Lecture 27: Nov 26, 2018

S3 Programming

- *OOP*
- *S3 Objects*
- *Unpaired (Two Sample) t-Test*

James Balamuta
STAT 385 @ UIUC

Announcements

- **Group Project Final Report, Demo Video,** and **Peer Evaluation** due **Tuesday, December 18th** at **10:00 PM**
  - Details: http://stat385.stat.illinois.edu/group-projects/
- **hw09** due **Friday, November 30th** at **6:00 PM**
- **Quiz 13** covers Week 12 contents @ CBTF.
  - Window: Nov 27th - Nov 29th
  - Sign up: https://cbtf.engr.illinois.edu/sched
- Want to review your homework or quiz grades? **Schedule an appointment.**

Last Time

- **Lists**
  - Provide the ability to return mixed data types
  - Act as a "dictionary"
- **Shiny**
  - Allows for R to communicate with a web browser
  - Lower the barrier of entry for sharing new algorithms with other users

Lecture Objectives

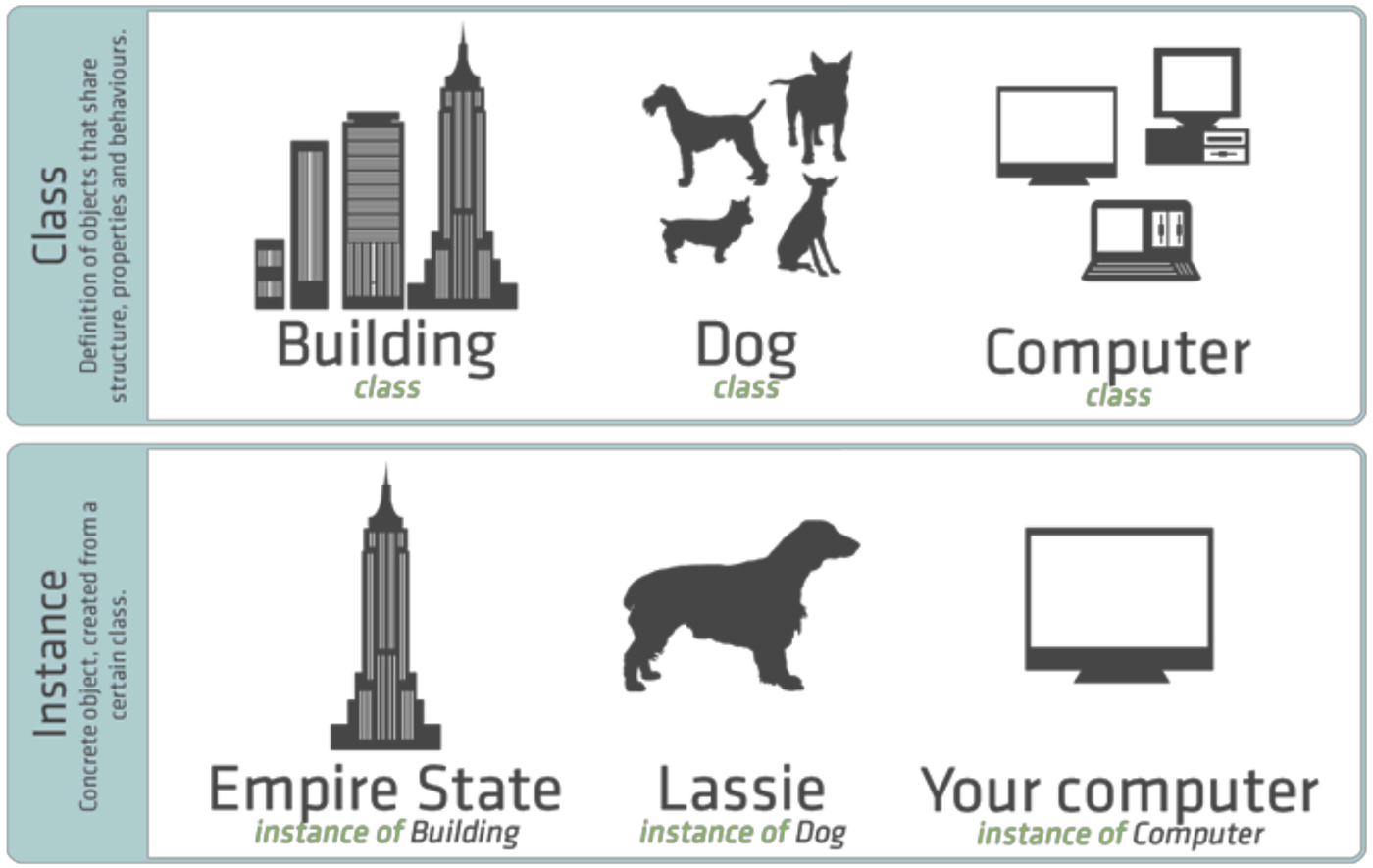
- **Describe** how objects are represented in programming.
- **Differentiate** dispatches between method functions.
- **Implement** an algorithm using S3 for different options

OOP

Focus has been on creating **functions**

How can we write functions to **act** on **different objects**?

Definition:
Object Oriented Programming (OOP) refers is a programming paradigm where real world ideas can be described as a collection of items that are able to interact together.

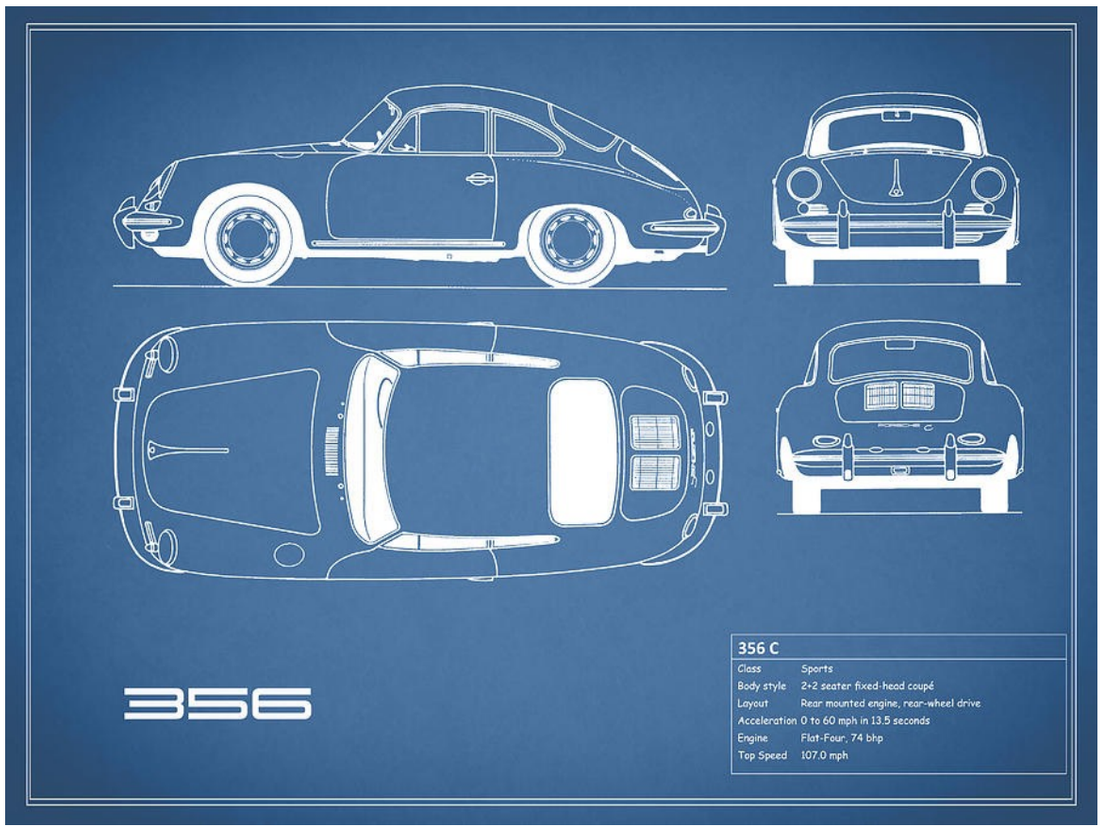


Source

Previously

Class to Object

… **blueprints** to **instances** …



Source

Source

Source

Source

OO System

… objects as they relate to students …

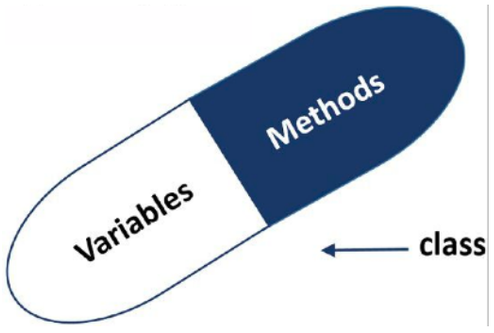
- **Definition**: *Classes* are definitions of what an *object* is.
  - *Student* has properties of *Name*, *NetID*, *Grades*, *Address*, ...

- **Definition**: *Objects* are instances of a *class*.
  - *Charlie* and *Danny* are instances of a *Student*

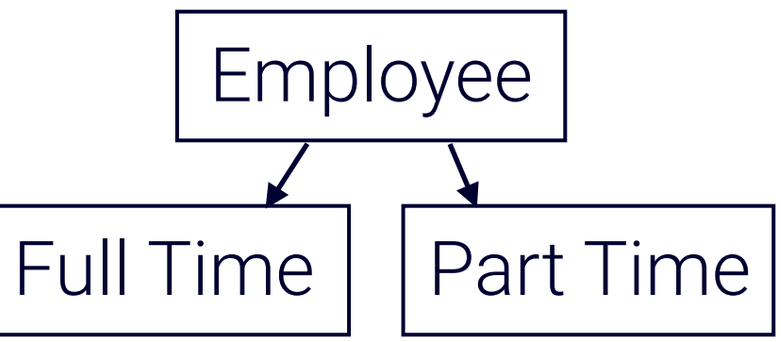
- **Definition**: *Methods* are functions that performs specific calculations on objects of a specific class. (verb)
  - *in_class()* and *get_grade()*

OO Tenets

… fundamentals of an OO system …



Encapsulation: Enables the combination of data and functions into classes



Inheritance: Extend a parent class by creating a child classes without copying!

"Cut"

Source

Source

Polymorphism: Functions are able to act differently across classes

Your Turn

How would an **instructor** *class* be defined?

How might we abstract both an **instructor** and **student** class so that they share a common base/ parent class?

S3 Objects

Previously

"Everything that *exists* in *R* is an **Object**.
Everything that *happens* in *R* is a **Function Call**.
Interfaces to other software are part of *R*"

–John M. Chambers, Extending R (2016) pg. 4

OOP in *R*

… different systems …

- **S3**: *Informal* system that is predominate throughout *R*.
- **S4**: *Formal* system with rigorous class definitions.
- **R6**: Mimics traditional OO systems of *Java* and *C++*

… insert new *R* OOP paradigm conceived while lecturing …

S3: data.frame

```
x = data.frame(
    grade = c("A", "B"),
    pts = c(95, 88))

class(x)
# [1] "data.frame"

attributes(x)
# $names
# [1] "grade" "pts"
# $class
# [1] "data.frame"
# $row.names
# [1] 1 2

unclass(x)
# $grade
# [1] A B
# Levels: A B
# $pts
# [1] 95 88
# attr(,"row.names")
# [1] 1 2
```

Constructing an S3 Object

```
# Option 1:
# Create and assign class
aj = structure(list(),
               class = "student")

# Option 2:
# Create object, then set class
sai = list()
class(sai) = "student"

all.equal(sai, aj)
# [1] TRUE
```

Inheritance

```
# Construct the aj object
# from the student class
aj = structure(list(),
               class = "student")

# Check for a class
inherits(aj, "student")
# [1] TRUE

inherits(aj, "list")
# [1] FALSE
```

S3 is Informal

```
# Construct a student object
qihui = structure(list(),
                  class = "student")

# Corrupting an object
class(qihui) = "data.frame"

# View result
qihui
# data frame with 0 columns
and 0 rows
```

Constructing a Class

- Consider classes with properties:
  - **human**: first_name
  - **instructor**: first_name, course
- Hierarchy would be: **instructor < human**

```
# Set up constructors for the class of human and instructor
new_human = function(first_name) {
  human_obj = structure(list(first_name = first_name), class = "human")
  return(human_obj)
}
new_instructor = function(first_name, course) {
  instructor_obj = structure(list(first_name = first_name, course = course),
                                  class = c("instructor ","human"))
  return(instructor_obj)
}
```

Your Turn

Write a constructor for the **student** class given a definition of:

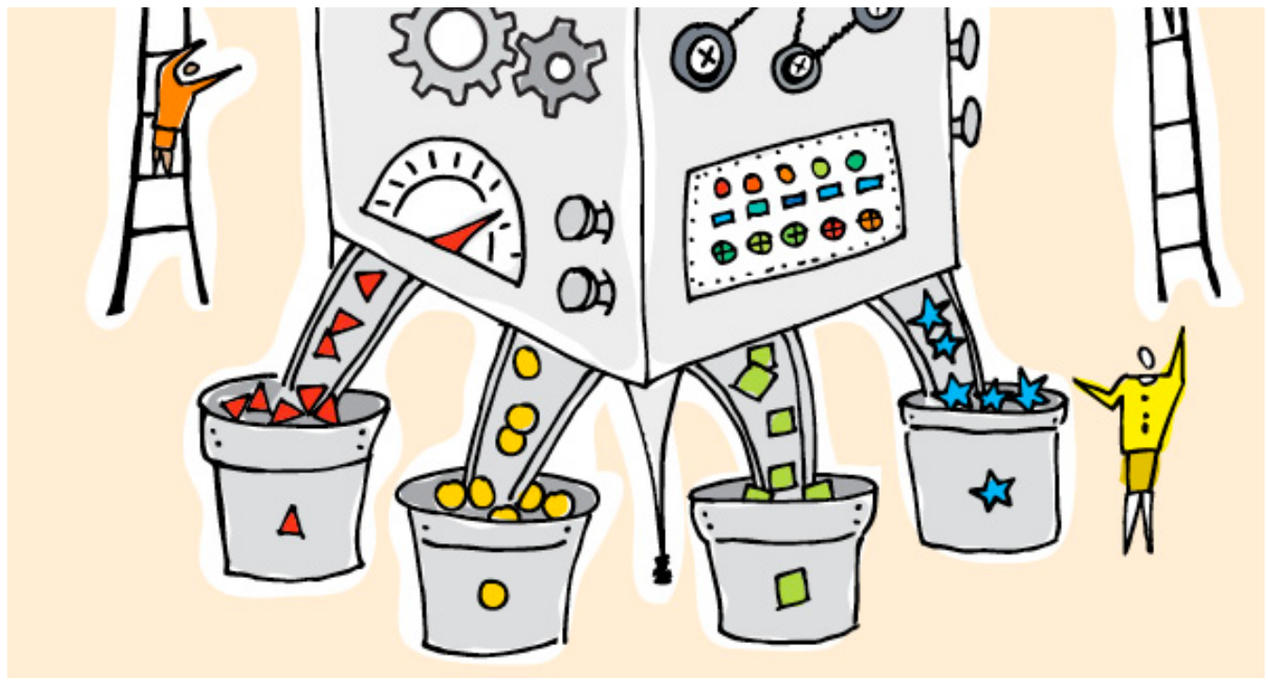
- **student**: first_name, course, grade

With a hierarchical relationship of:

- **student < instructor < human**

Definition:
A *generic function* determines the **class** of a single argument and dispatches to the appropriate **method**.

Sorting Machine



Source

S3 Generic Functions

… naming convention for methods …

```
# Sample of generic functions in R
summary(object, ...)
print(x, ...)
plot(x, y, ...)
mean(x, ...)


# These dispatch to a generic function defined for a class
# generic.class(x, ...)

# Implemented classes for mean
methods(mean)
# [1] mean.Date     mean.default  mean.difftime
# [4] mean.POSIXct  mean.POSIXlt
```

---

* If a class has not been defined for use in a generics, it will fail. To avoid the failure define **generic.default**() (e.g. **summary.default**())

Forming a Generic

… writing generic functions …

```
# Setting up a generic
my_generic = function(x, ...) {
  UseMethod("my_generic")
}

# Method for handling a data.frame object
my_generic.data.frame = function(x, ...) {
  message("This is a data.frame object.")
}

# Call generic function
my_generic(data.frame())
# This is a data.frame object.
```

Same Name

Previously

Definition:

Ellipsis or *dot-dot-dot* (...) allow for any number of parameters to be passed in to the function being called.

```
# Definition of min
min
# function (..., na.rm = FALSE)  .Primitive("min")

# Multiple parameters
min(3:9, -2, 1002, 58)
# [1] -2
```

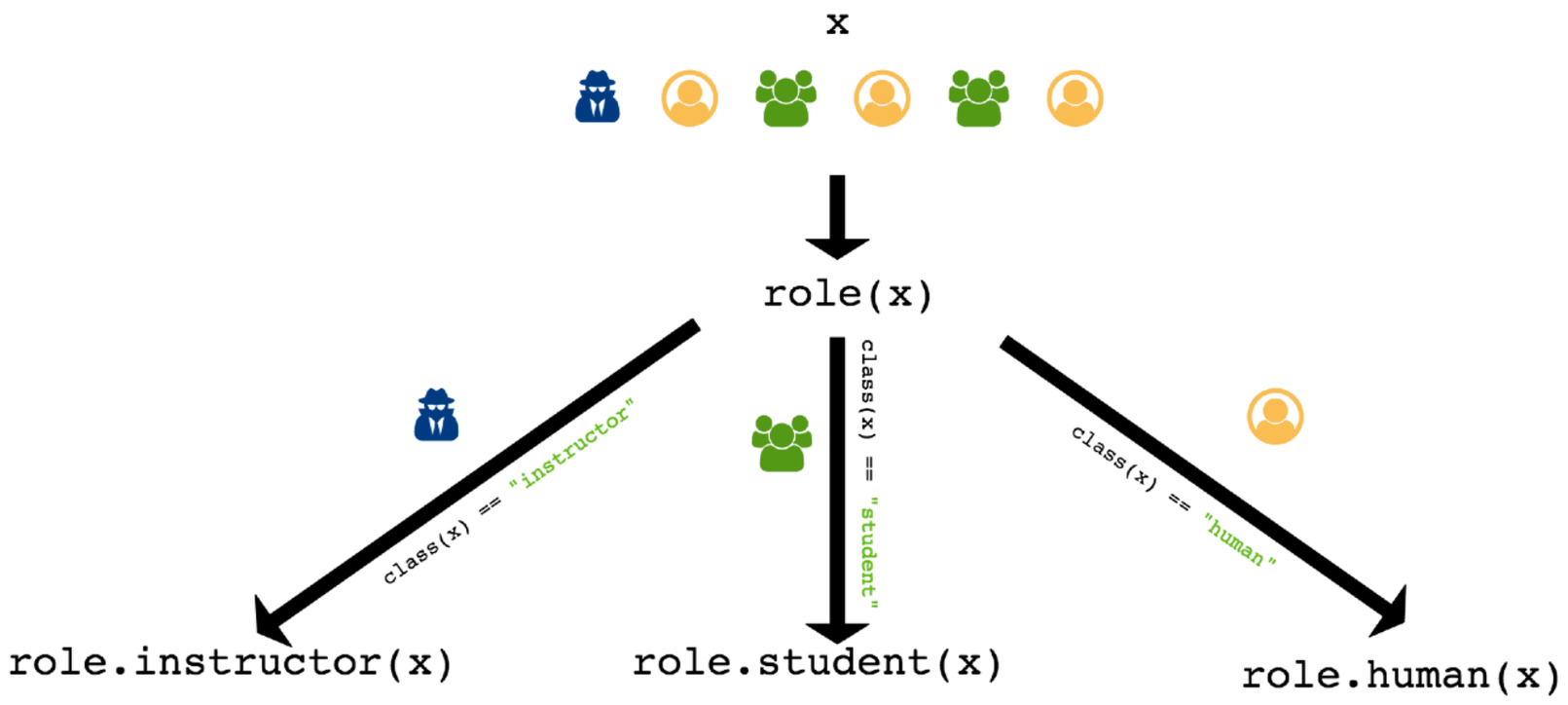
Announcing Roles

… example generic routing …

```
# Create a generic for `role`
role = function(x, ...) {
  UseMethod("role")
}
# Create a role identifier for class human and instructor
role.human = function(x, ...) {
  cat("Hi there human", x$first_name, "!")
}
role.instructor = function(x, ...) {
  cat("Instructor ", x$first_name, ",  you're teaching ", x$course , " this semester.")
}
```

```r
# Example use of generic functions to route to the appropriate method
jjb = new_instructor("James", "STAT 385")
role(jjb)
# Instructor James, you're teaching STAT 385 this semester.

tw = new_human("Teng")
role(tw)
# Hi there human Teng!
```



Your Turn

Try passing an object with the **student** class into **role()**, what happens? Why?

Try now with **role(3)**, what happens?

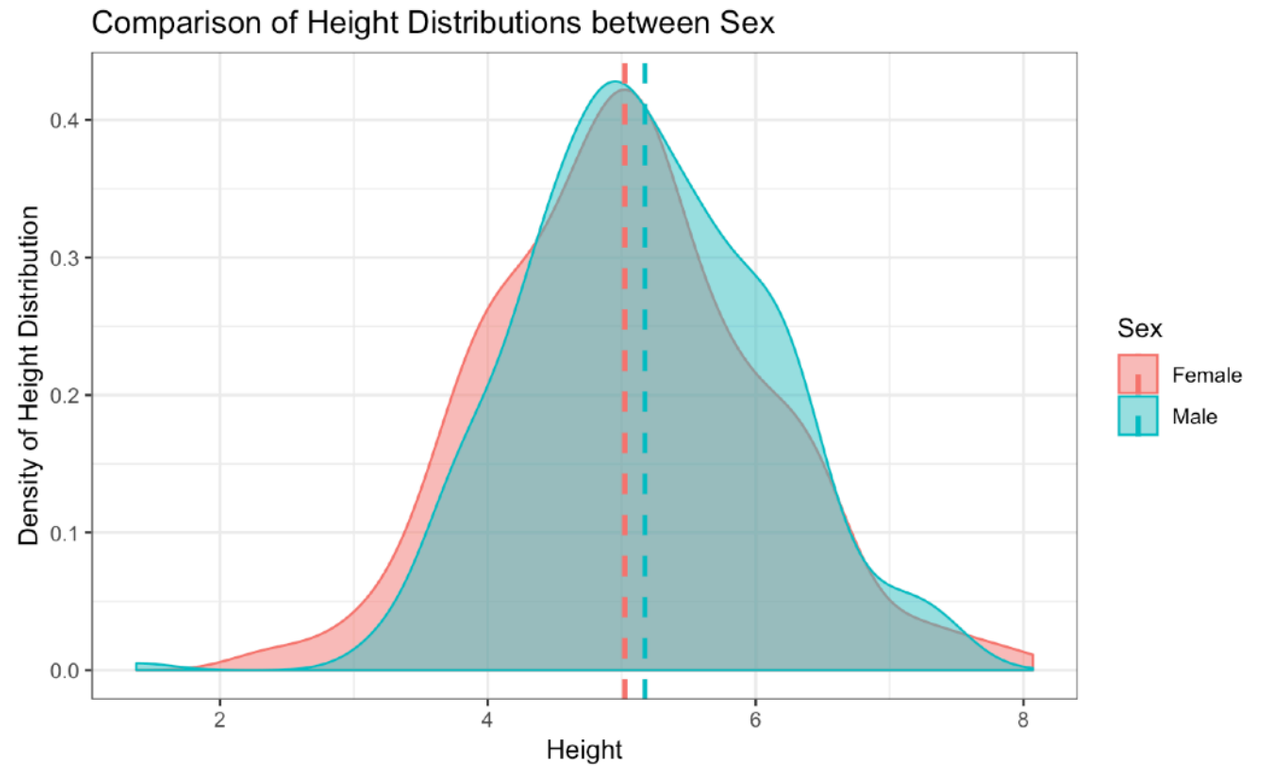
Construct a **role.default(x, …)** and **try again.**

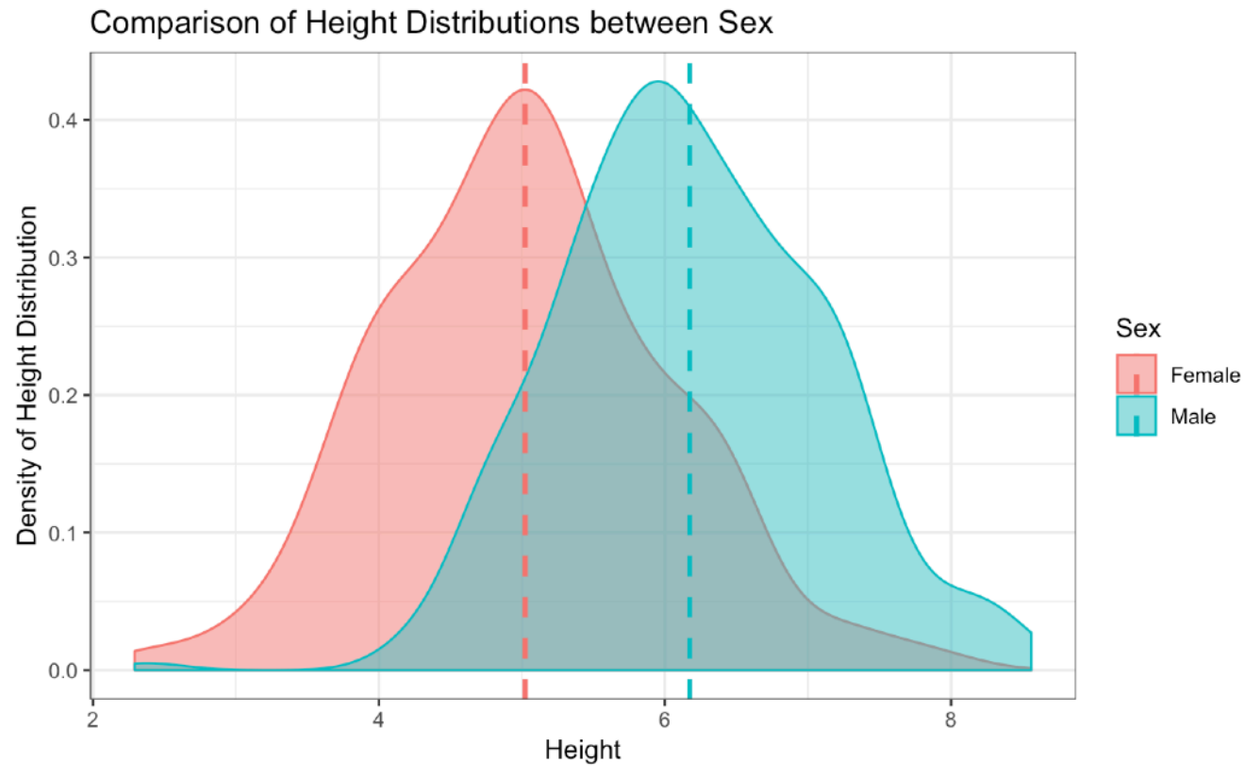
Unpaired (Two Sample) t-Test

Previously

Comparing Groups

… does one group differ from the next ???





VS.

Previously

Formulas

… breaking down the formula …

Statistic $t = \dfrac{\bar{x}_1 - \bar{x}_2}{\sqrt{s^2\left(\dfrac{1}{n_1} + \dfrac{1}{n_2}\right)}}$ where $s^2 = \dfrac{\sum_{i=1}^{n_1}\left(x_i - \bar{x}_1\right)^2 + \sum_{j=1}^{n_2}\left(x_j - \bar{x}_2\right)^2}{n_1 + n_2 - 2}$

n_1 and n_2 represent the sample size of data,

and the sample mean is $\bar{x} = \dfrac{1}{n}\sum_{i=1}^{n} x_i$

Previously

my_ttest()

… implementation as a function…

```
my_ttest = function(x1, x2, test = c("two-sided", "lower", "upper"), alpha = 0.05) {
  # Force `test` to hold a pre-defined value
  test = match.arg(test)
  # Compute length and degrees of freedom
  n1 = length(x1); n2 = length(x2); ndf = n1 + n2 - 2
  # Calculate t-statistic
  s2 = ((n1 - 1) * var(x1) + (n2 - 1) * var(x2)) / ndf
  tstat = (mean(x1) - mean(x2)) / sqrt(s2 * (1 / n1 + 1 / n2))
  # Compute tail probability
  tail_prob = switch(test,
    "two-sided" = 2 * (1 - pt(abs(tstat), ndf)),
    "lower" = pt(tstat, ndf),
    "upper" = 1 - pt(tstat, ndf))
  # Format and return results
  results = list(tstat = tstat, df = ndf, reject = tail_prob < alpha, prob = tail_prob)
  return(results)
}
```

Testing my_ttest()

```r
# Set seed for reproducibility
set.seed(881)

# Generate data
n = 10
x1 = round(rnorm(n), 1)
x2 = round(rnorm(n) + 1, 1)
test_result = my_ttest(x1, x2)
test_result
# $tstat
# [1] -1.979717
# $df
# [1] 18
# $reject
# [1] FALSE
# $prob
# [1] 0.06323542

# Check against built in implementation
all.equal(test_result[-3],
          t.test(x1, x2, var.equal = TRUE)[1:3],
          check.attributes = FALSE)
# [1] TRUE
```

What if we wanted to supply a **data.frame**, **matrix**, **list**, or **factor**?

Use a Generic!

… implementing my_ttest() as a generic …

```
# Create a generic for `my_ttest`
my_ttest = function(x, ...) {
  UseMethod("my_ttest")
}
```

```
# Implement methods for objects to subset data.
my_ttest.matrix = function(x, ... ) { my_ttest(x[, 1], x[, 2], ... ) }
my_ttest.data.frame = function(x, ... ) { my_ttest(x[, 1], x[, 2], ... ) }
my_ttest.list = function(x, ... ) { my_ttest(x[[1]], x[[2]], ... ) }
my_ttest.factor = function(x, ... ) {
    lev = levels(x)
    my_ttest(x[x == lev[1]], x[x == lev[2]])
}
```

Create a Default Method

... transitioning from a function to a default ...

```
my_ttest.default = function(x1, x2, test = c("two-sided", "lower", "upper"),
                            alpha = 0.05) {
  # Force `test` to hold a pre-defined value
  test = match.arg(test)
  # Compute length and degrees of freedom
  n1 = length(x1); n2 = length(x2); ndf = n1 + n2 - 2
  # Calculate t-statistic
  s2 = ((n1 - 1) * var(x1) + (n2 - 1) * var(x2)) / ndf
  tstat = (mean(x1) - mean(x2)) / sqrt(s2 * (1 / n1 + 1 / n2))
  # Compute tail probability
  tail_prob = switch(test,
    "two-sided" = 2 * (1 - pt(abs(tstat), ndf)),
    "lower" = pt(tstat, ndf),
    "upper" = 1 - pt(tstat, ndf))
  # Format and return results
  results = structure(list(tstat = tstat, df = ndf, reject = tail_prob < alpha,
                           prob = tail_prob), class = "my_ttest")
  return(results)
}
```

Testing Generic Implementation

```r
# Set seed for reproducibility
set.seed(881)

# Generate data
n = 10
my_data = data.frame(
    x1 = round(rnorm(n), 1),
    x2 = round(rnorm(n) + 1, 1)
)
test_result = my_ttest(my_data)
attributes(test_result)
# $names
# [1] "tstat"  "df"     "reject" "prob"
# $class
# [1] "my_ttest"

# Check against built in
implementation
all.equal(test_result[-3],
  t.test(x1, x2, var.equal = TRUE)[1:3],
  check.attributes = FALSE)
# [1] TRUE
```

How can we **customize object output**?

Extending a Generic

```
# Create extend the base R
# print method for my_ttest
print.my_ttest = function(x, ... ) {

  # Only print the first two list items
  print(unlist(x[1:2]))

  # Return the original object silently
  invisible(x)
}

# Controlled output in non-list form!
test_result
#          tstat             df
# -1.979717 18.000000
```

Extending an Extension of a Generic

```
# Extend the base R
# summary method to support my_ttest
summary.my_ttest =
                    function(object, ... ) {
 structure(object,
              class = c("sum_my_ttest",
                        class(object)))
}

# Custom print method for summary
# of my_ttest
print.sum_my_ttest =
                    function(x, ... ) {
  cat("Unpaired (Two-Sample) t-Test Results \n")
  cat("t =", format(x$tstat, ... ), "on", x$ndf, "d.f.\n")
  cat("p-value:", format(x$prob, ...), "\n")
  cat("Null hypothesis is",
         if(!x$reject) "not" else "", "rejected.\n")
  invisible(x)
}

# Beautiful output!
summary(test_result)
# Unpaired (Two-Sample) t-Test Results
# t = -1.979717 on d.f.
# p-value: 0.06323542
# Null hypothesis is not rejected.
```

Code Demo

… implementing a custom lm object …

Recap

- **OOP**
  - **O**bject **o**riented **p**rogramming emphasizes objects in a collection with interactions.
  - Focuses on **encapsulation**, **inheritance**, and **polymorphism**.
- **S3 Objects**
  - Each object has an underlying **class** attribute.
  - **Generic functions** are used to dispatch to the appropriate method function for an object.
- **Unpaired (Two-Sample) t-Test**
  - Case study in implementing a method for assessing group difference.

Acknowledgements

Acknowledgements

- Chapter 4: Classes of S Programming by W.N. Venable and B.D. Ripley