basic concept and syntax

Code

basic built-in operations

1. system and R lang related

1. help: `help(<method_name>)` or `?<method_name>`
2. `mode()` , `class` : details see 1a
3. `ls()` will display all the objects exist in this R session
4. `rm()` remove a existing object
   `rm(list=ls())` : remove all
5. `options()` : configurations
    1. `options(digits=3)` : display number with 3 digits
6. `proc.time()` return the time like python `time`
7. `exists("name")` : whether exists an object called `name`
8. `traceback()` : When an error occurs, R saves information about the current stack of active functions, and traceback() prints this list. (latest on top, different with python)

2. Very basic syntax

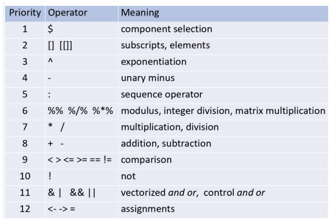
1. Comments Use `#` to comment.
2. (Especially in R command line,) if want 2 commands in one line, separate with `;`
3. assignment
    1. <CCLD>: when you use, when doing assignment always `<-` , when calling function param, always `=` . But also remember the special case below.
    2. explanation:
        1. similar to assignment of other languages. Can use symbols `=` or `<-`
        2. but R uses = for yet another purpose: associating function arguments with values. E.g., `f(x=3)` .
           But can also `f(x<-3)` , and in this way after the funciton, `x` is still there since it is global.
        3. Technically, `<-` is a global assignment operator, `=` is a local assignment operator. But the local in the outmost environ is global.
    3. Other notes:
        1. similar to C, the return value of assignment is the value it assigns

```
Fibonacci[1] <- Fibonacci[2] <- 1
```

        2. But in functions, `<-` is still not “global”. `<<-` is. See section 5a for details

3. math and logic calculation

1. Operations and precedence



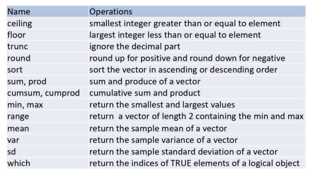
Priority	Operator	Meaning
1	$	component selection
2	[] [[]]	subscripts, elements
3	^	exponentiation
4	-	unary minus
5	:	sequence operator
6	%% %/% %*%	modulus, integer division, matrix multiplication
7	* /	multiplication, division
8	+ -	addition, subtraction
9	< > <= >= == !=	comparison
10	!	not
11	& | && ||	vectorized *and or*, control *and or*
12	<- -> =	assignments

1. Notes:
    1. Associativity
       In the same row of the table (which means same priority), use associativity rules.
       Most operators in R have associativity from left to right.
       Exponent `^` and leftward assignment (`<-` , `=`) are from right to left.
       eg.
        1. `5%%3%/%2` means `(5%%3)%/%2` (left to right)
        2. `2^3^2` means `2^(3^2)` (right to left)
    2. Inside `()` rearrange priority
    3. Unary minus means the negative sign (?)
    4. The `&` and `|` operator performs the element-wise comparison and returns a logical vector of the same length as its input.
       `&&` and `||` : beside is 2 logical expression with value `T` or `F` .
    5. NA involved in any calculation (math / logic) will result in NA
2. Special numbers

```
> 1/0
[1] Inf

> 0/0
[1] NaN
```

3. Built-in math functions
    1. sqrt, abs, sin, cos, log, exp
    2. others



Name	Operations
ceiling	smallest integer greater than or equal to element
floor	largest integer less than or equal to element
trunc	ignore the decimal part
round	round up for positive and round down for negative
sort	sort the vector in ascending or descending order
sum, prod	sum and produce of a vector
cumsum, cumprod	cumulative sum and product
min, max	return the smallest and largest values
range	return a vector of length 2 containing the min and max
mean	return the sample mean of a vector
var	return the sample variance of a vector
sd	return the sample standard deviation of a vector
which	return the indices of TRUE elements of a logical object

        1. notes / eg use
            1. `ceiling` , `floor` , `trunc` , `round` : note, the return value is still numeric not integer

```
> round(1.5)
[1] 2
> class(round(1.5))
[1] "numeric"
```

            2. `round` has a paramm `digit` , indicating the number of decimal places. Default is 0
            3. stats functions like `mean` , `var` , `sd` : if has `NaN` inside, the result is `NaN`

```
mean(c(NaN, 1, 2))
[1] NaN
```

            4. `cumsum` :

```
> cumsum(c(1, 2, 3, 4))
[1]  1  3  6 10
```

    3. other misc
        1. diff(v)
           return the diff `v[i+1]-v[i]`

```
> diff(c(1, 2, 4))
[1] 1 2
```

4. Built-in logic functions
    1. `all()` and `any()`
        1. functions in R can be used to check if all or any values in a vector evaluate to TRUE for some expression. Return is a single `TRUE` or `FALSE` .
        2. Directly do eg `&&` to multi-ele vector can result in error

4. string operations

1. `cat`
    1. idea: just input multiple string, concatenate them together with auto coersion. No automatic space.
    2. eg

```
> cat("iteration = ", 7, "\n")
iteration =  7
```

2. `paste`
    1. syntax

```
paste (..., sep = " ", collapse = NULL, recycle0 = FALSE)
```

    2. eg

```
> paste("Tree", 1:5)
[1] "Tree 1" "Tree 2" "Tree 3" "Tree 4" "Tree 5"

> (nth <- paste0(1:12, c("st", "nd", "rd", rep("th", 9))))
 [1] "1st"  "2nd"  "3rd"  "4th"  "5th"  "6th"  "7th"  "8th"  "9th"  "10th"
[11] "11th" "12th"
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
> paste(month.abb, "is the", nth, "month of the year.")
 [1] "Jan is the 1st month of the year."  "Feb is the 2nd month of the year."
 [3] "Mar is the 3rd month of the year."  "Apr is the 4th month of the year."
 [5] "May is the 5th month of the year."  "Jun is the 6th month of the year."
 [7] "Jul is the 7th month of the year."  "Aug is the 8th month of the year."
 [9] "Sep is the 9th month of the year."  "Oct is the 10th month of the year."
[11] "Nov is the 11th month of the year." "Dec is the 12th month of the year."
```

4. datatype related opereations

1. Inspect datatype: see section 1
2. Convert datatype:
    1. Syntax: `as.<type>()`

```
> v = c(1, 2, 3, 'a', 'b', 4)
> as.numeric(v)
[1]  1  2  3 NA NA  4
> as.logical(v)
[1] NA NA NA NA NA NA
> as.character(v)   # actually originally v is of mode character
[1] "1" "2" "3" "a" "b" "4"
```

    2. Note: can cause `NA` if not able to convert
    3. Note: can also have `as.interger()`
       But note:

```
> as.integer(1.5)
[1] 1
```

       So the way to check for integer:

```
(input-floor(input)>0)
```

tricky

1. `y<-x<2` is actually: `y = (x<2)`