lab

Data Types

Vector, matrices, data frames.

How to create a vector

Let's say you want to create a vector `x = [2, 10, 11, 1, 23]`. The easiest way to create this vector is through the `c` function:

```
x <- c(2, 10, 11, 1, 23)
```

In case you want to know what's the current value of variable x, you can either just add the command `x`. Alternative, you can use `print(x)`:

```
x # this will print the current value of x
```

```
## [1]  2 10 11  1 23
```

```
print(x) # this will also print it
```

```
## [1]  2 10 11  1 23
```

Now, let's say you want to create a vector with a sequence of numbers, such as creating a vector with the numbers from 1 to 10. There are different ways of accomplishing this, such as:

```
x <- 1:10
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

In this case `1:10` denotes a way of creating a sequence of numbers from 1 to 10. You can also specify it in decreasing order, in case you want to create a sequence from 10 to 1:

```
x <- 10:1
x
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

You can add to vectors together with the `c()` function:

```
x <- c(1:5, 5:1)
x
```

```
##  [1] 1 2 3 4 5 5 4 3 2 1
```

```
x <- c(10, 100, 1000)
y <- c(1, 2, 3)
z <- c(x, y, x)
z
```

```
## [1]   10  100 1000    1    2    3   10  100 1000
```

You can also use a different function to create this vector, with the function `seq`:

```
x <- seq(1:10)
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Alternativity, you can use different syntax:

```
x <- seq(from=1, to=10)
x
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

So this line `x <- seq(1:10)` is equivalent to `x <- seq(from=1, to=10)`. The latter has some advantages, such as defining other parameters. Let's say that in the sequence you want to have numbers increasing by 2:

```
x <- seq(from=1, to=10, by=2)
x
```

```
## [1] 1 3 5 7 9
```

Or, you want exactly 7 numbers:

```
x <- seq(from=1, to=10, length.out=7)
x
```

```
## [1]  1.0  2.5  4.0  5.5  7.0  8.5 10.0
```

R will automatically create the equally spaced sequence in the [1, 10] bounds. To see more parameters of seq function, check the R help (`?seq`).

There are other way of specifying vectors, with the funcion `rep`. Let's say you want to create a vector of length 10, all of them being 1:

```
x <- rep(1, 10)
x
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

There are other types of sequences you can do with rep, such as:

```
rep(1:3)
```

```
## [1] 1 2 3
```

```
rep(1:3, times=3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

```
rep(1:3, each=3)
```

```
## [1] 1 1 1 2 2 2 3 3 3
```

Let's say you want to create a sequence of square numbers, `1  4  9 16 25`. You can do so by creating a vector, and then squaring it. R works as element base, so the square function will be applied to each element of the sequence:

```
x <- 1:5
x
```

```
## [1] 1 2 3 4 5
```

```
y <- x ^ 2 # alternative, you can use x ** 2
y
```

```
## [1]  1  4  9 16 25
```

```
## 1  4  9 16 25
```

Now, let's say you want to create the sequence 1, 2, 4, 8, 16, 32. Notice this sequence is 2ˆn:

```
x <- 2**(0:10)
x
```

```
##  [1]    1    2    4    8   16   32   64  128  256  512 1024
```

```
# [1]    1    2    4    8   16   32   64  128  256  512 1024
```

In this case, a vector is applied as a pow to a single number, creating a sequence of numbers. Note the use of the parenthesis.

Operations with vectors:

You can do all kind of operations with vectors, +, -, /, etc. Operations are performed element-wise.

```
x <- c(2, 3, 4, 5)
y <- c(1, 0, 3, 2)

x + y
```

```
## [1] 3 3 7 7
```

```
(x + y) * 2
```

```
## [1]  6  6 14 14
```

```
x + y * 2
```

```
## [1]  4  3 10  9
```

Note the different values in the third operation, as the `* 2` is applied to y before the sum to x.

How to access a vector element. Elements in R are index based, with the first index being 1 (instead of 0 like in other programming languages)

```
x <- c(2, 4, 5, 8, 10)
```

In this case, 2 is index 1, 4 is index 2, 5 is index 3.

```
x[1]
```

```
## [1] 2
```

```
x[2]
```

```
## [1] 4
```

```
x[3]
```

```
## [1] 5
```

If you want to access more than 1 element, you can specify a vector of indexes:

```
x[1:3]
```

```
## [1] 2 4 5
```

However, if you specify indexes that don't exist in the vector, R will return NAs:

```
x[3:10]
```

```
## [1]  5  8 10 NA NA NA NA NA
```

You can also subset this vector by removing some indices:

```
x
```

```
## [1]  2  4  5  8 10
```

```
x[-5]
```

```
## [1] 2 4 5 8
```

```
x[-c(4, 5)]
```

```
## [1] 2 4 5
```

The commands above remove index 5 (second command) and 4,5 indices (third command).

You can assign a value from the same way. Let's say you want to change the first number to be 9 instead of 2:

```
x[1]
```

```
## [1] 2
```

```
x[1] <- 9

x[1]
```

```
## [1] 9
```

```
x
```

```
## [1]  9  4  5  8 10
```

You can also assign a sequence of numbers in the same operation. Let's say you want the first thre elements to be 1, 2, 3:

```
x
```

```
## [1]  9  4  5  8 10
```

```
x[1:3] <- c(1, 2, 3)
x
```

```
## [1]  1  2  3  8 10
```

Now let's assume you want to subset the vector based on some logic. If you have a vector from 1:5, you can specify which indices you want by a logic vector:

```
x <- 1:5
x
```

```
## [1] 1 2 3 4 5
```

```
x[c(TRUE, TRUE, FALSE, FALSE, TRUE)]
```

```
## [1] 1 2 5
```

```
# Equivalent to:
x[c(1, 2, 5)]
```

```
## [1] 1 2 5
```

This allow us to perform some logic for subsetting. Let's first see what happens when you perform comparison equations over a vector:

```
x <- 1:5
y <- x < 3
y
```

```
## [1]  TRUE  TRUE FALSE FALSE FALSE
```

If we have a vector of 1:5 (elements 1, 2, 3, 4, 5) and we use a comparison method of `x < 3` we obtain a vector with `c(TRUE, TRUE, FALSE, FALSE, FALSE)`. This means that the first 2 elements of vector `x` are `<3` and the rest elements of vector are not. Now we can use that vector to subset x in case we only want those elements:

```
x[y]
```

```
## [1] 1 2
```

What if we wanted the oposite? You want the numbers that are equal or higher than 3. You can imagine that an easy way to do it is:

```
x <- 1:5
z <- x >= 3
z
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

```
x[z]
```

```
## [1] 3 4 5
```

However, you can use our previous vector and negate it:

```
!y
```

```
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

```
x[!y]
```

```
## [1] 3 4 5
```

Subsetting though does not require to specify a different variable, you can apply the logic directly:

```
x[x<3]
```

```
## [1] 1 2
```

```
x[!(x<3)]
```

```
## [1] 3 4 5
```

```
x[x>=3]
```

```
## [1] 3 4 5
```

You can use more than one logic operations:

```
x[x<3 | x==5]
```

```
## [1] 1 2 5
```

What if you only want even numbers?

```
x[x%%2 == 0]
```

```
## [1] 2 4
```

Operations with vectors

You can do operations with vectors

How to create a matrix