R basics

September 15, 2019

1 Coffee Sipping Bastard's awesome R introduction file

1.1 Variables and value assignment

1.1.1 Introduction

R is a dynamically typed language, ie. you don't have to specify data types; the interpretator infers it from the entered values.

[1]:
```
a = 1
b <- 1.2
c <- c(1,2,3)
d = 4:9
e <- 4.5:9.5
print(a)
print(b)
print(c)
print(d)
print(e)
typeof(a)
typeof(b)
typeof(c)
typeof(d)
typeof(e)
```

```
[1] 1
[1] 1.2
[1] 1 2 3
[1] 4 5 6 7 8 9
[1] 4.5 5.5 6.5 7.5 8.5 9.5
```

'double'
'double'
'double'
'integer'
'double'

1.1.2 Operators and Keywords

R has the following in-built operators and keywords, and these keywords should not be used as variable names.

- keywords
  - **if**: if condition statement
  - **else**: else condition statement
  - **repeat**: repeat function returns a vector with specified parameters
  - **for**: for loop
  - **while**: while loop
  - **NULL**: Empty value
  - **FALSE**: FALSE boolean value
  - **TRUE**: TRUE boolean value
  - **NAN**: "Not a number" a keyword inserted when non numeric value is encountered. Example: string returned when numeric data was expected
  - **function**: function keyword for defining custom/user defined functions.
  - **break**: Used to break loops.
  - **NA**: "Not available", the imputed value when a field is left blank or the value is unavailable. **"?reserved" command can be used to get a full list of reserved keywords**
- Operators
  1. **Arithmatic**
     - `+`: Addition
     - `-`: Subtraction
     - `/`: Division
     - `*`: Multiplication
     - `^`: Raise to
     - `%%`: remainder/moudul
     - `%/%`: Matrix Multiplication
  2. **Boolean Operators**
     - `!`: not/negation
     - `<`: less than
     - `<=`: less than equal to
     - `>`: more than
     - `>=`: more than equal to
     - `==`: is equal to
     - `!=`: not equal to
     - `|`: or
     - `&`: and

Arithmatic Operators

[2]:
```
1+2
4+3.2
a<- c(1,2,4)
a
b <- 1:7
b
c<- 4:12
a+b
c
a+c
```

3
7.2
1. 1 2. 2 3. 4
1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7

```
Warning message in a + b:
```

1. 2 2. 4 3. 7 4. 5 5. 7 6. 10 7. 8
1. 4 2. 5 3. 6 4. 7 5. 8 6. 9 7. 10 8. 11 9. 12
1. 5 2. 7 3. 10 4. 8 5. 10 6. 13 7. 11 8. 13 9. 16

[3]:
```
2-5
-4-8
a
b
a-b
c
a-c
```

-3
-12
1. 1 2. 2 3. 4
1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7

```
Warning message in a - b:
```

1. 0 2. 0 3. 1 4. -3 5. -3 6. -2 7. -6
1. 4 2. 5 3. 6 4. 7 5. 8 6. 9 7. 10 8. 11 9. 12
1. -3 2. -3 3. -2 4. -6 5. -6 6. -5 7. -9 8. -9 9. -8

[4]:
```
2*5
-4*8
a
b
a*b
c
a*c
```

10
-32
1. 1 2. 2 3. 4
1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7

```
Warning message in a * b:
```

1. 1 2. 4 3. 12 4. 4 5. 10 6. 24 7. 7
1. 4 2. 5 3. 6 4. 7 5. 8 6. 9 7. 10 8. 11 9. 12
1. 4 2. 10 3. 24 4. 7 5. 16 6. 36 7. 10 8. 22 9. 48

[5]:
```
2/5
-4/8
a
b
a/b
c
a/c
```

0.4
-0.5
1. 1 2. 2 3. 4
1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7

```
Warning message in a/b:
```

1. 1 2. 1 3. 1.33333333333333 4. 0.25 5. 0.4 6. 0.666666666666667 7. 0.142857142857143
1. 4 2. 5 3. 6 4. 7 5. 8 6. 9 7. 10 8. 11 9. 12
1. 0.25 2. 0.4 3. 0.666666666666667 4. 0.142857142857143 5. 0.25 6. 0.444444444444444 7. 0.1 8. 0.181818181818182 9. 0.333333333333333

[6]:
```
2^5
-4^8
a
b
a^b
c
a^c
```

32
-65536
1. 1 2. 2 3. 4
1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7

```
Warning message in a^b:
```

1. 1 2. 4 3. 64 4. 1 5. 32 6. 4096 7. 1
1. 4 2. 5 3. 6 4. 7 5. 8 6. 9 7. 10 8. 11 9. 12
1. 1 2. 32 3. 4096 4. 1 5. 256 6. 262144 7. 1 8. 2048 9. 16777216

[7]:
```
2%%5
-4%%8
a
b
a%%b
c
a%%c
```

2
4
1. 1 2. 2 3. 4
1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7

```
Warning message in a%%b:
```

1. 0 2. 0 3. 1 4. 1 5. 2 6. 4 7. 1
1. 4 2. 5 3. 6 4. 7 5. 8 6. 9 7. 10 8. 11 9. 12
1. 1 2. 2 3. 4 4. 1 5. 2 6. 4 7. 1 8. 2 9. 4

Boolean Operators

[8]:
```
a = "hello"
b = "goodbye"
c = "hello"
x = 1
y = 2
z = -3
```

[9]:
```
a == b
a == c
x > y
x < y
x <= y
x >= y
z != x
z < x
```

FALSE
TRUE
FALSE
TRUE
TRUE
FALSE
TRUE
TRUE

[10]:
```
TRUE & FALSE
TRUE & TRUE
TRUE | TRUE
TRUE | FALSE
```

FALSE
TRUE
TRUE
TRUE

1.1.3 Conditional statements

There is two primary type of flow control/conditional statement in any programming language 1. if-else: * if: if given condition is true only then the bracketed statements

will be executed `if (condition){ statement 1 statement 2 ... }` * if-else: remains the same for if, but additionally you can specify what to do when the condition isn't met `if (condition 1){ statement 1 statement 2 ... }else if(condition 2){ statement 1 statement 2 ... }else{ statement 1 statement 2 .. }` * nested if: condtional statements can be nested `if (condition 1){ statement 1 if(condition 2){ statement 2 } statement 3 }else if(condition 3) { if(condition 4){ statement 4 }else{ statement 5 } statement 6 }else{ statement7 }` 2. switch: Jumps to different pieces of code depending on the case. To be avoided as far as possible * basic switch `switch(check, option 1 = statement, option 2 = statement, stop("Error Statement") )`

Additionally there is "**ifelse**" statement available in R, which compresses an if-else block into a single line.

```
ifelse(condition, to do if true, to do if false)
```

break is another flow control statement that terminated a loop.

[11]:
```
hell = 0
hot = 0
devil = "home"
if (hell == hot){
    print("hell is hot.")
    if(devil == "home")
        print("devil is home.")
}else{
    print("The fires of hell have calmed down, ragnarok is here.")
}

hell = 1
ifelse(hell==hot,print("hell is hot"),print("Hella is dead"))
```

```
[1] "hell is hot."
[1] "devil is home."
[1] "Hella is dead"
```

'Hella is dead'

1.1.4 Loops

There are two types of loop in R

- for loop:

```
for(till all the elements in given data are parsed){
    statement one
```

```
    statement two
    ...
}
```

- while loop:

```
while(condition is statisfied){
    statement one
    statement two
    ...
}
```

[12]:
```
for (i in c(1,2,3,4,5)){
    print(i)
}
for (i in matrix(1:9,3,3)){
    print(i)
    if(i == 8){
        print("found 8")
        break
    }
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] "found 8"
```

[13]:
```
i = 1
odd = c()
even = c()
while(i <=100){
    ifelse(test = (i%%2==0),yes = (even=c(even,i)),no = (odd=c(odd,i)))
    i=i+1
}
print(even)
print(odd)
```

```
 [1]   2   4   6   8  10  12  14  16  18  20  22  24  26  28  30  32  34  36  38
[20]  40  42  44  46  48  50  52  54  56  58  60  62  64  66  68  70  72  74  76
[39]  78  80  82  84  86  88  90  92  94  96  98 100
 [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
[26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

1.1.5 User defined functions

R allows users to define their own functions using the function keyword. These functions can be called upon at a later time to do tasks they're programmed for.

```
var_name <- function(para1,para2,...,paran){
    statment1
    statment2
    ...
    statment3
    return(return_value)
}
```

[6]:
```
factors <- function(g){
    fact<- c()
    for (i in 1:g){
        if(g%%i==0)
            fact<-c(fact,i)
    }
    return(fact)
}
print(factors(44))
```

```
[1]  1  2  4 11 22 44
```

[12]:
```
is.prime <- function(g){
    if (length(factors(g)) == 2)
        return(T)
    return(F)
}
is.prime(12)
is.prime(79)
```

FALSE
TRUE

[13]:
```
for (i in c(24,16,134,53,85,37,87)){
    if (is.prime(i))
        print(i)
}
```

```
[1] 53
[1] 37
```

1.1.6 Inbuilt functions

rep(x, no_times) : returns a vector

[17]:
```
rep(c(10,24,6,3),10)
```

1. 10 2. 24 3. 6 4. 3 5. 10 6. 24 7. 6 8. 3 9. 10 10. 24 11. 6 12. 3 13. 10 14. 24 15. 6 16. 3 17. 10 18. 24 19. 6 20. 3 21. 10 22. 24 23. 6 24. 3 25. 10 26. 24 27. 6 28. 3 29. 10 30. 24 31. 6 32. 3 33. 10 34. 24 35. 6 36. 3 37. 10 38. 24 39. 6 40. 3