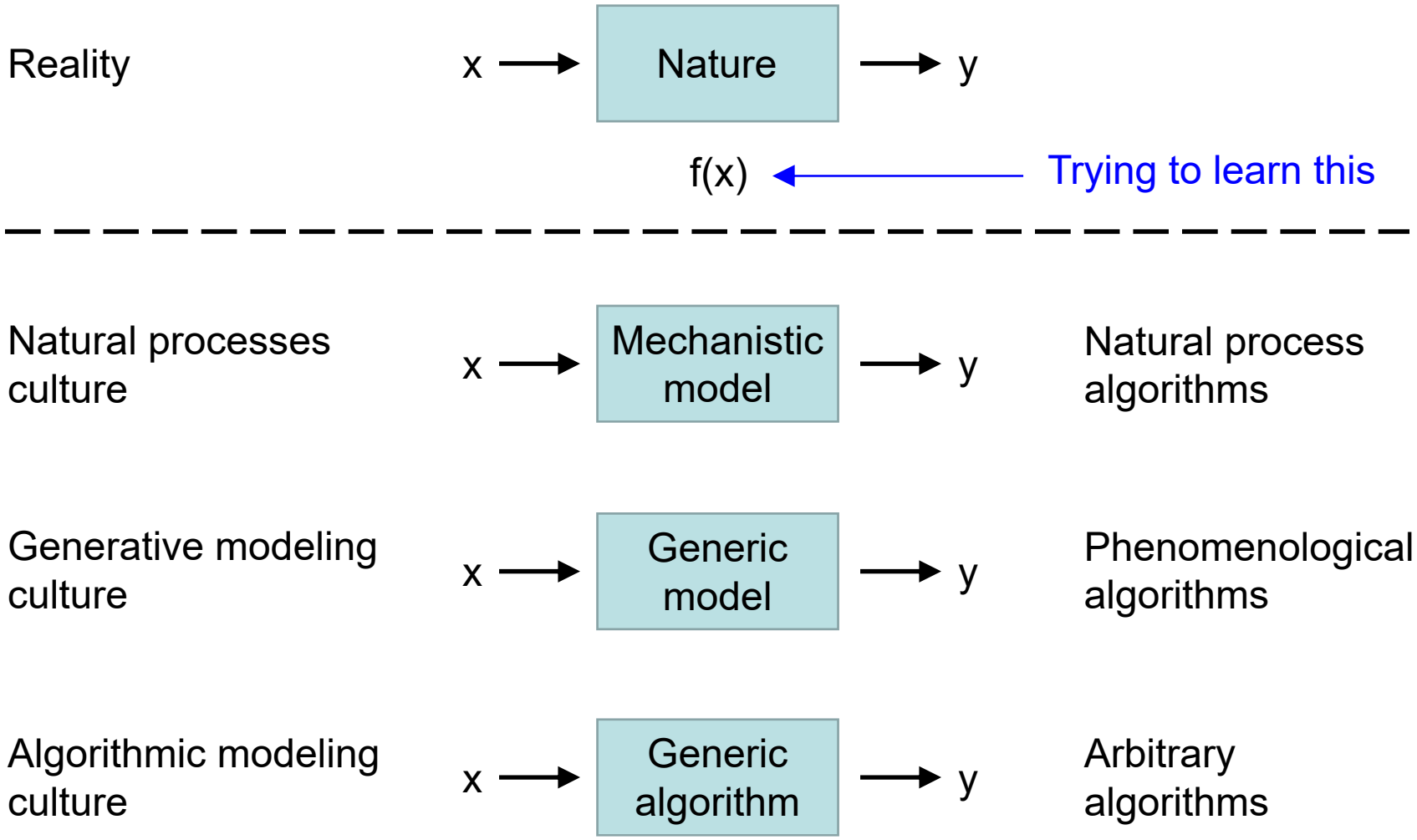
Today

- Data science cultures
- R & RStudio
- Base R basics
- Algorithms
- Structured programming

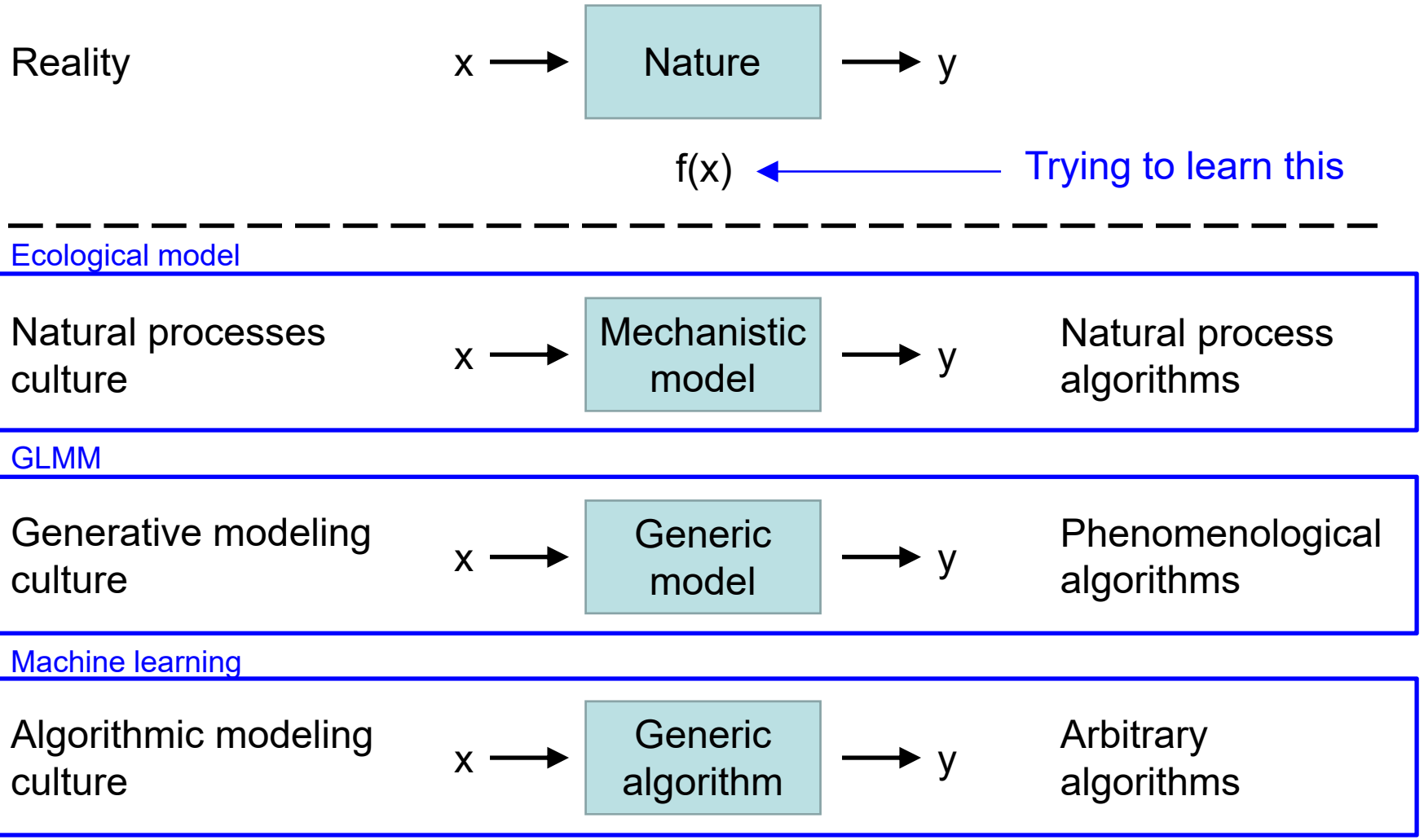
Data science cultures



f can mean different things in different cultures

Breiman (2001)
Denoho (2017)

Data science cultures



f can mean different things in different cultures

Breiman (2001)
Denoho (2017)

Discussion: data science in EBIO

- What are some goals of ecology and evolutionary biology?
- ... or what do we want to do with data?
- How can the different cultures help us with those goals?

R

- An environment for statistical and scientific computing
- An implementation of the language “S” (a stat. programming language, Bell Labs)
- Ihaka and Gentleman (U Auckland, New Zealand 1995), now many core developers
- Open source, free software

R

- Vehicle of choice for statistical research
- Most popular in ecology
- Important in biology
- Related and competing tools
  - Python
  - Matlab
  - Julia (up and coming, watch out!)

R components

- 1) Base: programming language, data handling, calculations, data analysis, graphics.
- 2) Contributed packages: 18533 CRAN + many others (e.g. on Github).

RStudio the organization

- Future: Posit
- RStudio IDE
- Tidyverse packages
- RMarkdown (future: Quarto)

RStudio IDE

- IDE: integrated development environment
- Has become quite complex
- Now much more than R
- Quick orientation
- Projects (.proj)

R basics: important concept list

- Operator precedence
- Assignment
- Objects

Assignment; updating an object

```
> a <- 1
> a + 1
> a
```

What is the value of "a" now?

```
> a <- 1
> a <- a + 1
> a
```

R basics: important concept list

- Operator precedence
- Assignment
- Objects
- Functions (and their arguments)
- Data structures
  - e.g. scalars, vectors, matrices, data frames

Vectors (1D array)

MyVec

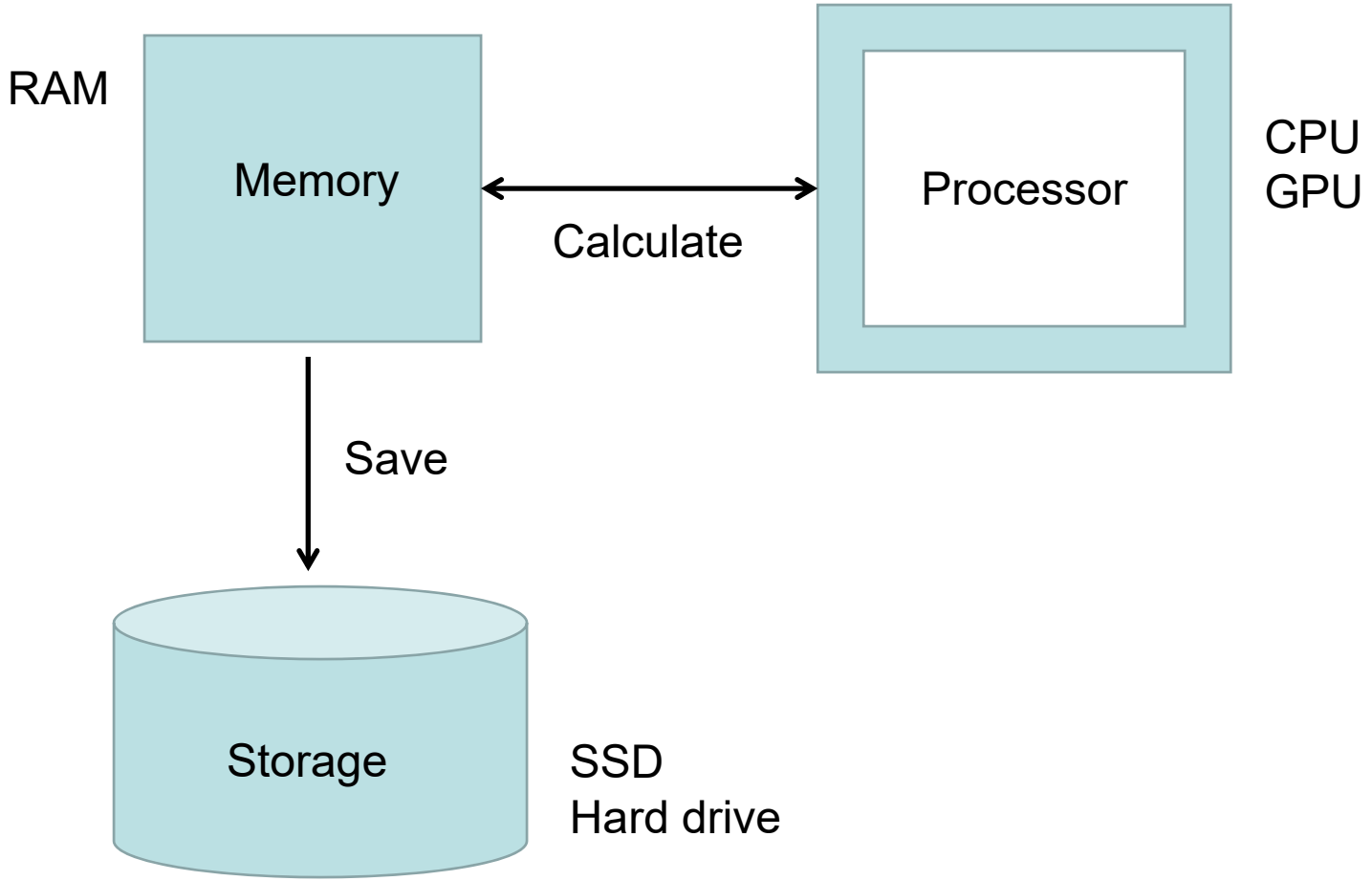
Element 1	6.115
Element 2	7.726
Element 3	8.352
etc	6.289
	1.087
	7.344
	2.911
	3.209
	5.290
	4.445
	2.505
	4.541
	5.568
	6.873
	5.208
	3.631

Each element is a slot in the computer's memory (RAM).

A vector is stored in contiguous memory slots.

Assigning an object allocates the memory space and records the address of the first slot.

How does my computer work?



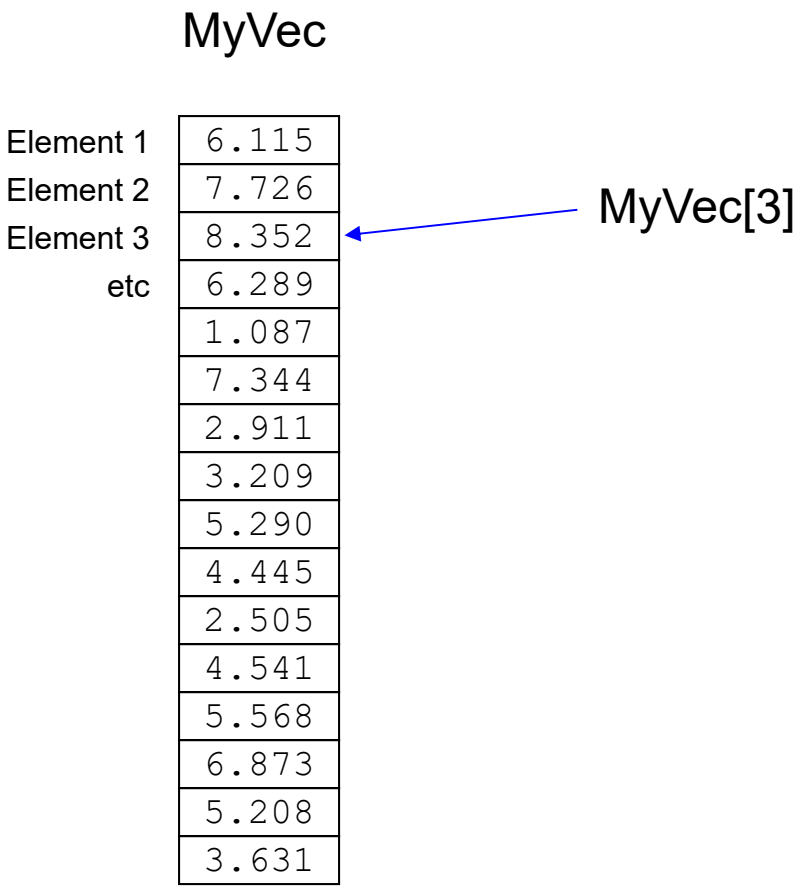
R basics: important concept list

- Operator precedence
- Assignment
- Objects
- Functions (and their arguments)
- Data structures
  - e.g. scalars, vectors, matrices, data frames
- Relational and logical operators
- Element-by-element operations

R basics: important concept list Part 2

- Use of [] to extract using object's indices

Vectors

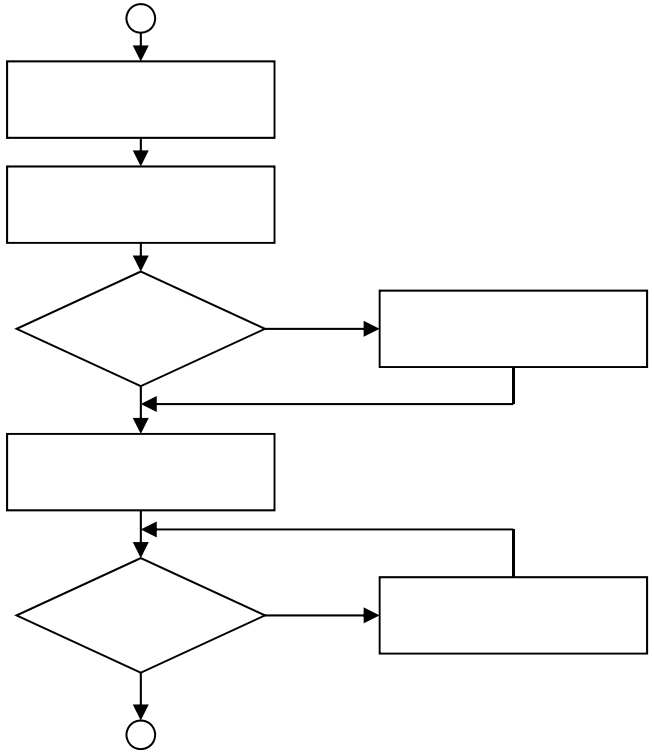

MyVec
Element 1
Element 2
Element 3
etc
6.115
7.726
8.352
6.289
1.087
7.344
2.911
3.209
5.290
4.445
2.505
4.541
5.568
6.873
5.208
3.631
MyVec[3]

R basics: important concept list Part 2

- Use of [] to extract using object's indices
- Numeric vs character vectors
- Getting data into R using .csv files
- Base graphics
- Packages

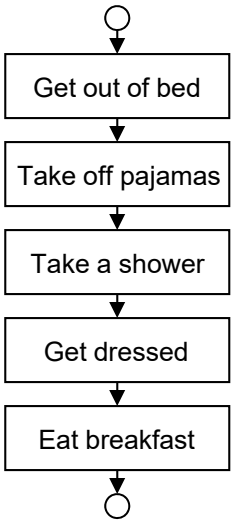
What is an algorithm?

Sequence of actions

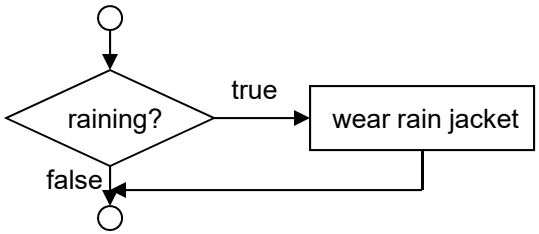


Algorithm structures

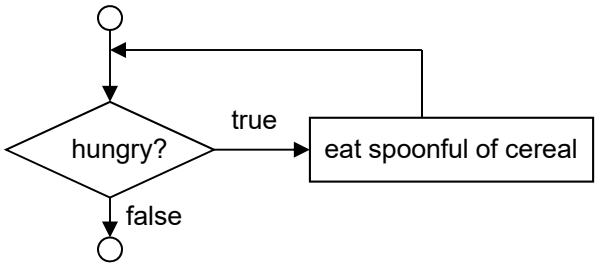
Sequence



Selection

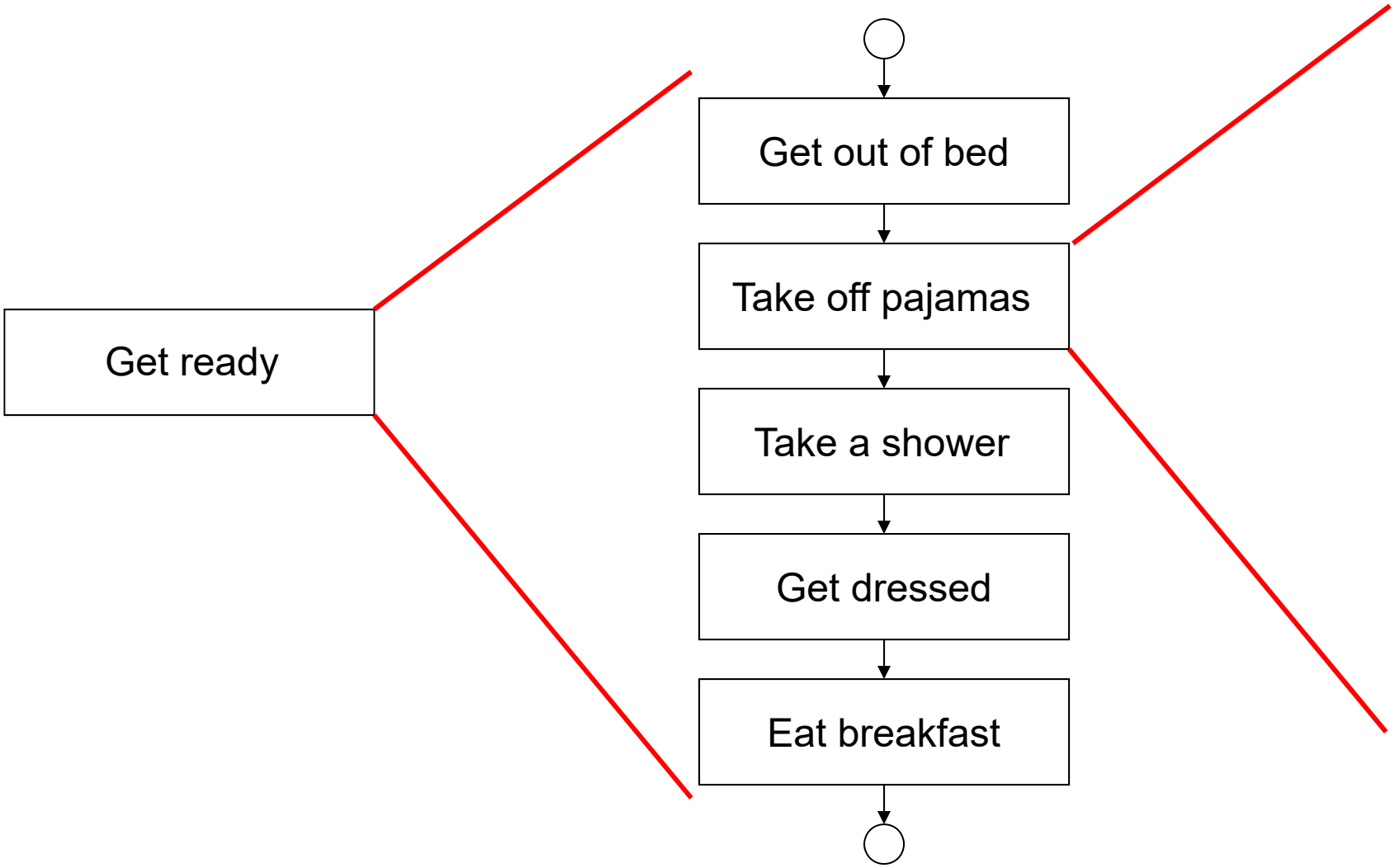


Repetition



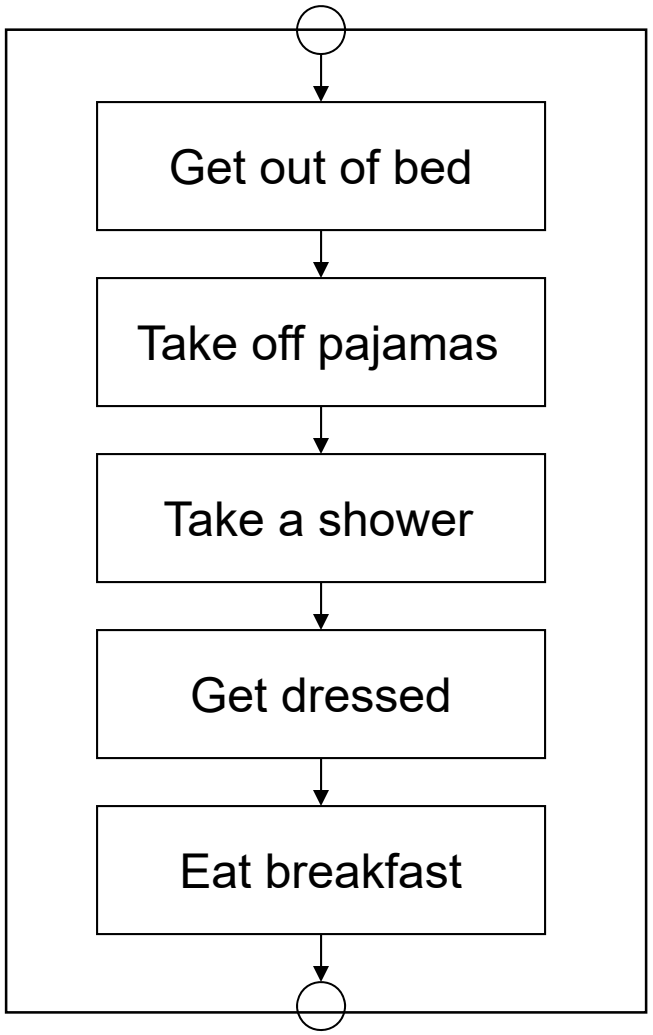
All problems can be solved!

Top down refinement



Functions

get_ready()



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 (not to be pretty or fancy, or even user friendly)

Languages

- Lower-level* programming languages
  - Interact directly with the computer's memory space
  - Compiled into an executable program: *.exe
  - 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 space
  - Programs are "scripts"
  - R, Matlab, Python, Mathematica, Julia
  - Run slower (sometimes only slightly); faster to write the code

Recommendations

- Learn a higher-level language first
  - R, Python, Julia
  - Learn a lower-level language too
  - most scientific algorithms are available in C or Fortran, with a slight edge to C
  - I recommend C because it is most like other programming languages that are widely used in software development (many are C-like, e.g. Objective C, Rust)

R

- R is written in C, C++ and Fortran (this is called the source code)
- The R source code is compiled into an executable program: R.exe
- Many packages have underlying C/C++ code

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 combines 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 something is
  - functions transform objects to other objects
  - input x -> f(x) -> output y
- R combines these too

Structured programming

- Best for most problems in science
- 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)

Control structures

- Sequence structure
  - order to perform actions
- Selection structure (branches)
  - what to do depending on a decision
- Repetition structure (loops)
  - do something many times
- That's it!! All languages have these.

```
> ?Control #for help in R
```

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 code

Selection structures

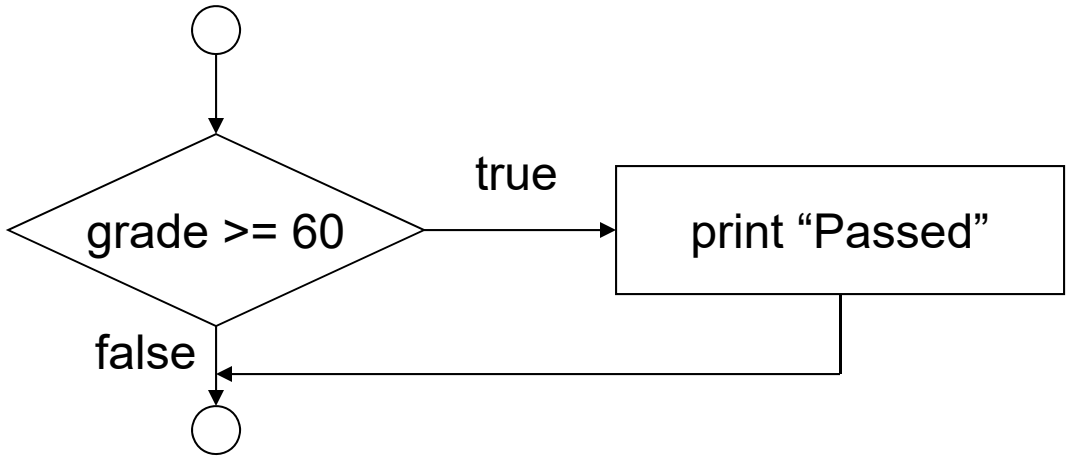
- Decisions: what to do if ...
- Pseudocode:

  If student's grade is greater than or equal to 60

      Print "Passed"

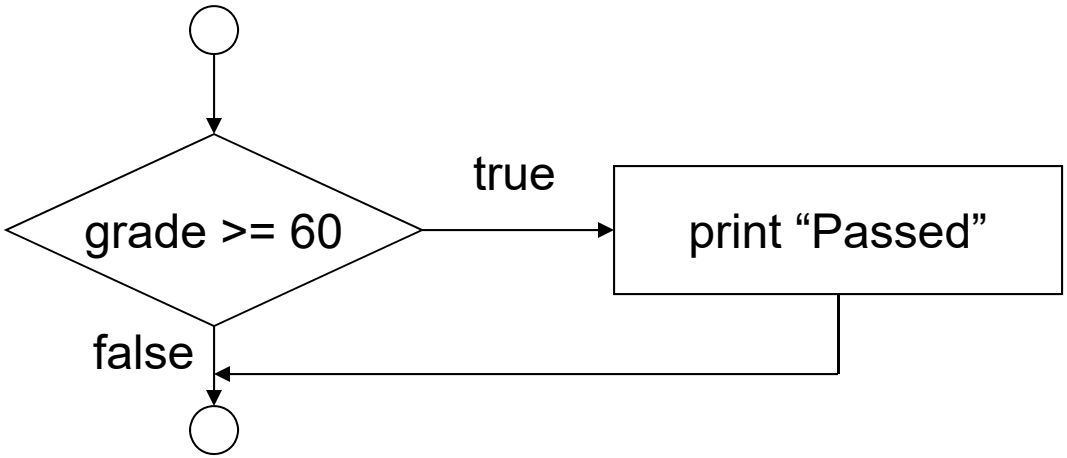
  indent (4 spaces)

- Flowchart:



R's if selection structure

```
> if ( condition ) expression
```



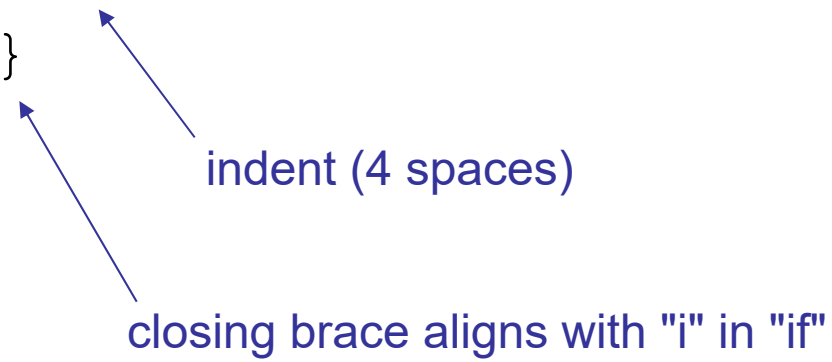

```
> student_grade <- 74

> if ( student_grade >= 60 ) print("Passed")
```

Predict: What is the output if you initialize the student's grade to be less than 60? Then try it.

Good programming practice

- Use braces {} and indenting to identify control structures

```
student_grade <- 74
if ( student_grade >= 60 ) {
  print(“Passed”)
}
```



Explicit vs implicit printing

- Explicit

```
> print(“Passed”)
> print(v1)
```

- Implicit

```
> “Passed”
> v1
```

- Use explicit printing within braces

```
> ?"{" #see R help for why
```

Multiple line expressions

```
> if ( condition ) {
>   expression1
>   expression2
>   etc
> }
```

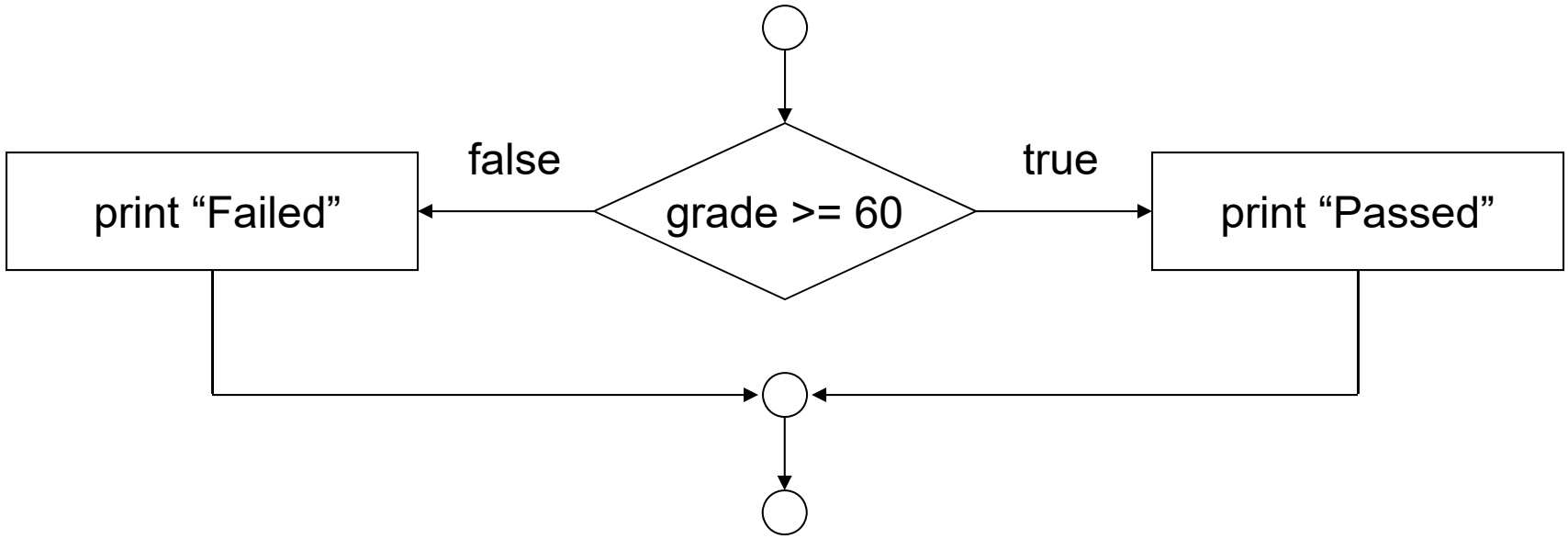

We call this a "block" of code

R’s selection structures

- if - single selection structure
- if/else - double selection structure
- if/else if - multiple selection structure

if/else selection structure

```
> if (condition) expr1 else expr2
```



if/else selection structure

```
> if ( condition ) {
>     expr1
> } else {
>     expr2
> }
```

Good programming practice:
Always use braces, spacing, and indenting

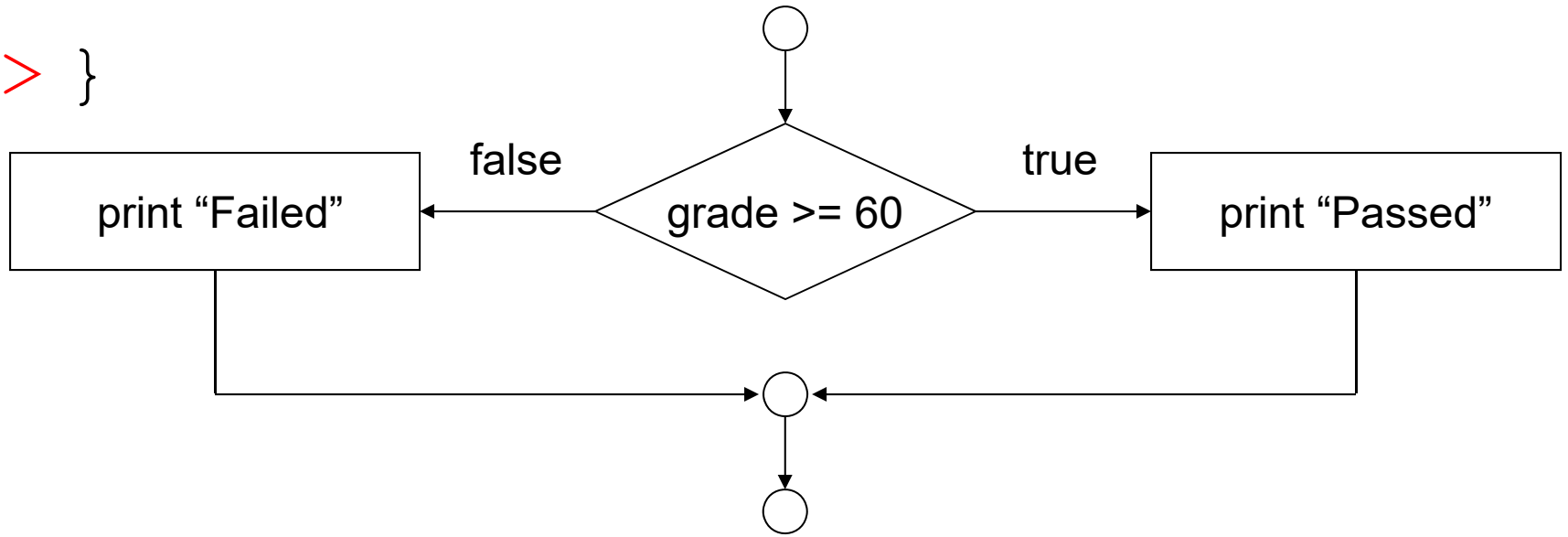
all lines between braces indented 4 spaces

"} else" must be on same line

if/else selection structure

```
> if ( condition ) {
>     expr1
> } else {
>     expr2
> }
```

Exercise:
Modify your code from the single student example to print “Passed” or “Failed” depending on the student’s grade.



Combining control structures

- Stacking
  - one after another
- Nesting
  - one inside another

Nested selection structures

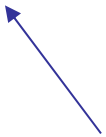
```
> if ( exam >= 70 ) {
>   if ( exam < 90 ) {
>     grade <- “B”
>   }
> }
```

What does this do?

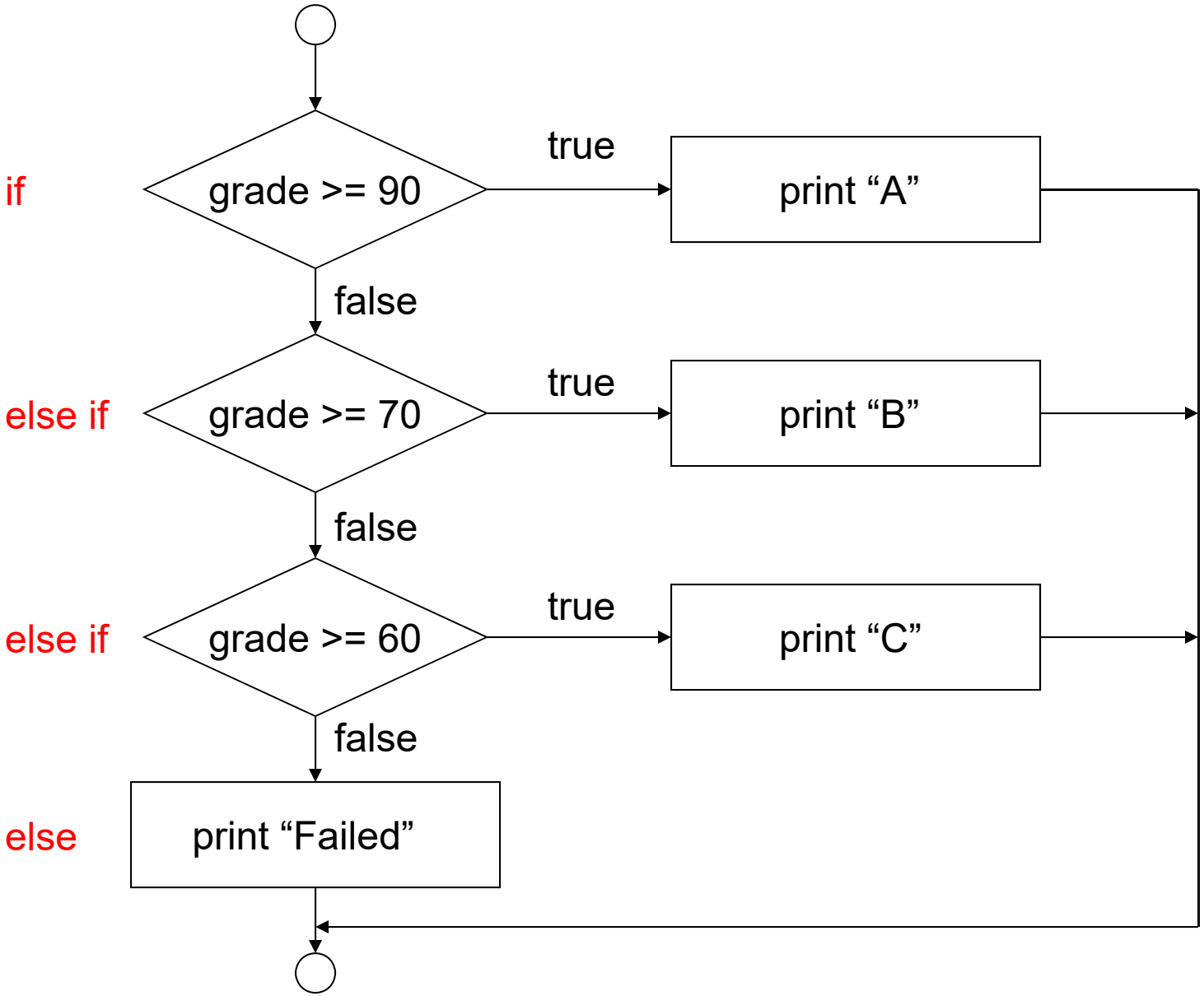
Nested selection structures

- if/else if
- creates a multiple selection structure

```
> if ( cond1 ) {
>     expr1
> } else if ( cond2 ) {
>     expr2
> } else {
>     expr3
> }
```



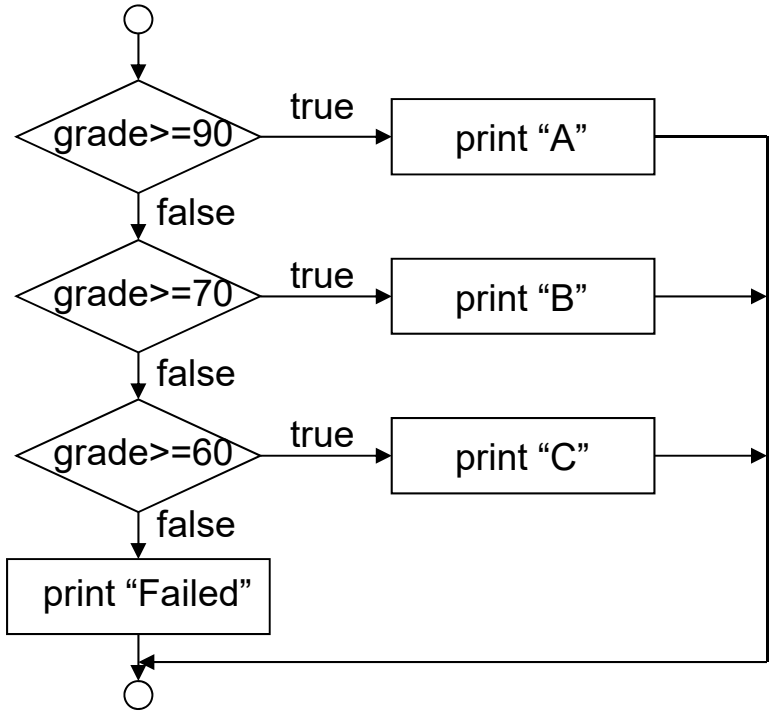
all lines between braces indented 4 spaces


if
grade >= 90
true
print “A”
false
else if
grade >= 70
true
print “B”
false
else if
grade >= 60
true
print “C”
false
else
print “Failed”

Exercise: nested if/else

- if/else if selection structure

```
if ( cond1 ) {
   expr1
} else if ( cond2 ) {
   expr2
} else {
   expr3
}
```



Exercise:
Modify your code from the single student example to print “A”, “B”, “C”, or “Failed” depending on the student’s grade.