Getting characters

you can use emojipedia.org

If you're familiar with other programming languages, you may be wondering about a `Character` or `char` type. Dart doesn't have that. Take a look at this example:

```
const letter = "a";
```

So here, even though `letter` is only one character long, it's still of type `String`.

But strings are a collection of characters, right? What if you want to know the underlying number value of the character? No problem. Keep reading.

Dart strings are a collection of **UTF-16 code units**. UTF-16 is a way to encode Unicode values by using 16-bit numbers. If you want to find out what those UTF-16 codes are, you can do it like so:

```
var salutation = "Hello!";
print(salutation.codeUnits);
```

This will print the following list of numbers in decimal notation:

```
[72, 101, 108, 108, 111, 33]
```

H is 72, **e** is 101, and so on.

These UTF-16 code units have the same value as Unicode code points for most of the characters you see on a day to day basis. However, 16 bits only give you 65,536 combinations, and believe it or not, there are more than 65,536 characters in the world! Remember the large numbers that the emojis had in the last section? You'll need more than 16 bits to represent those values.

UTF-16 has a special way of encoding code points higher than 65,536 by using two code units called **surrogate pairs**.

```
const dart = '🎯';
print(dart.codeUnits);

// [55356, 57263]
```

The code point for 🎯 is 127919, but the surrogate pair for that in UTF-16 is 55356 and 57263. No one wants to mess with surrogate pairs. It would be much nicer to just get Unicode code points directly. And you can! Dart calls them **runes**

```
const dart = '🎯';
print(dart.runes);

// (127919)
```

Problem solved, right? If only it were.

Unicode grapheme clusters

Unfortunately, language is messy and so is Unicode. Have a look at this example:

```
const flag = '🇲🇳';
print(flag.runes);

// (127474, 127475)
```

Why are there two Unicode code points!? Well, it's because Unicode doesn't create a new code point every time there is a new country flag. It uses a pair of code points called **regional indicator symbols** to represent a flag. That's what you see in the example for the Mongolian flag above. 127474 is the code for the regional indicator symbol letter M, and 127475 is the code for the regional indicator symbol letter N. MN represents Mongolia.

If you thought that was complicated, look at this one:

```
const family = '👨‍👩‍👧‍👦';
print(family.runes);

// (128104, 8205, 128105, 8205, 128103, 8205, 12
8102)
```

That list of Unicode code points is a man, a woman, a girl and a boy all glued together with a characters called a **Zero Width Joiner ZWJ**

Now image trying to find the length of that string:

```
const family = '👨‍👩‍👧‍👦';

family.length;                  // 11
family.codeUnits.length; // 11
family.runes.length;        // 7
```

Getting the length of the string with `family.length` is equivalent to finding the number of UTF-16 code units: There are surrogate pairs for each of the four people plus the three ZWJ characters for a total of 11. Finding the runes gives you the seven Unicode code points that make up the emoji: man + ZWJ + woman + ZWJ + girl + ZWJ + boy. However, neither 11 nor 7 is what you'd expect. The family emoji looks like it's just one character. You'd think the length should be one!

When a string with multiple code points looks like a single character, this is known as a **user perceived character**. In technical terms it's called a **Unicode extended grapheme cluster** or more commonly, **grapheme cluster**

Although the creators of Dart didn't support grapheme clusters in the language itself, they did make an add-on package that handles them.

Adding the characters package

This is a good opportunity to try out your first Pub package. In the root folder of your project, open **pubspec.yaml**.

> **Note**: If you don't see **pubspec.yaml**, go back to Chapter 1 to see how to create a new project. Alternatively, open the starter project that comes with the supplemental materials for Chapter 3 of this book.

Find the line that says `dependencies:` and add the `characters` package and version directly below that. It should look like this when you're done:

```
dependencies:
  characters: ^1.1.0
```

Indentation is important in `.yaml` files, so make sure to indent the package name with two spaces. The ^ carat character means that any version higher than or equal to 1.1.0 but lower than 2.0.0 is OK to use in your project. This is known **semantic versioning**.

Now press **Command+S** on a Mac or **Control+S** on a PC to save the changes to **pubspec.yaml**. VS Code will automatically fetch the package from Pub. Alternatively, you can press the **Get Packages button** in the top right. It looks like a down arrow:

Both of these methods are equivalent to running the following command in the root folder of your project using the terminal:

```
dart pub get
```

> **Note**: Whenever you download and open a new Dart project that contains Pub packages, you'll need to run `dart pub get` first. This includes the final and challenge projects included in the supplemental materials for this chapter.

Now that you've added the `characters` package to your project, go back to your Dart code file and add the following import to the top of the page:

```
import 'package:characters/characters.dart';
```

Now you can use the code in the `characters` package to handle grapheme clusters. This package adds extra functionality to the String type.

```
const family = '👨‍👩‍👧';
family.characters.length; // 1
```

Aha! Now that's what you'd hope to see: just one character for the family emoji. The `characters` package extended `String` to include a collection of grapheme clusters called `characters`.

In your own projects, you can decide whether you want to work with UTF-16 code units, Unicode code points or grapheme clusters. However, as a general rule, you should strongly consider using grapheme clusters any time you're receiving text input from the outside world. That includes fetching data over the network or users typing things into your app.