UCLA Department of Statistics
Statistical Consulting Center

Intermediate R

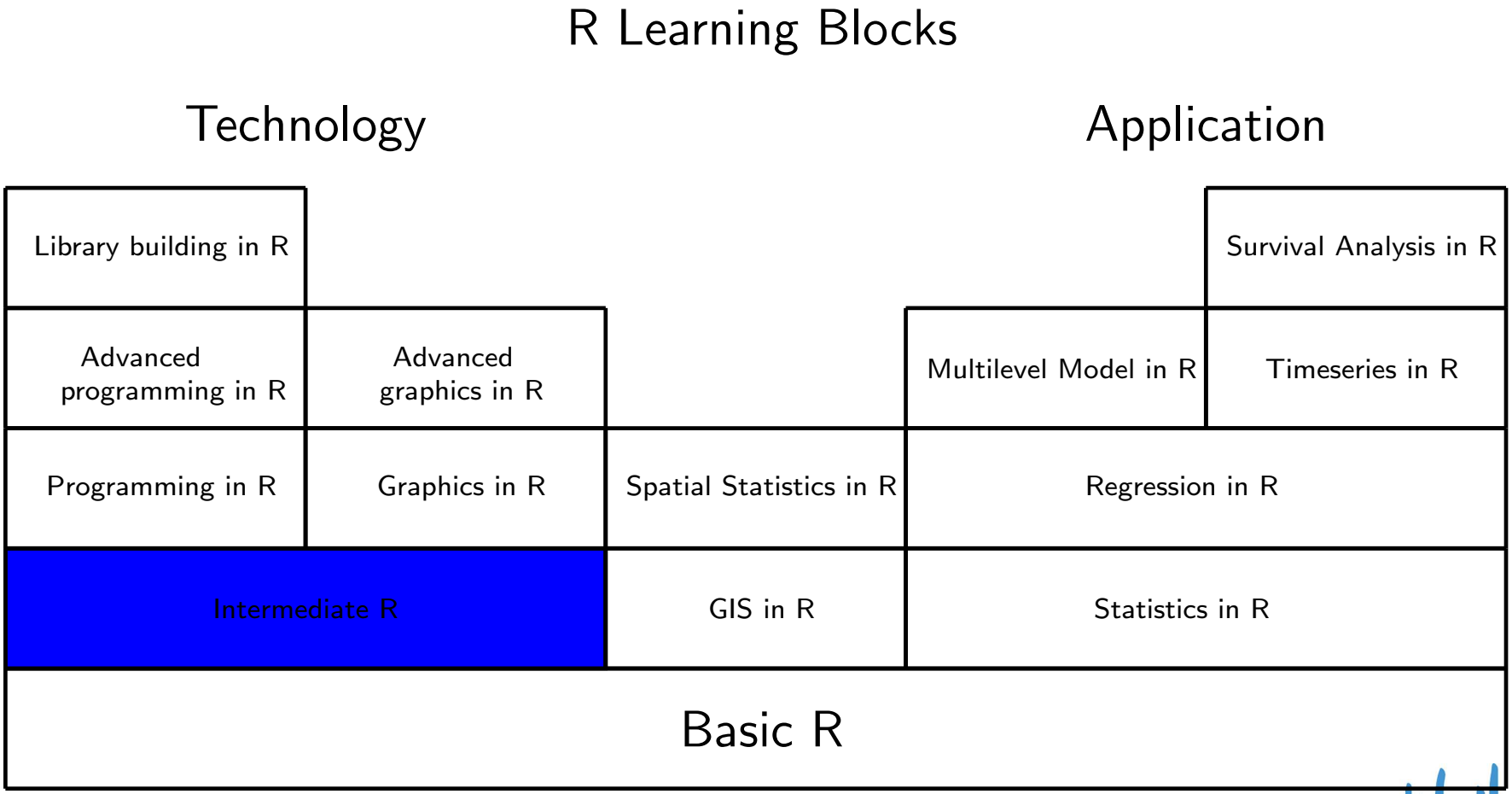
Masanao Yajima
yajima@stat.ucla.edu

May 28, 2009

Welcome to Intermediate R I

- This course is for people with basic knowledge of R.
  - You know how to install R, use R, look up help files in R,
  - You know how to work with vectors, matrices, and data,
  - You know a few functions,
  - You know how to make some plots, etc..
- Today we will try to add a few more important concepts, while applying them to graphics and function which are also new.
- It's an ambitious plan, but I promise you it will be fun!
- Also, we will do many activities on the way, so get ready...
- Any questions or comments before we begin?

Welcome to Intermediate R II



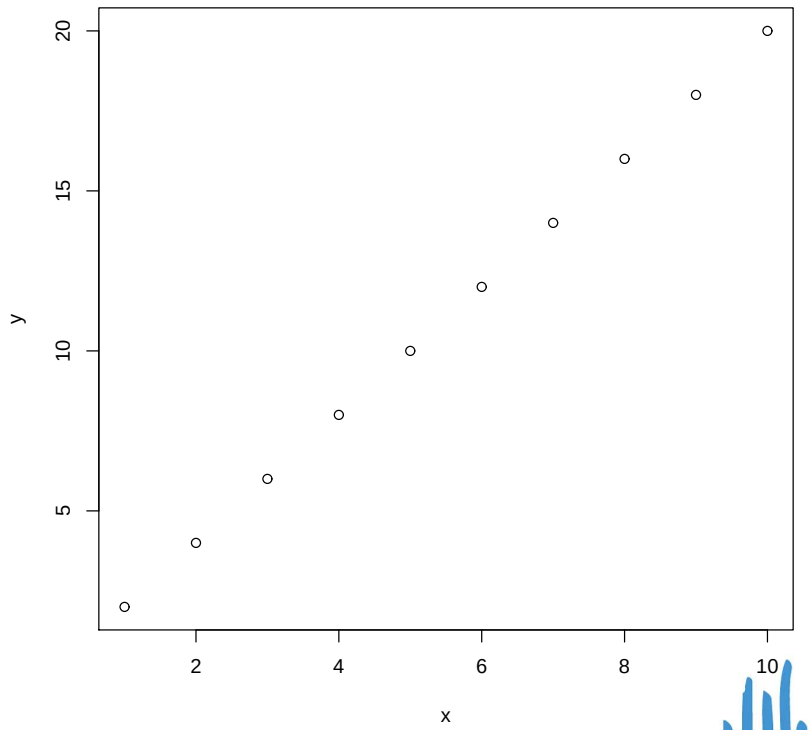
Lets start with quick review exercises

1. Start up a R console
2. Create a new script
3. make a vector `x` as sequence of integer from 1 to 10.
   (hint: you can use `seq(1,10)` or `1:10`)
4. make a vector `y` as sequence of even integer from 2 to 20.
   (hint: you can use `seq(2,20,by=2)` or use `x`)
5. make a matrix `M` that has `x` and `y` as its columns.
   (hint: you can use `cbind(x,y)`)
6. look at the 5th row of `M` (hint: you can use subset M[,])
7. look at the 2nd column of `M` (hint: you can use subset M[,])
8. plot x and y using `plot(x,y)` what do you see?

Answer

Here is suggested answers to the problem:

```
> x <- seq(1,10,by=1)
> y <- 2*x
> M <- cbind(x,y)
> M[5,]
 x  y
 5 10
> M[,2]
 [1]  2  4  6  8 10 12 14 16 18 20
> plot(x,y)
```



I hope that was not too bad... Let us move on.

Part II

Working with plot

Basic Plot

We have jumped into making our first plot...
This is the syntax to create basic plot

```
plot( [x coordinates], [y coordinates], [options])
```

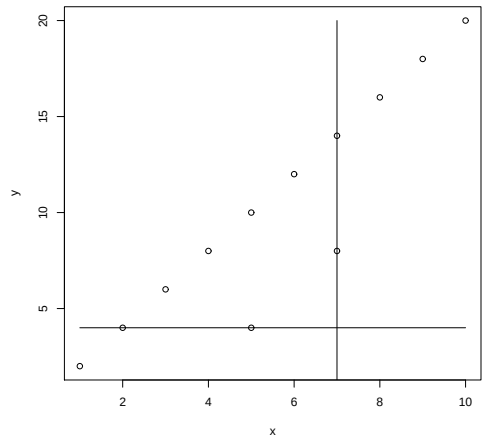
- There are many options that let you tailor your plot to your needs.
- Those of you who are interested, there should be mini course on graphics with R in the future.
- But for now we will use this simple plot to learn more about R.
- I will provide you with necessary options along the way.

Adding points and lines in to your plot

You can add points and lines where ever you want in the plot by calling `points` and `lines` function.

```
points( [x coordinate(s)], [y coordinate(s)])
lines( c([x start],[x end]), c([y start],[y end]))
```

```
plot(x,y)
points(c(5,7), c(4,8))
lines(c(1,10),c(4,4))
lines(c(7,7),c(1,20))
```



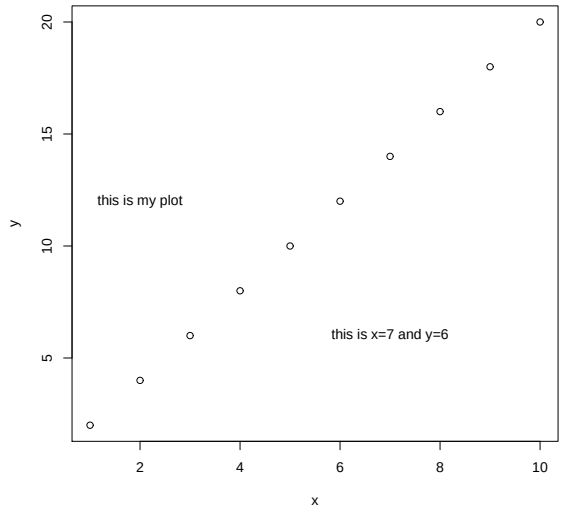
One thing to note is, you need to have a plot active before you add points or lines.

Adding text in to your plot

You can add texts into your plot also using the `text` function.

```
text( [x coordinates], [y coordinates],labels = [text])
```

```
plot(x,y)
text(2,12,"this is my plot")
text(7,6,"this is x=7 and y=6")
```



Experiment with lines, points, and text

- Now its your turn, add some text, points, and lines into your plot.

Loop

Lets say you want to add many lines into your plot. You can write the lines function again and again.... or use looping in R. There are 3 loops in R.

```
for( [index name] in [vector of index] ) {
  [What you want to do repeatedly]
}

while( [statement that is TRUE/FALSE] ) {
  [What you want to do repeatedly]
}

repeat {
  [What you want to do repeatedly]
}
```

Example of a for loop

Here is a simple example of for loop

```
> for( i in 1:10 ) {
+   print(i)
+ }

[1] 1
[1] 2
[1] 3
.....
[1] 9
[1] 10
```

Index i takes on each value in vector of 1 to 10 with every iteration.

Another example of for loop

You can loop nest another loop with in a loop

```
> for( i in 1:3 ) {
+   for( j in 1:2 ) {
+     print(c(i,j))
+   }
+ }

[1] 1 1
[1] 1 2
[1] 2 1
....
[1] 3 2
```

For each value of that index i takes, index j takes values 1 and 2.

Yet another example of for loop

Index does not have to always take numerical values.

```
> for( i in c("hi", "my name is", "R") ) {
+   print(i)
+ }

[1] "hi"
[1] "my name is"
[1] "R"
```

Example of while loop

While loop is useful when terminating loop when condition is met.

```
> x <- 1
> while(x < 3){
+   print(x)
+   x <- x+1
+ }
[1] 1
[1] 2
```

Example of repeat loop

- repeat loop is similar to while loop because you have to specify the terminating condition your self.
- Otherwise it keeps looping until you hit Ctrl + c.
- termination is done by break command.

```
> x <- 1
> repeat{
+   print(x)
+   x<-x+1
+   if(x>=3) { break }
+ }
[1] 1
[1] 2
```

if is a conditional statement. We will get to it in the next section.

Exercise using loop

Lets use loop to add grid lines into a plot. How would you do this?

Exercise using loop

Lets use loop to add grid lines into a plot. How would you do this? Here is one approach,

1. initialize your plot

```
plot(x,y,type="n")
```

2. add vertical lines

```
for(i in 1:10){
  lines(c(i,i),c(1,20))
}
```

3. add horizontal lines

```
for(j in 1:20){
  lines(c(1,10),c(j,j))
}
```

Exercise using loop

Lets use loop to add grid lines into a plot. How would you do this? Here is one approach,

1. initialize your plot

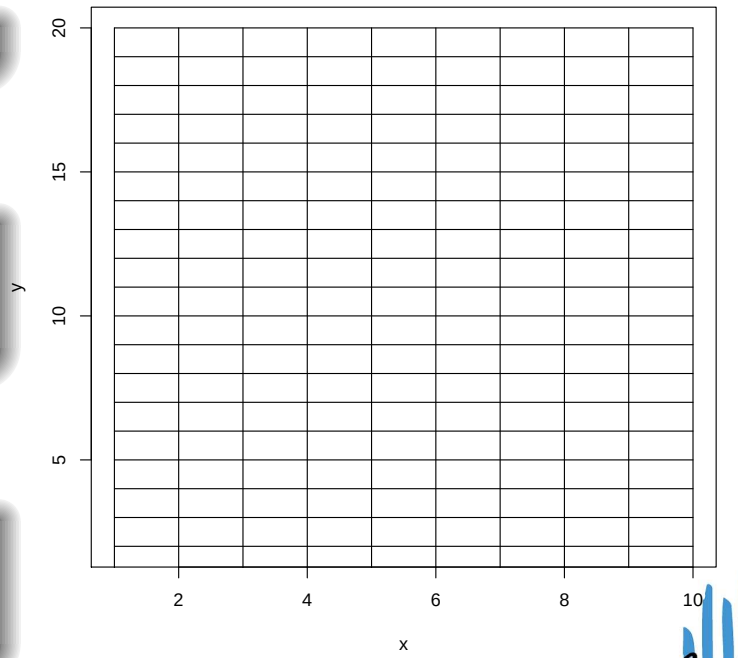
```
plot(x,y,type="n")
```

2. add vertical lines

```
for(i in 1:10){
  lines(c(i,i),c(1,20))
}
```

3. add horizontal lines

```
for(j in 1:20){
  lines(c(1,10),c(j,j))
}
```



Review question

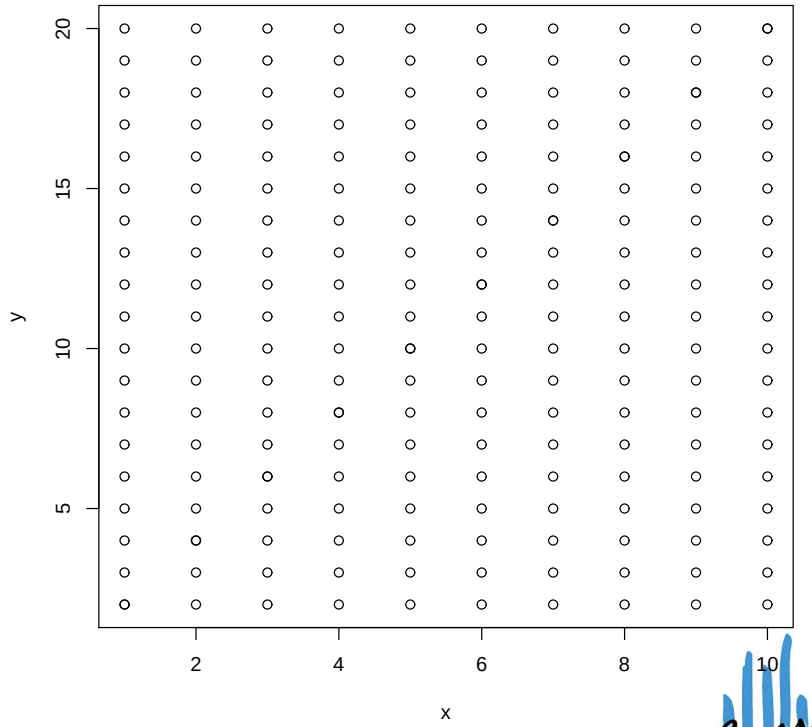
Do you see what this command will do?

```
plot(x,y)
for(i in 1:10){
  for(j in 1:20){
    points(i,j)
  }
}
```

Review question

Do you see what this command will do?

```
plot(x,y)
for(i in 1:10){
  for(j in 1:20){
    points(i,j)
  }
}
```



What if...

- what if we want to change the color of the lines for even and odd number. We need a new tool for that.
- We will study conditional statements.
- There are 2 conditional statements, if and ifelse.

```
if([1st condition]){
  [executed if 1st condition is TRUE]
} else if([2nd condition]){
  [executed if 2nd condition is TRUE]
} else if ([3rd condition]){
....
} else{
  [executed if all other conditions are FALSE]
}

ifelse([vector condition], [return value for TRUE]
                         , [return value for FALSE])
```

Example of if statement

```
a <- 2
if( a > 4 ){
  print("bigger than 4")
} else if( a <=1) {
  print("less than or equal to 1")
}else {
  print("between 1 and 4")
}
[1] "between 1 and 4"
```

If saw a is not bigger than 4, then it goes to else if and see it is not less than or equal to 1, so it returned statement in else.

Example of ifelse statement

Problem with if is it takes only 1 value at a time. ifelse is a useful alternative if you are working with a vector.

```
b <- c(1,3,5,6)
ifelse( b > 4,  "bigger than 4","less than or equal to 4")

[1] "less than or equal to 4"
[2] "less than or equal to 4"
[3] "bigger than 4"
[4] "bigger than 4"
```

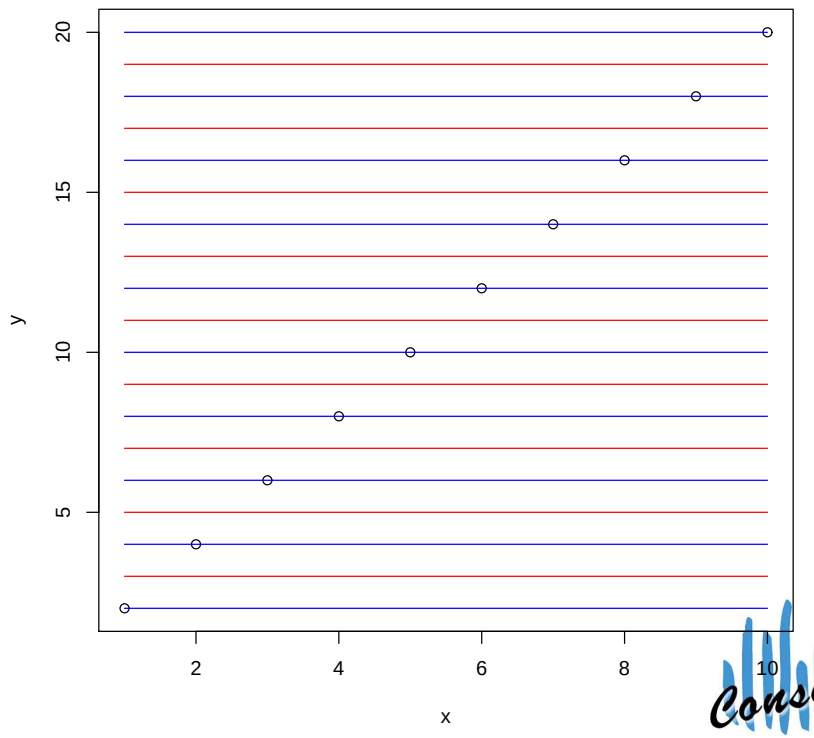
Lets get back to our plot

We want to alternate colors for odd and even number lines. How can we do that?

Lets get back to our plot

We want to alternate colors for odd and even number lines. How can we do that?

```
plot(x,y)
for(j in 1:20){
  color <- if( j %% 2 ==0 ){ "blue" }
             else { "red"}
  lines( c(1,10), c(j,j), col=color )
}
```



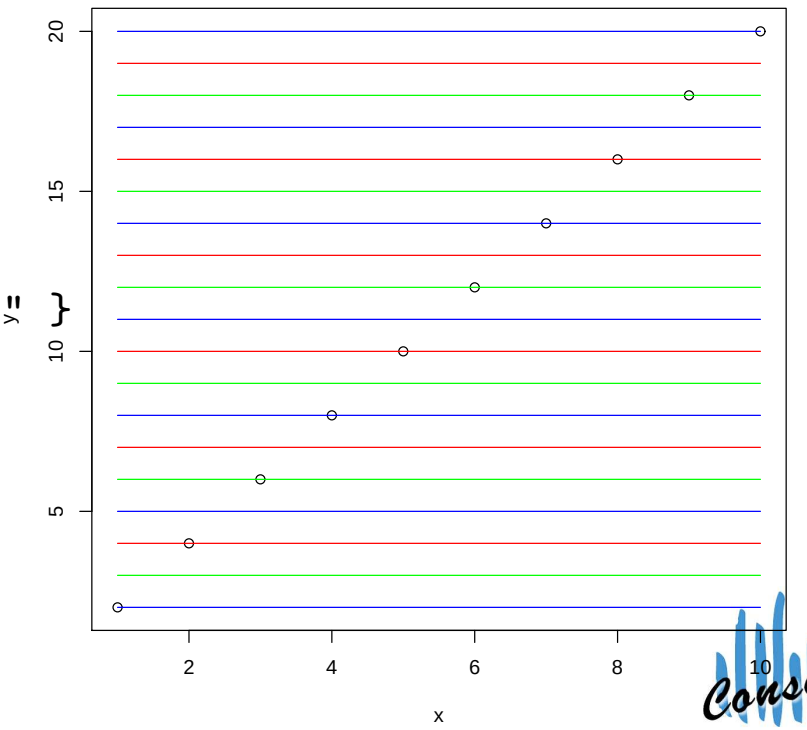
Exercise with conditional statement

Can you make line color alternate between "red", "blue", and "green"?

Exercise with conditional statement

Can you make line color alternate between "red", "blue", and "green"?

```
plot(x,y)
for(j in 1:20){
  color <- if( j %% 3 ==1 ){ "red" }
           else if( j %% 3 ==2 ){ "blue" }
           else { "green"}
  lines( c(1,10), c(j,j), col=color )
}
```



Working with characters

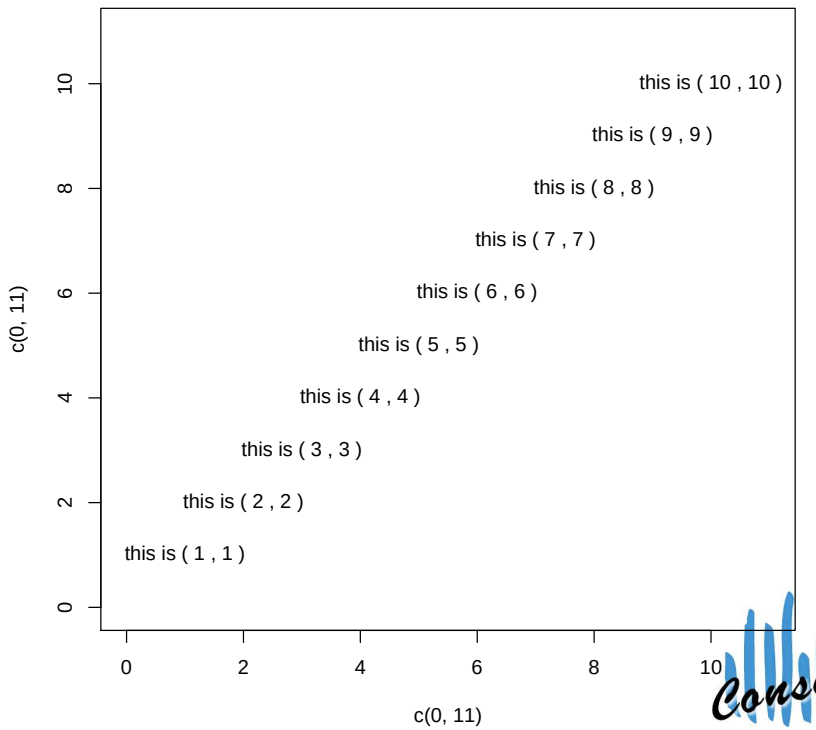
- R has functions that lets you work with character strings
  - `paste()` which pastes together the vector into one string
  - `substr()` which cuts out some portion of the string
  - `gsub()` which substitutes part of character string with some other string

```
> words <- paste("hi", "my", "name", "is", "R")

> words
[1] "hi my name is R"

> substr(words, 3,10)
[1] " my name"

> gsub("my","our",words)
[1] "hi our name is R"
```

An example with words

Paste is convenient when you want to make labels dynamically. Here is an example:

```
plot(c(0,11),c(0,11),type="n")
for( i in 1:10){
 text(i, i, paste("this is (",i,",",i,")"))
}
```



Part VI

Working with function

Why make a function?

Up to now, we've used functions that someone else has written for us. Most of the time that will do the job.

- But, sometimes there is no function available to do the job,
- Or the function available does not give you exactly what you want,
- Or its written so poorly you want to make a better version of it,
- Or you simply want to reuse what you did many times with different parameters,
- and many more...

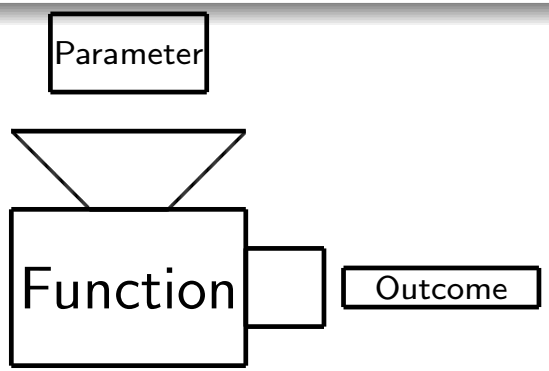
Function

Making function is first step into real programming, but you need not be afraid. It's as easy as this:

```
[name of function] <- function( [parameter(s)] ) {
  [body of function]
 return( [result of function to return] )
}
```

and you can call it by

```
[variable to store result] <- [name of function]( [parameter(s)to give it] )
```



Lets make your first function

- We will follow the tradition of programming world and write a "hello world" function.

```
helloWorld <- function(){
    return("hello world!!")
}
```

And call it

```
helloWorld()
```

Do you see what you expected to see?

Function: Parameter

There are few ways to specify a parameter when you call a function

- by order
- by name

```
> pfun <- function(x,y){ cat("x is", x, "y is", y,"\n")}
> pfun( 1, 2)
x is 1 y is 2
> pfun( y=2, x=1)
x is 1 y is 2
```

Function: Default value

You can set a default value to the parameters also

```
> pfun2 <- function(x,y=2){ cat("x is", x, "y is", y,"\n")}
> pfun2( 1 )
x is 1 y is 2
> pfun2( y=4, x=1)
x is 1 y is 4
```

Function: Dynamic programming

You can also pass a function as parameter.

```
> ffun <- function( func=mean, x ){ func(x)  }
> ffun(x=1:10)
[1] 5.5
> ffun(min,x=1:10)
[1] 1
> ffun(max,x=1:10)
[1] 10
```

This is just some of the cool things you can do with R. You can learn more in the programming with R mini course.

Function: scope of variable

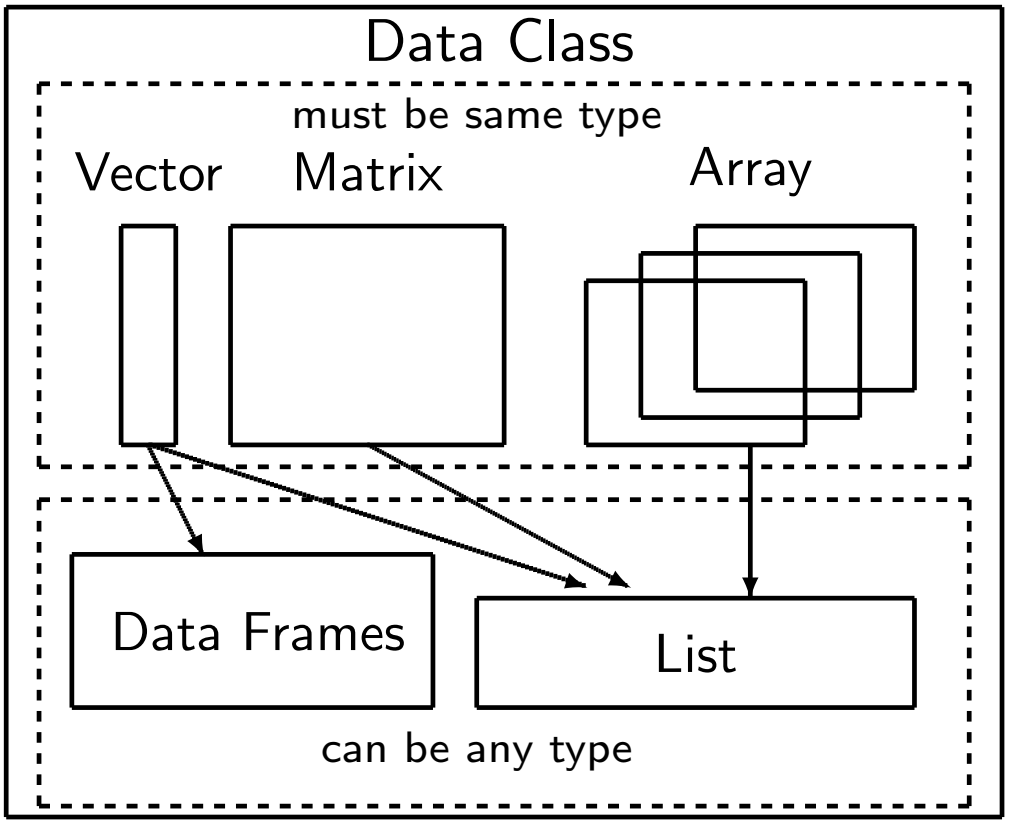
What you do in function stays in function. This is incorrect statement but at this level, you do not have to worry too much about it.

```
> x<-1
> testx <- function( val ) { x <- val;  print(x) }
> testx( 200 )
[1] 200
> print(x)
[1] 1
```

As you can see x in the environment did not get updated even though inside the function x was set to 200.

more data objects...

So far we have been working with vector, matrix, and data frames. There are other useful data objects in R.



Vector, Matrix, and Arrays

vector, matrix, and arrays are all same things. They are called differently just for practical reasons. Let me illustrate what that means. First a matrix is vector + dimension.

```
> avector<-1:12
> avector
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> amatrix <- avector
> amatrix
 [1]  1  2  3  4  5  6  7  8  9 10 11 12
> dim(amatrix)<-c(3,4)
> amatrix
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> amatrix[2,2]
[1] 5
> amatrix[5]
[1] 5
> avector[5]
[1] 5
```

Vector, Matrix, and Arrays

Similarly you can think of array as just multi dimensional matrix.

```
> aarray <-array(1,c(2,2,3))
> arrayormat<-aarray[,,1,drop=TRUE]
> arrayormat
     [,1] [,2]
[1,]    1    1
[2,]    1    1
> class(arrayormat)
[1] "matrix"
```

Here `drop=TRUE` allows R to drop the 3rd dimension, which turns array into matrix.

List

List is something different. You can have a list of vectors, a list of matrix, and even a list of list. The biggest difference is that you can have many different types of objects in one list.

```
> list1 <- list(a=1,b=c(1,2),c=matrix(0,2,2),d=list(a="hi",b="ho"))
> list1
$a
[1] 1
$b
[1] 1 2

$c
     [,1] [,2]
[1,]    0    0
[2,]    0    0

$d
$d$a
[1] "hi"
$d$b
[1] "ho"
```

Data frame is one special type of list where each of the element is a vector of same length.

Creating List

There are 2 ways to create an empty list

```
> list2 <- list()
> list2
list()

> list3 <- vector("list",3)
> list3
[[1]]
NULL

[[2]]
NULL

[[3]]
NULL
```

Unless you definitely do not know the length of the list, you should always choose latter one from programming perspective. We will not go into the details in this course.

List names

You can name each element of the list

```
> list4<- vector("list",2)
> names(list4)<-c("e1","e2")
> list4
$e1
NULL

$e2
NULL
```

List index

- Accessing element in list is a little different from other objects.
- You could think of it in 2 steps
  1. First, use `$variable name` or `[[]]` to access an object in the list
  2. then you can use regular indexing to access elements within that object.

```
> list1 <- list(a=1,b=c(1,2),c=matrix(0,2,2),d=list(a="hi",b="ho"))
> list1$c
     [,1] [,2]
[1,]    0    0
[2,]    0    0

> list1[[3]]
     [,1] [,2]
[1,]    0    0
[2,]    0    0

> list1[[3]][1,1]
[1] 0

> list1[["b"]]
[1] 1 2
```

List as return value

Getting back to functions.

- there will be times when you want to return more than just one variable.
- problem is, `return` only lets you return one thing.
- This is where list comes in handy.

```
> splitat5<- function(x){
+   less <- x[x<=5]
+   more <- x[x>5]
+   return(list(less=less,more=more))
+ }
> result <- splitat5(1:10)
> result$less
[1] 1 2 3 4 5
> result$more
[1]  6  7  8  9 10
```

Exercise using list

Let us write a function that takes 1 argument which is vector of numbers. Then it returns a list with 2 elements, one is odd numbers and one is even numbers.

Exercise using list

Let us write a function that takes 1 argument which is vector of numbers. Then it returns a list with 2 elements, one is odd numbers and one is even numbers.

```
sepOddEven <- function( numbers ){
  even<-numbers[numbers %%2==0]
  odd <-numbers[numbers %%2!=0]
  return(list(even=even, odd=odd))
}
```

Exercise using list

Let us write a function that takes 1 argument which is vector of numbers. Then it returns a list with 2 elements, one is odd numbers and one is even numbers.

```
sepOddEven <- function( numbers ){
  even<-numbers[numbers %%2==0]
  odd <-numbers[numbers %%2!=0]
  return(list(even=even, odd=odd))
}
```

```
> sepOddEven(1:10)
$even
[1]  2  4  6  8 10

$odd
[1] 1 3 5 7 9
```

Logical operation in R

In R logical values are coded as TRUE and FALSE to indicate true and false. Note that they are capital letters. Some people use T and F for short, but this is not a good practice.
Here are some of the logical operators you may use.

$\&, \&\&$	– AND
$\|, \|\|$	– OR
xor()	– exclusive OR
$<, <=, >, >=$	– smaller/bigger or smaller/bigger than or equal to
$==$	– equal
$!=$	– Not equal
any()	– TRUE if any element is TRUE
all()	– TRUE when all of element is TRUE

Logical operation in R

```
> x<-1:5
> x<4
[1]  TRUE  TRUE  TRUE FALSE FALSE
> x>2
[1] FALSE FALSE  TRUE  TRUE  TRUE
> x>2 & x<4
[1] FALSE FALSE  TRUE FALSE FALSE
> x>2 && x<4
[1] FALSE
> x>2 | x<4
[1] TRUE TRUE TRUE TRUE TRUE
> x>2 || x<4
[1] TRUE
> x==3
[1] FALSE FALSE  TRUE FALSE FALSE
> x!=3
[1]  TRUE  TRUE FALSE  TRUE  TRUE
```

apply function

There is another useful way to do recursion in R. Which are the family of `apply` functions. First is the `apply` function that can be used on arrays and matrices. You can apply same function to specified slices of matrix or array.

```
> x<-matrix(1:9,3,3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> apply(x,1,mean)
[1] 4 5 6
> apply(x,2,sum)
[1]  6 15 24
```

lapply and sapply

`lapply` and `sapply` are list version of apply. The only difference is `lapply` returns a list and `sapply` returns a vector.

```
> y<- list(c(1,2,3),c(4,5,6))
> y
[[1]]
[1] 1 2 3

[[2]]
[1] 4 5 6

> lapply(y,sum)
[[1]]
[1] 6

[[2]]
[1] 15

> sapply(y,sum)
[1]  6 15
```

outer

`outer` is another powerful function that does recursion implicitly. It takes 2 vectors or matrices and applies a specified function to every comination of element in both objects. Here is an example:

```
> x<-c(1,2,3)
> y<-c(2,3,4)
> outer(x,y,"+")
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    4    5    6
[3,]    5    6    7
```

For each element of x outer applied "+" to every element of y.

Derivative with R

You can use R to do numerical calculations such as derivative and integral. Function for derivative is `D()`. You have to specify the function using `expression()` which prevents R from evaluating it. Here is example of taking derivative of function ax^4.
($\frac{\partial}{\partial x} ax^4 = 4ax^3$)

```
> f <- expression( a*x^4 )
> D(f, "x")
a * (4 * x^3)
```

Numerical Integration with R

- Similarly you can do numerical integration using R. Integration is simpler.
- You specify function to integrate as regular R function and use the `integrate` function.
- Here is example $\int_0^1 x^2 dx$:

```
> f <- function(x) x^2
> integrate(f, 0, 1)
0.3333333 with absolute error < 3.7e-15
```

Optimization with R

- You can do numerical optimization in R as well.
- There are many options for optimization in R.
- One the more popular methods is the `optim` function.
- Here is example of finding minimum of binomial function

$$f(p) = \left(\begin{array}{c} 10 \\ 3 \end{array} \right) p^3(1-p)^7$$

```
> LL <- function(p) {
+    choose(10,3)*p^(3)*(1-p)^(7)
+ }
>  optim(1, LL, control=list(fnscale=-1))
$par
[1] 0.3

$value
[1] 0.2668279

$counts
function gradient
      58       NA
....
```

Minimum is attained at $p = 0.3$ with value 0.2668279.

Online Resources for R

Download R: *http://cran.stat.ucla.edu/*

Online Resources for R

Download R: *http://cran.stat.ucla.edu/*

Search Engine for R: *rseek.org*

Online Resources for R

Download R: *http://cran.stat.ucla.edu/*

Search Engine for R: *rseek.org*

R Reference Card:

http://cran.r-project.org/doc/contrib/Short-refcard.pdf

Online Resources for R

Download R: *http://cran.stat.ucla.edu/*

Search Engine for R: *rseek.org*

R Reference Card:
http://cran.r-project.org/doc/contrib/Short-refcard.pdf

UCLA Statistics Information Portal: *http://info.stat.ucla.edu/grad/*

Online Resources for R

Download R: *http://cran.stat.ucla.edu/*

Search Engine for R: *rseek.org*

R Reference Card:
http://cran.r-project.org/doc/contrib/Short-refcard.pdf

UCLA Statistics Information Portal: *http://info.stat.ucla.edu/grad/*

UCLA Statistical Consulting Center: *http://scc.stat.ucla.edu*

Upcoming Mini-Courses

- Next week:
  - Spatial Statistics in R (June 2, Tuesday)
  - GIS and R (June 4, Thursday)
- For a schedule of all mini-courses offered please visit *http://scc.stat.ucla.edu/mini-courses* .

Thank you

Please take the time to complete a survey about this mini-course:

http://scc.stat.ucla.edu/survey

Your feedback is greatly appreciated.