Announcements

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

Today

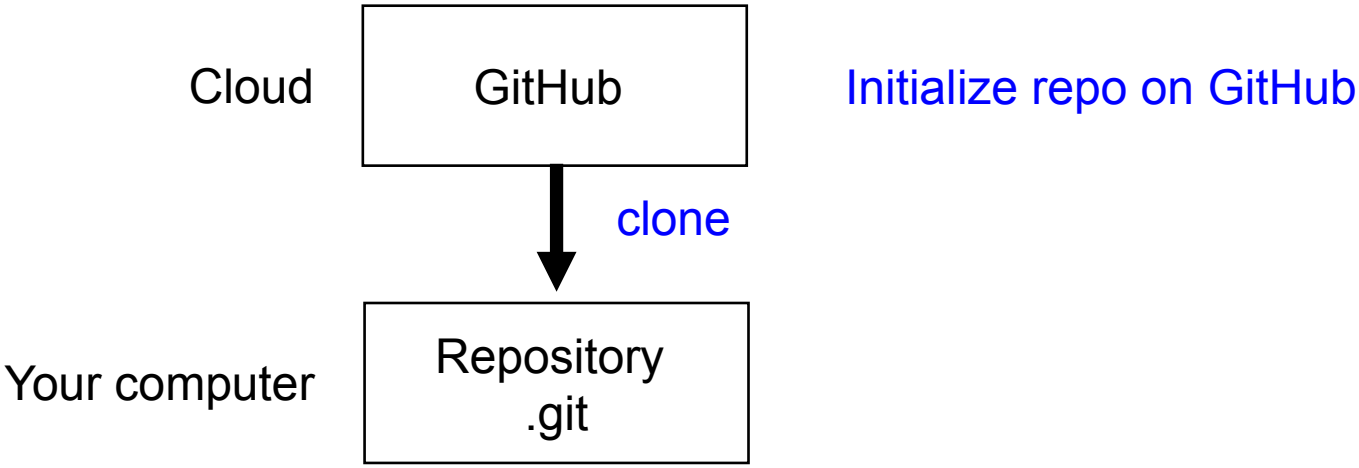
- Git & GitHub
- Programming algorithms

Git & GitHub

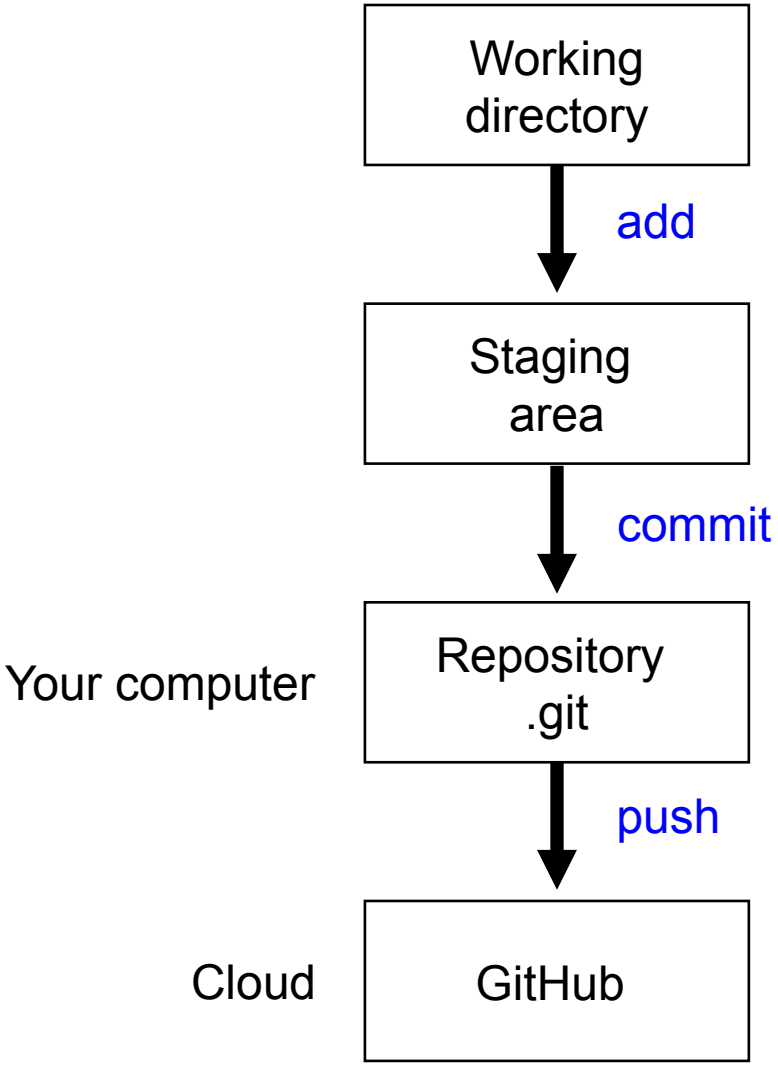
- Git
  - version control software
  - tracking changes, experimenting, merging contributions from collaborators
- GitHub
  - cloud service for storing and collaborating on git repositories

Initialize a Git repo

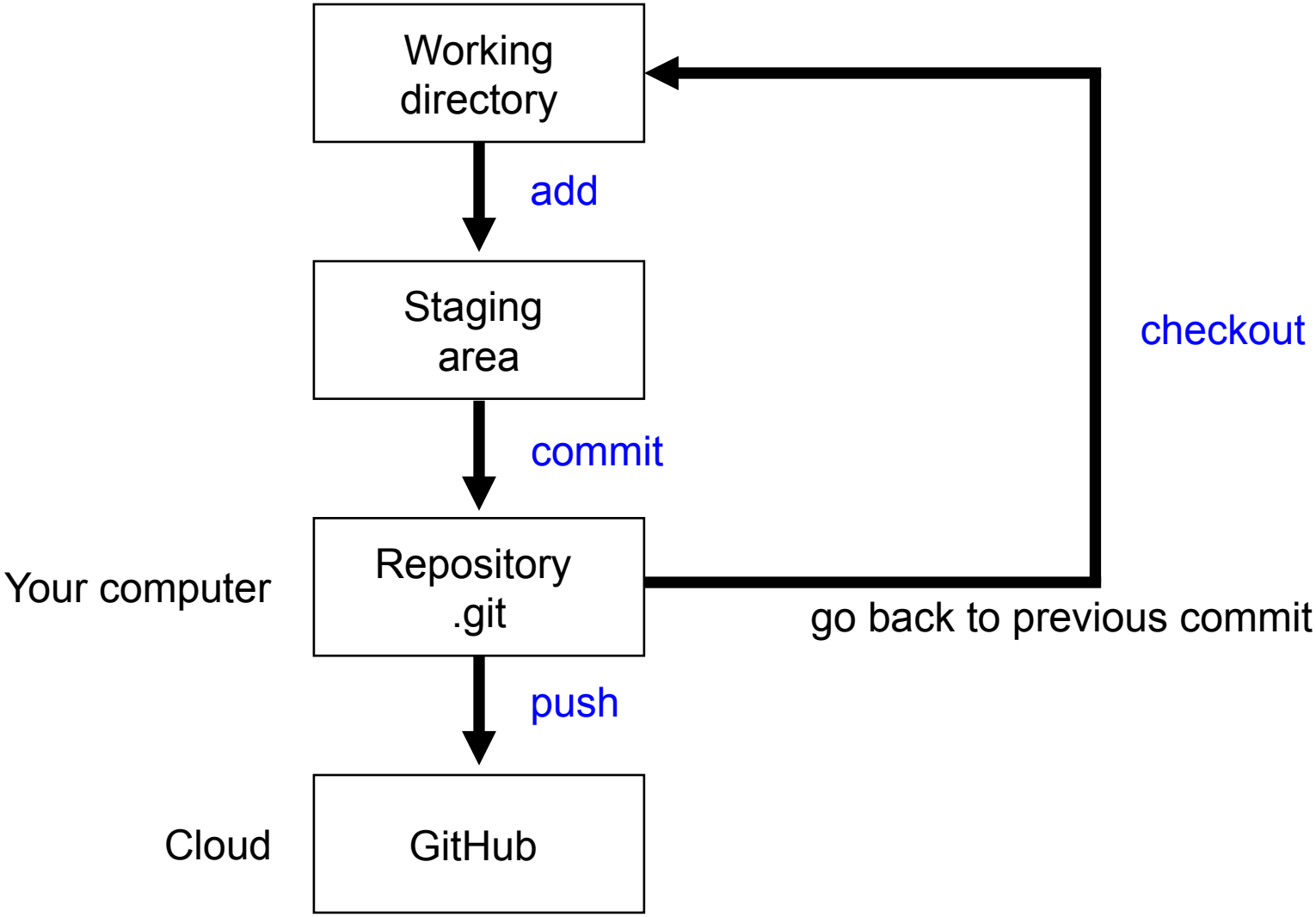
“GitHub first” workflow



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

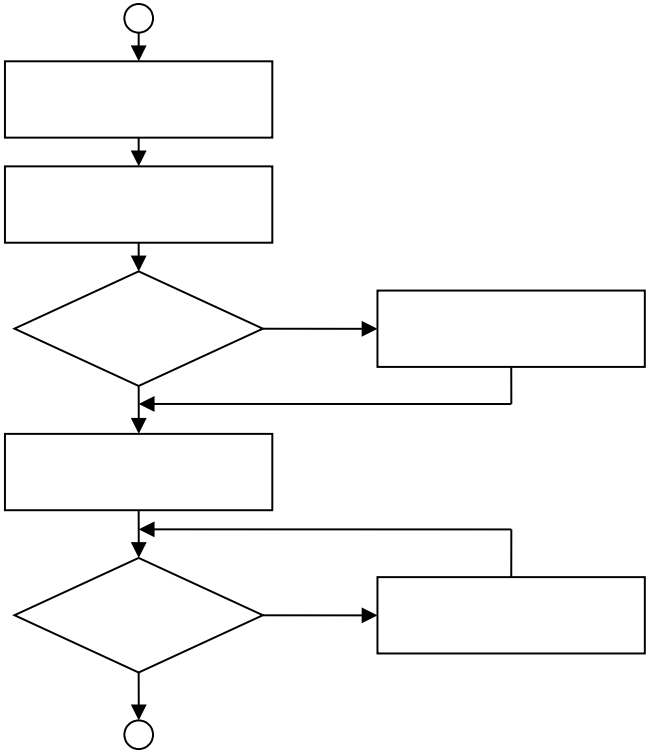
- Programming: code to implement an algorithm
- Scientific programming
  - Custom algorithms for specific problems, often “one off” (but often incorporate well-known algorithms for part of the problem)
  - Aims:
    - get the job done
    - be correct
    - be clear to other scientists
    - be reproducible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

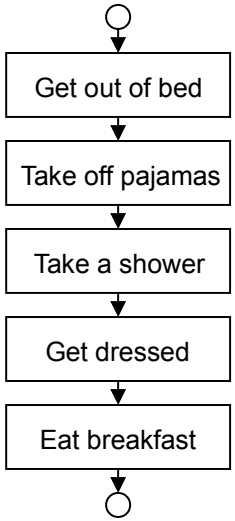
- Imperative programming
  - tell the computer what to do
  - objects can change state (side effects)
- Declarative programming
  - tell the computer what you want
- Functional programming
  - declarative via functions
  - tell the computer what the relationship is
  - functions transform objects to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

Structured programming

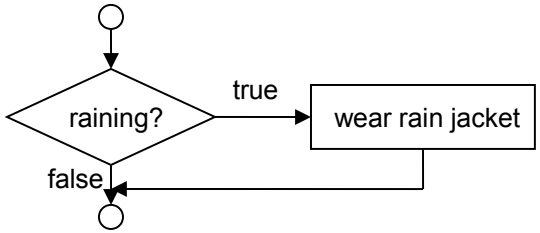
- Most algorithms are expressed in this form
- Algorithm structures determine the order
- Functions encapsulate tasks

Algorithm structures (3)

Sequence

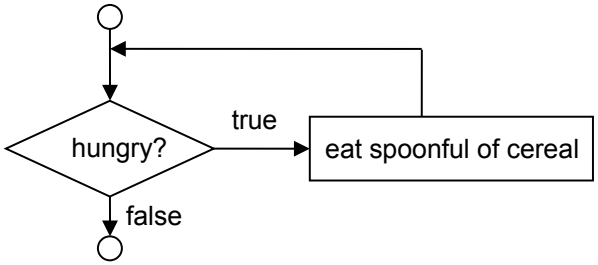


Selection



if

Repetition



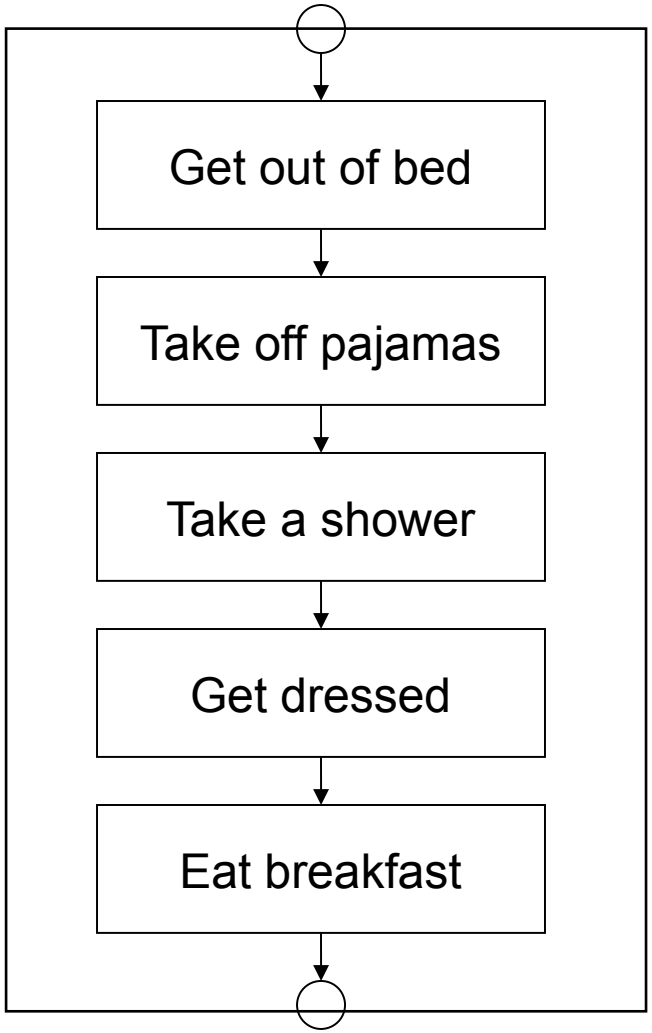
while

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

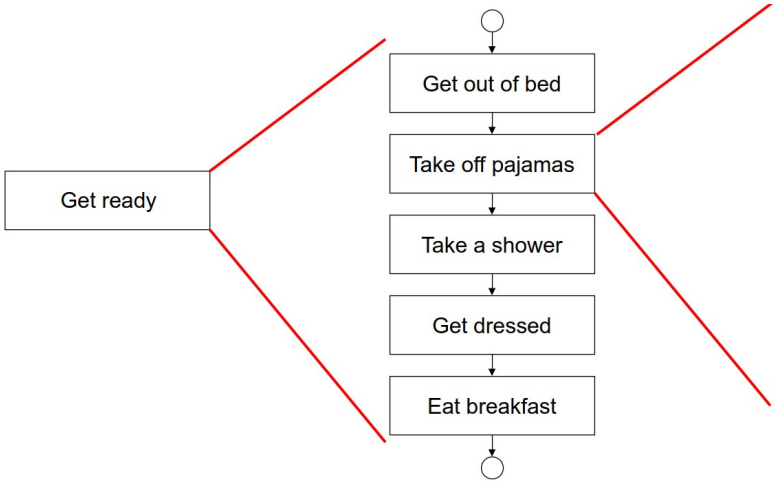
Sequence structure

"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

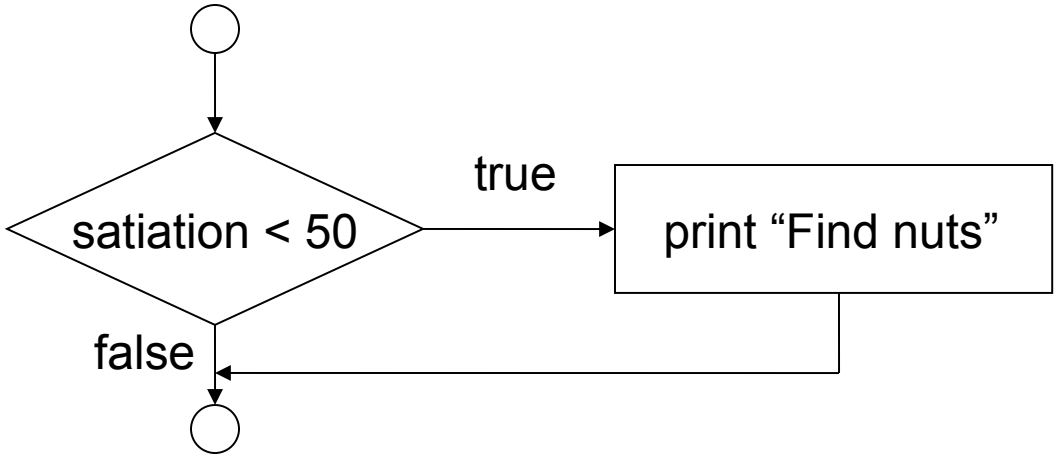
- Decisions: what to do if ...
- Pseudocode:

  If squirrel's satiation is less than 50

  Print “Find nuts”

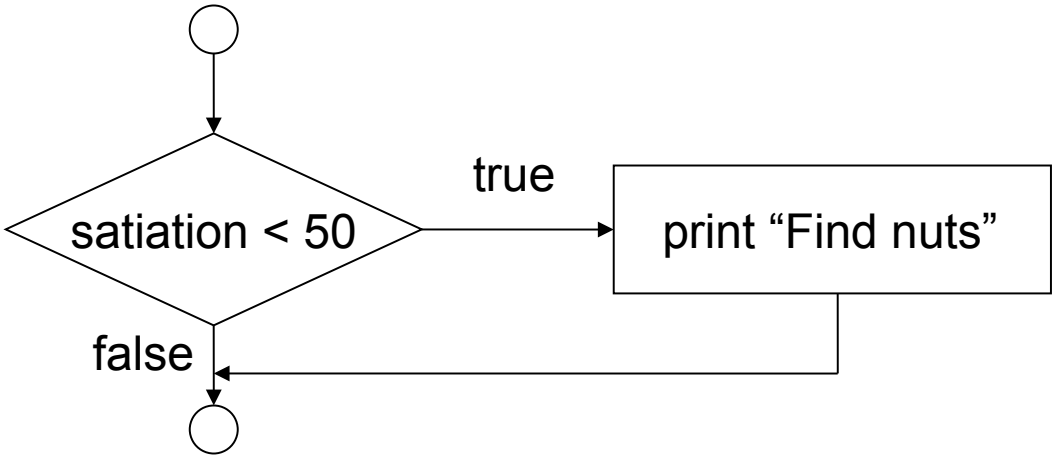
  indent (4 spaces)

- 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

spaces around operators

add spaces for control structures

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

Class style

indent (4 spaces)

closing brace aligns with "i" in "if"

R

Variety of styles

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

Tidyverse style

indent 2 spaces

no space

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

Another style

brace on new line

R

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

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

colon

4 space indent: official python style
indents define control structures

Py

Python's if selection structure

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

colon

4 space indent: official python style
indents define control structures

Py

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
```

R

Example patterns

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

R

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

Py

Example patterns

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

R

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

Py

Example patterns

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

R

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

Py