R Training part 2

Functions, Loops, and more on Sapelo2
Goals of Training

By the end of this training you should be able to:
● Use loops and conditional logic
● Use basic vectorization
● Create your own functions
● Find and replace text
● Monitor your code
● Use what you have learned to run a batch job on Sapelo2
Control structures

- **if** and **else**: testing a condition and acting on it
- **for**: execute a loop a fixed number of times
- **while**: execute a loop **while** a condition is true
- **repeat**: execute an infinite loop (must **break** out of it to stop)
- **break**: break the execution of a loop
- **next**: skip an iteration of a loop
if(<condition>){
  Do something
}

Can add else Clause

ifelse()

Multi if statements
Or
dplyr:case_when
For Loops

for(x in list, vector, etc){
Do something involving x}

If statement INSIDE a loops! & is used for “and” and | is used for “or”
While Loops

Careful about infinite loops!
Next and Break

Next skips iterations

```r
> for(num in 1:10){
    if( num < 4){next}
    print(num)
}
```

Output:
```
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Break exits a Loop

```r
> for(num in 1:10){
    if( num > 4){break}
    print(num)
}
```

Output:
```
[1] 1
[1] 2
[1] 3
[1] 4
```

Useful Built in DataSets

To see available datasets, run

data()

To load a dataset we can run

data(iris)
Vectorization
Allows efficient calculations to occur.
Vectorization is often built into functions.
Much Faster Than Loops!

```r
> testnumbers
[1]  1  2  3  4  5  6  7  8  9 10
> testnumbers + 3
[1]  4  5  6  7  8  9 10 11 12 13
>
> testnumbers[testnumbers<5]
[1]  1  2  3  4
>
> testnumbers < 5
[1]  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
Apply Functions

Input includes the dataframe, whether the function should be applied row or column wise and the name of the function.

Faster than Loops!

```r
> x = 1:10
> y = 1:10
> z = 1:10
> test = data.frame(x,y,z)
> test
   x y z
1  1 1 1
2  2 2 2
3  3 3 3
4  4 4 4
5  5 5 5
6  6 6 6
7  7 7 7
8  8 8 8
9  9 9 9
10 10 10 10
```

```r
> apply(test,1,mean)
 [1] 1 2 3 4 5 6 7 8 9 10
```

```r
> apply(test,2,mean)
   x y z
5.5 5.5 5.5
```

Create Functions inside apply function

```r
> apply(test,2, function(x) mean(x)-1)
   x y z
4.5 4.5 4.5
```
Functions

As you saw with apply(), a function can be an argument in another function! Functions can also be nested.

```r
<name> = function(arguments){
  Evaluate this code
}
```

The return value is the last Expression executed.

Can explicitly select return value With return()
Multiple arguments

```r
> my_third_function = function(x,y)
>     newlist = x*y
>     newlist
> my_third_function(1:3,3)
> [1] 3 6 9
>```

Default values

```r
> my_fourth_function = function(x,y=3)
>     newlist = x*y
>     newlist
> my_fourth_function(1:3)
> [1] 3 6 9
>```
Regular expressions

```r
> grep("e", testPeople$name)
[1] 1 3 4
> grepl("e", testPeople$name)
[1] TRUE FALSE TRUE TRUE
```

```r
> sub("e", "p", testPeople$name)
[1] "kpeko" "Anna" "Ellpn" "Lukp"
> gsub("e", "p", testPeople$name)
[1] "kppko" "Anna" "Ellpn" "Lukp"
```
Monitoring code

The traceback() function slows roughly where an error occurred
debug() function initiates an interactive debugger:

- \texttt{n} executes the current expression and moves to the next expression
- \texttt{c} continues execution of the function and does not stop until either an error or the function exits
- \texttt{q} quits the debugger

Use undebug() to turn off debugger
system.time() contains user time and elapsed time (time passing for user)