5_programming

basic built-in functions for function

1. misc

1. import functions:

```
source("ch4.r")
```

2. Functions can be entered or edited using `fix(function_name)`
3. `stop(message)` : terminates evaluation of the current function and display message

2. basic function syntax

1. Define
    1. syntax

    ```
    myfunction <- function(arg1, arg2, ... ) {
      statements
      ...
      return(object)
    }
    ```

    2. notes
        1. If there are no explicit returns from a function, the value of the last evaluated expression is returned automatically.
        2. Can return only one object. If multiple, wrap it in a list/vector …

3. Function related syntax

1. Scope
    1. variables defined within functions have local scope
        1. Even if using `<-` .
        2. A variable with the same name could be created in a different function but there is no risk of a clash
    2. super assignment
        1. ***!!!*** In functions to modify global variable, always use super assignment
        2. syntax: use `<<-`
        3. eg

        ```
        f <- function() {
            if (!exists("f_count"))
                # check existence of f_count
                f_count <<- 1
            else
                f_count <<- f_count + 1
            return(f_count)
        }
        ```

2. default value of function arguments
    1. If not specified below, is same with python
    2. the default valued arguments don’t have to go after the arguments w/0 default value
    3. will receive keyword first, then do it by order

    ```
    f<-function(a,b=2,c)

    f(1, 3)   # error
    f(a=1, c=3)   # a, b, c: 1, 2, 3
    f(1, a=0, 3)  # a, b, c: 0, 2, 3
    ```

3. flexible num of arguments

Use `...`

eg.

```
maxlen <- function (...) { # allow flexible arguments
  arg <- list(...) # save the argument list to arg
  mx <- 0 # initialize mx
  for (x in arg) mx <-max(mx,length(x)) # find max length
  return(mx)
}

# indexing `...`
s<-function(multiplier,...){
    print(..1)    # first element
}
```

recursive functions

1. syntax

    eg

    ```
    fac<-function(n){
        if (n<=2) return(n)
        else return(n*fac(n-1))
    }
    ```

2. template

```
function() {

if (base_case) {
    return
}

# non base case

}
```

error handling

1. raise error: by `stop`

    eg. `if (low>up) stop ("Error: first arg. > second arg.")`

2. raise warning: by `warning()`

    eg. `warning("W1")`

“rename” functions

1. User defined operator
    1. note: name must naming starts and ends with `%`
    2. eg.

    ```
    > "%+-%" <- function(x,s) { c(x-s,x+s) }
    > 3 %+-% 5    # way 1
    [1] -2 8
    > "%+-%"(3, 5)  # way 2
    [1] -2  8
    ```

2. Replacement functions
    1. syntax:
        1. the fun name must end with `<-`
        2. the value is a keyword argument, cannot change its name
    2. explain on eg

    ```
    > "modify<-" <- function(x, position, value) {
          x[position] <- value
          x
      }
    > x <- 1:10
    > modify(x,2) <- 5L
    > x
    [1]  1  5  3  4  5  6  7  8  9 10
    ```

    which is equivalent to

    ```
    x <- "modify<-"(x, 2, 5L)
    ```