```
---
output: html_document
editor_options:
  chunk_output_type: console
---
```

Downloading DNA sequences as FASTA files in R {#download-FASTA-inR}

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? -->

NOTE: There is some reduncnacy in this current draft that needs to be eliminated.

Functions

* library()
* help()
* length
* table
* seqinr::GC()
* seqinr::count()
* seqinr::write.fasta()

Software/websites

* www.ncbi.nlm.nih.gov
* Text editors (e.g. Notepad++, TextWrangler)

R vocabulary

* list
* library
* package
* CRAN
* wrapper

File types

* FASTA

Bioinformatics vocabulary

* accession, accession number
* NCBI
* NCBI Sequence Database
* EMBL Sequence Database
* FASTA file

Organisms and Sequence accessions

* Dengue virus: DEN-1, DEN-2, DEN-3, and DEN-4.

The NCBI accessions for the DNA sequences of the DEN-1, DEN-2, DEN-3, and DEN-4 Dengue viruses are NC_001477, NC_001474, NC_001475 and NC_002640, respectively.

According to Wikipedia

> "Dengue virus (DENV) is the cause of dengue fever. It is a mosquito-borne, single positive-stranded RNA virus ... Five serotypes of the virus have been found, all of which can cause the full spectrum of disease. Nevertheless, scientists' understanding of dengue virus may be simplistic, as rather than distinct ... groups, a continuum appears to exist."
https://en.wikipedia.org/wiki/Dengue_virus

Preliminaries

```{r}
library(rentrez)
library(compbio4all)
```

DNA Sequence Statistics: Part 1

Using R for Bioinformatics

The chapter will guide you through the process of using R to carry out simple analyses that are common in bioinformatics and computational biology. In particular, the focus is on computational analysis of biological sequence data such as genome sequences and protein sequences. The programming approaches, however, are broadly generalizable to statistics and data science.

The tutorials assume that the reader has some basic knowledge of biology, but not necessarily of bioinformatics. The focus is to explain simple bioinformatics analysis, and to explain how to carry out these analyses using *R*.

R packages for bioinformatics: Bioconductor and SeqinR

Many authors have written *R* packages for performing a wide variety of analyses. These do not come with the standard *R* installation, but must be installed and loaded as “add-ons”.

<!-- old Loading the dayoff package will automatically load most of the packages we need. -->
<!-- have these as dependencies of combio4all? -->

Bioinformaticians have written numerous specialized packages for *R*. In this tutorial, you will learn to use some of the function in the [`SeqinR`](https://cran.r-project.org/web/packages/seqinr/index.html) package to to carry out simple analyses of DNA sequences. (`SeqinR` can retrieve sequences from a DNA sequence database, but this has largely been replaced by the functions in the package `rentrez`)

Many well-known bioinformatics packages for *R* are in the Bioconductor set of *R* packages (www.bioconductor.org), which contains packages with many *R* functions for analyzing biological data sets such as microarray data. The [`SeqinR`](https://cran.r-project.org/web/packages/seqinr/index.html) package is from CRAN, which contains R functions for obtaining sequences from DNA and protein sequence databases, and for analyzing DNA and protein sequences.

<!-- SequinR will automatically load when you load dayoff.   For instructions on how to install an R package on your own see [How to install an R package](https://a-little-book-of-r-for-bioinformatics.readthedocs.io/en/latest/src/installr.html). -->

We will also use functions from the `rentrez` and `ape` packages.

<!-- Both of these also are loaded automatically by the dayoff package. -->
<!-- ape still used?. -->

Remember that you can ask for more information about a particular *R* command by using the `help()` function. For example, to ask for more information about the `library()`, you can type:

```{r, eval = F}
help("library")
```

You can also do this
```{r, eval = F}
?library
```

FASTA file format

The FASTA format is a simple and widely used format for storing biological (e.g. DNA or protein) sequences. It was first used by the [FASTA program](https://en.wikipedia.org/wiki/FASTA) for sequence alignment in the 1980s and has been adopted as standard by many other programs.

FASTA files begin with a single-line description starting with a greater-than sign `>` character, followed on the next line by the sequences. Here is an example of a FASTA file. (If you're looking at the source script for this lesson you'll see the `cat()` command, which is just a text display function used format the text when you run the code).

```{r, eval = T, echo = F}
cat(">A06852 183 residues
MPRLFSYLLGVWLLLSQLPREIPGQSTNDFIKACGRELVRLWVEICGSVSWGRTALSLEEPQLETGPPAETMPSSITKDAE
ILKMMLEFVPNLPQELKATLSERQPSLRELQQSASKDSNLNFEEFKKIILNRQNEAEDKSLLELKNLGLDKHSRKKRLFRM
TLSEKCCQVGCIRKDIARLC")
```

The NCBI sequence database

The US [National Centre for Biotechnology Information (NCBI)](www.ncbi.nlm.nih.gov) maintains the **NCBI Sequence Database**, a huge database of all the DNA and protein sequence data that has been collected. There are also similar databases in Europe, the [European Molecular Biology Laboratory (EMBL) Sequence Database](www.ebi.ac.uk/embl), and Japan, the [DNA Data Bank of Japan (DDBJ)](www.ddbj.nig.ac.jp). These three databases exchange data every night, so at any one point in time, they contain almost identical data.

Each sequence in the NCBI Sequence Database is stored in a separate **record**, and is assigned a unique identifier that can be used to refer to that record. The identifier is known as an **accession**, and consists of a mixture of numbers and letters.

For example, Dengue virus causes Dengue fever, which is classified as a **neglected tropical disease** by the World Health Organization (WHO), is classified by any one of four types of Dengue virus: DEN-1, DEN-2, DEN-3, and DEN-4. The NCBI accessions for the DNA sequences of the DEN-1, DEN-2, DEN-3, and DEN-4 Dengue viruses are

* NC_001477
* NC_001474
* NC_001475
* NC_002640

Note that because the NCBI Sequence Database, the EMBL Sequence Database, and DDBJ exchange data every night, the DEN-1 (and DEN-2, DEN-3, DEN-4) Dengue virus sequence are present in all three databases, but they have different accessions in each database, as they each use their own numbering systems for referring to their own sequence records.

Retrieving genome sequence data using rentrez

You can retrieve sequence data from NCBI directly from *R* using the `rentrez` package. The DEN-1 Dengue virus genome sequence has NCBI accession NC_001477. To retrieve a sequence with a particular NCBI accession, you can use the function `entrez_fetch()` from the `rentrez` package. Note that to be specific where the function comes from I write it as `package::function()`.

<!-- QUESTION: How many arguments does entrez_fetch take in this call? -->
```{r, eval = F}
dengueseq_fasta <- rentrez::entrez_fetch(db = "nucleotide", 
                          id = "NC_001477", 
                          rettype = "fasta")
```

```{r, echo = F}
data(dengueseq_fasta)
```

Note that the "_" in the name is just an arbitrary way to separate two words. Another common format would be `dengueseq.fasta`. Some people like `dengueseqFasta`, called **camel case** because the capital letter makes a hump in the middle of the word. Underscores are becoming most common and are favored by developers associated with RStudio and the **tidyverse** of packages that many data scientists use. I switch between "." and "_" as separators, usually favoring "_" for function names and "." for objects; I personally find camel case harder to read and to type.

Ok, so what exactly have we done when we made `dengueseq_fasta`? We have an R object `dengueseq_fasta` which has the sequence linked to the accession number "NC_001477." So where is the sequence, and what is it?

First, what is it?

~~~
```{r}
is(dengueseq_fasta)
class(dengueseq_fasta)
```
~~~

How big is it? Try the `dim()` and `length()` commands and see which one works. Do you know why one works and the other doesn't?

~~~
<!-- question:1 which one works -->
```{r}
dim(dengueseq_fasta)
length(dengueseq_fasta)
```
~~~

The size of the object is 1. Why is this? This is the genomics sequence of a virus, so you'd expect it to be fairly large. We'll use another function below to explore that issue. Think about this first: how many pieces of unique information are in the `dengueseq` object? In what sense is there only _one_ piece of information?

If we want to actually see the sequence we can type just type `dengueseq_fasta` and press enter. This will print the WHOLE genomic sequence out but it will probably run of your screen.

~~~
```{r eval = F}
dengueseq_fasta
```
~~~

This is a whole genome sequence, but its stored as single entry in a vector, so the `length()` command just tells us how many entries there are in the vector, which is just one! What this means is that the entire genomic sequence is stored in a single entry of the vector `dengueseq_fasta`. (If you're not following along with this, no worries - its not essential to actually working with the data)

If we want to actually know how long the sequence is, we need to use the function `nchar()`.

~~~
```{r}
nchar(dengueseq_fasta)
```
~~~

The sequence is 10935 bases long. All of these bases are stored as a single **character string** with no spaces in a single entry of our `dengueseq_fasta` vector. This isn't actually a useful format for us, so below were' going to convert it to something more useful.

If we want to see just part of the sequence we can use the `strtrim()` function. Before you run the code below, predict what the 100 means.

~~~
```{r, echo = T}
strtrim(dengueseq_fasta, 100)
```
~~~

Note that at the end of the name is a slash followed by an n, which indicates to the computer that this is a **newline**; this is read by text editor, but is ignored by R in this context.

```{r, echo = T}
strtrim(dengueseq_fasta, 45)
```

After the `\\n` begins the sequence, which will continue on for a LOOOOOONG way. Let's just print a little bit.

```{r, echo = T}
strtrim(dengueseq_fasta, 52)
```

Let's print some more. Do you notice anything beside A, T, C and G in the sequence?

```{r}
strtrim(dengueseq_fasta, 200)
```

Again, there are the `\\n` newline characters, which tell text editors and wordprocessors how to display the file.

Now that we a sense of what we're looking at let's explore the `dengueseq_fasta` a bit more.

We can find out more information about what it is using the `class() `command.

```{r}
class(dengueseq_fasta)
```

As noted before, this is character data.

Many things in R are vectors so we can ask *R *`is.vector()`

```{r}
is.vector(dengueseq_fasta)
```

Yup, that's true.

Ok, let's see what else. A handy though often verbose command is `is()`, which tells us what an object, well, what it is:

```{r}
is(dengueseq_fasta)
```

There is a lot here but if you scan for some key words you will see "character" and "vector" at the top. The other stuff you can ignore. The first two things, though, tell us the dengueseq_fasta is a **vector** of the class **character**: that is, a **character vector**.

Another handy function is `str()`, which gives us a peak at the context and structure of an *R* object. This is most useful when you are working in the R console or with dataframes, but is a useful function to run on all *R* objects. How does this output differ from other ways we've displayed dengueseq_fasta?

```{r}
str(dengueseq_fasta)
```

We know it contains character data - how many characters? `nchar()` for "number of characters" answers that:

```{r}
nchar(dengueseq_fasta)
```

OPTIONAL: Saving FASTA files

We can save our data as .fasta file for safe keeping. The `write()` function will save the data we downloaded as a plain text file.

```{r, eval = F}
write(dengueseq_fasta,
      file="dengueseq.fasta")
```

If you do this, you'll need to figure out where *R* is saving things, which requires and understanding *R's* **file system**, which can take some getting used to, especially if you're new to programming. As a start, you can see where *R* saves things by using the `getwd()` command, which tells you where on your hard drive R currently is using as its home base for files.

```{r}
getwd()
```

Next steps

On their own, FASTA files in R are not directly useful. In the next lesson we'll process our `dengueseq_fasta` file so that we can use it in analyses.