Atomicity
Vectors are atomic structures
Examples

\[ x \leftarrow c(1, 2, 3, 4, 5) \]

\[ y \leftarrow c("one", "two", "three") \]

\[ z \leftarrow c(\text{TRUE}, \text{FALSE}, \text{TRUE}) \]
Atomic vectors

Vectors are atomic structures

The values in a vector must be **ALL** of the same type!

Either all integers, or reals, or complex, or characters, or logicals

You **CANNOT** have a vector of different data types
Coercion
What happens if you mix different data values in a vector?
Mixing data types within a vector?

\[
x <- c(1, 2, 3, "four", "five")
\]

\[
y <- c(TRUE, FALSE, 3, 4)
\]

\[
z <- c(TRUE, 1L, 2 + 3i, pi)
\]
Implicit Coercion

If you mix different data values, R will *implicitly coerce* them so they are ALL of the same type

```r
x <- c(1, 2, 3, "four", "five")
```

```r
y <- c(TRUE, FALSE, 3, 4)
```
How does R coerce data types in vectors?

R follows two basic rules of implicit coercion:

1) If a character is present, R will coerce everything else to characters.

2) If a vector contains logicals and numbers, R will convert the logicals to numbers (TRUE to 1, FALSE to 0).
Hierarchy of data types

Logical < Integer < Double < Character
Coercion functions

R provides a set of explicit coercion functions that allow you to “convert” one type of data into another

- `as.character()`
- `as.numeric()`
- `as.double()`
- `as.integer()`
- `as.logical()`
Vectorization
Vectorization

A vectorized computation is any computation that when applied to a vector operates on all of its elements

$$\mathbf{c}(1, 2, 3) + \mathbf{c}(3, 2, 1)$$

$$\mathbf{c}(1, 2, 3) \times \mathbf{c}(3, 2, 1)$$

$$\mathbf{c}(1, 2, 3) ^ \mathbf{c}(3, 2, 1)$$
Vectorized code
Recycling
Recycling

When vectorized computations are applied, some conflicts may occur when dealing with two vectors of different length

\[ \mathbf{c}(2, 1) + \mathbf{c}(1, 2, 3) \]

\[ \mathbf{c}(1, 2, 3, 4) + \mathbf{c}(1, 2) \]
Recycling Rule

The recycling rule can be very useful, like when operating between a vector and a “scalar”

```r
x <- c(2, 4, 6, 8)
x + 3
```

18
Recycling (and vectorization)

\[
\begin{align*}
1 & \quad 2 & \quad 3 & \quad 4 \\
+ & \quad + & \quad + & \quad + \\
3 & \quad 3 & \quad 3 & \quad 3
\end{align*}
\]

\[+ 3\]

\[
\begin{align*}
1 & \quad 2 & \quad 3 & \quad 4 \\
+ & \quad + & \quad + & \quad + \\
3 & \quad 3 & \quad 3 & \quad 3
\end{align*}
\]

\[=\]

\[
\begin{align*}
4 & \quad 5 & \quad 6 & \quad 7
\end{align*}
\]
Recycling (and vectorization)

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
+ & + & + & + \\
2 & 4 & 2 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
2 & 4 \\
\end{array}
\]

= \[
\begin{array}{ccccc}
3 & 6 & 5 & 8 \\
\end{array}
\]
Recycling (and vectorization)

\[
\begin{array}{ccc}
1 & 2 & 3 \\
+ & + & + \\
2 & 4 & 2 \\
\end{array} + \begin{array}{cc}
2 & 4 \\
\end{array} = \begin{array}{ccc}
3 & 6 & 5 \\
\end{array}
\]
Subsetting and Indexing
Bracket notation for vectors

$\text{vec}[\text{index}]$
Bracket Notation System

To extract values from R objects use brackets: [ ]

Inside the brackets specify vector(s) of indices

Use as many indices, separated by commas, as dimensions in the object

Vector(s) of indices can be numbers, logicals, and sometimes characters
Bracket Notation System

# some vector
x <- c(2, 4, 6, 8)

# adding names
names(x) <- letters[1:4]
Numeric index

# first element
\[x[1]\]

# second element
\[x[2]\]

# last element
\[x[length(x)]\]
Numeric index

# first 3 elements
x[1:3]

# non-consecutive elements
x[c(1, 3)]

# different order
x[c(3, 2, 4, 1)]
Logical index

# first element
x[c(TRUE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE)]

# elements equal to 2
x[x == 2]

# elements different to 2
x[x != 2]
Character index

# element names "a"
\( x["a"] \)

# "b" and "d"
\( x[c("b", "d")]) \)

# what about this?
\( x[\text{rep("a", 5)}] \)
Logical index

# elements greater than 1
x[x > 1]

# try this
x[TRUE]

# what about this?
x[as.logical(c(0, 1, pi, -10))]