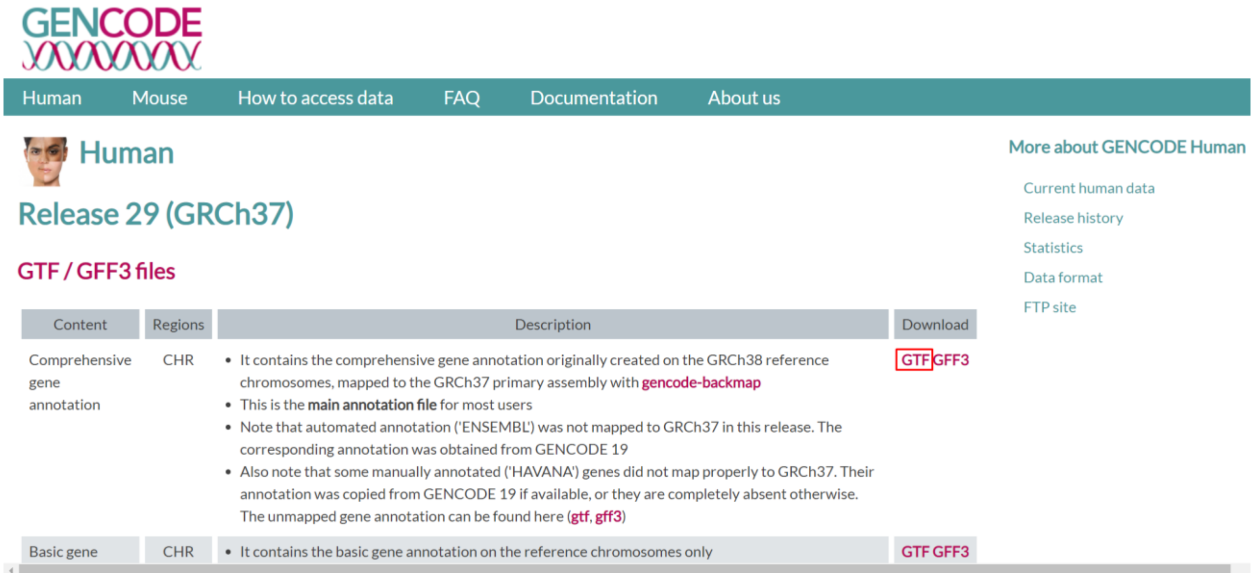
Homework Exercise 3

1)

Download **gencode.v29lift37.annotation.gtf.gz** from GENCODE at: https://www.gencodegenes.org/human/release_29lift37.html.



Human

Release 29 (GRCh37)

GTF / GFF3 files

Content	Regions	Description	Download
Comprehensive gene annotation	CHR	• It contains the comprehensive gene annotation originally created on the GRCh38 reference chromosomes, mapped to the GRCh37 primary assembly with **gencode-backmap** • This is the **main annotation file** for most users • Note that automated annotation ('ENSEMBL') was not mapped to GRCh37 in this release. The corresponding annotation was obtained from GENCODE 19 • Also note that some manually annotated ('HAVANA') genes did not map properly to GRCh37. Their annotation was copied from GENCODE 19 if available, or they are completely absent otherwise. The unmapped gene annotation can be found here (**gtf**, **gff3**)	GTF GFF3
Basic gene	CHR	• It contains the basic gene annotation on the reference chromosomes only	GTF GFF3

GENCODE provides gene annotations for the human reference genome (GRCh37 in this case). It's a compressed CSV file with tab delimiter (look at this week's notebook for an example code dealing with this file).

A)

According to this annotation file, how many protein-coding and miRNA genes exist in each chromosome?

Write all the protein-coding genes in each chromosome as a single JSON file.

B)

According to the same file, what is the average exon length of protein-coding genes per chromosome?

C)

What is the longest intergenic region in the human reference genome (according to this version of GENCODE’s annotations)?

Why is the notion of “intergenic region” somewhat tricky to define? How likely do you think it is that your answer would be different were you to work with a different annotation database?

Bonus Questions

1)

What does the **yield** statement do in Python? What does the **iter** function do? Why do **range**, **map** and **filter** no longer return lists in Python 3?

All of these relate to a general concept in Python called “generators”. Why are generators useful?

Which of the following options is better:

```
sum([x ** 4 for x in range(10 ** 7)])
```

Or:

```
sum(x ** 4 for x in range(10 ** 7))
```

2)

Consider the following code:

```
x = 0.1
y = 0.2
z = 0.1 + 0.2
print(x, y, z, x == 0.1, y == 0.2, z == 0.3, sep = '\n')
```

What do you expect the output to be? What is the actual output? Explain the floating point representation of fractions, what issues it may present, and suggest ways to more reliably work with fractions in Python.