The **return** Statement

The **return** Statement

- General form:

```
return <expression>;
```

- Every function with a return type must have at least one **`return`** statement

- The returned expression's data type must match the return type of its function

```
// Function Definition
let max = (x: number, y: number): number => {
    if (x > y) {
        return x;
    } else {
        return y;
    }
};
```

The **return** Statement

- IMPORTANT: As the processor is evaluating a function call, when it reaches *any* **`return`** statement in the function definition, then the call is complete.
    - The computer evaluates the expression and returns the value immediately to its bookmark.
    - The rest of the function is ignored, skipped over, and not processed.
    - ***<u>This is ALWAYS, ALWAYS, ALWAYS true!</u>***

Return Semantics: Consider the following **function**

- Consider an alternate implementation of the **max** function

- *Is it still correct?* What happens when **a** is greater?

```
let max = (a: number, b: number): number => {

      if (a > b) {
            return a;
      }

      return b;

};
```

Returning from a **function**

```
let result: number;
result = max(10, 5);
```

1. The **max** function is called with arguments:
   `10, 5`

2. The processor jumps to max function.
   - if (a > b) evaluates to true, enters **then block**

3. **`return`** Statement encountered. Expression **`a`** evaluates to **`10`**. The function call is complete and this value is returned to step 4.

4. Processor jumps back to bookmark it left at #1 and "`max(10, 5)`" evaluates to **`10`**.

```
let max = (a: number, b: number): number => {

        if (a > b) {
                return a;
        }

        return b;

};
```

Parameters	
a	10
b	5

Every function call ***returns only one value***

- A function definition *may* have many **`return`** statements, however, for any given call only one return statement will be evaluated

- A function *may* contain a **`return`** statement inside of a loop, however, as soon as return is first encountered it will stop and return immediately

- Generally: as soon as the computer reaches *any* return statement within a function, the function call is complete and its job is done.