R intro

Huitian Diao

- Reference:
- Quick-R by DataCamp
- R for Beginners by emmanuel Paradis

R syntax

1. Value assignment and function

In [1]:
```
my.string <- "Hello, world!"
print(my.string)
```

```
[1] "Hello, world!"
```

- Exercise: make R print your name

2. Comments

In [2]:
```
print("Hello Apple!")
#print("Hello Banana?")
print ("Hello Cranberry!") #Cranberry is delicious
```

```
[1] "Hello Apple!"
[1] "Hello Cranberry!"
```

Data in R

1. Objects: name, content and attribute

In [3]:
```
my.string <- "foo"
my.number <- 888
my.logic <- TRUE
mode(my.string)
mode(my.number)
mode(my.logic)
```

'character'

'numeric'

'logical'

2. Data types

2.1 Vectors: a sequence of data elements of the same basic type. Members in a vector are called components.

In [4]:
```
my.numbers <- c(1,2,3)
my.fruits <- c("Apple","Banana","Cranberry")
my.logic <- c(TRUE,FALSE,TRUE)
my.logic[1]
my.fruits[c(1,3)]
```

TRUE

'Apple' 'Cranberry'

2.2 Matrix: elements arranged in a two-dimensional rectangular layout. Elements in matrix are the same type.

In [5]:
```
my.matrix <- matrix(1:20, nrow=5, ncol=4)
my.matrix

my.fruits <- c("Apple","Bananna","Cranberry","Durian")
row.names <- c("Row1", "Row2")
column.names <- c("ColA", "ColB")
my.fruits.matrix <- matrix(my.fruits, nrow=2, ncol=2, byrow=TRUE, dimnam
es=list(row.names, column.names))
my.fruits.matrix
```

1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19
5	10	15	20

	ColA	**ColB**
Row1	Apple	Bananna
Row2	Cranberry	Durian

2.3 Array: like matrices but can store more than 2 dimensions.

In [6]:
```
vector.1 <- seq(0,26,by=1) # Create a vector of numbers from 0 to 26.
column.names <- c("ColA","ColB","ColC")
row.names <- c("Row1","Row2","Row3")
matrix.names <- c("theMatrix","theMatrixReloaded","theMatrixRevolutions"
)
array.1 <- array(vector.1, dim=c(3,3,3), dimnames=list(row.names,column.
names,matrix.names))
print(array.1)
```

```
, , theMatrix

     ColA ColB ColC
Row1    0    3    6
Row2    1    4    7
Row3    2    5    8

, , theMatrixReloaded

     ColA ColB ColC
Row1    9   12   15
Row2   10   13   16
Row3   11   14   17

, , theMatrixRevolutions

     ColA ColB ColC
Row1   18   21   24
Row2   19   22   25
Row3   20   23   26
```

2.4 Data Frames: like matrices, but different columns can have different modes.

In [7]:
```
fruits.type <- c("Apple","Bananna","Cranberry","Durian")
fruits.numbers <- c(10,20,100,4)
fruits.price <- c(5,10,15,20)
fruits.smell <- c(FALSE, FALSE, FALSE, TRUE)
df.fruits <- data.frame(fruits.type, fruits.numbers, fruits.price, fruit
s.smell)
names(df.fruits) <- c("Type","Number","Price","Smell")
df.fruits
df.fruits[c("Type","Price")] # Extract columns with IDs "Type" and "Pric
e"
df.fruits[1:2] # Extract columns 1-2 of data frame
df.fruits$Type # Extract "Type" from data frame
```

Type	Number	Price	Smell
Apple	10	5	FALSE
Bananna	20	10	FALSE
Cranberry	100	15	FALSE
Durian	4	20	TRUE

Type	Price
Apple	5
Bananna	10
Cranberry	15
Durian	20

Type	Number
Apple	10
Bananna	20
Cranberry	100
Durian	4

Apple Bananna Cranberry Durian

2.5 List: an ordered collection of objects. A list allows you to gather a variety of objects.

In [8]:
```
nutella.favorite <- c("chocolate", "peanut", "cheese")
nutella.matrix <- matrix(1:20, nrow=5, ncol=4)
nutella.info <- list(my.name="Nutella", my.species="Mus.Musculus", my.ag
e=0.2, my.color="Brown",
                     my.favorite=nutella.favorite, my.matrix=nutella.matrix)
nutella.info
```

$my.name
'Nutella'
$my.species
'Mus.Musculus'
$my.age
0.2
$my.color
'Brown'
$my.favorite
'chocolate' 'peanut' 'cheese'
$my.matrix

1	6	11	16
2	7	12	17
3	8	13	18
4	9	14	19
5	10	15	20

2.6 Factors: a nominal variable

In [9]:
```
gender <- c(rep("male",20), rep("female", 30))
summary(gender)
gender <- factor(gender)
summary(gender)
```

```
   Length     Class      Mode
       50 character character
```

female 30
male 20

2.7 Data type conversion

In [10]:
```
my.numbers <- c("1","2","3","4")
class(my.numbers)
my.numbers <- as.numeric(my.numbers)
class(my.numbers)

my.logic <- c(TRUE, TRUE, FALSE)
class(my.logic)
my.logic <- as.character(my.logic)
class(my.logic)
```

'character'

'numeric'

'logical'

'character'

3. Basic functions for objects

In [11]:
```
class(nutella.info) # Type of object
names(nutella.info) # Names
str(nutella.info) # Structure of an object
length(nutella.info) # Number of elements
ls() # List current objects
```

'list'

'my.name' 'my.species' 'my.age' 'my.color' 'my.favorite' 'my.matrix'

```
List of 6
 $ my.name    : chr "Nutella"
 $ my.species : chr "Mus.Musculus"
 $ my.age     : num 0.2
 $ my.color   : chr "Brown"
 $ my.favorite: chr [1:3] "chocolate" "peanut" "cheese"
 $ my.matrix  : int [1:5, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
```

6

'array.1' 'column.names' 'df.fruits' 'fruits.numbers' 'fruits.price' 'fruits.smell' 'fruits.type' 'gender' 'matrix.names' 'my.fruits' 'my.fruits.matrix' 'my.logic' 'my.matrix' 'my.number' 'my.numbers' 'my.string' 'nutella.favorite' 'nutella.info' 'nutella.matrix' 'row.names' 'vector.1'

In [12]:
```
rm(nutella.info) # Delete a object
ls()
```

'array.1' 'column.names' 'df.fruits' 'fruits.numbers' 'fruits.price' 'fruits.smell' 'fruits.type' 'gender' 'matrix.names' 'my.fruits' 'my.fruits.matrix' 'my.logic' 'my.matrix' 'my.number' 'my.numbers' 'my.string' 'nutella.favorite' 'nutella.matrix' 'row.names' 'vector.1'

Basic data operations

1. Operators

1.1 Arithemetic operators

In [13]:
```
x <- 2
y <- 3
x+y
x-y
x*y
x/y
x**y # exponentiation
x%%y # modulus
x%/%y # integer division
```

```
5

-1

6

0.666666666666667

8

2

0
```

1.2 Logical operators

In [14]:
```
x <- 5
y <- 11
like.chocolate <- TRUE
like.spam <- FALSE
x < y
x > y
x <= y
x >= y
x == y
x != y
!like.chocolate
like.chocolate | like.spam
like.chocolate & like.spam
isTRUE(like.spam)
```

TRUE

FALSE

TRUE

FALSE

FALSE

TRUE

FALSE

TRUE

FALSE

FALSE

1.3 Built-in functions

In [15]:
```
x <- -16
y <- 1.239
z <- -2.251
abs(x)
sqrt(-x)
ceiling(y)
floor(z)
trunc(z)
round(y, digits=2)
signif(y, digits=2)
cos(x)
sin(x)
tan(x)
log(y)
log10(y)
exp(x)
```

16

4

2

-3

-2

1.24

1.2

-0.957659480323385

0.287903316665065

-0.300632242023903

0.214304602647005

0.0930713063760635

1.12535174719259e-07

In [16]:
```
my.name <- "Nutella"
my.fav <- c("chocolate", "peanut", "cheese")
my.motto <- "I love chocolate. --Nutella"

substr(my.name, start=1, stop=5) # Extract part of the string
grep("cheese", my.fav) # Search for pattern
sub( "chocolate","cheese", my.motto) # Replace pattern
strsplit(my.motto, " ") # Split elements
paste(1:3, my.fav, sep=": ") # Concatenate strings
toupper(my.motto)
tolower(my.motto)
```

'Nutel'

3

'I love cheese. --Nutella'

1. 'I' 'love' 'chocolate.' '--Nutella'

'1: chocolate' '2: peanut' '3: cheese'

'I LOVE CHOCOLATE. --NUTELLA'

'i love chocolate. --nutella'

2. Control structures

2.1 if-else

In [17]:
```
like.peanut <- TRUE
if (like.peanut){
    print("I like peanuts!")
} else {
    print("What do I like?")
}
```

```
[1] "I like peanuts!"
```

2.2 for

In [18]:
```
for (i in c(1:10)) {
    print("More chocolate please!")
}
```

```
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
```

2.3 while

In [19]:
```
i <- 0
while (i < 10) {
    print("More chocolate please!")
    i <- i+1
}
```

```
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
[1] "More chocolate please!"
```

More on data operations

1. Import data

In [20]:
```
setwd("/Users/yolandatiao/Documents/0_Bioinformatics2017/2018_Bioinforma
tics/Unit1-module2-R")
nav.d14 <- read.csv("NAV-D14_DEseq2.csv", header=TRUE)
class(nav.d14)
```

'data.frame'

2. View data

In [21]:
```
str(nav.d14)
dim(nav.d14)
head(nav.d14, n=5)
tail(nav.d14, n=10)
#levels() # List levels of factor
```

```
'data.frame':   500 obs. of  7 variables:
 $ gene.name     : Factor w/ 500 levels "AASS","ABCB4",..: 452 448 137
380 58 159 92 164 171 284 ...
 $ baseMean      : num  25.257 0.289 163.508 296.462 199.012 ...
 $ log2FoldChange: num  1.3025 -0.0508 -0.055 0.2737 -1.6201 ...
 $ lfcSE         : num  0.417 0.488 0.249 0.196 0.283 ...
 $ stat          : num  3.126 -0.104 -0.221 1.393 -5.718 ...
 $ pvalue        : num  1.77e-03 9.17e-01 8.25e-01 1.63e-01 1.07e-08
...
 $ padj          : num  8.21e-03 NA 9.14e-01 3.31e-01 1.59e-07 ...
```

```
   500  7
```

gene.name	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
TSPAN6	25.256745	1.30246882	0.4166710	3.1258926	0.0017726620	0.008209
TNMD	0.288508	-0.05077872	0.4875003	-0.1041614	0.9170412620	NA
DPM1	163.507773	-0.05502833	0.2492486	-0.2207769	0.8252661650	0.914449
SCYL3	296.461713	0.27370099	0.1964251	1.3934117	0.1634953240	0.331239
C1orf112	199.011986	-1.62012005	0.2833121	-5.7184998	0.0000000107	0.000000

	gene.name	baseMean	log2FoldChange	lfcSE	stat	pvalue	padj
491	SARS	383.40062	-0.01094582	0.1850942	-0.0591365	0.95284338	0.9797
492	RANBP3	379.84288	0.08522877	0.1708371	0.4988890	0.61785761	0.7848
493	ARID4A	1435.32399	0.04869732	0.1904170	0.2557404	0.79815131	0.9010
494	EIPR1	87.86504	-0.95503525	0.2356304	-4.0531067	0.00005050	0.0003
495	PNPLA6	146.41751	-0.14635328	0.2900234	-0.5046257	0.61382175	0.7821
496	IFT88	111.71539	0.60100567	0.3082850	1.9495130	0.05123419	0.1361
497	ALG1	72.37193	0.33922670	0.2449200	1.3850512	0.16603683	0.3352
498	ZCCHC8	270.00278	-0.22916478	0.1966266	-1.1654819	0.24382397	0.4418
499	ABCF2	288.47676	0.10264088	0.2711067	0.3785995	0.70498528	0.8456
500	CHPF2	171.74016	-0.07133449	0.2608077	-0.2735138	0.78445831	0.8936

3. Subsetting / Merging data

In [22]:
```
nav.d14.sub <- nav.d14[c("gene.name", "log2FoldChange", "pvalue")] # Sub
setting by column names
```

In [23]:
```
nav.d14.sub <- cbind(nav.d14.sub, nav.d14["baseMean"]) # Merging columns
head(nav.d14.sub)
```

gene.name	log2FoldChange	pvalue	baseMean
TSPAN6	1.30246882	1.772662e-03	25.256745
TNMD	-0.05077872	9.170413e-01	0.288508
DPM1	-0.05502833	8.252662e-01	163.507773
SCYL3	0.27370099	1.634953e-01	296.461713
C1orf112	-1.62012005	1.070000e-08	199.011986
FGR	-5.60936805	1.440000e-18	122.792530

In [24]:
```
nav.d14.sub.sig.less <- subset(nav.d14.sub, pvalue <= 0.05 & log2FoldCha
nge < 0) # Subsetting by value
nav.d14.sub.sig.more <- subset(nav.d14.sub, pvalue <= 0.05 & log2FoldCha
nge > 0)
```

In [25]:
```
nav.d14.sub.sig.all <- rbind(nav.d14.sub.sig.less, nav.d14.sub.sig.more)
 # Merging rows
dim(nav.d14.sub.sig.less)
dim(nav.d14.sub.sig.more)
dim(nav.d14.sub.sig.all)
```

116 4

73 4

189 4

*** Change column names**

In [26]:
```
colnames(nav.d14.sub.sig.all) <- c("gn","fc","pval","bm")
head(nav.d14.sub.sig.all)
```

	gn	fc	pval	bm
5	C1orf112	-1.6201201	1.070000e-08	199.01199
6	FGR	-5.6093681	1.440000e-18	122.79253
8	FUCA2	-0.7644220	5.400523e-03	72.03849
9	GCLC	-0.6690268	1.563919e-03	232.82360
14	ENPP4	-1.0797584	5.970000e-05	166.51521
22	BAD	-1.0572374	1.526298e-02	17.43336

4. Sort data

In [27]:
```
nav.d14.sub.sig.all.srt <- nav.d14.sub.sig.all[order(nav.d14.sub.sig.all
$bm),]
head(nav.d14.sub.sig.all, n=5)
head(nav.d14.sub.sig.all.srt, n=5)
```

	gn	fc	pval	bm
5	C1orf112	-1.6201201	1.070000e-08	199.01199
6	FGR	-5.6093681	1.440000e-18	122.79253
8	FUCA2	-0.7644220	5.400523e-03	72.03849
9	GCLC	-0.6690268	1.563919e-03	232.82360
14	ENPP4	-1.0797584	5.970000e-05	166.51521

	gn	fc	pval	bm
220	PGLYRP1	1.794950	0.013660222	2.276910
163	UPP2	1.062059	0.042900000	9.993211
253	CD9	1.817403	0.002944941	10.048876
37	DBNDD1	1.824313	0.003734245	10.811359
307	TYROBP	-2.110031	0.001129604	12.701733

5. User defined function and apply

In [28]:
```
# Define a function to evaluate fold change, up-regulation/ down-regulat
ion / no fold change
fcEval <- function(x) {
    if(x>0){
        return("UP")
    }
    else if(x<0){
        return("DOWN")
    }
    else{
        return("NONE")
    }
}
```

In [29]:
```
fc.eval.results <- lapply(nav.d14.sub.sig.all$fc,fcEval) # Use lapply to
 process all the
fc.eval.results <- factor(unlist(fc.eval.results)) # Convert list to vec
tor
summary(fc.eval.results)
```

```
DOWN  116
  UP   73
```

In [ ]: