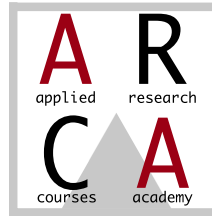


Introduzione a R

Primi passi con R



ARCA - @DPSS

Filippo Gambarota

Primi passi con R

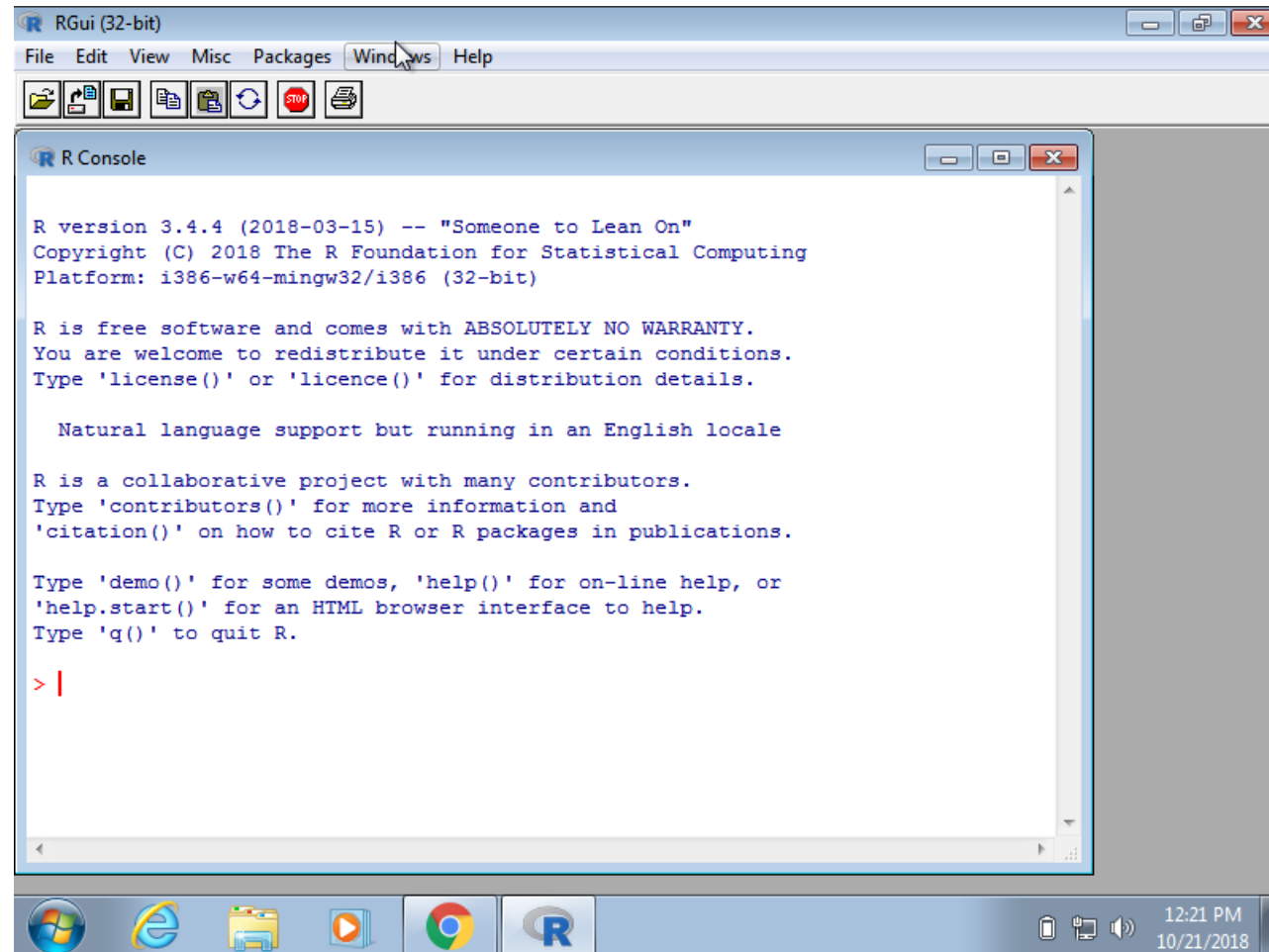
Installazione

Per l'installazione trovate le indicazioni nella sezione **Installare R e RStudio** del libro. In generale i passaggi sono:

- scaricare R e installare **R** per il vostro sistema operativo
- scaricare e installare **RStudio**

Come si presenta R

Console



The screenshot shows the RGui (32-bit) application window. The title bar reads "RGui (32-bit)". The menu bar includes "File", "Edit", "View", "Misc", "Packages", "Windows", and "Help". Below the menu bar is a toolbar with icons for file operations and execution. The main window is titled "R Console" and contains the following text:

```
R version 3.4.4 (2018-03-15) -- "Someone to Lean On"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

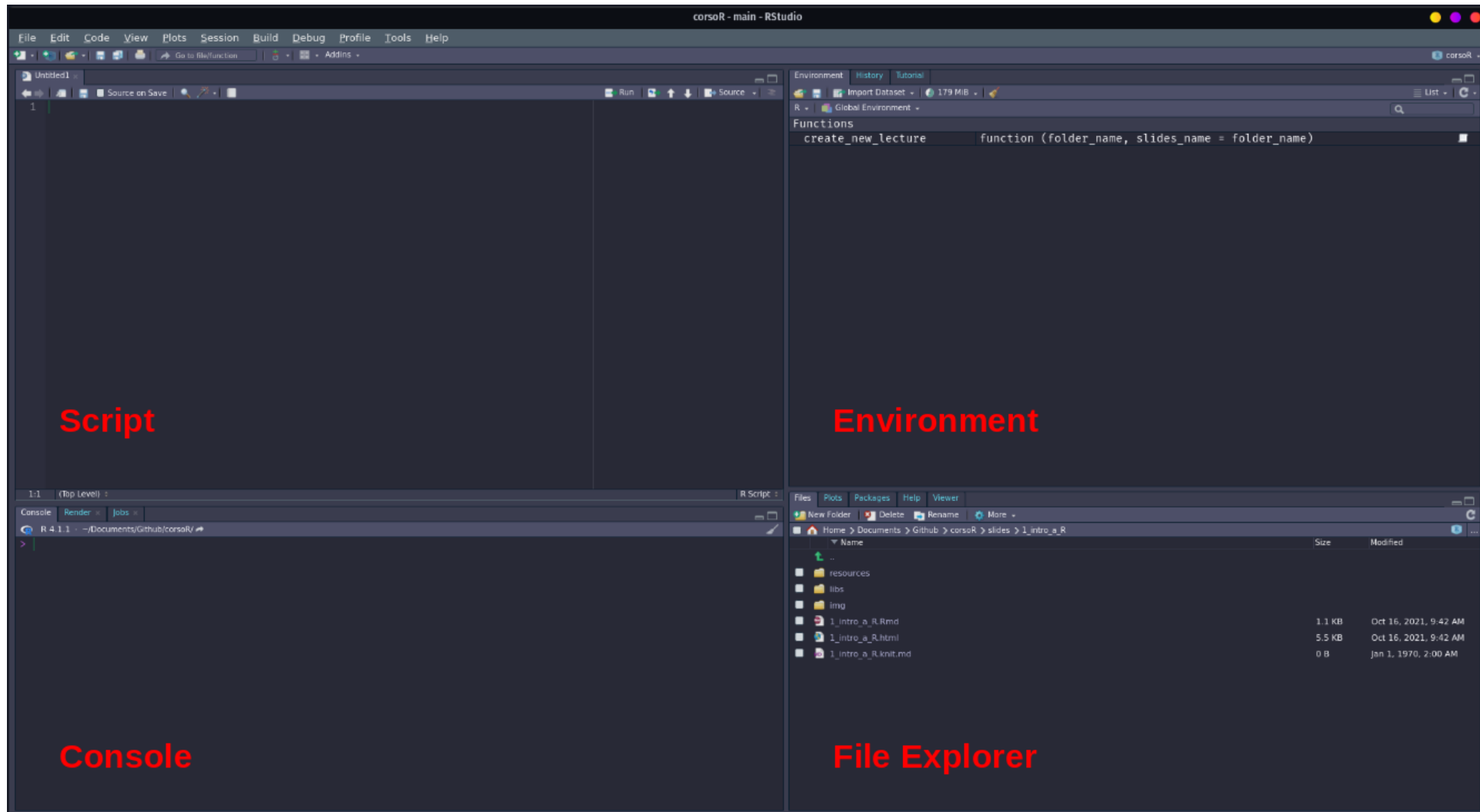
R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

The Windows taskbar at the bottom shows the Start button, icons for Internet Explorer, File Explorer, a media player, Google Chrome, and the RGui application. The system clock in the bottom right corner displays "12:21 PM" and "10/21/2018".

RStudio



I primi passi in R

R come calcolatrice

In R è possibile effettuare tutte le **operazioni matematiche** e algebriche dalle più semplici alle più avanzate

Funzione	Nome	Esempio
<code>x + y</code>	Addizione	<code>> 5 + 3</code> <code>[1] 8</code>
<code>x - y</code>	Sottrazione	<code>> 7 - 2</code> <code>[1] 5</code>
<code>x * y</code>	Moltiplicazione	<code>> 4 * 3</code> <code>[1] 12</code>
<code>x / y</code>	Divisione	<code>> 8 / 3</code> <code>[1] 2.666667</code>
<code>x %% y</code>	Resto della divisione	<code>> 7 %% 5</code> <code>[1] 2</code>

Operatori matematici

- Importante considerare l'**ordine delle operazioni** analogo alle regole della matematica: $2 \times 3 + 1$ prima 2×3 e poi $+ 1$. Analogamente in R:

```
# Senza parentesi
```

```
2 * 3 + 1
```

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

```
# Con le parentesi
```

```
(2 * 3) + 1
```

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

```
# Con le parentesi forzando un ordine diverso
```

```
2 * (3 + 1)
```

```
## [1] 8
```

Operatori relazionali

Gli operatori relazionali sono molto utili dentro le **funzioni**, per **selezionare elementi dalle strutture dati** (vedremo più avanti) e in generale per **controllare** alcune sezioni del nostro codice:

```
3 > 4
```

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

```
3 >= 3
```

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

```
10 < 100
```

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

```
40 == 40
```

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

```
10 != 50
```

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

Operatori logici

Gli operatori logici permettono di **combinare espressioni relazionali** e ottenere sempre un valore `TRUE` o `FALSE`:

```
3 > 4 & 10 < 100
```

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

```
10 < 100 | 50 > 2
```

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

```
!5 > 4
```

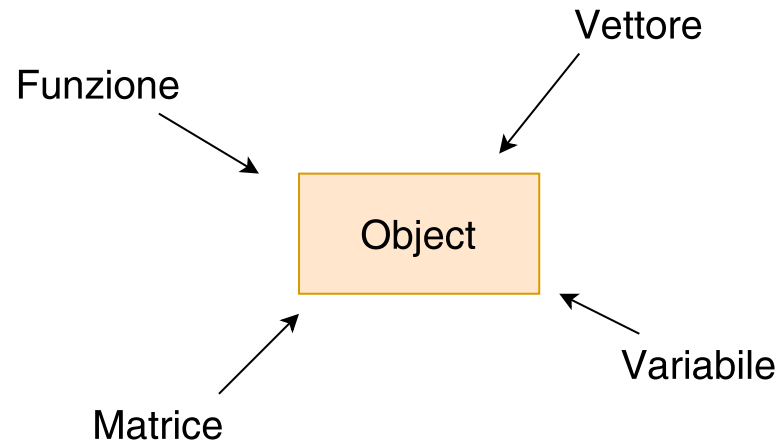
```
## [1] FALSE
```

R e gli oggetti

R e gli oggetti

| “Everything that exists in R is an object” - John Chambers

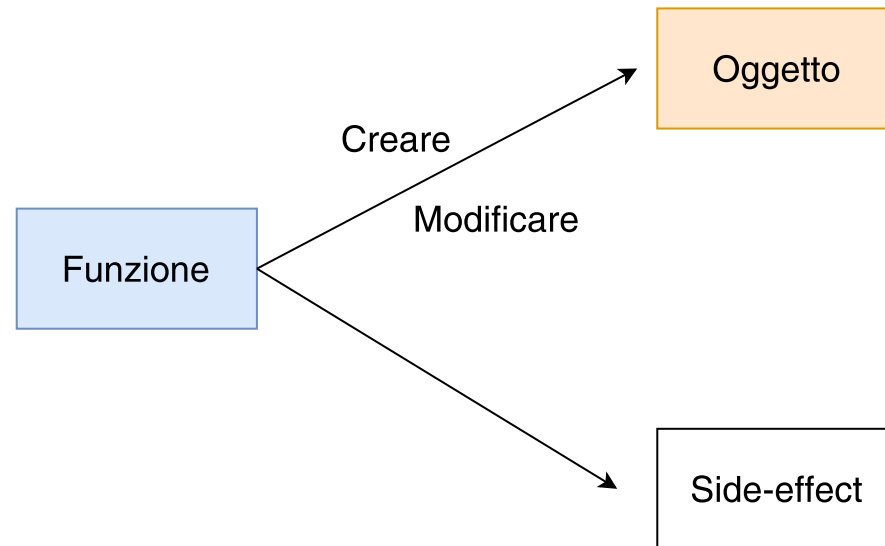
Il concetto di **oggetto** è fondamentale in R. Essenzialmente tutto quello che possiamo creare o utilizzare in R come un numero, un vettore, dei caratteri o delle funzioni sono creati come oggetti.



R e le funzioni

| “Everything that happen in R is a function call” - John Chambers

Anche il concetto di **funzione** è fondamentale in R. Essenzialmente tutto quello che facciamo è chiamare **funzioni** su oggetti ottenendo un nuovo oggetto o modificando un oggetto esistente



Cosa possiamo usare/creare in R?

- **Numeri:** 100, 20, 6, 5.6 sono tutti numeri interpretati e trattati come tali
- **Stringhe:** "ciao", "1" sono *caratteri* che vengono interpretati letteralmente devono essere dichiarati con `""`
- **Nomi:** ciao, x sono nomi (senza virgolette) e sono utilizzati per essere associati ad un oggetto (variabile, funzione, etc.)
 - **operatori:** sono delle funzioni (e quindi oggetti con un nome associato) che si utilizzano in modo particolare. `3 + 4` in questo caso `+` è un operatore (funzione) che si può usare anche come `+(3, 4)`

R e gli oggetti

- Come creare un oggetto?
- Oggetti e nomi
- Dove viene creato l'oggetto?

Come creare un oggetto?

La creazione di un oggetto avviene tramite il comando `<-` oppure `=` in questo modo: `nome <- oggetto`:

```
x
```

```
## Error in eval(expr, envir, enclos): object 'x' not found
```

```
10 # questo non è un oggetto, non è salvato
```

```
## [1] 10
```

```
x <- 10 # ora il valore numerico 10 è associato al nome "x"
```

```
x
```

```
## [1] 10
```

Convenzioni vs regole

Ci sono alcune cose da considerare quando si scrive codice ed in particolare si creano oggetti:

- **alcune modalità sono errate** --> R ci fornisce un messaggio di errore
- **alcune modalità sono sconsigliate** --> funziona tutto ma ci potrebbero essere problemi
- **alcune modalità sono stilisticamente errate** --> funziona tutto, nessun problema ma... anche l'occhio vuole la sua parte

Oggetti e nomi

Il nome di un oggetto è importante sia per l'utente che per il software stesso:

```
1 <- 10 # errore  
  
_ciao <- 10 # errore  
  
mean <- 10 # possibile ma pericoloso  
  
`1` <- 10 # con i backticks si può usare qualsiasi nome ma poco pratico
```

```
## Error: <text>:4:2: unexpected symbol  
## 3:  
## 4: _ciao  
##      ^
```

```
my_obj <- 10  
  
my.obj <- 10  
  
My_obj <- 10 # attenzione a maiuscole e minuscole
```

Oggetti e nomi (proibiti)

In R ci sono anche dei nomi non solo sconsigliati ma proprio **proibiti** che nonostante siano sintatticamente corretti, non possono essere usati (per ovvie ragioni):

```
mean <- 10 # ok ma sconsigliato
```

```
function <- 10
```

```
## Error: <text>:4:10: unexpected assignment
## 3:
## 4: function <-
##           ^
```

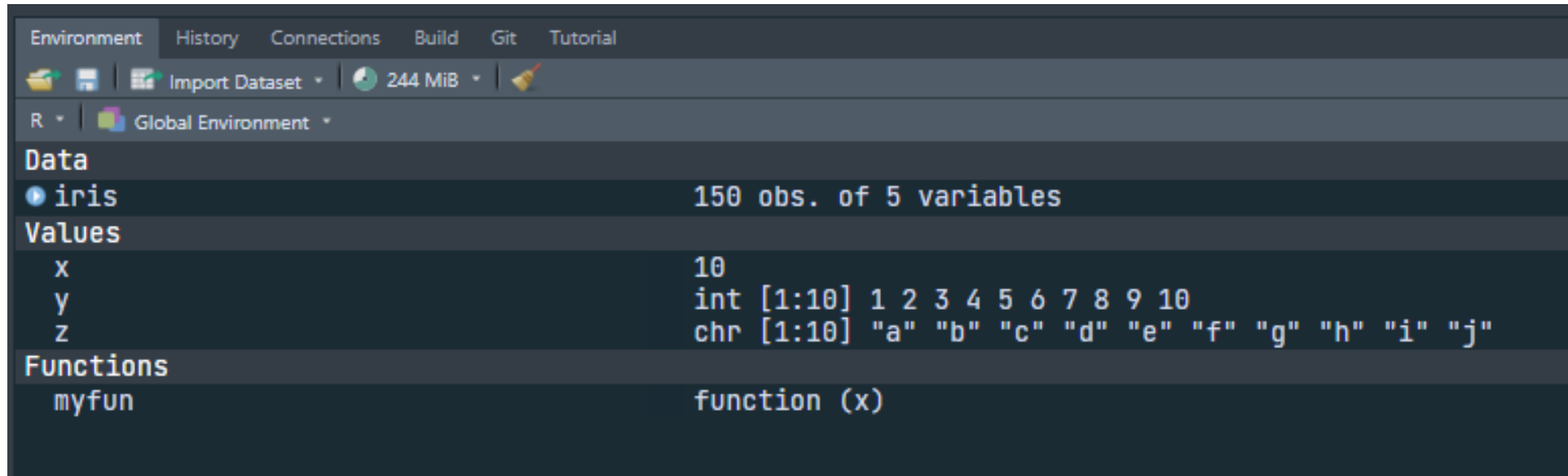
```
TRUE <- 4
```

```
## Error in TRUE <- 4: invalid (do_set) left-hand side to assignment
```

```
T <- 2 # attenzione
```

Dove viene creato l'oggetto?

Di default gli oggetti sono creati nel **global environment** accessibile con `ls()` o visibile in R Studio con anche alcune informazioni aggiuntive:



Non solo numeri (anticipazione)

Non solo numeri (anticipazione)

In R possiamo usare oltre ai numeri (in senso matematico) anche le **stringhe** ovvero parole, lettere interpretate così come sono:

```
"ciao" # stringa formata da 5 caratteri
```

```
## [1] "ciao"
```

```
x <- "ciao" # associa la stringa ad un oggetto
```

```
x + 1 # operazioni matematiche con stringhe (ha senso?)
```

```
## Error in x + 1: non-numeric argument to binary operator
```

```
x == "ciao"
```

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

```
x > 10
```

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

Funzioni

Funzioni

Le funzioni sono un argomento relativamente complesso ed avanzato. Lo tratteremo più avanti nella sezione **Funzioni** del capitolo Programmazione in R. Siccome le usiamo fin da subito è importante avere chiari alcuni aspetti:

- Funzioni come oggetti
- Argomenti obbligatori, opzionali e default
- Ordine degli argomenti
- Documentazione

Funzioni come oggetti

Abbiamo già visto che ogni cosa in R è un oggetto. Anche le funzioni seppur molto diverse da altri elementi sono creati e trattati in R come oggetti:

```
myfun <- function(x) {  
  return(x + 3)  
}  
  
ls()
```

```
## [1] "file"          "math_operators" "my.obj"         "my_obj"         "My_obj"  
## [6] "myfun"         "names_function" "pdf"            "T"              "x"
```

Possiamo crearle, eliminarle o sovrascriverle come un normale oggetto. Vedremo più avanti come crearle ma tenete in considerazione che tutte le funzioni che usiamo sono create come oggetti e salvati nell'ambiente (quando facciamo `library()` sono rese disponibili)

Argomenti

Gli argomenti delle funzioni sono quelli che da *utenti* dobbiamo conoscere ed impostare nel modo corretto per fare in modo che la funzioni faccia quello per cui è stata pensata. Vediamo l'`help` della funzione `mean()`

RDocumentation

base (version 3.6.2)

mean: Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

`mean(x, ...)`

Argomenti

- `x` è un oggetto (ovviamente 😊). [...] "currently there are methods for numeric/logical vectors...". Quindi `x` deve essere numerico o logico. Ha senso fare la media di caratteri? 😞
- `trim`
- `na.rm`

Per impostare questi argomenti ci sono 2 regole:

- l'ordine non conta SE DEFINISCO NOME DELL'ARGOMENTO con `x = vettore`, `na.rm = TRUE`, etc.
- l'ordine conta SE NON DEFINISCO IL NOME DELL'ARGOMENTO. Posso quindi omettere `argomento = valore` ma devo rispettare l'ordine con cui è stata scritta la funzione

Argomenti

In questo caso proviamo ad usare la funzione `mean()`:

```
myvec <- rnorm(100, 10, 5)
mean(myvec) # x definito, trim non definito, na.rm non definito
```

```
## [1] 9.847225
```

```
mean(myvec, trim = 0.10) # x definito, trim definito, na.rm non definito
```

```
## [1] 9.751822
```

```
mean(myvec, na.rm = TRUE) # x definito trim non definito, na.rm definito
```

```
## [1] 9.847225
```

```
mean(myvec, TRUE) # cosa succede?
```

```
## Error in mean.default(myvec, TRUE): 'trim' must be numeric of length one
```

Formula Syntax (extra, but useful)

- In R vedrete spesso l'utilizzo dell'operatore `~` per fare grafici, statistiche descrittive, modelli lineari etc. L'utilizzo di `y ~ x` permette di creare del codice R che non viene eseguito subito ma può essere eseguito successivamente in un ambiente specifico.
- è l'unico caso dove nomi non assegnati possono essere utilizzati senza errori
- questo tipo di programmazione si chiama **non-standard evaluation** perchè appunto non funziona come il solito codice R

```
y # y non esiste e quindi ho un errore
```

```
## Error in eval(expr, envir, enclos): object 'y' not found
```

```
y ~ x # usando ~ non ho errori perchè il codice non viene eseguito
```

```
## y ~ x  
## <environment: 0x000001e27d1fff38>
```

```
`~`(y, x) # l'operatore ~ non è altro che una funzione come quelle che abbiamo visto fino ad ora
```

```
## y ~ x  
## <environment: 0x000001e27d1fff38>
```

Formula Syntax (extra, but useful)

Le formule vengono utilizzate in tantissimi contesti.

Per fare modelli di regressione

```
# un modello lineare -> dipendente ~ indipendenti  
lm(y ~ x1 + x2)
```

Per fare aggregare un dataset

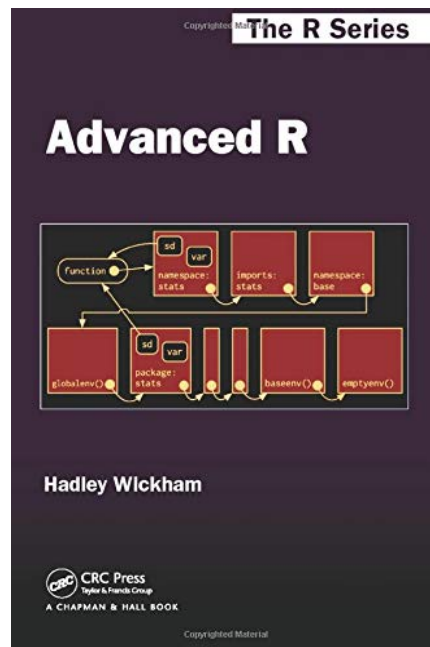
```
# per aggregare un dataset (vedremo più avanti :) )  
aggregate(y ~ x, data = data, FUN = mean)
```

Per fare grafici

```
# per fare grafici  
boxplot(y ~ x, data = data)
```

Formula Syntax (extra, but useful)

In generale, ogni volta che usate delle variabili *unquoted* (senza virgolette) e queste non sono dichiarate nell'ambiente, state probabilmente usando la **non-standard evaluation** e c'è una formula da qualche parte 😊. Per approfondire:



Capitolo Metaprogramming

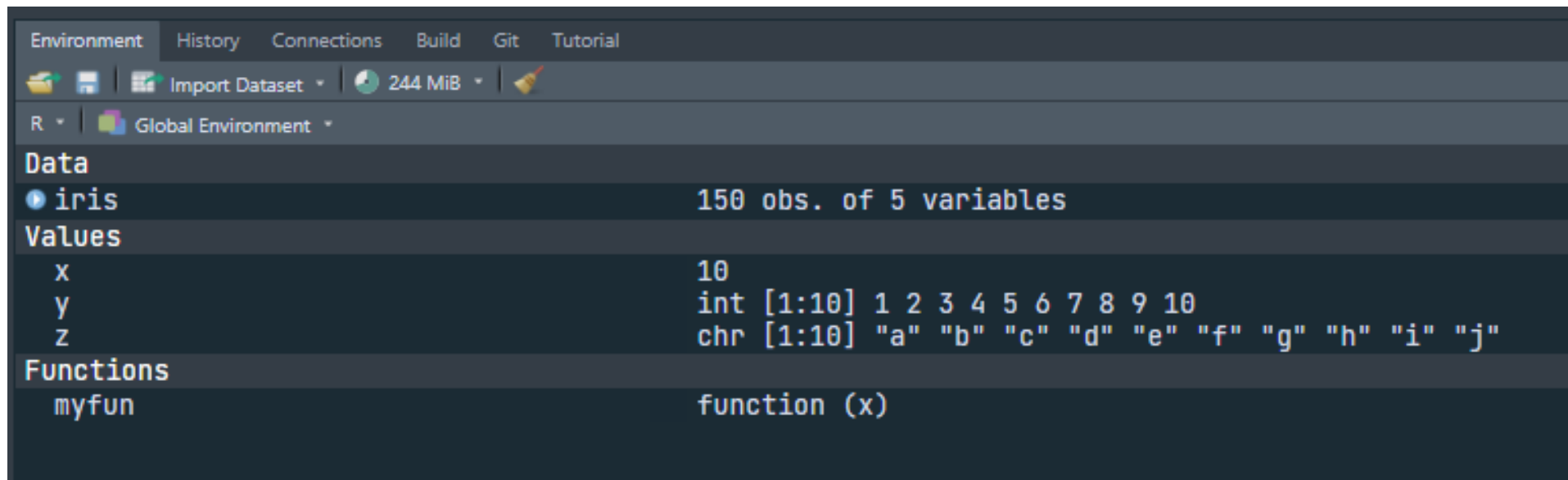
Ambiente di lavoro 

Ambiente di lavoro

- Environment
- Working directory
- Packages

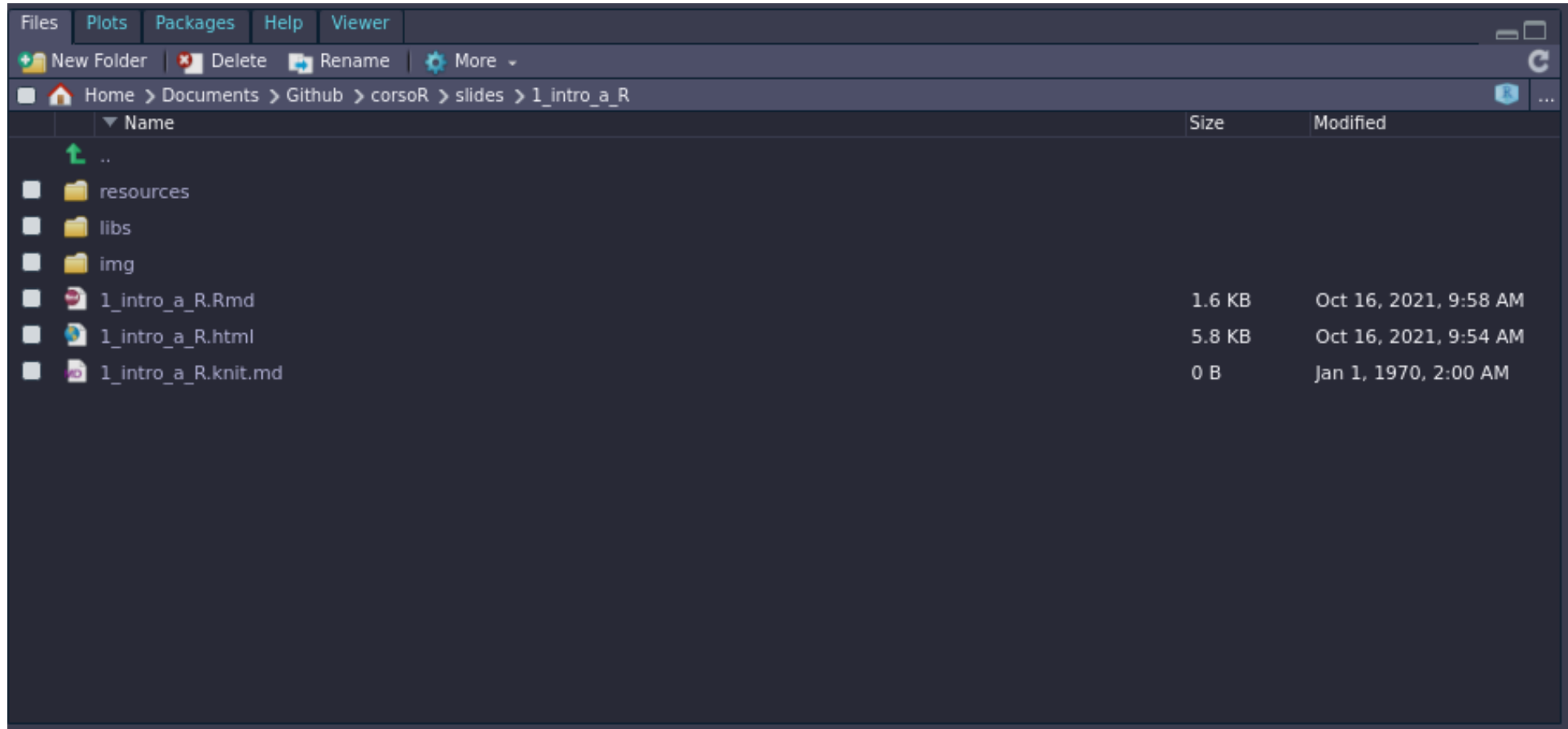
Environment

Il **working environment** è la vostra *scrivania* quando lavorate in R. Contiene tutti gli oggetti (variabili) creati durante la sessione di lavoro.



Working Directory

La working directory è la posizione (cartella) sul vostro PC dove R sta lavorando e nella quale R si aspetta di trovare i vostri file, se non specificato altrimenti



Packages

In R è possibile installare e caricare pacchetti aggiuntivi che non fanno altro che rendere disponibili librerie di funzioni create da altri utenti. Per utilizzare un pacchetto:

- Installare il pacchetto con `install.packages("nomepacchetto")`
- Caricare il pacchetto con `library(nomepacchetto)`
- Accedere ad una funzione senza caricare il pacchetto `nomepacchetto::nomefunzione()`. Utile se serve solo una funzione o ci sono conflitti

Packages



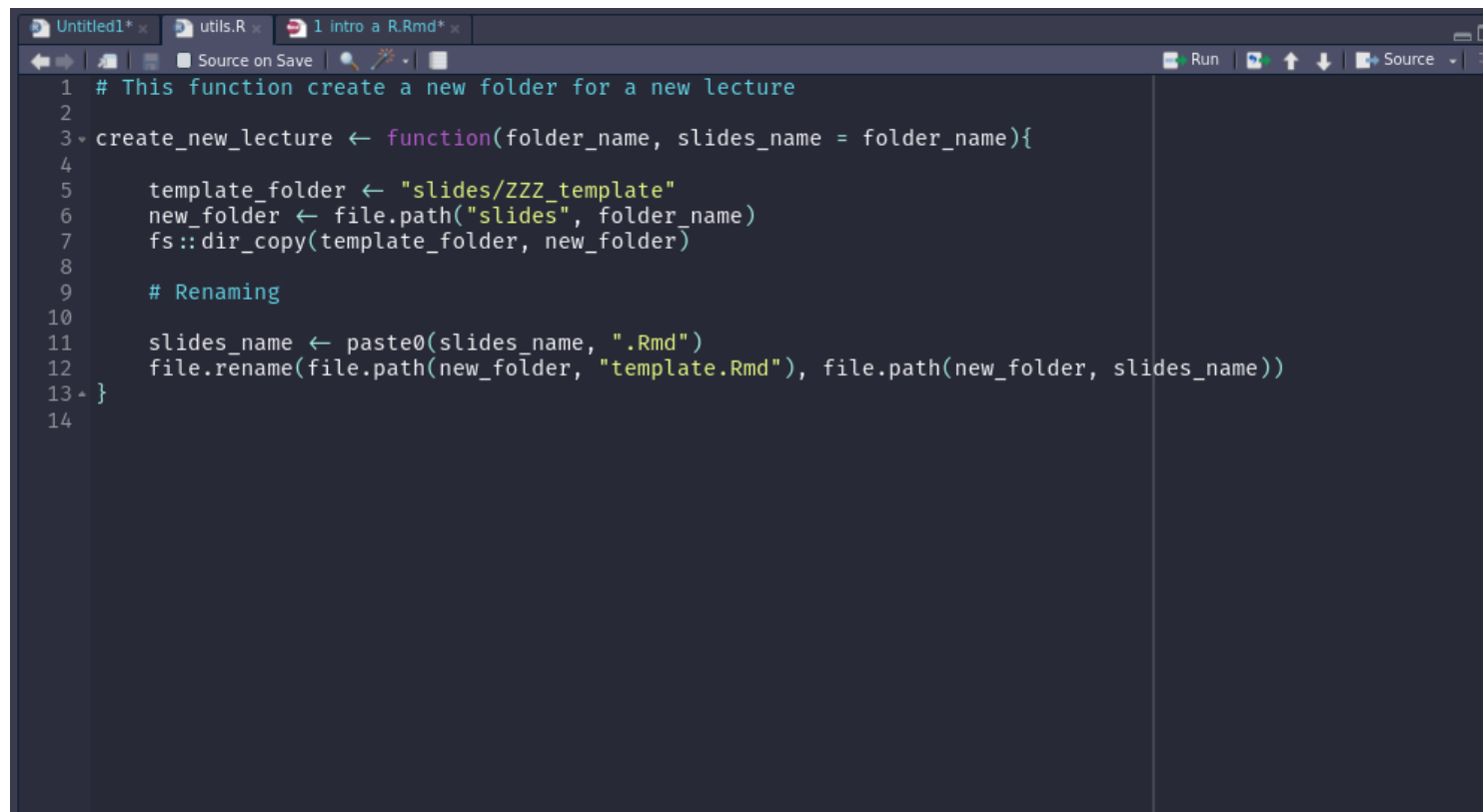
The screenshot shows the RStudio interface with the 'Packages' tab selected. The 'User Library' section is expanded, displaying a list of installed packages. Each row includes a checkbox, the package name, a brief description, the version number, and two circular icons representing the package's status (e.g., installed, updated).

	Name	Description	Version		
User Library					
<input type="checkbox"/>	abind	Combine Multidimensional Arrays	1.4-5		
<input type="checkbox"/>	anytime	Anything to 'POSIXct' or 'Date' Converter	0.3.9		
<input type="checkbox"/>	arrayhelpers	Convenience Functions for Arrays	1.1-0		
<input type="checkbox"/>	AsioHeaders	'Asio' C++ Header Files	1.16.1-1		
<input type="checkbox"/>	askpass	Safe Password Entry for R, Git, and SSH	1.1		
<input type="checkbox"/>	assertthat	Easy Pre and Post Assertions	0.2.1		
<input type="checkbox"/>	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.2.1		
<input type="checkbox"/>	base64enc	Tools for base64 encoding	0.1-3		
<input type="checkbox"/>	bayesplot	Plotting for Bayesian Models	1.8.1		
<input type="checkbox"/>	BayesRS	Bayes Factors for Hierarchical Linear Models with Continuous Predictors	0.1.3		
<input type="checkbox"/>	bayestestR	Understand and Describe Bayesian Models and Posterior Distributions	0.11.0		
<input type="checkbox"/>	BH	Boost C++ Header Files	1.75.0-0		
<input type="checkbox"/>	binom	Binomial Confidence Intervals For Several Parameterizations	1.1-1		
<input type="checkbox"/>	bitops	Bitwise Operations	1.0-7		
<input type="checkbox"/>	blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.2		
<input type="checkbox"/>	bookdown	Authoring Books and Technical Documents with R Markdown	0.24		
<input type="checkbox"/>	bridgesampling	Bridge Sampling for Marginal Likelihoods and Bayes Factors	1.1-2		
<input type="checkbox"/>	brio	Basic R Input Output	1.1.2		
<input type="checkbox"/>	brms	Bayesian Regression Models using 'Stan'	2.16.1		
<input type="checkbox"/>	Broddingnag	Very Large Numbers in R	1.2-6		
<input type="checkbox"/>	broom	Convert Statistical Objects into Tidy Tibbles	0.7.9		

Come lavorare in R

Scrivere e organizzare script

- Lo script è un file di testo dove il codice viene salvato e può essere lanciato in successione
- Nello script è possibile combinare **codice** e **commenti**



```
1 # This function create a new folder for a new lecture
2
3 create_new_lecture <- function(folder_name, slides_name = folder_name){
4
5     template_folder <- "slides/ZZZ_template"
6     new_folder <- file.path("slides", folder_name)
7     fs::dir_copy(template_folder, new_folder)
8
9     # Renaming
10
11     slides_name <- paste0(slides_name, ".Rmd")
12     file.rename(file.path(new_folder, "template.Rmd"), file.path(new_folder, slides_name))
13 }
14
```

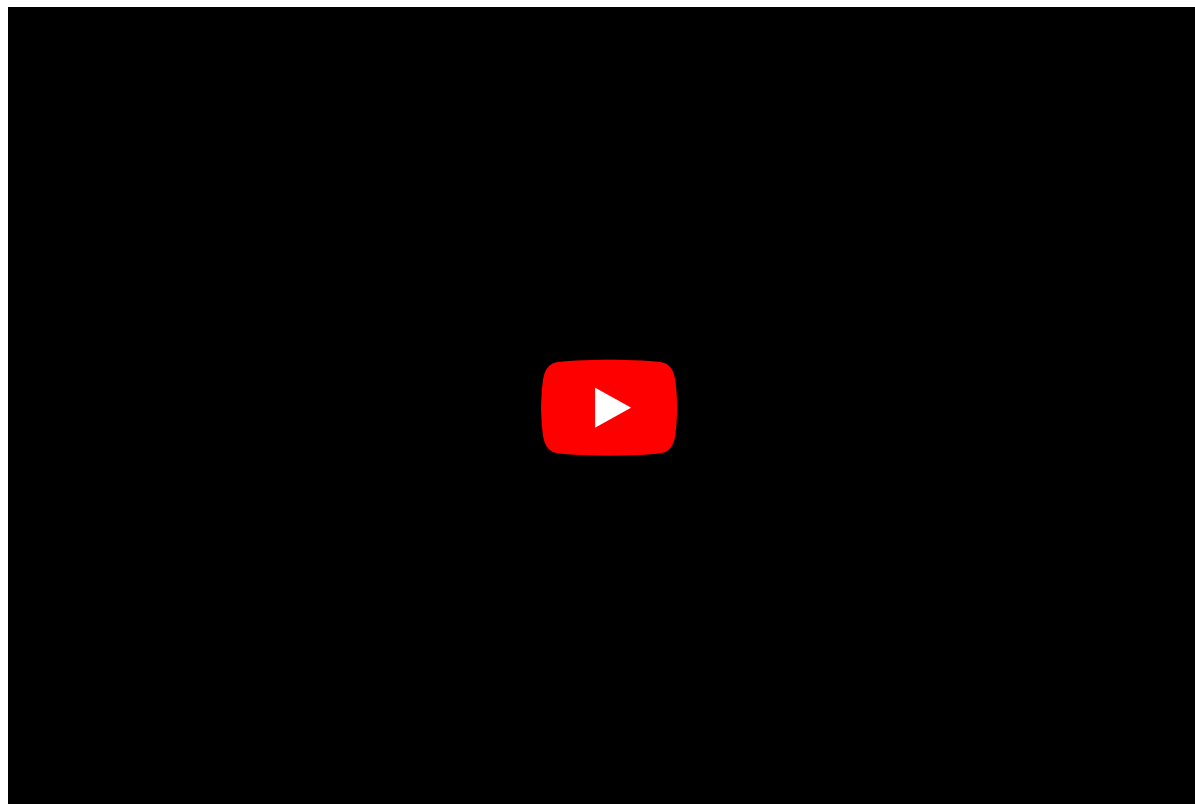

R Projects

Gli `R projects` sono una feature implementata in R Studio per organizzare una cartella di lavoro

- permettono di impostare la **working directory** in automatico
- permettono di usare **relative path** invece che **absolute path**
- rendono più **riproducibile** e **trasportabile** il progetto
- permettono un **veloce accesso** ad un determinato progetto

R Projects

Per capire meglio il funzionamento degli R `R projects` e di come sono organizzati i file ho fatto un video che può chiarire la questione:



Come risolvere i problemi ~~nella vita~~ in R

Come risolvere i problemi ~~nella vita~~ in R

In R gli errori sono:

- inevitabili
- parte del codice stesso
- educativi

Resta solo da capire come affrontarli



R ed errori

Ci sono diversi livelli di **allerta** quando scriviamo codice:

- **messaggi**: la funzione ci restituisce qualcosa che è utile sapere, ma tutto liscio
- **warnings**: la funzione ci informa di qualcosa di *potenzialmente* problematico, ma (circa) tutto liscio
- **error**: la funzione non solo ci informa di un **errore** ma le operazioni richieste non sono state eseguite

Ne vedremo e vedrete molti usando R 😊

Come risolvere un errore?

- capire il messaggio
- leggere la documentazione della funzione
- cercare il messaggio su Google
- chiedere aiuto nei forum dedicati



Come risolvere un errore?

- Ogni funzione ha una pagina di documentazione accessibile con `?nomefunzione`, `??nomefunzione` oppure `help(nomefunzione)`
- Possiamo cercare anche la documentazione del pacchetto
- Possiamo cercare su Google il nome della funzione o l'eventuale messaggio che riceviamo

```
remove {base} R Documentation  
  
Remove Objects from a Specified Environment  
  
Description  
remove and rm can be used to remove objects. These can be specified successively as character strings, or in the character vector list, or through a combination of both.  
All objects thus specified will be removed.  
  
If envir is NULL then the currently active environment is searched first.  
  
If inherits is TRUE then parents of the supplied directory are searched until a variable with the given name is encountered. A warning is printed for each variable that is not found.  
  
Usage  
  
remove(..., list = character(), pos = -1,  
        envir = as.environment(pos), inherits = FALSE)  
  
rm      (... , list = character(), pos = -1,  
        envir = as.environment(pos), inherits = FALSE)
```

help(rm)

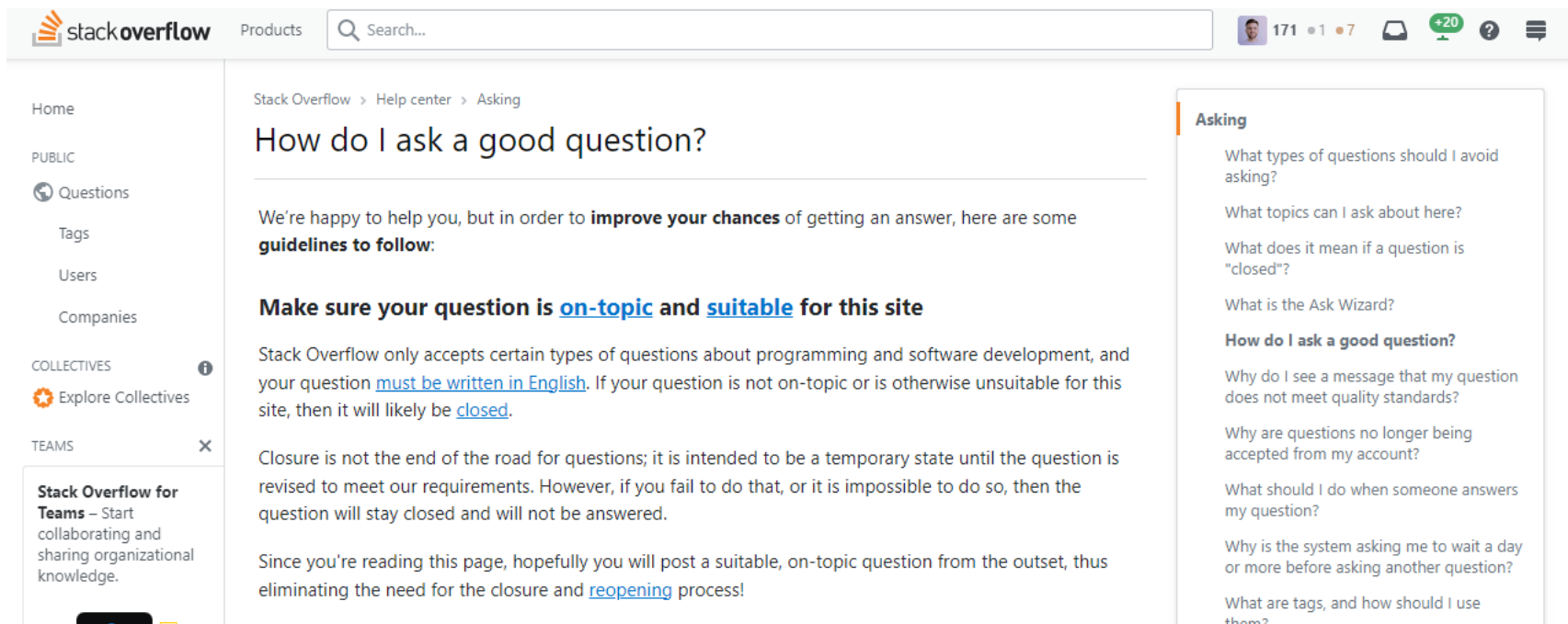
Stack overflow

Stack overflow è un forum di discussione riguardo qualsiasi cosa coinvolga codice (statistica, programmazione, etc.). E' pieno di errori comuni, *How to do ...* e di risposte/soluzioni estremamente utili. Nel 90% dei casi il problema che avete è comune ed è già presente una soluzione.

The screenshot shows the Stack Overflow interface. At the top, there's a navigation bar with the Stack Overflow logo, a search bar, and user statistics (171, 1, 7, +20). The left sidebar contains navigation links: Home, PUBLIC, Questions (selected), Tags, Users, Companies, COLLECTIVES (with an info icon), Explore Collectives, and TEAMS (with a close icon). The main content area displays a question titled "How to make a great R reproducible example" with a blue "Ask Question" button. Below the title, it says "Asked 11 years, 5 months ago Modified 2 months ago Viewed 414k times". A blue banner indicates that the question's answers are a community effort and are not currently accepting new answers. The question text reads: "When discussing performance with colleagues, teaching, sending a bug report or searching for guidance on mailing lists and here on Stack Overflow, a [reproducible example](#) is often asked and always helpful." Below this, the question asks for tips on creating an excellent example, pasting data structures from R in a text format, and what other information to include. The right sidebar features "The Overflow Blog" with two articles: "Introducing the Ask Wizard: Your guide to crafting high-quality questions" and "How to get more engineers entangled with quantum computing (Ep. 501)". It also includes a "Featured on Meta" section with a link to "The 2022 Community-a-thon has begun!".

Se non trovo una soluzione?

Se non trovo una soluzione posso chiedere. Fare una domanda riguardo un errore o un problema di codice non è semplice come sembra. Le fonti dell'errore possono essere molteplici (il mio specifico computer, un pacchetto che ho installato, il codice sorgente etc.). Ecco una guida per chiedere in modo efficace:



<https://stackoverflow.com/help/how-to-ask>