# Statistics 360: Advanced R for Data Science

## Lecture 3

Becky Lin

# Control Flow

# Control Flow

- Reading: text, chapter 5
- if/if-else, ifelse, switch
- for
- while
- break

# if and if-else

- `if` tests a condition and executes code if the condition is true. Optionally, can couple with an `else` to specify code to execute when condition is false.
- In short, `if(condition){ "true_action" }else{ "false_action" }`

```
if("cat" == "dog") {
  print("cat is dog")
} else {
  print("cat is not dog")
}
```

```
## [1] "cat is not dog"
```

# if returns a value

- The body of the if-else can evaluate expressions and store results, but note that if-else also returns a value.

```
cnd <- if("cat" == "dog")  "cat is dog" else "cat is not dog"
cnd
```

```
## [1] "cat is not dog"
```

# `if` expects a single logical

- most other inputs will cause an error
- **logical vectors will not throw an error**, but `if` will only use the first element

```
try(if("cat")  print("cat"))
## Error in if ("cat") print("cat") :
## argument is not interpretable as logical

if(c("cat"=="dog","cat" == "cat")) print("hello world")
## Warning in if (c("cat" == "dog", "cat" == "cat")) print("hello world"): the
## condition has length > 1 and only the first element will be used
```

Exercise (from text): predict the output

```
x <- 1:10
if (length(x)) "not empty" else "empty"
x - numeric()
if (length(x)) "not empty" else "empty"
```

# if-else if -else statement

The proper syntax to use `if-else if- else` statements:

```
if (condition1) {
  # do something
} else if (condition2) {
  # do something else
} else {
  # do default behavior
}
```

Exercise: What is the output of the following R code:

```
grade <- function(x) {
  if (x > 90) {
    "A"
  } else if (x > 80) {
    "B"
  } else if (x > 50) {
    "C"
  } else {
    "F"
  }
}
grade(78)
grade(80)
```

# ifelse(): vectorized if

- If you have a vector of conditions, instead of using a for-loop, use `ifelse()`.
- `ifelse()` can handle logical vectors
- syntax is condition, what to return if expression true, what to return if expression false

```
x <- 1:6
ifelse(x %% 2 == 0, "even","odd")
```

```
## [1] "odd"  "even" "odd"  "even" "odd"  "even"
```

# switch

- If you have multiple conditions to check, consider `switch` instead of repeated if-else; e.g.
  - `if(x==1) "cat" else if(x==2) "dog" else if (x==3) "mouse"`

```
x <- 2 # numeric argument
switch(x,"cat","dog","mouse") # evaluate the x'th element
```

```
## [1] "dog"
```

```
x <- "dog" #character argument
switch(x,
       cat="hi cat",
       dog="hi dog",
       mouse="hi mouse",
       warning("unknown animal")) # if we make it to the last condition
```

```
## [1] "hi dog"
```

```
switch("kangaroo",
       cat="hi cat",
       dog="hi dog",
       mouse="hi mouse",
       warning("unknown animal"))
```

```
## Warning: unknown animal
```

Exercise: predict the result

```
#switch() statement will fall-through to a result
switch_x <- function(x) {
  switch(x,
         a = ,
         b = ,
         c = 3,
         d = ,
         e = 4,
         stop("x not 1, 2, or 3"))
}
switch_x("a")
switch_x("d")
switch_x("f")
```

# for loops

- The basic for-loop

```
for(variable in a vector){
  do stuff
}
```

- Example

```
n <- 10;
nreps <- 100;
x <- vector(mode="numeric",length=nreps)

for(i in 1:nreps) {
  # Code you want to repeat nreps times
  x[i] <- mean(rnorm(n))
}
summary(x)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.57696 -0.16510  0.04930  0.06521  0.29345  0.86190
```

```
print(i)
```

```
## [1] 100
```

# for loop: next, break

- **`next`** in a for-loop exits the current iteration

```
for(i in 1:4){
  if(i==2) next
  print(i)
}
```

```
## [1] 1
## [1] 3
## [1] 4
```

- **`break`** in a for-loop exits the entire for loop.

```
for(i in 1:4){
  if(i==2) break
  print(i)
}
```

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

# Speed up `for` loop

Because of copy-on-modifying, the tip is to always preallocate output before using a for-loop

```
## BAD
x <- c()
system.time( for (i in 1:10000) x <- c(x, i) )
```

```
##    user  system elapsed
##   0.084   0.026   0.111
```

```
## GOOD
## NA_character_, NA_real_, NA_integer_, NA
x <- rep(NA_real_, length.out = 10000)
system.time( for (i in 1:10000) x[[i]] <- i )
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```
x <- rep(NA, length.out=10000)
system.time( for (i in 1:10000) x[[i]] <- i )
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

# for loop index set

- Index sets such as `1:n` are most common, but can be almost any atomic vector.

```
ind <- c("cat","dog","mouse")

# bad practice: it is harder to read and is less standard
for(i in ind) {
  print(paste("There is a",i,"in my house"))
}
```

```
## [1] "There is a cat in my house"
## [1] "There is a dog in my house"
## [1] "There is a mouse in my house"
```

```
# good practice: use seq_along()
for(i in seq_along(ind)) {
  print(paste("There is a",ind[[i]],"in my house"))
}
```

```
## [1] "There is a cat in my house"
## [1] "There is a dog in my house"
## [1] "There is a mouse in my house"
```

# seq_along

- A common use of for loops is to iterate over elements of a vector, say x.
- Try not to use `1:length(x)`. This will fail if x has length 0 (e.g., x is NULL).
- Instead use `seq_along()`

```
x <- NULL
for(i in 1:length(x)) print(x[i])
```

```
## NULL
## NULL
```

```
for(i in seq_along(x)) print(x[i])

is.na(x) <- NA
for(i in 1:length(x)) print(x[i])
```

```
## [1] NA
```

```
for(i in seq_along(x)) print(x[i])
```

```
## [1] NA
```

# Exercise: for loop

Q1: Why doesn't this for-loop go on forever?

```
xs <- c(1, 2, 3)
for (x in xs) {
  xs <- c(xs, x * 2)
}
xs
```

```
## [1] 1 2 3 2 4 6
```

Q2: Predict the output

```
for (i in 1:3) {
  i <- i * 2
  print(i)
}
```

Hint: In a for loop the index is updated in the beginning of each iteration

# while loops

- Use a `while` loop when you want to continue until some logical condition is met.

```
set.seed(1)
# Number of coin tosses until first success (geometric distn)
p <- 0.1
counter <- 0
success <- FALSE
while(!success)  {
  success <- as.logical(rbinom(n=1,size=1,prob=p))
  counter <- counter + 1
}
counter
```

```
## [1] 4
```

# break in a while loop

- break can be used to break out of a while loop.

```
i <- 1
while(i<5) {
  if(i==3) break
  cat("i = ",i,"\n")
  i <- i+1
}
```

```
## i =  1
## i =  2
```

# `repeat`

- `repeat` continues indefinitely until it encounters a `break`
- The text considers `repeat` to be the most flexible of `for`, `while` and `repeat`.

# Up next

- Reading: Text, chapter 6