillOval( evt.getX() - 30, evt.getY() - 15, 60, 30 )
    }
    else {
        g.setFill( Color.RED );
        g.fillRect( evt.getX() - 30, evt.getY() - 15, 60, 30 );
    }
} );
```

and define a generic functional interface `Function`

First-Class Functions (More or Less) in Java

```
import java.util.ArrayList;
import java.util.function.Function;

public class FirstClass{

    public static void main(String... arguments){
        ArrayList<Function<Double, Double>> functions = new ArrayList<>();

        functions.add(Math::cos);
        functions.add(Math::tan);
        functions.add(x -> x * x);

        ArrayList<Function<Double, Double>> inverse = new ArrayList<>();

        inverse.add(Math::acos);
        inverse.add(Math::atan);
        inverse.add(Math::sqrt);
        System.out.println("Compositions:");
        for (int i = 0; i < functions.size(); i++){
            System.out.println(functions.get(i).compose(inverse.get(i)).apply(0.5));
        }
        System.out.println("Hard-coded compositions:");
        System.out.println(Math.cos(Math.acos(0.5)));
        System.out.println(Math.tan(Math.atan(0.5)));
        System.out.println(Math.pow(Math.sqrt(0.5), 2));
    }
}
```