R programming basics: control structures

Ecological Systems Modeling

Jan 15-19, 2024

Active participation (optional)

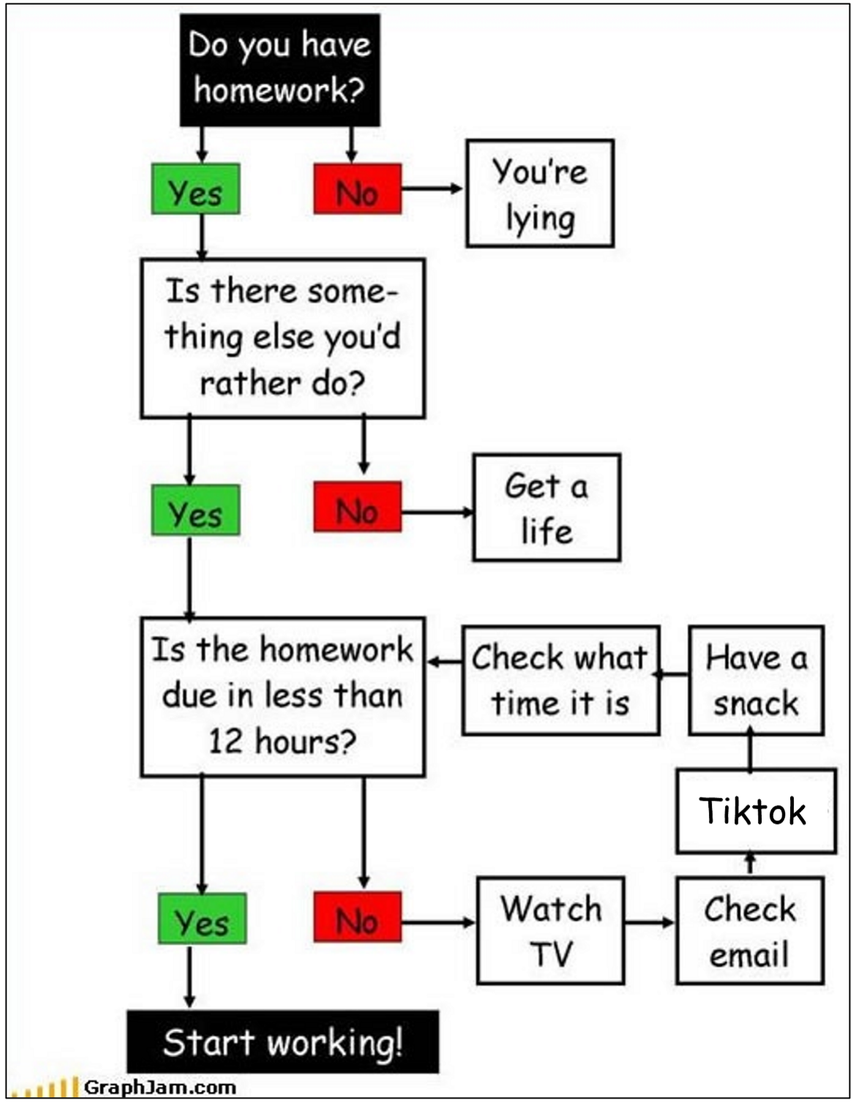
- Open RStudio in Jupyter Hub
- In the Files/Plots/etc. pane, navigate to: `$HOME/Labs/Intro_R_part2/`
- Click on `Intro_to_R_part2c_controlStructures.rmd`
- File should open in the Source pane
- Run the code chunks, add to chunks, or type code in Console

Learning objectives

- Understand the purpose of control structures in R
- Provide examples of conditional statements and loops
- Describe what is meant by “nested” in context of control structures

Control structures

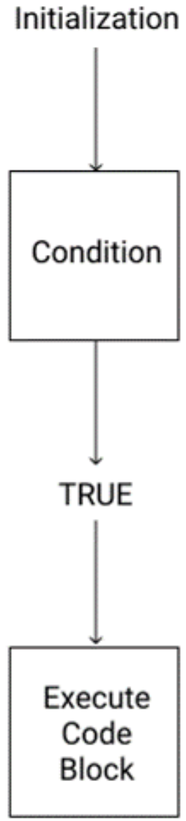
- **if-else** statements and **loops**
  - Decision based on Boolean condition(s)
  - If `TRUE`, execute some code
  - If `FALSE`, do nothing or execute some other code



Conditional statements: `if` statement

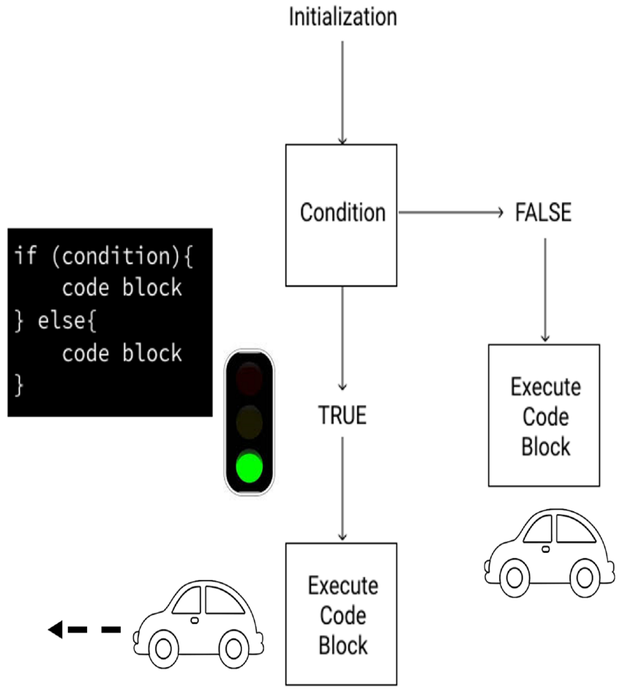
- `if`: single expression with corresponding instruction
  - If it's cold: Turn up the heater
  - Code run if condition evaluates to `TRUE`

```
if (condition){
    code block
}
```



Conditional statements: `if else` statement

- `if else`: corresponding and alternative `else` instruction
  - If the traffic light is green: Drive
  - Else: Don’t drive
- Different code blocks run when `TRUE` vs.`FALSE`



`if` vs. `if else` statement

This doesn’t print anything

```
# If statement
if (0 > 1) {
  print(TRUE)
}
```

This prints `FALSE`

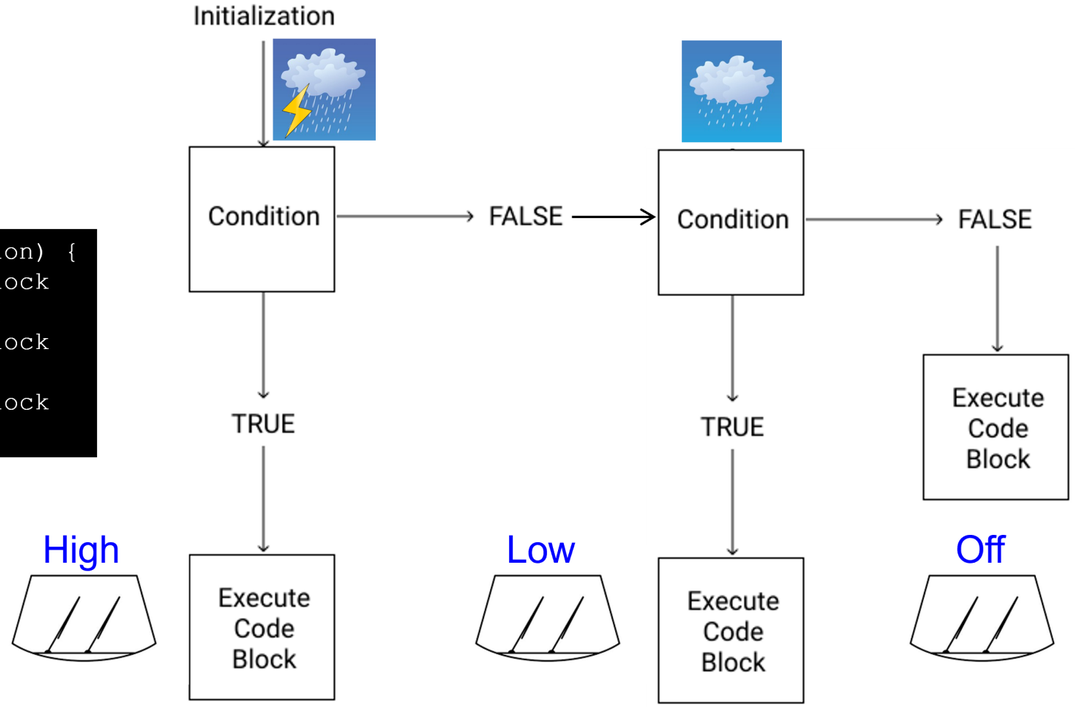
```
# If-else statement
if (0 > 1) {
  print(TRUE)
} else {
  print(FALSE)
}
```

```
[1] FALSE
```

Multiple `if else` statements

- Multiple expressions with corresponding instructions
  - `if` raining hard: wipers on high speed
  - `else` if raining lightly: wipers on low speed
  - `else`: wipers off

```
if (condition) {
    code block
} if else {
    code block
} else {
    code block
}
```



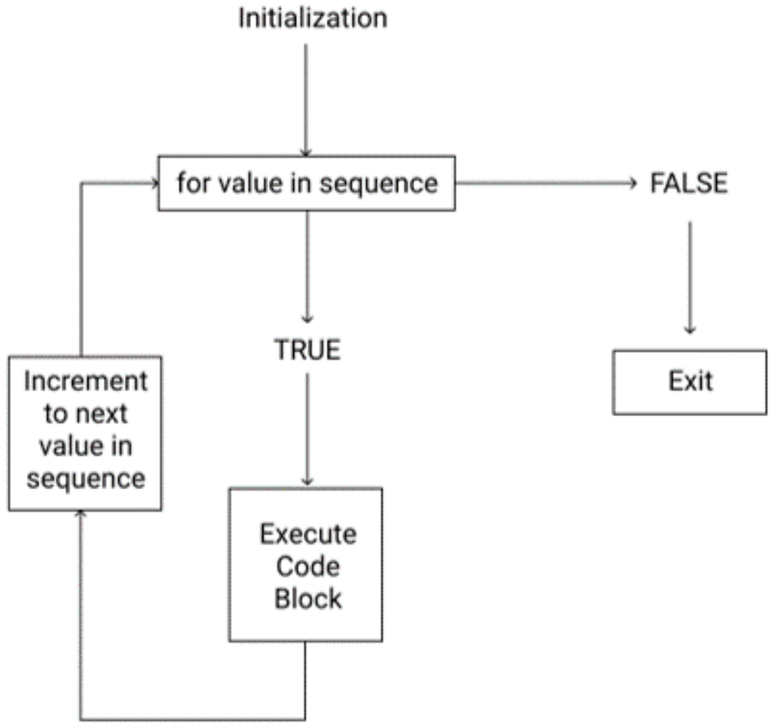
Loops in R

- Purpose: repeat a set of operations several times
- How: Execute instructions a specified number of times or until a specific condition is met
- Benefits: allows us to write less code (fewer mistakes, more succinct)
- Three types: `for` loop, `while` loop, `repeat` loop (not covered)

The `for` loop

- Iterates over a sequence
  - Code block repeated multiple times for each value
  - Loop ends when there are no more values in sequence
  - i.e., value is `FALSE`

```
for (value in sequence){
     code block
}
```



The `for` loop

- For loops can be used on multiple types of objects
- For example, loop through a sequence of numbers
  - Create object that increases with each loop (e.g., `i`)
  - Loop will stop once last value is reached

```r
# i = index
# 1:8 = values 1 through 8
# For loop
for (i in 1:8) {
  print(i) # print the value
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
```

The `for` loop: iterate over rows

- Loop through rows of a data frame, e.g.,
  - `nrow()` returns the number of rows
  - `1:nrow()` is sequence of values (1 to total no. of rows)

```r
# A data frame
df <- data.frame(
  x = c("A", "B", "C"), y = c(1:3))
df
```

```
  x y
1 A 1
2 B 2
3 C 3
```

```r
# Return number of rows in df
nrow(df)
```

```
[1] 3
```

```r
# Sequence (1 to total no. of rows)
1:nrow(df)
```

```
[1] 1 2 3
```

The `for` loop: iterate over rows

- Loop through rows of a data frame, e.g.,
  - `nrow()` returns the number of rows
  - `1:nrow()` is sequence of 1 through last row
  - `[i,]` used for indexing rows

```r
# A data frame
df <- data.frame(
  x = c("A", "B", "C"), y = c(1:3))
df
```

```
  x y
1 A 1
2 B 2
3 C 3
```

```r
# Loop through data frame rows
for (i in 1:nrow(df)) {
  print(df[i,])
}
```

```
  x y
1 A 1
  x y
2 B 2
  x y
3 C 3
```

The `for` loop: iterate over columns

- Loop through columns of a data frame, e.g.,
  - `ncol()` returns the number of columns
  - `1:ncol()` is sequence of 1 through last column
  - `[,i]` used for indexing columns

```r
# A data frame
df <- data.frame(
  x = c("A", "B", "C"), y = c(1:3))
df
```

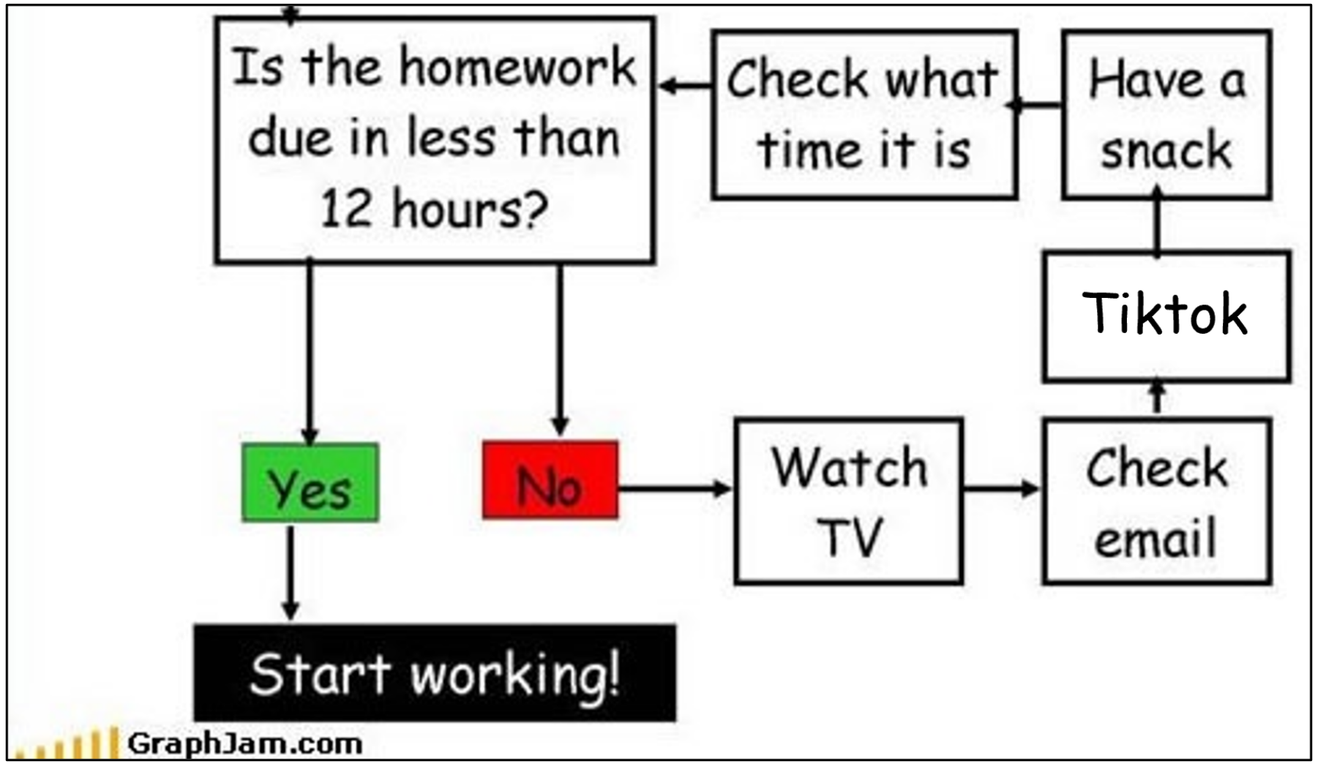
```
  x y
1 A 1
2 B 2
3 C 3
```

```r
# Loop through data frame rows
for (i in 1:ncol(df)) {
  print(df[,i])
}
```

```
[1] "A" "B" "C"
[1] 1 2 3
```

The `while` loop

- `while` loops used less frequently
- Keep looping until a specific logical condition is satisfied



The `while` loop

- Example
  - Create a vector: `doy` equal to `1`
  - Iteratively add `1` to `doy`
  - R exits loop when `doy <= 10` is `FALSE`

```
1  # First day of year (doy)
2  doy <- 1
3
4  # While loop
5  while (doy <= 10) {
6    print(doy)
7    doy <- doy + 1
8  }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Combining control structures

- Loops can be nested
  - Two or more occur in a code block
  - e.g., loop through `years` and `days` in a year

```
# Years and days
years <- 2022:2023
days <- 1:3

# Loop through each year in years
for (year in years) {
  print(year)
  # Next loop through each day
  for (day in days) {
    # paste() concatenates strings
    msg <- paste("Day", day, "in", year
    print(msg)
  }
}
```

```
[1] 2022
[1] "Day 1 in 2022"
[1] "Day 2 in 2022"
[1] "Day 3 in 2022"
[1] 2023
[1] "Day 1 in 2023"
[1] "Day 2 in 2023"
[1] "Day 3 in 2023"
```

Combining control structures

- If-else statements and loops can be combined
  - e.g., operation in a loop is conditional

```r
# Years and days
years <- 2022:2023
days <- 1:3

# Loop through each year in `years`
for (year in years) {
  if (year %% 2 == 0) {
    # Next loop through each day
    for (day in days) {
      # The "paste()" function concatenates strings
      msg <- paste("Day", day, "in", year)
      print(msg)
    }
  }
```

```
[1] "Day 1 in 2022"
[1] "Day 2 in 2022"
[1] "Day 3 in 2022"
```