Announcements

- 3 credits?
- Use Piazza!
- Positron: optional

Today

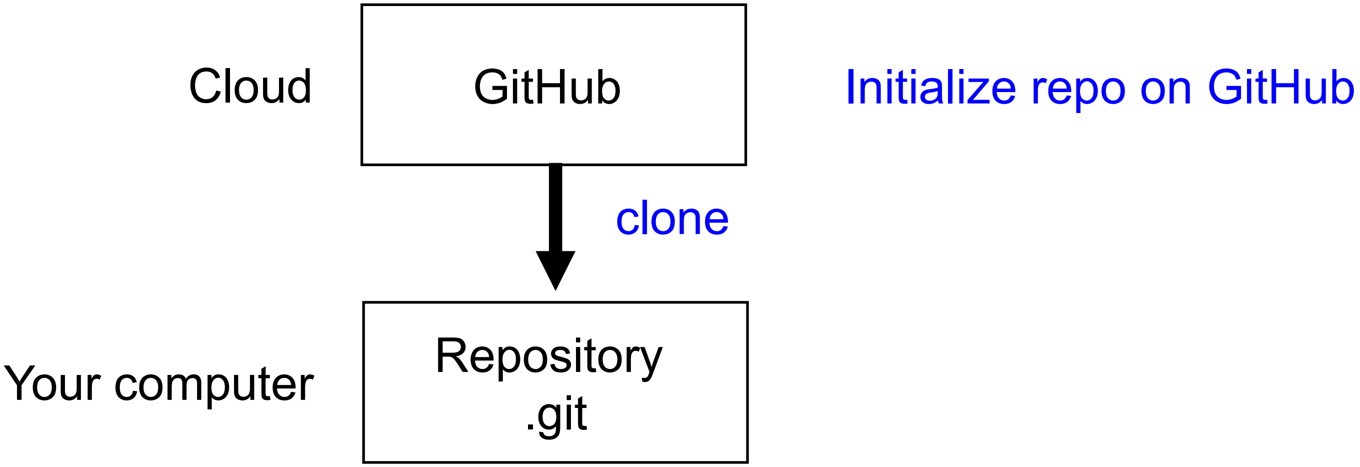
- Git & GitHub
- Programming algorithms

Git & GitHub

- Git
  - version control software
  - tracking changes, experimenting, merging contributions from collaborators
- GitHub
  - cloud service for storing and collaborating on git repositories

Initialize a Git repo

“GitHub first” workflow



Cloud

GitHub

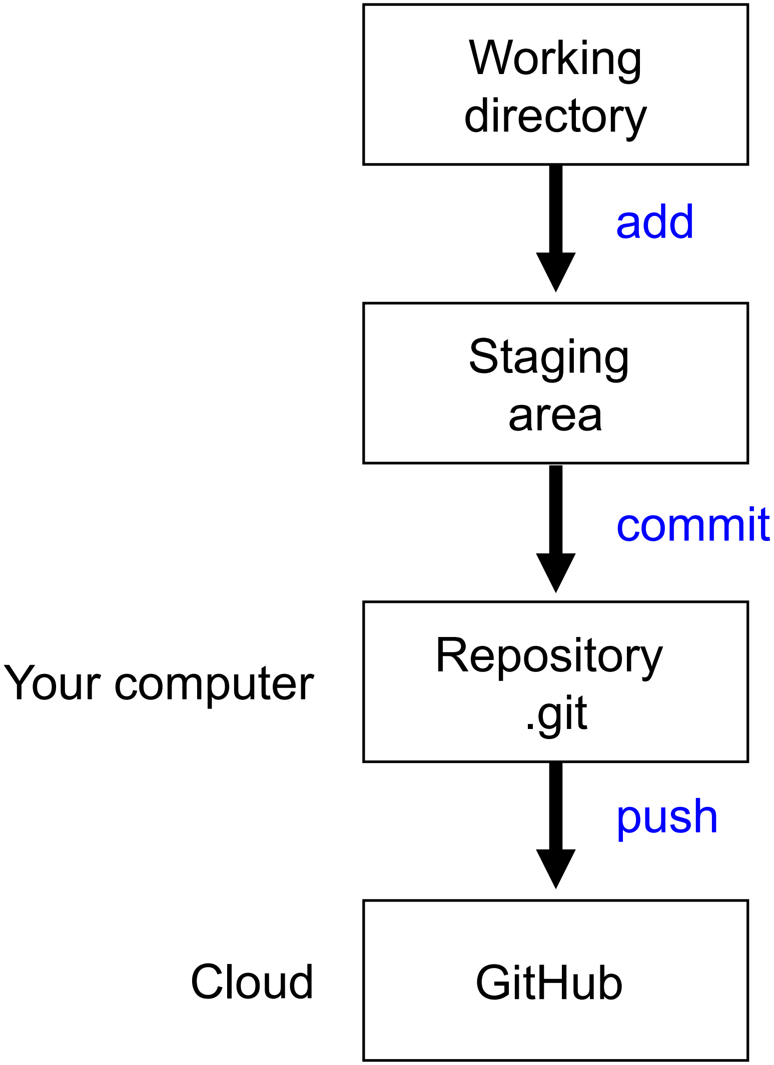
Initialize repo on GitHub

clone

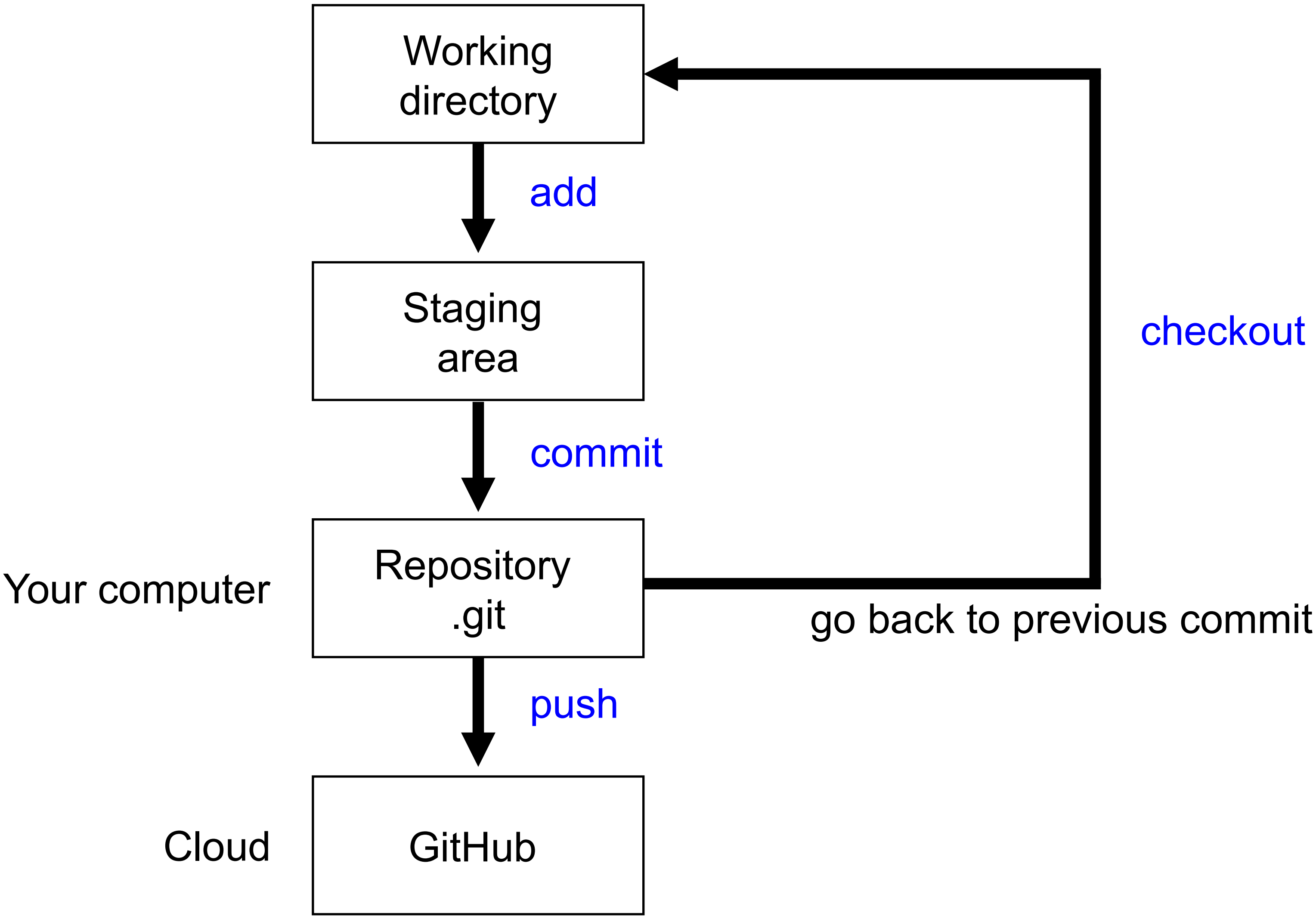
Your computer

Repository
.git

Version control workflow



Version control workflow



Gotchas

- GitHub web interface
  - upload or modify files (don't do this yet)
  - GitHub is now out of sync with your local repo
  - need more advanced techniques
- Clone once
  - cloning a second time into an existing repo will make a new repo nested within
- To recover
  - blow it all away (see happygit)

Scientific programming

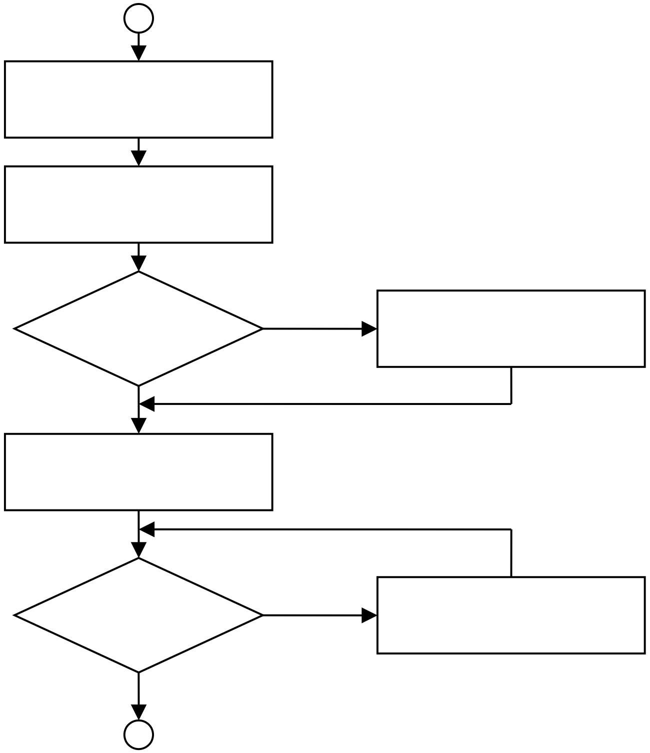
- Programming: code to implement an algorithm
- Scientific programming
  - Custom algorithms for specific problems, often “one off” (but often incorporate well-known algorithms for part of the problem)
  - Aims:
    - get the job done
    - be correct
    - be clear to other scientists
    - be reproducible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

Learning philosophy

- Algorithms first
  - models, data generating processes
  - understanding (nature, stats, etc)
  - getting stuff done (solving problems, automating)
- Other stuff is housekeeping
  - data structures, data types, libraries

What is an algorithm?

Sequence of actions



Step by step
(so is nature)

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 have all these
- C is structured
- C++ is object oriented

Programming paradigms

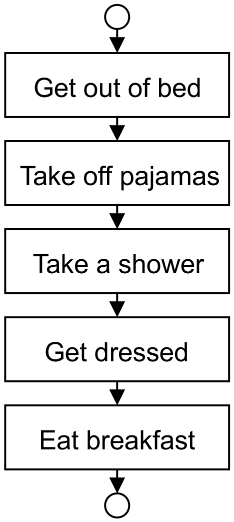
- Imperative programming
  - tell the computer what to do
  - objects can change state (side effects)
- Declarative programming
  - tell the computer what you want
- Functional programming
  - declarative via functions
  - tell the computer what the relationship is
  - functions transform objects to other objects
  - input x -> f(x) -> output y (no side effects)
- R & Python have all these
- C is imperative

Structured programming

- Most algorithms are expressed in this form
- Algorithm structures determine the order
- Functions encapsulate tasks

Algorithm structures (3)

Sequence


Get out of bed

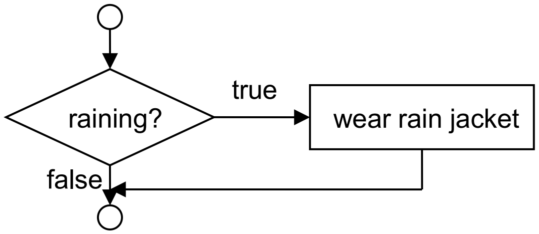
Take off pajamas

Take a shower

Get dressed

Eat breakfast

Selection


raining?

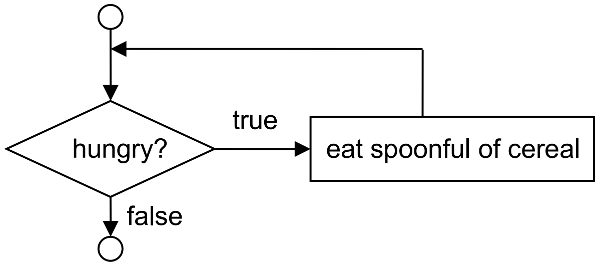
true

wear rain jacket

false

if

Repetition


hungry?

true

eat spoonful of cereal

false

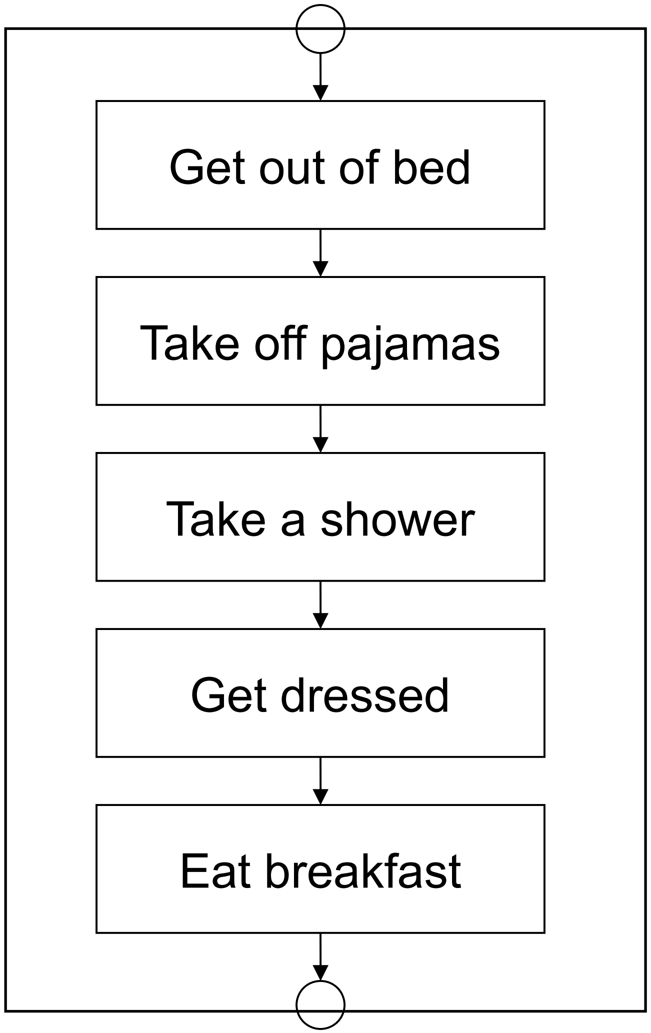
while

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"

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

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
```

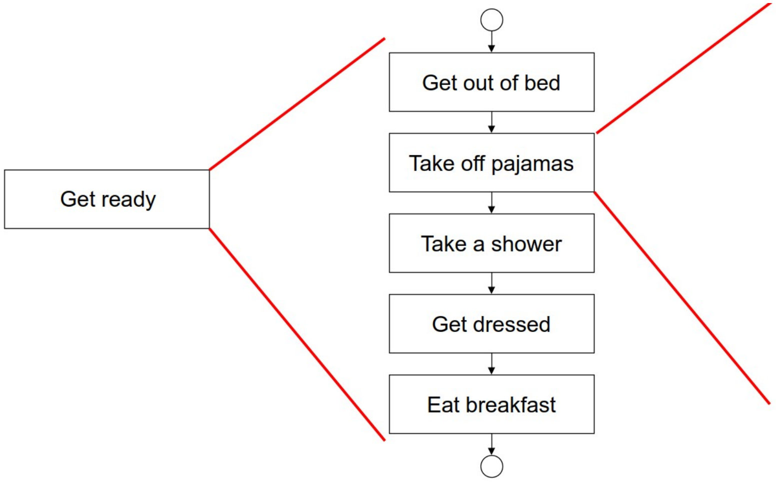
Sequence structure

"Too easy" ?

It's the most common source of programming errors!

Programming tools

- Flowcharts (see above)
- Top down refinement



- Pseudocode

Pseudocode

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

Structured programming

- 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

Structured programming

- 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

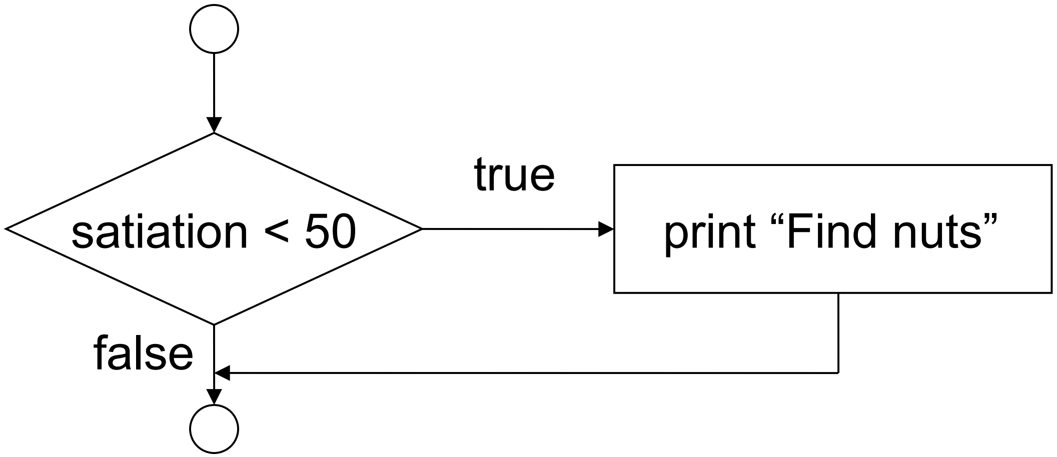
Selection structures

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

```
If squirrel's satiation is less than 50
    Print “Find nuts”
```

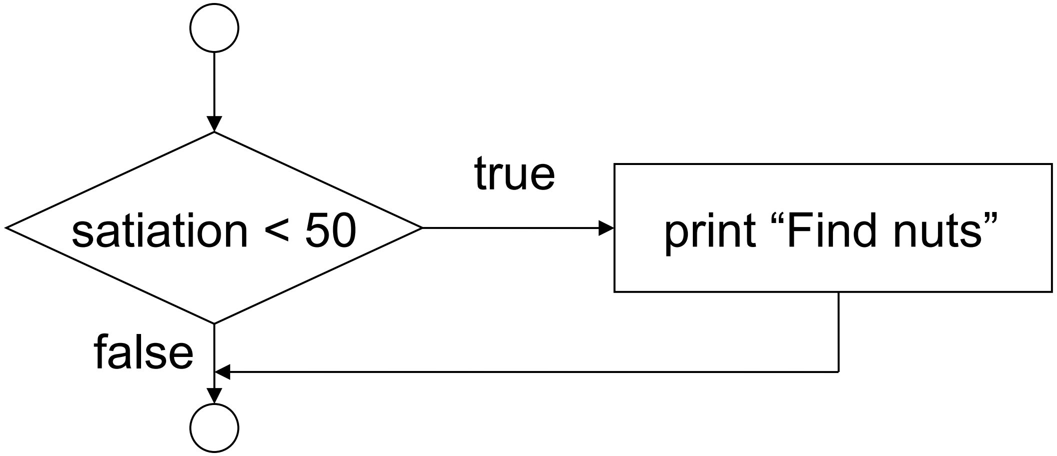
indent (4 spaces)

- Flowchart:



R's if selection structure

```
if(condition) expression
```



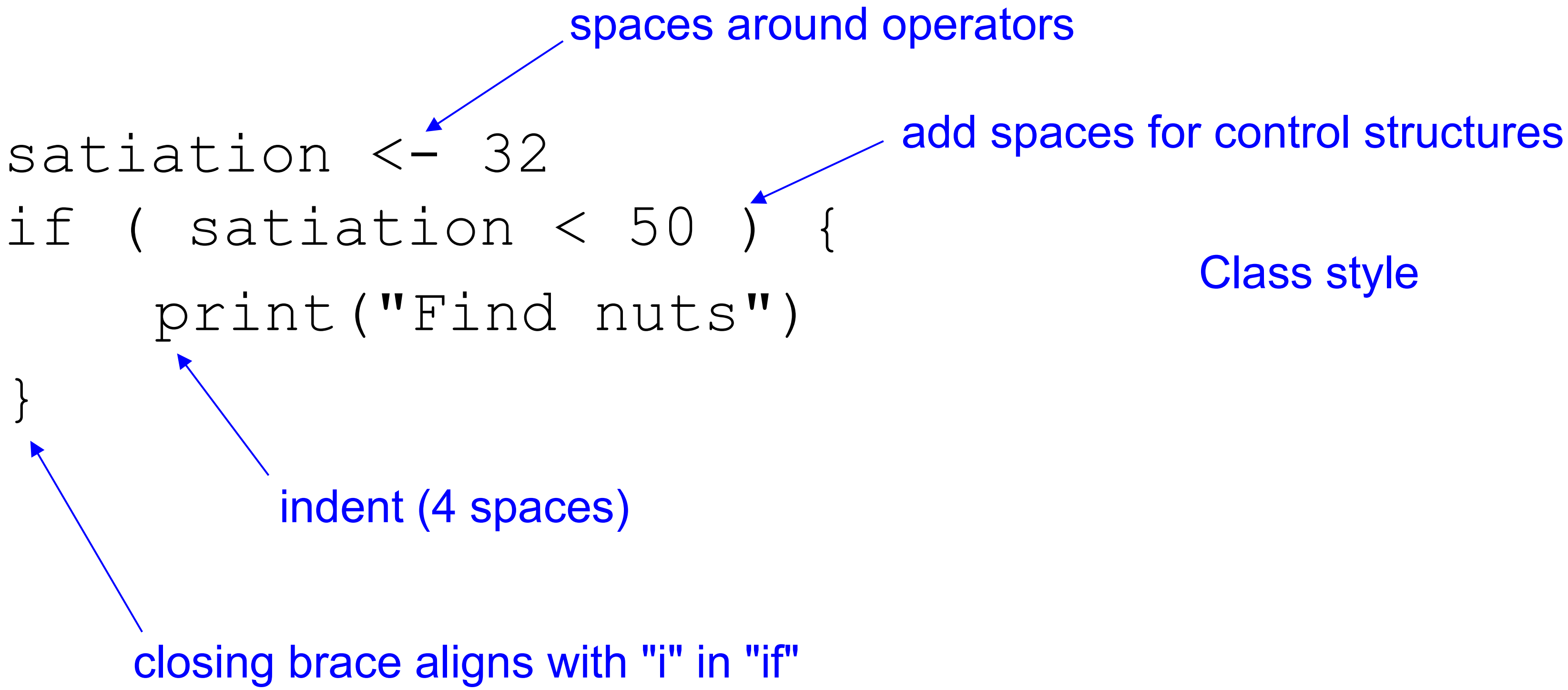
```
satiation <- 32

if(satiation < 50) print("Find nuts")
```

Predict: What is the output if you initialize satiation to be greater than 50? Then try it.

Good programming practice

- Use braces {}, spacing and indenting to identify control structures



```
satiation <- 32
if ( satiation < 50 ) {
    print("Find nuts")
}
```

spaces around operators

add spaces for control structures

Class style

indent (4 spaces)

closing brace aligns with "i" in "if"

A variety of styles

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

Tidyverse style

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

Another style

C’s if selection structure

```
if ( satiation < 50 ) {
    printf("Find nuts\n");
}
```

semicolons end action lines

C's if selection structure

type

assignment operator

```
int satiation = 32;
if ( satiation < 50 ) {
    printf("Find nuts\n");
}
```

semicolons end action lines

Python’s if selection structure

colon

```
if satiation < 50:
    print("Find nuts")
```

4 space indent: official python style
indents define control structures

Python's if selection structure

```
satiation = 32
if satiation < 50:
    print("Find nuts")
```

colon

4 space indent: official python style
indents define control structures

R: Explicit vs implicit printing

- Explicit

```
print("Hungry")
print(my_object)
```

- Implicit

```
"Hungry"
my_object
```

- Use explicit printing within braces

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

Example patterns

```
hungry <- TRUE
if ( hungry ) {
    print("Squirrel is hungry")
}
```

```
hungry = True
if hungry:
    print("Squirrel is hungry")
```

Example patterns

```
soil_moisture <- 0.08
if ( soil_moisture < 0.2 ) {
    print("Please water the plant")
}
```

```
soil_moisture = 0.08
if soil_moisture < 0.2:
    print("Please water the plant")
```

Example patterns

```
plant_stressed <- FALSE
soil_moisture <- 0.08
if ( soil_moisture < 0.2 ) {
    plant_stressed <- TRUE
}
```

```
plant_stressed = False
soil_moisture = 0.08
if soil_moisture < 0.2:
    plant_stressed <- True
```

Multiple line expressions

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

all lines indented (4 spaces)

This is a block of code

Multiple line expressions

```
satiation <- 42
if ( satiation < 50 ) {
    print("Squirrel is hungry")
    satiation <- satiation + 10
    print("Squirrel ate 10 nuts")
    print(paste("Satiation:", satiation))
}
```

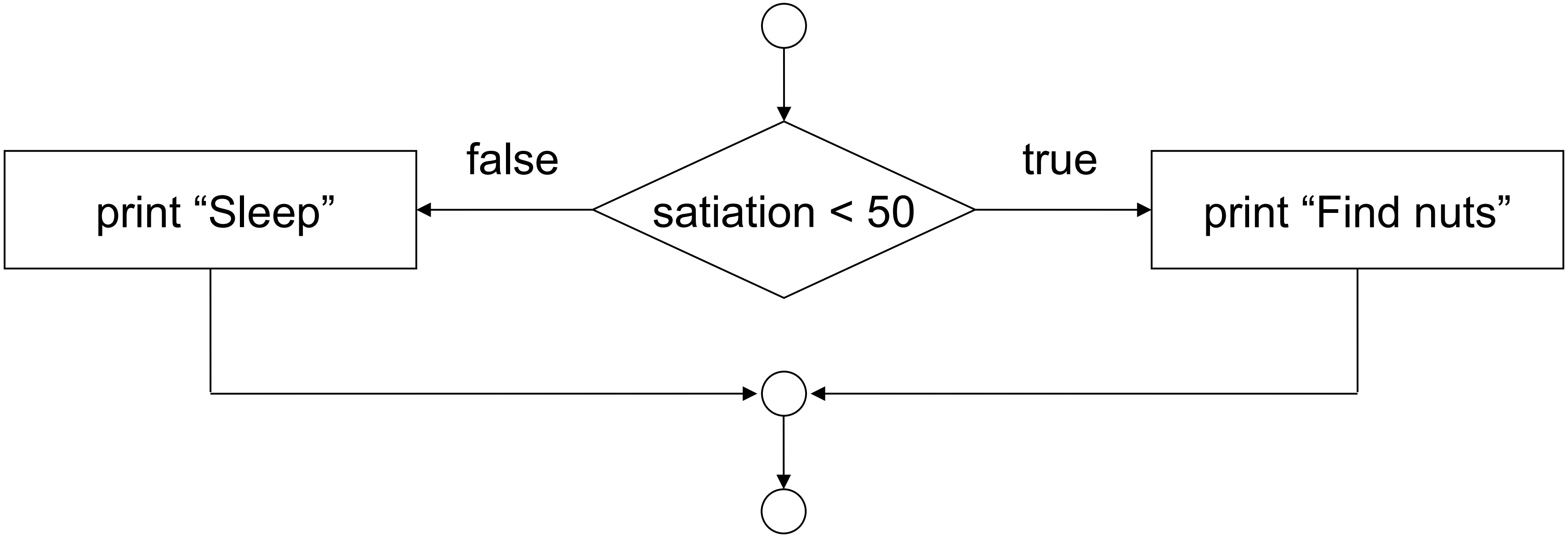
Multiple line expressions

```
satiation = 42
if satiation < 50:
    print("Squirrel is hungry")
    satiation <- satiation + 10
    print("Squirrel ate 10 nuts")
    print("Satiation:", satiation)
```

3 selection structures

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

if/else selection structure



if/else selection structure

```
if ( condition ) {
    expression_1
} else {
    expression_2
}
```

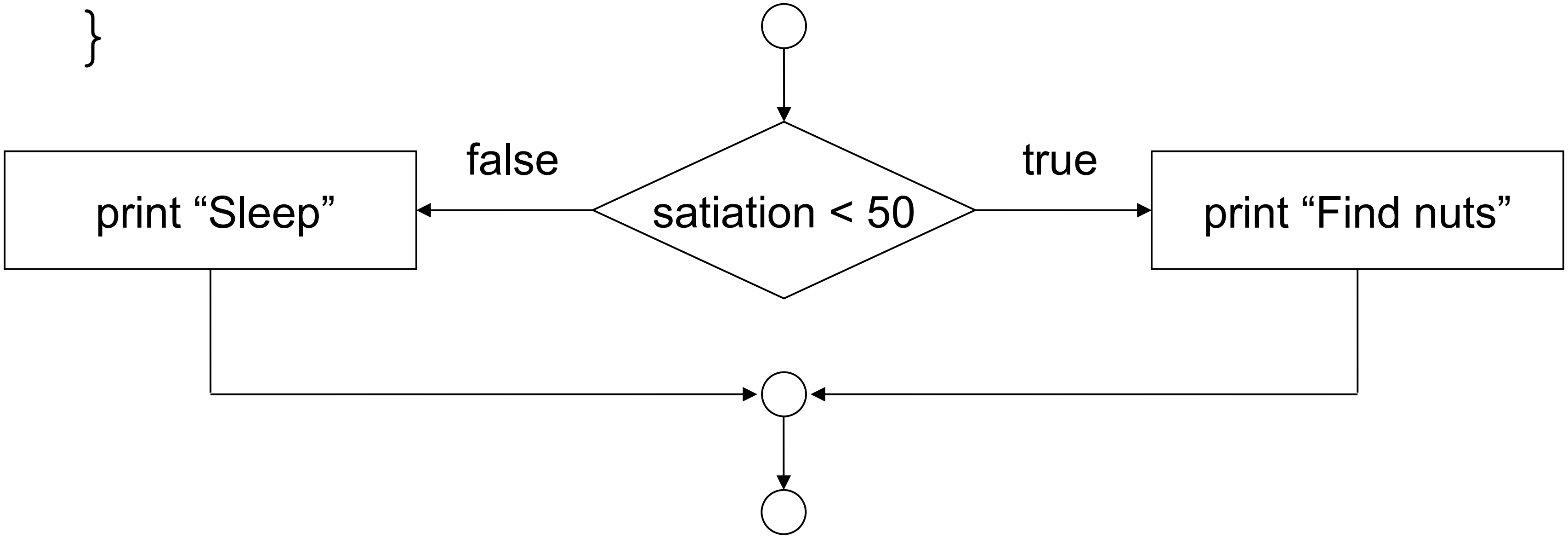
all lines between braces indented 4 spaces

"} else" must be on same line

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

if/else selection structure

```
if ( condition ) {
      expression_1
} else {
      expression_2
}
```



if/else selection structure

```
if ( condition ) {
    expression_1
} else {
    expression_2
}
```

all lines between braces indented 4 spaces

"} else" must be on same line

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

if/else selection structure

```
if ( condition ) {
    expression_1;
} else {
    expression_2;
}
```

if/else selection structure

```
if condition:
    expression_1
else:
    expression_2
```

Combining control structures

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

Stacked selection structures

```
plant_stressed <- FALSE
soil_moisture <- 0.35
solar_radiation <- 2000
if ( soil_moisture < 0.2 ) {
    plant_stressed <- TRUE
}
if ( solar_radiation > 1600 ) {
    plant_stressed <- TRUE
}
if ( plant_stressed ) {
    print("Plant is stressed")
}
```

Nested selection structures

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

What does this do?
Consider different values for exam

Nested selection structures

- nested if/else structures
- creates an if/else if multiple selection structure

```
if ( cond1 ) {
    expression_1
} else {
    if ( cond2 ) {
        expression_2
    } else {
        expression_3
    }
}
```

But don’t write it this way.

Nested selection structures

- nested if/else structures
- creates an if/else if multiple selection structure

```
if ( cond1 ) {
    expression_1
} else if ( cond2 ) {
    expression_2
} else {
    expression_3
}
```

all lines between braces indented 4 spaces

Nested selection structures

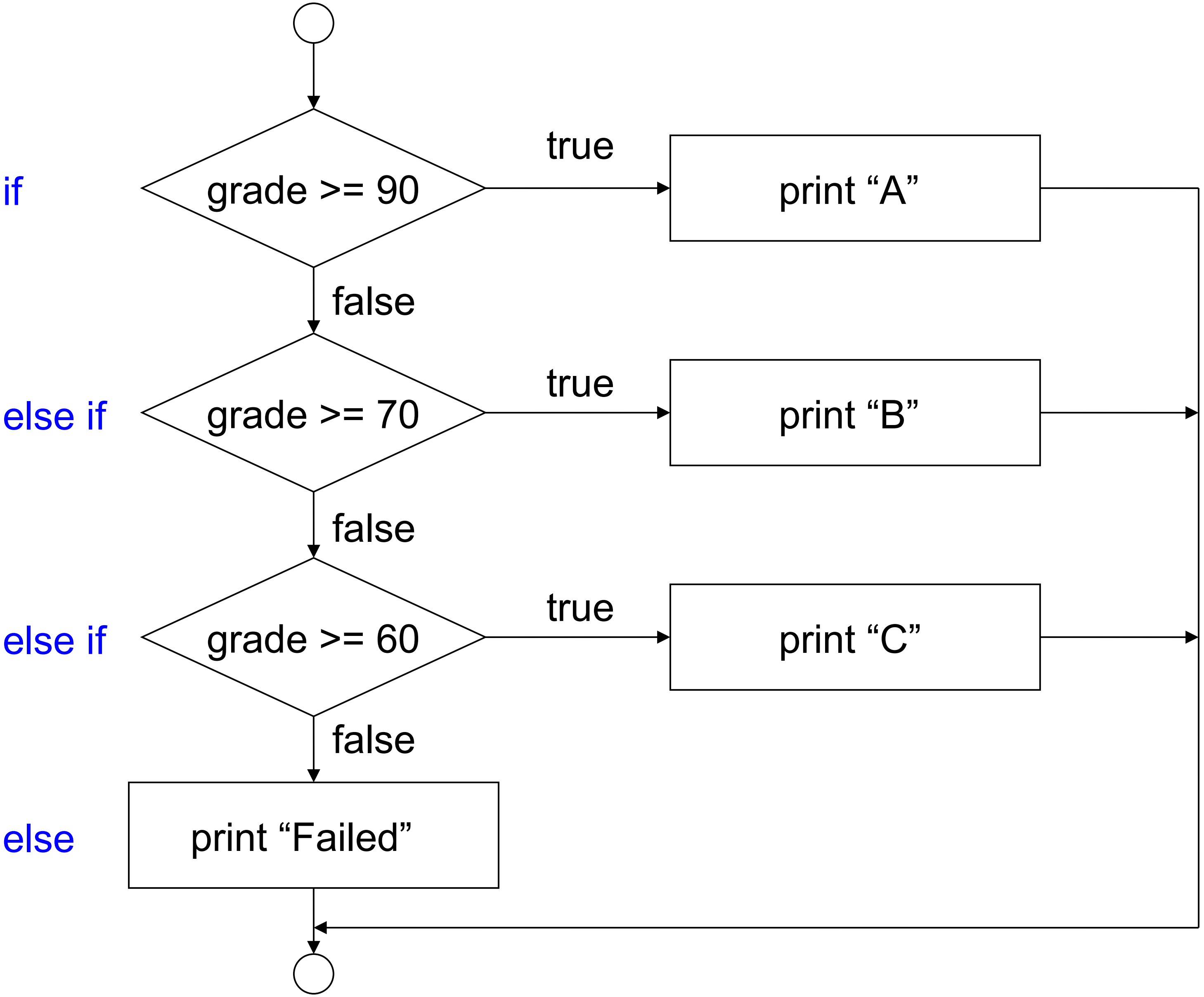
- nested if/else structures
- creates an if/else if multiple selection structure

```
if ( cond1 ) {
    expression_1;
} else if ( cond2 ) {
    expression_2;
} else {
    expression_3;
}
```

Nested selection structures

- nested if/else structures
- creates an if/else if multiple selection structure

```
if cond1:
    expression_1
elif cond2:
    expression_2
else:
    expression_3
```



if

grade >= 90

true

print “A”

false

else if

grade >= 70

true

print “B”

false

else if

grade >= 60

true

print “C”

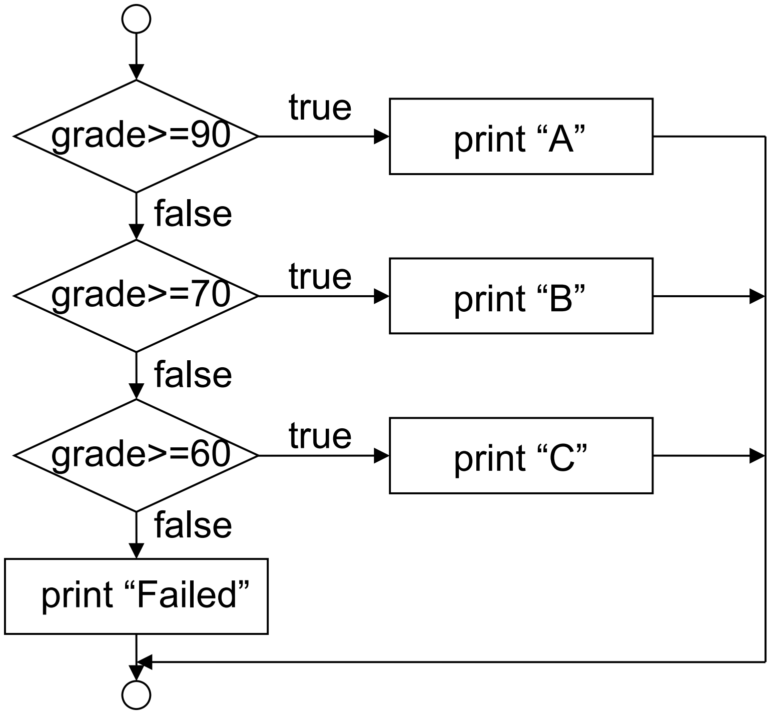
false

else

print “Failed”

if/else if selection structure

```
if ( cond1 ) {
    expr1
} else if ( cond2 ) {
    expr2
} else if ( cond3 ) {
    expr3
} else {
    expr4
}
```



Exercise:
Code this in R or Python to print “A”, “B”, “C”, or “Failed” depending on the student’s grade.