Control Flow

Lecture 3

Objectives

- To use logical operators
- To write conditional statements
- To call functions

Motivation

	patient_ID	sex	age_year	***is <2 y.o.?***
1	P001	female	1	✔️
2	P002	female	4	❌
3	P003	female	2	❌
4	P004	male	3	❌
5	P005	male	4	❌

In R code

```
> age_year <− c(1, 4, 2, 3, 4)
> age_year < 2
```

[1] TRUE FALSE FALSE FALSE FALSE

Logical vectors

- A logical vector has elements with values **TRUE** or **FALSE**
- **NA** (not available), a logical value for missing value

```
> sex <− c("female", "female", "female", "male", "male")
> sex == "female"
```

```
[1]  TRUE  TRUE  TRUE FALSE FALSE
```

Relational and logical operators

- Use to compare two things or evaluation a vector or an object

Operator	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x | y	x OR y
x & y	x AND y

Equality ==

```
> 4 == 4
```

[1] TRUE

```
> 1 == 2
```

[1] FALSE

```
> TRUE == TRUE
```

[1] TRUE

```
> "R" == "r"
```

[1] FALSE

Inequality !=

```
> 4 != 4
```

[1] FALSE

```
> 1 != 2
```

[1] TRUE

```
> TRUE != TRUE
```

[1] FALSE

```
> "R" != "r"
```

[1] TRUE

Greater than >
Less than <

```
> 4 > 2
```

[1] TRUE

```
# Alphabetical order
>  "Z" > "Y"
```

[1] TRUE

```
> 4 < 2
```

[1] FALSE

```
# TRUE (value = 1)
# FALSE (value = 0)
> TRUE < FALSE
```

[1] FALSE

Greater than or equal to >=
Less than or equal to <=

```
> 4 >= 4
```

[1] TRUE

```
> 3 <= 2
```

[1] FALSE

```
> TRUE >= TRUE
```

[1] TRUE

```
> "Z" <= "Y"
```

[1] FALSE

Logical operators

- AND operator &
- OR operator |
- NOT operator !

AND operator &

```
> TRUE & TRUE
```

[1] TRUE

```
> TRUE & FALSE
```

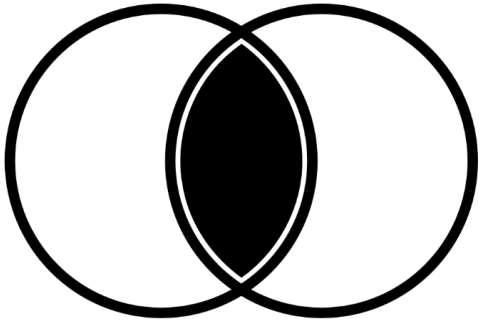
[1] FALSE

```
> a <- 5
> a > 2 & a < 10
```

[1] TRUE

```
> a > 2 & a < 1
```

[1] FALSE



OR operator |

```
> TRUE | TRUE
```

[1] TRUE

```
> TRUE | FALSE
```

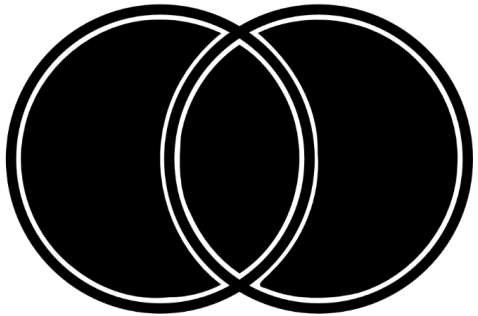
[1] TRUE

```
> b <- 13
> b > 2 | b < 10
```

[1] TRUE

```
> b < 2 | b < 10
```

[1] FALSE



NOT operator !

```
> ! FALSE
```

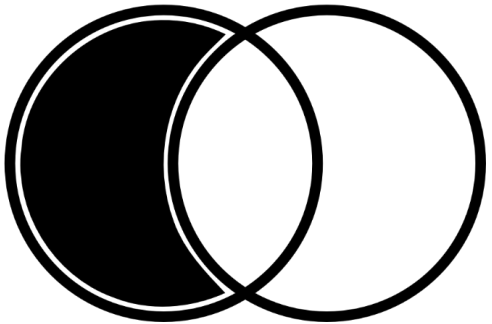
[1] TRUE

```
> y <- 4
> !(y < 1)
```

[1] TRUE

```
> ! (!TRUE)
```

[1] TRUE



Conditional statements

- Control the flow of execution of a series of R commands
- No need to always execute the same R code every time
- Commonly used control structures
  - **if-else** statement: testing a condition and doing something on it
  - **for loop**: execute a loop of commands for a specified number of times

if-else statement

- Test a condition and act on it depending on whether it is true or false

```
if (condition) {
        # do something
}
else {
        # do something else
}
```

if-else statement

```
> sex <− c("female", "female", "female", "male", "male")

# recode female=0 and male=1 using ifelse( ) function
> ifelse(sex=="male", 1, 0)
```

```
[1] 0 0 0 1 1
```

for loops

- takes an iterator variable and assign it successive values from a sequence or vector
- commonly used for iterating over the elements of an object (vector, list)
- useful for doing the same thing on different columns, datasets

```
output <- vector(mode, length(x))          # 1. output
for (i in seq_along(x)) {                  # 2. sequence
      output(i)  <- function(x(i))         # 3. body
}
output
```

for loops

```
x <- c("US", "India", "Brazil", "Russia", "France")

output <- vector(mode="character", length=length(x)) # 1. output
for (i in seq_along(x)) {                            # 2. sequence
      output[i] <- print(output[i])                  # 3. body
}
output
```

```
[1] "US"  "India"  "Brazil" "Russia" "France"
```

Functions

- Allow to automate repeated tasks or lines of codes
- Use a function when copying and pasting chunks of code several times

```
function_name(arg1 = val1, arg2 = val2, …)
> ?function_name
```

Calling functions

> ?seq

seq {base} R Documentation

Sequence Generation
Description
Generate regular sequences. seq is a standard generic with a default method. seq.int is a primitive which can be much faster but has a few restrictions. seq_along and seq_len are very fast primitives for two common cases.

Default S3 method:
seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
    length.out = NULL, along.with = NULL, ...) ...

Arguments
from, to
the starting and (maximal) end values of the sequence. Of length 1 unless just from is supplied as an unnamed argument.

by
number: increment of the sequence.

length.out
desired length of the sequence. A non-negative number, which for seq and seq.int will be rounded up if fractional.

along.with
take the length from the length of this argument.

Calling functions

```
# Create a series of numbers from 1 to 10
> seq(from=1, to=10)          # omitted arguments will use default values
```

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

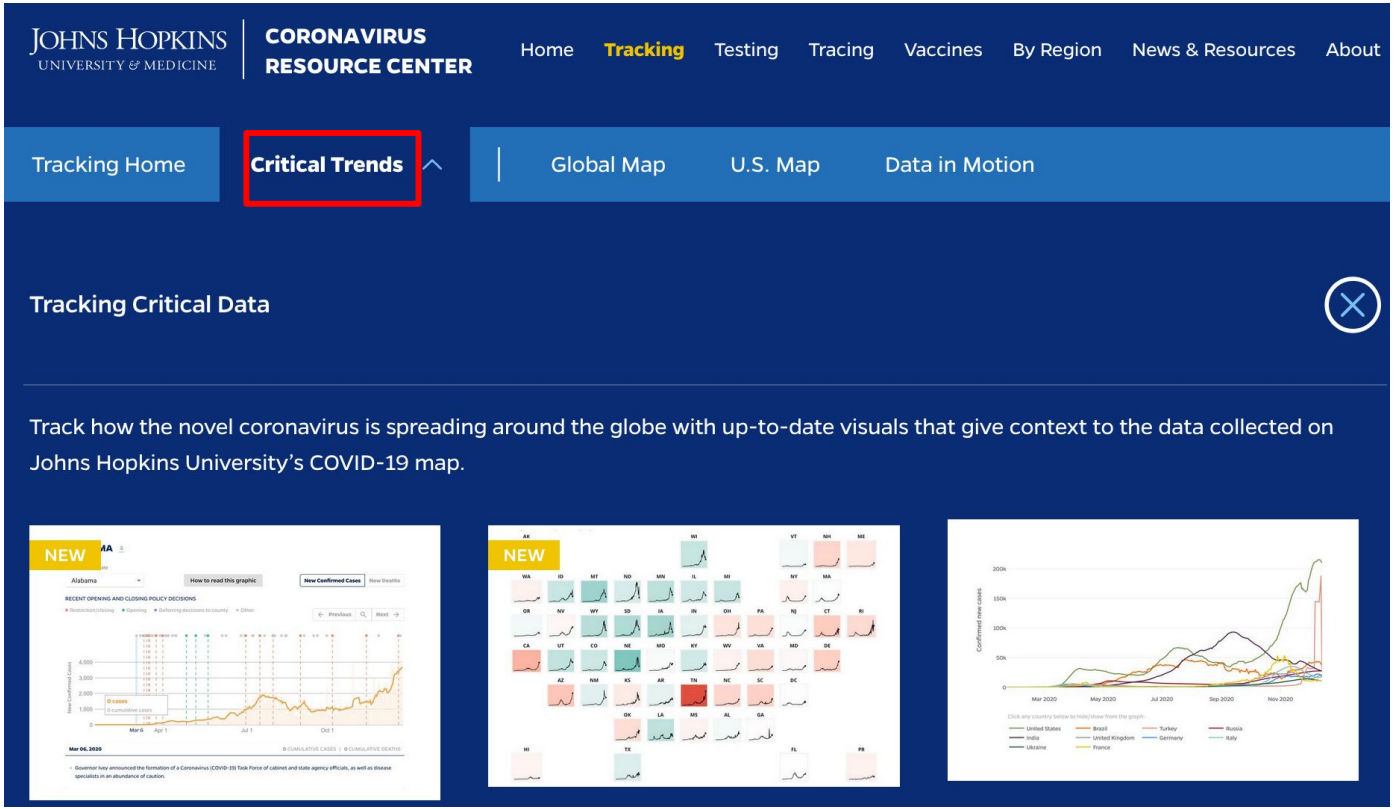
```
# Create a series of odd numbers from 1 to 10
> seq(from=1, to=10, by=2)
```

```
[1] 1 3 5 7 9
```

```
# Argument position matching
> seq(1, 10, 2)
```

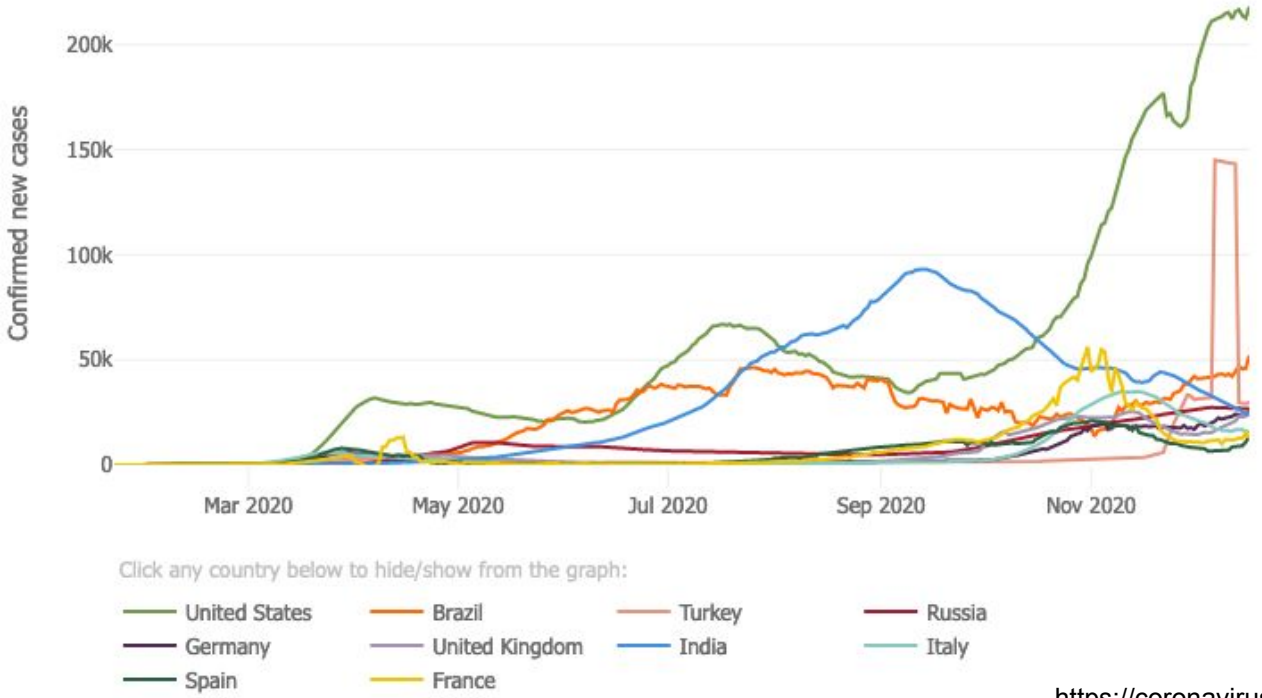
```
[1] 1 3 5 7 9
```

Ideas for Class Project



Ideas for Class Project

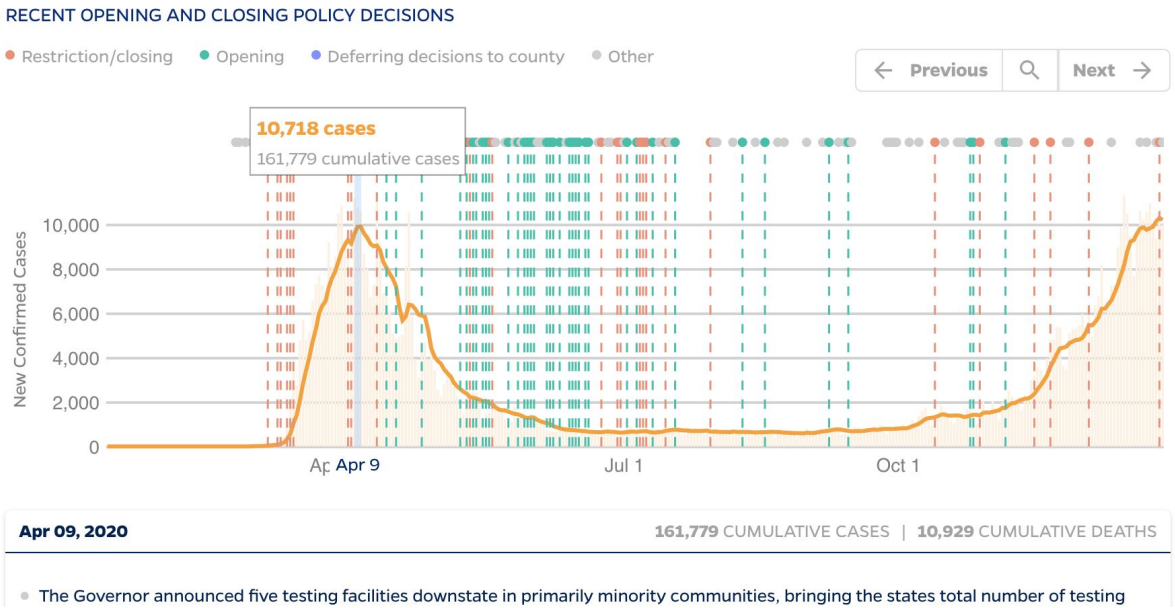
Has your country flatten the curve?



https://coronavirus.jhu.edu/data/new-cases

Ideas for Class Project

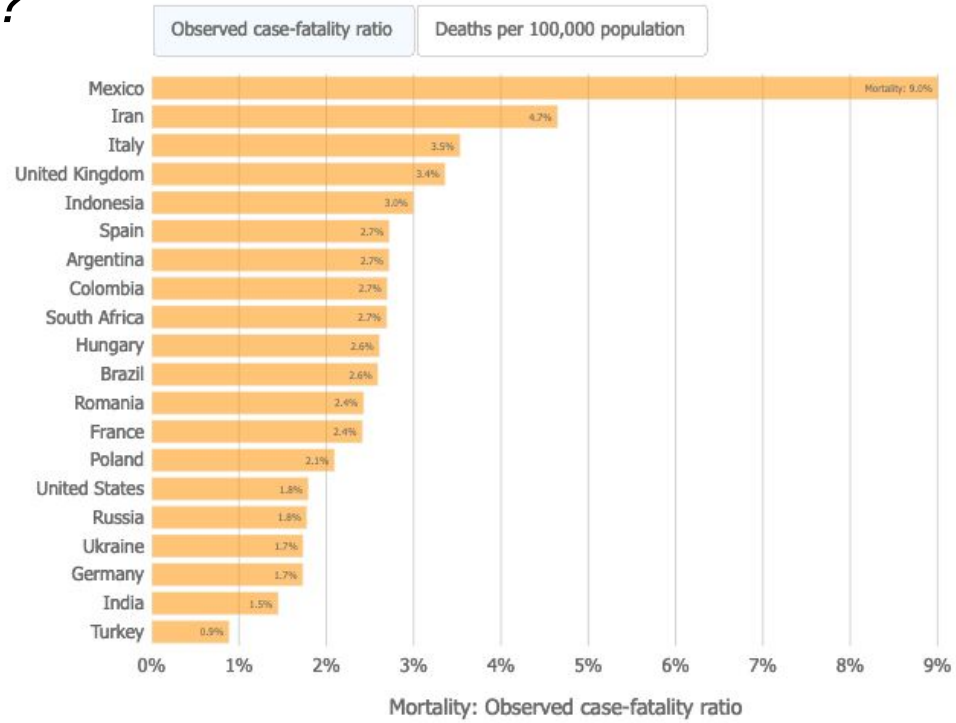
How physical distancing measures affect COVID-19 cases and deaths?



https://coronavirus.jhu.edu/data/state-timeline/new-confirmed-cases/new-york/29

Ideas for Class Project

How is the COVID-19 mortality rate in your country differ from other countries?



https://coronavirus.jhu.edu/data/mortality

Take-away message

- Relational and logical operators are useful when comparing objects
- Conditional statements are essential controls in writing codes
- Use functions to avoid writing lengthy codes