DATA 133 - Introduction to Data Science I

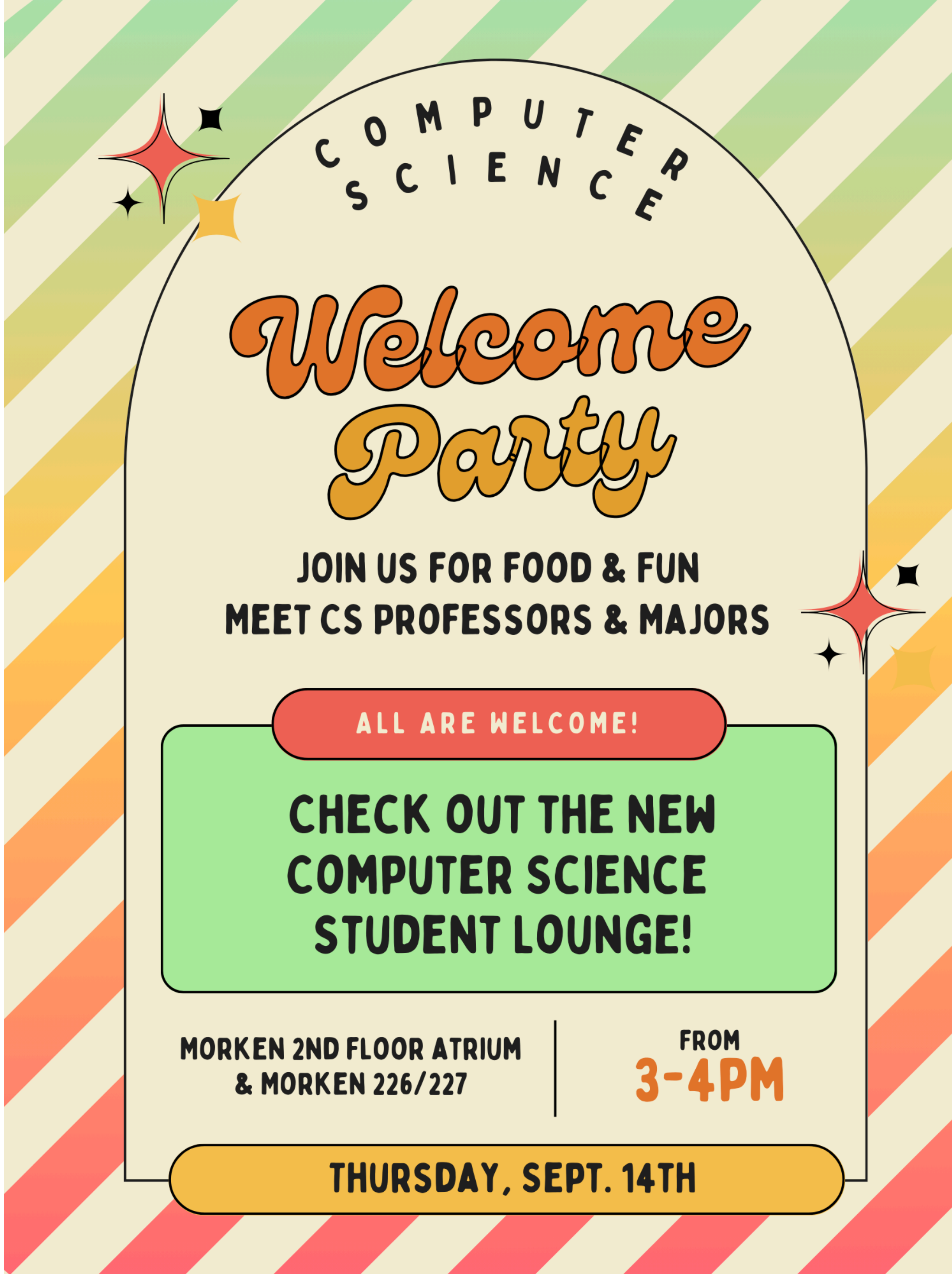
Instructor: Renzhi Cao
Computer Science Department
Pacific Lutheran University

Announcements

- Practice quiz (not counted for final grade)
- Go over Quiz 1 (10% off per day for late penalty). Go over it together.
- Read books: Page 23-41
- Quiz 2 on next Tuesday

- Computer Science Welcome party

- Declare the data science minor: https://www.plu.edu/computer-science/documents/declare-csci-or-data-major-or-minor/



COMPUTER SCIENCE

Welcome Party

JOIN US FOR FOOD & FUN
MEET CS PROFESSORS & MAJORS

ALL ARE WELCOME!

CHECK OUT THE NEW COMPUTER SCIENCE STUDENT LOUNGE!

MORKEN 2ND FLOOR ATRIUM
& MORKEN 226/227

FROM
3-4PM

THURSDAY, SEPT. 14TH

Reference book

- R Programming for Data Science. By Roger Peng. **ISBN-10:** 1365056821, April 20, 2016.

Pictures from: https://www.google.com/search?

Learning in today

- R basics - Matrices, Factor, Data Frame, subsetting, etc.

Matrices

Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (number of rows, number of columns)

```
> m <- matrix(nrow = 2, ncol = 3)
>m
[,1] [,2] [,3] [1,] NA NA NA [2,] NA NA NA
> dim(m)
[1] 2 3
> attributes(m)
$dim
[1] 2 3
```

Matrices

Matrices are constructed column-wise, so entries can be thought of starting in the "upper left" corner and running down the columns.

```
> m <- matrix(1:6, nrow = 2, ncol = 3)
>m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrices

Matrices can also be created directly from vectors by adding a dimension attribute.

```
> m <- 1:10
>m
[1] 1 2 3 4 5 6 7 8 9 10
> dim(m) <- c(2, 5)
>m

>m[1,2]
> n <- seq(1,10,2)
>n
```

Matrices

Matrices can be created by column-binding or row-binding with the cbind() and rbind() functions.

```
> x <- 1:3
> y <- 10:12
> cbind(x, y)
> rbind(x, y)
```

Guided practice

1.Create the following matrices and print it out:

$$\begin{matrix} 1 & 3 & 5 \\ 7 & 9 & 11 \\ 13 & 15 & 17 \end{matrix}$$

2. Create the following matrices and print it out:

$$\begin{matrix} 1 & 41 & 455 & 474 \\ 2 & 239 & 121 & 357 \\ 61 & 65 & 178 & 533 \end{matrix}$$

Guided practice solution

1.
```
c1 <- seq(1,5,2)
c2 <- seq(7,11,2)
c3 <- seq(13,17,2)
m <- cbind(c1,c2,c3)
```

2.
```
tem <- c(1,2,61,41,239,65,455,121,178,474,357,533)
m <- matrix(tem,nrow=3, ncol=4)
```

Factors

Factors are used to represent categorical data (unordered or ordered), like integer vector where each integer has a label.

- Self-describing. “Male” and “Female” is better value compared to 1 and 2.
- Use factor() function to create a factor.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
>x
>table(x)
## See the underlying representation of factor
> unclass(x)
```

Factors

The order of the levels of a factor can be set using the levels argument to factor(). This can be important in linear modelling because the first level is used as the baseline level.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x ## Levels are put in alphabetical order
[1] yes yes no yes no
Levels: no yes
> x <- factor(c("yes", "yes", "no", "yes", "no"), levels <- c("yes", "no"))
> x
[1] yes yes no yes no
Levels: yes no
```

Missing values

Missing values are denoted by NA or NaN for undefined mathematical operations. (NaN means not a number, like 0/0. NA means missing values)

- is.na() is used to test objects if they are NA
- is.nan() is used to test for NaN
- NA values have a class also, so there are integer NA, character NA, etc.
- A NaN is also NA but the converse is not true

Missing values

```
> ## Create a vector with NAs in it
> x <- c(1, 2, NA, 10, 3)
> ## Return a logical vector indicating which elements are NA
> is.na(x)
> is.nan(x)
> ## Now create a vector with both NA and NaN values
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
> is.nan(x)
```

Practice

1. Create a vector with the values of 1, 3, NA, 5, NaN

2. Test NA

3. Test NaN

Guided Practice Solution

1. Create a vector with the values of 1, 3, NA, 5, NaN

```
v <- c(1,3,NA,5,NaN)
```

2. Test NA

```
is.na(v)
```

3. Test NaN

```
is.nan(v)
```

Data Frames

Data frames are used to store tabular data in R.
Data frames are represented as a special type of list where every element of the list has to have the same length.

Each element of the list can be thought of as a column and the length of each element of the list is the number of rows.
What is this looks like and what is the difference?

Unlike matrices, data frames can store different classes of objects in each column. Matrices must have every element be the same class (e.g. all integers or all numeric).
Data frames have a special attribute called row.names which indicate information about each row of the data frame.

Data Frames

Data frames are usually created by reading in a dataset using the read.table() or read.csv(). Also, be created explicitly with the **data.frame**() function

Data frames can be converted to a matrix by calling **data.matrix**().

```
> x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
>nrow(x)
>ncol(x)
```

Names

R objects can have names, which is very useful for writing readable code and self-describing objects.

```
> x <- 1:3
> names(x)

> names(x) <- c("New York", "Seattle", "Los Angeles")
```

Lists can also have names, which is often very useful.

```
> x <- list("Los Angeles" = 1, Boston = 2, London = 3)
> x
```

Names

Matrices can have both column and row names.

```
> m <- matrix(1:4, nrow = 2, ncol = 2)
> dimnames(m) <- list(c("a", "b"), c("c", "d"))
>m
```

Column names and row names can be set separately using the colnames() and rownames() functions.

```
> colnames(m) <- c("h", "f")
> rownames(m) <- c("x", "z")
```

Practice

1.Create a data frame "df" with the following values:

ID	Score	DATA133
1	89	TRUE
2	30	FALSE
3	0	FALSE
4	99	TRUE

2. Convert data frame "df" to matrix m, and print the score of ID 3.

Break

Guided Practice Solution

1. df <-
   data.frame(ID=1:4,Score=c(89,30,0,99),DATA133=c(T,F,F,T))

2. print(m[3,2])

Subsetting of R objects

There are three operators that can be used to extract subsets of R objects.

- The [operator always returns an object of the same class as the original. It can be used to select multiple elements of an object
- The [[operator is used to extract elements of a list or a data frame. It can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame.
- The $operator is used to extract elements of a list or data frame by literal name. Its semantics are similar to that of [[.

Subsetting of vector

```
> x <- c("a", "b", "c", "c", "d", "a")
> x[1] ## Extract the first element
> x[2] ## Extract the second element
```

The [operator can be used to extract multiple elements of a vector by passing the operator an integer sequence.

```
> x[1:4]
> x[c(1, 3, 4)]
```

Subsetting of vector

We can also pass a logical sequence to the [operator to extract elements of a vector that satisfy a given condition.

```
> u <- x > "a"
> u
> x[u]
> x[x > “a”]       # another convenient way
```

Subsetting of Matrix

Matrices can be subsetted in the usual way with (i,j) type indices. Here, we create simple 2*3 matrix with the matrix function.

```
> x <- matrix(1:6, 2, 3)
>x
```

We can access the $(1, 2)$ or the $(2, 1)$ element of this matrix using the appropriate indices.

```
> x[1, 2]
> x[2, 1]

> x[1, ] ## Extract the first row
> x[, 2] ## Extract the second column
```

Subsetting of Matrix

Dropping matrix dimensions

By default, when a single element of a matrix is retrieved, it is returned as a vector of length 1 rather than a 1*1 matrix. Often, this is exactly what we want, but this behavior can be turned off by setting drop = FALSE.

```
> x <- matrix(1:6, 2, 3)
> x[1, 2]
> x[1, 2, drop = FALSE]
> x[1, ]
> x[1, , drop = FALSE]
```

Subsetting of List

Lists in R can be subsetted using all three of the operators mentioned above, and all three are used for different purposes.

```
> x <- list(foo = 1:4, bar = 0.6)
>x
```

The [[operator can be used to extract single elements from a list. Here we extract the first element of the list.

```
> x[[1]]
```

Subsetting of List

The [[operator can also use named indices so that you don’t have to remember the exact ordering of every element of the list. You can also use the $ operator to extract elements by name.

```
> x[["bar"]]
> x$bar
```

Subsetting of List

One thing that differentiates the [[operator from the $ is that the [[operator can be used with computed indices. The $ operator can only be used with literal names.

```
> x <- list(foo = 1:4, bar = 0.6, baz = “hello")
> name <- "foo"
>
> ## computed index for "foo"
> x[[name]]

>## the element “name” doesn’t exists
> x$name

> ## element "foo" does exist
> x$foo
```

Subsetting Nested Elements of a List

The [[operator can take an integer sequence if you want to extract a nested element of a list.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))
>
> ## Get the 3rd element of the 1st element
> x[[c(1, 3)]]
> ## Same as above
> x[[1]][[3]]

> ## 1st element of the 2nd element
> x[[c(2, 1)]]
```

Partial matching

Partial matching of names is allowed with [[and $. This is often very useful during interactive work if the object you're working with has very long element names.

```
> x <- list(aardvark = 1:5)
> x$a

> x[[“a"]]

> x[["a", exact = FALSE]]
```

Removing NA values

A common task in data analysis is removing missing values (NAs).

```
> x <- c(1, 2, NA, 4, NA, 5)
> bad <- is.na(x)
> print(bad)

> x[!bad]
```

Removing NA values

What if there are multiple R objects and you want to take the subset with no missing values in any of those objects?

```
> x <- c(1, 2, NA, 4, NA, 5)
> y <- c("a", "b", NA, "d", NA, "f")

> good <- complete.cases(x, y)

> good
> x[good]

> y[good]
```

Removing NA values

You can use complete.cases on data frames too.

```
> head(airquality)

> good <- complete.cases(airquality)

> head(airquality[good, ])
```

Vectorized operations

Many operations in R are vectorized, meaning that operations occur in parallel in certain R objects. This allows you to write code that is efficient, concise, and easier to read than in non-vectorized languages.

```
> x <- 1:4
> y <- 6:9
> z <- x + y
>z
> x >= 2
>x-y
>x*y
```

Vectorized operations

Matrix operations are also vectorized, making for nicely compact notation.

```
> x <- matrix(1:4, 2, 2)
> y <- matrix(rep(10, 4), 2, 2)
> ## element-wise multiplication
>x*y
> ## element-wise division
>x/y
> ## true matrix multiplication
> x %*% y
```

Practice of R

Try it in Pairs, recommend https://replit.com/
Today's Pair-programming is due by next Tuesday
Quiz 2 on next Tuesday
Read book 12 - 41