

# Sequence Ranges

INTRODUCTION TO BIOCONDUCTOR IN R



**Paula Andrea Martinez, PhD.**

Data scientist

# IRanges with numeric arguments

```
# Loading IRanges  
library(IRanges)
```

A range is defined by `start` and `end`

```
myIRanges <- IRanges(start = 20, end = 30)  
myIRanges
```

```
IRanges object with 1 range and 0 metadata columns:
```

```
      start      end      width  
<integer> <integer> <integer>  
[1]    20      30      11
```

```
(myIRanges_width <- IRanges(start = c(1, 20), width = c(30, 11)))
```

IRanges object with 2 ranges and 0 metadata columns:

	start	end	width
	<integer>	<integer>	<integer>
[1]	1	30	30
[2]	20	30	11

```
(myIRanges_end <- IRanges(start = c(1, 20), end = 30))
```

IRanges object with 2 ranges and 0 metadata columns:

	start	end	width
	<integer>	<integer>	<integer>
[1]	1	30	30
[2]	20	30	11

Equation:  $\text{width} = \text{end} - \text{start} + 1$

# Rle - run length encoding

- Rle stands for Run length encoding
- Computes and stores the **lengths** and **values** of a vector or factor
- Rle is general S4 container used to save long repetitive vectors efficiently

```
(some_numbers <- c(3, 2, 2, 2, 3, 3, 4, 2))
```

```
3 2 2 2 3 3 4 2
```

```
(Rle(some_numbers))
```

```
numeric-Rle of length 8 with 5 runs  
Lengths: 1 3 2 1 1  
Values : 3 2 3 4 2
```

# IRanges with logical vector

```
IRanges(start = c(FALSE, FALSE, TRUE, TRUE))
```

```
IRanges object with 1 range and 0 metadata columns:
```

	start	end	width
	<integer>	<integer>	<integer>
[1]	3	4	2

# IRanges with logical Rle

```
gi <- c(TRUE, TRUE, FALSE, FALSE, TRUE, TRUE, TRUE)
myRle <- Rle(gi)
```

```
logical-Rle of length 7 with 3 runs
Lengths:      2      2      3
Values :  TRUE FALSE  TRUE
```

```
IRanges(start = myRle)
```

```
IRanges object with 2 ranges and 0 metadata columns:
```

	start	end	width
	<integer>	<integer>	<integer>
[1]	1	2	2
[2]	5	7	3

# In summary

IRanges are hierarchical data structures can contain metadata.

To construct IRanges objects:

- `start` , `end` , or `width` as numeric vectors (or NULL).
- `start` argument as a logical vector or logical Rle object.
  - Rle stands for Run length encoding and is storage efficient.
  - IRanges arguments get recycled (fill in the blanks).
  - equation for sequence range:  $\text{width} = \text{end} - \text{start} + 1$  .

# Let's practice using sequence ranges!

INTRODUCTION TO BIOCONDUCTOR IN R



# Gene of interest

INTRODUCTION TO BIOCONDUCTOR IN R



**Paula Andrea Martinez, PhD.**  
Data Scientist

# Examples of genomic intervals

- Reads aligned to a reference
- Genes of interest
- Exonic regions
- Single nucleotide polymorphisms (SNPs)
- Regions of transcription or binding sites, RNA-seq or ChIP-seq

# Genomic Ranges

```
library(GenomicRanges)
(myGR <- GRanges("chr1:200-300"))
```

```
GRanges object with 1 range and 0 metadata columns:
  seqnames      ranges strand
   <Rle>   <IRanges>  <Rle>
[1]    chr1 [200, 300]    *
-----
seqinfo: 1 sequence from an unspecified genome; no seqlengths
```

- GRanges class is a container to save genomic intervals by chromosome
- Minimum arguments `chr1:200-300`
- GRanges `seqnames` and `seqinfo`

```
# df a data.frame like structure
  seqnames start end strand score GC
1    chrX    50 120      +     1 0.25
2    chrX   130 140      +     2 0.25
3    chrX   153 154      +     3 0.25
4    chrY    30  40      *     4 0.25
5    chrY    50  55      -     5 0.25
```

```
(myGR <- as(df, "GRanges")) # transform df into GRanges
```

```
GRanges object with 5 ranges and 2 metadata columns:
      seqnames      ranges strand |      score      GC
      <Rle>  <IRanges>  <Rle> | <integer> <numeric>
[1]    chrX [ 50, 120]      + |         1     0.25
[2]    chrX [130, 140]      + |         2     0.25
[3]    chrX [153, 154]      + |         3     0.25
[4]    chrY [ 30,  40]      * |         4     0.25
[5]    chrY [ 50,  55]      - |         5     0.25
-----
seqinfo: 2 sequences from an unspecified genome; no seqlengths
```

# Genomic Ranges accessors

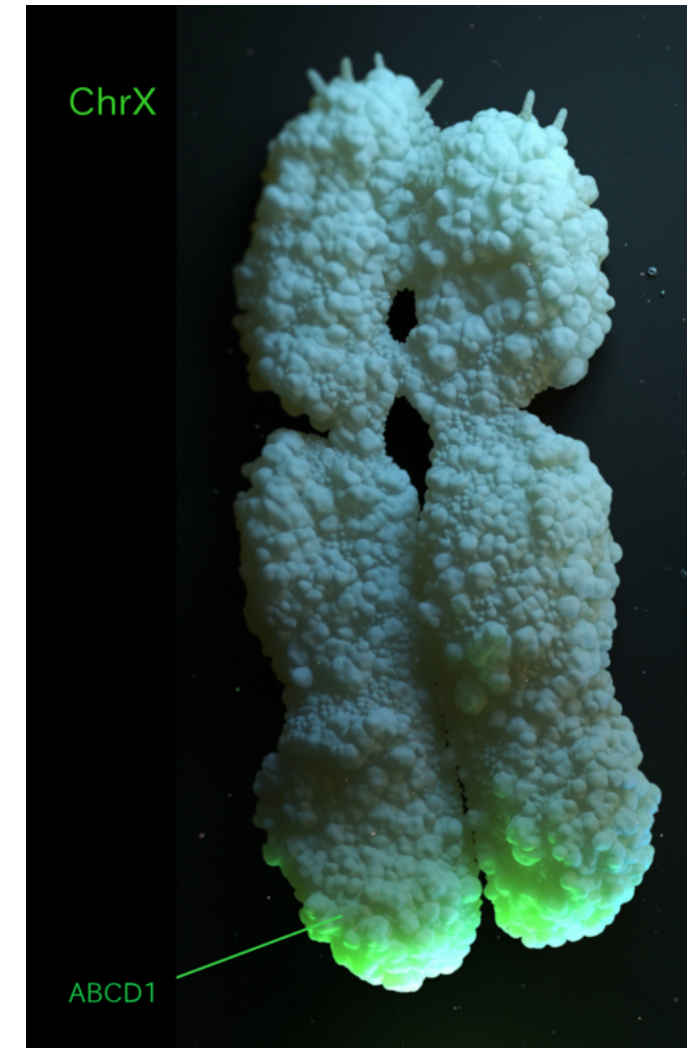
```
methods(class = "GRanges") # to check available accessors
# used for chromosome names
seqnames(gr)
# returns an IRanges object for ranges
ranges(gr)
# stores metadata columns
mcols(gr)
# generic function to store sequence information
seqinfo(gr)
# stores the genome name
genome(gr)
```

- Accessors are both setter and getter functions
- Accessors can be inherited thanks to S4 definitions

# Gene of interest: ABCD1

- ABCD1 is located at the end of chromosome X long arm
- encodes a protein relevant for the well functioning of brain and lung cells in mammals
- chrX is ~ 156 mi bp
- Located chrX ~ 153.70 mi bp

<https://www.ncbi.nlm.nih.gov/gene/215>



# Chromosome X GRanges

```
library(TxDb.Hsapiens.UCSC.hg38.knownGene)
hg <- TxDb.Hsapiens.UCSC.hg38.knownGene
```

Select genes from chromosome X

```
hg_chrXg <- genes(hg, filter = list(tx_chrom = c("chrX")))
```

GRanges object with 1192 ranges and 1 metadata column:

	seqnames	ranges	strand	gene_id
	<Rle>	<IRanges>	<Rle>	<character>
100008586	chrX	49551278-49568218	+	100008586
10009	chrX	120250752-120258398	+	10009
100093698	chrX	13310652-13319933	+	100093698
...	...	...	...	...

-----

seqinfo: 640 sequences (1 circular) from hg38 genome

**Let's practice  
looking for a gene of  
interest in the  
human genome!**

**INTRODUCTION TO BIOCONDUCTOR IN R**



# Manipulating collections of GRanges

INTRODUCTION TO BIOCONDUCTOR IN R



**Paula Andrea Martinez, PhD.**  
Data Scientist

# GRangesList

- The `GRangesList`-class is a container for storing a collection of `GRanges`
  - Efficient for storing a large number of elements.
- To construct a `GRangesList`
  - `as(mylist, "GRangesList")`
  - `GRangesList(myGranges1, myGranges2, ...)`
- To convert back to `GRanges`
  - `unlist(myGRangesList)`
- Accessors `methods(class = "GRangesList")`

# When to use lists?

- Multiple GRanges objects may be combined into a GRangesList
  - GRanges in a list will be taken as compound features of a larger object
- Examples of GRangesLists are
  - transcripts by gene
  - exons by transcripts
  - read alignments
  - sliding windows

```
# GRanges object with 983 genes
hg_chrX
slidingWindows(hg_chrX, width = 20000, step = 10000)
```

```
# showing only two elements of the list
GRangesList object of length 983:
[[1]]
GRanges object with 2 ranges and 0 metadata columns:
      seqnames      ranges strand
   <Rle>      <IRanges>  <Rle>
[1] chrX [276322, 296321]      +
[2] chrX [286322, 303356]      +
[[2]]
GRanges object with 3 ranges and 0 metadata columns:
      seqnames      ranges strand
   <Rle>      <IRanges>  <Rle>
[1] chrX [624344, 644343]      +
[2] chrX [634344, 654343]      +
[3] chrX [644344, 659411]      +
...
```

**GenomicFeatures** uses transcript database ( **TxDb** ) objects to **store metadata, manage genomic locations and relationships between features and its identifiers.**

```
library(TxDb.Hsapiens.UCSC.hg38.knownGene)
(hg <- TxDb.Hsapiens.UCSC.hg38.knownGene)
```

```
Db type: TxDb
Supporting package: GenomicFeatures
Data source: UCSC
Genome: hg38
Organism: Homo sapiens
Taxonomy ID: 9606
Resource URL: http://genome.ucsc.edu/
Type of Gene ID: Entrez Gene ID
transcript_nrow: 197782
exon_nrow: 581036
cds_nrow: 293052
Db created by: GenomicFeatures package from Bioconductor
Creation time: 2016-09-29 13:02:09 +0000 (Thu, 29 Sep 2016)
```

# Genes, transcripts, exons

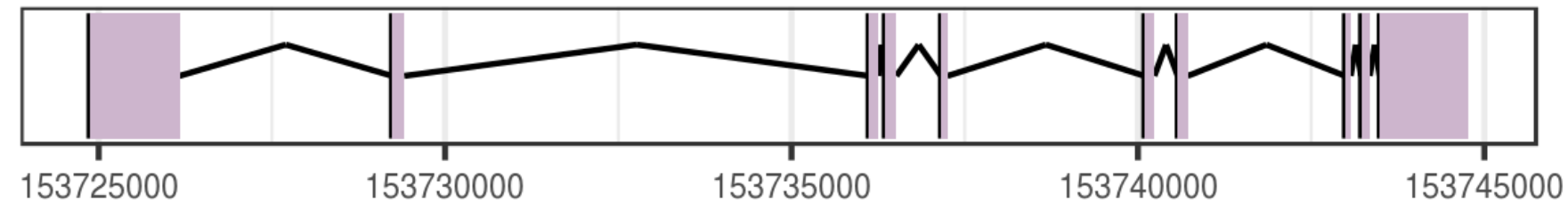
```
library(TxDb.Hsapiens.UCSC.hg38.knownGene)
hg <- TxDb.Hsapiens.UCSC.hg38.knownGene # hg is a TxDb object
seqlevels(hg) <- c("chrX") # prefilter results to chrX
# transcripts
transcripts(hg, columns = c("tx_id", "tx_name"), filter = NULL)
# exons
exons(hg, columns = c("tx_id", "exon_id"), filter = list(tx_id = "179161"))
```

`columns` and `filter` can be `NULL` or any of these:

```
"gene_id", "tx_id", "tx_name", "tx_chrom", "tx_strand",
"exon_id", "exon_name", "exon_chrom", "exon_strand",
"cds_id", "cds_name", "cds_chrom", "cds_strand" and "exon_rank"
```

# Exons by transcripts

ABCD1 exons



```
hg <- TxDb.Hsapiens.UCSC.hg38.knownGene
seqlevels(hg) <- c("chrX") # prefilter chromosome X
exonsBytx <- exonsBy(hg, by = "tx") # exons by transcript
abcd1_179161 <- exonsBytx[["179161"]] # transcript id
width(abcd1_179161) # width of each exon, the purple regions of the figure
```

```
1299  181  143  169   95  146  146   85  126 1274
```

# Overlaps

```
# countOverlaps results in an integer vector of counts
```

```
countOverlaps(query, subject)
```

```
# findOverlaps results in a Hits object
```

```
findOverlaps(query, subject)
```

```
# subsetByOverlaps returns a GRangesList object
```

```
subsetByOverlaps(query, subject)
```

- Query and subject are either a GRanges or GRangesList objects.
- Overlaps might be complete all partial.



**It's your turn to put  
this into practice!**

INTRODUCTION TO BIOCONDUCTOR IN R