

R programming basics: control structures

Ecological Systems Modeling

Jan 13-17, 2025

Active participation (optional)

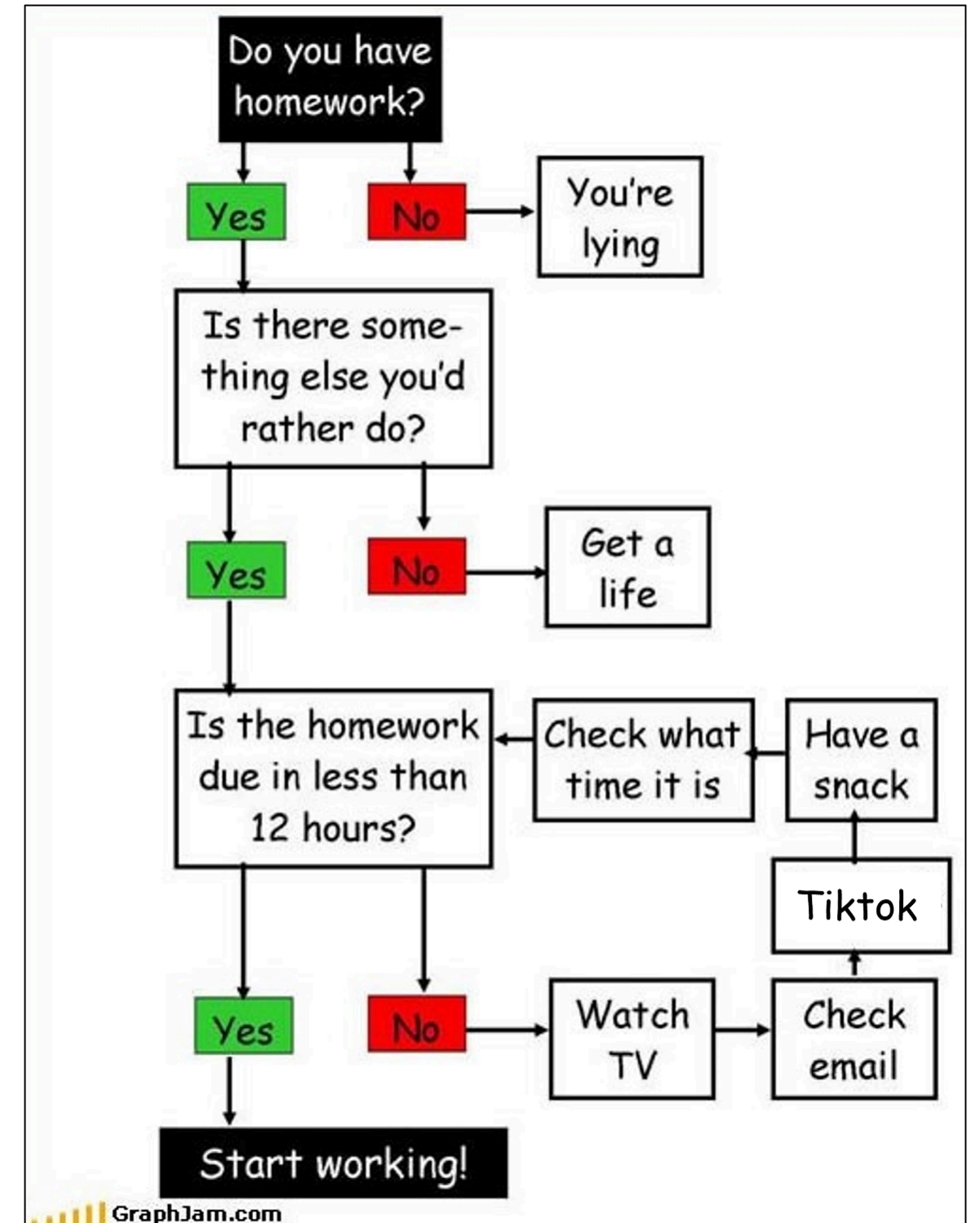
- Open RStudio in Jupyter Hub
- In the Files/Plots/etc. pane, navigate to: `$HOME/Labs/Intro_R_part2/`
- Click on `Intro_to_R_part2c_controlStructures.rmd`
- File should open in the Source pane
- Run the code chunks, add to chunks, or type code in Console

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

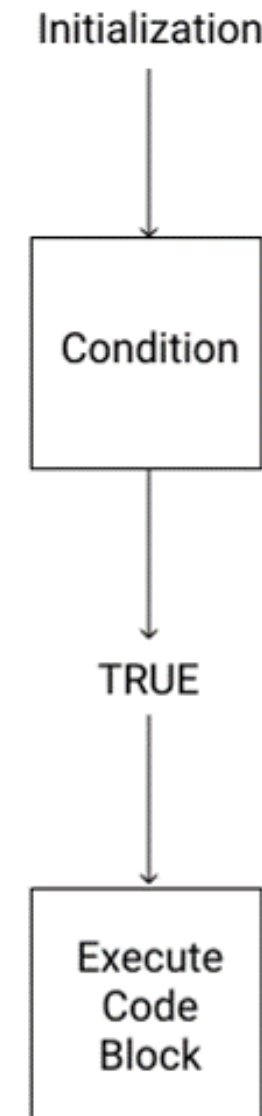
- **if-else** statements and **loops**
 - Decision based on Boolean condition(s)
 - If **TRUE**, execute some code
 - If **FALSE**, do nothing or execute some other code



Conditional statements: **if** statement

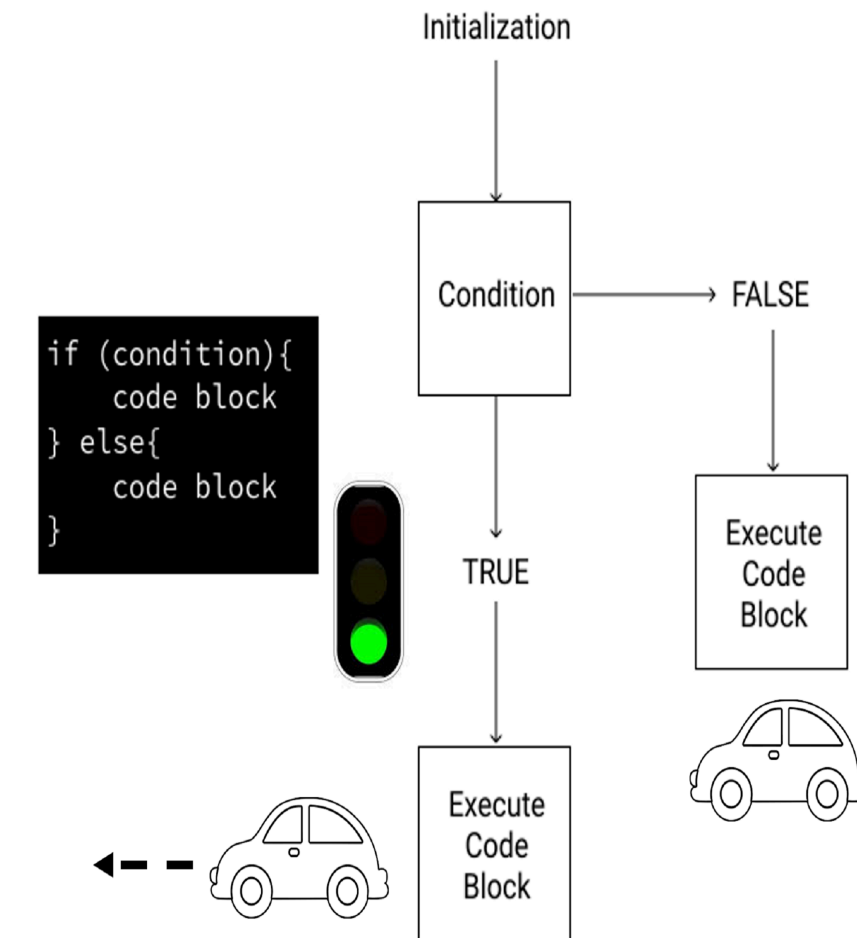
- **if**: single expression with corresponding instruction
 - If it's cold: Turn up the heater
 - Code run if condition evaluates to **TRUE**

```
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

```
1 # If statement
2 if (0 > 1) {
3     print(TRUE)
4 }
```

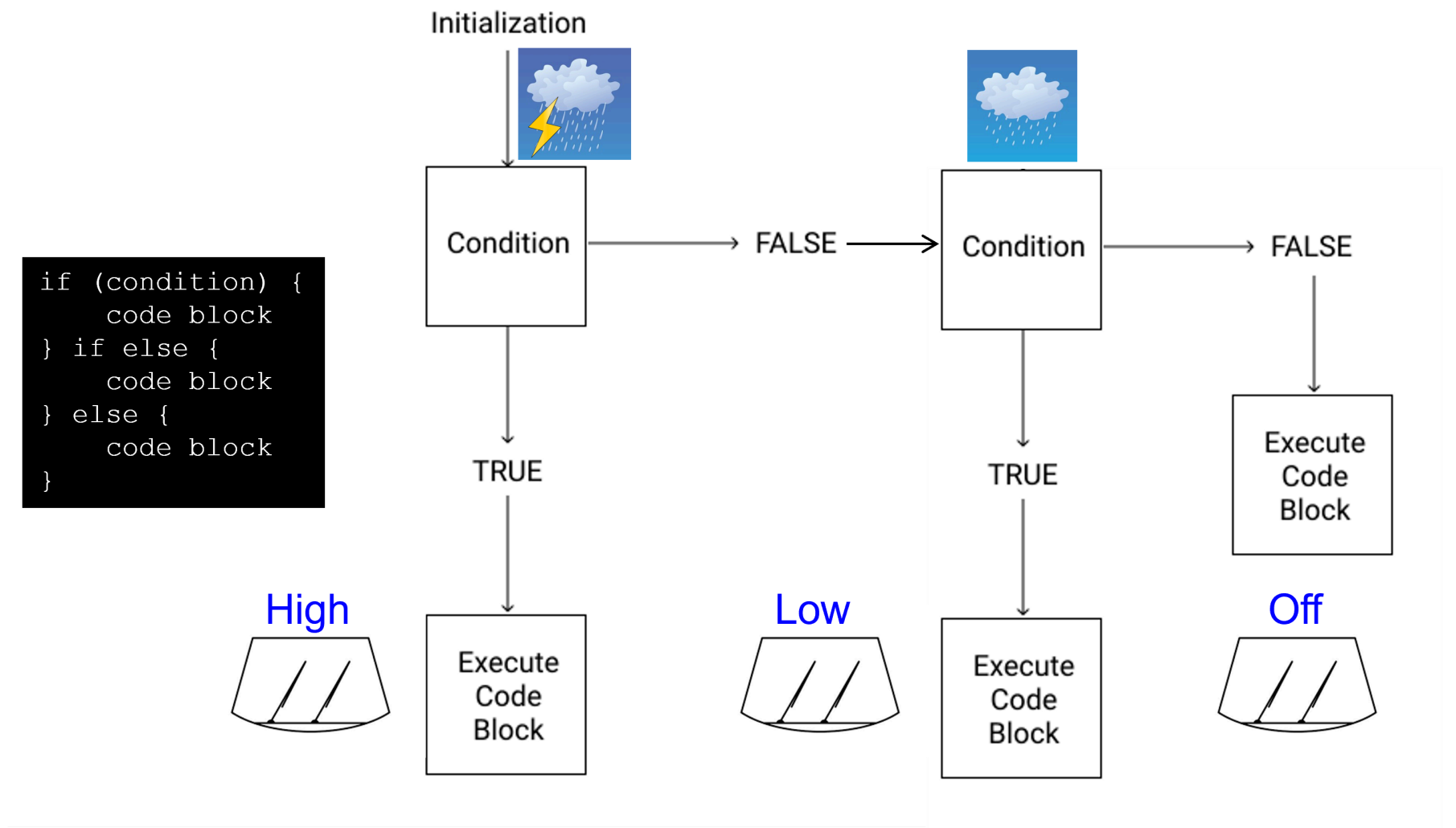
This prints **FALSE**

```
1 # If-else statement
2 if (0 > 1) {
3     print(TRUE)
4 } else {
5     print(FALSE)
6 }
```

```
[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



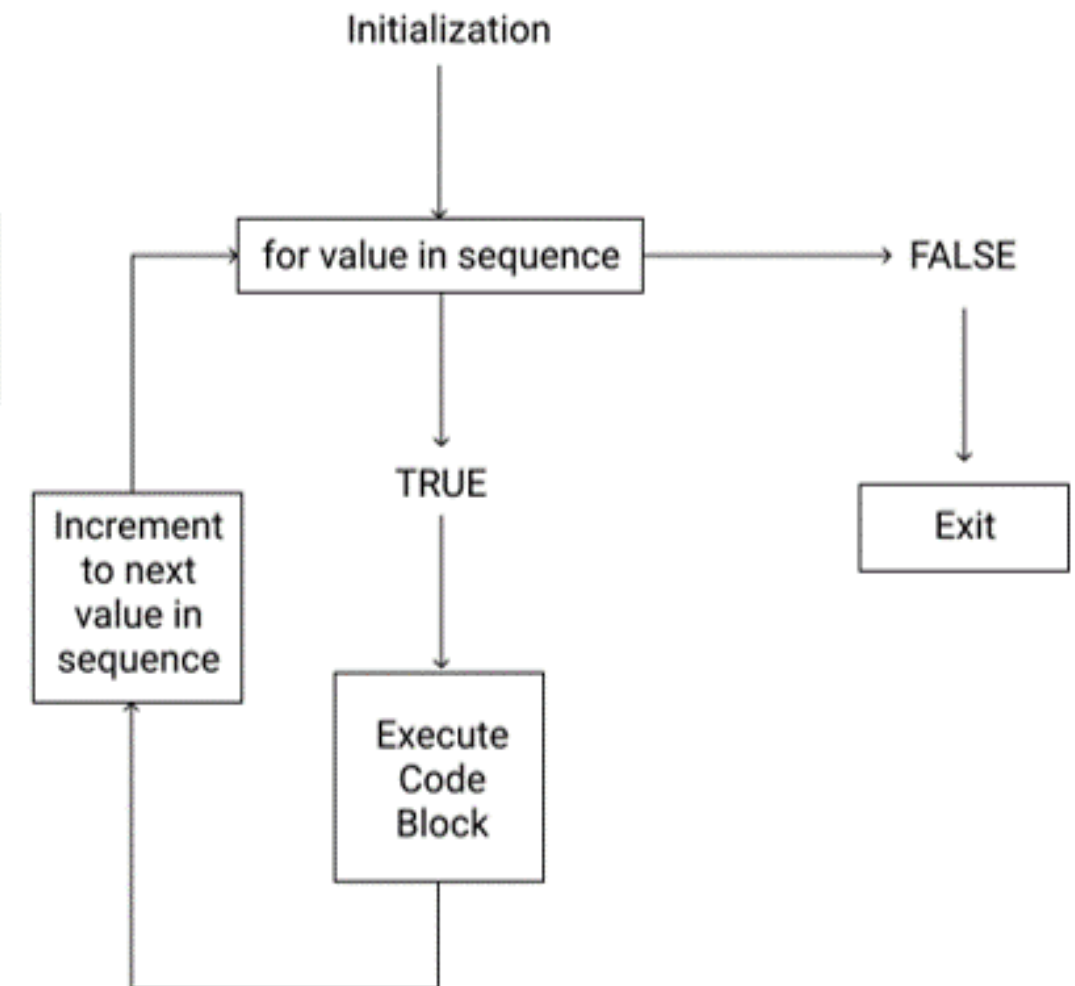
Loops in R

- 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

```
1 # i = index
2 # 1:8 = values 1 through 8
3 # For loop
4 for (i in 1:8) {
5     print(i) # print the value
6 }
```

```
[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)

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

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

```
1 # Return number of rows in df
2 nrow(df)
```

```
[1] 3
```

```
1 # Sequence (1 to total no. of rows)
2 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

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

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

```
1 # Loop through data frame rows
2 for (i in 1:nrow(df)) {
3   print(df[i,])
4 }
```

```
  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

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

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

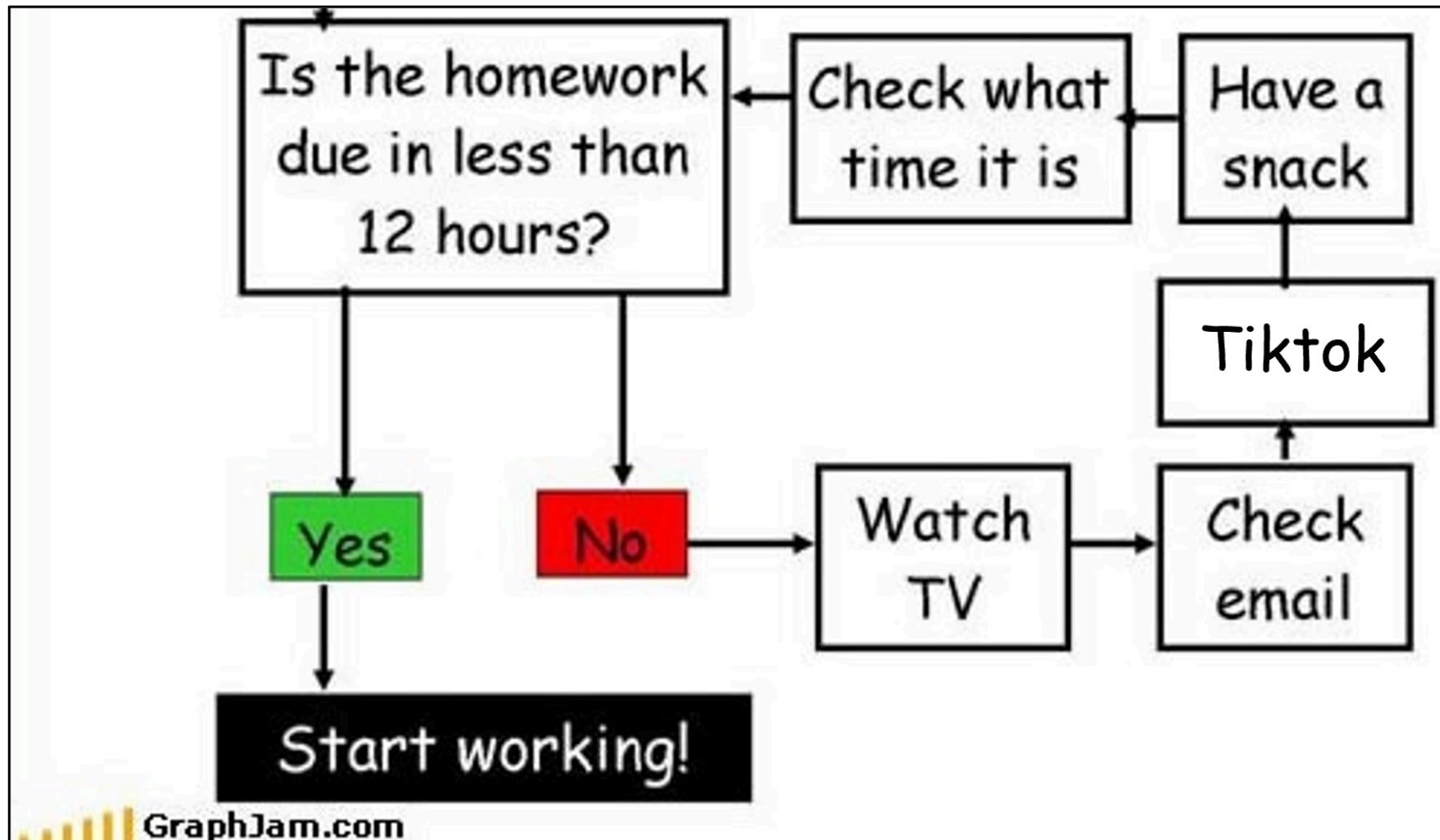
```
1 # Loop through data frame rows
2 for (i in 1:ncol(df)) {
3   print(df[,i])
4 }
```

```
[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

```
1 # Years and days
2 years <- 2022:2023
3 days <- 1:3
4
5 # Loop through each year in years
6 for (year in years) {
7   print(year)
8   # Next loop through each day
9   for (day in days) {
10     # paste() concatenates strings
11     msg <- paste("Day", day, "in", year
12     print(msg)
13   }
14 }
```

```
[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

```
1 # Years and days
2 years <- 2022:2023
3 days <- 1:3
4
5 # Loop through each year in `years`
6 for (year in years) {
7   if (year %% 2 == 0) {
8     # Next loop through each day
9     for (day in days) {
10      # The "paste()" function concatenates strings
11      msg <- paste("Day", day, "in", year)
12      print(msg)
13    }
14  }
```

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

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

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

