HEALTH
UNIVERSITY OF UTAH

# Principles

Andrew Redd, PhD.

R Bootcamp 2020

# R Foundation Principles

# R is Open

- R is GNU
- R is open source
  - Source code is almost always available
  - Easy to audit

# R is collaborative

## Packages

Packages are collections of functions put together by other programmers for your use. (You can also publish packages.)

- CRAN (https://cran.r-project.org/) stands for: Comprehensive * BioConductor (http://www.bioconductor.org/)
- GitHub (https://github.com/search/advanced)

# R is a Multi-Paradigm Programming Language

- Imperative & Declarative
- Functional & Object-oriented
- Procedural & Reflective
- Interpreted & Compiled.
- But above all **Data-centric*

# R is a Multi-Paradigm Programming Language

- **Imperative & Declarative**
- **Functional & Object-oriented**
- Procedural & Reflective
- Interpreted & Compiled.
- But above all **Data-centric*

# Imperative vs. Declarative

In **imperative** languages you tell the computer *how* you want something done.

In **declarative** languages you tell the compute *what* you want done.

> *"You can tell me what to do or how to do it but not both"*

# Imperative Languages

*“How to do it”*

## Examples

- C (https://en.wikipedia.org/wiki/The_C_Programming_Language)
- Fortran (https://en.wikipedia.org/wiki/Fortran)
- Assembly (https://en.wikipedia.org/wiki/Assembly_language)

## R

```
x <- 1:100
total = 0
for (i in x)
    total <- total + i
```

Your total is 5050.

# Declarative Languages

*"What to do"*

## Examples

- SQL
  - Give me a data set that contains everyone born in or after the year 2000.
- SAS
  - Perform a `PROC FREQ` to give me a count of birth years.

## R

```
filter(data, dob >= "2000-01-01")
count(data, year(dob))
```

# Functional vs. Object

**Functional** paradigm centers around functions and connecting them together; output from one is input to another. Output depends only on the input, i.e. idempotent.

- May not **ever** change the inputs to a function.
- Minimize side effects and state changes.

**Object** oriented programming centers around objects and having objects perform actions.

- Centers around state changes and side effects.
- An argument to a function often is changed by the function.

# Functional Languages

- Never changes the inputs
- you want something changed, explicitly say so

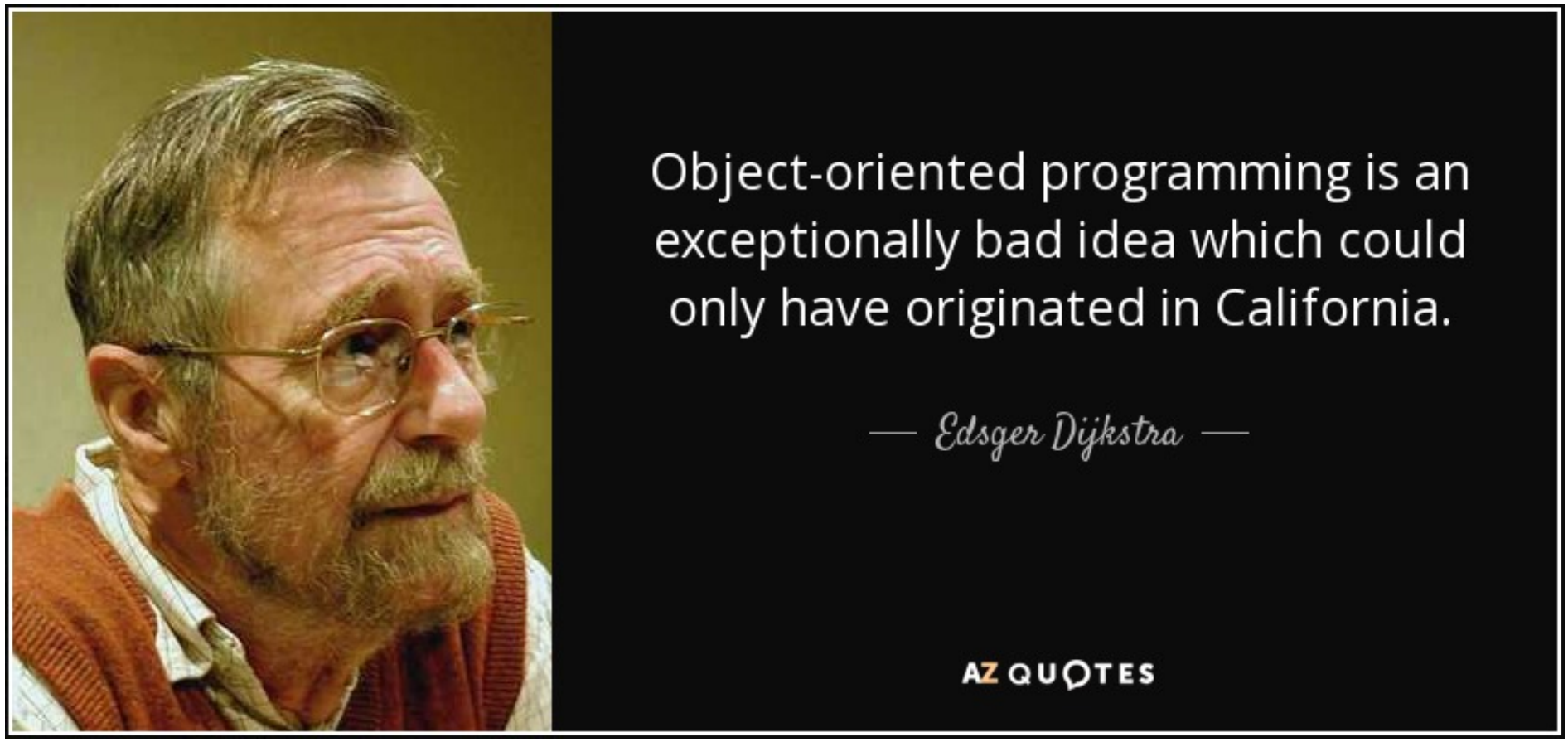
## Examples

- Lisp (https://en.wikipedia.org/wiki/Common_Lisp)
- JavaScript (https://en.wikipedia.org/wiki/JavaScript)

## R

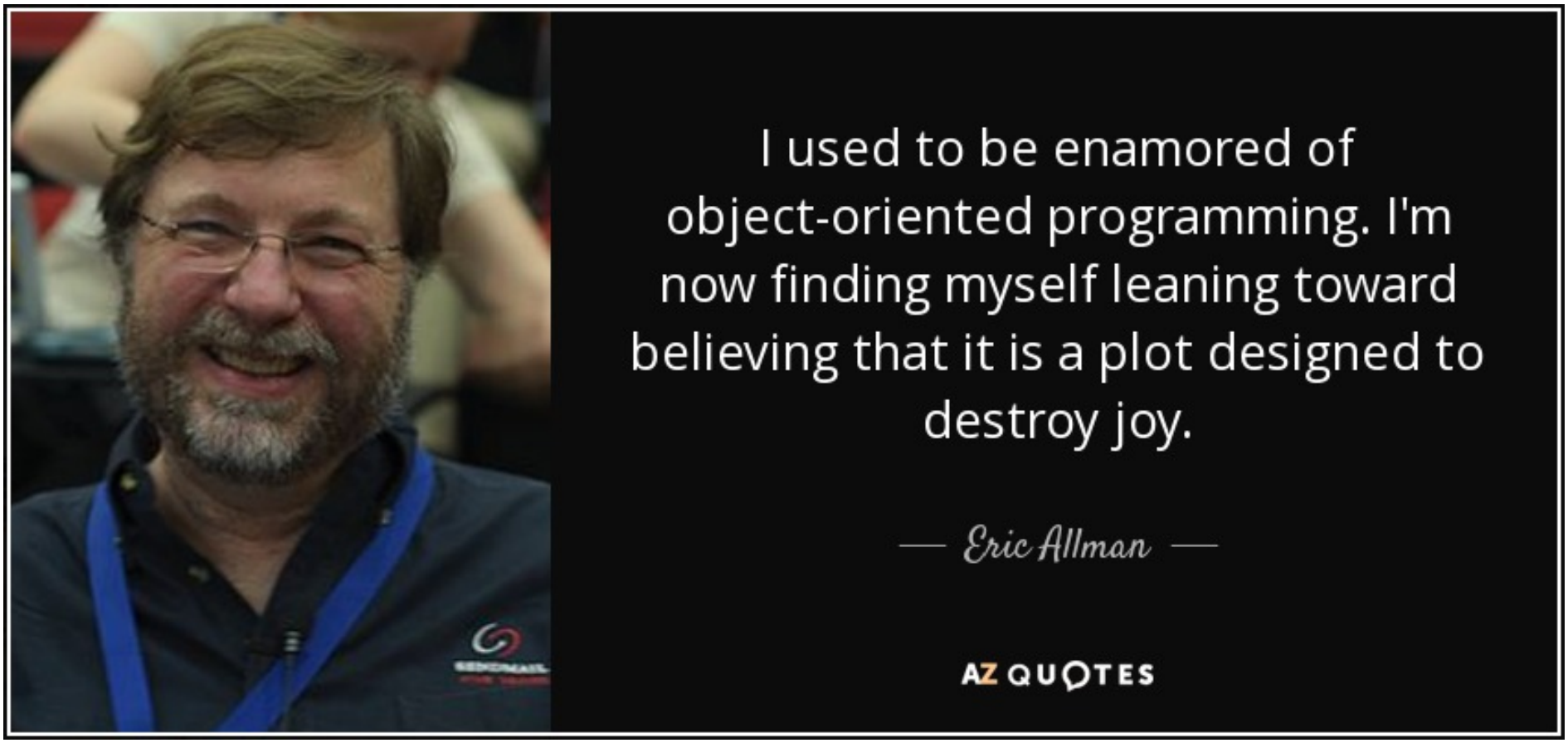
```
filter(iris, Species == "setosa")          #< Does nothing
iris <- filter(iris, Species == "setosa") #< Actually filters
```

# Object Oriented Programming




(https://www.azquotes.com/quote/78525)

# Object Oriented Programming




(https://www.azquotes.com/quote/932334)

So what?

# Why this maters if you are coming from SAS

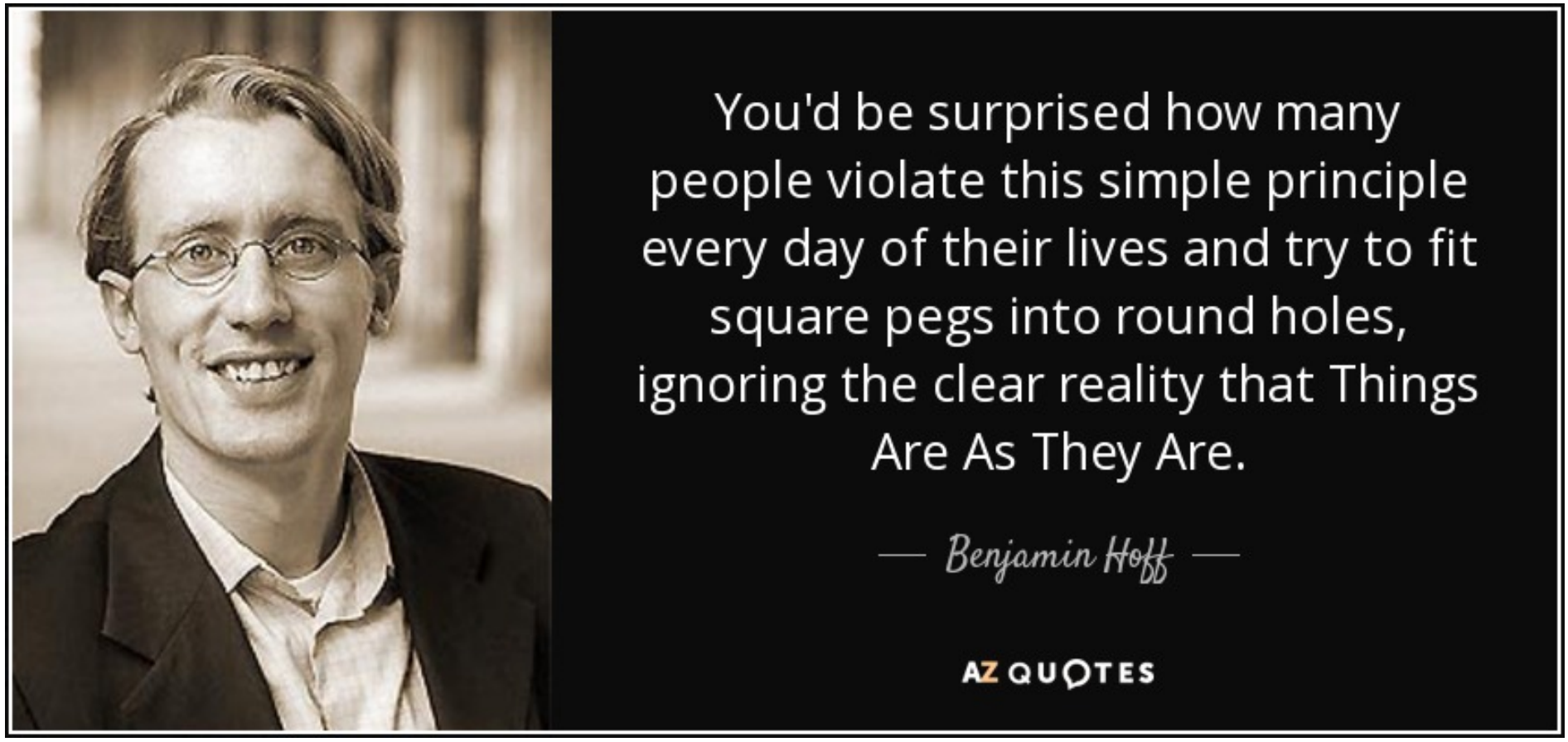
- R is **VERY** different than SAS.
- You will have to step back from the Data step/Proc paradigm
- More procedures and structures less declarations.

# Why this maters if you are coming from STATA

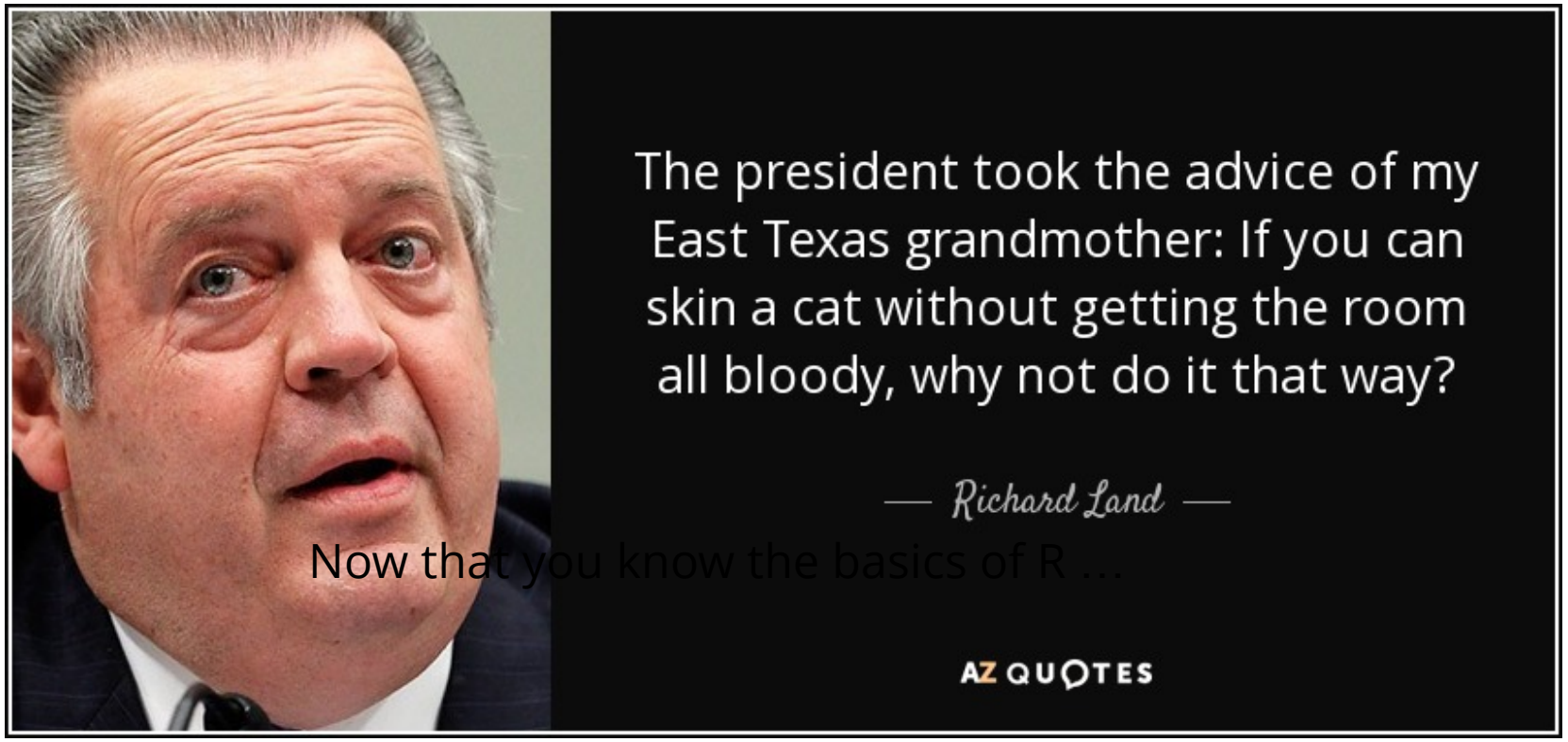
- Data is an object in R
- There is no ‘active’ data set.

# Why this maters to everybody

- Knowing how a language is designed makes you more effective with it.




(https://www.azquotes.com/quote/395721)




Now that you know the basics of R ...

(https://www.azquotes.com/quote/906668)

# How I teach R

# How I teach R

- The way I use R
  - Programmer time is most important.
  - Simplest is very often the fastest computationaly as well
  - Reuse of code.
- Data focused
  - results not details
- Base R
  - Because you need to know this to be effective.
- Tidyverse
  - Because this is better.

# In this class R is

- Functional,
- Imperative, and
- Declarative.