CSC102

Branching and Looping

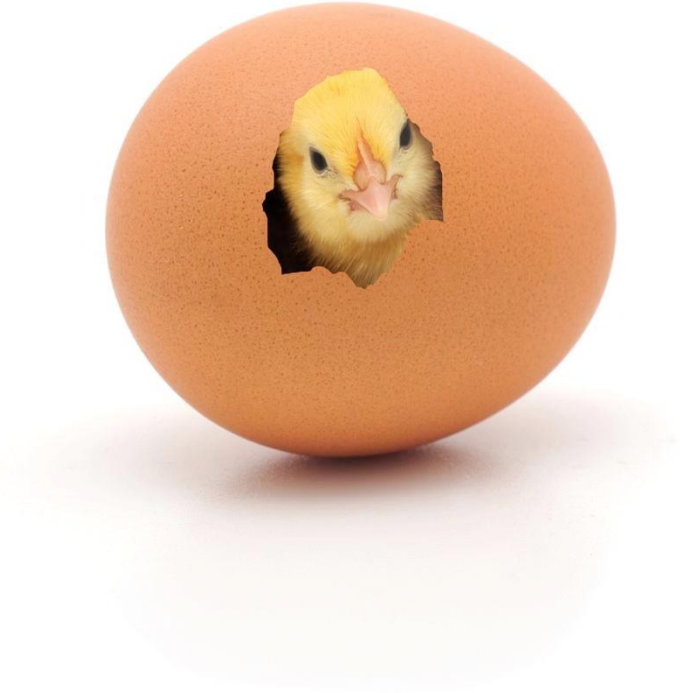
Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Implement conditional branching
2. Discover and use loops in your code

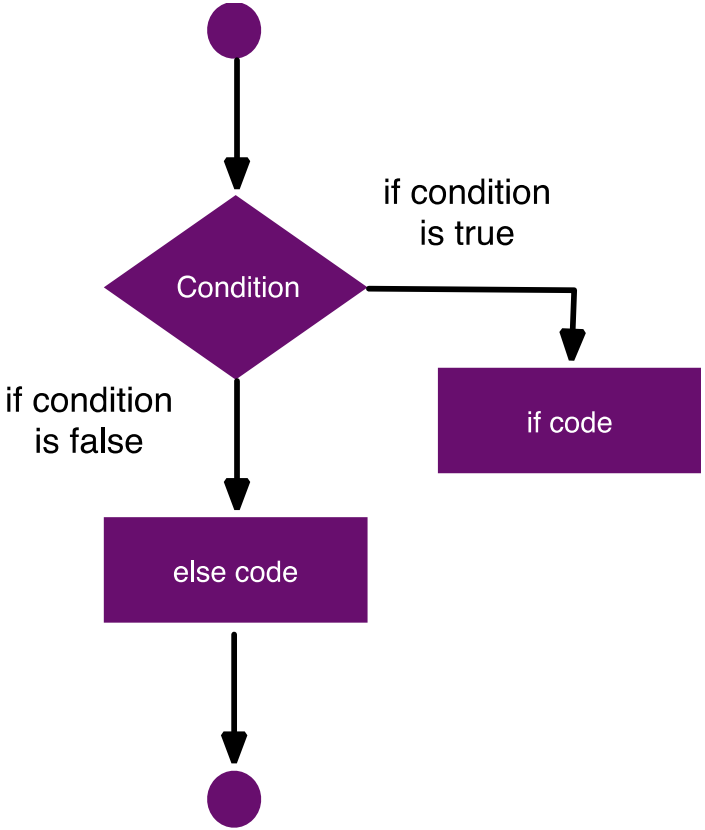


Implement conditional branching

Xamarin University

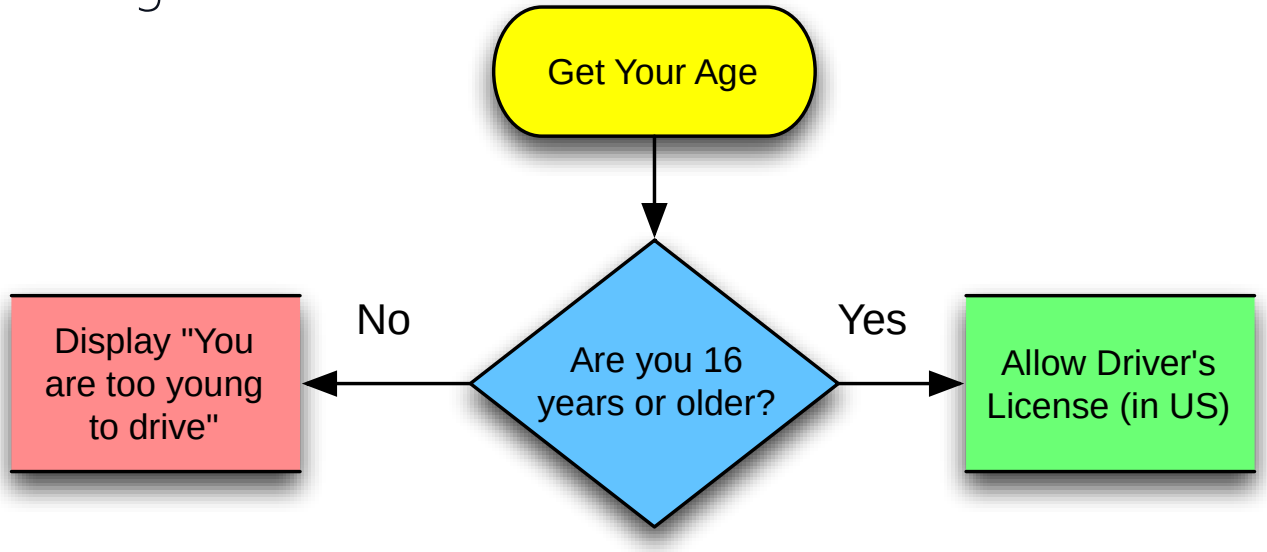
Tasks

1. Explore conditional statements
2. Using the if and else statements
3. Expressing nested if statements
4. Creating more complex conditionals with switch
5. Utilize string formatting and parsing



Decisions, decisions

- Making decisions based on values of variables is common in programming



Conditional execution

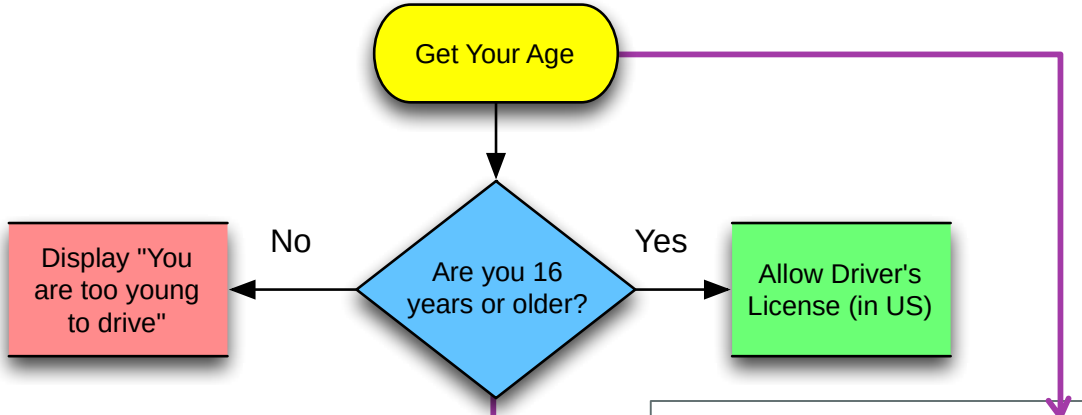
- An **if** statement evaluates a boolean expression (true or false) and executes the associated statements if true

```
int yourAge = 30;
const int minimumDriverAge = 16;

if (yourAge >= minimumDriverAge)
  Console.WriteLine ("You can drive!");
```

Output is only displayed if the age is greater than or equal to the minimum required

How does if work?




```
int yourAge = 30;
const int minimumDriverAge = 16;

if (yourAge >= minimumDriverAge)
  Console.WriteLine ("You can drive!");
```

Logical AND operator

- The && operator allows your code to evaluate two expressions together and execute the associated statements when *both* are true

My age must be between 16 and 100 in order to drive →

```
int myAge = 25, minAge = 16, maxAge = 100;

if ( myAge >= minAge && myAge <= maxAge )
    Console.WriteLine ("You can drive.");
```

Logical OR operator

- The || operator allows your code to evaluate two expressions together and execute the associated statements when *either* is true

Bring an umbrella when it is cloudy or rainy

```
string weather = ...;

if ( weather == "Cloudy" || weather ==  "Rainy" )
   Console.WriteLine ("Bring an umbrella.");
```

Grouping statements together

- A *block* is a sequence of related statements grouped together inside "{" braces "}"

block starts here

```
bool needUmbrella = false;
string weather = {...};

if (weather == "Cloudy" || weather == "Rainy") {
    needUmbrella = true;
    Console.WriteLine("Bring an umbrella!");
}
```

These two statements are *both* executed when the if condition is true

block ends here

When should I use braces?

- Braces are only required when multiple statements need to be executed, however it is better to use braces for clarity – even with single statements

```
string weather = ...;

if ( weather == "Cloudy" || weather ==  "Rainy" ) {
   Console.WriteLine ("Bring an umbrella.");
}
```

Braces are unnecessary here, but preferred. If you add additional statements later, there will be less confusion since the braces establish the block to be executed.

Choosing between two alternatives

- An else statement is executed when the if condition is false

```
int myAge = 25;
int yourAge = 30;

if (yourAge > myAge) {
  Console.WriteLine ("You are older than I am.");
}
else {
  Console.WriteLine ("I am older than you!");
}
```

We execute this block when only when our if condition is false

Using Nested if statements

- A nested **if** statement allows you to test cascading conditions

We can add another if statement as a statement in the else

```
double temperature = ...;

if (temperature < 65) {
  Console.WriteLine ("Wear a coat.");
}
else {
  if (temperature > 90) {
      Console.WriteLine ("Stay inside.");
  }
}
```

Nested if shorthand

- C# allows a simpler syntax for nested **if** statements which reads easier

```
double temperature = ...;

if (temperature < 65) {
  Console.WriteLine ("Wear a coat.");
}
else if (temperature > 90){
  Console.WriteLine ("Stay inside.");
}
```

Notice there is no block with braces, but it's really the same code

Final else condition

- Can add a final **else** to catch conditions that do not match prior **if** statements

```
double temperature = ...;

if (temperature < 65) {
  Console.WriteLine ("Wear a coat.");
}
else if (temperature > 90){
  Console.WriteLine ("Stay inside.");
}
else {
  Console.WriteLine ("It's very pleasant outside.");
}
```

Using a Switch statement

- Complex if/else conditions can be more easily represented with a **switch** statement which selects one matching value from a set of alternatives

Each alternative has a case label

```
int answer = 2;
switch (answer) {
    case 1:
        Console.WriteLine ("the answer was 1");
        break;
    case 2:
        Console.WriteLine ("the answer was 2");
        break;
}
```

value being tested must be a numeric or string expression

Switch statement blocks

- Statement block for case is terminated with the break statement, there is no need for braces although you can include them if you like

execution for "1" stops here

```
int answer = 2;
switch (answer) {
    case 1:
        Console.WriteLine ("the answer was 1");
        break;
    case 2: {
        Console.WriteLine ("the answer was 2");
        break;
  }
}
```

When there is no match

- Can express a *default* match which is executed when none of the case statements match

```
switch (answer) {
    case 1:
        Console.WriteLine ("the answer was 1");
        break;
  case 2:
        Console.WriteLine ("the answer was 2");
        break;
  default:
        Console.WriteLine ("the answer was more than 2");
        break;
}
```

this is executed when the value is not 1 or 2

Falling through statements

- Can combine `case` statements (OR) to execute the same code in either match

1 OR 2 will match and execute this code

```
switch (answer) {
    case 1:
    case 2:
        Console.WriteLine("the answer was 1 or 2");
        break;
    case 3:
    case 4:
        Console.WriteLine("the answer was 3 or 4");
        break;
}
```

Falling through statements

- Can only fall through if no statements are present on the case label

```
switch(answer){
    //...
                    Error!
case 3:
    Console.WriteLine( "Falling through..." );  // Nope!
case 4:
    Console.WriteLine( "the answer was 3 or 4" );
    break;
}
```

Flash Quiz

Xamarin University

Flash Quiz #1

① What is the final value of z in the following block of code?

```
int x = 3, y = 10, z = 0;
if ( x > y )
    z = 20;
```

a) 20

b) 0

c) Unknown

Flash Quiz #1

① What is the final value of z in the following block of code?

```
int x = 3, y = 10, z = 0;
if ( x > y )
    z = 20;
```

a) 20

b) 0

c) Unknown

Flash Quiz #2

② What is the final value of z in this block of code?

```
int x = 3, y = 10, z = 0;
if ( x > y )
   z = 20;
else
   z = 15;
```

a) 20

b) 15

c) 0

Flash Quiz #2

② What is the final value of z in this block of code?

```
int x = 3, y = 10, z = 0;
if ( x > y )
   z = 20;
else
   z = 15;
```

a) 20

b) 15

c) 0

Reminder: Console

- The Console lets us interact with the computer's keyboard and display

`Console.Read` and `Console.ReadLine` get text from the keyboard

`Console.Write` and `Console.WriteLine` output text to the display

```
Console.Write("What's your favorite color? ");

string color = Console.ReadLine();

Console.WriteLine("You said {0}.", color);
```

Turning text into numbers

- Text can be translated into a numeric value by *parsing* the string

```
Console.Write("Choice (1-3): ");

string choice = Console.ReadLine();

int value = int.Parse(choice);
```

`int.Parse` converts a string value to an integer

Beware: the string value you pass to `int.Parse` must convert to a valid numeric value – otherwise the program will stop execution and give you an error

Converting values to text

- Variables can be turned into strings by calling **`ToString()`** on the value

ToString converts our double to a string

```
double temperature = 65.0;

string temperatureText = temperature.ToString();

Console.WriteLine("It is {0} degrees", temperatureText);
```

Group Exercise

Write a program that asks your age and prints a response

Xamarin University

Flash Quiz

Xamarin University

Flash Quiz #1

① To take a string and make it an integer I can call _____

a) ToString()

b) ToInteger()

c) int.Parse()

d) None of the above

Flash Quiz #1

① To take a string and make it an integer I can call _____

a) ToString()

b) ToInteger()

c) int.Parse()

d) None of the above

Flash Quiz #2

② To take an integer and turn it into a string I can call _____

a) ToString()

b) ToInteger()

c) int.Parse()

d) None of the above

Flash Quiz #2

② To take an integer and turn it into a string I can call _____

a) ToString()

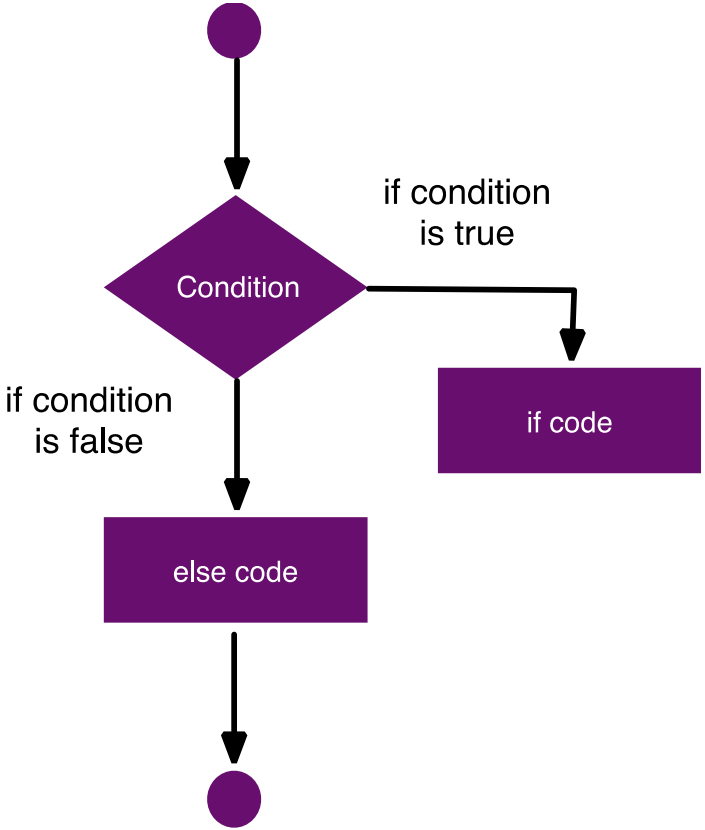
b) ToInteger()

c) int.Parse()

d) None of the above

Summary

1. Explore conditional statements
2. Using the if and else statements
3. Expressing nested if statements
4. Creating more complex conditionals with switch
5. Utilize string formatting and parsing

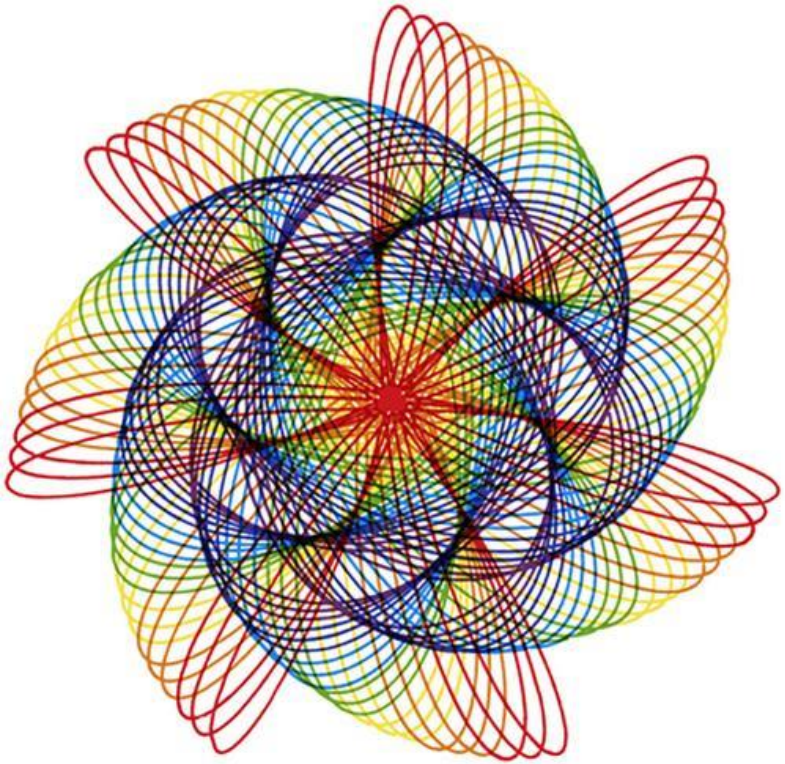


Discover and use loops in your code

Xamarin University

Tasks

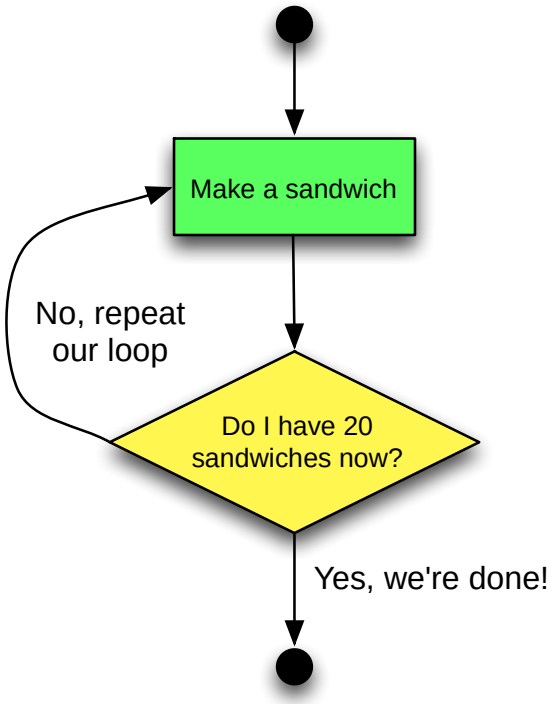
1. The importance of using loops
2. Using While and For loops
3. When to use break and continue statements



What is a loop?

- We often have to repeat blocks of code multiple times to accomplish a specific goal, in programming, we call this a *loop*

Let's make 20 sandwiches..



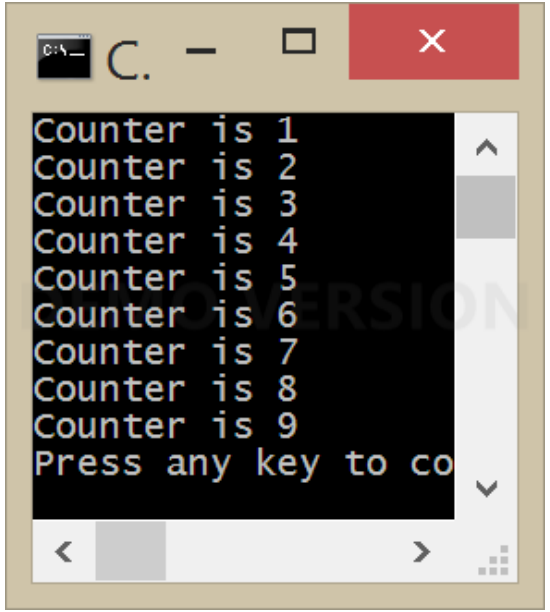
While loops

- A while loop continues to repeat the associated block of code *as long as* the test condition is true

test condition is true until counter reaches 10

```
int counter = 1;

while (counter < 10) {
    Console.WriteLine("Counter is {0}", counter);
    counter++; // increment
}
```



While loops examined

- A while loop has four parts

2 Test counter value (the "test condition")

1 Initialize counter

3 Execute Statement(s)

```
int counter = 1;

while (counter < 10) {
    Console.WriteLine ("Counter is {0}", counter);
    counter++; // increment
}
```

4 Modify counter

Counted loops

- While loops are oriented around a conditional expression, when that expression is a counter you can use a **for** statement as an alternative

```
for (int counter = 1; counter < 10; counter++) {
  Console.WriteLine ("Counter is {0}", counter);
}
```

This produces the same output as our **while** loop, but the code is generally easier to read and understand

For loop structure

- Same four steps used in while loops are part of the **for** statement, separated by semicolons

1 Initialize counter value

2 Test counter value

4 Modify counter

3 Execute Statement(s)

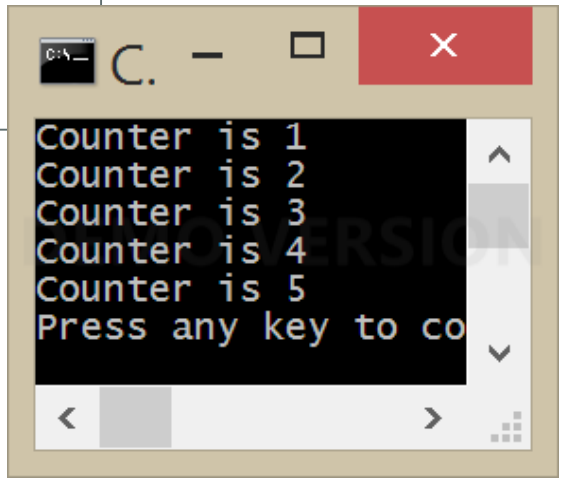
```
for (int counter = 1; counter < 10; counter++) {
  Console.WriteLine ("Counter is {0}", counter);
}
```

Exiting early from a loop

- A break statement terminates a **for** or **while** loop completely

```
for (int counter = 1; counter < 10; counter++) {
  Console.WriteLine ("Counter is {0}", counter);
  if ((counter % 5) == 0)
     break;
}
```

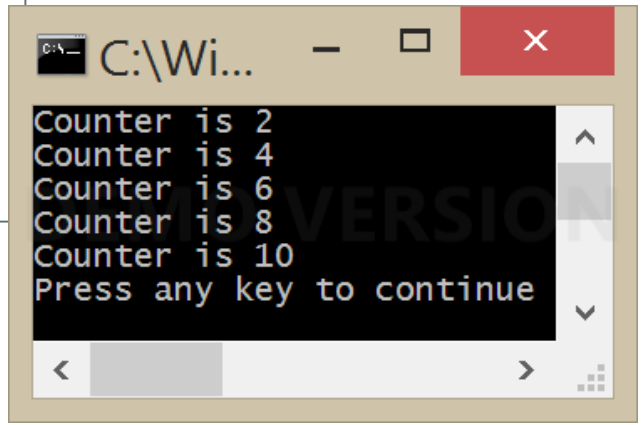
When this line is run, the loop will end without further evaluations when we hit the first number evenly divisible by five



Skipping a loop iteration

- A continue statement goes back the top, skipping remaining statements

```
int counter = 0;
while (counter < 10)
{
    counter++;
    if ((counter % 2) != 0)
        continue; // skip if not even number
    Console.WriteLine("Counter is {0}", counter);
}
```



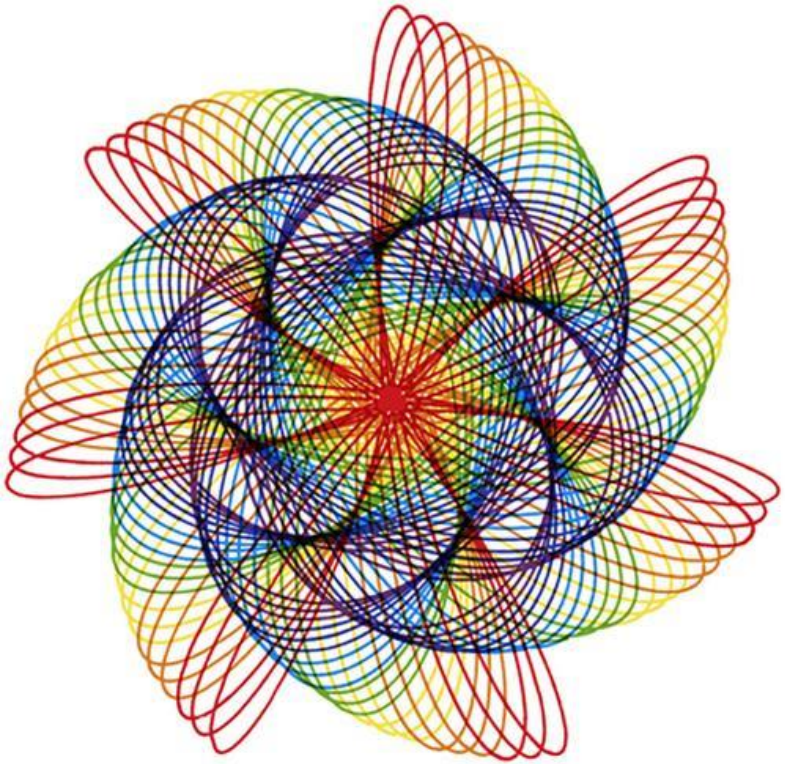
Individual Exercise

Write FizzBin 1…100

Xamarin University

Summary

1. The importance of using loops
2. Using While and For loops
3. When to use break and continue statements



Where are we going from here?

- You now know how to make decisions in your programs and how to execute statements multiple times with loops

- In the next course, we will start to look at how to structure our programs using *object oriented* design

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft