FUNCTION ARGUMENTS

DSC205 Introduction to Advanced Data Science

Marcus Birkenkrahe

March 10, 2023

Arguments

- Watch: "The Argument" by Monty Python (the British comedy group that gave the "Python" programming language its name).

- The English word 'argument' comes from Latin *argumentum* for *make clear, prove, accuse*, and stands for any 'process of reasoning'[1].
- Here: default arguments, missing argument values, and passing extra arguments using ellipses (...)

Lazy argument evaluation

- "Lazy" evaluation refers to the fact that expressions are evaluated only when they're needed by the program (saving time and effort).
- We're going to look at an extended working example: searching through a `list` for `matrix` objects and trying to multiply them with another `matrix` specified as a second argument.

Intermission: matrix multiplication

- In order to multiply two matrices A and B of size m x n and p x q, it must be true that n = p (column length of A == row length of B).

[1]Since I'm writing this lecture on the eve of the visit of the "Pope's astronomer", Br. Guy Consolmagno, a Jesuit and astronomer to Pope Francis (another Jesuit), I am reminded of the reputation of Jesuits for being great at arguing - something you could see in the Q&A session after Gr. Guy's talk on "Astronomy, Religion and the Art of Storytelling". One could argue that any story contains at least one argument (a complete process of reasoning), and that the best arguments are constructed like stories (remember the Freytag curve, which I've also used in my paper on scientific storytelling).

$$\begin{bmatrix} 2 & 5 & 2 \\ 6 & 1 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 & -3 \\ -1 & 1 \\ 1 & 5 \end{bmatrix}$$

$$= \begin{bmatrix} 2\times 3 + 5\times(-1) + 2\times 1 & 2\times(-3) + 5\times(1) + 2\times 5 \\ 6\times 3 + 1\times(-1) + 4\times 1 & 6\times(-3) + 1\times(1) + 4\times 5 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 9 \\ 21 & 3 \end{bmatrix}$$

- Unlike `matrix` addition, subtraction and scalar multiplication (with a number), matrix multiplication is not an element-wide operation, and the operator is not `*` but instead `%*%`.
- Example: create two matrices, print them and check their dimension:

```
A <- rbind(c(2,5,2),  # rbind is filled by row
           c(6,1,4))
A
dim(A)
B <- cbind(c(3,-1,1),  # cbind is filled by column
           c(-3,1,5))
B
dim(B)
```

```
     [,1] [,2] [,3]
[1,]    2    5    2
[2,]    6    1    4
[1] 2 3
     [,1] [,2]
[1,]    3   -3
[2,]   -1    1
[3,]    1    5
[1] 3 2
```

- Let's write a function `multiply` to check compatibility and run it:

1. `function` arguments are matrices `A` and `B`.
2. test if `ncol(A)` is equal to `nrow(B)`
3. multiply if `TRUE`, otherwise print message
4. test the function on the globally defined `A` and `B` matrices

```
multiply <- function(A,B) {
  if (ncol(A) == nrow(B)) {
    A %*%B
  } else {
    paste("Not compatible:", ncol(A), "!=", nrow(B))
  }
}
multiply(A,B)
```

```
     [,1] [,2]
[1,]    3    9
[2,]   21    3
```

- Solution for `multiply`:

```
multiply <- function(A,B) {
  if (ncol(A) == nrow(B)) {
    A %*%B
  } else {
    paste("Not compatible:", ncol(A), "!=", nrow(B))
  }
}
multiply(A,B)
```

```
     [,1] [,2]
[1,]    3    9
[2,]   21    3
```

- Matrix multiplication is generally not commutative. Check that by running `multiply` on `B` and `A` in reverse order from before:

```
multiply(B,A)
```

```
     [,1] [,2] [,3]
[1,]  -12   12   -6
[2,]    4   -4    2
[3,]   32   10   22
```

- Finally, test the function on two incompatible matrices `C` and `D`:

```
C <- matrix(1:4,2)
D <- matrix(1:9,3)
multiply(C,D)

[1] "Not compatible: 2 != 3"
```

Example: multiple `function` arguments

- Write a function `mult1` that accepts as arguments:
  1. a list `x`
  2. a matrix `mat`
  3. two strings `str1` and `str2`
- The function will search through `x`, look for matrices that can be multiplied with `mat`, and store the return the result in a new list.
- If no (compatible) matrices are in the supplied list `x`, the user should be informed of these facts.
- Pseudocode for the function body (objects marked with `$`):

```
$flag matrices in the list $x
If $x contains no matrices
  return $str1
Otherwise:
  Make index from $flag
  Initialize matrix $counter and $result list
  Loop over matrices
     Store matrix in $temp
     If matrix is compatible with mat
        Increase matrix counter
        Multiply $temp and $mat, store in $result
  If $counter is 0 (no compatible matrices)
     return $str2
  Otherwise:
     return $result of matrix multiplication
```

- Let's code the function `mult1` with arguments `x`, `mat`, `str1`, `str2`:

```
mult1 <- function(x,mat,str1,str2) {
  ## $flag matrices - use 'sapply', FUN=is.matrix
  flag <- sapply(x, FUN=is.matrix)

  ## check if $x has 'any' matrices, otherwise 'return' $str1
  if(!any(flag)) return (str1)

  ## $x contains matrices! make index vector $idx from $flag
  idx <- which(flag)

  ## initialize matrix $counter to 0, and empty $result list
  counter <- 0
  result <- list()

  ## loop over matrices (use $idx as loopindex)
  for (i in idx) {
    ## store $x in $temp
    x[[i]] -> temp
    ## check if dim of $x and $mat are compatible
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    } ## end if
  } ## end loop over $idx

  ## check if $counter is still 0 then 'return' $str2
  ## otherwise 'return' $result
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
} ## end function definition
```

- Solution:

```
mult1 <- function(x,mat,str1,str2) {
  flag <- sapply(x, FUN=is.matrix)
  if(!any(flag)) return (str1)
  idx <- which(flag)
  counter <- 0
  result <- list()
  for (i in idx) {
    x[[i]] -> temp
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    }
  }
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
}
```

- Test suite with three `list` objects `foo`, `bar` and `baz`

```
foo <- list(matrix(1:4,2,2),
            "not a matrix",
            "definitely not a matrix",
            matrix(1:8,2,4),
            matrix(1:8,4,2))
bar <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??")
baz <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??",
            matrix(1:8,2,4))
```

- Test `mult1` with `foo` and set `mat` to the 2 x 2 identity matrix - so that post-multiplying any matrix with `mat` will simply return the original matrix, as well as appropriate messages `str1`, `str2`:

```
foo <- list(matrix(1:4,2,2),
            "not a matrix",
            "definitely not a matrix",
            matrix(1:8,2,4),
            matrix(1:8,4,2))
bar <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??")
baz <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??",
            matrix(1:8,2,4))
mult1 <- function(x,mat,str1,str2) {
  ## $flag matrices - use 'sapply', FUN=is.matrix
  flag <- sapply(x, FUN=is.matrix)

  ## check if $x has 'any' matrices, otherwise 'return' $str1
  if(!any(flag)) return (str1)

  ## $x contains matrices! make index vector $idx from $flag
  idx <- which(flag)

  ## initialize matrix $counter to 0, and empty $result list
  counter <- 0
  result <- list()

  ## loop over matrices (use $idx as loopindex)
  for (i in idx) {
    ## store $x in $temp
    x[[i]] -> temp
    ## check if dim of $x and $mat are compatible
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    } ## end if
  } ## end loop over $idx

  ## check if $counter is still 0 then 'return' $str2
  ## otherwise 'return' $result
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
} ## end function definition
mult1(x = foo,
      mat = diag(2),
      str1 = "no matrices in x",
      str2 = "no compatible matrices in x")

[[1]]
     [,1] [,2]
[1,]    1    3
[2,]    2    4

[[2]]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

- Test `mult1` with `bar`, which has no matrices at all, and the same arguments otherwise:

```
foo <- list(matrix(1:4,2,2),
            "not a matrix",
            "definitely not a matrix",
            matrix(1:8,2,4),
            matrix(1:8,4,2))
bar <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??")
baz <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??",
            matrix(1:8,2,4))
mult1 <- function(x,mat,str1,str2) {
  ## $flag matrices - use 'sapply', FUN=is.matrix
  flag <- sapply(x, FUN=is.matrix)

  ## check if $x has 'any' matrices, otherwise 'return' $str1
  if(!any(flag)) return (str1)

  ## $x contains matrices! make index vector $idx from $flag
  idx <- which(flag)

  ## initialize matrix $counter to 0, and empty $result list
  counter <- 0
  result <- list()

  ## loop over matrices (use $idx as loopindex)
  for (i in idx) {
    ## store $x in $temp
    x[[i]] -> temp
    ## check if dim of $x and $mat are compatible
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    } ## end if
  } ## end loop over $idx

  ## check if $counter is still 0 then 'return' $str2
  ## otherwise 'return' $result
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
} ## end function definition
mult1(x = bar,
      mat = diag(2),
      str1 = "no matrices in x",
      str2 = "no compatible matrices in x")

[1] "no matrices in x"
```

- Finally, test `mult1` with `baz`, which has one matrix but no compatibility for multiplication with `mat`:

```
foo <- list(matrix(1:4,2,2),
            "not a matrix",
            "definitely not a matrix",
            matrix(1:8,2,4),
            matrix(1:8,4,2))
bar <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??")
baz <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??",
            matrix(1:8,2,4))
mult1 <- function(x,mat,str1,str2) {
  ## $flag matrices - use 'sapply', FUN=is.matrix
  flag <- sapply(x, FUN=is.matrix)

  ## check if $x has 'any' matrices, otherwise 'return' $str1
  if(!any(flag)) return (str1)

  ## $x contains matrices! make index vector $idx from $flag
  idx <- which(flag)

  ## initialize matrix $counter to 0, and empty $result list
  counter <- 0
  result <- list()

  ## loop over matrices (use $idx as loopindex)
  for (i in idx) {
    ## store $x in $temp
    x[[i]] -> temp
    ## check if dim of $x and $mat are compatible
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    } ## end if
  } ## end loop over $idx

  ## check if $counter is still 0 then 'return' $str2
  ## otherwise 'return' $result
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
} ## end function definition
mult1(x = baz,
      mat = diag(2),
      str1 = "no matrices in x",
      str2 = "no compatible matrices in x")

[1] "no compatible matrices in x"
```

- Notice that the string arguments `str1` and `str2` are used only when the argument `x` does not contain a matrix with the appropriate dimensions.
- R evaluates the arguments "lazily": argument values are sought only when they are required during execution. For `x=foo` you could lazily ignore the string arguments.
- Run `mult1` again only for `x` and `mat`:

```
foo <- list(matrix(1:4,2,2),
            "not a matrix",
            "definitely not a matrix",
            matrix(1:8,2,4),
            matrix(1:8,4,2))
bar <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??")
baz <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??",
            matrix(1:8,2,4))
mult1 <- function(x,mat,str1,str2) {
  ## $flag matrices - use 'sapply', FUN=is.matrix
  flag <- sapply(x, FUN=is.matrix)

  ## check if $x has 'any' matrices, otherwise 'return' $str1
  if(!any(flag)) return (str1)

  ## $x contains matrices! make index vector $idx from $flag
  idx <- which(flag)

  ## initialize matrix $counter to 0, and empty $result list
  counter <- 0
  result <- list()

  ## loop over matrices (use $idx as loopindex)
  for (i in idx) {
    ## store $x in $temp
    x[[i]] -> temp
    ## check if dim of $x and $mat are compatible
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    } ## end if
  } ## end loop over $idx

  ## check if $counter is still 0 then 'return' $str2
  ## otherwise 'return' $result
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
} ## end function definition
mult1(x=foo,mat=diag(2))

[[1]]
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4

[[2]]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

- However, for `x=bar` this will not work - an argument is missing:

```
foo <- list(matrix(1:4,2,2),
            "not a matrix",
            "definitely not a matrix",
            matrix(1:8,2,4),
            matrix(1:8,4,2))
bar <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??")
baz <- list(1:4,
            "not a matrix",
            c(F,T,T,T),
            "??",
            matrix(1:8,2,4))
mult1 <- function(x,mat,str1,str2) {
  ## $flag matrices - use 'sapply', FUN=is.matrix
  flag <- sapply(x, FUN=is.matrix)

  ## check if $x has 'any' matrices, otherwise 'return' $str1
  if(!any(flag)) return (str1)

  ## $x contains matrices! make index vector $idx from $flag
  idx <- which(flag)

  ## initialize matrix $counter to 0, and empty $result list
  counter <- 0
  result <- list()
  ## loop over matrices (use $idx as loopindex)
  for (i in idx) {
    ## store $x in $temp
    x[[i]] -> temp
    ## check if dim of $x and $mat are compatible
    if (ncol(temp) == nrow(mat)) {
      counter <- counter + 1
      temp %*% mat -> result[[counter]]
    } ## end if
  } ## end loop over $idx

  ## check if $counter is still 0 then 'return' $str2
  ## otherwise 'return' $result
  if (counter == 0) {
    return (str2)
  } else {
    return (result)
  }
} ## end function definition
mult1(x=bar,mat=diag(2))

Error in mult1(x = bar, mat = diag(2)) :
  argument "str1" is missing, with no default
```

References

Argument Clinic. URL: wikipedia.org. Complete sketch on dailymotion.