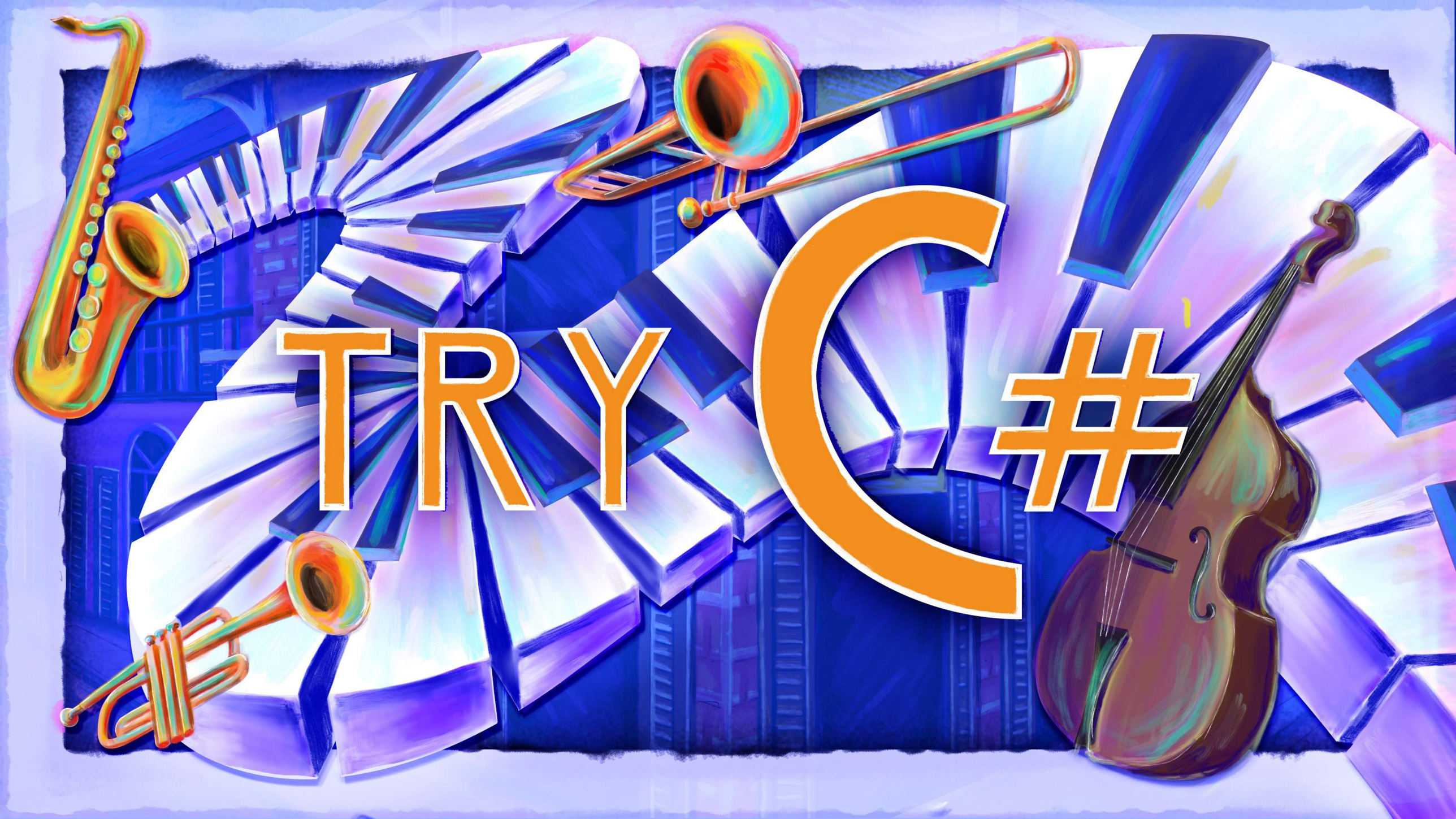
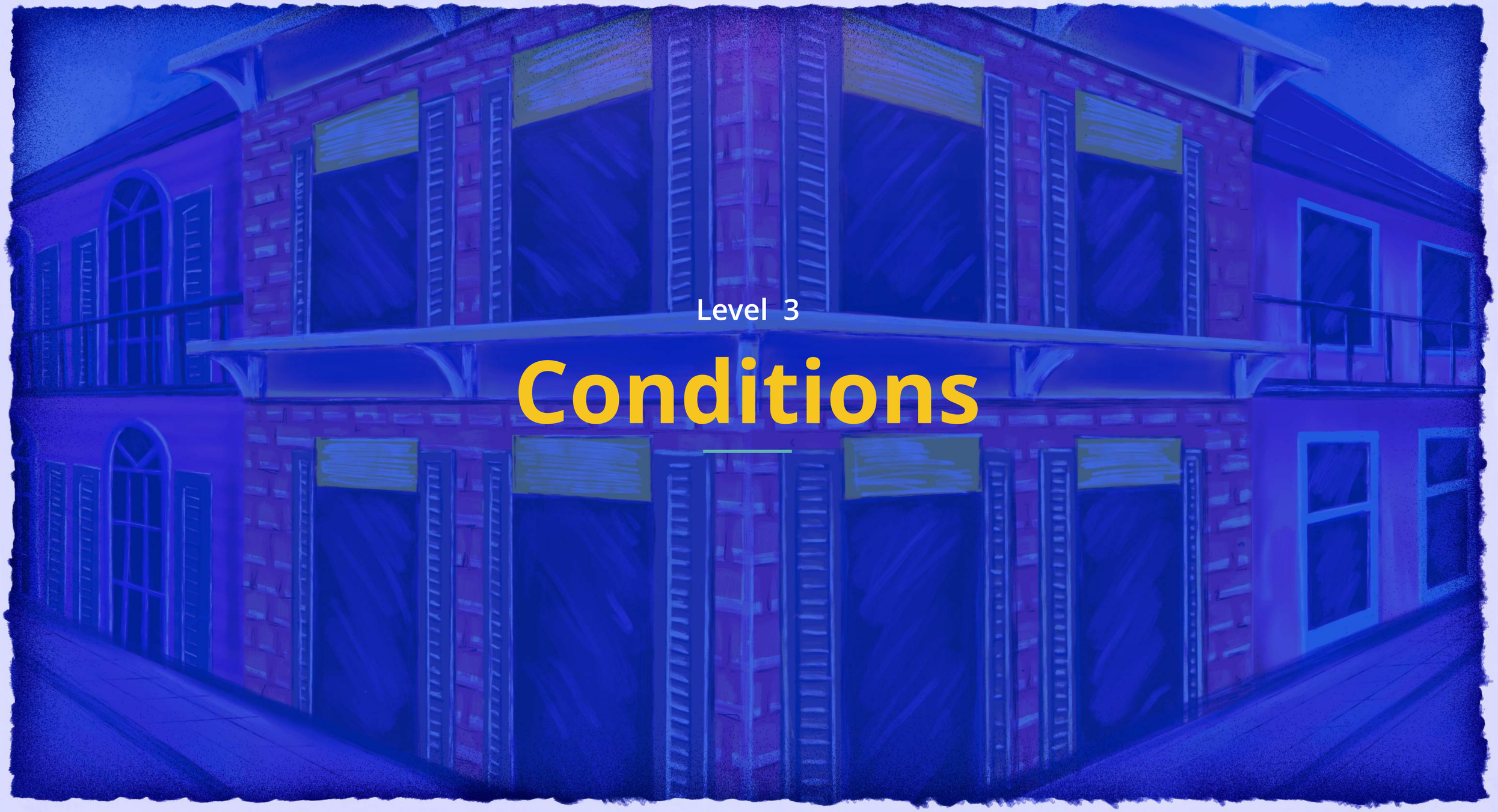

TRY C#



Level 3

Conditions

Handling Invalid User Input

When users enter something we can't parse, the application will break.

```
>>> $ dotnet run
What is the name of your band?
>>> $ Awesome Inc
How many people are in your band?
>>> $ Duck
Unhandled Exception:
System.FormatException: Input
string was not in correct format.
```

Our app is only expecting numbers here

TRY C#

Our Input Isn't Safe

Here, the Parse method is throwing an error because it can't covert the string to 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 = int.Parse(Console.ReadLine());

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

```
Input string was not in correct format.
```

TryParse **Lets Us Safely Parse Variables**

The int.TryParse **method will return** true **or** false **depending on if it was able to parse the** string**.**

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 = 0;
  int.TryParse(Console.ReadLine(),                    );

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

Second parameter is what we are going to output the parsed value to

First parameter is what we want to parse

Output Parameter

Some methods use “output parameters” that allow it to set variables using the out keyword.

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 = 0;
  int.TryParse(Console.ReadLine(), out numberOfMembers);

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

numberOfMembers will be set if parsing is successful

Conditions

Conditions allow us to change our application's behavior based on specific circumstances.

If whatever is in the parentheses is true...

If Condition

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

DoNoMatterWhat();
```

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

This will be run regardless of
if condition is true or false

Examples of Results

Depending on if ready **is** true **or** false **determines what code will get executed.**

If Condition

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

DoNoMatterWhat();
```

ready **is** *true*

```
DoIfTrue();
DoNoMatterWhat();
```

ready **is** *false*

```
DoNoMatterWhat();
```

if ready is true, then we will execute both DoIfTrue and DoNoMatterWhat methods – but if ready is false, we will skip the DoIfTrue method in the if block

int.TryParse **Returns** true **or** false

In the event the value could be parsed, TryParse **is** true — **otherwise, it's** false**.**

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 = 0;
  if(int.TryParse(Console.ReadLine(), out numberOfMembers))
  {

  }
…
```

Returns true if it can parse the first parameter – otherwise, returns false

Not "!" Expression

Since we only want to change our behavior when TryParse fails, we can use Not (!).

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 = 0;
  if(!int.TryParse(Console.ReadLine(), out numberOfMembers))
  {

  }
…
```

This is effectively saying "if TryParse is NOT true" then...

The Environment.Exit Method

The Environment.Exit method immediately exits the program.

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 = 0;
  if(!int.TryParse(Console.ReadLine(), out numberOfMembers))
  {
    Console.WriteLine("input was not valid");
    Environment.Exit(0);
  }
…
```

Providing a 0 here means the application will close saying it ran successfully

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*