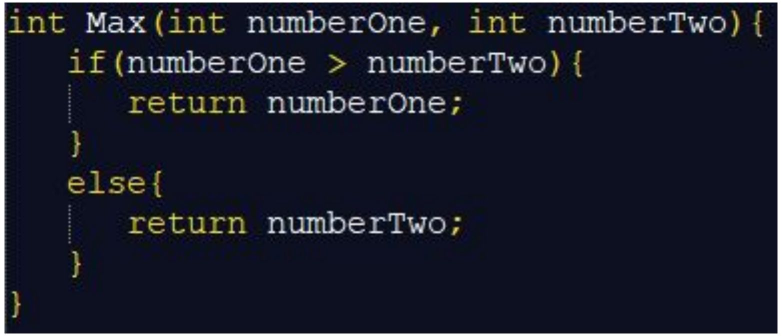
Matts Quick Guide to Functions

For Dan

Introduction

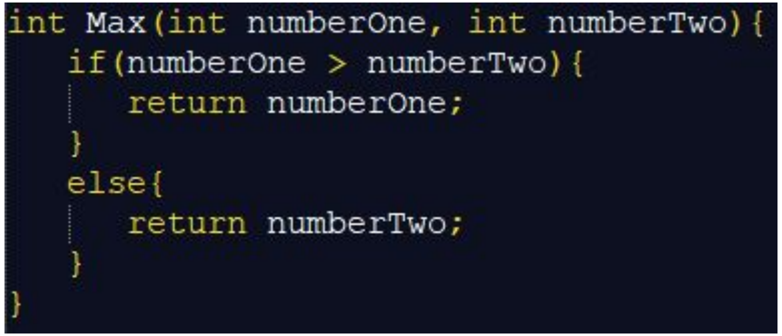
Functions are useful subroutines that can be referenced multiple times in the code. We use functions as a way of saving space by not having to repeat code that would otherwise need to be rewritten multiple times. We will analyse this function for the remainder of this lesson



```
int Max(int numberOne, int numberTwo){
    if(numberOne > numberTwo){
        return numberOne;
    }
    else{
        return numberTwo;
    }
}
```

What This Function Does

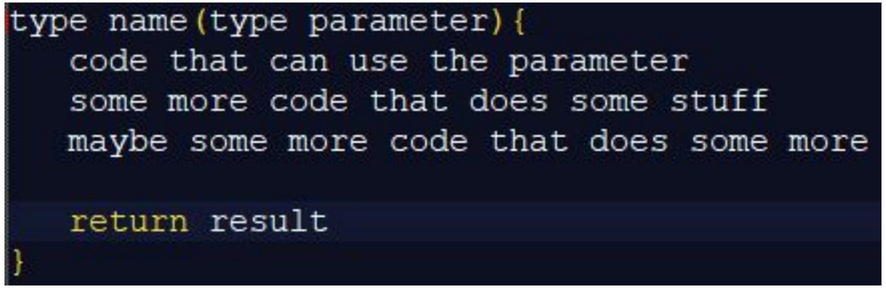
This function takes in two numbers, *numberOne* and *numberTwo*, and returns the number that is larger. More importantly though let's discuss what ANY function does.



```
int Max(int numberOne, int numberTwo){
    if(numberOne > numberTwo){
        return numberOne;
    }
    else{
        return numberTwo;
    }
}
```

The Generalized Function

A function starts with a **Type,** which defines what the expected **Return** result will be. This type can (in Java) be **void, int, float, double, Object** and more. **Object** can be any Java Object, which includes **String** or any class Objects you create. The **Parameter** is a variable you pass to the function to typically evaluate. A function can have any number of **Parameters**. The **Return** statement is what the function will send back to wherever it was called

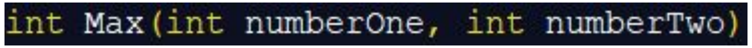


```
type name(type parameter){
   code that can use the parameter
   some more code that does some stuff
   maybe some more code that does some more

   return result
}
```

The Function Header

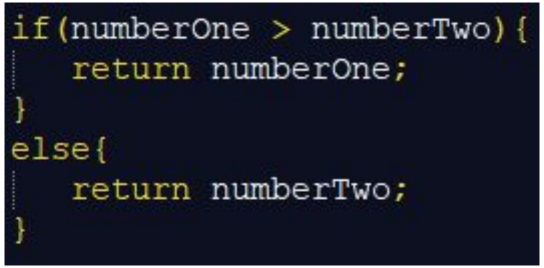
The start of the function is its type **int**. This means the function is expected to **return** an **int**. The function also has two parameters, an **int** called **numberOne** and **numberTwo**. These are our variables that we use in the function to make it dynamic. By dynamic we mean that the function can perform a process on these inputs to give us desired output. These **Parameters** have No Value on their own and should be treated as placeholders for when the function is called.



```
int Max(int numberOne, int numberTwo)
```

The Function Body

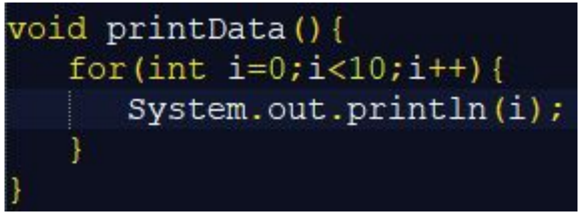
The body of the function contains our code to test. This code checks if our parameter **numberOne** is greater than **numberTwo**, otherwise it **Returns numberTwo**. The body of a function can be anything, but it must **Return** a variable that is the same **Type** as our function. Hence, in this situation since our function is of **Type int** and our **Parameters** are of **Type int**, we can return one of them as a result.



```
if(numberOne > numberTwo){
    return numberOne;
}
else{
    return numberTwo;
}
```

Advanced Function Topics

Functions can also **Return** a special type called **void**. **void** functions do not need to return a value, and can therefore perform tasks in code that normally wouldn't return anything. A common use of this is printing data out, since we would want to call that function, but not expect an explicit output.

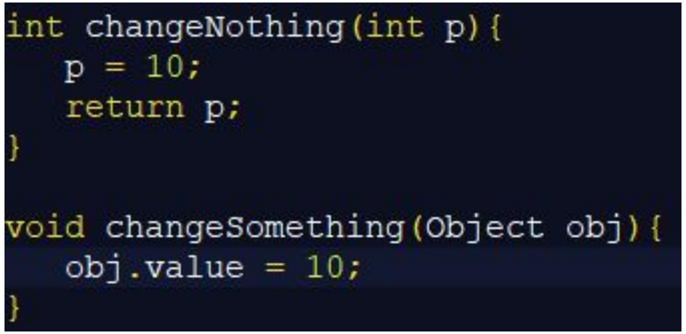


```
void printData(){
    for(int i=0;i<10;i++){
        System.out.println(i);
    }
}
```

Note in this example we have a function with no **Parameters** or **Return**, but it is still a valid function! It would print out 0 through 9 to the console.

Advanced Function Topics 2

Functions can do very complex tasks to specific parameters, specifically if those parameters are **Passed By Reference** instead of **Passed By Value**. These two concepts are important, so lets cover both separately to keep things clean.



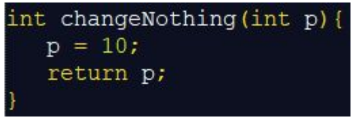
```
int changeNothing(int p){
   p = 10;
   return p;
}

void changeSomething(Object obj){
   obj.value = 10;
}
```

These two functions do two very different things!

Pass By Value

The concept of **Passing By Value** implies that the **Parameter** we are passing is just a **value**. By this we mean it is essentially a **Copy** of whatever we used as a parameter. Assume we give our function a **int** of value 300, when we run this function, our **int** will be unaffected, since it only gives a copy of itself to the function. All **Primitive Types (int, double, float, char, short, long, etc)** are **Passed By Value**.

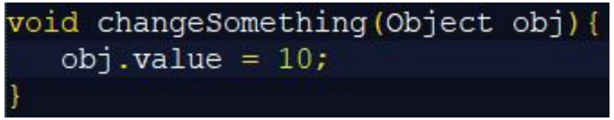


```
int changeNothing(int p){
   p = 10;
   return p;
}
```

Unsure what will happen to your variable?
Always ask first, "Is this a **Primitive Type**?"
If still unsure, just print out your variable
before and after!

Pass By Reference

Pass By Reference is when we pass the **actual Parameter and NOT a copy**. This is the default in Java for **Objects and CANNOT be changed!** This means whatever is changed in the function will reflect in the object **Outside the function.** Objects are typically anything that isn't a **Primitive**, and are most commonly what a **Class** would be used for. A common example is the **Car Object** example, which you can look up online.



```
void changeSomething(Object obj){
   obj.value = 10;
}
```

This simple function actually changes out Objects value! This can cause a massive headache if done by accident! When passing an Object, triple check!

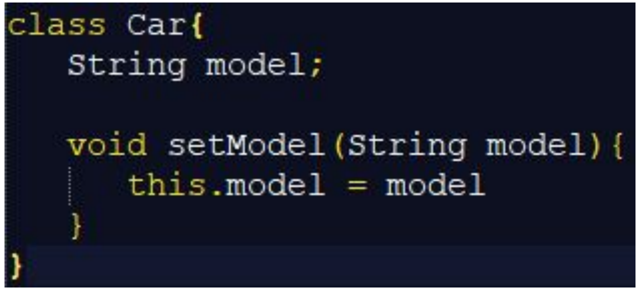
Advanced Function Topics 3

As a final regard, it’s important to know what the difference between a **Function** and a **Method** is. While they are used the same way, their **context** is different. A **Function** is typically not associated with a class, and can be called during execution of a program at any time, within the **Scope** of the code. **Scope** can be thought of as the current view of your program. If you need help on **Scope** contact me.

Methods are functions for an **Object** and are typically called using the syntax **obj.method()**

Using **Methods** is powerful because methods can manipulate **Object Variables**. A quick example of **Methods** is on the next slide.

Method Example



```
class Car{
   String model;

   void setModel(String model){
      this.model = model
   }
}
```

The **Method** setModel uses a **Parameter** called **model**, but in our **Object** we also have a variable called **model**. Methods can reference **Object Variables** by using the **this** keyword. When invoking **this** it's like saying, "Use **this objects** variable."

If you haven't learned about **Classes** and **Objects**, this is a crucial part.

That’s All

You now have the core of a **Function**. It is a **Subroutine** that can handle specific code, and we use **Parameters** to make our function dynamic, that way it can be used for several different operations. We can get a **Return** value from our function which we can use for further processing.

Functions are important for making code smaller but not having to retype the same code over and over every time we want to reference it. Use them wisely and you’ll be typing clean, understandable, non Italian Pasta Dish based code!