mattm / **r-cheat-sheet**

Branch: master ▾ r-cheat-sheet / Vectors.md Find file Copy path

mattm Update notes aaca237 on 2 May 2017

1 contributor

234 lines (184 sloc) 3.14 KB

Vectors

- All elements in a vector must have the same mode, which can be integer, numeric (floating-point number), character (string), logical (Boolean), complex, and so on. - The Art of R Programming
- R indeces begin at 1, not 0.
- Scalars are 1 element vectors.

Creating a vector

```
> x <- c(88, 5, 12, 13)
```

"We created a 4 element vector and assigned it to `x` "

The concatenate function, `c` , flattens vectors:

```
> x <- c(80, c(90, 100))
> x
[1] 80 90 100
```

Note that `NULL` values get removed:

```
> x <- c(NULL, 4)
> x
> [1] 4
```

Accessing elements in a vector

One value:

```
> x[1]
> [1] 88
```

Multiple sequential values:

```
> x[1:2]
> [1] 88 5
```

Multiple non-sequential values:

```
> x[c(1, 3)]
> [1] 88 12
```

Excluding elements from a vector

Use negative indeces:

```
> x[-1]
> [1] 5 12 13
```

Determining the length of a vector

```
> length(x)
[1] 4
```

All elements except the last

```
> x[1:length(x) - 1]
[1] 88  5 12
```

Adding vectors together

Functions will be applied element-wise:

```
> x <- c(1, 2, 4) + c(5, 0, -1)
[1] 6 2 3
```

Sequences

```
> seq(10)
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
> seq(2, 10)
 [1]  2  3  4  5  6  7  8  9 10
```

```
> seq(2, 10, by = 2)
[1]  2  4  6  8 10
```

Generating a vector with repeating elements

```
> rep(5, 3)
[1] 5 5 5
```

Any and all

```
> x <- 1:10
> any(x > 8)
[1] TRUE
> any(x > 88)
[1] FALSE
> all(x > 88)
[1] FALSE
> all(x > 0)
[1] TRUE
```

Removing NA values

```
> x <- c(4, NA, 6)
> x[!is.na(x)]
[1] 4 6
```

Functions are vectorized

Meaning they are applied to each element individually:

```
> x <- c(1.4, 2.6)
> round(x)
[1] 1 3
```

Remember that scalars are actually just single-element vectors:

```
> x <- 4.2
> round(x)
[1] 4
```

Filtering

```
> x <- 1:3
> x[c(TRUE, FALSE, TRUE)]
[1] 1 3

> x >= 2
[1] FALSE TRUE TRUE
> x[x >= 2]
[1] 2 3

# Which is the same as the following because 2 is actually a vector and the elements are repeated when comparing
them element-wise to the other vector:

> x[x >= c(2, 2, 2)]
[1] 2 3

# One more element-repeating example:

> x <- c(4, 5, 6, 7)
> x[x >= c(5, 6)]
[1] 6 7

# Which is the same as:

> x <- c(4, 5, 6, 7)
> x[x >= c(5, 6, 5, 6)]
[1] 6 7
```

We can also assign values to certain vector elements:

~~~
> x <- c(1,3,8,2,20)
> x[x > 3] <- 0
> x
[1] 1 3 0 2 0
```## Filtering with the `subset` function
`subset` doesn't return `NA` values:
~~~

> x <- c(6, 1:3, NA, 12) x [1] 6 1 2 3 NA 12 x[x > 5] [1] 6 NA 12 subset(x, x > 5) [1] 6 12 ```## Finding indeces using the `which` function `which` returns the indeces of a vector that satisfy a certain condition:

```
> x <- c(1, 5, 10)
> which(x > 4)
[1] 2 3
```

The `ifelse` function

```
> x <- c(5, 2, 9, 12)
> ifelse(x > 6, 2 * x, 3 * x)
[1] 15 6 18 24
```

Vector element names

```
> grades <- c(80, 90, 100)
> names(grades) <- c("John", "Alex", "Tracy")
> grades
 John  Alex Tracy
   80    90   100
> grades[1]
John
  80
> grades["John"]
 John
   80
```

Removing the names:

```
> names(grades) <- NULL
> grades
[1]  80  90 100
```

Counting how many elements are true

You can use `sum`:

```
> sum(c(TRUE, FALSE, TRUE))
[1] 2
```