Introduction to Computer Programming with R (FOR 6934)

Qing Zhao (School of Forest Resources & Conservation, qing.zhao@ufl.edu)
Daijiang Li (Department of Wildlife Ecology & Conservation, dli1@ufl.edu)
Denis Valle (School of Forest Resources & Conservation, drvalle@ufl.edu)

Class six

- Understand loops in R
- Use "for" loops to manipulate data in vectors
- Use "for" loops to create graphs

How do loops work in R?

- Loops repeat two steps: evaluation and execution
- Two outcomes in evaluation
  - Condition satisfied
  - Condition not satisfied
- Three types of loops in R
  - "for" loop
  - "while" loop
  - "repeat" loop

Three loop types

Type	Condition	Evaluation	Execution
for	if a variable is in a sequence	satisfied	execute
		not satisfied	stop
while	any	satisfied	execute
		not satisfied	stop
repeat	any	not satisfied	execute
		satisfied	stop

Why use loops?

- Loops represent element-level thinking
  - better understanding of data
- Loops can make your code shorter
  - easier to debug
  - produce consistent results

Form of "for" loop

General form
for (variable in sequence) {

command 1 to excute
command 2 to execute
...
command n to execute
}

Example
for (k in 1:5) {
print(k - 3)
print(k ^ 2)
}

Form of "while" loop

General form
while (condition) {
command 1 to excute
command 2 to execute
...
command n to execute
}

Example
while (b < 5) {
print(b)
b <- b + 1.5
}

Form of "repeat" loop

General form
repeat {
command 1 to excute
command 2 to execute
...
command n to execute
if (condition) {
break
}
}

Example
repeat {
a <- a + 1
b <- b * 2
if (a < b) {
break
}
}

Sample code

"for" loop

```r
a <- c(2, 6)
for (i in 1:2) {
    a[i] <- a[i] + 10
}
a
```

```
## [1] 12 16
```

Sample code

"for" loop code

```r
a <- c(2, 6)
for (i in 1:2) {
    print(paste('i', '=', i, sep=' ')) # This is just for illustration
    print(paste(c('a', '=', a), collapse=' ')) # This is just for illustration
    cat('execution\n') # This if just for illustration
    a[i] <- a[i] + 10 # This is executed
    print(paste(c('a', '=', a), collapse=' ')) # This is just for illustration
    print(paste('i', '=', i, sep=' ')) # This is just for illustration
    cat('next\n\n') # This is just for illustration
}
```

"for" loop illustration

```
## [1] "i = 1"
## [1] "a = 2 6"
## execution
## [1] "a = 12 6"
## [1] "i = 1"
## next
##
## [1] "i = 2"
## [1] "a = 12 6"
## execution
## [1] "a = 12 16"
## [1] "i = 2"
## next
```

Sample code

Change the same variable in a "for" loop

```r
b <- 1
for (i in 1:2) {
    b <- b + 10^i
}
b
```

```
## [1] 111
```

Sample code

"for" loop code

```
b <- 1
for (i in 1:2) {
    print(paste('i', '=', i, sep=' ')) # This is just for illustration
    print(paste(c('b', '=', b), collapse=' ')) # This is just for illustration
    cat('execution\n') # This if just for illustration
    b <- b + 10^i # This is executed
    print(paste(c('b', '=', b), collapse=' ')) # This is just for illustration
    print(paste('i', '=', i, sep=' ')) # This is just for illustration
    cat('next\n\n') # This is just for illustration
}
```

"for" loop illustration

```
## [1] "i = 1"
## [1] "b = 1"
## execution
## [1] "b = 11"
## [1] "i = 1"
## next
##
## [1] "i = 2"
## [1] "b = 11"
## execution
## [1] "b = 111"
## [1] "i = 2"
## next
```

This concludes Class 6, Section 1

Please continue on to the next video

Use "for" loops in vectors

- Codes and results of "for" loops
- What did the codes do in each step

Sample code

Numeric vectors

```
b <- numeric(5)
for (i in 1:5) {
    b[i] <- 3^i
}
b
```

```
## [1]   3   9  27  81 243
```

What did the above code do?

Step	i	b[i]	Math	Value
1	1	b[1]	3^1	3
2	2	b[2]	3^2	9
3	3	b[3]	3^3	27
4	4	b[4]	3^4	81
5	5	b[5]	3^5	243

Sample code

Character vectors

```
a <- c('tom', 'jerry')
b <- character(2)
for (i in 1:2) {
    b[i] <- toupper(a[i])
}
a
```

```
## [1] "tom"   "jerry"
```

```
b
```

```
## [1] "TOM"   "JERRY"
```

What did the above code do?

Step	i	a[i]'s value	b[i]'s value
1	1	"tom"	"TOM"
2	2	"jerry"	"JERRY"

Sample code

Character vectors, cont'd

```
a <- c('tom', 'jerry')
b <- character(2)
for (i in 1:2) {
    f <- substr(a[i], start=1, stop=1)
    g <- substr(a[i], start=2, stop=10)
    h <- toupper(f)
    b[i] <- paste(h, g, sep='')
}
a
```

```
## [1] "tom"   "jerry"
```

```
b
```

```
## [1] "Tom"   "Jerry"
```

What did the above code do?

Step	i	a[i]'s value	f's value	g's value	h's value	b[i]'s value
1	1	"tom"	"t"	"om"	"T"	"Tom"
2	2	"jerry"	"j"	"erry"	"J"	"Jerry"

Sample code

Multiple types of values

```
a <- c(0.5, 1.4, 2.3, 3.2)
b <- 1
c <- logical(4)
for (i in 1:4) {
    b <- b * a[i]
    c[i] <- b > 1
}
b
```

```
## [1] 5.152
```

```
c
```

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

What did the above code do?

Step	i	a[i]'s value	b's math	b's value	c[i]'s value
1	1	0.5	1 * 0.5	0.5	FALSE
2	2	1.4	0.5 * 1.4	0.7	FALSE
3	3	2.3	0.7 * 2.3	1.61	TRUE
4	4	3.2	1.61 * 3.2	5.152	TRUE

Sample code

Repetitive coding with mistakes

```
print(paste('Year', 2016, sep=' '))
print(paste('year', 2017, sep=' '))
print(paste('Year', 2018, sep=''))
print(paste('Year', 2018, sep=' '))
```

```
## [1] "Year 2016"
```

```
## [1] "year 2017"
```

```
## [1] "Year2018"
```

```
## [1] "Year 2018"
```

Sample code

Write above code in a loop

```
years <- 2016:2025
for (i in 1:10) {
    print(paste('Year', years[i], sep=' '))
}
```

```
## [1] "Year 2016"
## [1] "Year 2017"
## [1] "Year 2018"
## [1] "Year 2019"
## [1] "Year 2020"
## [1] "Year 2021"
## [1] "Year 2022"
## [1] "Year 2023"
## [1] "Year 2024"
## [1] "Year 2025"
```

Sample code

Use vectorization to create a series of character values

```
paste('Year', 2016:2025, sep=' ')
```

```
##  [1] "Year 2016" "Year 2017" "Year 2018" "Year 2019" "Year 2020"
##  [6] "Year 2021" "Year 2022" "Year 2023" "Year 2024" "Year 2025"
```

Sample code

Use vectorization for calculation

```
a <- c(6.4, 5.5, 4.6, 3.7, 2.8, 1.9)
b <- c(1, -1, 1, -1, 1, -1)
f <- a * b
g <- abs(f) - 1
f
```

```
## [1]  6.4 -5.5  4.6 -3.7  2.8 -1.9
```

```
g
```

```
## [1] 5.4 4.5 3.6 2.7 1.8 0.9
```

When do we want to use loops (and when don't we)?

- Loops represent element-level thinking and are helpful to understand what you do
- Loops are not efficient; use vectorization when possible
- There are complicated situations where vectorization is not applicable

This concludes Class 6, Section 2

Please continue on to the next video

Loops are also useful to make your graphs look good

- Again, shorter code, easier to debug
- Consistency among graphs

Sample code

Read in the duck data

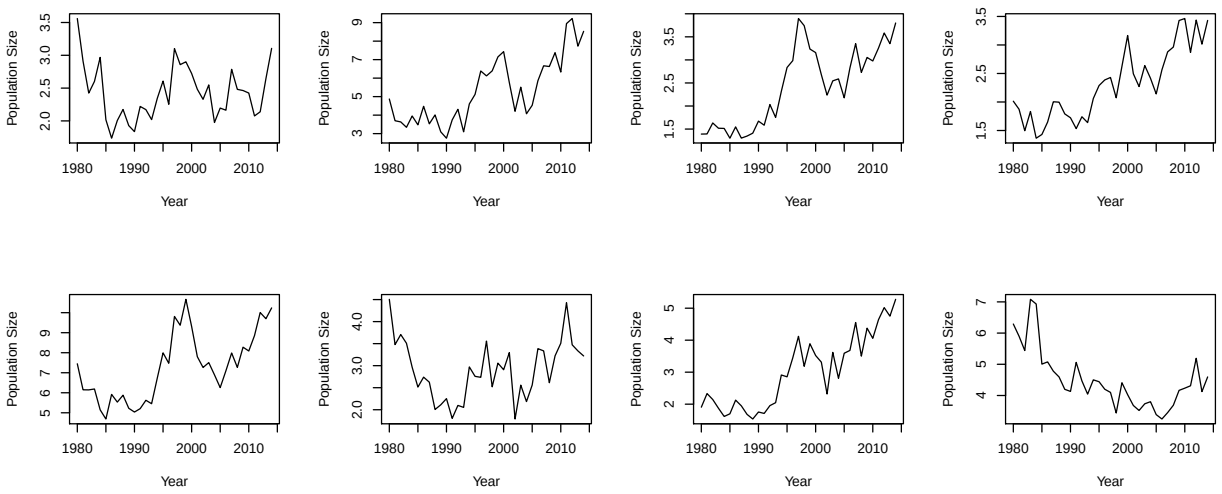
```
dat <- read.csv('c:/data/duck s.csv')
head(dat, n=3)
```

```
##   year American.Wigeon Blue.Winged.Teal  Gadwall Green.Winged.Teal
## 1 1980        3.560459         4.872159 1.391174          2.013855
## 2 1981        2.911041         3.696456 1.393768          1.872633
## 3 1982        2.423980         3.632262 1.632299          1.495983
##    Mallard Northern.Pintail Northern.Shoveler    Scaup
## 1 7.447523         4.505726          1.905971 6.290369
## 2 6.147409         3.477255          2.331135 5.899306
## 3 6.143347         3.706683          2.145041 5.440042
```

Sample code

Plot population trend for each species without using loops

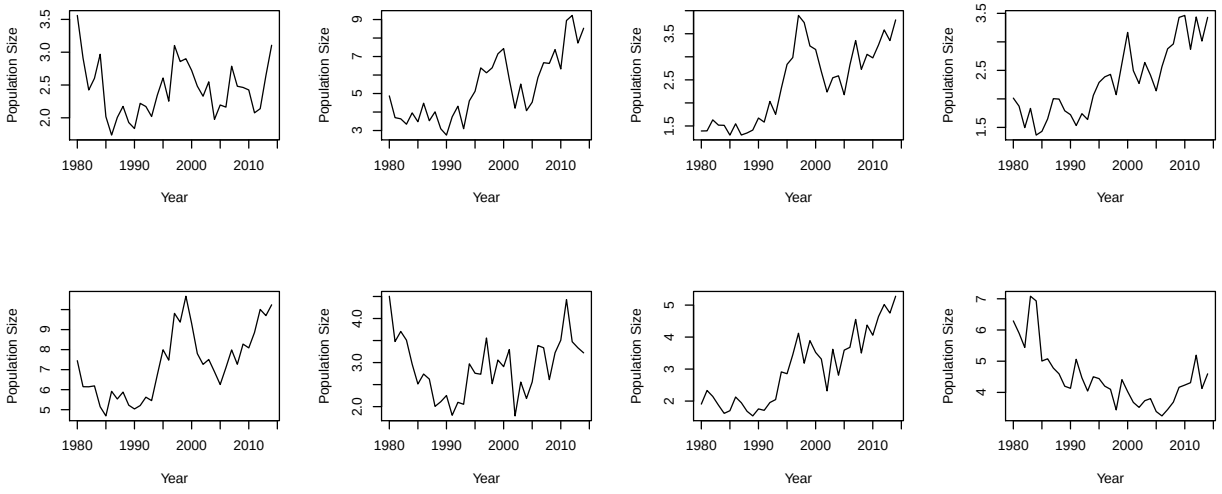
```
par(mfrow=c(2,4))
plot(dat$American.Wigeon ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Blue.Winged.Teal ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Gadwall ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Green.Winged.Teal ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Mallard ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Northern.Pintail ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Northern.Shoveler ~ dat$year, type='l', xlab='Year', ylab='Population Size')
plot(dat$Scaup ~ dat$year, type='l', xlab='Year', ylab='Population Size')
```



Sample code

Write the code in a loop

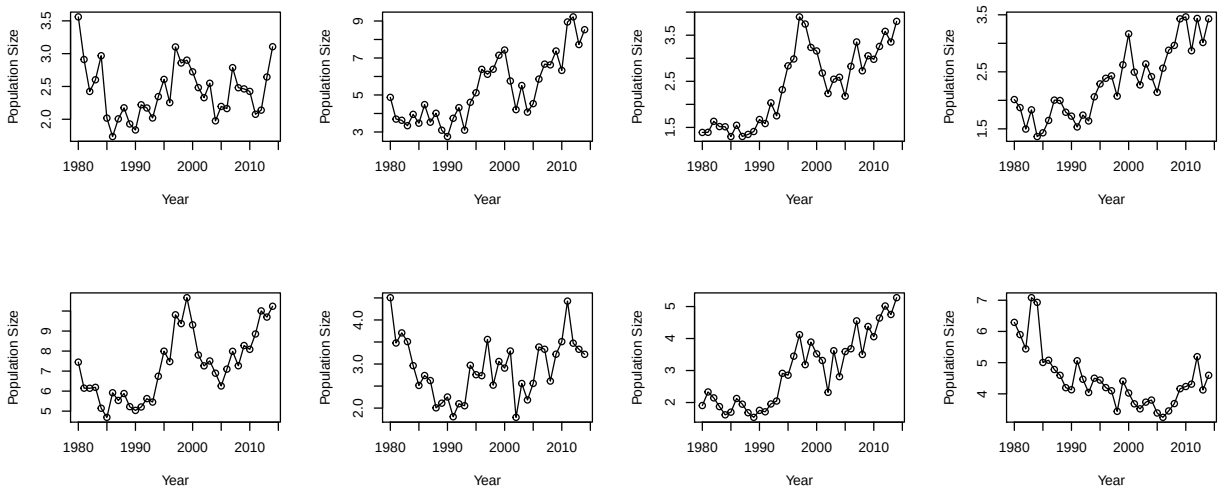
```
species <- names(dat)[-1]
par(mfrow=c(2,4))
for (i in 1:length(species)) {
    plot(dat[,species[i]] ~ dat$year, type='l', xlab='Year', ylab='Population Size')
}
```



Sample code

Change line type

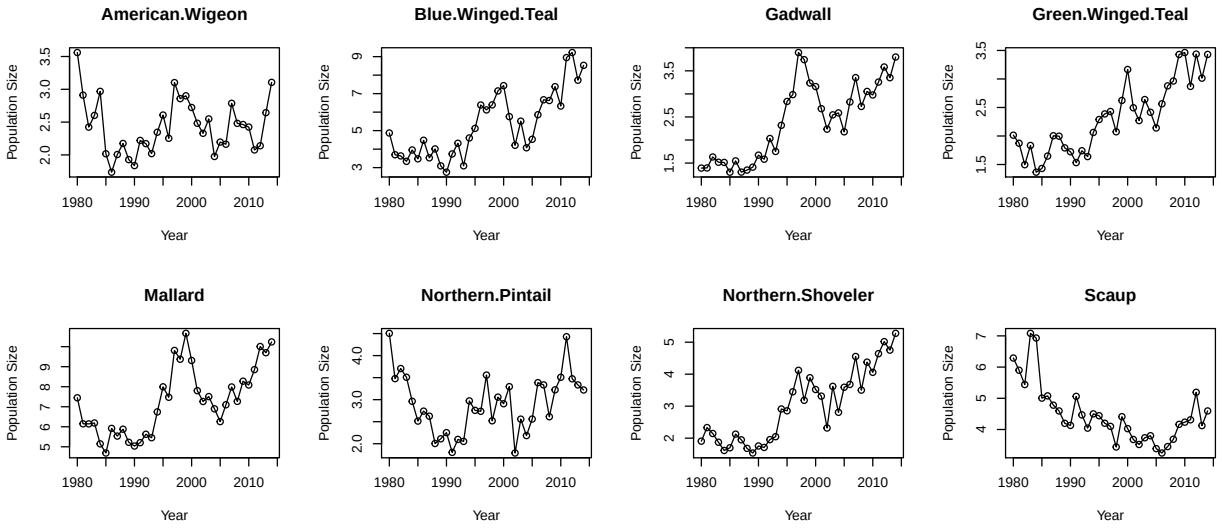
```
species <- names(dat)[-1]
par(mfrow=c(2,4))
for (i in 1:length(species)) {
    plot(dat[,species[i]] ~ dat$year, type='o', xlab='Year', ylab='Population Size')
}
```



Sample code

Add title

```
species <- names(dat)[-1]
par(mfrow=c(2,4))
for (i in 1:length(species)) {
    plot(dat[,species[i]] ~ dat$year, type='o', xlab='Year', ylab='Population Size',
         main=species[i])
}
```



Summary

- Loops repeat evaluation and execution
- Loops represent element-level thinking
- Loops can be used to prevent mistakes and achieve consistency

Thank you and see you next class