Downloading DNA sequences as FASTA files in R

This is a modification of ["DNA Sequence Statistics"](https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/chapter1.html) from Avril Coghlan's [*A little book of R for bioinformatics.*](https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/index.html). Most of the text and code was originally written by Dr. Coghlan and distributed under the [Creative Commons 3.0](https://creativecommons.org/licenses/by/3.0/us/) license.

<!-- how much is sequinr actually useD? -->

<!-- download stringr -->

Preliminaries

We'll need the `dengueseq_fasta` FASTA data object, which is in the `compbio4all` package. We'll also use the `stringr` package for cleaning up the FASTA data, which can be downloaded with `install.packages("stringr")`

```{r}
# compbio4all, which has dengueseq_fasta
library(compbio4all)
data(dengueseq_fasta)

# stringr, for data cleaning
library(stringr)
```

Convert FASTA sequence to an R variable

We can't actually do much with the contents of the `dengueseq_fasta` we downloaded with the `rentrez` package except read them. If we want do address some biological questions with the data we need is to convert it into a data structure *R* can work with.

There are several things we need to remove:

1. The **meta data** line `>NC_001477.1 Dengue virus 1, complete genome` (metadata is "data" about data, such as where it came from, what it is, who made it, etc.).
1. All the `\n` that show up in the file (these are the **line breaks**).
1. Put each nucleotide of the sequence into its own spot in a vector.

There are functions in other packages that can do this automatically, but

1. I haven't found one I like, and
1. Walking through this will help you understand the types of operations you can do on text data.

I have my own FASTA cleaning function in compbio4all, fasta_cleaner(), and the code below breaks down some of the concepts and functions it uses.

The first two steps for cleaning up a FASTAS file involve removing things from the existing **character string** that contains the sequence. The third step will split the single continuous character string like "AGTTGTTAGTCTACGT..." into a **character vector** like `c("A","G","T","T","G","T","T","A","G","T","C","T","A","C","G","T"...)`, where each element of the vector is a single character a stored in a separate slot in the vector.

Removing unwanted characters

The second item is the easiest to take care of. *R* and many programming languages have tools called **regular expressions** that allow you to manipulate text. R has a function called `gsub()` which allows you to substitute or delete character data from a string. First I'll remove all those `\n` values.

The regular expression function `gsub()` takes three arguments:
1. `pattern = ...`. This is what we need it to find so we can replace it.
1. `replacement = ...`. The replacement.
1. `x = ...`. A character string or vector where `gsub()` will do its work.

<!-- TODO: check that this is rendering correcting with the number of slashes -->

We need to get rid of the `\n` so that we are left with only A, T and G, which are the actually information of the sequence. We want `\n` completely removed, so the replacement will be `""`, which is as set of quotation marks with nothing in the middle, which means "delete the target pattern and put nothing in its place."

One thing that is tricky about regular expressions is that many characters have special meaning to the functions, such as slashes, dollar signs, and brackets. So, if you want to find and replace one of these specially designated characters you need to put a slash in front of them. So when we set the pattern, instead of setting the pattern to a slash before an n `\n`, we have to give it two slashes `\\n`.

Here is the regular expression to delete the newline character \\n.
```{r}
 # note: we want to find all the \n, but need to set the pattern as \\n
denguesеq_vector <- gsub(pattern = "\\n",
                         replacement = "",
                         x = denguesеq_fasta)
```

We can use `strtrim()` to see if it worked

```{r}
strtrim(dengueseq_vector, 80)
```

Now for the metadata header. This is a bit complex, but the following code is going to take all the that occurs before the beginning of the sequence ("AGTTGTTAGTC") and delete it.

First, I'll define what I want to get rid of in an *R* object. This will make the **call** to `gsub()` a little cleaner to read
```{r}
seq.header <- ">NC_001477.1 Dengue virus 1, complete genome"
```


Now I'll get rid of the header with with `gsub()`.
```{r}
dengueseq_vector <- gsub(pattern = seq.header, # object defined above
                         replacement = "",
                         x = dengueseq_vector)
```


See if it worked:
```{r}
strtrim(dengueseq_vector, 80)
```

Spliting unbroken strings in character vectors

Now the more complex part. We need to split up a continuous, unbroken string of letters into a vector where each letter is on its own. This can be done with the `str_split()` function ("string split") from the `stringr` package. The notation `stringr::str_split()` mean "use the `str_split` function from from the `stringr` package." More specifically, it temporarily loads the `stringr` package and gives R access to just the `str_split` function. These allows you to call a single function without loading the whole library.

There are several arguments to `str_split`, and I've tacked a `[[1]]` on to the end.

First, run the command

```{r}
dengueseq_vector_split <- stringr::str_split(dengueseq_vector,
                                             pattern = "",
                                             simplify = FALSE)[[1]]
```

Look at the output with str()
```{r}
```

```
str(dengueseq_vector_split)
```

We can explore what the different arguments do by modifying them. Change `pattern = "" to pattern = "A"`. Can you figure out what happened?

```{r}
# re-run the command with "pattern  = "A"
dengueseq_vector_split2 <- stringr::str_split(dengueseq_vector,
                                              pattern = "A",
                                              simplify = FALSE)[[1]]
str(dengueseq_vector_split2)
```

And try it with `pattern = "" to pattern = "G"`.

```{r}
# re-run the command with "pattern  = "G"
dengueseq_vector_split3 <- stringr::str_split(dengueseq_vector,
                                              pattern = "G",
                                              simplify = FALSE)[[1]]
str(dengueseq_vector_split3)
```

Run this code to compare the two ways we just used `str_split` (don't worry what it does). Does this help you see what's up?

```{r}
options(str = strOptions(vec.len = 10))
str(list(dengueseq_vector_split[1:20],
     dengueseq_vector_split2[1:10],
     dengueseq_vector_split3[1:10]))
```

So, what does the `pattern = ... ` argument do? For more info open up the help file for `str_split` by calling `?str_split`.

Something cool which we will explore in the next exercise is that we can do summaries on vectors of nucleotides, like this:

```{r}
table(dengueseq_vector_split)
```