(Introduction to R for Absolute Beginners)

VECTORS IN R

Created: 2021-05-08 Sa 21:11

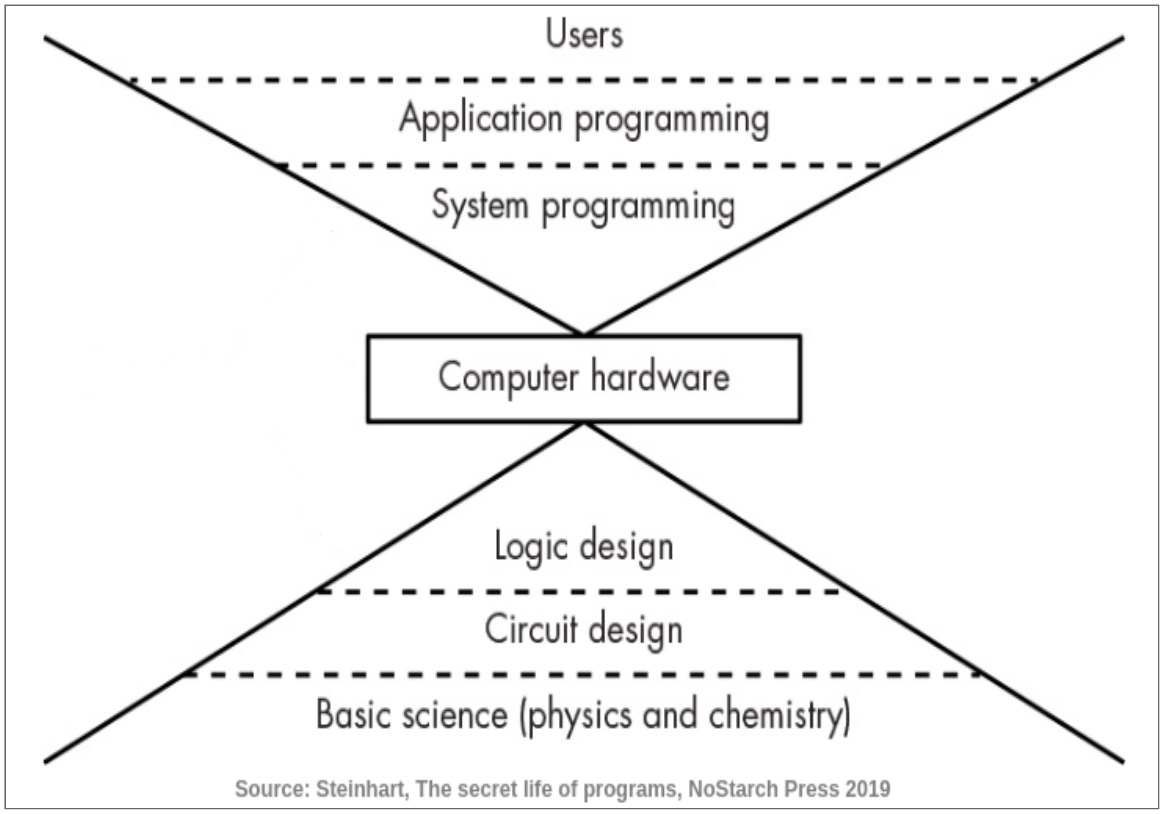
WHAT WILL YOU LEARN?

- Understand assignment in R
- Creating vectors, sequences and repetitions
- Sorting and measuring vector length
- Subsetting and extracting vector elements
- Vectorizing (rescaling)
- Classes and logical vectors

EVERYTHING IS AN OBJECT

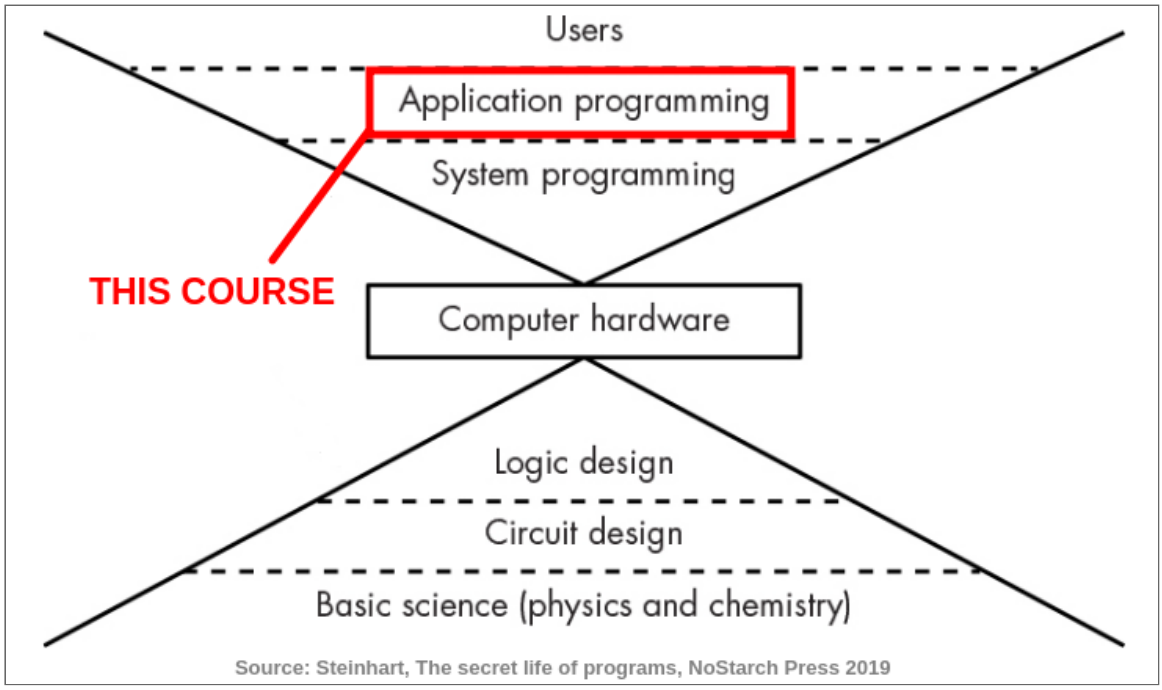
Performance	Big Data	`data.table` vs. `dplyr`
Ease-of-use	Fun factor	Interactive vs. file
Clarity	Communication	Base R vs. Tidyverse

ENTERING THE FORBIDDEN KINGDOM



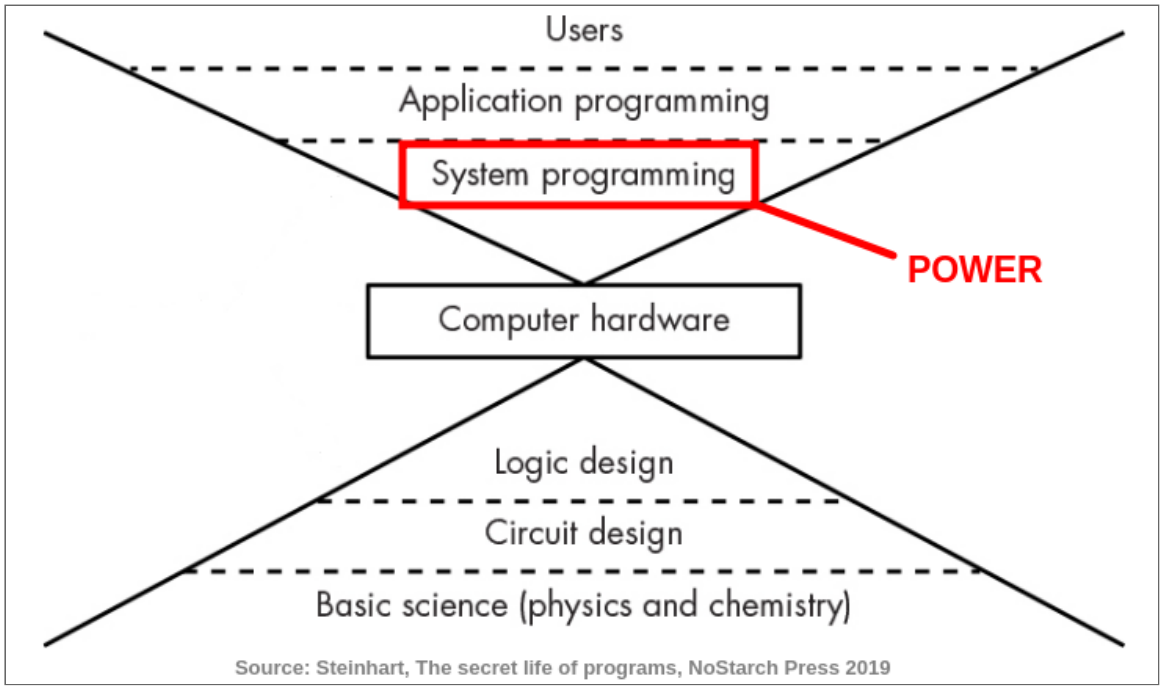
- Where are we at in this course?
- Who's really got the power?

WHERE THIS COURSE IS



- Where are we at in this course?

WHO'S GOT THE POWER



- Who's really got the power?

ASSIGNING OBJECTS

```
x <- 5     # assigns 5 to a variable x
x          # prints the content of x to the screen
y = 10     # assigns 10 to a variable y
print(y)   # prints the content of y to the screen
x+y        # adds content of both variables
x <- x + 1 # overwrites value of x
x          # prints new content of x
```

- Assigning/storing values to/in a variable
- You can use <- or =
- Use = for function arguments only

BE THE COMPUTER!

1. Create an object a that stores the value $3^2 \times 4^{\frac{1}{8}}$
2. Overwrite a with itself divided by 2.33. Print the result to the console.
3. Create a new object b with the value -8.2×10^{-13}
4. Print to the console the result of multiplying a and b.

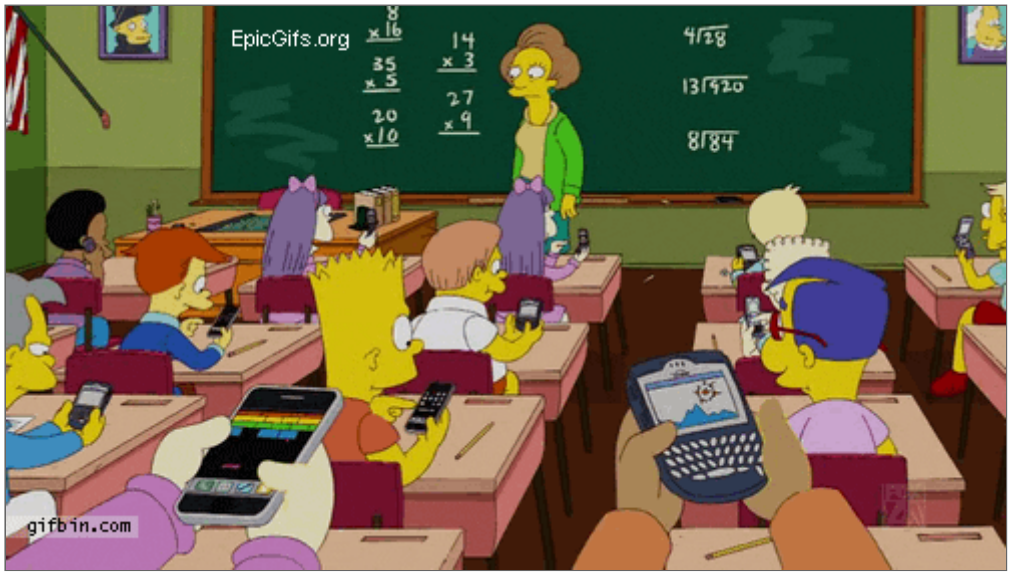
SOLUTION

```
a <- 3^2 * 4^(1/8)  # 1) 10.70286
a <- a/2.33         # 2) 4.593504
a
b <- (-8.2) * 10^(-13) # 3) -8.2e-13
a * b                  # 4) -3.766673e-12
```

WHY WE NEED VECTORS

- Handle multiple items (observations, data points)
- Single values of the same type
- Different types lead to coercion

BE THE COMPUTER!



Give examples of class data for
(1) numbers, (2) text, (3) logical values
that could be put into a vector.

SAMPLE SOLUTION

```
s_heights <- c(180, 181, 158, 175, 179, 168)
s_names <- c("Vincent", "Natalija", "Adrian", "Andres", "Helena")
s_male <- c(TRUE, FALSE, TRUE, TRUE, FALSE)
```

- `s_heights` is a `numeric` ("number") vector
- `s_names` is a `character` ("string") vector
- `s_male` is a `logical` ("Boolean") vector

CREATING VECTORS

- Vector of vectors
- Calculations in vectors
- The meaning of row labels
- Type coercion

CREATING AN INTEGER VECTOR

```
> myvec <- c(1,3,1,42)  # define an integer vector
> myvec                 # print vector
> class(myvec)          # check type
```

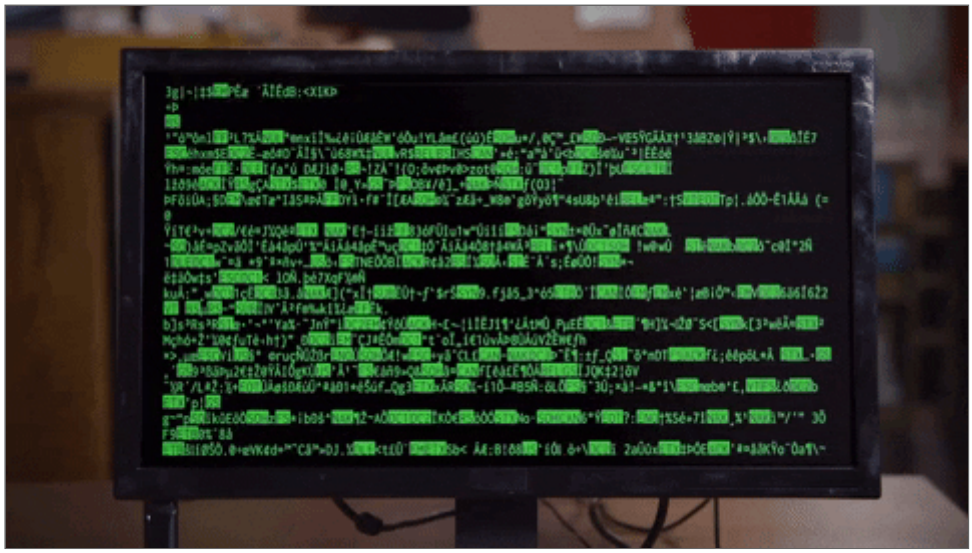
VECTORS CAN CONTAIN CALCULATIONS

```
> foo <- 32.1           # define floating point scalar
> # define vector of numbers, calculations, and vectors
> myvec2 <- c(3, -3, 3.45, 1e+03, 64^0.5, 2+(3-1.1)/9.44, foo)
> myvec2
```

VECTORS CAN CONTAIN VECTORS

```
> myvec3 <- c(myvec, myvec2) # define vector of vectors
> myvec3                     # print vector
> class(myvec3)              # assert object type
```

BE THE COMPUTER!



- What about NA, NaN, `Inf`, `-Inf`?
- Can you have vectors of these values?
- How would you test for these values?

SPECIAL VALUE VECTORS - CODE

```
> specvec <- c(NA,NaN) # a vector with NA and NaN
> specvec              # print vector
> class(specvec)       # what type is the vector?
> is.nan(specvec)      # does it have NaN elements?
> is.na(specvec)       # does it have NA elements?
> specvec <- c(NA,NaN) # a vector with a NA and a NaN
> is.nan(specvec)      # does it have NaN elements?
> specvec1 <- c(specvec, Inf, -Inf)  # add Inf elements
> is.finite(specvec1)  # test for finiteness
> is.infinite(specvec1) # test for infiniteness
> class(specvec1)       # what type is the vector?
```

DOWN THE NILE

1. How can you get info on the dataset `Nile`?
2. What type data set is `Nile`?
3. Which ways do you know to print `Nile` data?

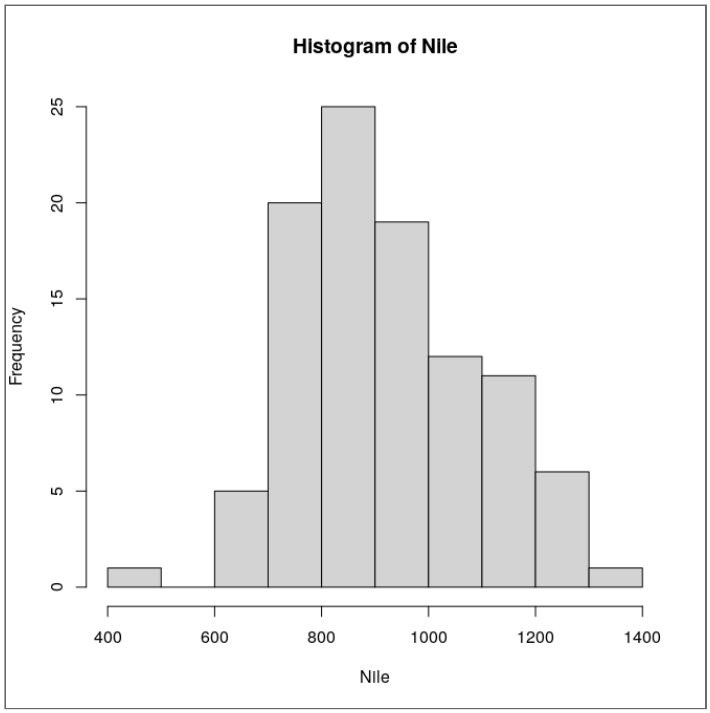
DOWN THE NILE - CODE

```
data()      # lists all available/loaded datasets
?Nile       # opens help page for the dataset Nile
```

```
class(Nile)  # what type of dataset is this?
```

```
str(Nile)   # show dataset structure
head(Nile)  # show first few elements
Nile        # this prints the whole dataset
```

PLOTTING THE NILE



- Type `hist(Nile)` - What is this?
- What does the function `hist` do?
- How can you see all measurements with `hist`?

CREATING SEQUENCES AND REPETITIONS

- The colon operator (`:`)
- Sequences with `seq`
- Repetitions with `rep`

THE COLON OPERATOR

- `3:27` generates integers from 3 to 27
- You can check the type with `class(3:27)`
- Calculations allowed (in parentheses)

COLON CALCULATIONS

```
> foo <- 5.3           # assign 5.3 to foo
> bar <- foo:(-47+1.5) # assign sequence to bar
```

- Step size is always =1
- Closest start/end point

SEQUENCES I

```
> seq(from = 3, to = 27, by = 3)  # stepsize 3

> seq(3,27,3)  # 3rd argument = step size

> seq(from = 3, to = 27, by = 3)

> seq(1,10,2)  # even end number, even step size

> seq(1,11,2)  # odd end number, even step size
```

- Vary step size with by
- Leads to increasing or decreasing sequence

SEQUENCES II

```
# 40 evenly space values exactly:
> seq(from = 3, to = 27, length.out = 40)

> foo <- 5.3
> myseq <- seq(from = foo, to = (-47+1.5), by = -2.4)
> myseq    # negative step size for decreasing sequence

> myseq2 <- seq(from = foo, to = (-47+1.5), length.out=5)
> myseq2   # decreasing sequence of five values
```

- Negative step size by `< 0`
- Exact sequence length with `length.out > 0`

REPETITION I

```
> rep(x = c(3,62,8,3), times = 3)  # repeat vector 3 times
> rep(x = c(3,62,8,3), each = 3)   # repeat each element three times
> rep(x = c(3,62,8,3), times=3, each=2)   # double elements, repeat thrice
> rep(x = c(3,62,8,3), each=2, times=3)   # order of arguments irrelevant
> rep(x = c(3,62,8,3))
```

- Arguments `x`, `times`, `each`

REPETITION II

```
> foo <- 4
> c(3, 8.3, rep(x=32,times=foo),seq(from=-2,to=1,length.out=foo+1))

> rep("data science", times=2)

> rep(c("data", "science"), times=2)

> rep(c("data", "science"), times=2, each=2)
```

- You can include `rep` in a vector
- Works for `characters`, too
- See also `vector` function

SORTING AND MEASURING LENGTHS

SORT VECTOR VALUES

```
> sort(x=c(2.5, -1, -10, 3.44)) # increasing: default
[1] -10.00  -1.00   2.50   3.44

> sort(x=c(2.5, -1, -10, 3.44), decreasing=TRUE)
[1]   3.44   2.50  -1.00 -10.00

sort(x=c(2.5, -1, -10, 3.44), decreasing=FALSE)
[1] -10.00  -1.00   2.50   3.44
```

- Decreasing or increasing?
- `Decreasing=FALSE` is the default
- Note: `logical` argument

MEASURE LENGTH

```
> length(x=c(3,2,8,1))
> length(x=5:13)

> foo <- 4
> bar <- c(3, 8.3, rep(x=32, times=foo), seq(from=-2, to=1, length.out=foo+1))
> length(bar)
```

- Get or set vector length

EXERCISES AND SOLUTIONS

CREATE SEQUENCE

(a) Create and store a sequence of values from `5` to `-11` that progresses in steps of `0.3`.

OVERWRITE VECTOR

(b) Overwrite the object from (a) using the same sequence with the order reversed.

REPEAT VECTOR

(c) Repeat the vector `c(-1,3,-5,7,-9)` *twice, with each element repeated* `10` *times, and store the result. Display the result sorted from largest to smallest.*

MEASURE VECTOR LENGTH

(d) Create and store a vector that contains, in any configuration, the following:

- *A sequence of integers from `6` to `12` (inclusive)*
- *A threefold repetition of the value `5.3`*
- *The number `-3`*
- *A sequence of nine values starting at `102` and ending at the number that is the total length of the vector created in problem (c).*
- *Confirm that the length of the vector created is `20`*

PART II - SUBSETTING

Video: **SUBSETTING**

VECTORIZATION 1

1. Operator/function acts on each vector element:

```
foo <- c(1, -1, 4, 4, 0, 59, 3)  # defin
foo + 3     # add a number to the vector
foo/3.2     # divide vector by number
bar <- foo[-c(4:length(foo))]  # delete
rep(x=bar, times=2)    # repeat a vector
class(exp(pi*1i)+1)  # Euler's formula
prod(c(1,2,3,4,5))   # product of elemen
```

VECTORIZATION 2

1. Function computes a summary statistic over a vector:

```
1:5        # ':' is a vector function
sum(1:5)   # sum computes the sum of elem
mean(1:5) # mean of vector = 1 argument
```

Where does this leave us?

- How to get to parts of a vector
- How to control the indexing
- How to rescale vectors
- How to create matrices and arrays
- How to mix different data types

VECTOR INDEX

- The row labels, like `[1]`, correspond to the **index**:

```
Nile     # prints all elements of the Nile data set
```

SUBSETTING

- Use [ ] to retrieve elements or subsets
- How would you retrieve the first element?
- How would you retrieve the last element?

```
Nile[1]              # retrieve the first
Nile[length(Nile)]   # retrieve the last
Nile[100]            # retrieve the last
```

INDEXING WITH :

- Define a vector of 6 elements
- Retrieve elements 2 through 5

```
foo <- c(-1,3.0,4,67,330,-3)  # assign v
bar <- foo[2:5]   # assign subset of foo
foo;bar
```

IS INDEXING ASSOCIATIVE?

- Is `foo[n]:foo[m]==foo[n:m]`?
- Define vector `foo` of six elements
- Try this with n=2 and `m`=5

```
foo <- c(-1, 3, 4, 67, 330, -3)
length(foo)

bar <- foo[2:5]   # assign a subset of f
length(bar)

baz <- foo[2]:foo[5]  # assign a sequenc
length(baz)

identical(bar,baz)  # are bar and baz ic
all.equal(bar,baz)  # are they near equa
```

STATISTICAL FUNCTIONS

- Stats functions work on any vector subset
- Examples: `mean`, `sum`, `summary`

```
foo <- c(-1, 3, 4, 67, 330, -3)
mean(foo[2:5])
sum(foo)
summary(foo)
```

LOGICAL OPERATORS

- Logical operators: <, >, !=, ==
- What do you get with `c(foo == 0)`?
- What could you use these for?

```
foo <- c(-1, 3, 4, 5, 67, 330, -3, -99,
foo_pos <- c(foo > 0) # select positive
foo_neg <- c(foo < 0) # select negative
foo_nul <- c(foo == 0)  # Why not = inst

foo_pos   # logical vectors
foo_neg
foo_nul

sum(foo_pos)   # What do you expect here
sum(foo_neg)
sum(foo_nul)
```

LOGICAL SUBSETTING

- TRUE and FALSE select vector values
- Remember: `foo_pos <- c(foo > 0)`
- `class(foo_pos)` is `"logical"`

```
foo               # target vector
foo_pos           # logical index vector c
foo[foo_pos]      # select positive values
foo[foo_neg]      # select negative values
foo[foo_nul]      # select zero values onl
```

ONCE MORE DOWN THE `Nile`

1. How many entries does `Nile` have?

```
length(Nile)
```

2. What is the last element of `Nile`?

```
tail(Nile)   # print last few elements
Nile[length(Nile)] # print last element
Nile[100]  # print last element
Nile       # print all of Nile
```

1. What is the third to last element of `Nile`?

```
Nile[length(Nile)-3]
```

2. What is the average flow of the Nile between 1909-1969?

```
1909-1871  # compute first index by hanc
1969-1871  # compute last index by hand

mean(Nile[(1909-1871):(1969-1871)])
```

1. How many values are below, how many above the mean value?

```
nile_pos <- c(Nile > mean(Nile))  # valu
head(nile_pos)   # logical index vector

nile_neg <- c(Nile < mean(Nile)) # value
head(nile_neg)   # logical index vector

sum(nile_pos)  # number of above mean va
sum(nile_neg)  # number of below mean va
```

1. How much water flowed down the Nile between 1871 and 1970?

```
sum(Nile)  # water volumne in 10^8 m^3 (
```

NEGATIVE INDEX OPERATOR

- Negative indices remove elements
- Only affects the output (unless you overwrite)
- Works with vectors, too

```
foo <- c(-1, 3, 4, 5, 67, 330, -3, -99,
foo; length(foo)  # print foo and length

foo[-1]         # remove first element
foo[-length(foo)]  # remove last element

length(foo)   # foo is not changed (not

foo[-c(1,3)]   # remove elements 1 and 3
foo[-(1:2)]    # remove elements 1 and 2
```

EXERCISE: FIX A VECTOR

- Use `myvec <- c(5,-2,3,4,6,8,10,40221,-8)`
- Change second element to `-2.3`
- Print fixed vector

```
myvec <- c(5,-2,3,4,6,8,10,40221,-8) # starting vector
## I want: 5 -2.3 4 6 8 10 40221 -8
myvec

myvec[2] <- -2.3     # change second element
myvec[-3]            # does NOT change myvec!
myvec                # myvec is still unchanged

myvec <- myvec[-3]   # delete third element
myvec                # myvec has been changed
```