



TRY



#



Level 2

Variables

Collecting Band Information

Let's make our console application more useful and have it accept band information.

- Ask for **name of band**
- Ask for **number of members**
- Announce **name of band** and **number of members**

We ask for the name of the band and number of members in one place...

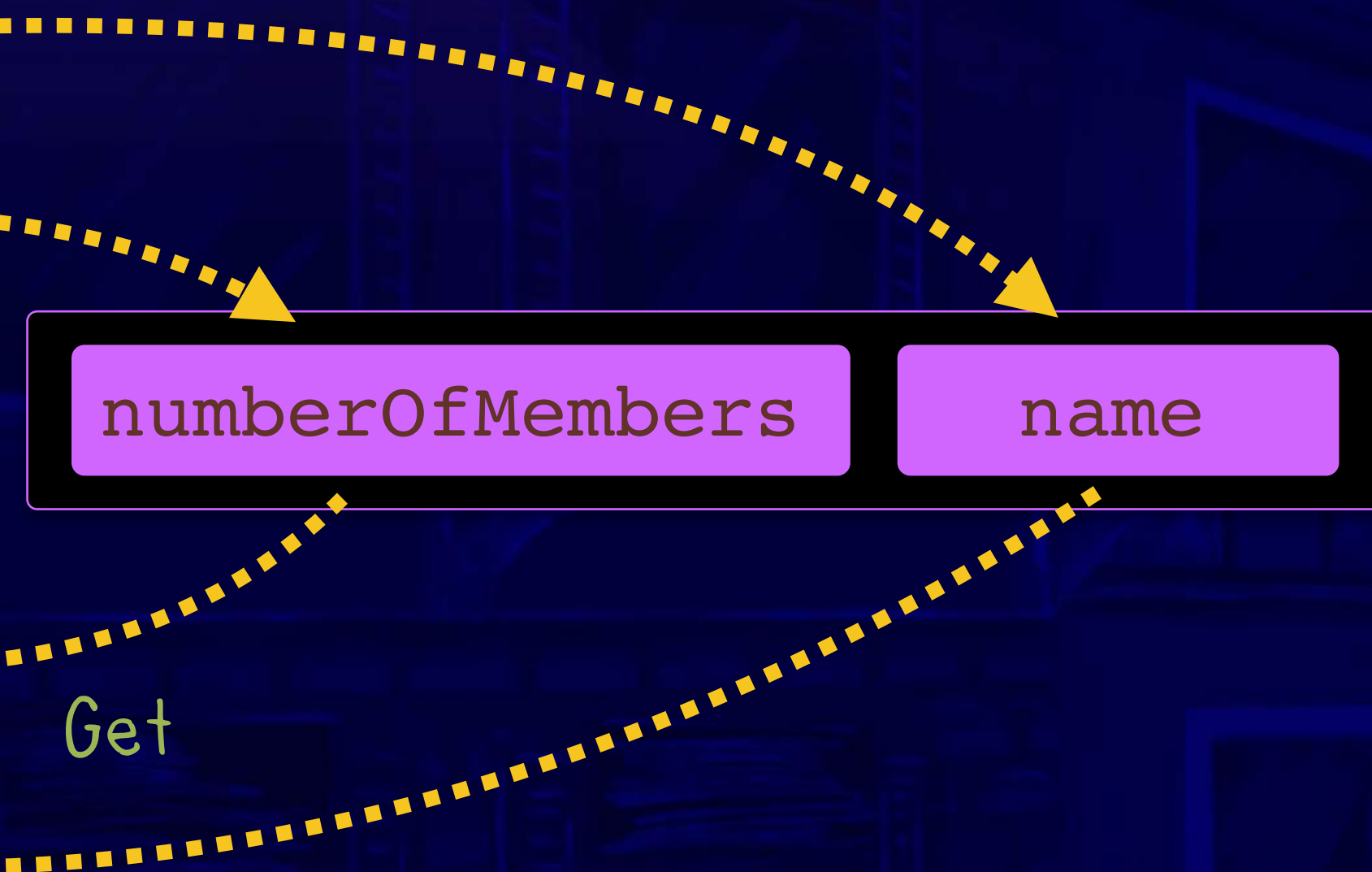


...but use that information someplace else.

Storing Information in Variables

Variables allow us to store information in memory for later use.

- Ask for **name of band**
- Ask for **number of members**
- Announce **name of band** and **number of members**



This saving and retrieving variables is known as "setting" and "getting"

Adding our Application's Messages

We'll first add all the things we expect to write to our users.

Program.cs

```
...
static void Main(string[] args)
{
    Console.WriteLine("What is the name your band?");

    Console.WriteLine("How many people are in your band?");

    Console.WriteLine("_____ has _____ members.");
}
...
```


Declaring Variables

To declare a variable, you specify the data type and then what you'd like to name the variable.

Data type

Name of variable



```
string name;
```

The diagram illustrates the components of a variable declaration. A central box contains the code `string name;`. A curved arrow points from the label 'Data type' to the word `string`. Another curved arrow points from the label 'Name of variable' to the word `name`. The word `string` is enclosed in a dashed green box, and the word `name` is enclosed in a dashed blue box.

Declaring Our Variable & Setting It With ReadLine

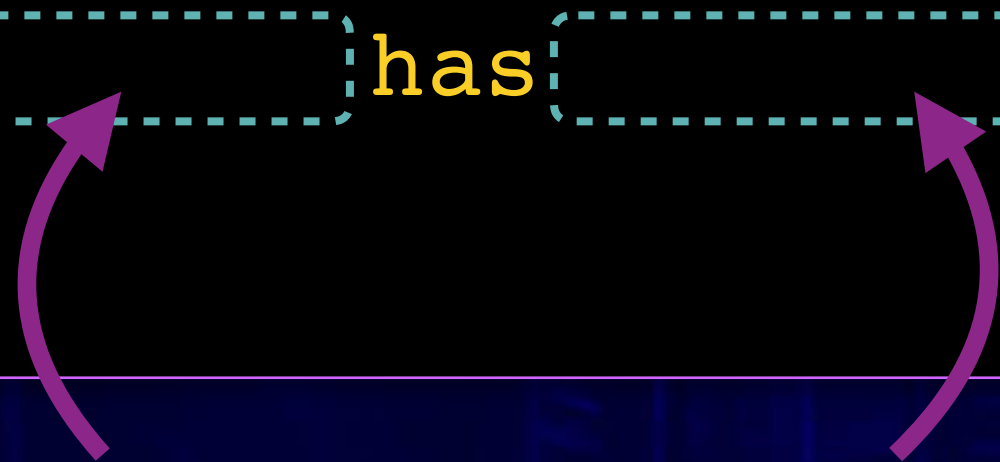
We can set a variable using '=', which sets the variable on the left to the value being assigned on the right.

Program.cs

```
...
static void Main(string[] args)
{
    Console.WriteLine("What is the name your band?");
    string name = Console.ReadLine();

    Console.WriteLine("How many people are in your band?");

    Console.WriteLine("[ ] has [ ] members.");
}
...
```



Our variables will need to go here

We Can Add Our Variable to Our Output

To use a variable, we just enter its name wherever we want to use it.

Program.cs

```
...
static void Main(string[] args)
{
    Console.WriteLine("What is the name your band?");
    string name = Console.ReadLine();

    Console.WriteLine("How many people are in your band?");

    Console.WriteLine(name + " has _____ members.");
}
...
```

We need to use string concatenation to combine the variable and hard-coded strings

Data Types & Errors

The Console.ReadLine method returns a string, but numberOfMembers is expecting an int!

Program.cs

```
...
static void Main(string[] args)
{
    Console.WriteLine("What is the name your band?");
    string name = Console.ReadLine();

    Console.WriteLine("How many people are in your band?");
    int numberOfMembers = Console.ReadLine();

    Console.WriteLine(name + " has _____ members.");
}
...
```

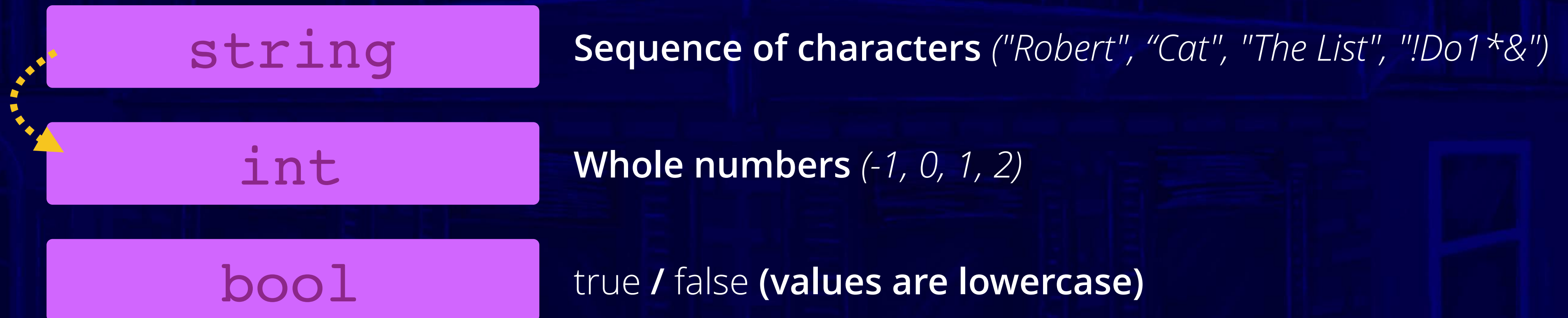


Cannot implicitly convert `string` to `int`

Data Types

There are 15 built-in data types in C#, but we're going to focus on string, int, and bool for now.

We can convert String to Int, but we need to do so explicitly.



Converting Our string to an int

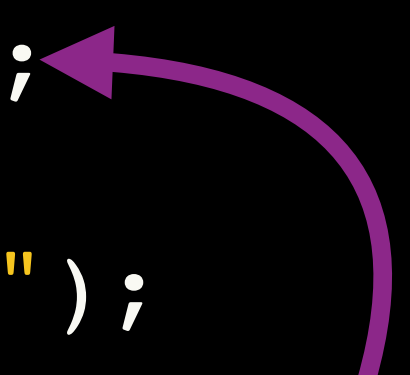
The `int.Parse` method allows us to convert our string into an int, fixing our error.

Program.cs

```
...
static void Main(string[] args)
{
    Console.WriteLine("What is the name your band?");
    string name = Console.ReadLine();

    Console.WriteLine("How many people are in your band?");
    int numberOfMembers = int.Parse(Console.ReadLine());
    Console.WriteLine(name + " has _____ members.");
}
...
```

Converts a string to an int



Adding Our numberOfMembers to Our Message


Since our string is between double quotes, we'll need to break it up to add our variable.

Program.cs

```
...
static void Main(string[] args)
{
    Console.WriteLine("What is the name your band?");
    string name = Console.ReadLine();

    Console.WriteLine("How many people are in your band?");
    int numberOfMembers = int.Parse(Console.ReadLine());

    Console.WriteLine(name + " has " + numberOfMembers + " members.");
}
...
```

 numberOfMembers is an int, but we do not need to explicitly convert it to a string. Most data types can implicitly be converted to a string.

The Application — Now With Variables

We have made the application able to store and retrieve values provided by the user.

>>>

```
$ dotnet run
```

```
What is the name of your band?
```

Ask for name of band

>>>

```
$ Awesome Inc
```

```
How many people are in your band?
```

Ask for number of band members

>>>

```
$ 3
```

```
Awesome Inc has 3 members.
```

Announce name of band and
number of members



Quick Recap on Variables & Data Types

Variables are used to store values in memory for later use.

- Storing a value in a variable is *setting* it
- Retrieving a value from a variable is *getting* it
- Set a variable by using the = character followed by the desired value
- Get a variable by using the variable's name
- Not all data types can be converted implicitly (in many cases we'll have built-in Parse methods)



Invalid Input Handled

Now when something invalid is entered, they'll get a simple user-friendly message.

```
>>>
```

```
$ dotnet run
```

```
What is the name of your band?
```

```
>>>
```

```
$ Awesome Inc
```

```
How many people are in your band?
```

```
>>>
```

```
$ Duck
```

```
input was not valid
```



Expressions

We often need to compare two things for a condition in an application.

- Do we have 1 band member?
- Do we have more than 0 band members?
- Is our band not named "Awesome"?

```
>>>
```

```
1 == 1
```



```
true
```

```
>>>
```

```
1 > 0
```



```
true
```

```
>>>
```

```
band != "Awesome"
```



```
true
```


else Conditions

The **else condition** executes a block of code if a condition is false.

If whatever is in the parentheses is false...

...do whatever is in the else code block.

If Else Conditions

```
if (ready)
{
    DoIfTrue();
}
else
{
    DoIfFalse();
}
```


Series of Conditions

You can use `else if` to create a series of conditions.

If `ready` is true, do this...

If `ready` is false and the band is named "Awesome", do this...

If `ready` is false and the band is not named "Awesome", do this...

Series of Conditions

```
if(ready)
{
    DoIfTrue();
}
else if(name == "Awesome")
{
    DoFalseAndAwesome();
}
else
{
    DoIfFalseAndNotAwesome();
}
```

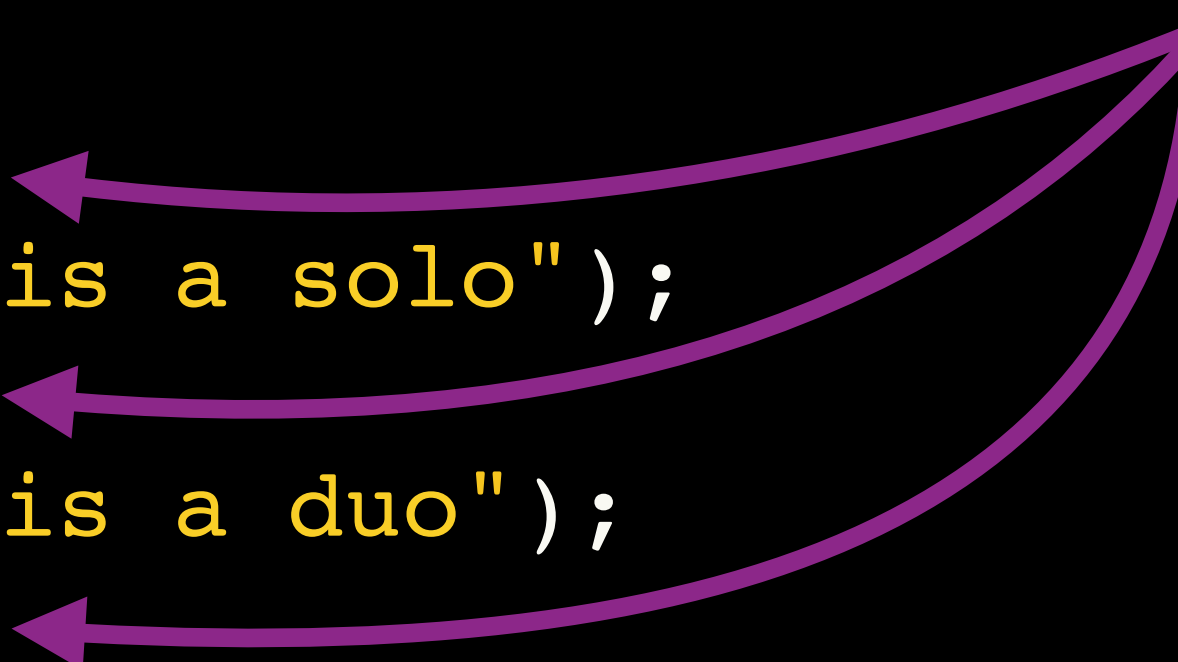

Declaring Our Band Type Using Conditions

We can use `if`, `else if`, and `else` to write our type of band to the console.

Program.cs

```
...  
if(numberOfMembers < 1)  
{  
    Console.WriteLine("You must have at least 1 member");  
    Environment.Exit(0);  
}  
else if(numberOfMembers == 1)  
    Console.WriteLine(name + " is a solo");  
else if(numberOfMembers == 2)  
    Console.WriteLine(name + " is a duo");  
else  
    Console.WriteLine(name + " has " + numberOfMembers + " members");  
...
```

Curly braces not required here



Curly braces are optional when you only have one line of code in a code block

The Application Is Complete

Our application is now more robust and ready to handle different information.
We've added two features:

- Printing the type of band (solo, duo, etc.) based on number of members
- Handling invalid user input

```
>>>
```

```
$ dotnet run
```

```
What is the name of your band?
```

```
>>>
```

```
$ Awesome Inc
```

```
How many people are in your band?
```

```
>>>
```

```
$ 2
```

```
Awesome Inc is a duo
```



Quick Recap on Conditions & Expressions

We can change the flow of our code using conditions and expressions.

- The *if* statement only executes its block when the condition is *true*.
- The *else* statement only executes its block when the *if* condition is *false*.

Expressions

`==` is equal to

`!=` is **NOT** equal to

`!` before a condition passes the condition when it's **NOT** *true*

