Dice Game

Here we will talk about how to use variables to store, modify, and use data.

Variables

Variables, or vars, allow us to store values. The uses for them are near endless but common ones include storing players health, keeping track of turns, determining values for attacks or moves.
You likely have an understanding of variables from math. While there are some things that carry over, there will also be a lot of new things. Let's start with the types of variables we can have in code.

Type	Declaration	Stores	Example
Integer	`int`	Whole numbers	-1, 0, 1, 2
Float	`float`	Decimal numbers	1.0, 2.3, 4.75
Boolean	`bool`	True or False	T or F
Character	`char`	Single letters	'a', 'b'
String	`string`	A string of letters	"Hello World"

There are more types and later on we can actually make our own. These here are just the most commonly used ones.
So now lets talk more about each specific one.

Integers

Integers, or ints, are the same as in math. They can only store whole numbers, positive or negative.
One key thing to note about is using them in math. If you were to do a division like `3 / 4` you would expect to get `0.75` back. If the numbers used are ints the output must also be a whole number and anything after a decimal is truncated.
Meaning `3 / 4` is equal to `0`. It does not get rounded, everything after the decimal is cut off.

Here we'll stick in the fact that all math we use in code does follow the order of operations (BEDMAS). If we were to do `5 + 3 / 4` with ints we would get `0` and not `2`. Though you can write `(5 + 3) / 4` and get `2` back.
This goes for the next variable type, floats, as well.

Floats

Floating point numbers, or floats, are real numbers. They can store decimal points unlike ints.
Float math does not truncate the result. Meaning when dividing two floats we keep the decimal, so `3 / 4` is equal to `0.75`. No rounding occurs.
Now that's a bit of a fib. That equation would still equal `0`. This is because the numbers are still ints. If we wanted the computer to use float math for these we could write it like :

`3.0 / 4.0` or `3f / 4f`

Adding a decimal, even zero, or a 'f' tells the code that the numbers are floats and it will use float math.

Booleans

Booleans, despite only having two possible values, are likely the most useful and can become very complex.
Acceptable values for booleans are, `true` and `false` or `1` and `0`.
The zero and one are just like binary if you know what that is.
Booleans are how we interact with "control structures" which we will get to later and are the most useful part of coding.

Booleans can be set with values like `5 > 4` which would give the variable a value of true. A similar order of operations applies to booleans as well, but we will talk about that during control structures.

Characters

Characters, or chars, are single letters. There isn't too much fancy stuff here.
Characters must be wrapped in single quotes like `'a'` and contain no extra spaces. Most symbols, like `'!'` or `'&'` will work. Some do have specific codes you will have to look up.
Capitals do matter as `'a'` is not the same as `'A'`.

Strings

Strings, or strings (yeah there's not really a nickname), are actually just a list of characters. They can contain all letters and symbols that are valid characters. Again, certain special characters will require you to look up how to use them.
When creating a string it must be wrapped in quotations `""`, like `"Hello World"`.
Strings are very specific and case-sensitive. `"Hello World"` is not the same as `"hello world"` or `"HelloWorld"`.

Now we move on to setting up, manipulating, and using variables.

Declaring Variables

In code, when we make a variable it is called a declaration.
C# is what's called a "strongly typed" language. This means that once a variable is declared it can **not** change type. This also means we need to tell the computer what type of variable it will be before we declare one.
Let's take a look at the parts. Go ahead and type these out and play with them throughout.

```
int myInt = 4;

float myFloat = 3f / 4f;

bool myBool = true;

char myChar = 'd';

string myString = "Hello World";
```

Here we've just shown an example of each of the types we've used so far. What you might notice is each of them looks very similar. This is because they are.
Variable declarations are made of three parts no matter the type :

- First we need to declare the type as stated earlier. This is represented by the first part of the line.
  For instance with the first line we wanted an integer value, so we used `int` as the first word. We can get the type name we want from the table at the top of this lesson under the "Declaration" column.
  These declarations do require at least one space after them.

- Next is the variable name. This is the name associated with our variable and how we will access them later in our code.
  It comes after the type of variable with at least one space in between them. Variables name can include all letters, hyphens `-`, underscores `_`, and the dollar sign `$`.
  You'll notice my variable names are written in lowerCamelCase. This is because Unity's pre-written variables use this format and it's nice to have everything match. Keeps the code readable like we talked about in the last lesson.
  Another rule to follow when making variable names is to make them descriptive, but don't make it redundant. Then later when you want to use your variable you will be able to tell what it does.
  If you had a variable to store your bank account balance a good name could be `accountBalance`. A bad name would be `x` or `myCurrentBankAccountBalance`.

- Last we *can* set the value of our variable.
  It would be perfectly valid to say `int myInt;` (remembering lines need to end with a semi-colon `;`). This simply would create a variable of type `int` with the name `myInt` with no value. We will use this functionality in the future.
  More often you'll want to set the data in the variable right away. We do this with an equal sign `=`.
  The `=` comes right after our variable name and does not need a space, though we use one to make it more readable. After the `=` we need a value of the same type as the variable.
  What you will have noticed above is that `myFloat` is a little different from the other variables. We've done this to show that you can use equations when setting variables. Below is exactly the same as above.

```
int myInt = 2 + 2;

float myFloat = 0.75;

bool myBool = 6 < 8;
```

```
string myString = "Hello " + "World";
```

I've omitted the char as it is not usually used like this.
Also note the white space left within the quotations in `myString`.

Go ahead and try declaring your own variables. Your editor will likely tell you if something is wrong.
You can write these within the `Start` function like we did with `Debug.Log()`.

Using Variables

There are a few main reasons we use variables in our code.
They make it easier to read through, quicker to change values, and allow for storing of info.\

Let's start with how we use variables. We're going to pretend we've declared all the variables in the above section.
If we wanted access to our variable we could simply write `myInt;` or any of the other names. This will do absolutely nothing though, we need to use this variable for something.
Currently myInt stores the value `4`. What if we wanted to change that?
Pretty simply you can just type `myInt = 5` or whatever or number / equation you wanted.
We can even set a variable equal to another variable. The easiest way to think of it is just when you type the name of the variable imagine it being replaced verbatim with the value.

```
float myNewFloat = 4.56;
myFloat = myNewFloat;
```

Go ahead and try setting some other variables to new values.

Before we mentioned we can set these variables via an equation. Let's try making one with a variable.

```
int myNewInt = 4;
myInt = myNewInt + 3 * 2;
```

This here will set `myInt` to 10. This is because the order of operations is still followed.

Another important thing to mention is that you can set a variable with itself.

```
myInt = myInt + 1;
```

This here will increase `myInt` by one. Operations like this are very common, for instance incrementing, decrementing a number, dividing, or multiplying. Since these are common they have shortcuts.

```
myInt = myInt + 1;
myInt += 1;
myInt++;

myInt = myInt - 1;
myInt -= 1;
myInt--;

myInt = myInt * 2;
myInt *= 2;

myInt = myInt / 2;
myInt /= 2;
```

Each group of commands is equivalent. You'll notice that incrementing and decrementing are so common they've been shortened even further.
Try these out and experiment.

The last big point of using variables we need to touch on using them in the context of functions.
We talked earlier how we used `"Hello World"` in the `Debug.Log` function to print it to the console. In this context the `"Hello World"` is just a string. We just didn't bother setting it to a variable.
Try replacing your current `Debug.Log` with this :

```
myString = "Print this variable";
Debug.Log(myString);
```

This works just as well. As said before when we type the variables name it's easy to just pretend that it gets replaced with the value of the variable.
As long as the type of the variable matches the type the function needs you can use it.
You can also combine strings and variables.

```
int numOfApples = 4;
Debug.Log("I have " + numOfApples + " apples.");
```

This will print out : `I have 4 apples.`

On Your Own

Things like this can be very complex. It can be much harder to understand if you only read through. Here are some things you can try to help you understand the material so far.

1. Print out a small story using `Debug.Log` .
2. Print a shape like :

```
  *
 ***
*****
 ***
  *
```

3. Print a count down using a variable to keep track of the number.
4. Use variables and equations to give you the volume of a sphere (4/3 * pi * r^3).

In the next lesson we will be talking more about booleans and how we can use them to create intelligent code.