Basic Concepts in Programming/
Introduction to R

Jan 9-10, 2012
Instructor: Wing-Yee Chow
Basic Workflow

• Characterize the problem
• Spell out the steps towards solving the problem
• Materialize the steps in a programming language
• Test and debug
Basic Terminology

- variables
- types
- vectors
- indices
- operators
- functions
- conditionals
- loops
- input/output
Variables

• Placeholders for values
  – In R, you can assign values to a variable using "<-"

```r
> a <- 5
> b <- 4
> a + b
[1] 9
> c <- a*b
> c
[1] 20
```
Variable Names in R

• Case-sensitive, so \( x \) is not the same as \( X \).
• Must not begin with numbers (e.g. 1\( x \)) or symbols (e.g. \( \%x \)).
• Must not contain blank spaces or operator symbols:
  – use \texttt{subject.list} or \texttt{subject_list}
  but \textbf{not} \texttt{subject list} or \texttt{subject-list}
Types

- Different types $\rightarrow$ different operations
  - In R, the most important types are numbers and strings.

```
> a <- 5      # number
> b <- "Hello"  # string
> c <- 1<2    # boolean
> c
[1] TRUE     # logical
```
In R, the basic values are vectors.

```
a <- c(1,2,5.3,6,-2,4)
    # numeric vector
b <- c("apples","books","cats")
    # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE)
    # logical vector
```
Indices

• **Address particular members of an object**
  
  – In R, use square brackets `[]` for indices, and round brackets `()` for functions, e.g., `length()`
  
  – In R, the first element in a vector has the index 1. Thus, the index of the last element is the length of the vector.

```r
> a <- c(37, 42, 89)
> a[1]
[1] 37
> length(a)
[1] 3
> a[length(a)]
[1] 89
```
Indices

- You can address the same things in different ways
  - Indices can be a vector of integers or logical values. (We will expand on this when we deal with data frames)

```r
> a <- c(37,42,89)
> a[1:2]
[1] 37 42
> a[c(1,3)]
[1] 37 89
> a[c(TRUE, FALSE, FALSE)]
[1] 37
> a[a>40]  # where a>40 returns
[1] 42 89  # FALSE TRUE TRUE
```
Operators

- Programming language-dependent
- Common operators in R:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ - * / %% ^</td>
<td>arithmetic</td>
</tr>
<tr>
<td>&gt; &gt;= &lt; &lt;= == !=</td>
<td>relational</td>
</tr>
<tr>
<td>! &amp;</td>
<td>logical</td>
</tr>
<tr>
<td>~</td>
<td>model formulae</td>
</tr>
<tr>
<td>&lt;- -&gt;</td>
<td>assignment</td>
</tr>
<tr>
<td>$</td>
<td>list indexing (the ‘element name’ operator)</td>
</tr>
<tr>
<td>:</td>
<td>create a sequence</td>
</tr>
</tbody>
</table>
Exercise 1

• Use R to find all the numbers between 1 and 2000 which are multiples of 317.
• You will need the operators : and %%

```r
> 1:5
[1] 1 2 3 4 5
> 18 %% 5
[1] 3
> 4 %% 2
[1] 0
```
Exercise 1

• Use R to find all the numbers between 1 and 2000 which are multiples of 317.

• Solution:

```
> a <- 1:2000
> a[ a%%317 ==0]  
[1]  317  634  951 1268 1585 1902
```

• Follow-up: Use R to find out how many numbers between 1 and 2000 are multiples of 17.
Exercise 2

• Find all the words with less than 6 or more than 8 characters in the vector `c("Maine", "Maryland", "Massachusetts", "Michigan", "Minnesota", "Mississippi", "Missouri", "Montana")`.

• You will need the OR operator `|` and function `nchar()`

```
> a <- "Maryland"
> nchar(a)
[1] 8
```
Exercise 2

• Find all the words with less than 6 or more than 8 characters in the given vector.

• Solution:

```r
> a[nchar(a) > 8 | nchar(a) < 6]
[1] "Maine" "Massachusetts" "Minnesota" "Mississippi"
```

• Follow-up: Modify this function to show the number of characters of the returned values.
Functions

• R has a wide range of built-in functions.

```r
length(c(4,2,9))
[1] 3
max(c(1,2,4,2,5,-1,1))
[1] 5
sum(c(1,2,3,4))
[1] 10
mean(c(0.5, 3.4, 8.9, 4,4, 6.7))
[1] 4.583333
sd(c(0.5, 3.4, 8.9, 4,4, 6.7))
[1] 2.893729
```
Functions

• Use `help()` to look up functions
• You can also write your own functions

```r
> myfunction <- function(x,y){
+   z <- x + y^2
+   return(z)
+ }
> answer <- myfunction(3,4)
> answer
[1] 19
```

• Local vs. global variables
Please download the new slides and exercises
Functions

Creating a function

function name

Argument(s) taken by this function

Local variables

Do computations with the argument(s)
[You can also refer to global variables, but be very careful]

Output of the function

Calling/Using your function

Global variables

You can use global variables as the input to a function

Call your function

myfunction <- function(x,y){
  z <- x + y^2
  return(z)
}

> inputA <- 3
> inputB <- 4
> answer <- myfunction(inputA,inputB)
> answer
[1] 19
Exercise 3

• Write a function that returns the product of the minimum and maximum of the input vector.
• You will need max() and min().
Exercise 3

- Write a function that returns the product of the minimum and maximum of the input vector.

Solution:

```r
exercise3 <- function(x){
  product <- max(x) * min(x)
  return(product)
}
```

```r
> exercise3(c(7, -3, 2, 30))
[1] -90
```

Follow-up: Modify this function to consider the maximum and minimum of the absolute value of the input vector. [Hint: use abs()]
Exercise 4

• Write a function that converts temperatures from Celsius to Fahrenheit.
• \[ {^\circ F} = \frac{9}{5} {^\circ C} + 32 \]
Exercise 4

- Write a function that converts temperatures from Celsius to Fahrenheit, where \( ^\circ F = \frac{9}{5} ^\circ C + 32 \).

- Solution:

```r
exercise4.c2f <- function(celsius) {
  fahrenheit <- (celsius*9/5)+32
  return(fahrenheit)
}

> exercise4.c2f(20)
[1] 68
```

- Follow-up: Write a function that converts from Fahrenheit to Celsius.
Exercise 5

• Write a function that finds all the numbers within a vector $x$ that are multiples of an integer $y$. [a general version of Ex. 1]
Exercise 5

• Write a function that finds all the numbers within a vector \( x \) that are multiples of an integer \( y \). [a general version of Ex. 1]

• Solution:

```r
exercise5 <- function(x, y) {
  z <- x[ (x%%y) == 0 ]
  return(z)
}

> ans5 <- exercise5(1:2000, 317)
```
Conditionals

• Carry out an operation when a condition is satisfied, e.g., if A is true then do B, otherwise do C (or nothing)

• Example (finding the absolute value of an input without using `abs()`):

```r
> ifelse(x<0,-x,x)
> if (x<0) {-x} else {x}
```

• Quick exercise: try to input a vector and see how these two functions differ
Conditionals

• Compare `ifelse` and `if`:

```r
abs.ifelse <- function(x){
  result <- ifelse(x<0,-x,x)
  return(result)
}

> a <- -10
> abs.ifelse(a)
[1] 10
>
> b <- c(2,3,-4)
> abs.ifelse(b)
[1] 2 3 4
```

Test your function
Conditionals

• **Compare if else and if:**

```r
abs.if <- function(x){
  if (x<0) {result <- -x} else {result <- x}
  return(result)
}
```

```r
> a <- -10
> abs.if(a)
[1] 10
> b <- c(2,3,-4)
> abs.if(b)
[1] 2 3 -4
Warning message:
In if (x < 0) { : 
  the condition has length > 1 and only the first element will be used
```

Test your function
Conditionals

- `if else` evaluates each element in turn.
  - Good for setting values of variables based on a vector of logical conditions
  - e.g., taking the absolute value of a vector

- `if` evaluates only the first element. All other elements are ignored.
  - Use this for flow control – to execute one or more statements based on a condition
Exercise 6

• Write a function that takes two arguments, \( x \) and \( y \), and outputs a message about whether \( x \) exists in \( y \).

• You will need the operator \%in\%.

```r
> content<- c("D2","V4","B6","N5","F3")
> "N5" %in% content
[1] TRUE
> "N4" %in% content
[1] FALSE
```

• Optional: You might also need \texttt{cat()}.

```r
> a <- "apple"
> b <- "tree"
> cat(a,b)
apple tree
```
Exercise 6

• Write a function that takes two arguments, \( x \) and \( y \), and outputs a message about whether \( x \) exists in \( y \).

• Solution:

```r
exercise6 <- function(x,y){
  if (x %in% y) {
    cat(x,"exists in",y)
  } else {
    cat(x,"does not exist in",y)
  }
}

> target <- c("D2")
> content <- c("C1","B4","D2","F5")
> exercise6(target, content)
D2 exists in C1 B4 D2 F5
```
Loops

• For-loops
  – repeat an action for a predetermined number of times
  – the value of the local variable changes in each iteration

```r
> for (i in 3:5){
+    print(i)
+ }
[1] 3
[1] 4
[1] 5
```

For each element in this vector the local variable is set to the value of that element and the statements in the loop are evaluated.
Loops

• For-loops
  – another example:

```r
> x <- c("Happy","New","Year")
> for (i in x) {print(i)}
[1] "Happy"
[1] "New"
[1] "Year"
> for (i in x[2:length(x)]) {print(i)}
[1] "New"
[1] "Year"
```
Loops

• For-loops
  – What is the difference?

```r
> x <- c("Happy","New","Year")
> for (i in x[2:length(x)]) {print(i)}
[1] "New"
[1] "Year"
```

```r
> for (i in 2:length(x)) {print(x[i])}
[1] "New"
[1] "Year"
```
Exercise 7

• Using a for-loop, write a function that prints each of the values in a vector, followed by “Even” if it is divisible by 2 and “Odd” otherwise.

• You can use `cat()` to display the number and the string together

```r
> a <- "R Course"
> cat(a, "Day 2")
R Course Day 2
```
Exercise 7

• Using a for-loop, write a function that prints each of the values in a vector, followed by “Even” if it is divisible by 2 and “Odd” otherwise.

• Solution:

```r
> exercise7 <- function(x) {
+   for (i in x) {
+     if (i%%2==0) {result <- "Even"}
+     } else {result <- "Odd"}
+     cat(i, result, "\n")
+   }
+ }
> exercise7(c(95,46))
95 Odd
46 Even
```

“\n” is a line break

WinterStorm 2012
Exercise 7

• Follow-up: modify this function to print “Not an integer” (instead of “Odd”) if a value is not an integer. [Hint: you can use % % to determine if a number is an integer]
Exercise 8

• Using two for-loops, print 1 instance of “Apples” followed by 4 instances of “Oranges”, and do this 3 times.
Exercise 8

• Using two for-loops, print 1 instance of “Apples” followed by 4 instances of “Oranges”, and do this 3 times.

• Solution:

```r
for (i in 1:3){
    print ("Apples")
    for (j in 1:4){
        print ("Oranges")
    }
}
```

• Follow-up: Make this a function that can do the same action to different word pairs.
Exercise 8

• Follow-up: Make this a function that can do the same action to different word pairs.

```r
exercise8 <- function(word1, word2) {
  for (i in 1:3) {
    print (word1)
    for (j in 1:4) {
      print (word2)
    }
  }
}

> exercise8("Apples","Oranges")
```
Loops

• While-loops
  – similar to for-loops, execute some codes repeatedly until a condition is satisfied

```r
> i <- 1
> while (i <= 3) {
+   print(i)
+   i <- i + 1
+ }

[1] 1
[1] 2
[1] 3
```
Exercise 9

• Using a while-loop, write a function that computes the sum of the values in a vector. [Call it `my_sum()`. This should give the same result as R’s built-in `sum()` function.]
Exercise 9

• Using a while-loop, write a function that computes the sum of the values in a vector.

• Solution:

```r
my_sum <- function(x){
  i <- 1
  temp.sum <- 0
  while (i <= length(x)) {
    temp.sum <- temp.sum + x[i]
    i <- i + 1
  }
  return(temp.sum)
}
```
Exercise 10

• Using a while-loop, compute the factorial of 53. [This should give the same result as \texttt{factorial(53)}.]

• Follow-up: Make this a function. [Call it \texttt{my_factorial()}.]
Exercise 10

• Follow-up: Make this a function that computes the factorial of an input. [Call it `my_factorial()`]

• Solution:

```r
my_factorial <- function (x) {
    fac <- 1
    while (x > 0) {
        fac <- fac * x
        x <- x - 1
    }
    return(fac)
}
```