Tutorial 3 | Control Structures and Functions

Control Structures

Control sctructures allow you to control the flow of execution of the program. Some common examples of control structures include:

- if, else: testing a condition
- for: execute a loop a fixed number of times
- while: execute a loop while a condition is true
- repeat: execute an infinite loop
- break: break the execution of the loop
- next: skip an iteration of a loop
- return: exit a function

If-Else if(condition 1){
. . . do this
}else if(condition 2){
. . . do this
}else{
. . . do that
}
There is also a formulation that most of us aren't really used to and it looks like this:

```
x<-8;
y<-if(x>3){
  1
} else {
  0
}
y
```

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

I think it is pretty obvious what the program above is doing, but it is sometimes more useful to write if statements this way, especially when an entire if-else construct is build just about assigning a value to a certain variable.

```
x<- c("a","b","c")
for(i in 1:3){
  print(x[i]);
}
```

For loops

```
## [1] "a"
## [1] "b"
## [1] "c"
```

```
##and there is more sufficient way
for(i in seq_along(x)){
  print(x[i]);
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
```

```
##and also this one
for(letter in x){
  print(letter);
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
```

Loops in R just like in any over high-level language can be nested

```
i<-0;
while(i<10){
  print(i);
  i<-i+2;
}
```

While loops

```
## [1] 0
## [1] 2
## [1] 4
## [1] 6
## [1] 8
```

be careful to not allow it run infinitely, in other words check to see that the loop actually reaches the condition at which it should stop

Repeat, next and break Repeat is in a way very similar to the while loop, as it can go on forever, and actually it is used even less frequently than the while loop. Because is it as weel dangerous and way easier to miss specifying the exit condition, which may make you loop run forever.

```
i<-0;
repeat {
  i<-i+3;
```

```
  print(i);
  if(i>10){
    break
  }
}
```

```
## [1] 3
## [1] 6
## [1] 9
## [1] 12
```

Functions

Below you can see the way functions are written

```
adding <- function (a,b){
  a+b
}
adding(4,5)
```

```
## [1] 9
```

Functions in R are "first class objects", meaning that they can be treated much like any other object in R. Importantly:

- Functions can be passed as arguments for other functions
- Functions can be nested, so that you can define a function inside of another function

Functions have named arguments that can have default values, which makes it easier to adapt function for ither users

- The formal arguments are the arguments included in the function definition
- The *formals* function return a list of all the formal arguments of a functions
- Not every function call in R makes use of all the formal arguments
- Function arguments can be missing or might have defalut values
- When calling a function, if the input arguments are named they are not supposed to be called in the special order

Interesting feature of functions in R is that their arguments are evaluated using so-called *laxy evaluation*, meaning that if the argument is not used in the body of the function it doesn't have a need to be specified at all when calling the fucntion. For example:

```
fun <- function (a,b){
  a^2
}
fun(3)
```

```
## [1] 9
```

See? The function itself never actually uses the argument b, so calling fun(3) will not produce any error because the 2 gets positionally matched to a, and b is never evaluated

Scoping Rules

Scoping rules determine how a value is associated with a free variable in a function. R uses *lexical scoping* or *static scoping* (a common alternative is a dynamic scoping)

Lexical Scoping in R means that the values of free variables are searched for in the environment in which the function was defined. For example:

```
x<-8
some_fun <- function(a,b){
  (a+b)*x
}
some_fun(1,2)
```

```
## [1] 24
```

So this code above works even though x wasn't defined anywere in the function (it is a free variable), but it was previously defined while we called other functions, so it is now "saved" in the environment.