Parameters and Arguments

Introducing **Parameters**

- Parameters allow functions to require additional pieces of information in order to be called
- Parameters are specified within the parenthesis of function definition
- Parameters look a lot like variable declarations... *because they are!*
- Parameters are local variables to the function. Their names are scoped inside of the function body's block.

General Form

```
// Function Definition
let <name> = (<parameters>): <returnT> => {
   <statements>
};
```

Example

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

What effect does declaring parameters have?

Function Definition

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

Function Call Usage

```
max(3, 4);
```

Incorrect Function Call Usage

```
max(3);
```

Incorrect Function Call Usage

```
max(3, 4, 50);
```

- When a function declares **parameters**, it is declaring:
  "you must give me these extra pieces of information in order to call me"

- The function ***definition*** on the left says:
  "in order to call **`max`**, you must give me two number values"

- In the *usage* to the right, when we ***call*** max, we must give it two **`number`** values.

Parameters vs Arguments

These are **arguments.**

```
max(3, 4);
```

- Arguments are the *values* we assign to parameters
- The type of the arguments must match the types of the parameters
- We couldn't write `max("oh","no");`

These are **parameters.**

Example

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

Parameter Passing: Step-by-Step (1 / 3)

```
max(8, 9);
```

1. When a function is called…
   a. A "bookmark" is dropped at this place in code. We'll come back!
   b. The processor finds the function definition with the *same name*.
   c. Error if no match is found!

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

Notice the argument *matches* the parameters in type (number) and count (2)!

Parameter Passing: Step-by-Step (2 / 3)

```
max(8, 9);
```

```
// Function Definition
let max = (x: number, y: number): number => {
    x = 8;
    y = 9;
    if (x > y) {
        return x;
    } else {
        return y;
    }
};
```

2. Argument values are assigned to parameters
    a. This happens invisibly when the code is running. *You* will never see the lines to the right.
    b. However, each time a call happens, the processor assigns each argument value to its parameter.
    c. This is called "parameter passing" because we are copying arguments from one point in code *into* another function's block.

Parameter Passing: Step-by-Step (3 / 3)

```
perim(8, 9);
```

```
// Function Definition
let max = (x: number, y: number): number => {
    x = 8;
    y = 9;
    if (x > y) {
        return x;
    } else {
        return y;
    }
};
```

3. Finally, the processor then *jumps into* the function and continues onto the first line of the function body block