Evolutionary Genetics

LV 25600-01 | Lecture with exercises | 4KP

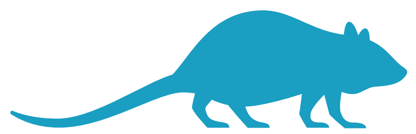
Bioinformatics

Jean-Claude Walser
jean-claude.walser [at] usys.ethz.ch

HS2024

What is

Bioinformatics?



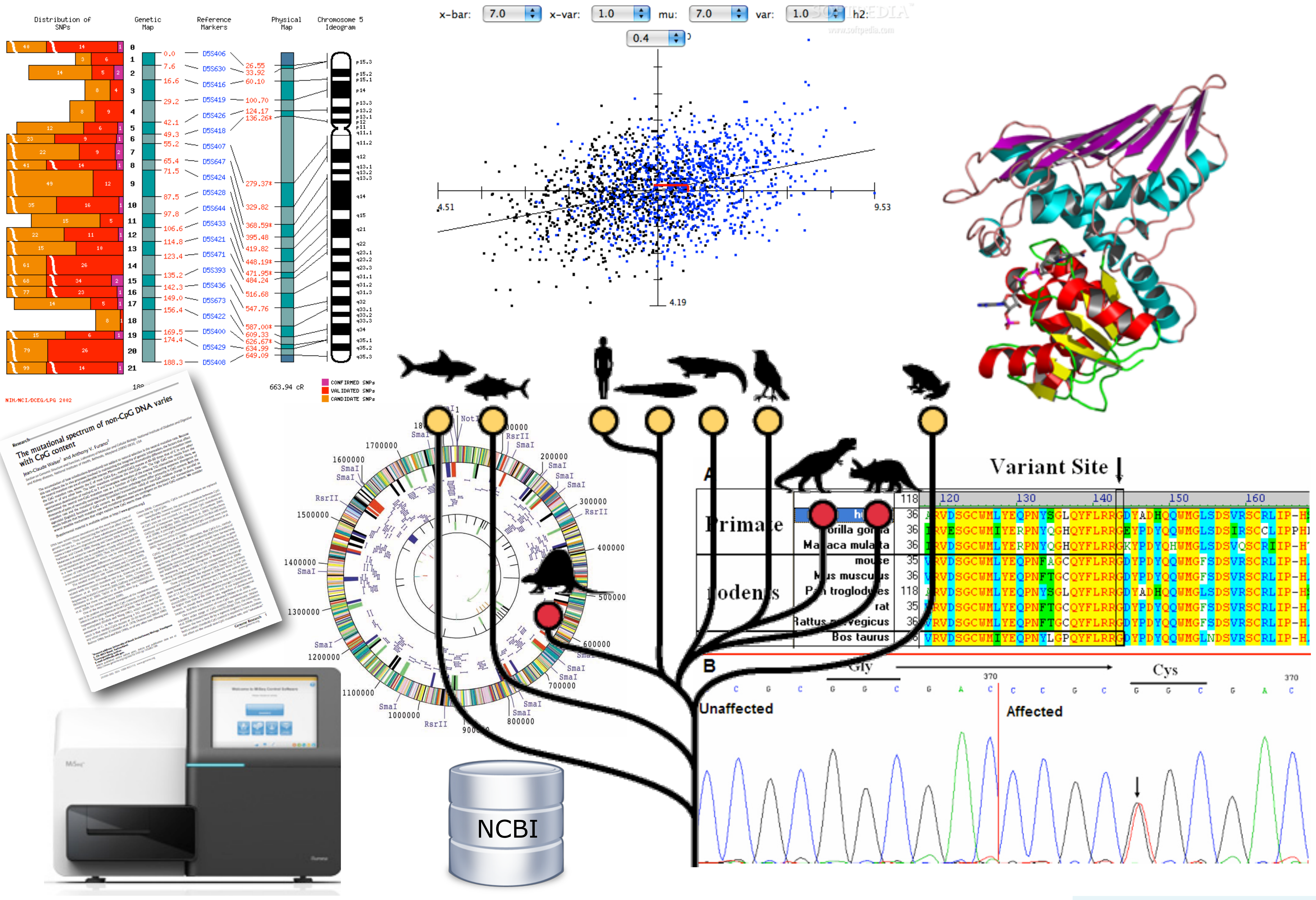
in vivo - in vitro - in silico

Do I need to know bioinformatics ?

Yes, absolutely. You don't have to be an expert* (or geek) but you should know the basics.

"An **expert** is a person who has made all the mistakes that can be made in a very narrow field." -Niels Bohr

Bioinformatics ▷ Introduction



Introduction

Imagine, you are part of the **Covid broadcast team** and you are responsible for the **daily statistics**. All laboratories conducting Covid testing have to send daily reports. It is your duty to collect these reports. The labs are using standardised **CSV files**. At the end of a day you have **hundreds of CSV files** and you need to add them to your existing table. Your team-leader recommends to use the **spreadsheet editor** Microsoft Excel to store the data because he already has create a summary report template. Any questions?

The Guardian

CovidNews: how Excel may have caused loss of 16,000 test results in England

Public Health England data error blamed on limitations of Microsoft spreadsheet

One lab had sent its daily test report to PHE (Public Health England) in the form of a **CSV file** – the simplest possible database format, just a list of values separated by commas. That report was then **loaded into Microsoft Excel**, and the new tests at the bottom were added to the main database.

But while CSV files can be any size, Microsoft Excel files can only be 1,048,576 rows long – or, in older versions which PHE may have still been using, a mere 65,536. When a CSV file longer than that is opened, the bottom rows get cut off and are no longer displayed. That means that, once the lab had performed more than a million tests, it was only a matter of time before its reports failed to be read by PHE.

Alex Hern *UK technology editor*

@alexhern

Tue 6 Oct 2020 08.21 BST

Introduction

Microsoft's spreadsheet software is one of the world's most popular business tools, but it is regularly implicated in errors which can be costly, or even dangerous, because of the ease with which it can be used in situations it was not designed for.

In 2013, an Excel error at JPMorgan masked the loss of almost $6bn (£4.6bn), after a cell mistakenly divided by the sum of two interest rates, rather than the average. The news led James Kwak, a professor of law at the University of Connecticut, to warn that Excel is "incredibly fragile".

"There is no way to trace where your data comes from, there's no audit trail (so you can overtype numbers and not know it), and there's no easy way to test spreadsheets, for starters. The biggest problem is that anyone can create Excel spreadsheets – badly. Because it's so easy to use, the creation of even important spreadsheets is not restricted to people who understand programming and do it in a methodical, well-documented way," Kwak wrote.

Errors from the spreadsheet software have even changed the very foundations of human genetics. The names of 27 genes have been changed over the past year by the Human Gene Nomenclature Committee, after Microsoft's program continually misformatted them. The genes SEPT1 and MARCH1, for instance, have been changed to SEPTIN1 and MARCHF1 after they were repeatedly turned into dates, while symbols that were common words have been altered so that grammar tools didn't autocorrect them: WARS is now WARS1, for instance.

The Guardian

Alex Hern *UK technology editor*

@alexhern

Tue 6 Oct 2020 08.21 BST

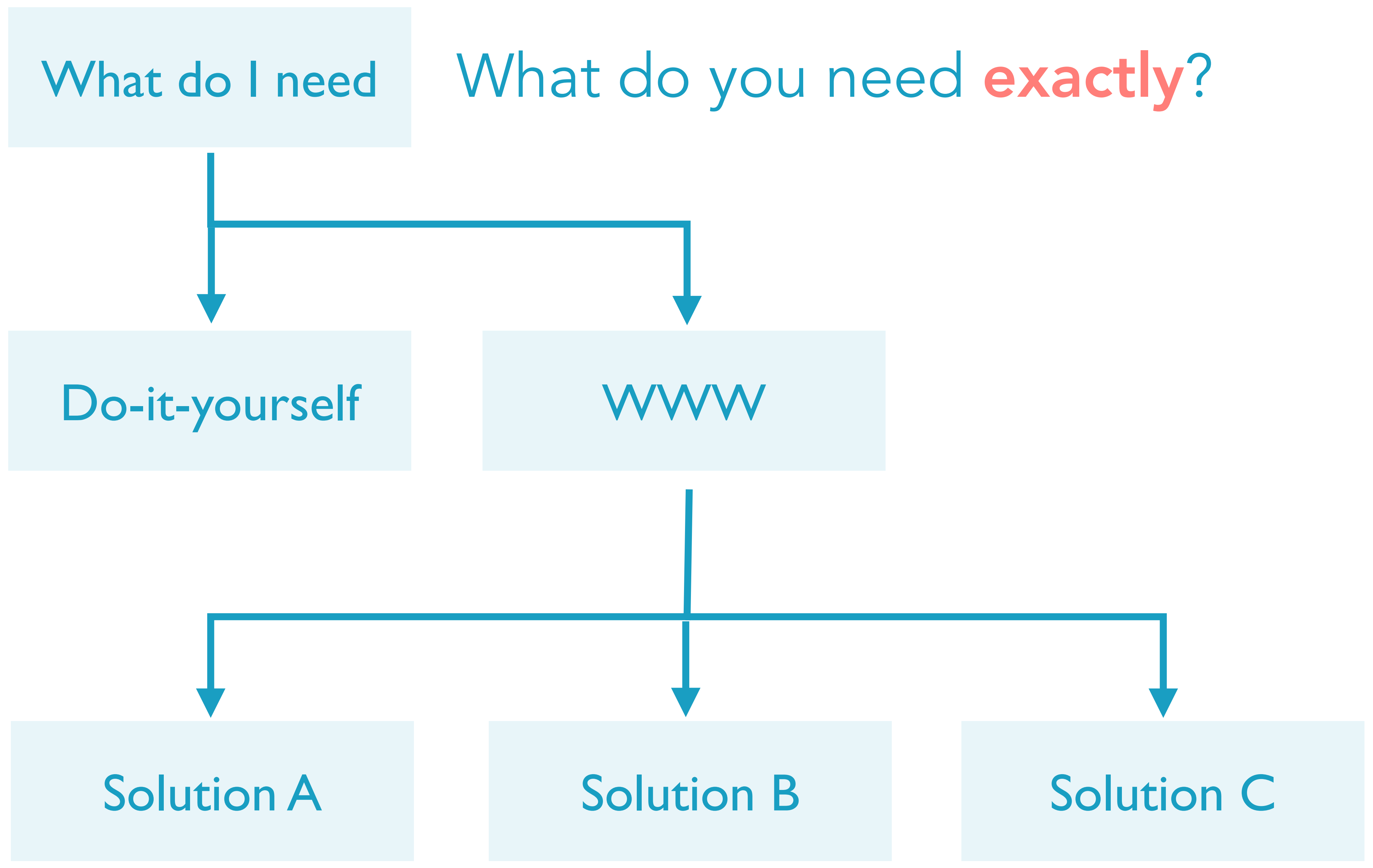
Introduction

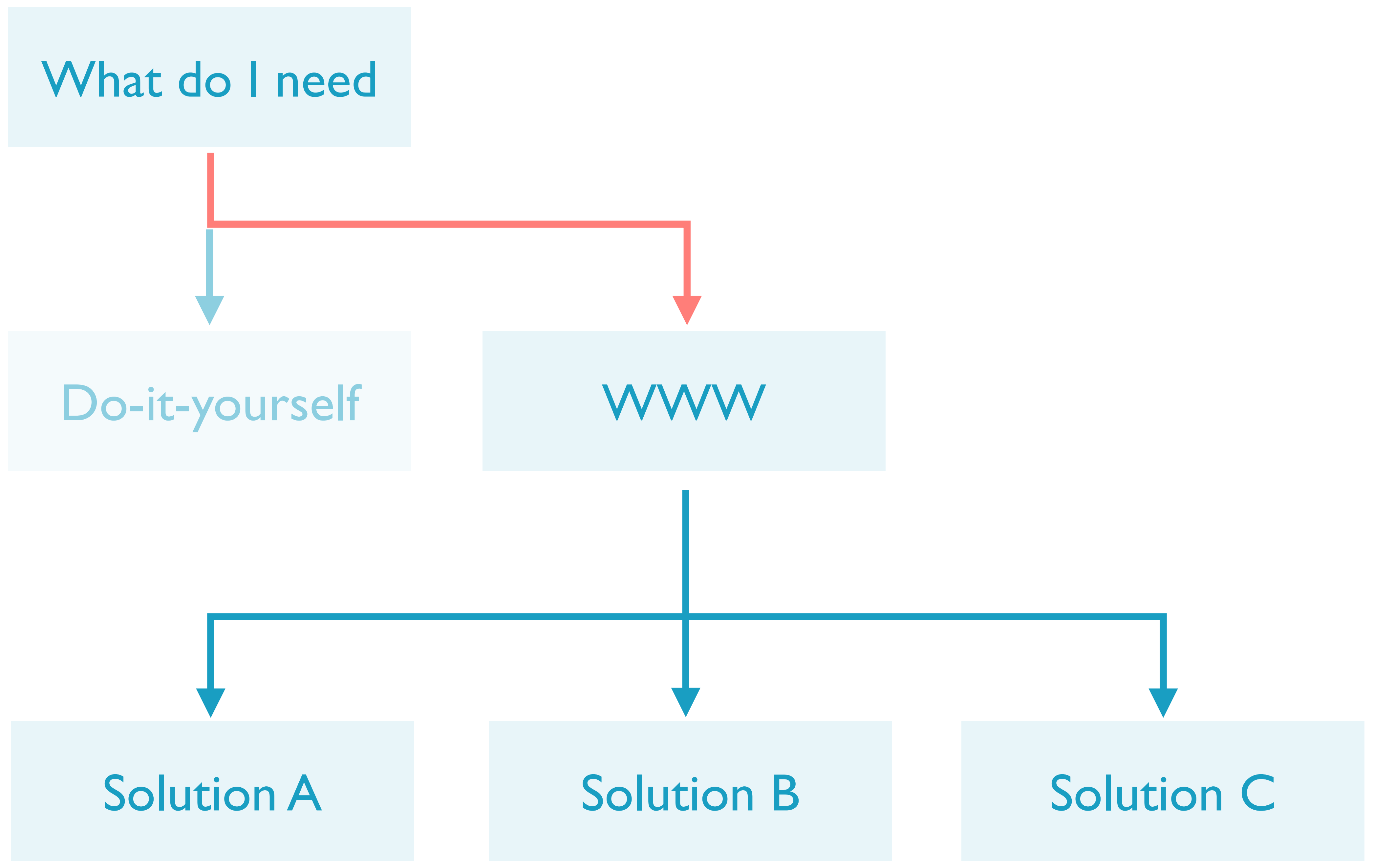
"A new scientific truth does not triumph by convincing its opponents and making them see the light, but rather because its opponents eventually die, and a new generation grows up that is familiar with it."

—Max Planck

Applications

What do you need **exactly**?




What do I need
Do-it-yourself
WWW
Solution A
Solution B
Solution C

Introduction

Bioinformatics ▷ Introduction

- What should you know **before** you download?
- What should you do **before** you start analysing your data?
- Where can you get **help**?
- What do you need for the **publication / report**?

What should you know **before** you download?

- support / history / usage
- requirements
- limitations
- type (e.g. freeware)
- in- and output file format
- last update / update history
- literature search
- what language is used

What should you do **before** you start analysing your data?

- avoid installation errors
- understand installation warnings
- run "simple" tests and/or examples
- explore limitations
- check in- and output format
- speed and resource managment

Where can you get **help**?

- manual(s) ☛ RFM!
- help options (e.g. help button, -help, --h)
- README file(s)
- log file(s)
- user forum
- author(s)

What do you need for the **publication / report** ?

- literature reference or web link
- version
- parameters (better: command line)

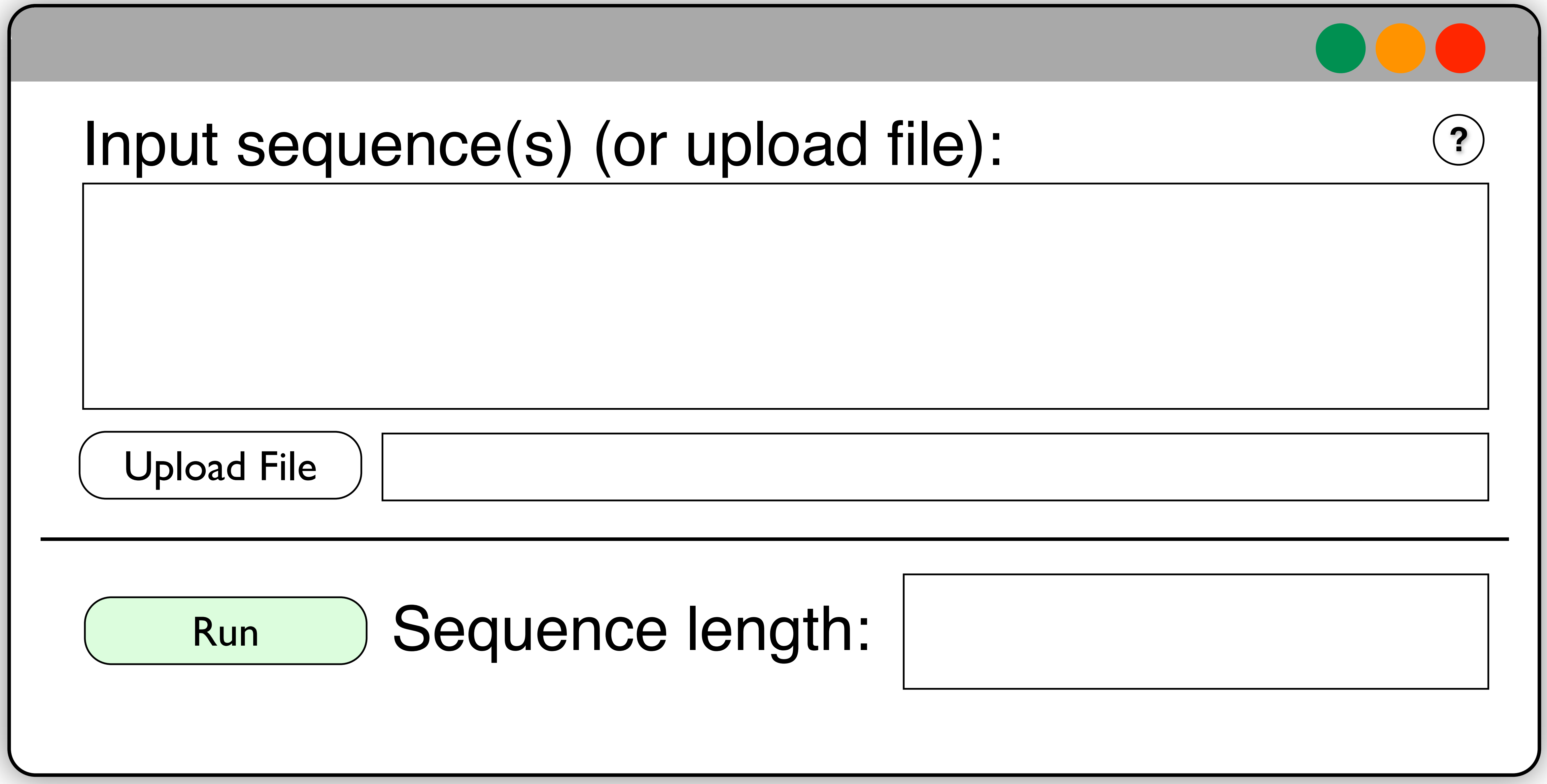
Introduction

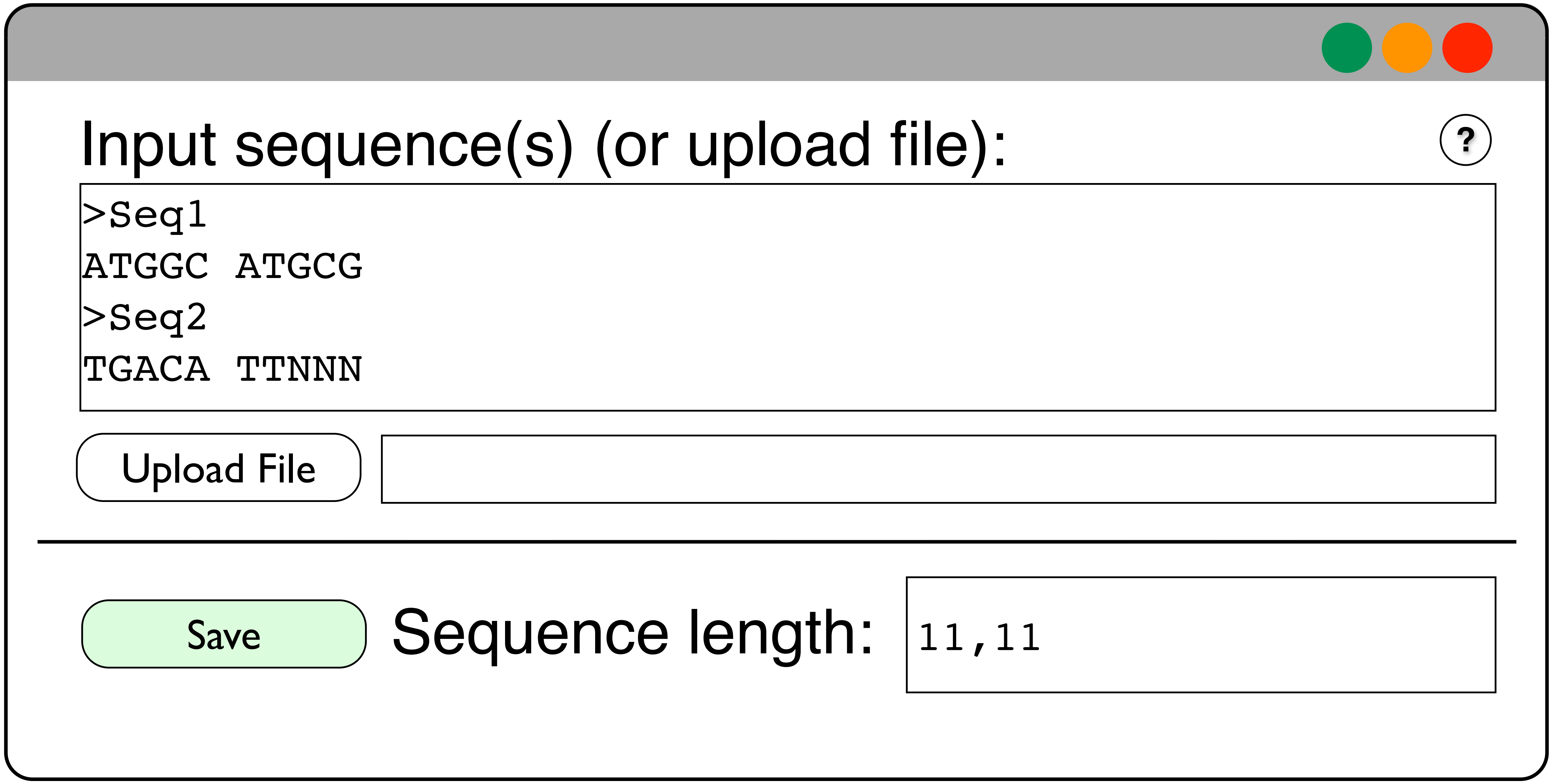
Freeware (from "free" and "software") is computer software that is available for use at no cost or for an optional fee (donation).

The **GNU General Public License** (GNU GPL or simply GPL) is the most widely used free software license.

Commercial software, or less commonly, **payware**, is computer software that is produced for sale or that serves commercial purposes.

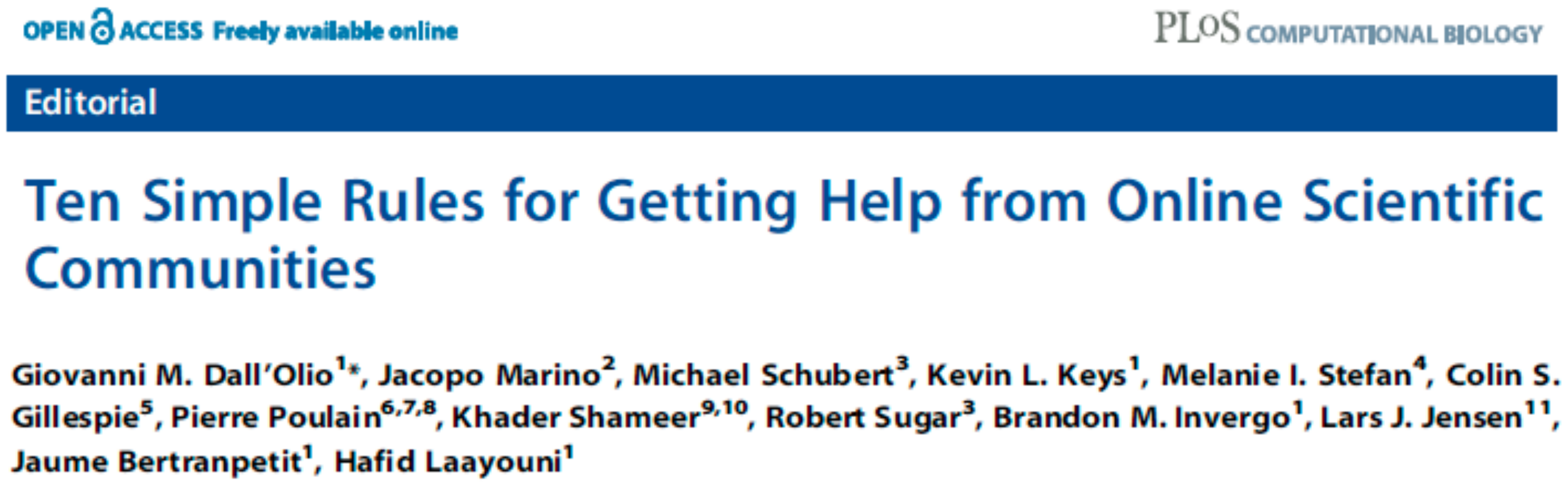
The term **shareware** (also known as trialware or demoware) refers to proprietary software that is provided to users without payment on a trial basis and is often limited by any combination of functionality, availability or convenience.


Input sequence(s) (or upload file):
?
Upload File
Run
Sequence length:


Input sequence(s) (or upload file):

```
>Seq1
ATGGC ATGCG
>Seq2
TGACA TTNNN
```

Sequence length: 11,11


OPEN ACCESS Freely available online

PLOS COMPUTATIONAL BIOLOGY

Editorial

Ten Simple Rules for Getting Help from Online Scientific Communities

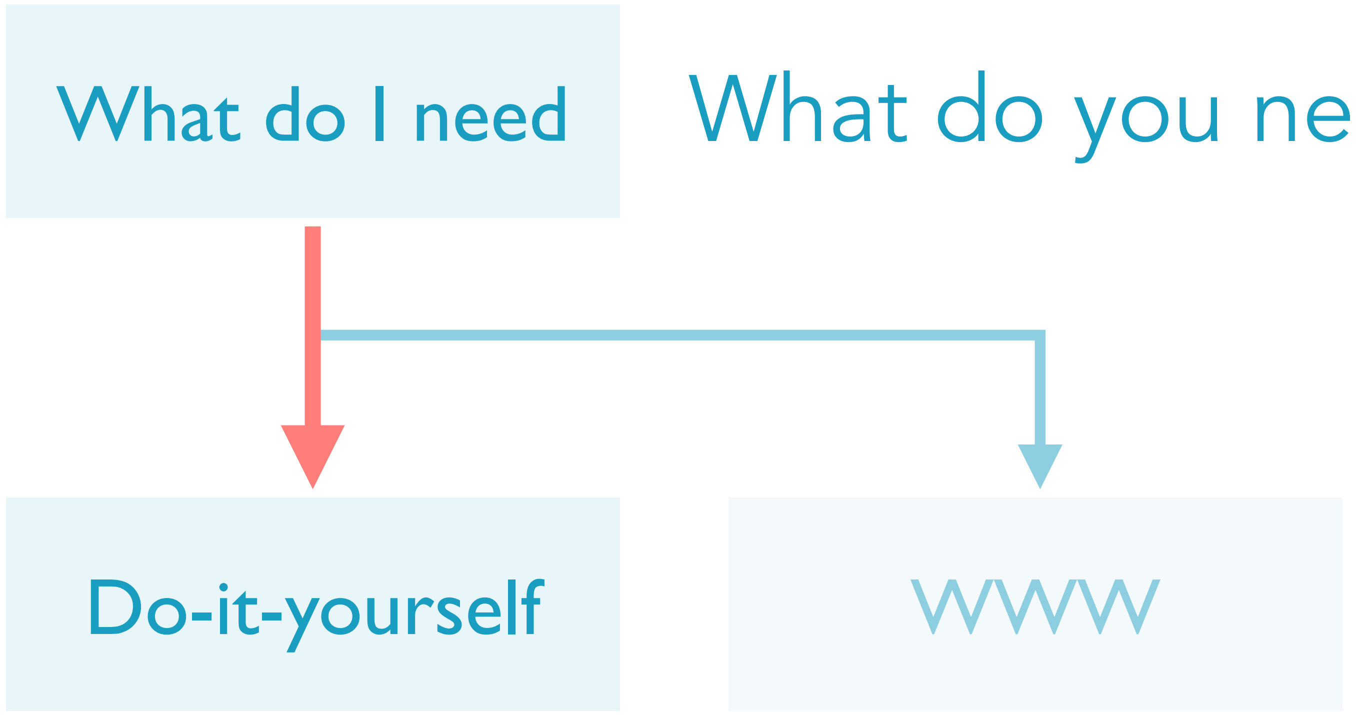
Giovanni M. Dall'Olio[1*], Jacopo Marino[2], Michael Schubert[3], Kevin L. Keys[1], Melanie I. Stefan[4], Colin S. Gillespie[5], Pierre Poulain[6,7,8], Khader Shameer[9,10], Robert Sugar[3], Brandon M. Invergo[1], Lars J. Jensen[11], Jaume Bertranpetit[1], Hafid Laayouni[1]

Netiquette

Rule 1. Do Not Be Afraid to Ask a Question
Rule 2. State the Question Clearly
Rule 3. New to a Mailing List? Learn the Established Customs before Posting
Rule 4. Do Not Ask What Has Already Been Answered
Rule 5. Always Use a Good Title
Rule 6. Do Your Homework before Posting
Rule 7. Proofread your Post and Write in Correct English
Rule 8. Be Courteous to Other Forum Members
Rule 9. Remember That the Archive of Your Discussion Can Be Useful to Other People
Rule 10. Give Back to the Community

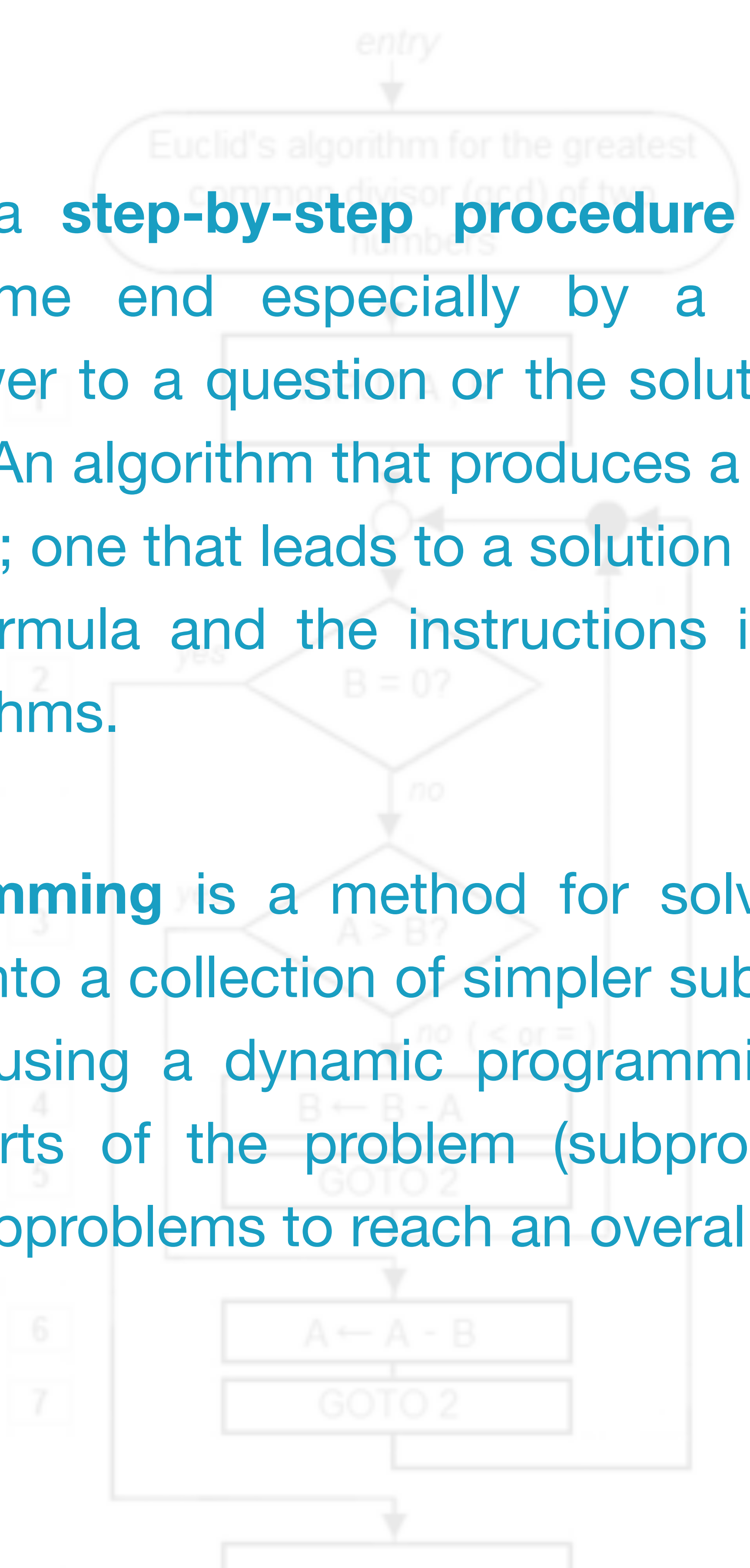
Bioinformatics ▷ **Introduction**

What do you need **exactly**?



PROGRAMMING / SCRIPTING IN BIOINFORMATICS

- **R**
- **Python (BioPython)**
- Perl (BioPerl)
- SQL
- C and C++ (Bio++)
- Ruby (BioRuby)
- PHP and JavaScript
- Java (BioJava)
- Go (BioGo)
- **Shell Scripting**


entry
Euclid's algorithm for the greatest common divisor (gcd) of two numbers
B = 0?
yes
no
A > B?
no (< or =)
B ← B - A
GOTO 2
A ← A - B
GOTO 2
1
2
3
4
5
6
7

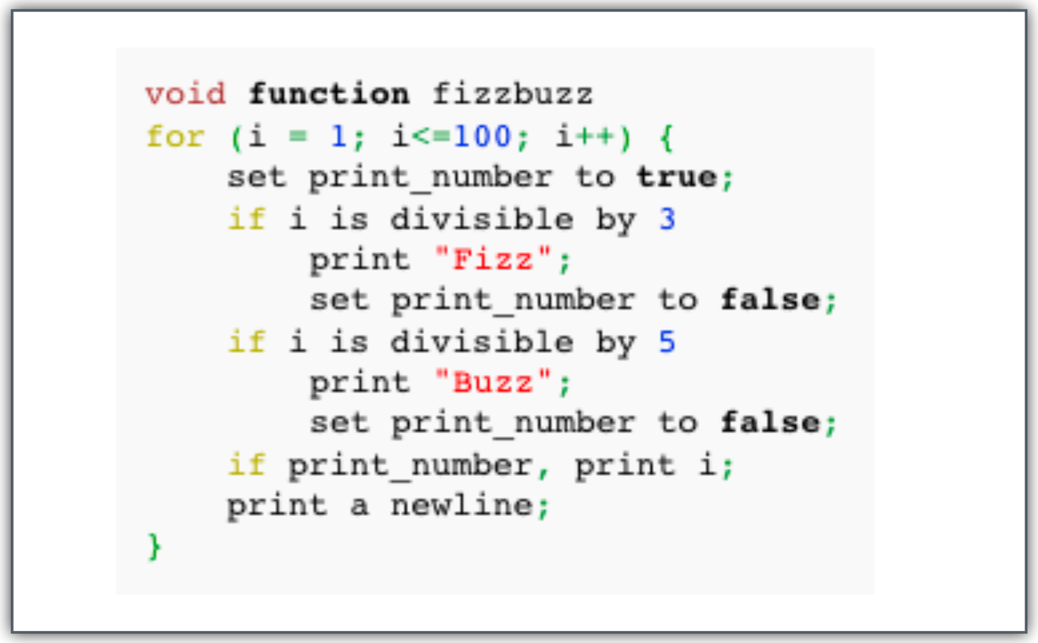
An **algorithm** is a **step-by-step procedure** for solving a problem or accomplishing some end especially by a computer. Procedure that produces the answer to a question or the solution to a problem in **a finite number of steps**. An algorithm that produces a yes or no answer is called a decision procedure; one that leads to a solution is a computation procedure. A mathematical formula and the instructions in a computer program are examples of algorithms.

Dynamic Programming is a method for solving complex problems by breaking it down into a collection of simpler subproblems. In order to solve a given problem using a dynamic programming approach, we need to solve different parts of the problem (subproblems), then combine the solutions of the subproblems to reach an overall solution.

Bioinformatics ▷ Introduction

Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural conventions of a programming language, but is **intended for human reading** rather than machine reading.

```
1.. If student's grade is greater than or equal to 20
    Print "passed"
else
    Print "failed"
```

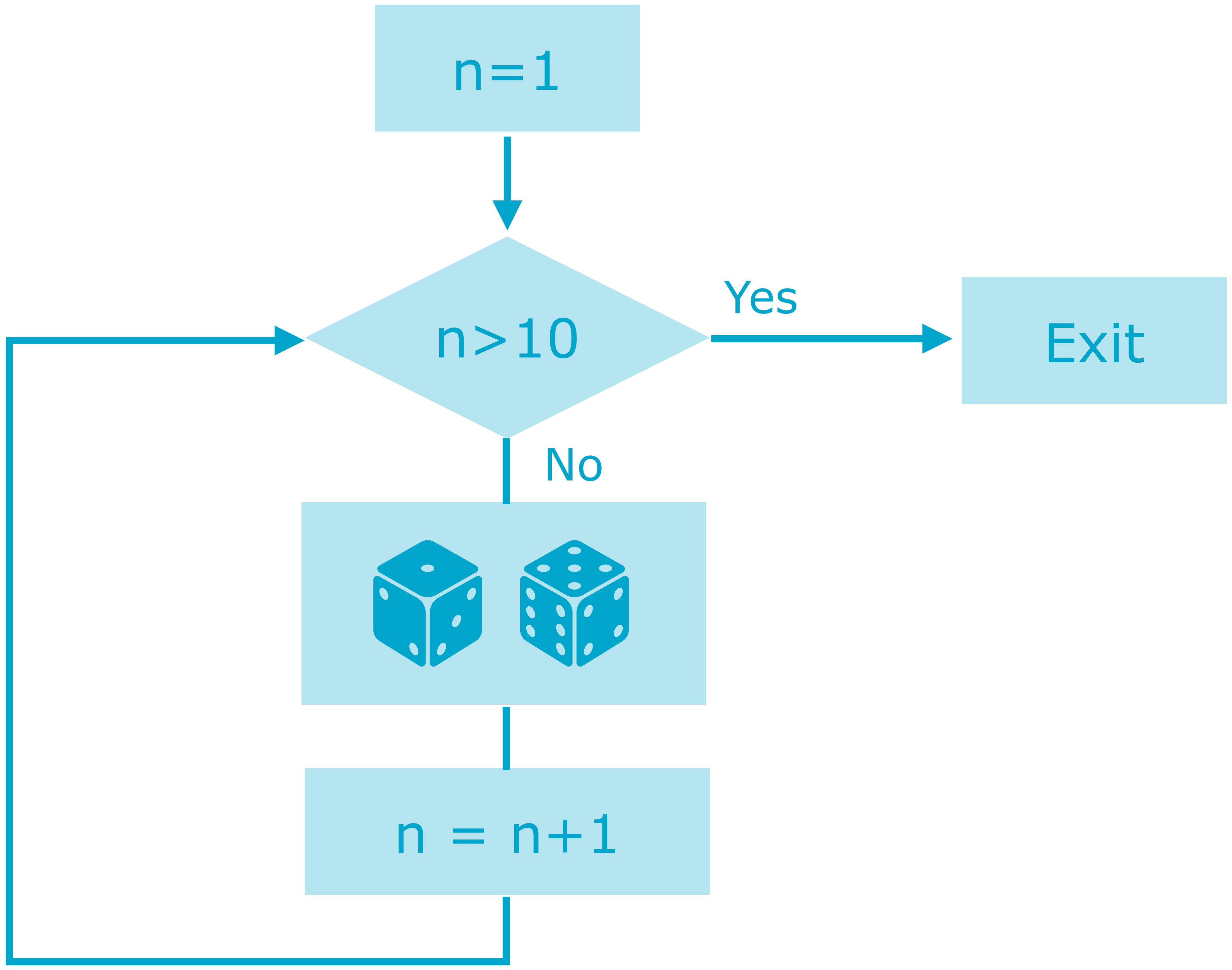


```
void function fizzbuzz
for (i = 1; i<=100; i++) {
    set print_number to true;
    if i is divisible by 3
        print "Fizz";
        set print_number to false;
    if i is divisible by 5
        print "Buzz";
        set print_number to false;
    if print_number, print i;
    print a newline;
}
```

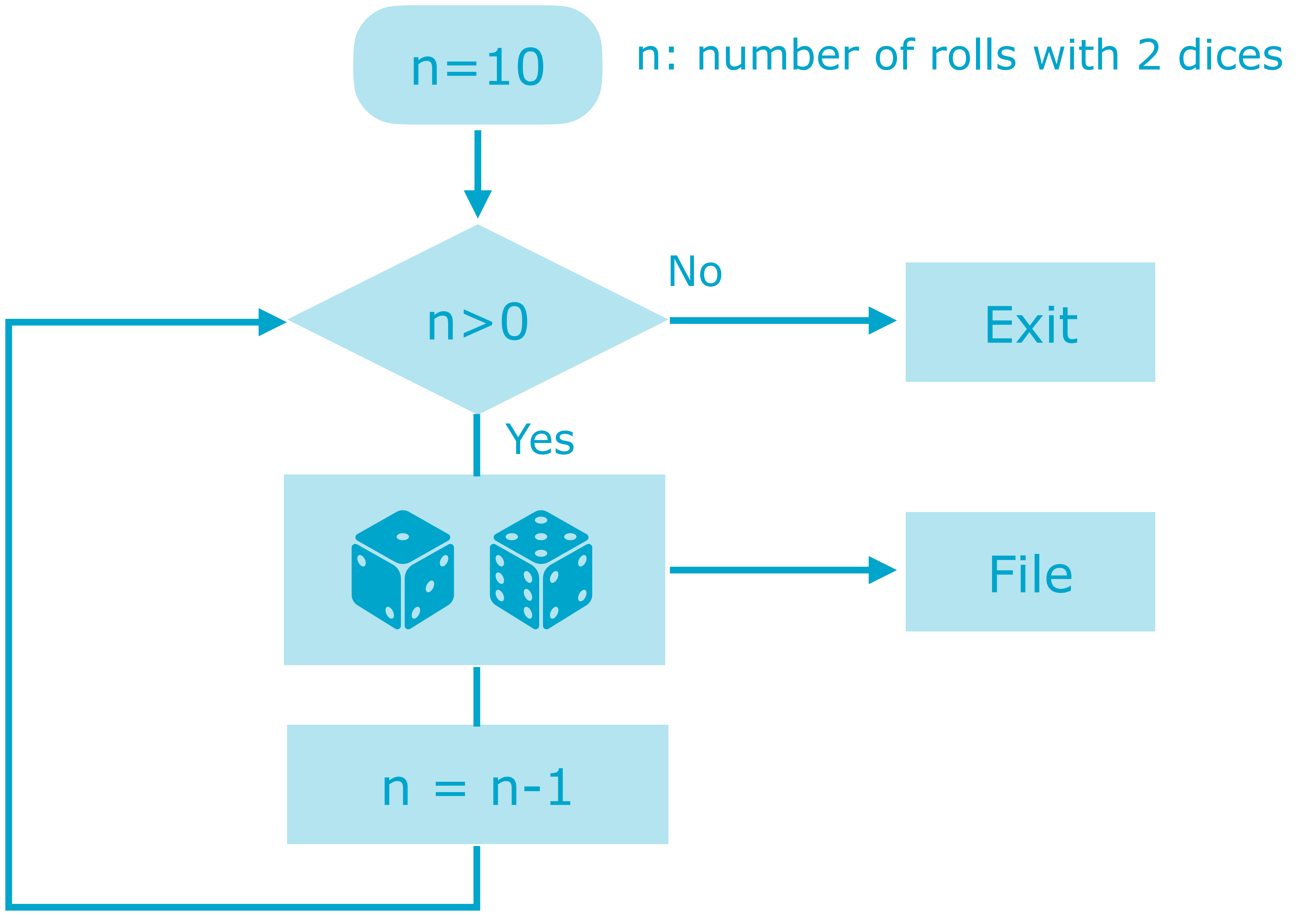
Write some pseudocode for a robot vacuum cleaner.

Example:

```
1. go straight until you hit something - stop
2. turn around 45° - start again
3. go to step 1
```

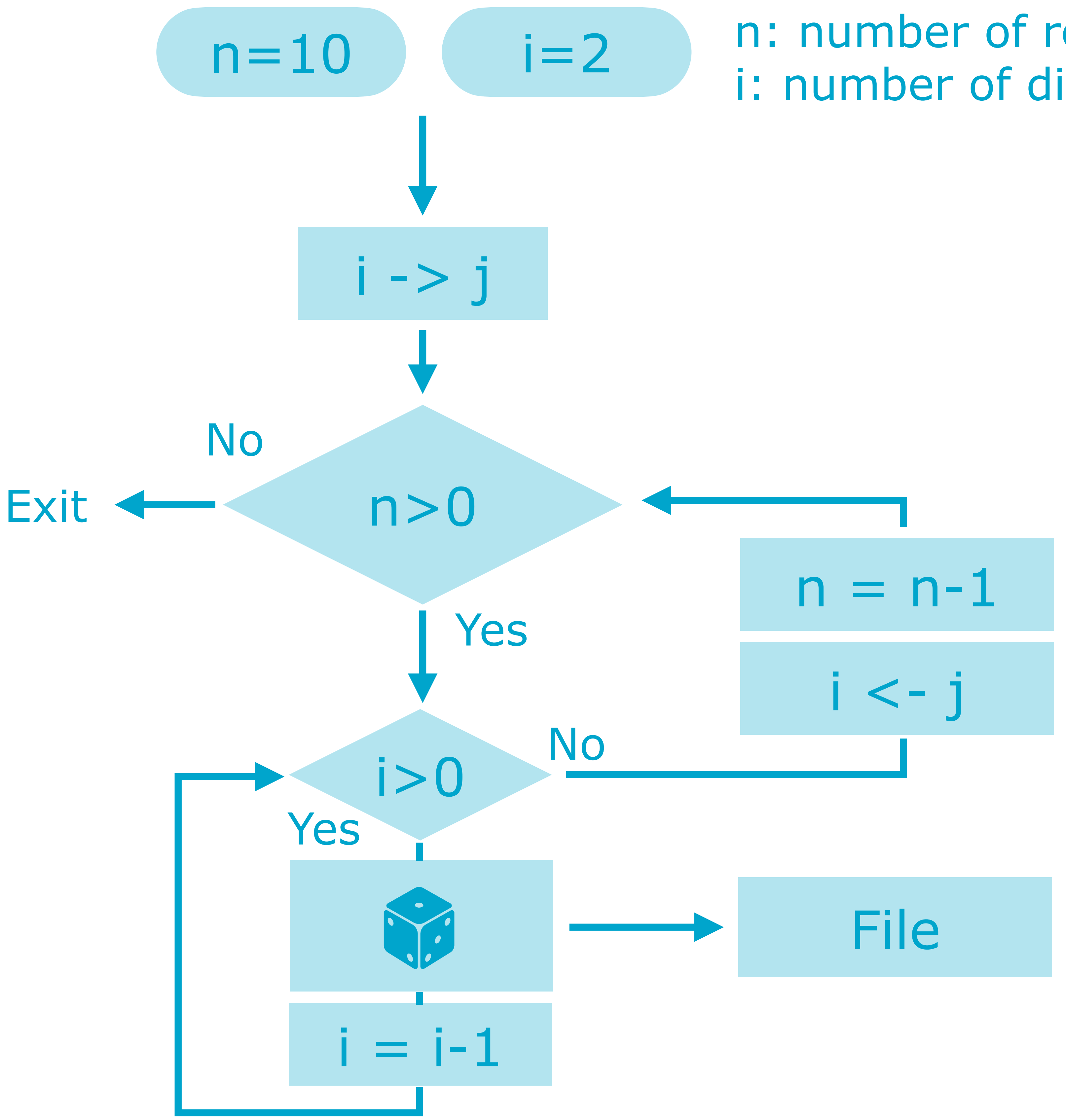

n=1
n>10
Yes
Exit
No
n = n+1

Introduction

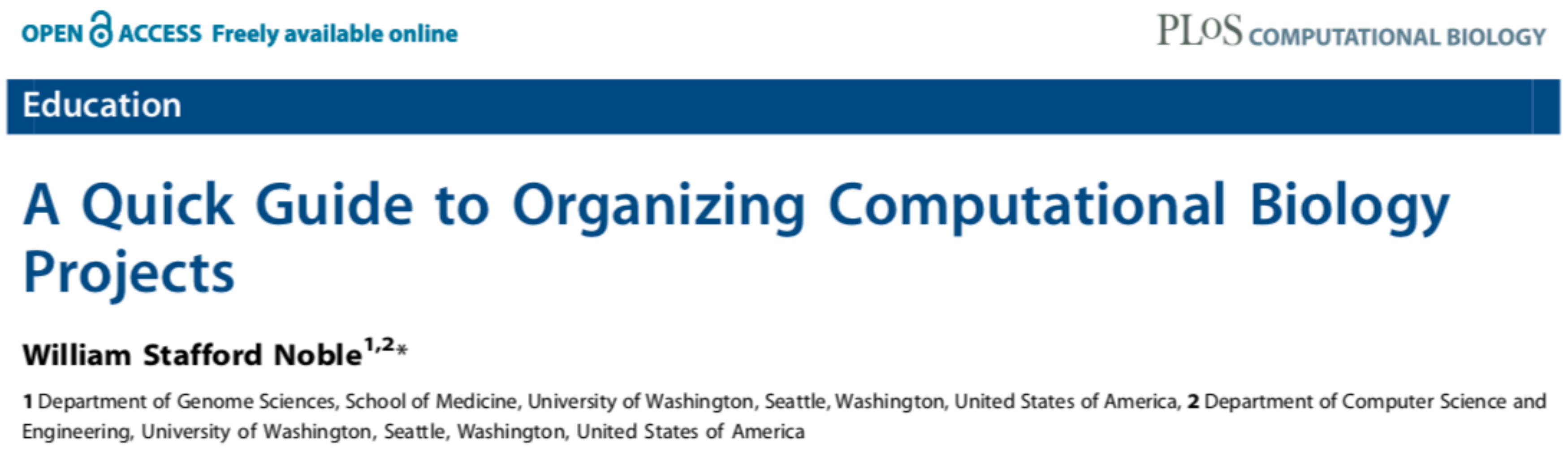


Introduction

n: number of rolls
i: number of dices



Bioinformatics ▷ Introduction


OPEN ACCESS Freely available online

PLoS COMPUTATIONAL BIOLOGY

Education

A Quick Guide to Organizing Computational Biology Projects

William Stafford Noble[1,2]*

1 Department of Genome Sciences, School of Medicine, University of Washington, Seattle, Washington, United States of America, 2 Department of Computer Science and Engineering, University of Washington, Seattle, Washington, United States of America

Most bioinformatics coursework **focuses on algorithms**, with perhaps some **components devoted to learning programming skills and learning how to use existing bioinformatics software**. Unfortunately, for students who are preparing **for a research career, this type of curriculum fails to address many of the day-to-day organisational challenges** associated with performing computational experiments. In practice, the principles behind organising and **documenting computational experiments are often learned on the fly**, and this learning is strongly influenced by personal predilections as well as by chance interactions with collaborators or colleagues.

Specific help (e.g. complition)

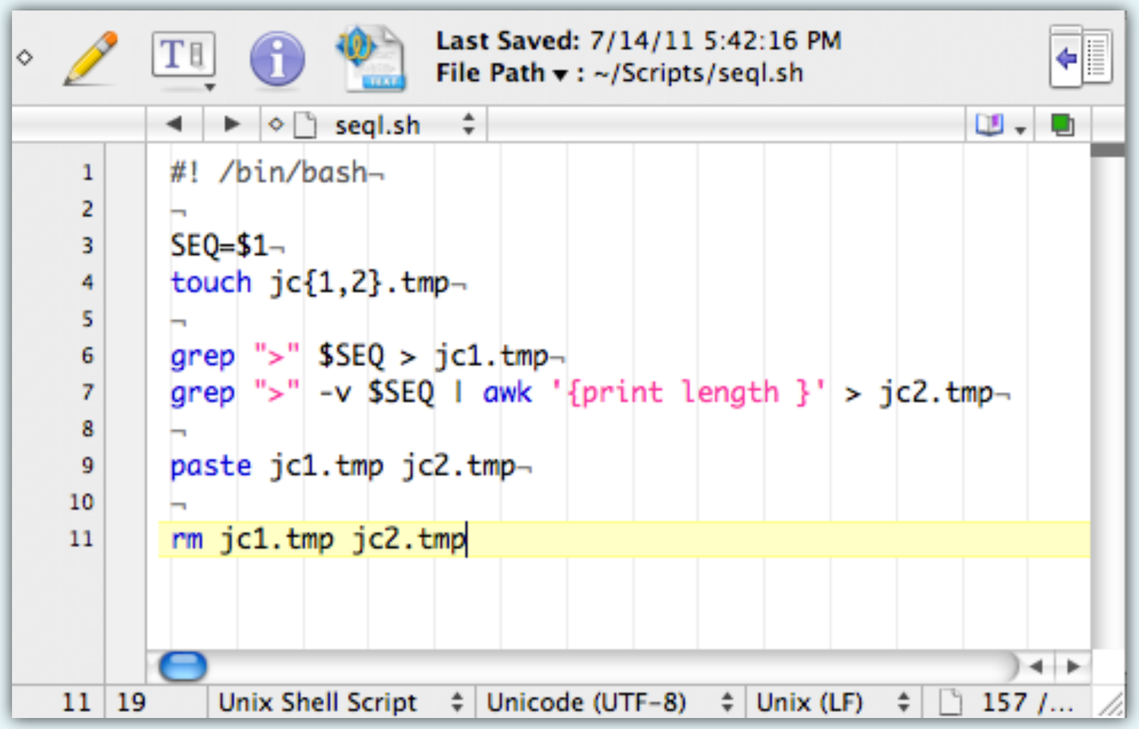
Syntax colouring (e.g. trouble shooting)

Line numbers (e.g. debuging)

Find and replace (e.g. RegEx)

Charater Encoding (e.g. line breaks)

Special Features (e.g. code folding)



Last Saved: 7/14/11 5:42:16 PM
File Path ▾ : ~/Scripts/seql.sh

seql.sh

```
#! /bin/bash

SEQ=$1
touch jc{1,2}.tmp

grep ">" $SEQ > jc1.tmp
grep ">" -v $SEQ | awk '{print length }' > jc2.tmp

paste jc1.tmp jc2.tmp

rm jc1.tmp jc2.tmp
```

11 19 Unix Shell Script Unicode (UTF-8) Unix (LF) 157 /...

>Most Fonts
1234567890
WWWWWWWWWW
IIIIIIIIII

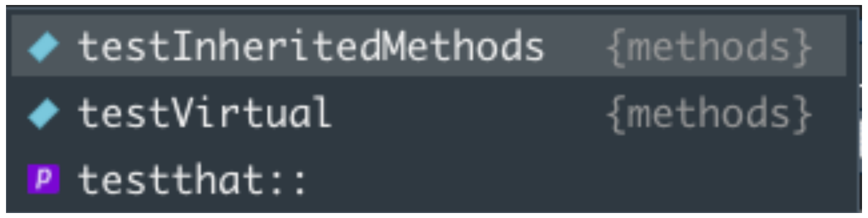
AXAXAXAXAX

```
>Courier
1234567890
WWWWWWWWWW
IIIIIIIIII
----------
AXAXAXAXAX
```

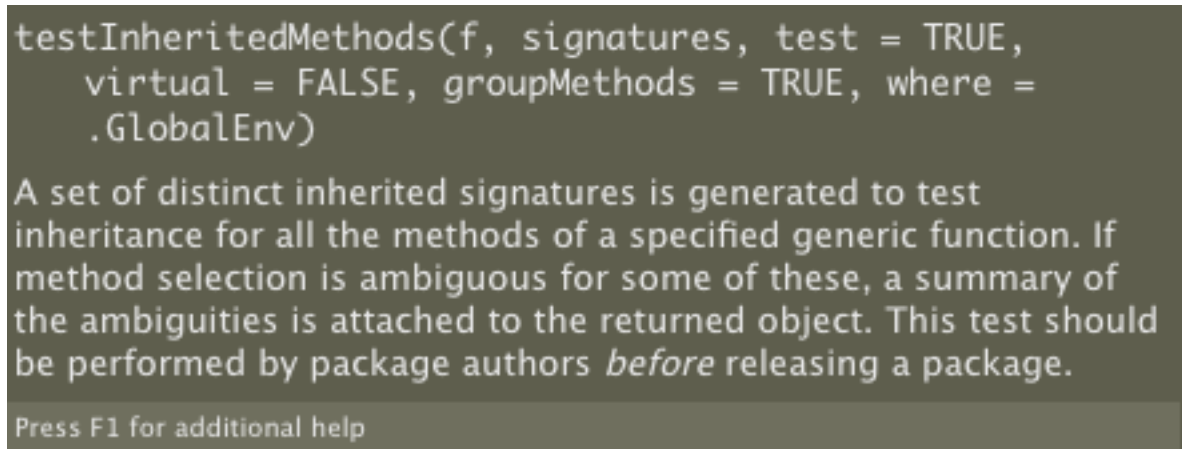
Specific help (e.g. complition)

Example: RStudio

```
> test
```



```
testInheritedMethods   {methods}
testVirtual            {methods}
testthat::
```



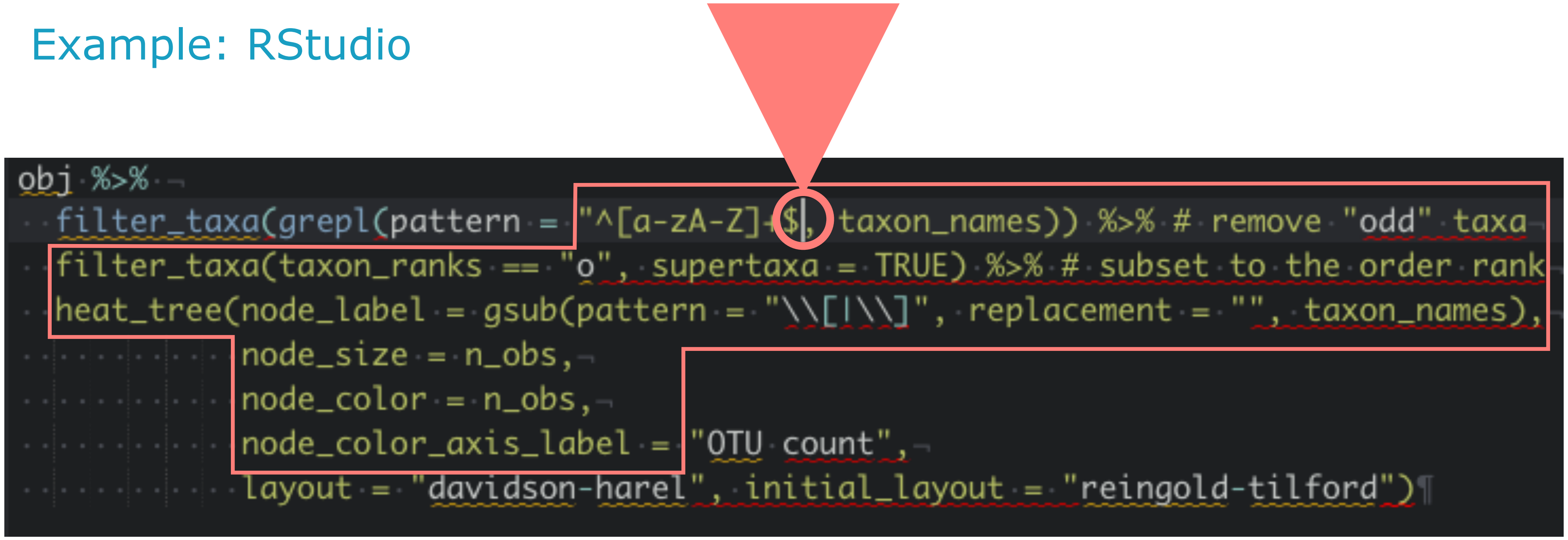
```
testInheritedMethods(f, signatures, test = TRUE,
    virtual = FALSE, groupMethods = TRUE, where =
    .GlobalEnv)
```

A set of distinct inherited signatures is generated to test inheritance for all the methods of a specified generic function. If method selection is ambiguous for some of these, a summary of the ambiguities is attached to the returned object. This test should be performed by package authors *before* releasing a package.

Press F1 for additional help

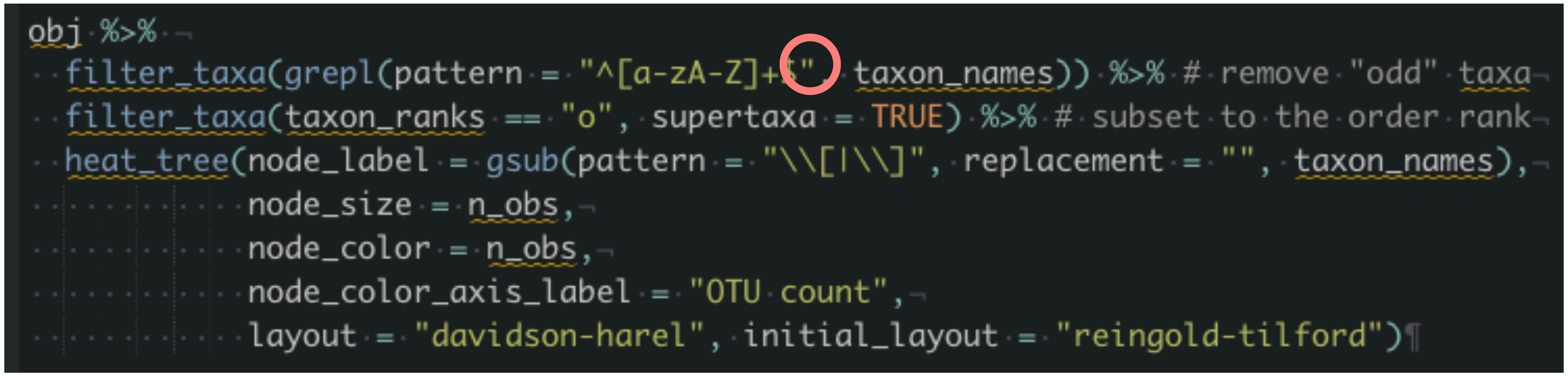
Syntax colouring (e.g. trouble shooting)

Example: RStudio

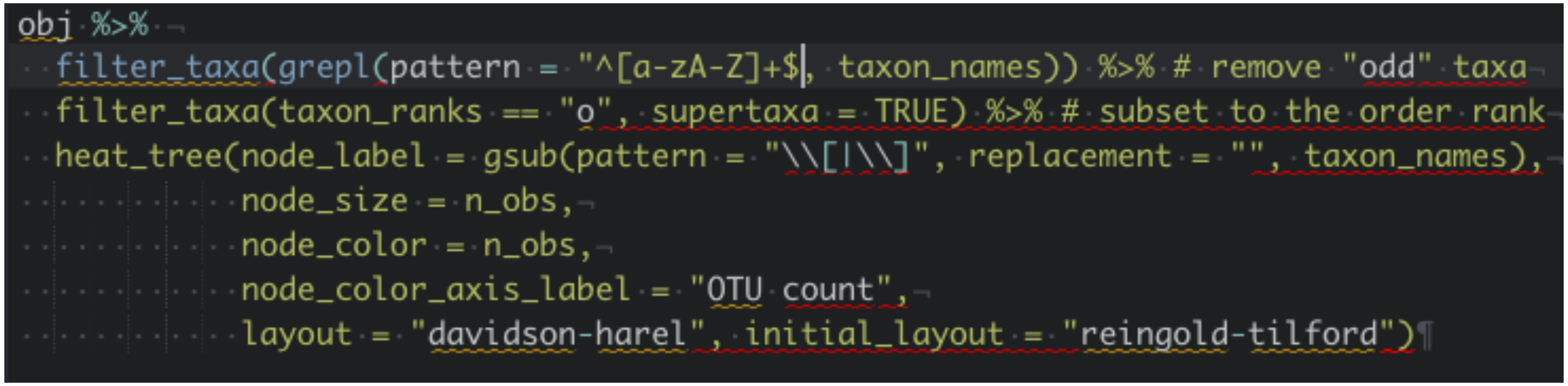


```
obj %>%
  filter_taxa(grepl(pattern = "^[a-zA-Z]+$, taxon_names)) %>% # remove "odd" taxa
  filter_taxa(taxon_ranks == "o", supertaxa = TRUE) %>% # subset to the order rank
  heat_tree(node_label = gsub(pattern = "\\[|\\]", replacement = "", taxon_names),
            node_size = n_obs,
            node_color = n_obs,
            node_color_axis_label = "OTU count",
            layout = "davidson-harel", initial_layout = "reingold-tilford")
```

Syntax colouring (e.g. trouble shooting)



```
obj %>%
  filter_taxa(grepl(pattern = "^[a-zA-Z]+$", taxon_names)) %>% # remove "odd" taxa
  filter_taxa(taxon_ranks == "o", supertaxa = TRUE) %>% # subset to the order rank
  heat_tree(node_label = gsub(pattern = "\\[|\\]", replacement = "", taxon_names),
            node_size = n_obs,
            node_color = n_obs,
            node_color_axis_label = "OTU count",
            layout = "davidson-harel", initial_layout = "reingold-tilford")
```

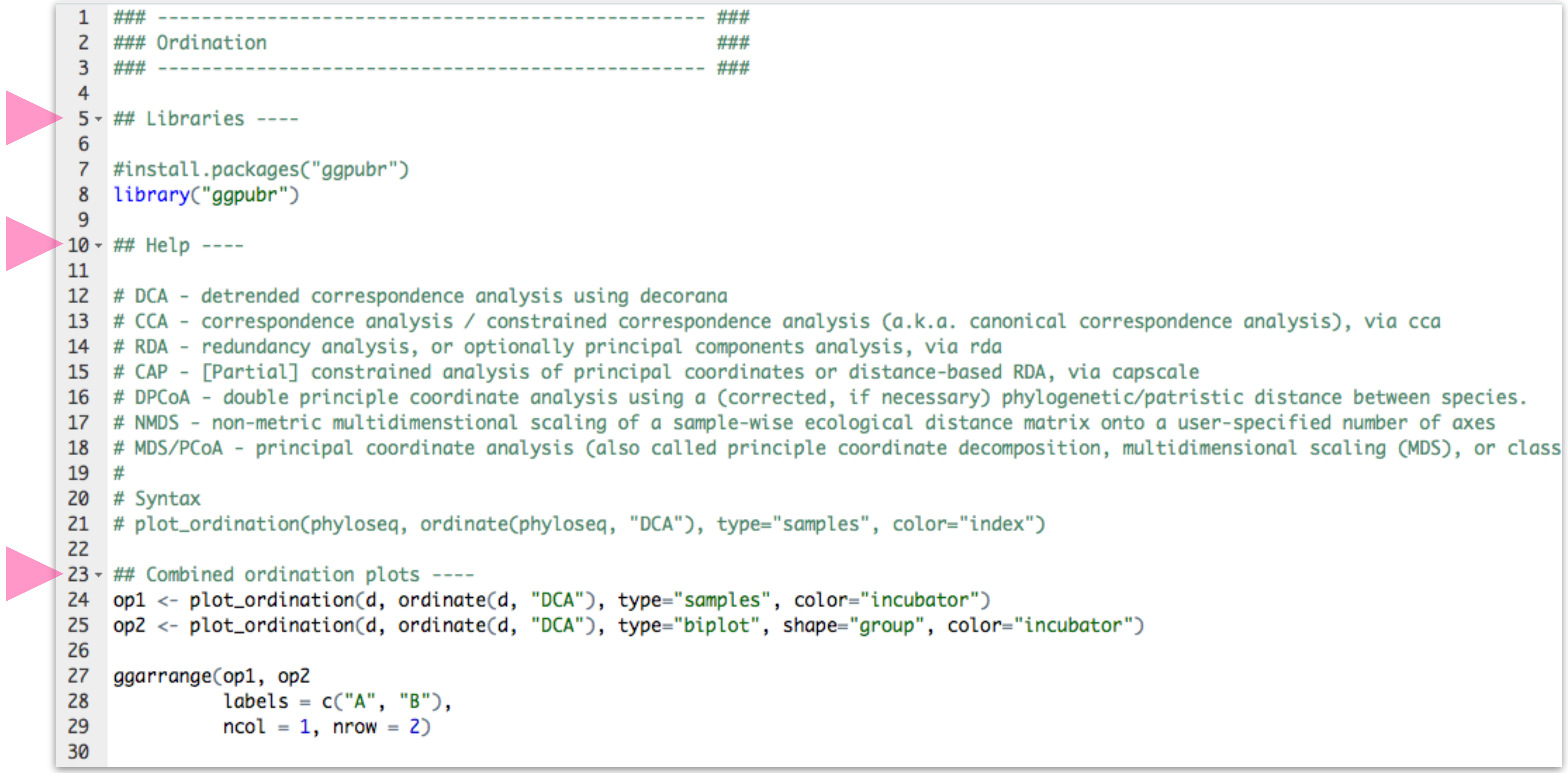


```
obj %>%
  filter_taxa(grepl(pattern = "^[a-zA-Z]+$, taxon_names)) %>% # remove "odd" taxa
  filter_taxa(taxon_ranks == "o", supertaxa = TRUE) %>% # subset to the order rank
  heat_tree(node_label = gsub(pattern = "\\[|\\]", replacement = "", taxon_names),
            node_size = n_obs,
            node_color = n_obs,
            node_color_axis_label = "OTU count",
            layout = "davidson-harel", initial_layout = "reingold-tilford")
```

Code Folding - Code folding allows you to easily **show and hide blocks of code** to make it easier to navigate your source file and focus on the coding task at hand.

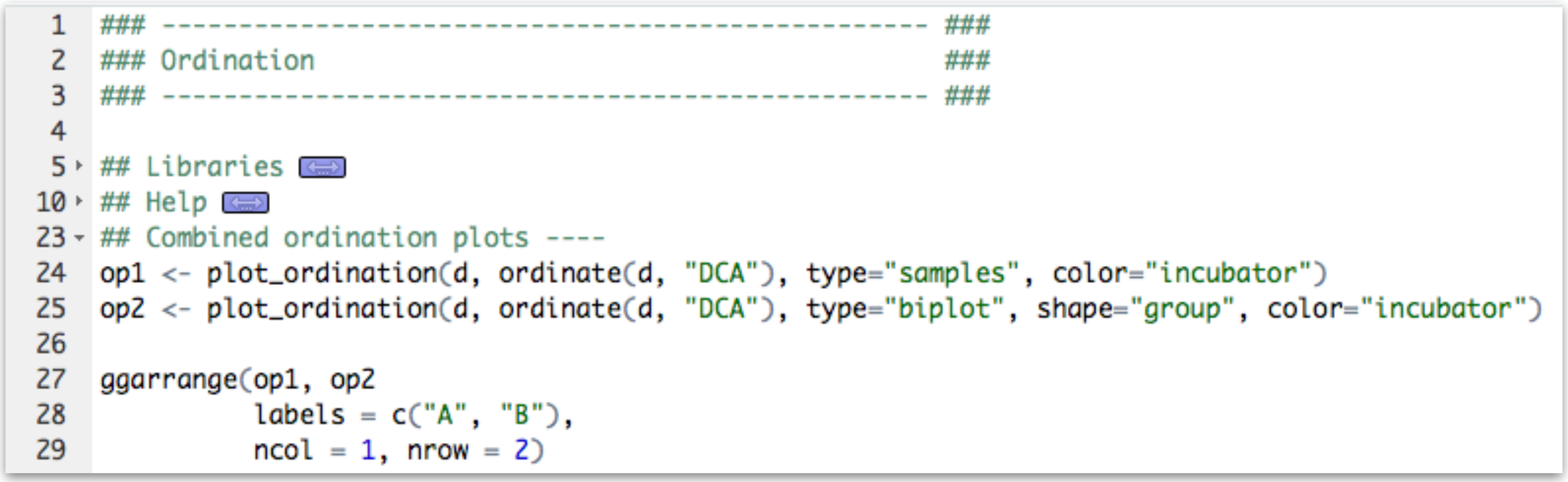
Example RStudio: To insert a new code section you can use the **Code > Insert Section** command. Alternatively, any comment line which includes at least four trailing dashes (-), equal signs (=), or pound signs (#) automatically creates a code section.

Code Folding in RStudio



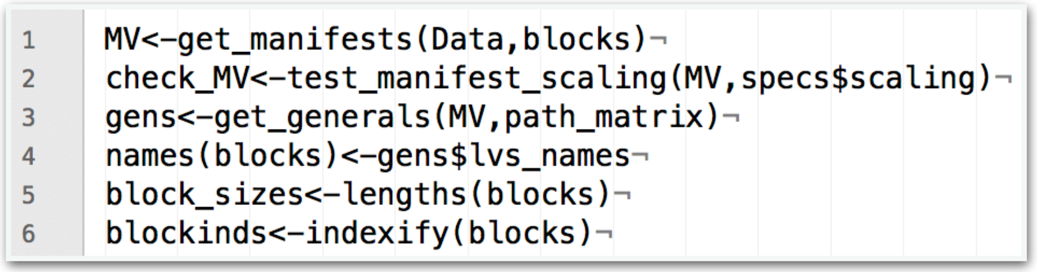
```r
### -------------------------------------------------- ###
### Ordination                                         ###
### -------------------------------------------------- ###

## Libraries ----

#install.packages("ggpubr")
library("ggpubr")

## Help ----

# DCA - detrended correspondence analysis using decorana
# CCA - correspondence analysis / constrained correspondence analysis (a.k.a. canonical correspondence analysis), via cca
# RDA - redundancy analysis, or optionally principal components analysis, via rda
# CAP - [Partial] constrained analysis of principal coordinates or distance-based RDA, via capscale
# DPCoA - double principle coordinate analysis using a (corrected, if necessary) phylogenetic/patristic distance between species.
# NMDS - non-metric multidimensional scaling of a sample-wise ecological distance matrix onto a user-specified number of axes
# MDS/PCoA - principal coordinate analysis (also called principle coordinate decomposition, multidimensional scaling (MDS), or class
#
# Syntax
# plot_ordination(phyloseq, ordinate(phyloseq, "DCA"), type="samples", color="index")

## Combined ordination plots ----
op1 <- plot_ordination(d, ordinate(d, "DCA"), type="samples", color="incubator")
op2 <- plot_ordination(d, ordinate(d, "DCA"), type="biplot", shape="group", color="incubator")

ggarrange(op1, op2
          labels = c("A", "B"),
          ncol = 1, nrow = 2)

```

Code Folding in RStudio



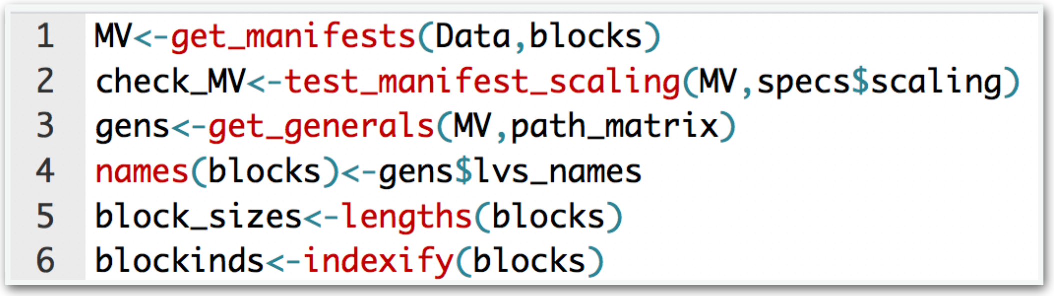
```r
### ------------------------------------------------- ###
### Ordination                                         ###
### ------------------------------------------------- ###

## Libraries
## Help
## Combined ordination plots ----
op1 <- plot_ordination(d, ordinate(d, "DCA"), type="samples", color="incubator")
op2 <- plot_ordination(d, ordinate(d, "DCA"), type="biplot", shape="group", color="incubator")

ggarrange(op1, op2
          labels = c("A", "B"),
          ncol = 1, nrow = 2)
```

Built-in Help: Example for RStudio



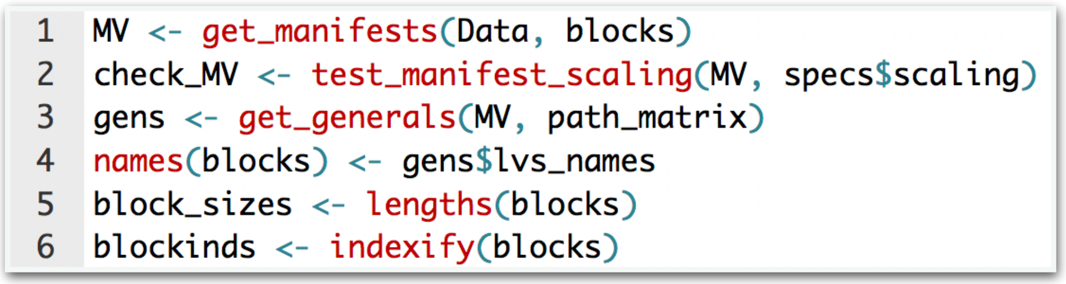
```
MV<-get_manifests(Data,blocks)¬
check_MV<-test_manifest_scaling(MV,specs$scaling)¬
gens<-get_generals(MV,path_matrix)¬
names(blocks)<-gens$lvs_names¬
block_sizes<-lengths(blocks)¬
blockinds<-indexify(blocks)¬
```

✓ Show syntax highlighting in console input



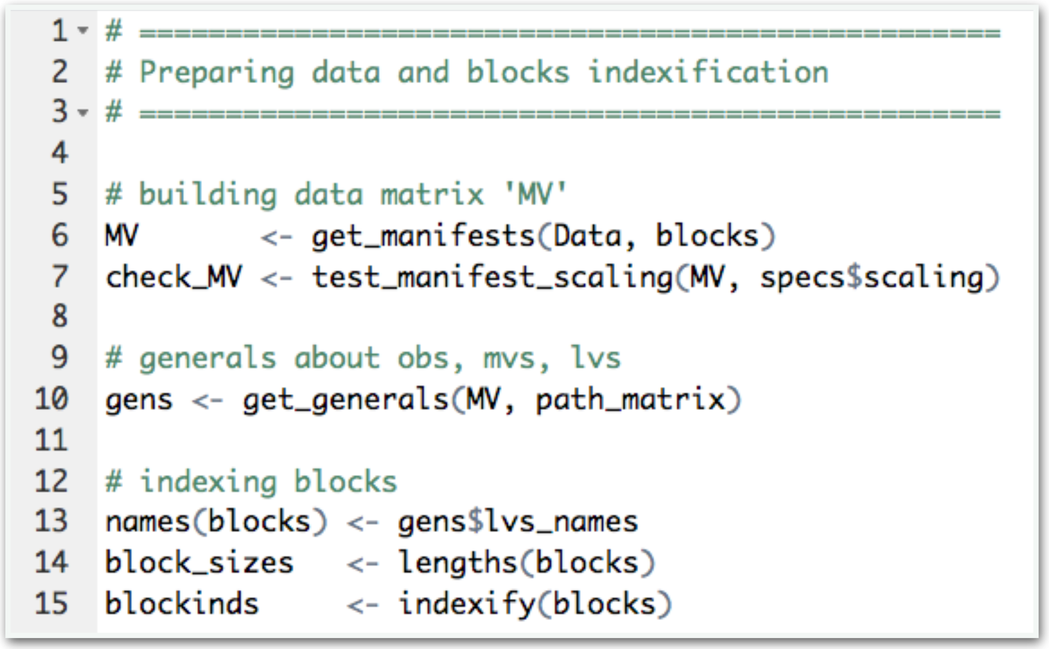
```
MV<-get_manifests(Data,blocks)
check_MV<-test_manifest_scaling(MV,specs$scaling)
gens<-get_generals(MV,path_matrix)
names(blocks)<-gens$lvs_names
block_sizes<-lengths(blocks)
blockinds<-indexify(blocks)
```

Reformat Code ⇧⌘A



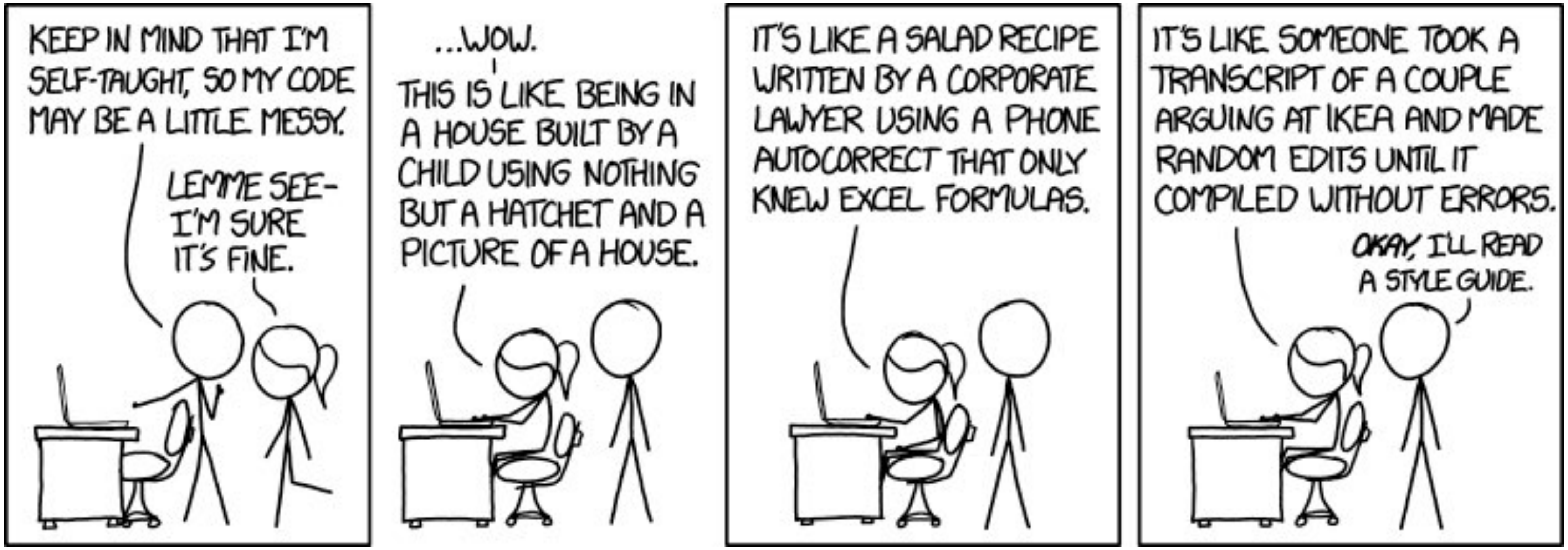
```
MV <- get_manifests(Data, blocks)
check_MV <- test_manifest_scaling(MV, specs$scaling)
gens <- get_generals(MV, path_matrix)
names(blocks) <- gens$lvs_names
block_sizes <- lengths(blocks)
blockinds <- indexify(blocks)
```

#



```
# ==================================================
# Preparing data and blocks indexification
# ==================================================

# building data matrix 'MV'
MV        <- get_manifests(Data, blocks)
check_MV <- test_manifest_scaling(MV, specs$scaling)

# generals about obs, mvs, lvs
gens <- get_generals(MV, path_matrix)

# indexing blocks
names(blocks) <- gens$lvs_names
block_sizes   <- lengths(blocks)
blockinds     <- indexify(blocks)
```

Code Style


KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

What is a **coding style**?

Language is a tool that allows people to interact and communicate with each other. The clearer we express ourselves, the better the idea is transferred from our minds to others. The same applies to programming languages: **concise, clear and consistent code is easier to read, more fun to work with and ultimately easier to reproduce.**

Bioinformatics ▷ Introduction

ggplot2isanrpackageforproducingstatisticalordatagraphicsbutitisunli
kemostothergraphicspackagesbecauseithasadeepunderlyinggramm
arthisgrammarbasedonthegrammarofgraphicswilkinson2005iscompo
sedofasetofindependentcomponentsthatcanbecomposedinmanydiff
erentwaysthismakesggplot2verypowerfulbecauseyouarenotlimitedto
asetofprespecifiedgraphicsbutyoucancreatenewgraphicsthatarepreci
selytailoredforyourproblemthismaysoundoverwhelmingbutbecauseth
ereisasimplesetofcoreprinciplesandveryfewspecialcasesggplot2isalso
easytolearnalthoughitmaytakealittletimetoforgetyourpreconceptions
fromothergraphicstools

Source: Wickham (2009) ggplot2 - Elegant Graphics for Data Analysis

Introduction

ggplot2 is an r package for producing statistical or data graphics but it is unlike most other graphics packages because it has a deep underlying grammar this grammar based on the grammar of graphics wilkinson 2005 is composed of a set of independent components that can be composed in many different ways this makes ggplot2 very powerful because you are not limited to a set of pre-specified graphics but you can create new graphics that are precisely tailored for your problem this may sound overwhelming but because there is a simple set of core principles and very few special cases ggplot2 is also easy to learn although it may take a little time to forget your preconceptions from other graphics tools

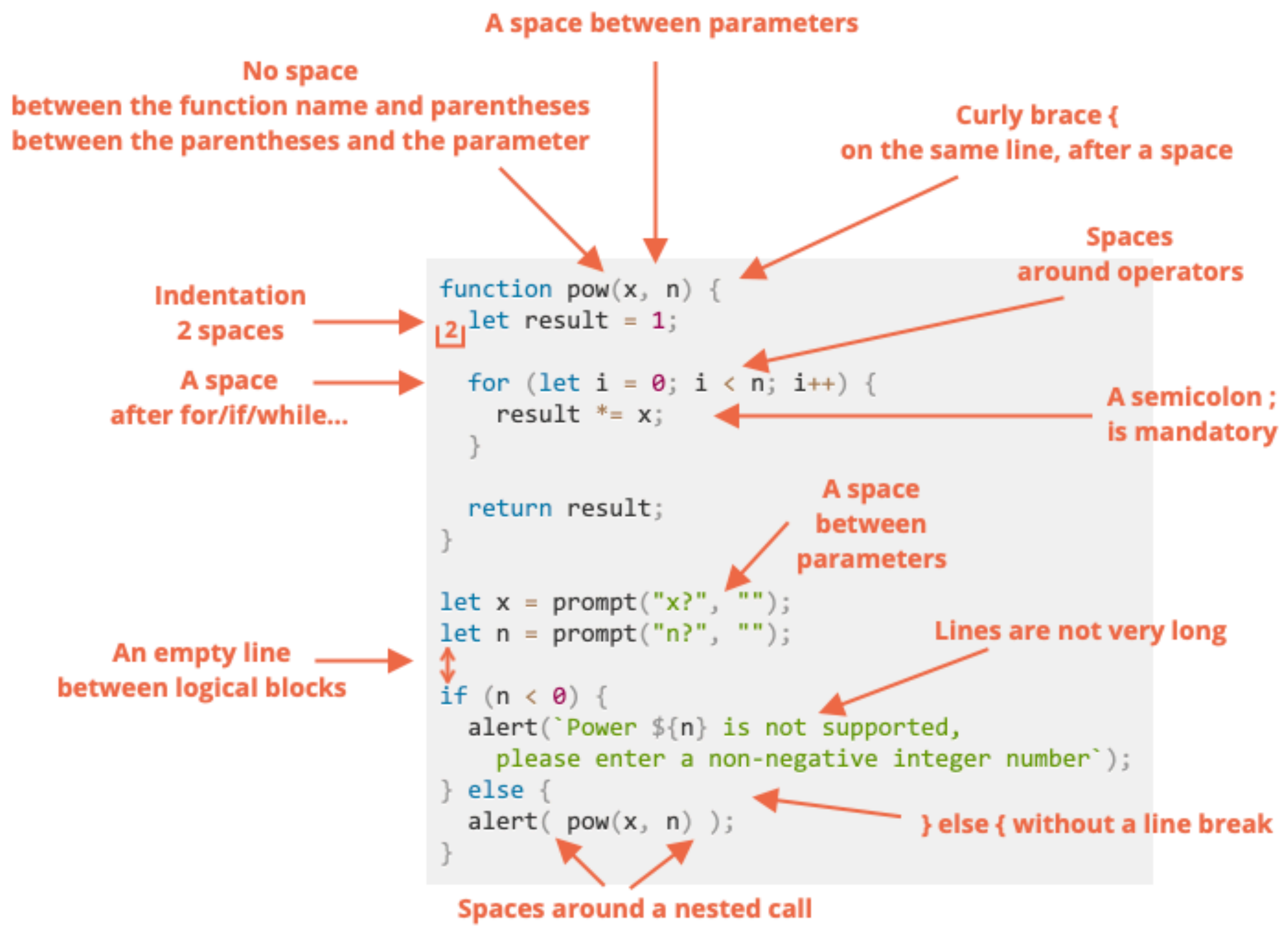
Source: Wickham (2009) ggplot2 - Elegant Graphics for Data Analysis

Bioinformatics ▷ Introduction

ggplot2 is an R package for producing statistical, or data, graphics, but it is unlike most other graphics packages because it has a deep underlying grammar. This grammar, based on the Grammar of Graphics (Wilkinson, 2005), is composed of a set of independent components that can be composed in many different ways. This makes ggplot2 very powerful, because you are not limited to a set of pre-specified graphics, but you can create new graphics that are precisely tailored for your problem. This may sound overwhelming, but because there is a simple set of core principles and very few special cases, ggplot2 is also easy to learn (although it may take a little time to forget your preconceptions from other graphics tools).

Source: Wickham (2009) ggplot2 - Elegant Graphics for Data Analysis

Code Style



```
function pow(x, n) {
  let result = 1;

  for (let i = 0; i < n; i++) {
    result *= x;
  }

  return result;
}

let x = prompt("x?", "");
let n = prompt("n?", "");

if (n < 0) {
  alert(`Power ${n} is not supported,
    please enter a non-negative integer number`);
} else {
  alert( pow(x, n) );
}
```

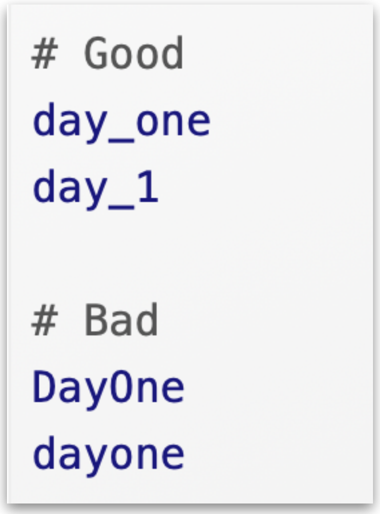
javascript example / source: https://javascript.info/coding-style

The tidyverse Style Guide

Google's R Style Guide

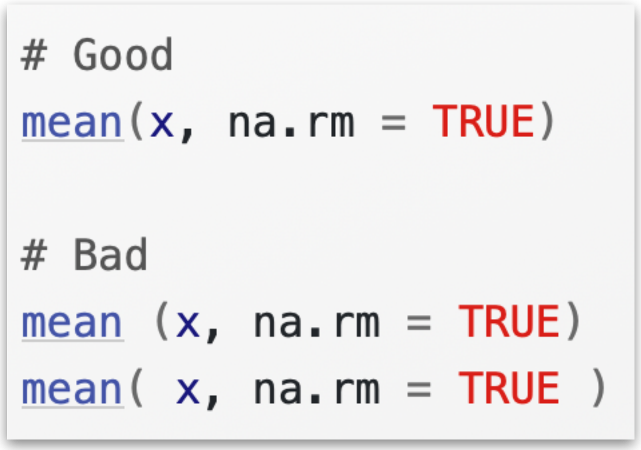
tidyverse style guide examples

Object names



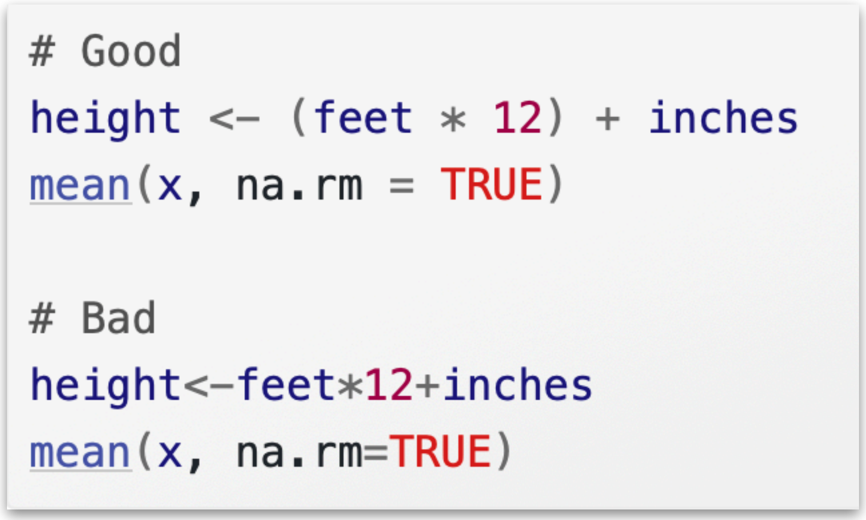
```
# Good
day_one
day_1

# Bad
DayOne
dayone
```

Parentheses



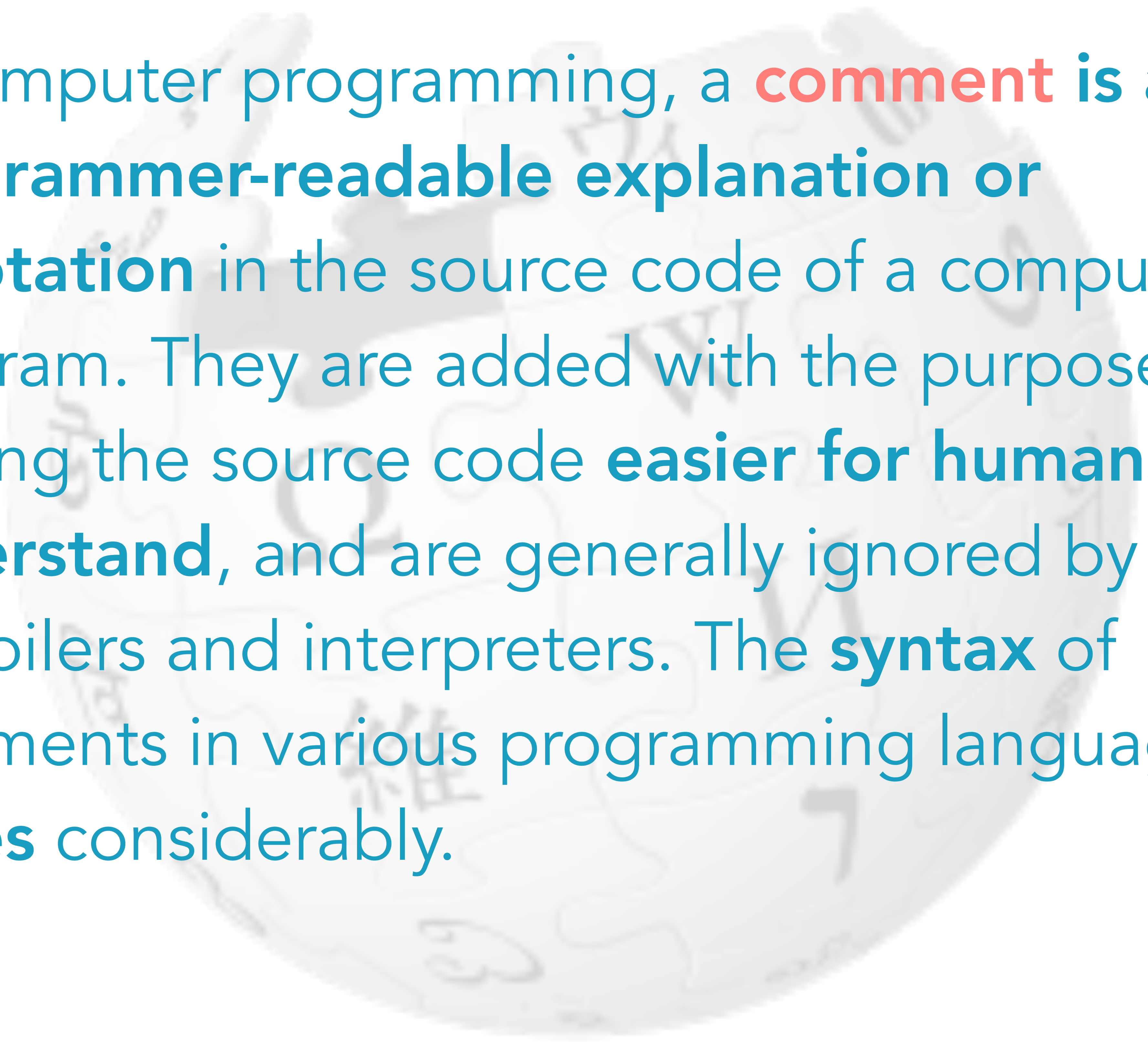
```
# Good
mean(x, na.rm = TRUE)

# Bad
mean (x, na.rm = TRUE)
mean( x, na.rm = TRUE )
```

Infix operators



```
# Good
height <- (feet * 12) + inches
mean(x, na.rm = TRUE)

# Bad
height<-feet*12+inches
mean(x, na.rm=TRUE)
```

Bioinformatics ▷ Introduction



In computer programming, a **comment is** a **programmer-readable explanation or annotation** in the source code of a computer program. They are added with the purpose of making the source code **easier for humans to understand**, and are generally ignored by compilers and interpreters. The **syntax** of comments in various programming languages **varies** considerably.

Python

```
# Comment in Python

""" multiline comment
  .
  .
"""
```

Perl

```
# Comment in Perl

=begin multiline comment
  .
  .
=cut
```

C/C++

```
// single line comment

/*Comment starts
  .
  .
Comment ends*/
```

Similar to Java

html

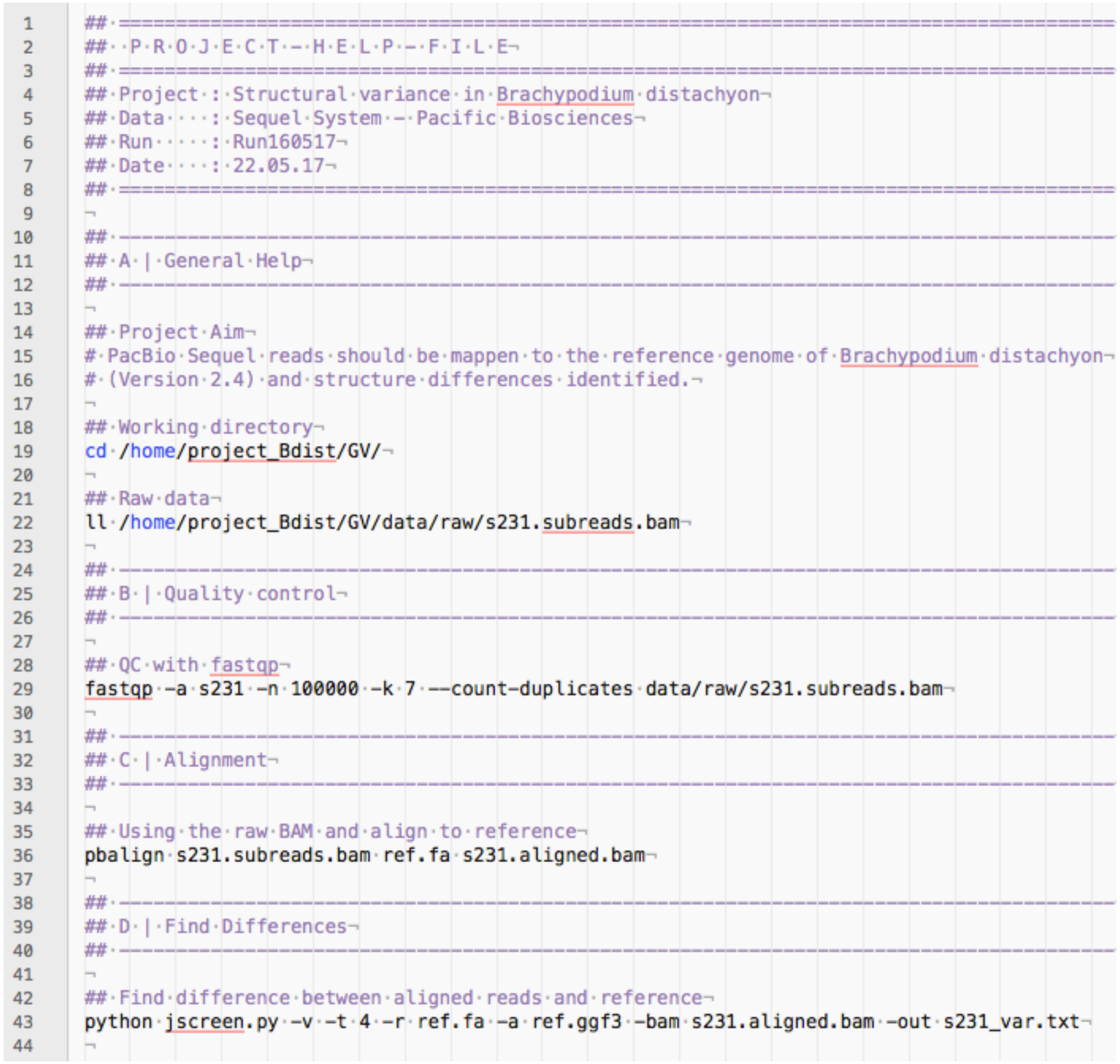
```
<!--single html comment-->

<!--multiline comment
  .
  .
//-->
```

Type of Comments

- Header / Titles
- Documentation
- Clarification

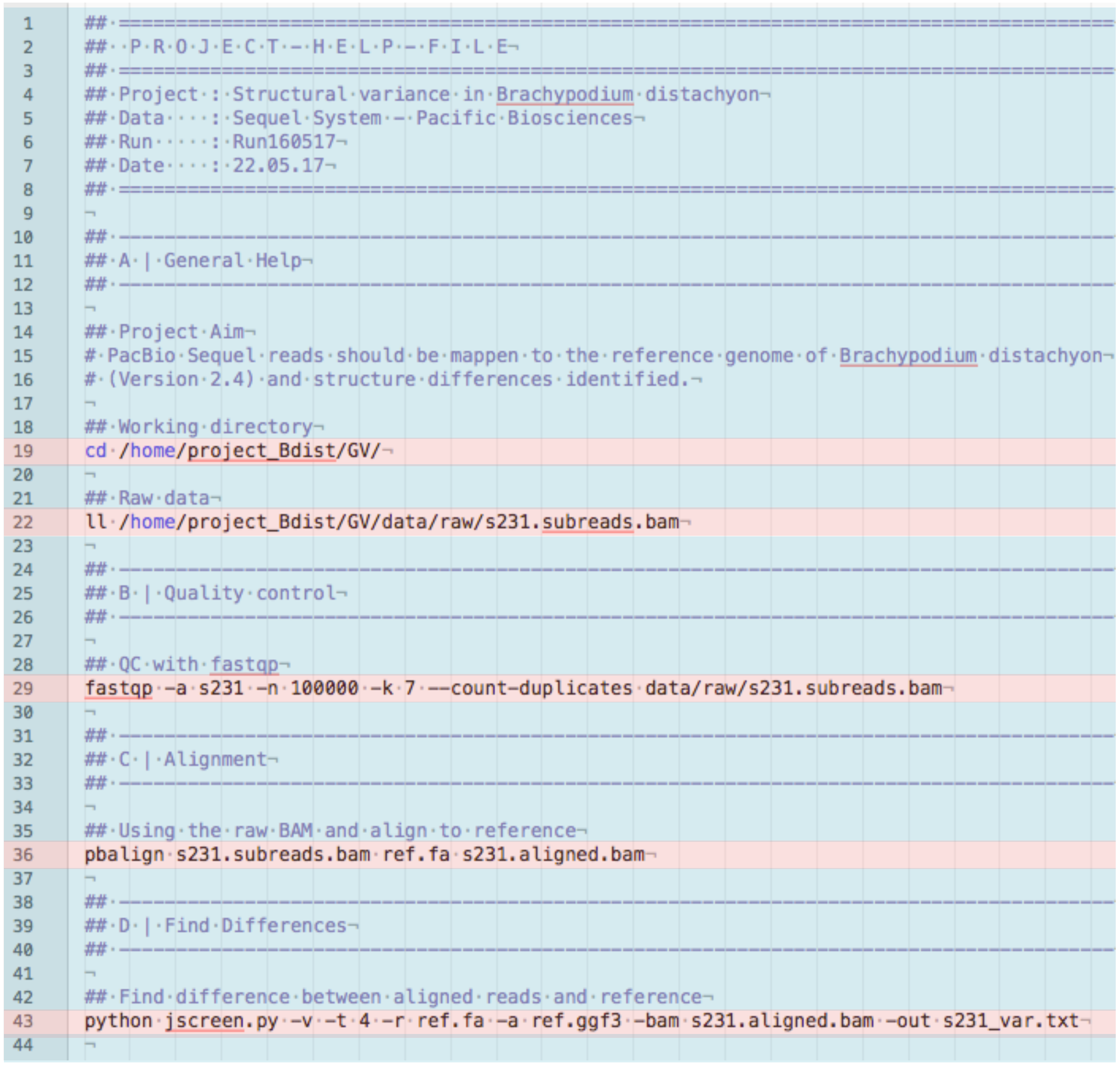
Headers are important structural elements.
Titles help to give the code structure.
Documentation comments are needed to explain your code.
Clarification comments are intended for anyone (including your future self) who may need to maintain, refactor, or extend your code.

Bioinformatics ▷ Introduction



```
## ==================================================================================
##  P R O J E C T - H E L P - F I L E
## ==================================================================================
## Project : Structural variance in Brachypodium distachyon
## Data    : Sequel System - Pacific Biosciences
## Run     : Run160517
## Date    : 22.05.17
## ==================================================================================

## ----------------------------------------------------------------------------------
## A | General Help
## ----------------------------------------------------------------------------------

## Project Aim
# PacBio Sequel reads should be mappen to the reference genome of Brachypodium distachyon
# (Version 2.4) and structure differences identified.

## Working directory
cd /home/project_Bdist/GV/

## Raw data
ll /home/project_Bdist/GV/data/raw/s231.subreads.bam

## ----------------------------------------------------------------------------------
## B | Quality control
## ----------------------------------------------------------------------------------

## QC with fastqp
fastqp -a s231 -n 100000 -k 7 --count-duplicates data/raw/s231.subreads.bam

## ----------------------------------------------------------------------------------
## C | Alignment
## ----------------------------------------------------------------------------------

## Using the raw BAM and align to reference
pbalign s231.subreads.bam ref.fa s231.aligned.bam

## ----------------------------------------------------------------------------------
## D | Find Differences
## ----------------------------------------------------------------------------------

## Find difference between aligned reads and reference
python jscreen.py -v -t 4 -r ref.fa -a ref.ggf3 -bam s231.aligned.bam -out s231_var.txt

```

Bioinformatics ▷ Introduction

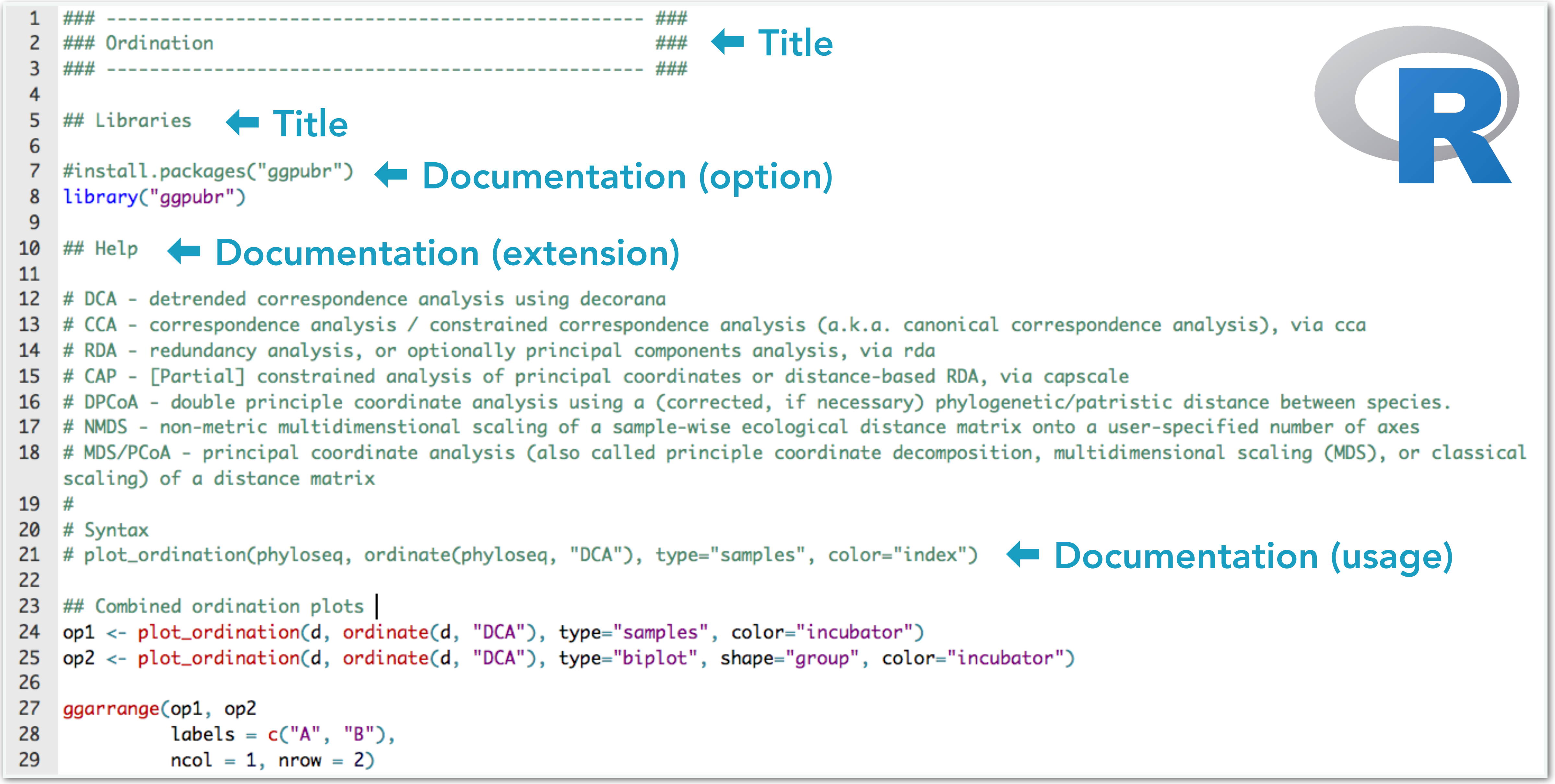


```
## ==========================================================================
##  P R O J E C T — H E L P — F I L E
## ==========================================================================
## Project : Structural variance in Brachypodium distachyon
## Data    : Sequel System - Pacific Biosciences
## Run     : Run160517
## Date    : 22.05.17
## ==========================================================================

## --------------------------------------------------------------------------
## A | General Help
## --------------------------------------------------------------------------

## Project Aim
# PacBio Sequel reads should be mappen to the reference genome of Brachypodium distachyon
# (Version 2.4) and structure differences identified.

## Working directory
cd /home/project_Bdist/GV/

## Raw data
ll /home/project_Bdist/GV/data/raw/s231.subreads.bam

## --------------------------------------------------------------------------
## B | Quality control
## --------------------------------------------------------------------------

## QC with fastqp
fastqp -a s231 -n 100000 -k 7 --count-duplicates data/raw/s231.subreads.bam

## --------------------------------------------------------------------------
## C | Alignment
## --------------------------------------------------------------------------

## Using the raw BAM and align to reference
pbalign s231.subreads.bam ref.fa s231.aligned.bam

## --------------------------------------------------------------------------
## D | Find Differences
## --------------------------------------------------------------------------

## Find difference between aligned reads and reference
python jscreen.py -v -t 4 -r ref.fa -a ref.gff3 -bam s231.aligned.bam -out s231_var.txt

```

Comments (#)
Spaces ➜ Structure

Code

Introduction



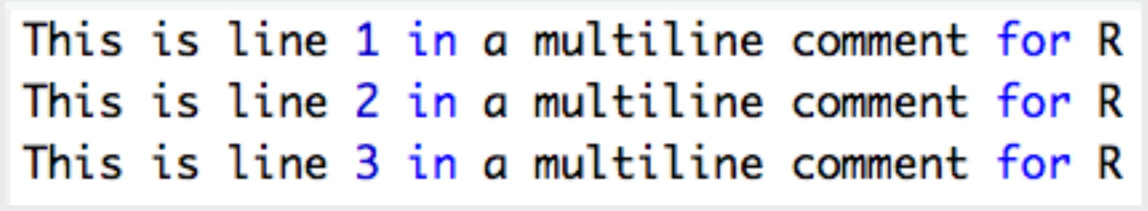
```r
### ------------------------------------------------ ###
### Ordination                                       ###
### ------------------------------------------------ ###

## Libraries

#install.packages("ggpubr")
library("ggpubr")

## Help

# DCA - detrended correspondence analysis using decorana
# CCA - correspondence analysis / constrained correspondence analysis (a.k.a. canonical correspondence analysis), via cca
# RDA - redundancy analysis, or optionally principal components analysis, via rda
# CAP - [Partial] constrained analysis of principal coordinates or distance-based RDA, via capscale
# DPCoA - double principle coordinate analysis using a (corrected, if necessary) phylogenetic/patristic distance between species.
# NMDS - non-metric multidimenstional scaling of a sample-wise ecological distance matrix onto a user-specified number of axes
# MDS/PCoA - principal coordinate analysis (also called principle coordinate decomposition, multidimensional scaling (MDS), or classical scaling) of a distance matrix
#
# Syntax
# plot_ordination(phyloseq, ordinate(phyloseq, "DCA"), type="samples", color="index")

## Combined ordination plots
op1 <- plot_ordination(d, ordinate(d, "DCA"), type="samples", color="incubator")
op2 <- plot_ordination(d, ordinate(d, "DCA"), type="biplot", shape="group", color="incubator")

ggarrange(op1, op2
          labels = c("A", "B"),
          ncol = 1, nrow = 2)
```

Bioinformatics ▷ Introduction

R

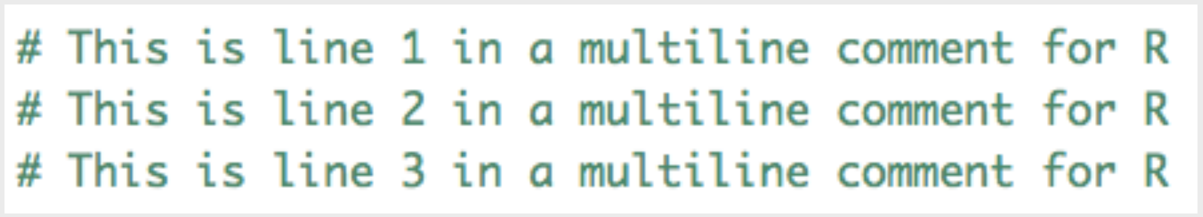
```
# Single line comment in R

# multiline comment
#   .
#   .
#   .
```

RStudio



```
This is line 1 in a multiline comment for R
This is line 2 in a multiline comment for R
This is line 3 in a multiline comment for R
```

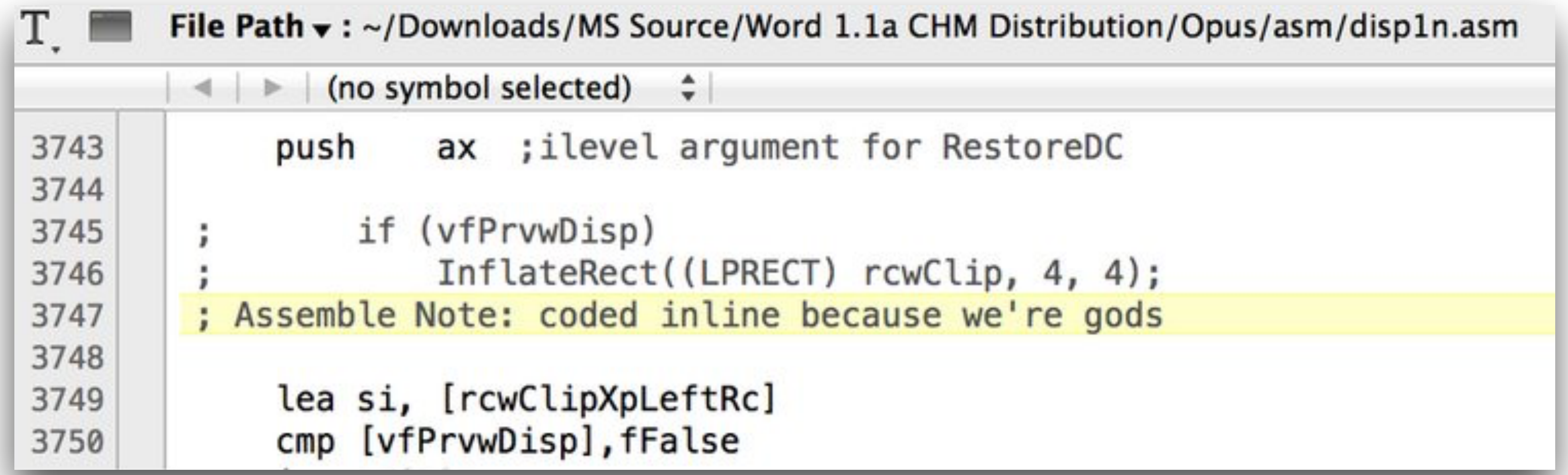
shift + cmd + C



```
# This is line 1 in a multiline comment for R
# This is line 2 in a multiline comment for R
# This is line 3 in a multiline comment for R
```

Profanity in Source Code

Introduction


File Path : ~/Downloads/MS Source/Word 1.1a CHM Distribution/Opus/asm/disp1n.asm

(no symbol selected)

```
    push    ax  ;ilevel argument for RestoreDC

;       if (vfPrvwDisp)
;           InflateRect((LPRECT) rcwClip, 4, 4);
; Assemble Note: coded inline because we're gods

    lea si, [rcwClipXpLeftRc]
    cmp [vfPrvwDisp],fFalse
```

Why not, if you must ….

```
# F@!+& piece of R code - it drives me avocados
```

What about a more constructive way …

```
#! I can't figure this out.
#! I need to extract only the records where
#! t < 12 but not > 18.
```

Good code is self-documenting.

Code tells you how, comments tell you why.

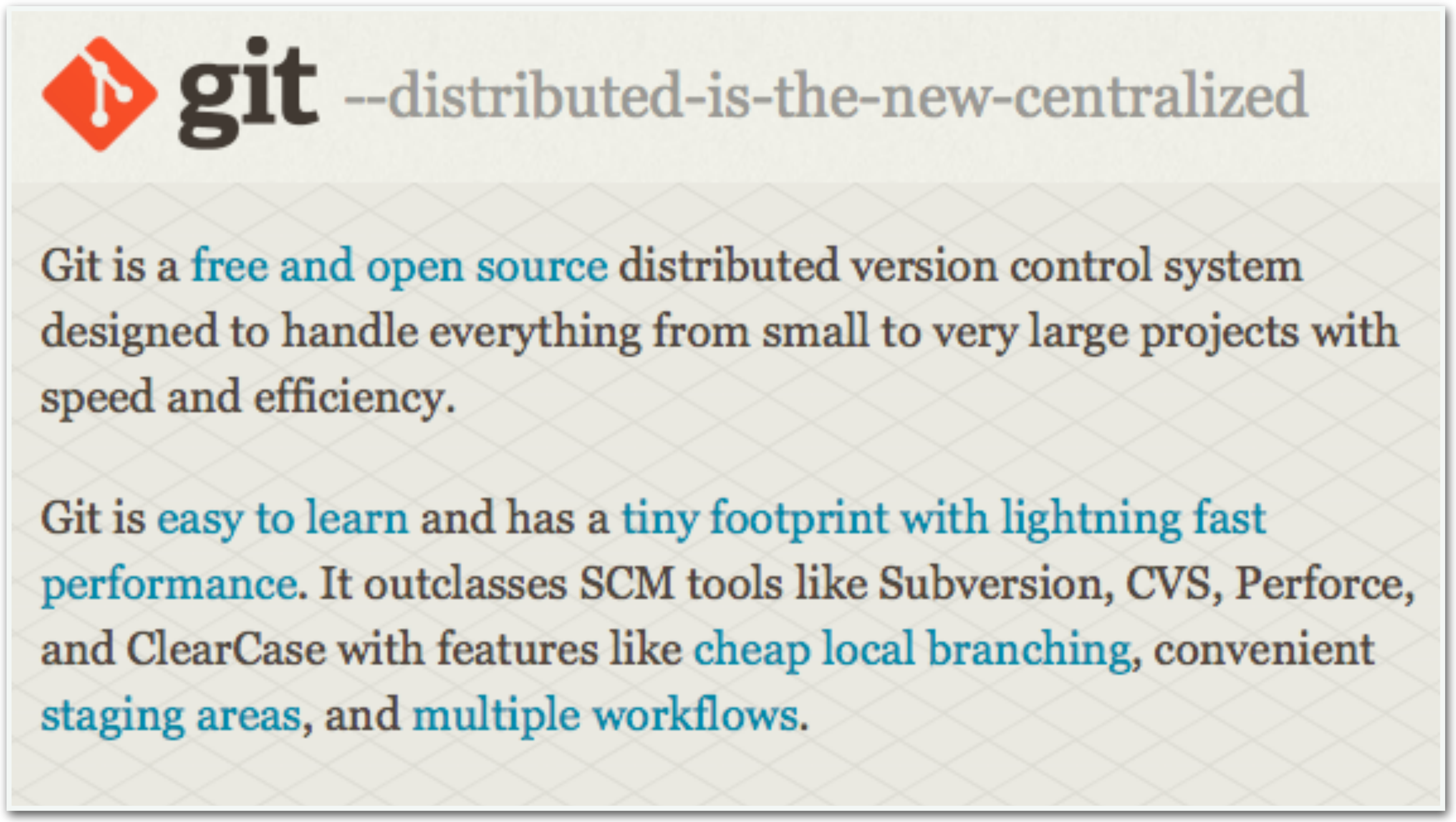
Good developers write good code;
great ones also write good comments.

Version Control

Bioinformatics ▷ Introduction

```
get-cpu-and-memory-usage_v190812_sz.sh.archive
get-cpu-and-memory-usage_v190823_nz.sh.archive
get-cpu-and-memory-usage_v190903_jcw.sh.archive
get-cpu-and-memory-usage_v200303_jcw.sh.archive
get-cpu-and-memory-usage_v200530_nz.sh.archive
get-cpu-and-memory-usage_v210117_jcw.sh
```

Bioinformatics ▷ Introduction


git --distributed-is-the-new-centralized

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Bioinformatics ▷ Introduction

GitLab

GitLab is a web-based Git repository manager with wiki and issue tracking features, using an open source license, developed by GitLab Inc.

A Quick Recap

1 **Code Style**
Learn and use a common style.

2 **#**
Comment your code and do it generously.

3 **Code Editor**
Make use of the different features like sintax highlighting and code folding.

4 **Version Control**
For bigger or collaborative projects use version control and/or cloud solutions.