R Programming For Natural Resource Professionals

Lecture 7

Iterating I

Discussion of assigned topics

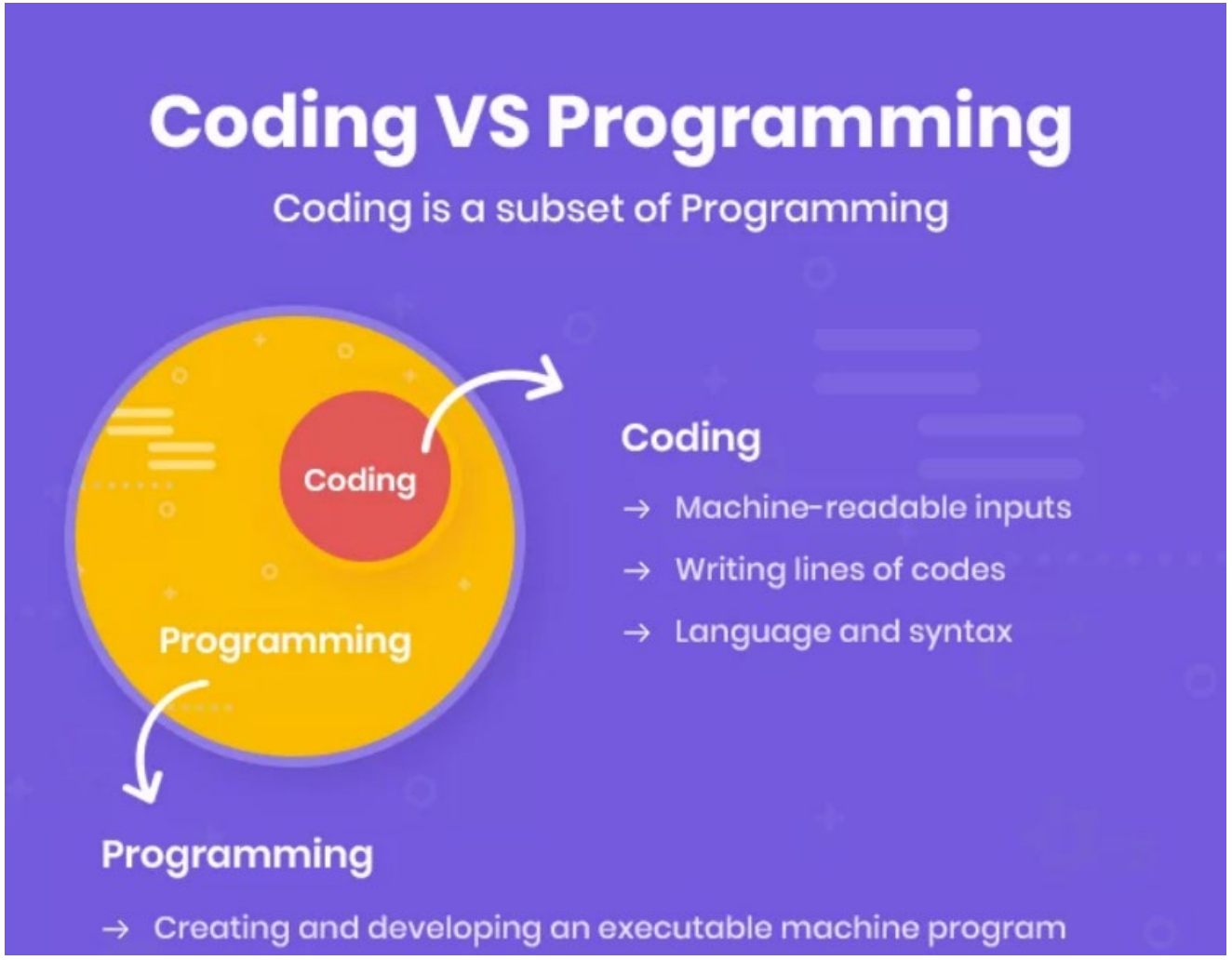
Algorithms: Braden, Jeff, Caden, Sam (Group 1)
Loops: Amanda, Emilia, Roz, Kevin (Group 2)
Functional programming: Ryan B., Ryan E., Max (Group 3)
Object-oriented programming: Jason, Erik, Jeff (Group 4)

Prep to lead class for 5-7 minutes to teach your peers about the assigned topic.

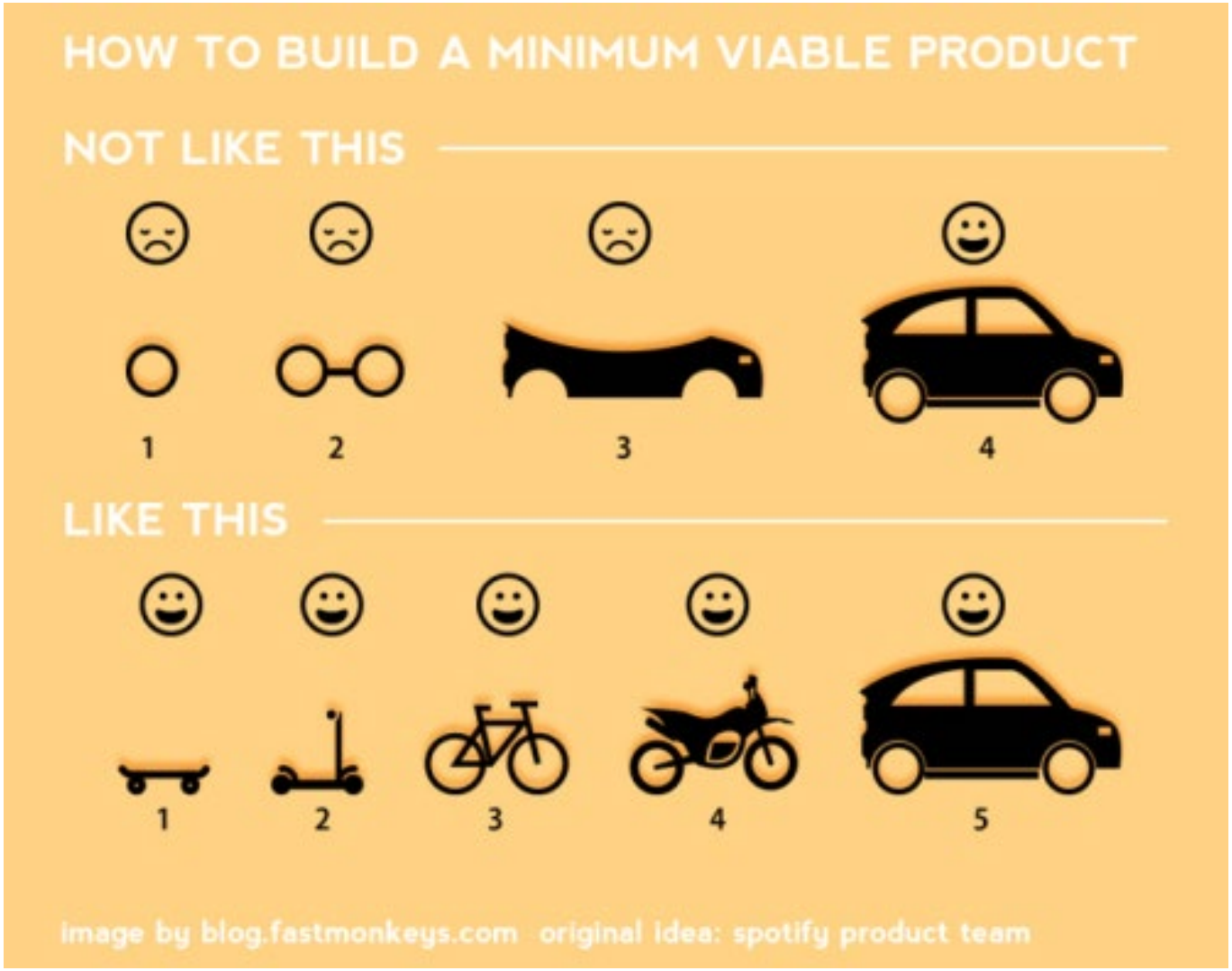
Potential discussion points:
1) Define it.
2) Explain it in an R context.
3) Explain it in a natural resources context.
4) Develop questions to ask the class.

Can email me a slide or two if you'd like (jhomola@uwsp.edu)

Leap from **coding** to **programming**



Programming words of wisdom



Programming words of wisdom

It is faster to make a four-inch mirror then a six-inch mirror than it is to make a six-inch mirror.

-- Programming Pearls, Communications of the ACM, September 1985

Programming words of wisdom

When you must fail,
fail noisily and as soon as possible.

Anatomy of an R function

4 parts of an R function

1) Function name and function() call

```
func_name <- function(inputs) {
operations
output
}
```

Anatomy of an R function

4 parts of an R function

1) Function name and function() call
2) Arguments

```
func_name <- function(inputs) {
operations
output
}
```

Anatomy of an R function

4 parts of an R function

1) Function name and function() call
2) Arguments
3) Body of the function

```
func_name <- function(inputs) {
operations
output
}
```

Anatomy of an R function

4 parts of an R function

1) Function name and function() call
2) Arguments
3) Body of the function

- Describe operations to be performed with the provided arguments
- Body of a function always exists within curly brackets
- Individual operations in the body of the function are separated by new lines

Anatomy of an R function

4 parts of an R function

1) Function name and function() call
2) Arguments
3) Body of the function
4) Outputs

```
func_name <- function(inputs) {
operations
output
}
```

- return() for displaying resulting objects
- print() for displaying simple variables

Let’s practice

```
pow <- function(x, y) {
    result <- x^y
    return(result)
}
```

Translated: given two values, x and y, calculate x to the power of y and return the result

Formal and informal arguments

Formal argument

- Explicitly assigning argument values makes their order interchangeable

Informal argument

- Providing only the argument value assumes values match their order in the function.

```
# Informal argument
pow(2, 10)
[1] 1024
pow(10, 2)
[1] 100
```

```
# Formal argument
pow(x=2, y=10)
[1] 1024
pow(y=10, x=2)
[1] 1024
```

Defining default values in functions

- Default values can be defined by declaring them within the function's input section.
- Note that they can be overruled.

```
pow2 <- function(x, y=2) {
    result <- x^y
    return(result)
}
```

In class exercise 1

- Write a function to calculate the coefficient of variation.
  - Coefficient of variation = standard deviation / mean

- Use the function you wrote to calculate the coefficient of variation for all height measurements in the mock data set.

Conditional functions

Function performs differently depending on specified conditions.

If the condition is true, run the code in the first curly brackets. Otherwise, run the code in the second curly brackets.

```
cond_funct <- function(x) {
      if (conditionTrue){
      #Do something
      } else {
      #Do something else
      }
}
```

Conditional functions

Example: Print a result that says whether the biggest tree in the mock dataset is “really tall” (>= 99th percentile) or not really tall (<99th percentile).

Specifying error conditions

`stopifnot()`

Purpose:
To stop the function if certain conditions are met and provide a generic error message.

Example:

```
func_name <- function(x) {
  stopifnot(is.numeric(x))
}
```

In class exercise 2

Use `stopifnot()` to cause your coefficient of variation function to throw an error if the input isn't numeric

Specifying error conditions

`if()` then `stop()`

Purpose:
To stop the function if certain conditions are met then provide a custom error message.

Example:

```
func_name <- function(x) {
  if(!is.numeric(x)) {
    stop('Hey! This isn't the right data type!')
  }
}
```

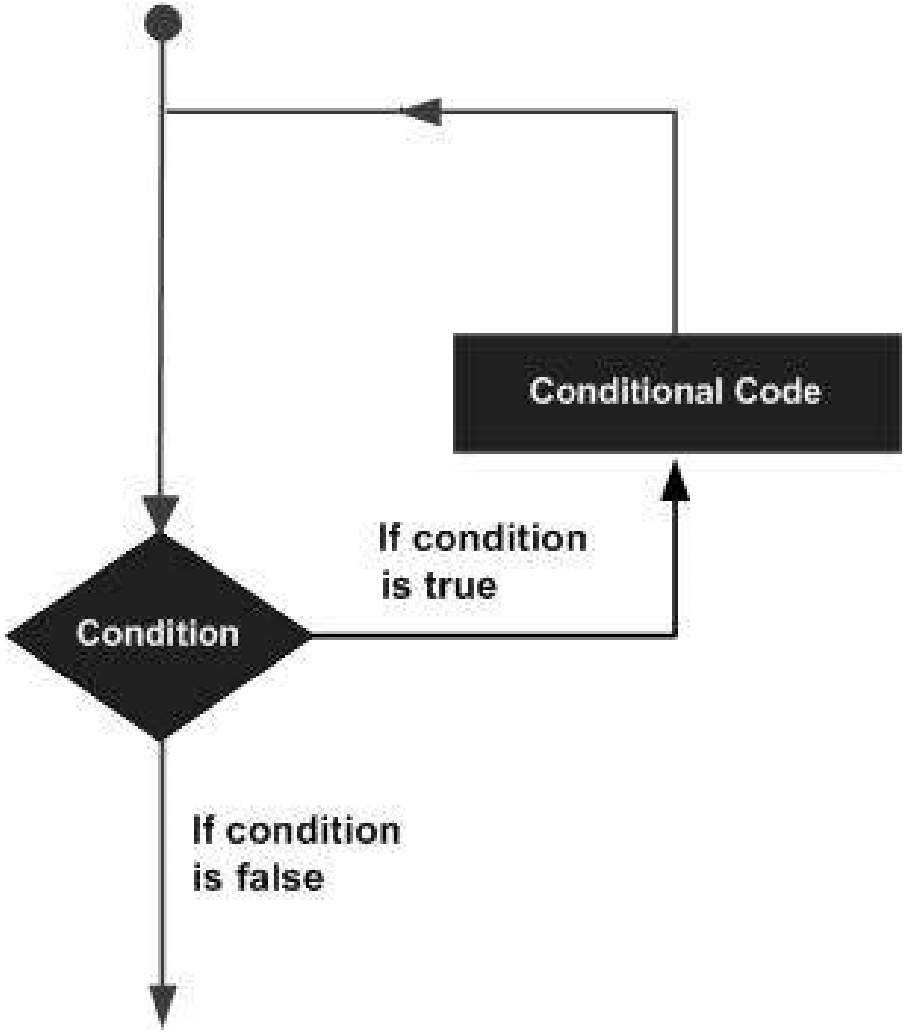
In class exercise 3

Use an `if()` then `stop()` series to cause your coefficient of variation function to throw an error if the input isn't numeric but provide a more useful error message.

Other notes on functions

- If you're copying and pasting a block of code more than once, write a function.
- Break down complex tasks into multiple function. Avoid "megafunctions"
- Writing complex functions isn't easy. It takes time and practice.

Loops



Two common loop types

while loop

Repeats a statement while a given condition is true. It tests the condition before executing the loop body.

for loop

Like a while statement, except that repeats a fixed number of times.

The `for()` loop

Most often used to execute a certain task using a “counter”

```
for (counter in vector_of_values) {
  #Do something.
}
```

The `for()` loop

A simple example

```
for (i in 1:5){
    print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

`i` is the counter, stepping through values from 1:5

Code that depends on `i` is simply printing the value of `i`.

In class exercise 4

Loop through the distinct values in `dat$species` and calculate the coefficient of variation of "height" for each

The `while()` loop

Execute a task repeatedly until a certain condition is met

```
while (conditionTest) {
  #Do something.
}
```

The `while()` loop

A simple example

```
i <- 0
while (i < 4) {
    print(i)
    i <- i+1
}
```

```
[1] 0
[1] 1
[1] 2
[1] 3
```

`i` must be initialized. In many while loops(), the test condition needs initialization.

`i` is then increased in each loop, advancing it closer to meeting the stated `while()` condition.

In class exercise 5

Use a while loop to `sample()` the `dat$height` vector repeatedly until it picks a value above the 97th quantile. Print the randomly selected height value for each loop made.