CT5102: Programming for Data Analytics

Lecture 3: Programming Structures, Lists & Apply Functions

Dr. Jim Duggan,

School of Engineering & Informatics

National University of Ireland Galway.

https://github.com/JimDuggan/PDAR

https://twitter.com/_jimduggan

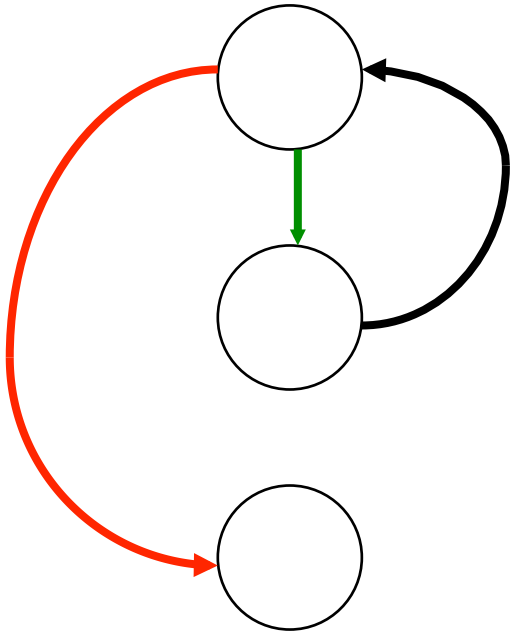
Overview

- R is a block-structured language, where blocks are delineated by {}
- Statements separated by newline characters, or with semicolon
- Variables are not declared (similar to JavaScript)

- Control Statements
- Arithmetic and Boolean Operators

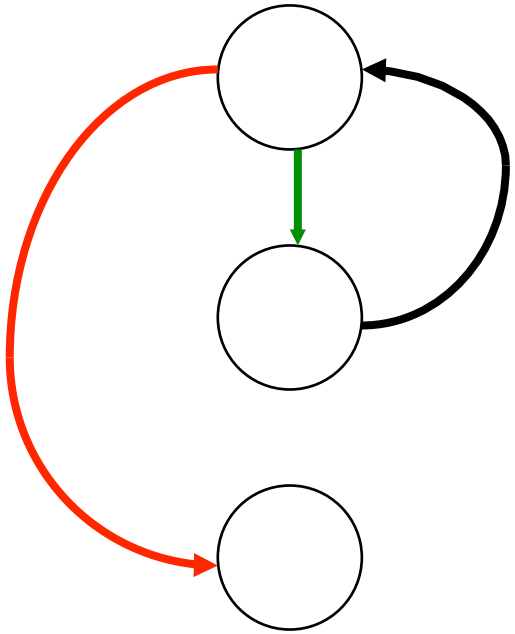
Loops - for

```
> x<-c(5,34,89)
>
> for(n in x){
+   print(n^2)
+ }
[1] 25
[1] 1156
[1] 7921
```



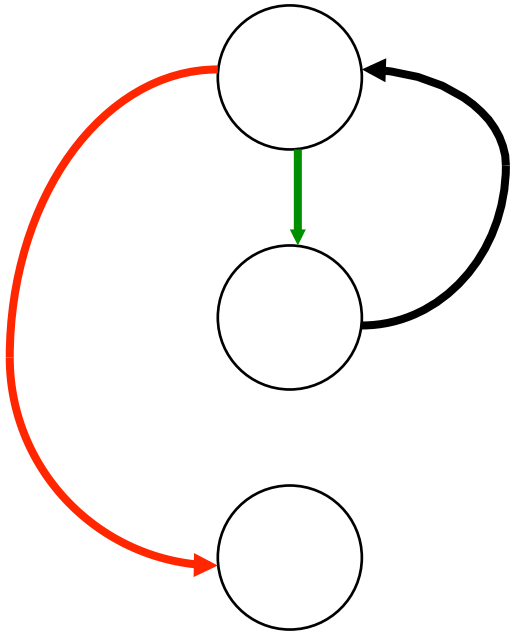
Loops - for

```
> x<-c(5,34,89)
>
> for(n in 1:length(x)){
+   print(x[n]^2)
+ }
[1] 25
[1] 1156
[1] 7921
```



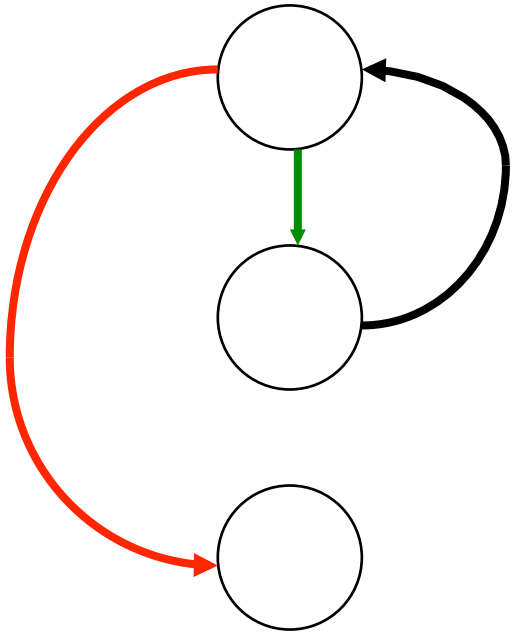
Loops - while

```
> i<-1
> while(i<=length(x)){
+   print(x[i]^2)
+   i<-i+1
+ }
[1] 25
[1] 1156
[1] 7921
```



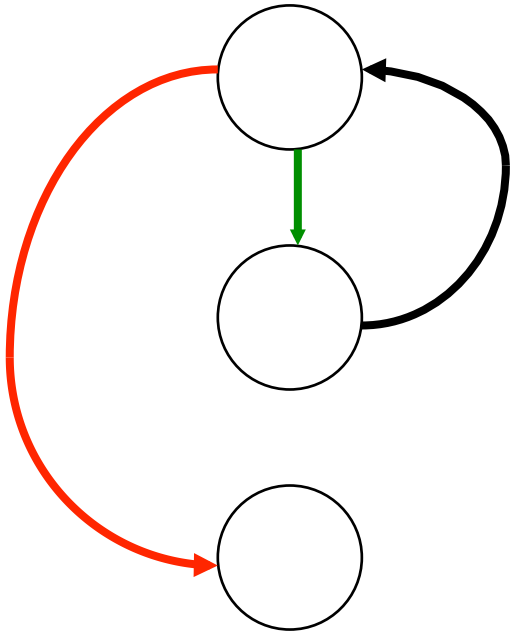
Loops - while

```
> i<-1
> while(TRUE){
+   print(x[i]^2)
+   i<-i+1
+   if(i>length(x)) break
+ }
[1] 25
[1] 1156
[1] 7921
```



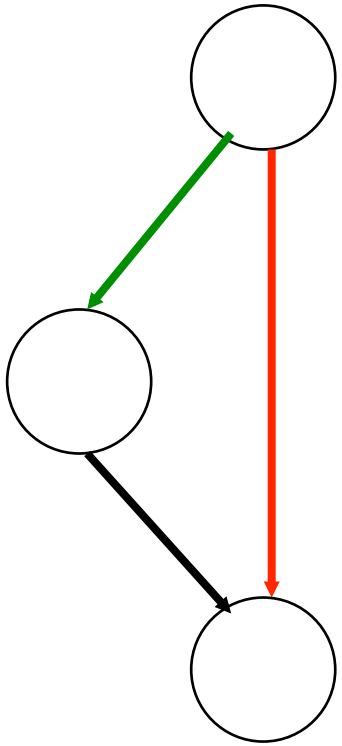
Loops – repeat (no condition)

```
> i<-1
> repeat {
+    print(x[i]^2)
+    i<-i+1
+    if(i>length(x)) break
+ }
[1] 25
[1] 1156
[1] 7921
```



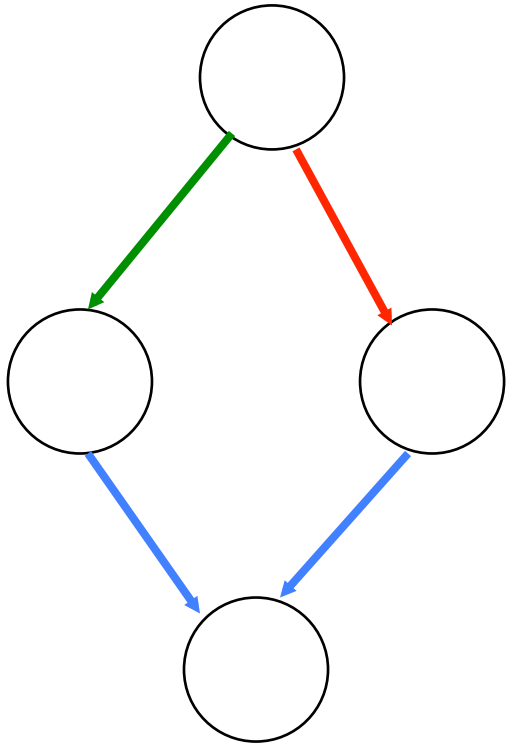
if

```
> x<-10
>
> if(x %% 2 == 0){
+    print("Even number...")
+ }
[1] "Even number..."
```



if else

```
> x<-11
> if(x %% 2 == 0){
+   print("Even number...")
+ } else
+   {
+   print("Odd number...")
+ }
[1] "Odd number..."
```



It is important to note that else must be in the same line as the closing braces of the if statements.

http://www.programiz.com/r-programming/if-else-statement

if else if

```
> x<-0
> if(x<0){
+    print("Negative number")
+ } else if (x > 0){
+    print("Positive number")
+ } else {
+    print("Zero!")
+ }
[1] "Zero!"
```

Challenge 3.1

- Implement the following decision table in an R function called getCost(age, card)

Conditions	Rules				
< 5 years	✓	×	×	×	×
>= 5 and < 18	×	✓	×	×	×
>= 18 and < 55 with concession card	×	×	✓	×	×
>= 18 and < 55 no concession card	×	×	×	✓	×
>= 55	×	×	×	×	✓
Actions					
Free Admission	✓	×	×	×	×
$8.00	×	✓	✓	×	×
$12.00	×	×	×	✓	×
$6.00	×	×	×	×	✓

http://hsc.csu.edu.au/ipt/project_work/3287/design_tools.htm

Solution

```
cost<-function(age, card=F){
  if(age < 5){
    return (0)
    } else if(age < 18){
       return (8)
    } else if(age < 55 && card){
      return (8)
    } else if (age < 55 && !card){
      return (12)
    } else{
      return (6)
    }
}
```

```
> cost(1)
[1] 0
> cost(6)
[1] 8
> cost(18)
[1] 12
> cost(1)
[1] 0
> cost(6)
[1] 8
> cost(17)
[1] 8
> cost(24)
[1] 12
> cost(24,T)
[1] 8
> cost(56)
[1] 6
```

R – Data Types

	Homogenous	Heterogenous
1d	Atomic Vector	**List**
2d	Matrix	Data Frame
nd	Array	

- To understand a data structure, use the str() function

```
> x
[1] 1 2 3 4 5
> str(x)
 int [1:5] 1 2 3 4 5
```

List in R

- Lists are different from atomic vectors because their elements can be of any type, including lists.
- list() creates a list, instead of c()

```
>
> x<- list(1:3, "a", c(T,F,T), c(2.3, 5.9))
>
> str(x)
List of 4
 $ : int [1:3] 1 2 3
 $ : chr "a"
 $ : logi [1:3] TRUE FALSE TRUE
 $ : num [1:2] 2.3 5.9
```

Using c()

- c() will combine several lists into one
- c() will coerce atomic vectors to a list before combining them

```
> str(x)
List of 4
 $ : int [1:3] 1 2 3
 $ : chr "a"
 $ : logi [1:3] TRUE FALSE TRUE
 $ : num [1:2] 2.3 5.9
>
> y <- c(1:3, 2:4)
>
> str(y)
 int [1:6] 1 2 3 2 3 4
```

```
> z<-c(x,y)
>
> str(z)
List of 10
 $ : int [1:3] 1 2 3
 $ : chr "a"
 $ : logi [1:3] TRUE FALSE TRUE
 $ : num [1:2] 2.3 5.9
 $ : int 1
 $ : int 2
 $ : int 3
 $ : int 2
 $ : int 3
 $ : int 4
```

Subsetting lists

- Works in the same way as subsetting an atomic vector
- Using [will always return a list
- [[and $ pull out the contents of a list
- To get the contents, you need [[

If list x is a train carrying objects, then x[[5]] is the object in car 5, x[4:6] is a train of cars 4-6" @RLangTip

Example

```
> x<- list(1:3, let="a", c(T,F,T))
>
> x
[[1]]
[1] 1 2 3

$let
[1] "a"

[[3]]
[1]  TRUE FALSE  TRUE
```

```
> x[1]
[[1]]
[1] 1 2 3

> str(x[1])
List of 1
 $ : int [1:3] 1 2 3

>
> x[[1]]
[1] 1 2 3
>
> str(x[[1]])
 int [1:3] 1 2 3
```

Simplifying vs Preserving Subsetting

- Simplifying subsets returns the simplest possible data structure that can represent the output
- Preserving subsetting keeps the structure of the input the same as the output.
- Omitting drop = FALSE when subsetting matrices and data frames is a common source of programming error.

	Simplifying	Preserving
Vector	x[[1]]	x[1]
List	x[[1]]	x[1]

$ operator

- $ is a shorthand operator, where x$y is equivalent to **x[[“y”,exact=FALSE]]**
- Often used to access variables in a data frame
- $ does partial matching

```
> x
[[1]]
[1] 1 2 3

$let
[1] "a"

[[3]]
[1]  TRUE FALSE  TRUE

> x$let
[1] "a"
>
> x$l
[1] "a"
```

Challenge 3.2

- Write a function that takes in a vector of numbers and returns a list containing the:
  - Minimum
  - Maximum
  - Median
  - Standard deviation
  - Range

sapply(x,f)

- Another use of user-defined functions in R is as parameters to the *apply* family of functions.
- The general form of the **sapply(x,f,fargs)** function is as follows:
  - **x** is the target vector or list
  - **f is the function to be called and applied to each element**
  - **fargs** are the optional set of arguments that can be applied to the function f.
  - **sapply()** returns a vector, **lapply()** returns a list

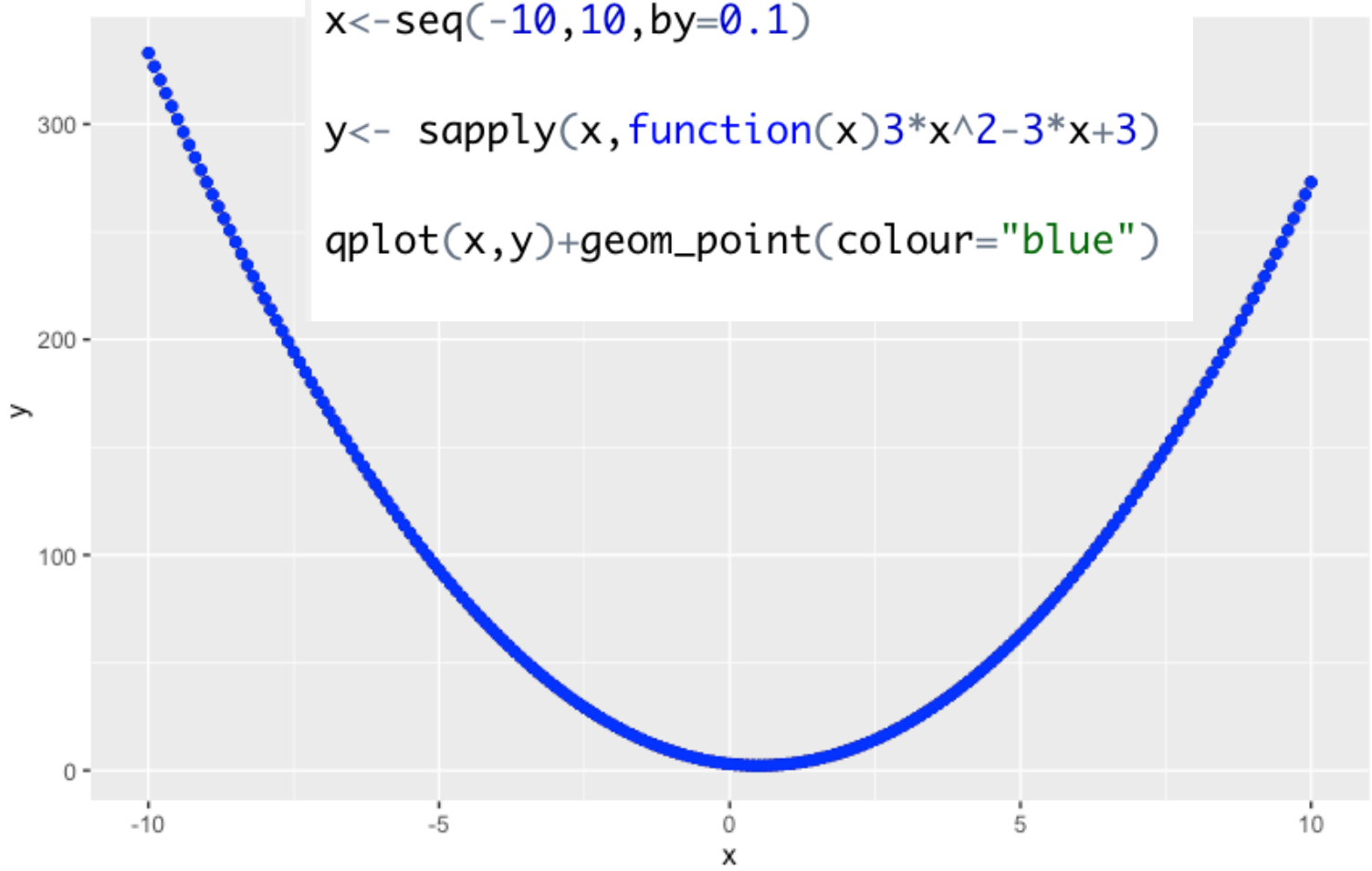
Example

```
library(ggplot2)

x<-seq(-10,10,by=0.1)

y<- sapply(x,function(x)3*x^2-3*x+3)

qplot(x,y)+geom_point(colour="blue")
```



Challenge 3.3

- Modify this example so that the quadratic parameters *a, b* and *c* are sent to the sapply() function.

$$f(x) = ax^2 + bx + c$$

Variable number of arguments

- Functions can take a variable number of arguments
- The special argument name is "…"
- This can be converted to a list and processed in the function (more on lists later)

```
as_list <- function(...){
  list(...)
}

> as_list(1:2,3:4,c(T,F))
[[1]]
[1] 1 2

[[2]]
[1] 3 4

[[3]]
[1]  TRUE FALSE
```

Challenge 3.4

- Write a function that takes in a variable number of numeric vectors, combines them, and returns the overall mean, minimum, and maximum (output should be in a list).

- Hint: the *unlist()* function in R may be useful.

- Use the browser() call inside the function to examine the variables.

References

- Wickham, H. 2015. Advanced R. Taylor & Francis