



First Steps in R (part 1)

Enrico Toffalini

PSICOSTAT

Let's see how to

Part 1:

- Create, name objects;
- Use basic operations (arithmetical, relational, logical);
- Basic types of data;

Part 2:

- Install, load from CRAN (and more), call functions, help;
- Using functions;
- Set up your Working Directory;
- Import, export workspace and data

Create, name objects

In R, **everything is an object**: variables, vectors, dataframes, functions, even entire environments.

Let's create a variable named “**age**” that contains a single numerical value:

```
age = 20 # assign number 20 to variable named "age"
```

Now let's simply inspect its content

```
age
```

```
[1] 20
```

```
# alternative way of showing content,  
# useful in programming when within functions or loops  
print(age)
```

```
[1] 20
```

```
# for more complex data structures the "str" function may be useful  
str(age)
```

```
num 20
```

Create, name objects

Assignment operators

In R, both the assignment operator “=” and “<-” can be used to assign values to objects. In fact, “<-” is considered more traditional in R. and preferred for clarity, also because it allows differentiating assignment from other uses of “=”.

```
# these two commands do the same thing  
age <- 20  
age = 20
```

However, unlike many other teachers, I will generally favor “=” as the assignment operator in order to maintain consistency with the convention in most other programming languages

Create, name objects

Rules for naming objects in R

Strict rules:

- Start with a letter or dot (if dot, must **not** be followed by a number);
- Include only letters, numbers, dots, underscores;
- No reserved words (e.g., “`if`”, “`for`”, “`NA`”, “`function`”).

Recommendations:

- Avoid names that conflict with common functions (e.g., “`mean`”, “`sum`”, “`c`”);
- Be concise: no length limit, but long names are difficult to read and type.

WARNING! R is Case sensitive: `age` and `Age` will be treated as two **different** objects!

Create, name objects

Rules for naming objects in R

Examples:

- Allowed: “age”, “age0”, “age1”, “total_score”, “.myData”, “my.data”,
- NOT allowed: “0age”, “_age”, “.0myData”, “my data”, “my-data”, “my, data”, “for”, “NA”

WARNING! Use of “.” in object names (e.g., “my.data”) is fine in R but not allowed in **Python**, where “.” is part of the language syntax.

Across different languages, *naming conventions* for longer, multi-word variable names favor **snake_case** (e.g., “my_data”) or **camelCase** (e.g., “myData”), and **abbreviations** where appropriate (e.g., “unipdData” better than “university_of_padova_dataset”)... preferably used in a consistent way!

Use basic operations

R as calculator: some basic operators

Operator	What it does	Example	Result
<code>+</code>	Addition	<code>5.4 + 6.1</code>	<code>11.5</code>
<code>-</code>	Subtraction	<code>9 - 4.3</code>	<code>4.7</code>
<code>*</code>	Multiplication	<code>7 * 1.4</code>	<code>9.8</code>
<code>/</code>	Division	<code>9 / 12</code>	<code>0.75</code>
<code>%/%</code>	Floor division	<code>13 %/% 4</code>	<code>3</code>
<code>%%</code>	Modulus	<code>13 %% 4</code>	<code>1</code>
<code>^</code>	Exponentiation	<code>15 ^ 2</code>	<code>225</code>

(also useful: object “`pi`” contains `3.1415927`)

Use basic operations

R as calculator: useful functions

Function	What it does	Example	Result
<code>abs</code>	absolute value	<code>abs(4.3-9.8)</code>	5.5
<code>sqrt</code>	square root	<code>sqrt(176.4)</code>	13.28157
<code>exp</code>	exponential function	<code>exp(2.2)</code>	9.025013 ($e^{2.2}$)
<code>log</code>	natural logarithm, base e	<code>log(9.025013)</code>	2.2
<code>log</code>	logarithm, given base	<code>log(10, base=2)</code>	3.321928
<code>round</code>	round to integer	<code>round(1.7384)</code>	2
<code>round</code>	round to digits	<code>round(1.7384, 2)</code>	1.74

Use basic operations

R as calculator: use of parentheses

The order of operations in R follows standard algebraic rules, unless you specify a different order using parentheses. In R, only round parentheses () are used for grouping in algebraic expressions, **NOT** square [] and curly { } brackets, because they have other specific syntactic purposes.

Examples:

```
2 * 3 + 3^2
```

```
[1] 15
```

```
2 * (3 + 3)^2
```

```
[1] 72
```

```
(2 * (3 + 3))^2
```

```
[1] 144
```

Use basic operations

Relational operators

They are used to compare values and return logical values (**TRUE**, **FALSE**).

Let's say that we defined `age = 20`, now let's make a few examples:

Operator	What it does	Example	Result
<code>==</code>	Equal to	<code>age == 18</code>	FALSE
<code>!=</code>	Not equal to	<code>age != 18</code>	TRUE
<code>></code>	Greater than	<code>age > 18</code>	TRUE
<code><</code>	Less than	<code>age < 18</code>	FALSE
<code>>=</code>	Greater than or equal to	<code>age >= 18</code>	TRUE
<code><=</code>	Less than or equal to	<code>age <= 18</code>	FALSE

Use basic operations

Basic logical operators

They are used to combine logical values (**TRUE**, **FALSE**).

Once again, let's say that we defined **age = 20**, now let's make a few examples:

Operator	What it does	Example	Result
&	AND	age > 25 & age < 60	FALSE
 	OR	age < 25 age > 60	TRUE
!	NOT	!(age < 18)	TRUE

Basic types of data

numeric and logical

So far, we have already encountered at least two types of data:

- **numeric** (e.g., `20`, `11.5`, `13.28157`);
- **logical/Boolean** (i.e., `TRUE`, `FALSE`).

Actually, **numeric** data could actually be of two types: *double* (i.e., “*double-precision floating-point*”) that is with decimals like `11.5`, and *integer* like `20`.

In fact, by default, numeric values are always treated as *double* (even if without decimals). To specify a number explicitly as integer, add an `L` after the number, like `age = 20L` (you likely **will not** need this, unless you explicitly need integers for some purposes, such as saving memory).

Basic types of data

characters

Another very important type of data is:

- **character** (often called *strings*). This is used to store any text, and must be enclosed in quotes (' ', or " "), like this:

```
myName = "Enrico"
```

You may perform many operations with strings like:

```
myName == "Bob" # is my name equal to Bob?
```

```
[1] FALSE
```

```
myName != "Bob" # is my name NOT equal to Bob?
```

```
[1] TRUE
```

```
myName > "Bob" # is my name larger than Bob? (??? alphabetically!)
```

```
[1] TRUE
```

Basic types of data

know the type of a variable

The **typeof()** function tells you what type of data you are handling:

```
myName = "Enrico"  
prof = TRUE  
coursesTaught = 4L  
age = 36
```

```
# see data types  
typeof(myName)
```

```
[1] "character"
```

```
typeof(prof)
```

```
[1] "logical"
```

```
typeof(coursesTaught)
```

```
[1] "integer"
```

```
typeof(age)
```

```
[1] "double"
```

Basic types of data

know the type of a variable

You may also inquire data type directly with functions `is.*`:

```
is.logical(prof)
```

```
[1] TRUE
```

```
is.logical(age)
```

```
[1] FALSE
```

```
is.numeric(age)
```

```
[1] TRUE
```

```
is.infinite(age)
```

```
[1] FALSE
```

```
is.character(myName)
```

```
[1] TRUE
```

```
is.na(myName) # checks if a value is missing (i.e., NA)
```

```
[1] FALSE
```