Programming Topics I: Control Flow



Abbie M Popa
BSDS 100 - Intro to Data Science with R

Programming Topics



In the next few classes we will turn some of your informal knowledge of programming topics into a more formalized understanding of how to write code.

We will focus on two general programming topics

- Control Flow
- Writing Functions

Control Flow



Control flow allows execution of statements repetitively, while only executing other statements if certain conditions are met. Control flow can be subdivided into two main topics:

- Repetition and Looping
 - for () loops
 - while() loops
 - repeat () loops
- Conditional Execution
 - if()
 - if() {} else if() {} else {}
 - ifelse()
 - switch()



- Executes a statement repetitively until a variable's value is no longer contained in the sequence seq
- The generic in-line syntax is

```
for (var in seq) {expression}
```

A simple example which prints "BSDS 100" 5 times

```
for (i in 1:5) print("BSDS 100")
```

 When writing multiple line for loops you must include the curly brackets

```
for (i in 1:5) {
print(BSDS 100)
}
```



What is going on?

- The loop repeats for each item in seq, in the previous example, it will repeat for the number 1, then the number 2, then the number 3, then the number 4, then the number 5
- If the iterator variable (in this case i) is called it will change value with each loop, see

```
for(i in 1:5) print(paste("i equals", i))
```



It is possible to iterate over more complex sequences

```
> myVector <- factor(c("A", "A", "B", "C", "C", "C", "ZzZ"))
> for (k in levels(myVector)) print(k)
[1] "A"
[1] "B"
[1] "C"
[1] "ZzZ"
```



- You don't have to name the iterator variable (also known as the looping variable) i, but that is a common name for it
- If you using shortcuts to run your code be sure you are running from the start or end of the loop, or select the whole thing to run
- You can also "nest" loops, put one inside of another

7/33



- The seq in a for () loop is evaluated at the start of the loop;
 changing it subsequently does not affect the loop
- You can make an assignment to the looping variable (e.g., *i*) within the body of the loop, but this will not affect the next iteration
- When the loop terminates, the looping variable contains its latest value

8/33

Practice



Write a for loop to print the even numbers 2-10

The while () loop



- Executes a statement repetitively until a condition is no longer true
- The generic in-line syntax is

```
while (condition) {expression}
```

• A simple example which prints "BSDS" 3 times

```
> n <- 3
> while (n > 0) {print("BSDS"); n <- n - 1}</pre>
```

 Note: the use of the semi-colon is only required when writing more than one in-line expression. When multiple lines are used, the semi-colon can be omitted.

for() versus while()



- Note that in the for() loop the looping/iterator variable is evaluated at the beginning, so subsequent changes do not affect the loop
- In the while() loop, however, changes to the looping variable are critical to the behavior of the loop!
- What would happen if we forgot the second statement in the previous code? i.e.,

```
> n <- 3
> while (n > 0) print("BSDS")
```

The while () loop



This is why it is critically important to avoid infinite loops!!

The repeat () loop



- The repeat () loop can be used when the terminal condition does not apply at the top of the loop
- A repeat () loop must be terminated with a break command placed somewhere inside repeat () loop
- The break command immediately exits the innermost active for (), while () or repeat () loop

The repeat() loop



Example

```
> x <- 7
> repeat {
+    print(x)
+    x <- x + 2
+    if (x > 10) break
+ }
[1] 7
[1] 9
```

The repeat () loop



- repeat () loops also run the risk of infinite looping, make sure the break condition will be met!
- Often either a repeat () loop or a while () loop will work for the task at hand, pick the loop that leads to cleaner, more readable code



Which of the following loops are infinite?

- for(i in 1:1000000) print(i)
- 2 n <- 0
 while(n < 10){
 print(n)
 n <- n + 1 }</pre>

```
3 x <- 10
  repeat{
  print(x)
  x <- x - 2
  if(x == 1) break</pre>
```

Loops



Note: loops are slow! If they can be avoided (e.g., with vectorized code) they should be

The if() statement



- The if() control structure executes a statement if a given condition is true
- The generic in-line syntax is

```
if (condition) {expression}
```

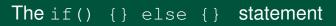
A simple example

```
> x <- 3
> if (x > 0) print(paste("x is: ", x, sep = ""))
[1] "x is: 3"
```

The if() statement



• The multi-line form for if () is





- The if() {} else {} control structure executes a statementif a given condition is true
- The generic in-line syntax is

```
if (condition) expression_01 else expression_02
```

A simple example

> x <- -3

```
> if (x > 0) print("x is positive") else print("x is negative")
[1] "x is negative"
```





• The multi-line form of if() {} else {} is

The above will run expressions if the condition is true, but will run alternate expressions if the condition is false.

Formatting Pitfalls



This code snippet will run without error

```
x <- -3

if (x > 0) {
   print(paste("x is: ", x, sep = ""))
   } else {
     print("x is negative")
   }

[1] "x is negative
```

Formatting Pitfalls



• The code snippet will throw an error

```
x <- -3

if (x > 0) {
   print(paste("x is: ", x, sep = ""))
  }

else {
   print("x is negative")
  }

Error: unexpected '}' in " }"
```

• the else must fall on the same line as the preceding if's }

A note on formatting



Though as we have seen it is possible to write an if() clause with an else() clause on one line, you will also note that it quickly becomes difficult to read. For this reason you must use the multiline version for your code in this class.





• The multi-line form of if() {} else if() {} else {} is

```
if (condition_01) {
    < expressions 01 >
} else if (condition_02) {
    < expressions 02 >
} else {
    < expressions 03 >
```

 As many else if () {} clauses may be chained (sequenced) together as desired

An else if() example

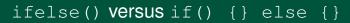


- in our previous example if we passed x < 0 we would still return "x is negative"
- we can repair this by adding an else if () clause
- when chaining ifs and elses it is important to account for all edge cases to avoid unexpected output!

Use of conditionals



- So far we have been checking a single object, which we already know the value of, against a conditional, which is a rather strange thing to do
- Generally you will put conditionals in loops (which we saw earlier) or functions (which we will see next class)
- e.g., check which of the values 1 to 10 are even





- If a vector x: |x| > 1 is passed to an if() statement, only the first element of the vector will be evaluated for conditional execution; moreover, R will throw a warning
- The ifelse() construct is a vectorized version of if() {}
 else {} which tests each element of a vector passed to it

ifelse() versus if() {} else {}



```
> x <- c(3, 2, 1)
> if ( x > 2) {print("first element in vector > 2")}
[1] "first element in vector > 2"
Warning message:
In if (x > 2) { :
   the condition has length > 1 and only the first element will be used
> ifelse(x > 2, ">2", "<=2")
[1] ">2" "<=2" "<=2"</pre>
```

The switch () function



- switch() chooses statements from a vector based on the value on an expression
- The multi-line form of switch () is

```
switch(expression,
  condition_01 = command_01,
  condition_02 = command_02,
  ...
  condition_n = command_n,
)
```

- If the expression passed to switch() is not a character, it is coerced to integer and that index from the switch is returned
- If the expression passed to switch() is a character string, then the string is matched exactly (with some small edge cases, see documentation)

The switch() function



```
grades <- c("A", "D", "F")
for (i in grades) {
  print(
    switch(i,
            "A" = "Well Done",
           "B" = "Alright",
            "C" = "C's get Degrees!",
           "D" = "Meh",
           "F" = "Uh-Oh"
[1] "Well Done"
[1] "Meh"
[1] "Uh-Oh"
```

The switch() function



Switches are primarily used when building functions, which we will cover in the next class!

In-Class Lab



Data located at:

https://raw.githubusercontent.com/abbiepopa/BSDS100/master/Data/titanic.csv

titanic.csv

- Using a for () loop and an if () conditional, recode the entries in the Survived variable with "Survived" and "Perished" into a new column survived_text
- ② Using the if() command and loop, create a new variable of type ordered factor in the data frame called ageClass, and map Age to: "Minor" if less than 18 yrs; 18 yrs ≤ "Adult" ≤ 65 yrs; and "Senior" if older than 65 yrs
- 3 Using a switch() statement, identify each passenger class, Pclass, as either "First Class", "Business Class" or "Economy", and print the results to the console