# Methods Camp 2025: Day 2

Christina Pao, Sofia Avila, Florencia Torche

September 5, 2025

# Outline

- [x] Working with Data
- [x] Logistics for Markdown
- [ ] Loops
- [ ] Data Visualization

Any questions so far?

# For Loops: Introduction

- For loops are a critical part of programming.
- They allow you to repeat a block of code multiple times, which is useful for tasks that require iteration.
- Consist of: 1) an iterator, 2) a list of elements and 3) a set of instructions. The loop tells R to execute the instructions for each element in the list.
- Syntax:

```
for(i in 1:3){
  print(i)
}
```

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

# For Loops: Introduction

```
1 for(i in 1:3){
2   print(i)
3 }
```

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

- Here, $i$ is the **iterator**. It loops over each element in `1:3` (`1, 2, 3`) and essentially "becomes" that element.
- E.g. in the first pass of the loop, $i$ becomes 1. Then, we can use $i$ to do something with that element, like print it.

# For Loops: Indexing

- For loops are often used to iterate over a list of elements by their index.
- Recall we can access elements in a vector by their index, e.g. `nums[1]` gives us the first element of the vector `nums`.
- In a for loop, we can use the iterator to access each element in the vector.

```
1  nums = c(10,20,30)
2  for(i in 1:3){
3    curr_num = nums[i]
4    x = curr_num + 2
5    print(x)
6  }
```

```
[1] 12
[1] 22
[1] 32
```

# For Loops vs Math Operations

- When learning for loops, I found it helpful to realize that the logic of the for loop was something I'd already seen in math class with **summation notation**
- For example, suppose we have a list of numbers: $x = \{10, 20, 30\}$ and we want to add 2 to each number in the list and then sum them up.
- In math notation, we could write this as:

$$x = \{10, 20, 30\} \sum_{i=1}^{3} (x_i + 2)$$

In typical math notation, the $x_i$ is the $i$th element of the list $x$. E.g., $x_1 = 10$. Just as with the for loop, $i$ is the iterator, which goes from 1 to 3. So, when $i = 2$, we get $x_2 = 20$.

# For Loops to Program Mathematical Operations

Let's use this to program a very common mathematical operation: the **mean**:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

```
x = c(10,15,20,25,30,35,40,45,50,55,60)
n = length(x) # number of elements in x. safer than hard-coding the number 100.
sum_x = 0
for(i in 1:n){
  sum_x = sum_x + x[i]
}
sum_x / n
```

```
[1] 35
```

```
# confirming same as:
mean(x)
```

```
[1] 35
```

# For Loops to Program Mathematical Operations: Using `seq_along()`

An alternative way to write the previous code is to use `seq_along()` to iterate over the indices of the vector. `seq_along()` creates a sequence from 1 to the length of the vector:

```
z = c(10,20,30)
list1 = 1:length(z)
list2 = seq_along(z)
list1
```

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

```
list2
```

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

# For Loops to Program Mathematical Operations: Using `seq_along()`

```
x = c(10,15,20,25,30,35,40,45,50,55,60)
sum_x = 0
for(i in seq_along(x)){
  sum_x = sum_x + x[i]
}
sum_x / length(x)
```

```
[1] 35
```

# For Loops to Program Mathematical Operations: Loop Over Elements Directly

An even easier way to write the above code is to loop over the elements of the vector directly, rather than their indices:

```
x = c(10,15,20,25,30,35,40,45,50,55,60)
sum_x = 0
for(i in x){
  sum_x = sum_x + i
}
```

Here, instead of $i$ becoming an index of the vector (i.e. a number between 1 and 11), $i$ becomes the actual element of the vector. So, in the first iteration, $i$ becomes 10, then 15, then 20, etc.

# For Loops for Data Analysis

Let's apply for loops to calculate summary statistics for different groups in a dataset. In the `ces` dataset we've been using, we will calculate the average age of respondents by ideology (`ideo` variable) using a for loop.

Below are the general steps to try to follow:

1. Develop the instructions for the loop. Use a single observation to test your code.
2. Initialize a vector to store results- this time it will store an entire vector of results rather than one result. Can either do:
    - Initialize a vector of a certain length: `vec = vector(length = desired length)`
    - Initialize an empty vector: `vec = c()`
3. Use the for statement to tell the loop what to iterate through.
    - Copy and paste the code from the single-observation case into the "meat" part of the loop sandwich

# Use single observation to test code

First, let's see what values the `ideo` variable can take. Two possible ways to do this:

```
ces = read_csv('data/ces.csv')
# two ways to see all the values of ideo
table(ces$ideo)
```

```
     conservative           liberal          moderate very_conservative
              171               196               321               101
     very_liberal
              145
```

```
unique(ces$ideo)
```

```
[1] "very_liberal"      "moderate"          "liberal"
[4] "very_conservative" "conservative"
```

# Use single observation to test code

Let's develop our code using a single element first. Here, we start by calculating average age for `moderates`

```
moderate_age = ces %>%
  filter(ideo == "moderate") %>%
  summarise(mean_age = mean(age, na.rm = TRUE)) %>%
  # "pulls" out value so result is a single number and
  # not a data frame
  pull(mean_age)

moderate_age
```

```
[1] 50.6729
```

# Initialize a vector to store results

Now, let's practice using a vector to store the results.

Here is the syntax we would use if using an empty vector:

```
# empty vector
results = c()
results
```

```
NULL
```

```
results = c(results, moderate_age)
results
```

```
[1] 50.6729
```

# Initialize a vector to store results

Now, let's practice using a vector to store the results.

Here is the syntax we would use if using a vector of a set length:

```
# how long do we want the vector to be?
# since we know we want to calculate the mean age for each of the ideologies,
# we can use the length of the unique values of ideo
num_ideo = length(unique(ces$ideo))
results = vector(length = num_ideo, mode = "numeric")
# now we can assign the value of moderate_age to the first element of results
results[1] = moderate_age
results
```

```
[1] 50.6729  0.0000  0.0000  0.0000  0.0000
```

# Putting it all together

Now, let's put it all together in a for loop:

```
# initialize vector to store results
results = c()
ideo_levels = unique(ces$ideo)
for(curr_ideo in ideo_levels){
  curr_age = ces %>%
    filter(ideo == curr_ideo) %>%
    summarise(mean_age = mean(age, na.rm = TRUE)) %>%
    pull(mean_age)
  results = c(results, curr_age)
}
results
```

```
[1] 43.48276 50.67290 51.81122 54.90099 52.12281
```

# Putting it all together

Let's present the results in a data frame:

```
results_df = data.frame(ideo = ideo_levels, mean_age = results)
results_df %>%
  arrange(mean_age)
```

```
                ideo mean_age
1      very_liberal 43.48276
2          moderate 50.67290
3           liberal 51.81122
4      conservative 52.12281
5 very_conservative 54.90099
```

# For loops for reading and writing files

- For loops can also be used to read and write multiple files.
- I often have large datasets split into multiple files (e.g. for each year, for each state, etc).
- We can use a for loop to read each file!

```
# get list of files
files = list.files(path = "data/ces_by_state", pattern = "*.csv", full.names = TRUE)
files
```

```
[1] "data/ces_by_state/ces_CA.csv" "data/ces_by_state/ces_FL.csv"
[3] "data/ces_by_state/ces_IL.csv" "data/ces_by_state/ces_NY.csv"
[5] "data/ces_by_state/ces_TX.csv"
```

# For loops for reading and writing files

Let's practice what we'd do with one single file:

Open first file in list. We can use `basename` to get the filename without the path (which includes the state name):

```
curr_df = read_csv(files[1])
basename(files[1])
```

```
[1] "ces_CA.csv"
```

Get state name from filename using regular expressions and the `stringr` package:

```
# get state name from filename
state_name = str_extract(basename(files[1]), "[A-Z]{2}")
state_name
```

```
[1] "CA"
```

Now, we can use `assign()` to save the data frame to an object with the state name:

```
assign(paste0("ces_", state_name), curr_df)
```

# For loops for reading and writing files

Now, let's put it all together in a for loop:

```
for(file in files){
  curr_df = read_csv(file)
  # get filename
  state_name = str_extract(basename(file), "[A-Z]{2}")
  # save curr_df to object with state name
  assign(paste0("ces_", state_name), curr_df)
}
```

Now, we can access the data frames by state name!

```
head(ces_IL)
```

```
# A tibble: 6 × 26
   year state county    state_fips county_fips zipcode urban_rural gender4   age
  <dbl> <chr> <chr>          <dbl> <chr>          <dbl> <chr>       <chr>   <dbl>
1  2024 IL    Champaig…         17 019            61820 suburb      woman      27
2  2024 IL    Cook Cou…         17 031            60618 city        woman      62
3  2023 IL    Cook Cou…         17 031            60622 city        man        26
4  2024 IL    Cook Cou…         17 031            60025 suburb      man        81
5  2023 IL    Cook Cou…         17 031            60654 city        woman      28
6  2023 IL    Cook Cou…         17 031           999999 city        man        67
# ℹ 17 more variables: us_citizen <dbl>, emp_status <chr>, own_home <chr>,
#   educ <chr>, has_children <dbl>, past_yr_exp_lost_job <dbl>,
#   vote_pres_2020 <dbl>, pid <chr>, ideo <chr>, daca_support <dbl>,
#   repeal_affordable_care_act <dbl>, easier_conceal_carry <dbl>,
#   grant_legal_status <dbl>, increase_border_patrol <dbl>,
#   hispanic_pop_share <dbl>, total_pop <dbl>, foreign_born_pop_share <dbl>
```

# Bonus: While Loops

- While loops are another type of loop that continue to execute as long as a specified condition is true. Syntax:

```
while(condition){ code to execute while condition is false }
```

- They are useful when you don't know in advance how many iterations you need to perform or if the condition depends on something that happens in the loop.
- E.g. suppose you're a government official with $500,000 to spend on a new program. You want to give $10,000 to every person who is not employed and has children.
- Assuming the CES data represents the population, let's randomly allocate the funds until we run out.

# Bonus: While Loops

Before we start, we initialize a vector to store program money. We set the default value to 0 (since no one has received any money yet). We shuffle the data to randomize the order of the rows and set the total funds to $500,000:

```
set.seed(1)
total_funds = 500000
# initialize vector to store program money, default to 0
program_money = rep(0, nrow(ces))
# shuffle the data to randomize the order of the rows
ces = ces %>% sample_n(nrow(ces))
```

Now, we can start the while loop. Note I'm also using an iterator `i` which I increase by 1 with each loop to get the next row of the data frame.

```
i = 1
while(total_funds > 0){
  curr_row = ces[i, ]
  # check if the person is unemployed or a student and has children
  if(curr_row$emp_status != 'full_time' & curr_row$has_children == 1){
    program_money[i] = 10000
    total_funds = total_funds - 10000
  }
  i = i + 1
}
ces$program_money = program_money
```

# Questions?

# Practice Time!