Methods, Scope of Variables and Conditional Statements

Engineering 1D04, Teaching
Session 3

An Example Method

A function that returns the square of its input argument: square(x) = x^2

```
double square(double x)
{
  return x*x;
}
```

Introduction to Methods

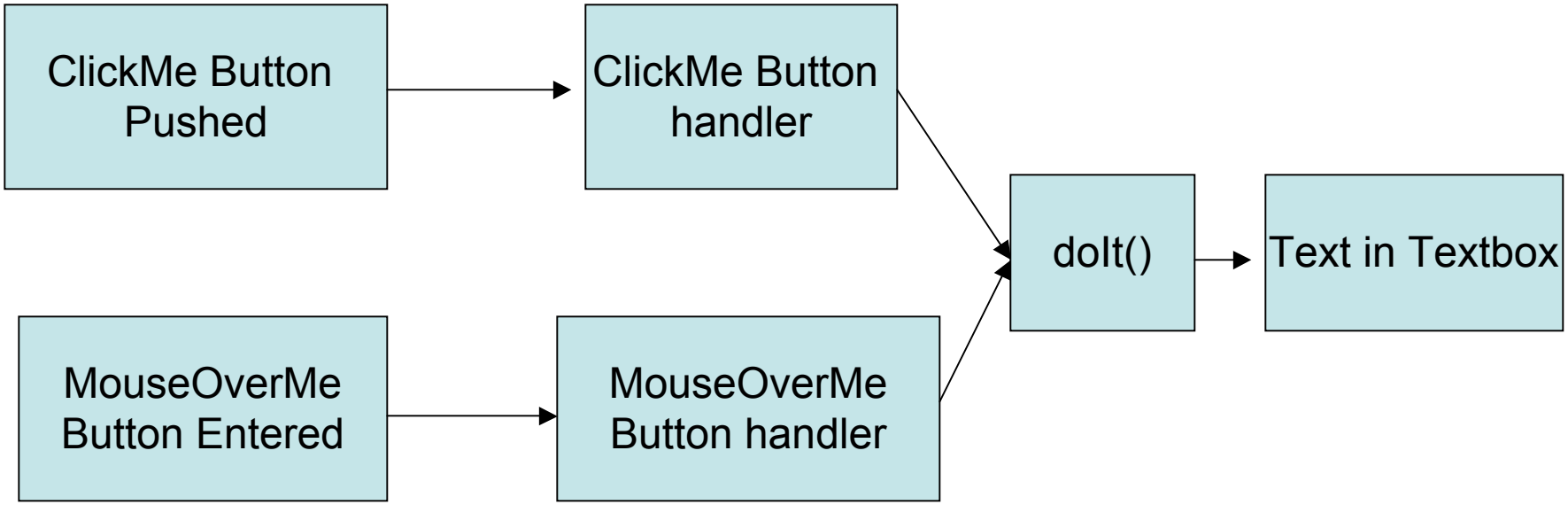
- In Lab 2 you were introduced to creating simple methods.

- Let's recap

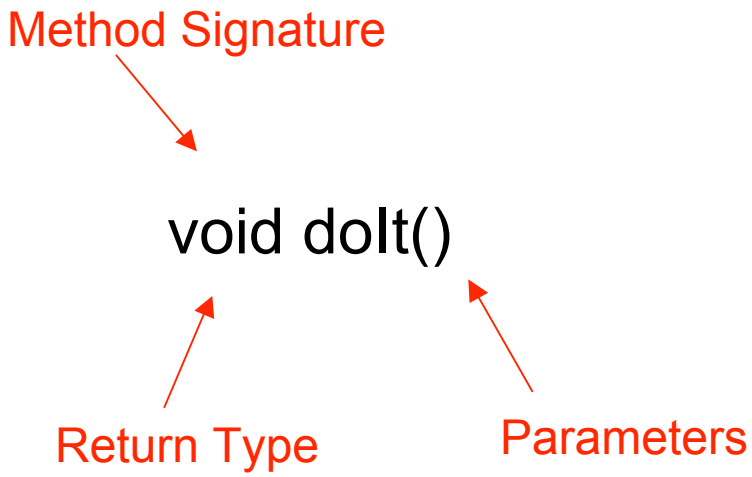
```
void doIt()
{
  textBox1.Text = "Hello Hamilton!!";
}
```

Introduction to Methods

```
void doIt()
{
  textBox1.Text = "Hello Hamilton!!";
}
```



Introduction to Methods



- *doIt* has no inputs (no parameters) and no outputs (a void return type).

Introduction to Methods

- To make doIt() more useful we can add a parameter (an input).

```
void doIt2(string string4Textbox)
{
      textBox1.Text = string4Textbox;
}
```

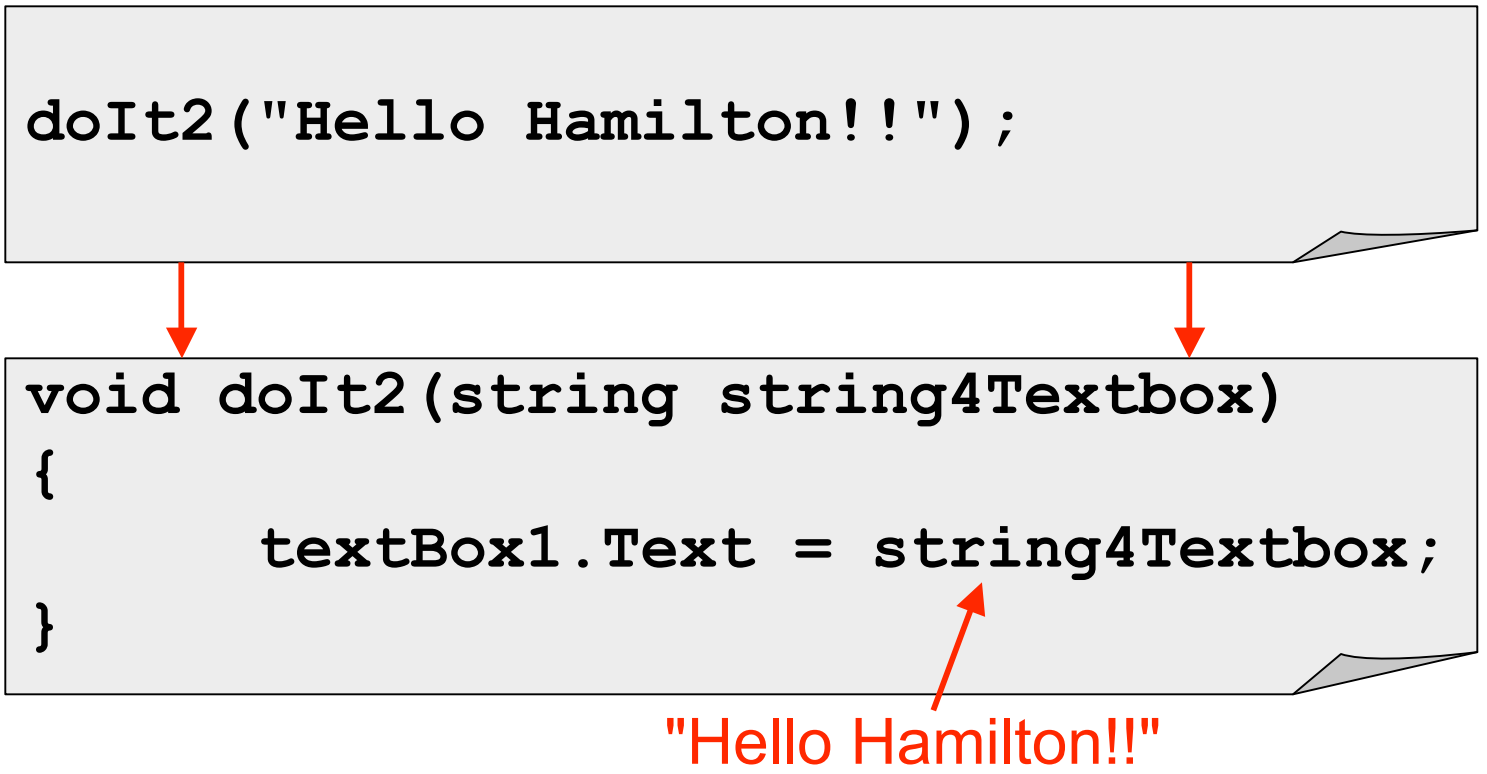
Introduction to Methods

- When doIt2 is called, we now need to pass it a parameter.

```
doIt2("Hello Hamilton!!");
```

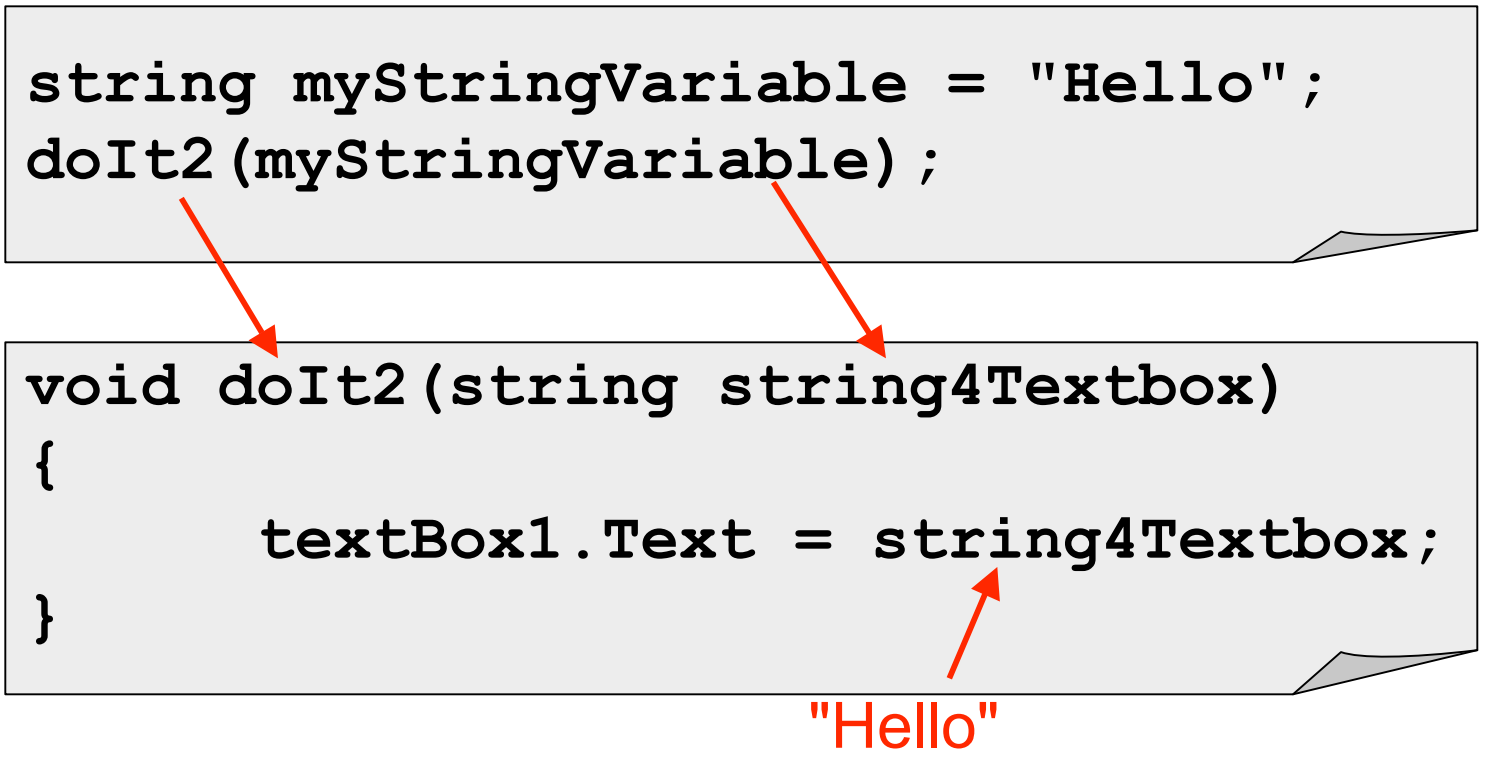
Introduction to Methods

- A copy of the string "Hello Hamilton!!" is then stored within the variable string4Textbox.



Introduction to Methods

- doIt2 could also be called with a string variable as a parameter.

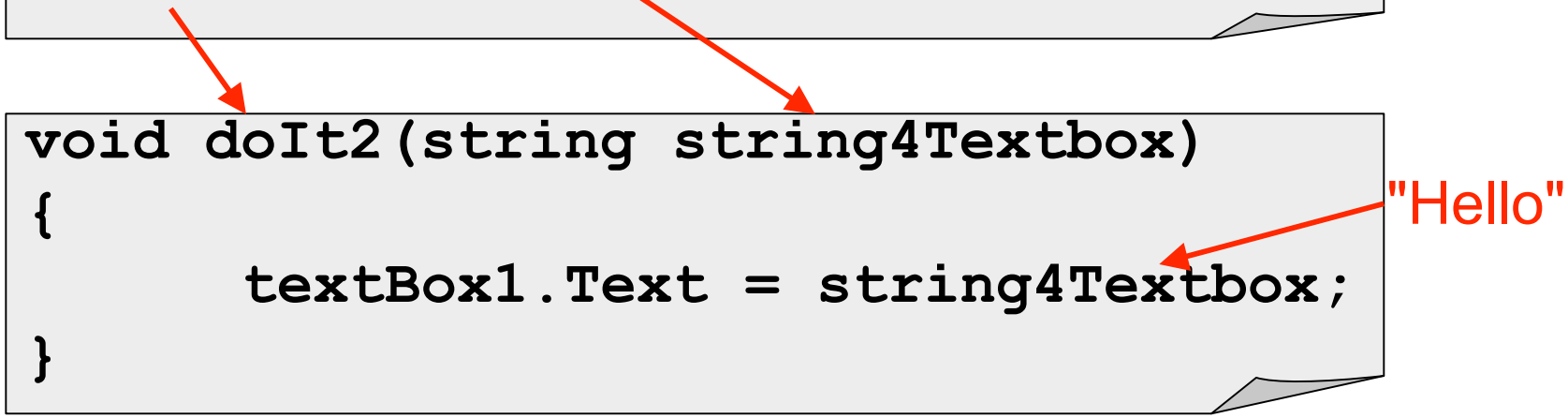


Introduction to Methods

- A copy of the variable's data is passed as a parameter. The copy is then no longer associated with myStringVariable.

```
string myStringVariable = "Hello";
doIt2(myStringVariable);
```

```
void doIt2(string string4Textbox)
{
        textBox1.Text = string4Textbox;
}
```



Why Methods?

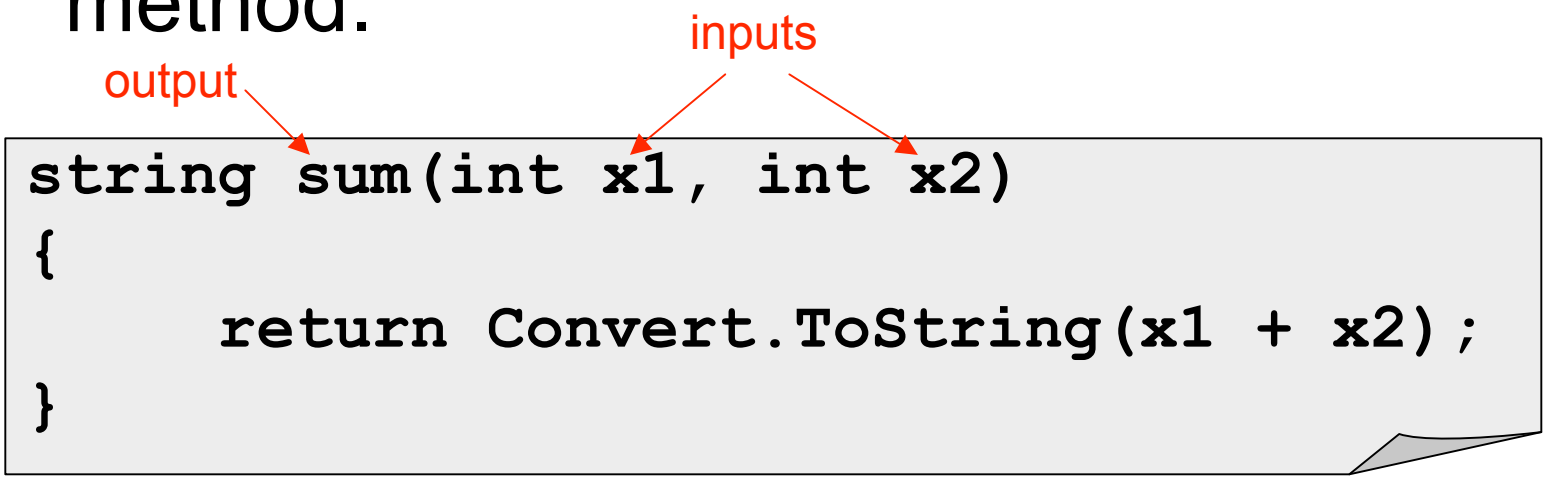
Why Methods?

- Separation of concerns
  - A natural way to handle complexity
  - Routinely used in all engineering fields
- Understandability
  - Provide meaningful names
  - Allows to hide details behind the interface
- Maintainability
- Reusability

Method Parameters

- Let's create a method to add two numbers together and return a string containing the result.

- We need 2 inputs and 1 output from the method.



Method Parameters

```
string sum(int x1, int x2)
{
      return Convert.ToString(x1 + x2);
}
```

- Multiple parameters can be given by separating them with commas.
- Replacing the void with string gives the method the return type string.
- The keyword return specifies the output value and ends the method call.

What does that mean?

Methods

- The keyword return specifies the output value and ends the method call.
  - Ending the call to the method means that there is no more processing done by that method and control returns to the action immediately after the call.

Full Sum Example

event handler created by double clicking a button in Visual C#

```
private void button1_Click(object sender, EventArgs e)
{
      textBox1.Text = sum(4,3);
      textBox1.Visible = true;
}
```

action that executes immediately after sum

```
string sum (int x1, int x2)
{
      return Convert.ToString(x1 + x2);
}
```

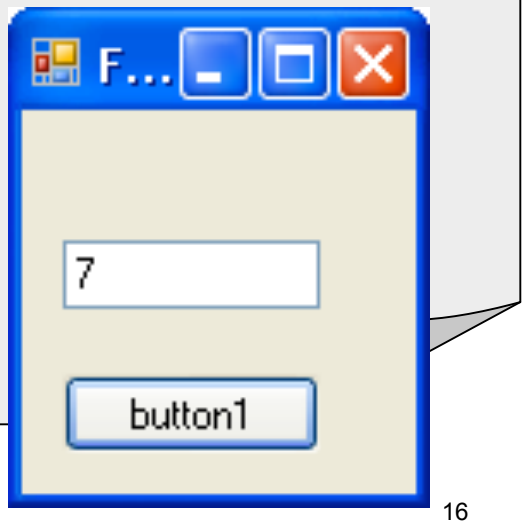
sum method

Full Sum Example

event handler created by double clicking a button in Visual C#

```
private void button1_Click(object sender, EventArgs e){
      textBox1.Text = sum(4,3);
      textBox1.Visible = true;
}

string sum (int x1, int x2){
      return Convert.ToString(x1 + x2);
}
```

sum method



Introduction to Methods

- Similar to variables, there are often naming conventions for Methods.
- In 1D04, the following convention will be adopted:
  - All first words are lowercase.
  - Subsequent words have their first letter capitalized.

Introduction to Methods

- Event handlers are simply methods that get called when an event happens.

```
private void button1_Click(object sender,
                           EventArgs e)
{
        ..do something here ..
}
```

- The *private* keyword will be discussed later

Introduction to Methods

- Method execution works like this:
  - New copy of the method is created
  - Method executes
  - This copy of the method is destroyed.
  - Control returns to the action immediately after the call to the method

- Because of the destruction of methods between calls, methods cannot remember the contents of their local variables from previous calls.

The Main Method

- Every C# program has a Main method
- It is written for you automatically by Visual C#
- By default the Main method is in Program.cs

```
static void Main()
{
 Application.EnableVisualStyles();
 …
 Application.Run(new Form1());
}
```

Scope of Variables

Introduction to Scope

- The scope of a variable defines where it is visible within a program.

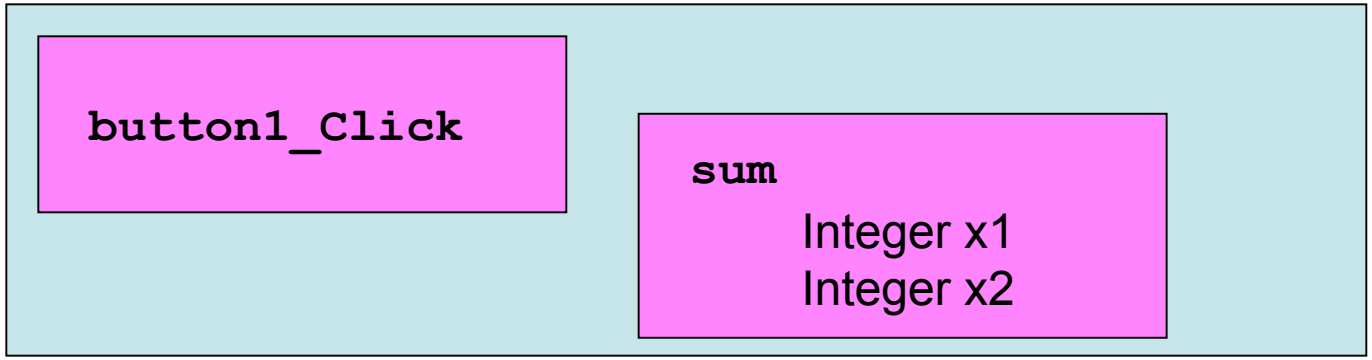
- All variables so far have had local scopes. They are only visible within the methods they are declared in and their values only last as long as the method is running.

Full Sum Example

- What are the scopes of the variables in this program?

```
private void button1_Click(object sender, EventArgs e)
{
      textBox1.Text = sum(4,3);
      textBox1.Visible = true;
}

string sum (int x1, int x2)
{
      return Convert.ToString(x1 + x2);
}
```

Full Sum Example




```
private void button1_Click(object sender, EventArgs e)
{
      textBox1.Text = sum(4,3);
      textBox1.Visible = true;

}

string sum (int x1, int x2)
{
      return Convert.ToString(x1 + x2);
}
```

ignore these for now

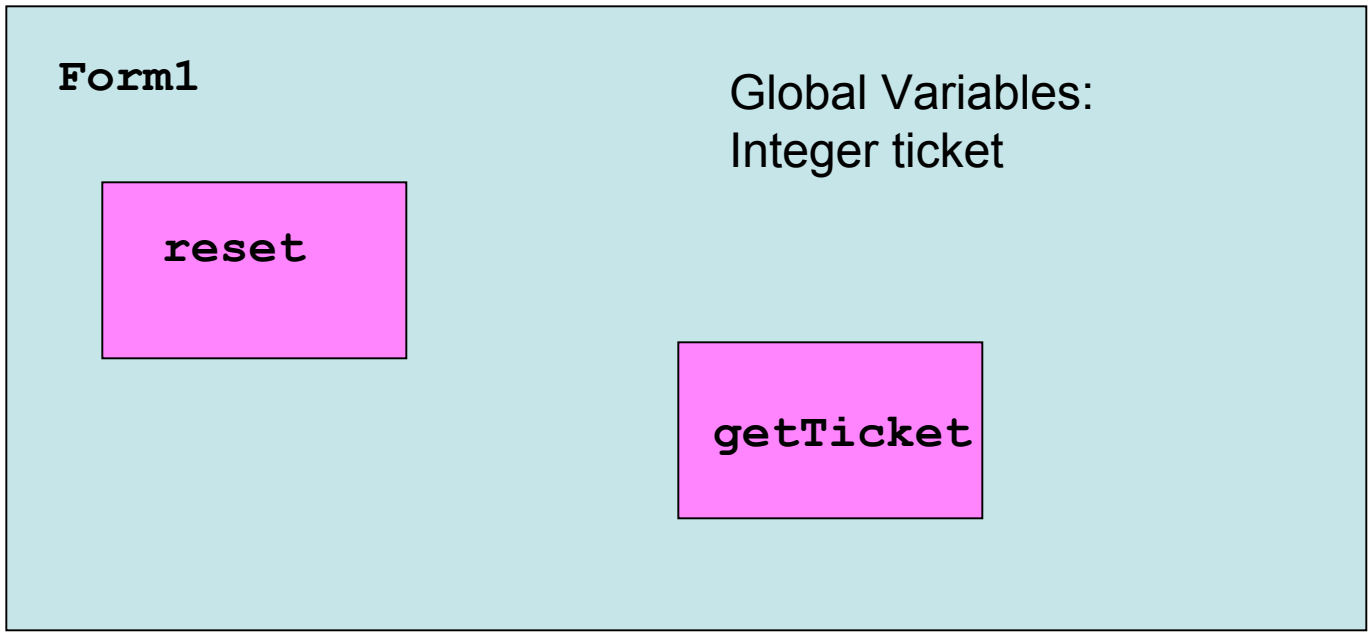
Variable Scope

- In general, the scope of a variable is between the {}'s that it is declared in.
- It is possible to have variables that are visible to all methods and retain their value as long as the program runs.
- These are "global" variables.

Variable Scope

```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }

  int ticket;

  void reset()
  {
    ticket = 0;
  }

  int getTicket()
  {
    ticket = ticket+1;
    return ticket;
  }
}
```

Variable Scope



Variable Scope

- The example uses a global variable to keep track of persistent information.
- The global variable is accessed by two methods and shares its value between them.
- It is declared inside the class {}'s, but outside of all the methods.

Variable Scope

- Each time get_ticket() is called the counter is increased and a new ticket number is returned.

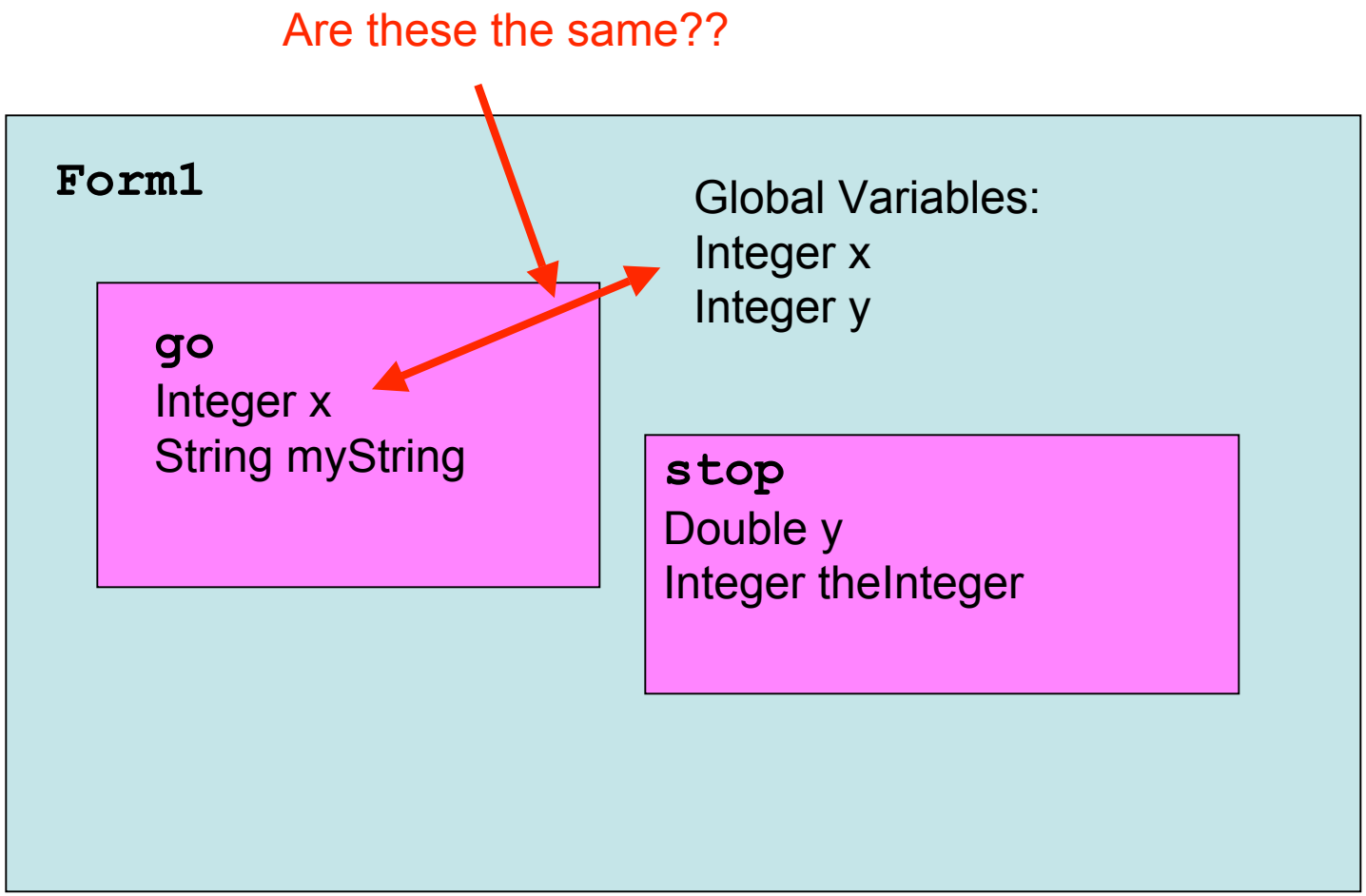
- Calling reset() resets the counter to 0.

Variable Scope

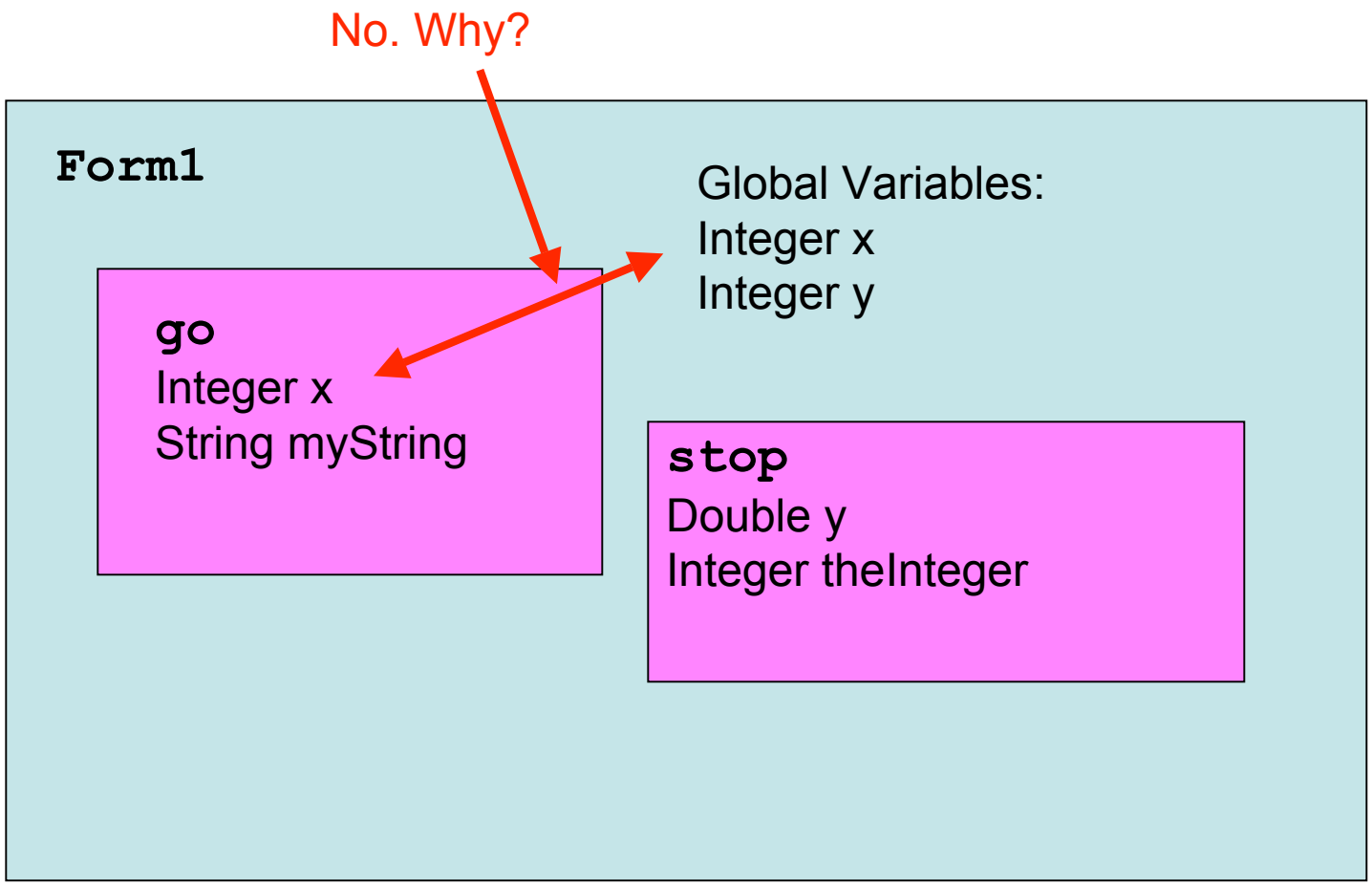
```
public partial class Form1 : Form
{
  public Form1()
  {
    InitializeComponent();
  }

  int x, y;

  void go()
  {
       int x; string myString;
       … more code here
  }

  int stop()
  {
    double y; int theInteger;
       … more code here
  }
}
```

Create a scope diagram for this code.

Variable Scope



Variable Scope



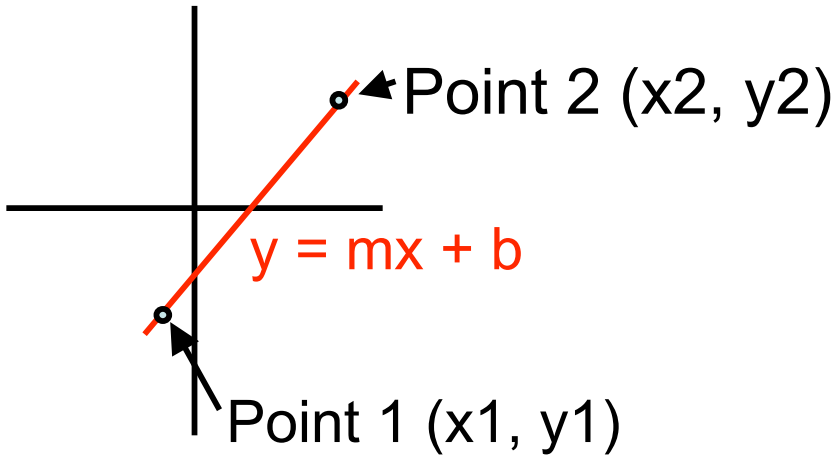
Variable Scope

- If there is a local and global variable with the same name they are not the same variable.
- If there is a local and global variable with the same name in scope, the local one is used while the global one is hidden.
- If you need to access a global variable, don't create a local variable with it's name.

Conditional Statements

A Problem: Slope of a Line

- We need a function that determines if the slope of a line is positive or negative.
- It must output either “Positive Slope” or “Negative Slope” to a textbox.



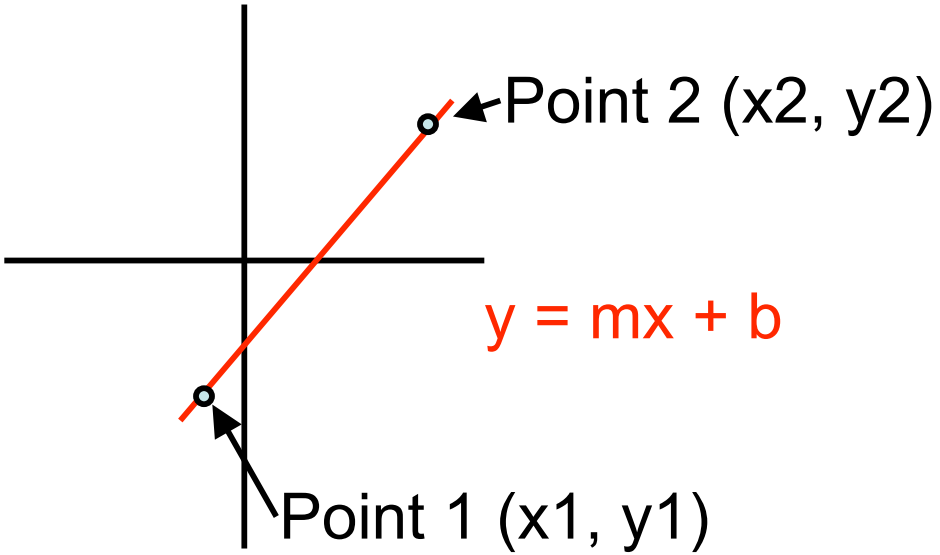
A line can be defined by the coordinates of its endpoints - that is what we will assume for this problem

A Problem: Slope of a Line

- We want to create a method (called pnSlope) to determine the slope and display it in a textbox

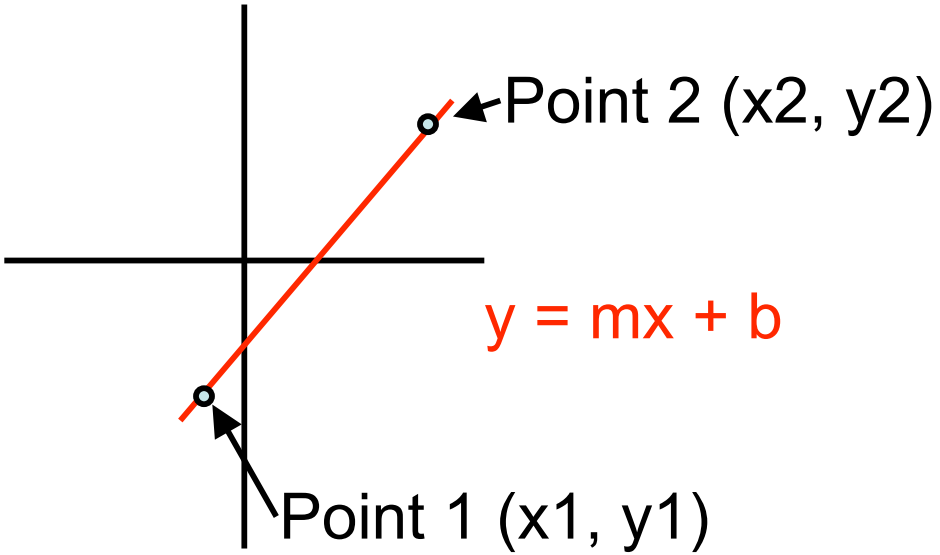
```
void pnSlope (double x1, double y1,
              double x2, double y2)
{
      . . . ?
}
```

A Problem: Slope of a Line



- How do we calculate the slope?

A Problem: Slope of a Line



- How do we calculate the slope?

$$\frac{y2 - y1}{x2 - x1}$$

A Problem: Slope of a Line

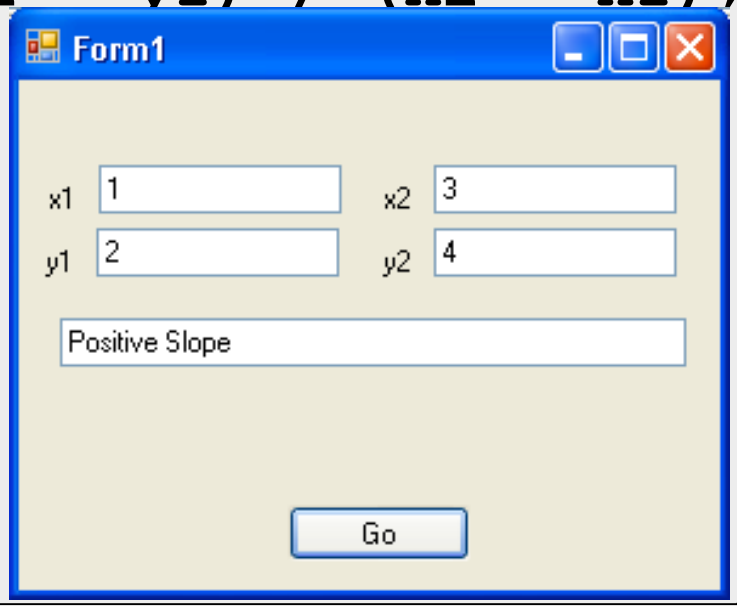
- How do we determine if it's positive or negative?

Conditional Statements

```
void pnSlope(double x1, double y1,
             double x2,double y2)
{
    double slope = (y2 - y1) / (x2 - x1);

    if (slope > 0)
      textBox1.Text = "Positive Slope";
    else
      textBox1.Text = "Negative Slope";
}
```

Conditional Statements

```
void pnSlope(double x1, double y1,
             double x2,double y2)
{
    double slope = (y2 - y1) / (x2 - x1);

    if (slope > 0)
      textBox1.Text =
    else
      textBox1.Text =
}
```



Conditional Statements

- (slope > 0) is called a condition.
- A condition evaluates to a Boolean (true or false).
- If slope > 0, the condition is true.
- If slope <= 0, the condition is false.

What does this mean?

Conditional Statements

- There are a number of operators that create conditions.
  - >, >=, <, <= do exactly what you think they should.
    - 5 >= 4 evaluates to true.
    - 5 >= 5 evaluates to true.
    - 5 >= 6 evaluates to false.

Conditional Statements

- == (Equality) Has to be different from the = assignment operator!
  - 5 == 5 evaluates to true.
  - 5 == 6 evaluates to false.
  - 5 = 6 assigns 6 to 5 and creates a compiler error.
- != (Not Equal)
  - 5 != 3 evaluates to true.
  - 5 != 5 evaluates to false.

Conditional Statements

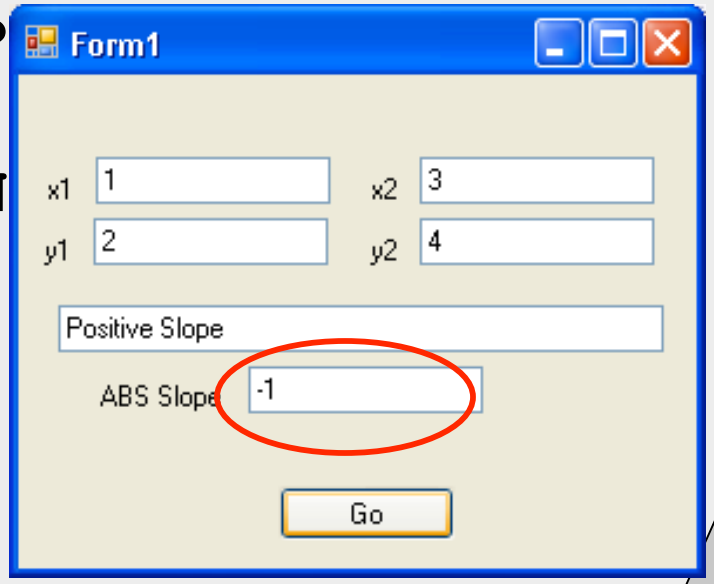
- What if we want to modify the function so that it also returns the absolute value of the slope.

Conditional Statements

```
double pnAbsSlope(double x1, double y1,
                  double x2, double y2)
{
    double slope = (y2 - y1) / (x2 - x1);

    if (slope > 0)
      textBox1.Text = "Positive Slope";
    else
      textBox1.Text = "Negative Slope";
      slope = -slope;

    return slope;
}
```

Conditional Statements

```
double pnAbsSlope(double x1, double y1,
                  double x2, double y2)
{
    double slope = (y2 - y1) / (x2 - x1);

    if (slope > 0)
      textBox1.Text = "P
    else
      textBox1.Text = "N
      slope = -slope;

    return slope;
}
```



Why is the output wrong?

If Statement

- We can only specify one statement for each branch of an if statement.

```
if (condition)
   statement1;
else
   statement2;
```

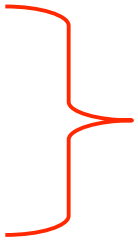
or

```
if (condition) statement1;
else statement2;
```

Compound Statements

- We can make multiple statements look like a single statement by enclosing them in matching braces

```
if (condition)
{
   statement1;
   statement2;
}
else
   statement3;
```



compound statement

Conditional Statements

```
double pnAbsSlope(double x1, double y1,
                  double x2, double y2)
{
    double slope = (y2 - y1) / (x2 - x1);
    if (slope > 0)
      textBox1.Text = "Positive Slope";
    else
    {
      textBox1.Text = "Negative Slope";
      slope = -slope;
    }
    return slope;
}
```

now it will work correctly ☺