KOTLIN

- Variables
  - Mutable vs. immutable (aka “read only”)
    - Immutable - val name = “Matt”
    - Mutable - var myAge = 31

- Kotlin uses type inference - knows variable type
- You can declare type if you want
  - var bigInt : Int = Int.MAX_VALUE
    - println(“Biggest Int : “ + bigInt)
  - var smallInt : Int = Int.MIN_VALUE
    - println(“Smallest Int : $smallInt”) <— String interpolation

- Has same data types as Java, but no semicolons

```
-if (true is Boolean) {
	print(“true is boolean\n”)
}
```

- var letterGrade : Char = ‘A’
  - println(“A is a Char: ${letterGrade is Char}”)
    - This prints out: “A is a Char: true”

- Convert Double to Integer
  - println(“3.14 to Int: “ + (3.14.toInt()))
    - “3.13 to Int: 3”

- Convert Character to Integer
  - println(“A to Int: “ + (‘A’.toInt()))
    - “A to Int: 65”

- Convert Integer to Character
  - println(“65 to Char: “ + 65.toChar())
    - “65 to Char: A”

STRINGS

- val myName = “Matt”
- val longString = “”” This is a
Long string”””

```
-var fName : String = “Doug”
-var lName : String = “Smith”
-fName = “Sally”
-var fullName = fName + “ “ + lName
-println(“Name : $fullName”)

-println(“1 + 2 = ${1 + 2}”)
-println(“String Length: ${longStr.length}”)

-var str1 = “A random string”
-var str2 = “a random string”
-println(“Strings Equal: ${str1.equals(str2)}”)

-println(“Compare A to B: ${“A”.compareTo(“B”)}”)
-println(“2nd Index: ${str1[2]}”) //This gets the second index
-println(“Index 2-7: ${str.subSequence(2,8)}”)
-println(“Contains random: ${str1.contains(“random”)}”)
```

ARRAYS

```
-var myArray = arrayOf(1, 1.23, “Doug”)
-println(myArray[2])
-myArray[1] = 3.14
-println(“Array Length : ${myArray.size}”)
-println(“Doug in Array : ${myArray.contains(“Doug”)}”)
-var partArray = myArray.copyOfRange(0, 1)
        -println(“First : ${myArray.first()}”)
-println(“Doug Index : ${myArray.indexOf(“Doug”)}”)

-var sqArray = Array(5, {x -> x*x})

-var arr2: Array<Int> = arrayOf(1,2,3)
```

RANGES

```
-val oneTo10 = 1..10  //prints range 1-10
-val alpha = “A”..”Z” //prints range A-Z
-println(“R in Alpha : ${“R” in alpha}”) //Checks if R is in alpha
-val tenTo1 = 10.downTo(1) //Creates a range that decrements
-val twoTo20 = 2.rangeTo(20) //Creates an array from 2 to 20
-val range3 = OneTo10.step(3) //Steps through a range adding 3 to each
-for(x in rng3) println(“rng3 : $x”) //prints all the values
-for(x in tenTo1.reversed()) //reverses a range
```

```
    -println(“Reverse : $x”) //prints out the reversed range
```

CONDITIONALS

-Essentially the same as Java

-When works like switch:

```
    -when(age){
        0,1,2,3,4 -> println(“Go to Preschool”)

        5 -> println(“Go to Kindergarten”)

        in 6..17 -> {
            val grade = age - 5
             println(“Go to Grade $grade”)
        }

        else -> println(“Go to College”)
    }
```

LOOPING

```
-for(x in 1..10){
    println(“Loop : $x”)
} //Loops through a range

-val rand = Random()
-val magicNum = rand.nextInt(50) + 1

-var guess = 0
-while(magicNum != guess) {
    guess += 1
}
println(“Magic Number was $guess”)

-for(x in 1..20){
    if(x % 2 == 0){
        continue
    }
    println(“Odd : $x”)

    if (x == 15) break
    }
```

```
}

-var arr3: Array<Int> = arrayOf(3,6,9)
-for(i in arr3.indices){
	println(Mult 3: ${arr3[i]}”)
}

-for((index, value) in arr3.withIndex()){
	println(“Index : $index Value : $value”)
}
```

FUNCTIONS

-Defining a function always starts with “fun”

```
-fun add(num1: Int, num2: Int) : Int = num1 + num 2
	-println(“5 + 4 = ${add(5,4)}”)
```

-You don’t need a return type when working with single line functions

-You can have default values

```
	-fun subtract(num1: Int = 1, num2: Int = 1) = num1 - num2
		-println(“5-4 = ${subtract(5,4)}”)
```

-Named parameters

```
	-println(“4-5 = ${subtract(num2 = 5 num1 = 4)}”)

-fun sayHello(name: String) : Unit = println(“Hello $name”)
	-//Unit is used for functions that don’t return something, like “void” in Java

-fun nextTwo(num: Int): Pair<Int, Int>{
	return Pair(num+1, num+2)
}

	-val (two, three) = nextTwo(1)
		-println(“1 $two $three”)

-fun getSum(vararg nums: Int): Int //receiving a variable number of arguments
{
	var sum = 0
	nums.forEach {n -> sum += n}
	return sum
}
```

```
        -println(“Sum = ${getSum(1,2,3,4,5)}”)

-val multiply = {num1: Int, num2: Int -> num1 * num2}
        -println(“5 * 3 = ${multiply(5,3)}”)//This is a function literal

-fun fact(x: Int): Int{
        tailrec fun factTail(y: Int, z: Int): Int {
        if(y ==0) return z
        else return factTail(y - 1, y * z)
        }
        return factTail(x, 1)
}

        -println(“5! = ${fact(5)}”) //This is for factorial
```

HIGHER ORDER FUNCTIONS

```
-val numList = 1..20
-val evenList = numList.filter { it % 2 == 0}
-evenList.forEach {n -> println(n)}
```

//If a function has only parameter you don’t have to declare it. You just use “it” instead

```
-fun makeMathFunc(num1: Int): (Int) -> Int = {num2 -> num1 * num2}
```

//This is a function that generates functions

```
-val mult3 = makeMathFunc(3)
        -println(“5 * 3 = ${mult3(5)}”)

-fun mathOnList(numList: Array<Int>, myFunc: (num: Int) -> Int) {
        for(num in numList) {
                println(“MathOnList ${myFunc(num)}”)
        }
}

-val multiply2 = {num1: I nt -> num1 * 2}
-val numList2 = arrayOf(1,2,3,4,5)
-mathOnList(numList2, multiply2)
```

//Returns “MathOnList 2, MathOnList 4, MathOnList 6, MathOnList 8, MathOnList

10”

COLLECTION OPERATORS

//Sum all the values in a list

```
-val numList2 = 1..20
-val listSum = numList2.reduce {x, y -> x + y}
        -println(“Reduce Sum : $listSum”)
        //Returns “Reduce Sum : 210”
```

//Fold is like reduce but starts with an initial value

```
-val listSum2 = numList2.fold(5) {x, y -> x + y}
        -println(“Fold Sum : $listSum2”)
        //Returns “Fold Sum : 215”
```

//Check if any values are able to meet a condition

```
-val numList 2 = 1..20
        -println(“Evens : ${numList2.any {it % 2 == 0}}”)
```

//Check if all values are able to meet a condition

```
-val numList 2 = 1..20
        -println(“Evens : ${numList2.all {it % 2 == 0}}”)
```

//Return a list of values that meet a certain condition

```
-val numList 2 = 1..20
        -val big 3 = numList2.filter {it > 3}
        -big3.forEach {n -> println(“Greater than 3 : $n”)}
```

//Perform an action on every single item and return a new list

```
-val numList2 = 1..20
        -val times7 = numList2.map { it * 7 }
        -times7.forEach {n -> println(“*7 : $n”)}
```

EXCEPTION HANDLING

```
-val divisor = 2
-try {
        if (divisor == 0){
                throw IllegalArgumentException(“Can’t divide by Zero”)
        } else {
                println(“5 / $divisor = ${5/divisor}”)
        }
} catch (e: IllegalArgumentException) {
        println(“${e.message}”)
}
```

LISTS

-There are mutable lists and immutable lists

```
-var list1: MutableList<Int> = mutableListOf(1,2,3,4,5)

-val list2: List<Int> = listOf(1,2,3)

//Add to list
-list1.add(6)

//Get first item in list
-println(“1st : ${list1.first()}”)

//Get last item in list
-println(“Last : ${list1.last()}”)

//Get a value at a specific index
-println(“2nd : ${list1[2]}”)

//Get a list starting from one index to another
-var list3 = list1.subList(0, 3)

//Get the size of a list
-println(“Length : ${list1.size}”)

//Clear a list
-list3.clear()

//Remove a value
-list1.remove(1)
-list1.removeAt(1)
```

//Add at an index
-list1[2] = 10

//Cycle through all the items
-list1.forEach {n -> println(“Mutable List : $n”)}

MAPS

-There are mutable and immutable maps

-val map = mutableMapOf<Int, Any?>() *//The “Any?” means “anything”*

//Loading values into a map when you first create it
-val map2 = mutableMapOf(1 to “Doug”, 2 to 25)

//Add additional values
-map[1] = “Derek”
-map[2] = 42

//Get the size of the map
-println(“Map Size : ${map.size}”)

//Add a key-value
-map.put(3, “Pittsburgh”)

//Remove a key-value
-map.remove(2)

//Iterate and get key values

```
for((x, y) in map) {
	println(“Key : $x Value : $y”)
}
```

CLASSES

-There are no static methods
-Classes are going to be marked as final by default unless they are marked as “open”
-Objects are initialized in an “init” function

```
-open class Animal(val name: String, var height: Double, var weight: Double) {
        init{
                val regex = Regex(“.*\\d+.*”) //Checking to see if there is a decimal/
number value anywhere inside of the string that will be assigned to name
                require(!name.matches(regex)){“Animal name can’t contain numbers”}
                require(heigh > 0) {“Height must be greater than zero”}
                require(weight > 0) {“Weight must be greater than zero”}
        }

        open fun getInfo(): Unit
        {
                println(“$name is $heigh tall and weighs $weight”)
        }
}

-fun main(args: Array<String>) {
        val bowser = Animal(“Bowser”, 20.0, 13.5)
        bowser.getInfo()
}
```

INHERITANCE

```
-class Dog(name: String, height: Double, weight: Double, var owner: String) :
Animal(name, height, weight){

        override fun getInfo(): Unit{
                println(“$name is $heigh tall and weighs $weight and is owned by
$owner”)
        }
}

-fun main(args: Array<String>) {
        val spot = Dog(“Spot”, 20.0, 14.5, “Paul Smith”)
        spot.getInfo()
}
```

INTERFACES

-A contract that states that a class must implement all fields and methods if it implements the interface

```
-interface Flyable{
    var flies: Boolean
    fun fly(distMiles: Double): Unit
}
```

//“Boolean = true” is a default value

```
Class Bird constructor(val name: String, override var flies: Boolean = true) : Flyable{
    override fun fly(distMiles: Double): Unit
    }
        if(flies){
            println(“$name flies $distMiles miles”)
        }
    }
}
```

```
-fun main(args: Array<String>) {
    val tweety = Bird(“Tweety”, true)
    tweety.fly(10.0)
}
```

NULL SAFETY

-Built directly into Kotlin
-By default you cannot assign null

-var nullVal: String = null (This doesn’t work!)

-var nullVal: String? = null (This works)

//A function may return null

```
-fun returnNull(): String? {
    return null
}
```

//Kotlin provides for the opportunity of a null value if an if statement is going to protect it from danger

-var nullVal2 = returnNull()

//This is referred to as a “smart cast”

```
-if(nullVal2 != null){
    println(“nullVal2.length”)
}
```

```
//You can use the force operator to force a null assignment
-var nullVal3 = nullVal2!!.length

//You can use the Elvis operator to assign a default value if the value could be null
-var nullVal4: String = returnNull() ?: “No Name”
```