Today

- Computing languages for science
- Productive workspaces
- Integrated Development Environments
- Algorithms
- Structured programming

Languages

- Lower-level* programming languages
  - Interact directly with the computer's memory
  - Compiled into an executable program
  - C, C++, Fortran
  - Fast to run, slower to write the code

* Technically "low-level" = machine code & assembly language, whereas C etc are "static high-level" languages. We are making a relative comparison.

Languages

- Higher-level programming languages
  - aka "dynamic" or "scripting" languages
  - Run within a parent program that interprets the code
  - Parent program manages the computer's memory
  - Programs are "scripts"
  - R, Python, Matlab
  - Run slower (sometimes only slightly); faster to write the code

R

- Implementation of “S” (Bell Labs)
- Ihaka and Gentleman (U Auckland, New Zealand 1995), now many core developers
- Open source, free software

R

- Dominates statistical research
- Dominates ecology
- Important in biology generally

R components

- 1) Base: programming language, data handling, calculations, data analysis, graphics
- 2) Contributed packages: 22545 CRAN + many others (e.g. on Github)
- 3) Tidyverse: set of R packages that implement a dialect (domain specific language) of R

Posit (a company)

- Formerly RStudio (the company)
- RStudio IDE
- Tidyverse packages
- RMarkdown (and the new Quarto)

Python

- General programming language
- Guido van Rossum, Dutch academic
- v1 1994, v3 2008
- Open source, free software

Python

- ... for science
- Numpy & scipi libraries
  - dev by grad students! 1995-2005
  - support for numerical calculations
  - arrays, matrices, numerical algorithms
- Matplotlib
  - plotting

Python

- Dominates certain areas of science
  - e.g. remote sensing, microbial ecology, geophysical sciences, DNA sequences
  - machine learning
- Important in biology generally

Similar languages

- Matlab
  - commercial
  - increasingly less relevant
- Julia
  - perhaps superior
  - but not widely used yet

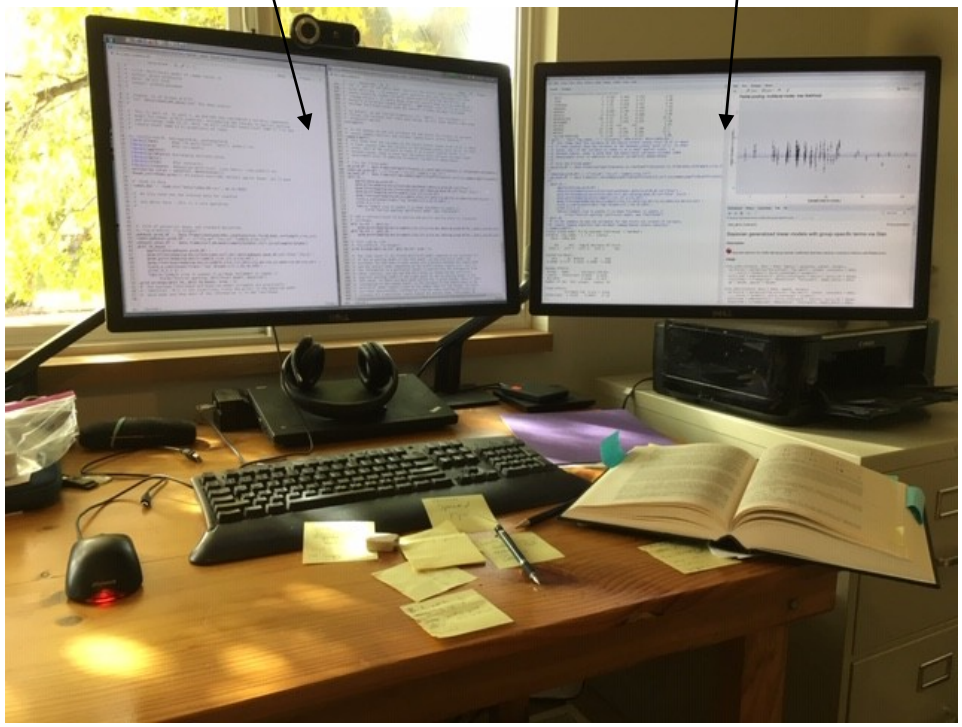
Underlying R & Python

- Much C, C++ and Fortran source code
- Many libraries have underlying C/C++ code
- e.g. in machine learning R & Python are essentially interfaces to C++ libraries

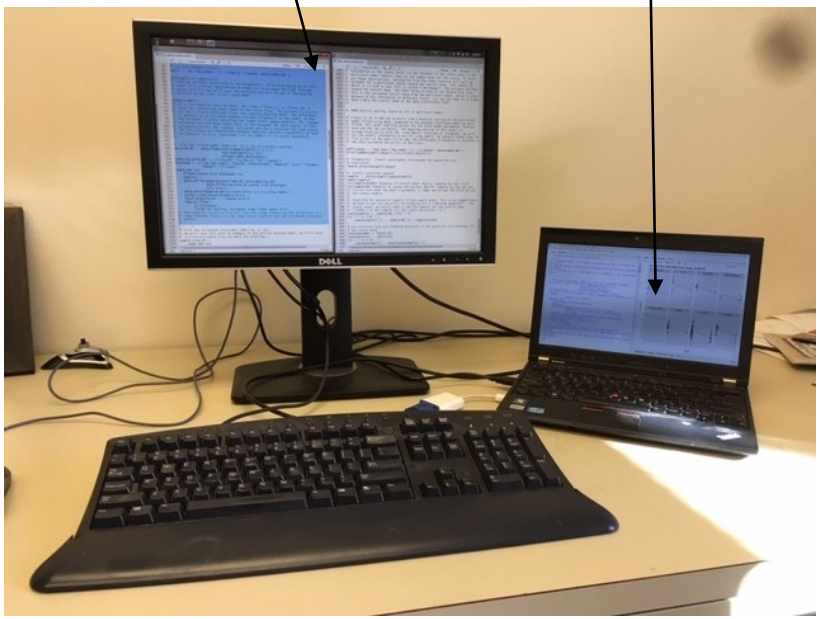
Productive workspaces

side by side layout

input output



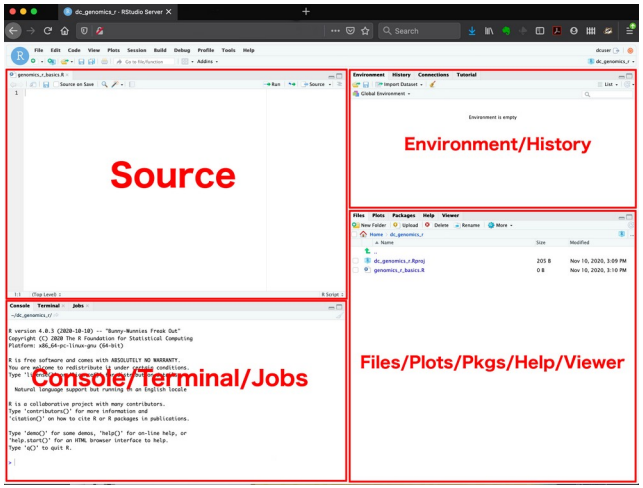
input output



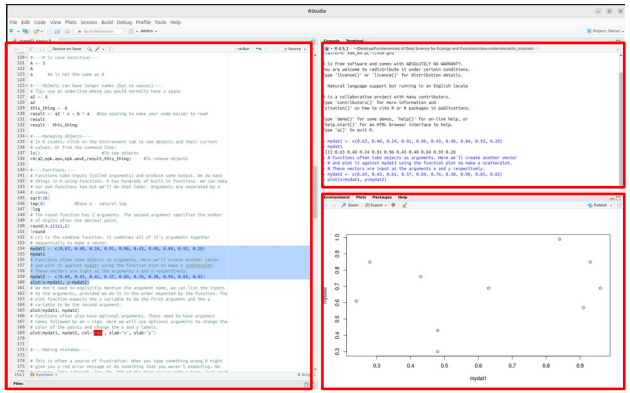
Multiple monitors are essential!

Productive workspaces

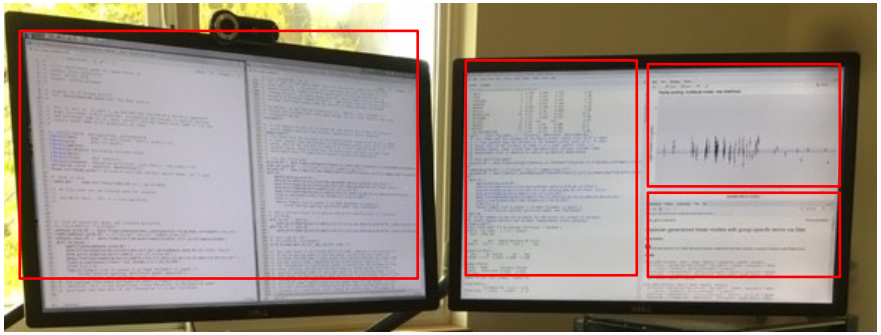
stacked



side by side



multimonitor side by side

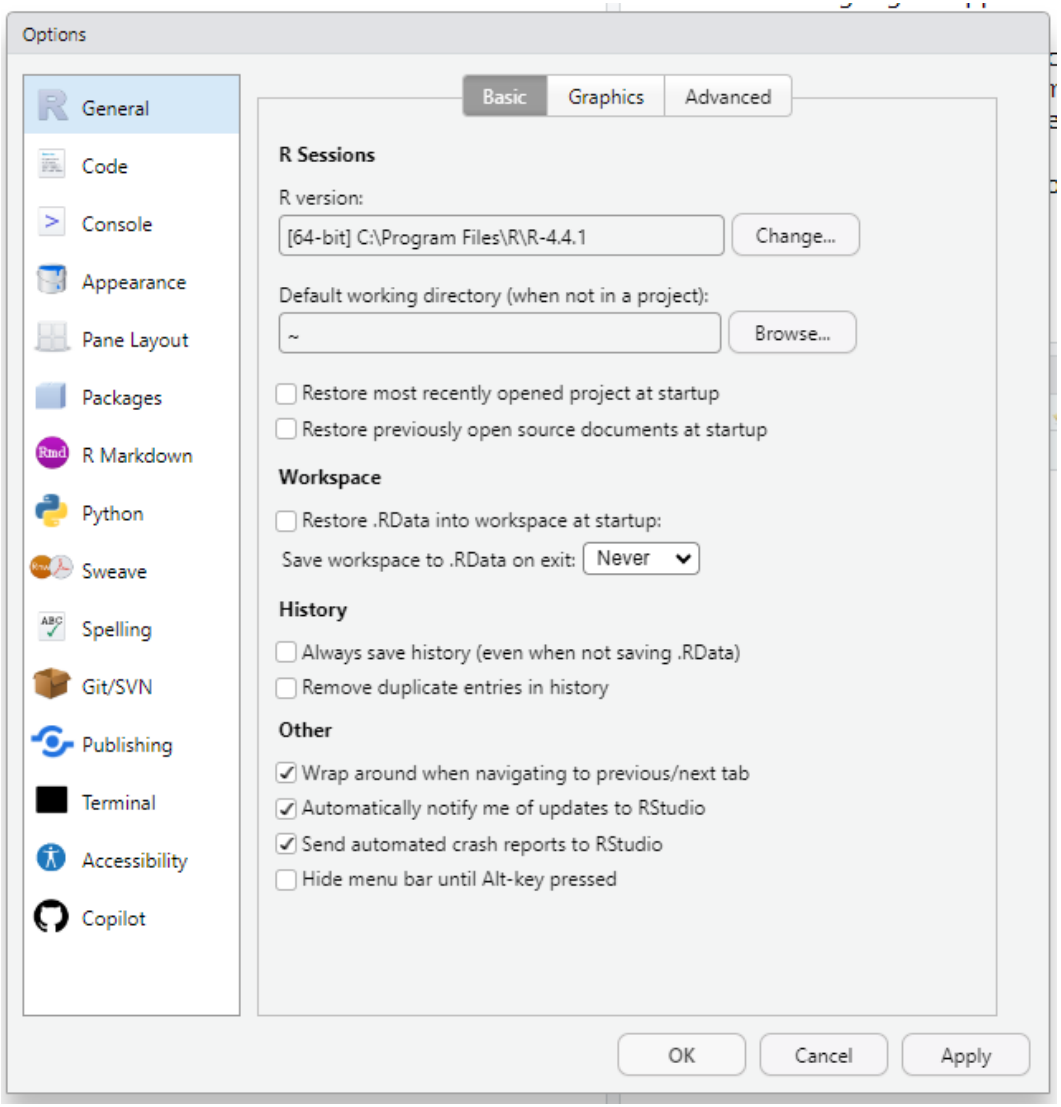


least productive → most productive

Productive workspaces

- IDE: Integrated development environment
  - all the tools for coding in one app
- RStudio
- VSCode
- Positron

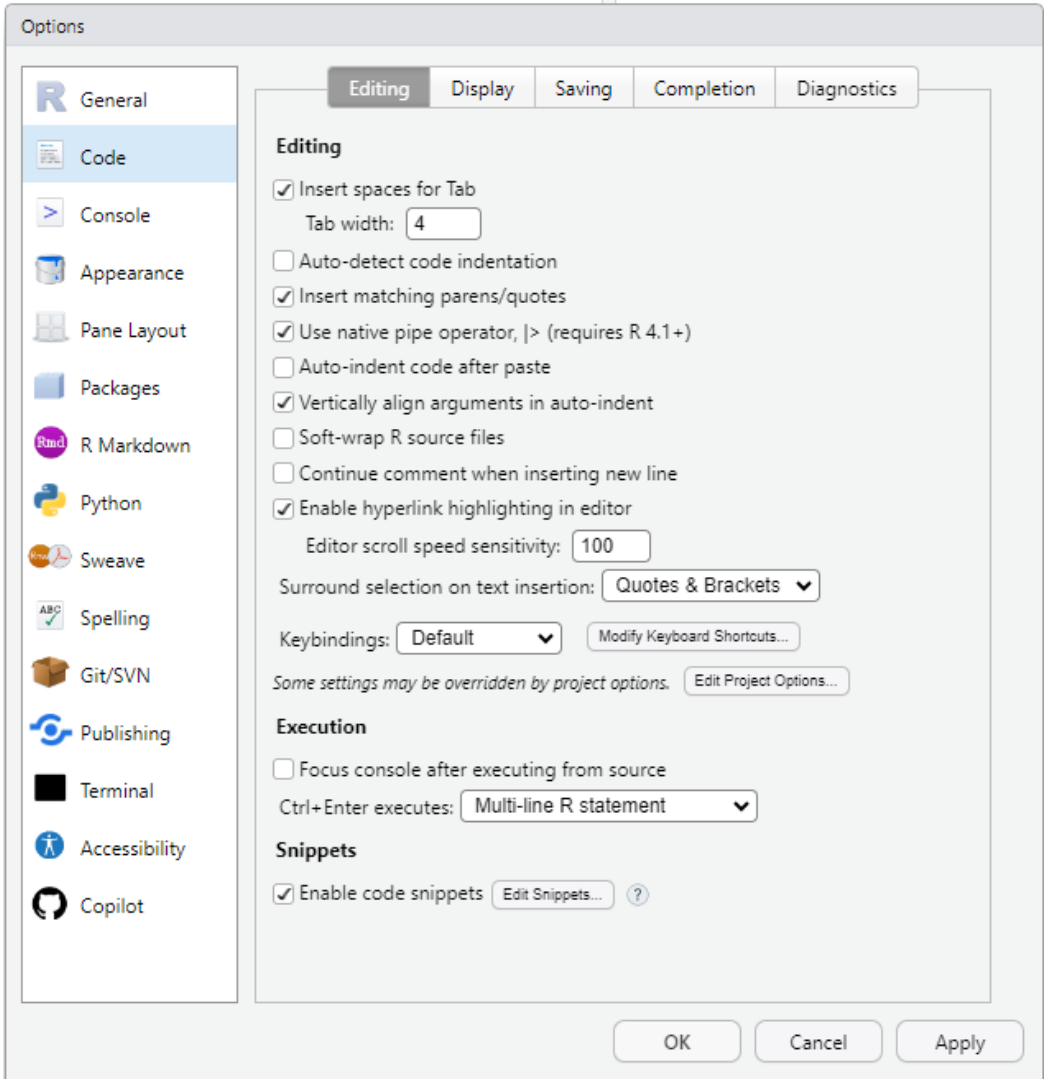
Productive workspaces



RStudio settings

Tools >
Global options

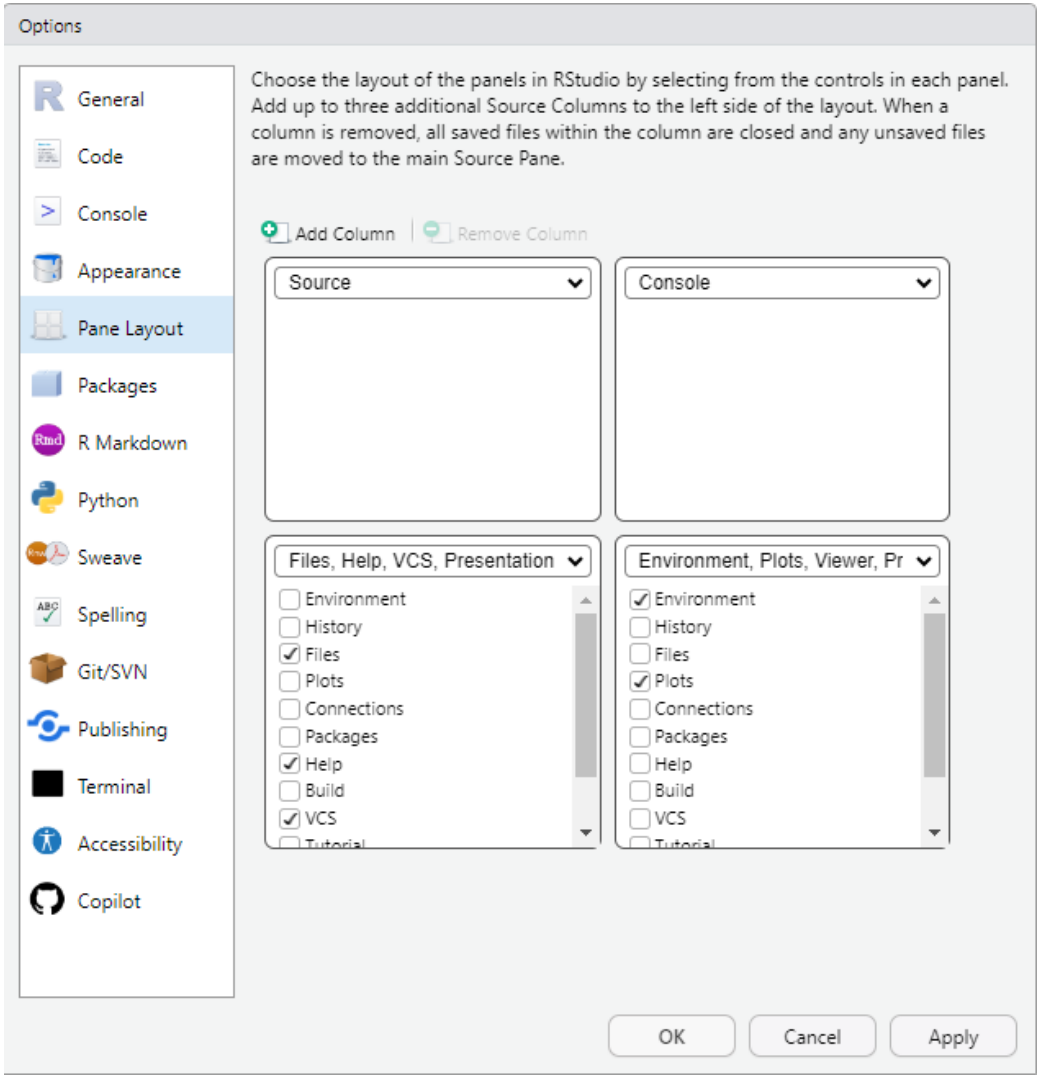
Productive workspaces



RStudio settings

Tools >
Global options

Productive workspaces



RStudio settings

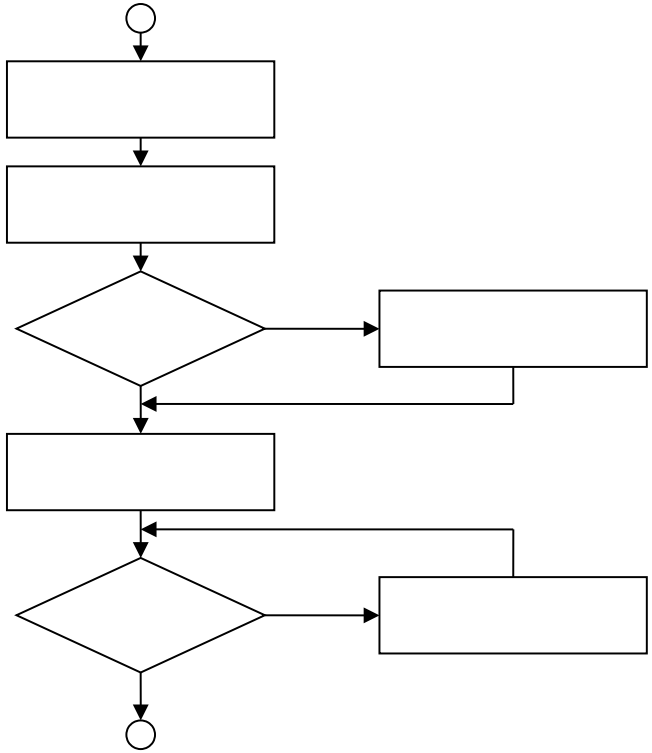
Tools >
Global options

Scientific programming

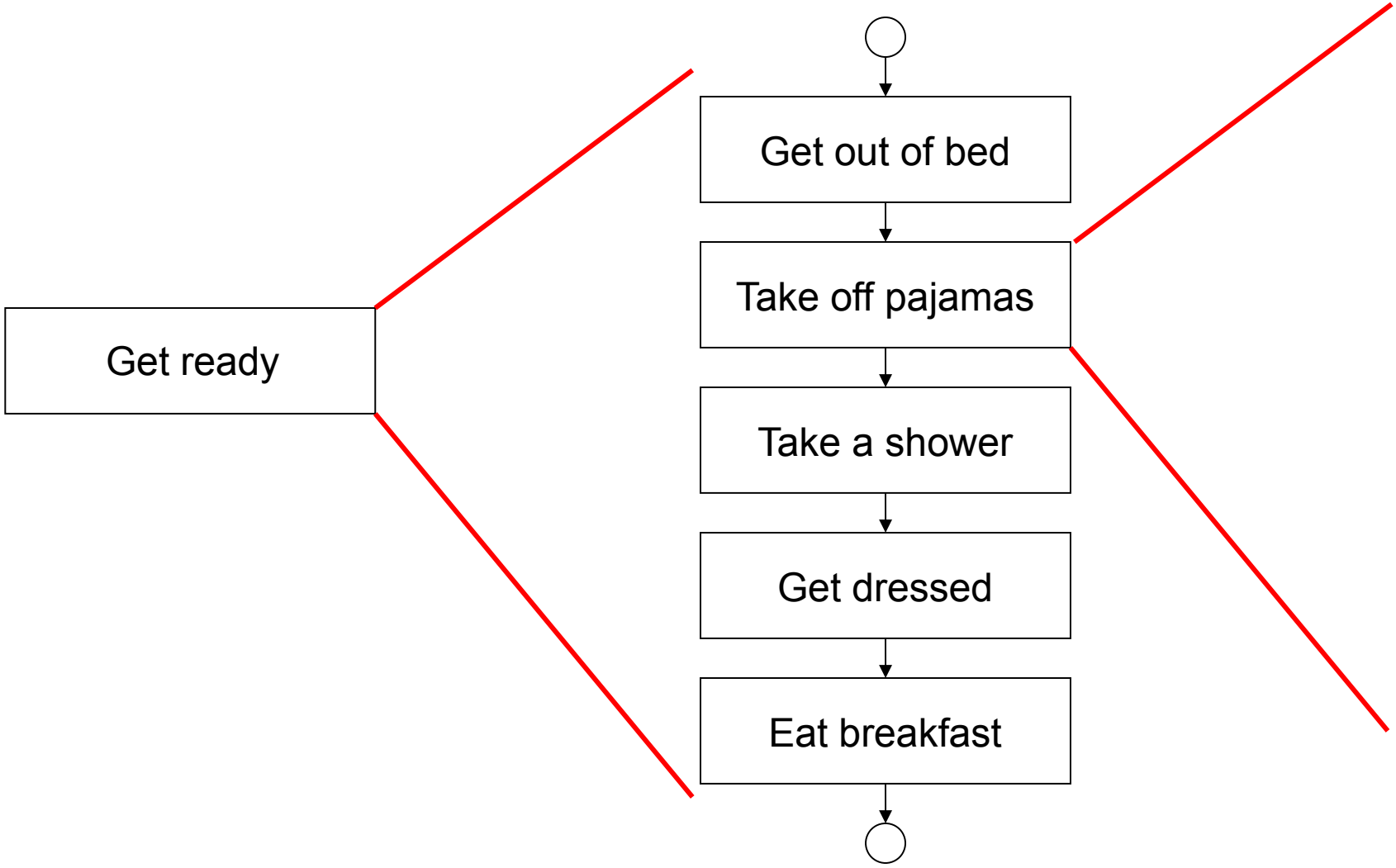
- Programming: code to implement an algorithm
- Scientific programming
  - Custom algorithms for specific problems, often “one off” (but often incorporate well-known algorithms for part of the problem)
  - Aim: Get the job done, be correct, be clear

What is an algorithm?

Sequence of actions



Top down refinement



Programming paradigms

- Structured programming
  - avoids jumping to arbitrary lines (“goto-less”)
  - fundamental to all other styles
- Object-oriented programming (OOP)
  - modularized design, objects "know" what they are supposed to do
  - useful for some specialized problems in science (e.g. individual based simulation models)
- Vectorized programming
  - a form of OOP, where vectors are the objects
- R & Python combine these

Programming paradigms

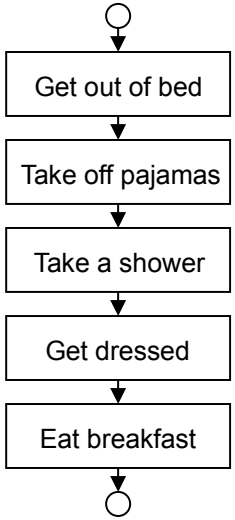
- Imperative programming
  - tell the computer what to do
  - objects can change state
- Declarative programming
  - tell the computer what you want
- Functional programming
  - declarative via functions
  - tell the computer what the relationship is
  - functions transform objects to other objects
  - input x -> f(x) -> output y
- R & Python combine these too

Structured programming

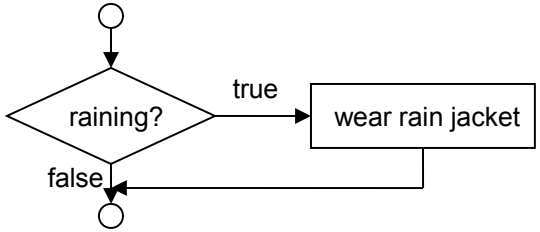
- Most algorithms are expressed in this form
- Control structures determine the order
- Functions encapsulate tasks
- You can solve any problem with a few general tools (structures)

Algorithm structures

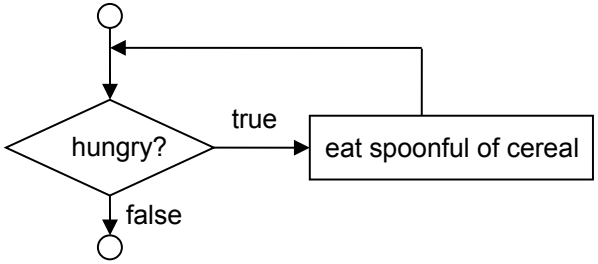
Sequence



Selection



Repetition

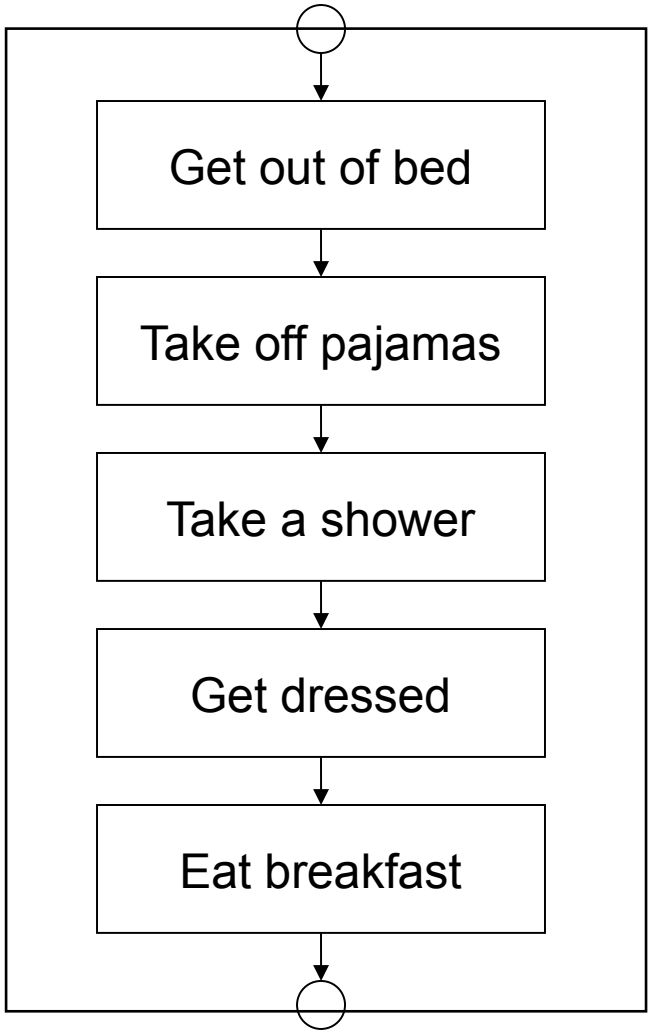


All problems can be solved!

Functions

get_ready()

Modularize algorithms



Algorithm structures

- Sequence structure
  - order to perform actions
- Selection structure (conditional, branches)
  - what to do depending on a decision
- Repetition structure (iteration, loops)
  - do something many times
- All languages have these
  - "flow control", "control structures"

Sequence structure

- Duh: one action after another in the order written in the program

Algorithm 1
Get out of bed
Take off pajamas
Take a shower
Get dressed
Eat breakfast
Cycle to work

Algorithm 2
Get out of bed
Take off pajamas
Get dressed
Take a shower
Eat breakfast
Cycle to work

Sequence structure

"Too easy" ?

It is still the most common source of programming errors

Programming tools

- Flowcharts (see above)
- Pseudocode

Pseudocode

- A tool to help you write a program
- Plain English “code”
- Formatted the same as code
- Pseudocode is “program like”
- Write pseudocode first, then translate to R, Python, or C code

Pseudocode & flowchart

If squirrel's hunger is greater than or equal to 60
    Print “Feed me”

indent (4 spaces)

Flowchart: