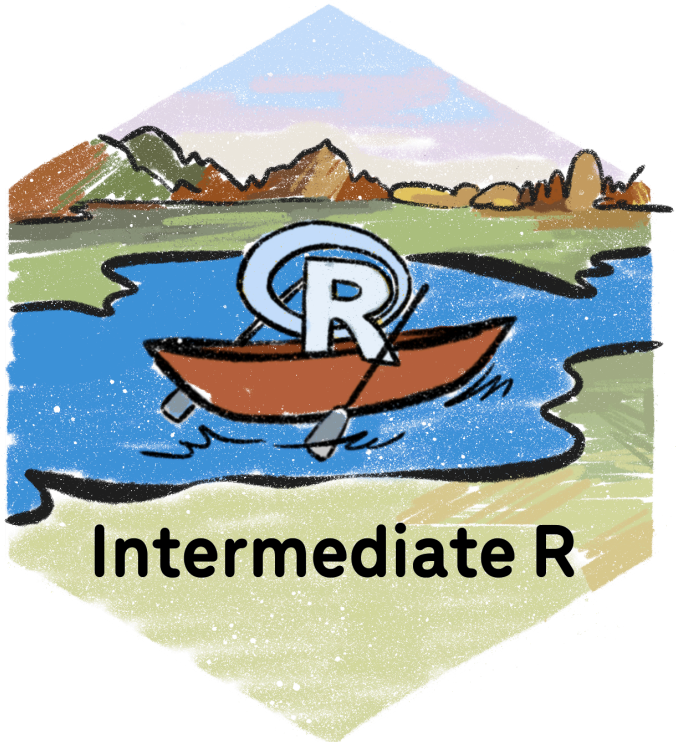
W1: Vectors, data.frames and lists

https://bit.ly/intr_wk1

Remember to Hit Record in Teams

https://bit.ly/intr_wk1

Welcome!




```
Downloading webR package: backports
Downloading webR package: cli
Downloading webR package: generics
Downloading webR package: glue
Downloading webR package: lifecycle
```

https://bit.ly/intr_wk1

Introductions

- Who am I?
- TA: Monica Gerber - in-class resource
- What is DaSL?

https://bit.ly/intr_wk1

Introductions

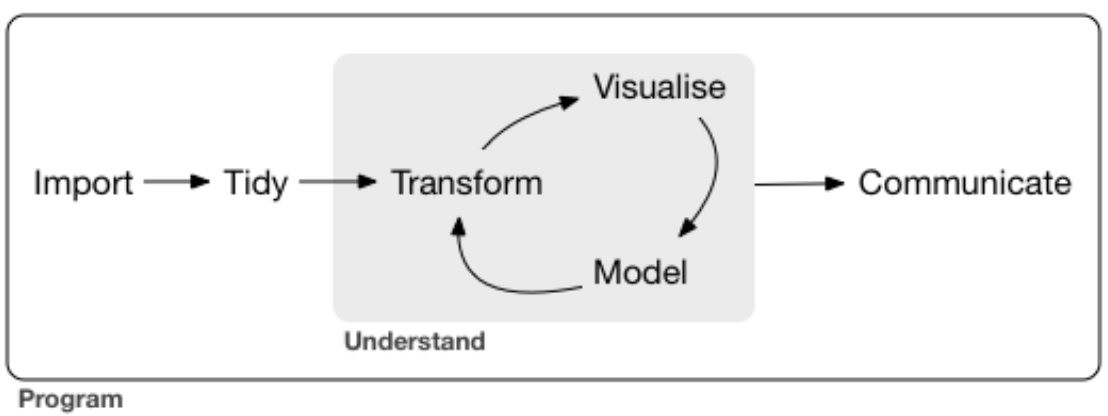
- Who are you? (Share in chat or with your neighbor)
  - Name, pronouns, group you work in
  - What you want to get out of the class
  - Favorite winter activity

https://bit.ly/intr_wk1

03:00

Goals of the course

- **Apply** tools for Tidying data to get a messy dataset into analysis-ready form, via data recoding, data transformations, and data subsetting.
- **Design** and **Create** simple, custom functions that can be reused throughout an analysis on multiple datasets.
- **Explain** and **utilize** *iteration* in programming to reduce repeated code and batch process collections (such as a folder of files or rows in a table)
- At the end of the course, you will be able to: conduct a full analysis in the data science workflow (minus model).



Data science workflow

https://bit.ly/intr_wk1

Culture of the course

- Learning on the job is challenging
  - I will move at learner's pace; we are learning together.
  - Teach not for mastery, but teach for empowerment to learn effectively.
- Various personal goals and applications: curate applications based on your interest!

https://bit.ly/intr_wk1

Culture of the course

- Challenge: We sometimes struggle with our data science in isolation, unaware that someone two doors down from us has gone through the same struggle.
- *We learn and work better with our peers.*
- *Know that if you have a question, other people will have it.*
- *Asking questions is our way of taking care of others.*

We ask you to follow Participation Guidelines and Code of Conduct.

Format of the course

- Wednesdays at 12-1:30 PM
- 6 classes: Jan 22, 29, Feb. 5, 12, 26, Mar 6
- **No class during Public School Week**
- First 20-30 minutes of class is dedicated to catching up (with last week's exercises)
- Streamed online and in person, recordings will be available.
- Announcements via Teams Classroom and by Google Doc
- 1-2 hour exercises after each session are strongly encouraged as they provide practice.
- Optional time to work on exercises together on Fridays 10 - 11 AM PST.
- I will have solution videos available on Monday morning after class (see cheatsheet) . . .
- Online discussion via Teams Space.

Content of the course

Week	Date	Subject
1	Jan 22*	Fundamentals: vectors, data.frames, and lists
2	Jan 29	Data Cleaning 1
3	Feb 5	Data Cleaning 2
4	Feb 12*	Writing Functions
-	Feb 19	No class - school week
5	Feb 26*	Iterating/Repeating Tasks
6	Mar 6*	Overflow/Celebratory Lunch

Schedule/Cheatsheet

*Ted on Campus

Post-Class Survey

- Fill out the post-class survey weekly
- Will discuss weekly
- Your opportunity for feedback/needs

https://bit.ly/intr_wk1

Office Hours

- Opportunity to Practice & ask questions
- 10 - 11 AM PST Fridays
- Outlook link will be shared

https://bit.ly/intr_wk1

Ask me two questions

https://bit.ly/intr_wk1

Break (5 minutes)

A pre-course survey:

https://forms.gle/4ouiHhP8Hbf25L9w5

05:00

Set up Posit Cloud and look at our workspace!

https://bit.ly/intr_wk1

Before we get started

- We'll do in-class exercises live in the slides
  - These slides actually run R on your computer!
- they are mirrored in your workspaces as `classwork`
  - You can do them there if you want to keep a record
- Exercises are in your projects

https://bit.ly/intr_wk1

Exercise Example

Make a vector with the following values: 3, 5, 10. Assign it to an object called `people`. Show the contents of `people`.

Exercise | Start Over | Show Solution | Run Code

```
1 people <- c(--,--,--)
2 people
```

Data types in R

- Numeric: `18`, `-21`, `65`, `1.25`
- Character: `"ATCG"`, `"Whatever"`, `"948-293-0000"`
- Logical: `TRUE`, `FALSE`
- Missing values: `NA`

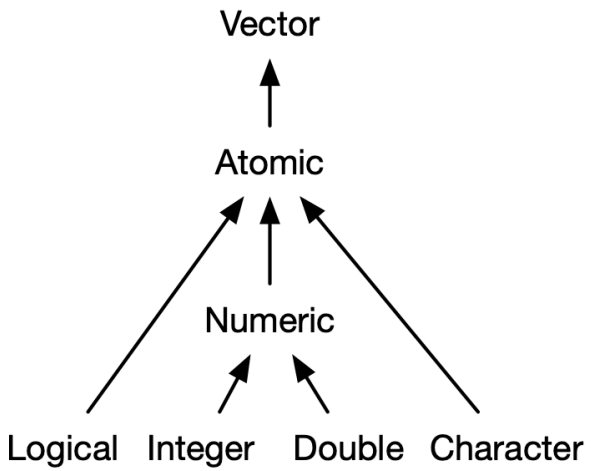
Data structures in R

- `vector`
- `data.frame`
- `list`

https://bit.ly/intr_wk1

Vectors

- A **vector** contains a data type, and all elements must be the same data type. We can have **logical vectors, numerical vectors**, etc.
- Within the Numeric type that we are familiar with, there are more specific types: **Integer vectors** consists of whole number values, and **Double vectors** consists of decimal values




```
1  fib = c(0, 1, 1, NA, 5)
```

Testing for a data type

We can test whether a vector is a certain type with `is.___()` functions, such as `is.character()`.

```
1 is.character(c("hello", "there"))
```

```
[1] TRUE
```

For `NA`, the test will return a vector testing each element, because `NA` can be mixed into other values:

```
1 is.na(c(34, NA))
```

```
[1] FALSE  TRUE
```

Coercing

We can **coerce** vectors from one type to the other with `as.___()` functions, such as `as.numeric()`

```
1 as.numeric(c("23", "45"))
```

```
[1] 23 45
```

```
1 as.numeric(c(TRUE, FALSE))
```

```
[1] 1 0
```

Try it out

What is the output if we use `as.character()` on `c(TRUE, FALSE)`?

R Code | Start Over | Run Code

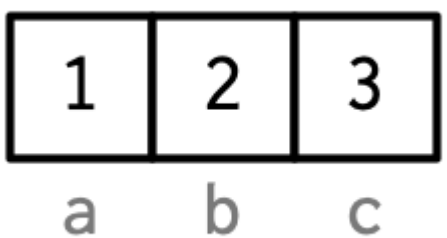
```
1 as.character(c(TRUE, FALSE))
```

Attributes of data structures

It is common to have metadata **attributes**, such as **names**, attached to R data structures.

```r
x = c(1, 2, 3)
names(x) = c("a", "b", "c")
x
```

```
a b c
1 2 3
```




```r
x["a"]
```

```
a
1
```

attributes()

We can look for more general attributes via the `attributes()` function:

```
1 attributes(x)
```

```
$names
[1] "a" "b" "c"
```

Review: explicit subsetting

- We know the indices for our subset, such as “The first two values”

```
1 data = c(2, 4, -1, -3, 2, -1, 10)
```

1. Positive numeric vector

```
1 data[c(1, 2, 7)]
```

```
[1]  2  4 10
```

2. Negative numeric vector performs *exclusion*

```
1 data[c(-1, -2)]
```

```
[1] -1 -3  2 -1 10
```

3. Logical vector

```
1 data[c(TRUE, TRUE, FALSE, FALSE, FALSE, FALSE, TRUE)]
```

```
[1]  2  4 10
```

Review: Implicit subsetting

Comparison operators, such as `>`, `<=`, `==`, `!=`, create logical vectors for subsetting.

```
1 data < 0
```

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

```
1 data[data < 0]
```

```
[1] -1 -3 -1
```

Try it out: Vectors 1 (5 minutes, go as far as you can…)

1. How do you subset the following vector so that it only has positive values?

Exercise | Start Over | Show Solution | Run Code

```
1 data = c(2, 4, -1, -3, 2, -1, 10)
2 data[data -- 0]
```

https://bit.ly/intr_wk1

05:00

Vectors 2

2. How do you subset the following vector so that it has doesn't have the character "temp"?

Exercise | Start Over | Show Solution | Run Code

```
1 chars = c("temp", "object", "temp", "wish", "bumblebee", "temp")
2 chars[chars -- "temp"]
```

Vectors 3

3. Challenge: How do you subset the following vector so that it has no NA values?

Exercise | Start Over | Show Solution | Run Code

```
1 vec_with_NA = c(2, 4, NA, NA, 3, NA)
2 vec_with_NA[!----(vec_with_NA)]
```

`data.frame`

Usually, we load in a `data.frame` from a spreadsheet or a package.

```
1 library(tidyverse)
2 library(palmerpenguins)
3 head(penguins)
```

species <fct>	island <fct>	bill_length_mm <dbl>	bill_depth_mm <dbl>	flipper_length_mm <int>
Adelie	Torgersen	39.1	18.7	181
Adelie	Torgersen	39.5	17.4	186
Adelie	Torgersen	40.3	18.0	195
Adelie	Torgersen	*NA*	*NA*	*NA*
Adelie	Torgersen	36.7	19.3	193
Adelie	Torgersen	39.3	20.6	190

6 rows | 1-5 of 8 columns

`data.frame` attributes

Let's take a look at a `data.frame`'s **attributes**.

```
1 attributes(penguins)
```

```
$class
[1] "tbl_df"     "tbl"        "data.frame"

$row.names
  [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
 [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
 [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
 [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
 [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
 [91]  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
[109] 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
[127] 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
[145] 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162
[163] 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
[181] 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198
[199] 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216
[217] 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234
[235] 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252
[253] 253 254 255 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270
[271] 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288
```

So, we can access the column names of the `data.frame` via `names()` instead of `colnames()`:

```
1 names(penguins)
```

Try it out: Subsetting `data.frames` 1 (5 minutes, go as far as you can)

`

Subset to the single column `bill_length_mm`:

Exercise | Start Over | Show Solution | Run Code

```
1 penguins
```


05:00

Subsetting `data.frames` 2

I want to select columns `bill_length_mm`, `bill_depth_mm`, `species`, and filter the rows so that `species` only has "Gentoo":

Exercise | Start Over | Show Solution | Run Code

```
1 penguins |>
2   select( ) |>
3   filter( )
```

Subsetting `data.frames` 3

Challenge: I want to filter out rows that have `NA`s in the column `bill_length_mm`:

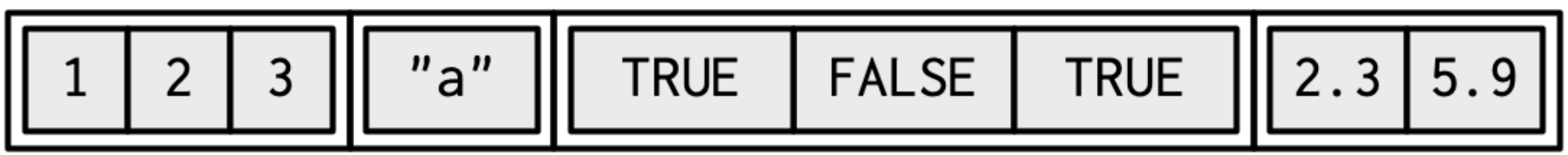
Exercise | Start Over | Show Solution | Run Code

```
1 penguins |>
2   filter(!----(bill_length_mm))
```

Lists

Lists operate similarly as vectors as they group data into one dimension, but each element of a list can be any data type *or data structure*!

```
l1 = list(
  c(1:3),
  "a",
  c(TRUE, FALSE, TRUE),
  c(2.3, 5.9)
)
```



Lists 2

Unlike vectors, you access the elements of a list via the double bracket `[[]]`. (You will access a smaller list with single bracket `[]`.)

```
l1 = list(
  c(1:3),
  "a",
  c(TRUE, FALSE, TRUE),
  c(2.3, 5.9)
)
```

```
l1[[1]]
```

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

List names

We can give **names** to lists:

```
l1 = list(
  ranking = c(1:3),
  name = "a",
  success =  c(TRUE, FALSE, TRUE),
  score = c(2.3, 5.9)
)
#or
names(l1) = c("ranking", "name", "success", "score")
```

Accessing List elements

And access named elements of lists via the `[[]]` or `$` operation:

```
l1[["score"]]
```

```
[1] 2.3 5.9
```

```
# or
l1$score
```

```
[1] 2.3 5.9
```

Therefore, `l1$score` is the same as `l1[[4]]` and is the same as `l1[["score"]]`.

What data structure does this remind you of?

Warning: `[ ]` versus `[ [ ] ]`

This always trips me up, you usually want `[[]]` (return an element) versus `[]` (returns a sublist).

```
1 l1["ranking"]
```

```
$ranking
[1] 1 2 3
```

...

You almost always want to use `[[]]`

```
1 l1[["ranking"]]
```

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

Two main uses for Lists

1. Return a mixed type list of objects, such as from running `lm()` - a lot of methods in R use this.

- Useful when programming functions that need to return multiple objects

2. Store multiple instances of the same data type, such as a `list` of `data.frame`s

- Iteration over these lists is possible

Try it Out: Lists 1

Return the element in the `id` slot:

Exercise | Start Over | Show Solution | Run Code

```
1 person = list(id=100031, age=40)
2 person
```

https://bit.ly/intr_wk1

05:00

Lists 2

Return the 2nd element of this list:

Exercise | Start Over | Show Solution | Run Code

```
1 new_list <- list(c(1,2,3), c(3,4,5), c(5,7,8))
2 new_list
```

Lists 3: Using Variables to Subset

How would you use the value of the `my_col` variable to subset the list?

Exercise | Start Over | Show Solution | Run Code

```
person = list(id=100031, age=40)
my_col <- "age"
person
```

`data.frame`s as Lists

A `data.frame` is just a named list of vectors of same length with **attributes** of (column) `names` and `row.names`, so all of the list methods we looked at above apply.

```
1 head(penguins)
```

species <fct>	island <fct>	bill_length_mm <dbl>	bill_depth_mm <dbl>	flipper_length_mm <int>
Adelie	Torgersen	39.1	18.7	181
Adelie	Torgersen	39.5	17.4	186
Adelie	Torgersen	40.3	18.0	195
Adelie	Torgersen	*NA*	*NA*	*NA*
Adelie	Torgersen	36.7	19.3	193
Adelie	Torgersen	39.3	20.6	190

6 rows | 1-5 of 8 columns

`data.frame`s as Lists

```
1 head(penguins[[3]])
```

```
[1] 39.1 39.5 40.3   NA 36.7 39.3
```

```
1 head(penguins$bill_length_mm)
```

```
[1] 39.1 39.5 40.3   NA 36.7 39.3
```

```
1 head(penguins[["bill_length_mm"]])
```

```
[1] 39.1 39.5 40.3   NA 36.7 39.3
```

Everything in R is a List, or related

https://bit.ly/intr_wk1

Tools for lists

- `lapply()` function - applies a function to each element of a list
- We'll explore in Week 5 the `{purrr}` package, which has methods for working with lists

That’s all!

Office Hours Friday 10 - 11 AM PST to practice together!

https://bit.ly/intr_wk1