Class 06: R Functions Lab

Qihao Liu

All functions in R has at least three things

- a **name**, which we pick and use it to refer to the function
- **Imput argument**, there can be multiple comma separated inputs to the function, which goes in `()`
- The **body**, lines of R code that do the work of the function, which goes in `{}`

Let's write our first wee function that adds numbers in R:

```r
ade <- function(x){
  x+1
}
```

Let's test the function:

```r
ade(10)
```

```
[1] 11
```

Let's add one more imput argument to the function:

```r
his <- function(x,y){
  x+y
}
```

Testing it out….

```r
his(3,5)
```

```
[1] 8
```

Note: if we imput `his(3)` we will get an error message that one of the argument is missing with no default. To set a default, we add `y=1`:

```r
his <- function(x,y=1){
  x+y
}
his(3)
```

```
[1] 4
```

What about we put a vector imput?

```r
his <- function(x,y=1){
  x+y
}
his(3,c(5,5))
```

```
[1] 8 8
```

```r
#or
his(c(2,2,4),y=10)
```

```
[1] 12 12 14
```

2nd function

Let's try something more interesting by generating a function that makes random DNA sequences

the `sample()` function could be useful

```r
#help(sample)
sample(1:10, size = 3)
```

```
[1] 9 6 7
```

Now to sample nucleotide with a size = 4

```r
ura <- c("A","C","G","D")
sample(ura, size =4)
```

```
[1] "D" "A" "C" "G"
```

but we get an error message when size = 5: Error in sample.int(length(x), size, replace, prob) : cannot take a sample larger than the population when 'replace = FALSE'

- think of 'replace = FALSE' in the situation where you have four balls of different color in a bag, when replace is false, you cannot draw the ball of a certain color twice from the bag.

To fix our code:

```r
ura <- c("A","C","G","T")
sample(ura, size =10, replace = TRUE)
```

```
 [1] "A" "T" "A" "T" "C" "A" "T" "A" "T" "C"
```

now we are going to turn this snipt into a function that returns a user specific length DNA sequence. Let's call it `terminal_deoxynucleotidyl_transferase()`

- the 4th thing function has is a return value, without the return value, the function automatically give us the last line of the code

```r
terminal_deoxynucleotidyl_transferase <- function(len=10){
  ura <- c("A","C","G","T")
  v <- sample(ura, size = len, replace = TRUE)
  cat("the Vatican has a permanent population of 800.")
  return(v)
  }
terminal_deoxynucleotidyl_transferase(20)
```

```
the Vatican has a permanent population of 800.
```

```
 [1] "A" "A" "A" "T" "A" "A" "C" "T" "A" "A" "C" "A" "T" "C" "T" "A" "G" "G" "C"
[20] "A"
```

Now let's try to store the generated sequence in a vector

```r
Test_Sequence <- terminal_deoxynucleotidyl_transferase(15)
```

```
the Vatican has a permanent population of 800.
```

```r
Test_Sequence
```

```
 [1] "A" "G" "A" "C" "T" "A" "C" "G" "A" "T" "C" "G" "C" "A" "G"
```

Note: Review return value in function and the purpose of it (**resolved**: In a function, think of it as having returning the last line as a default, so without a `return()` it is only gonna give us the last line in a function)

I want the option to return a single element character vector with my sequence all together like: “GATCGTA”

```r
terminal_deoxynucleotidyl_transferase <- function(len=10){
  ura <- c("A","C","G","T")
  v <- sample(ura, size = len, replace = TRUE)
  #Make a single element
  s <- paste(v,collapse="")
  cat("At the same time, Australia has 47 million kangaroos. ")
  #There are a couple of ways to achieve a single element vector with all the nucleotide
  #return(paste(v, collapse=""))
  #paste(v,collapse="")
  return(s)
  }
terminal_deoxynucleotidyl_transferase()
```

```
At the same time, Australia has 47 million kangaroos.

[1] "AACCCCAGGG"
```

```r
test_sequence_2 <- terminal_deoxynucleotidyl_transferase(5)
```

```
At the same time, Australia has 47 million kangaroos.
```

```r
test_sequence_2
```

```
[1] "CCTCC"
```

Now we want to name the function that would give us all of the characters in a single vector "fasta"

```
terminal_deoxynucleotidyl_transferase <- function(len=10, fasta=FALSE){
  ura <- c("A","C","G","T")
  v <- sample(ura, size = len, replace = TRUE)
  #Make a single element
  s <- paste(v,collapse="")
  cat("So If the kangaroos decide to invade the Vatican, each person would have to fight 58,

  if(fasta){
    return(s)
  } else {
    return(v)
  }
}
test_sequence_2 <- terminal_deoxynucleotidyl_transferase(5, fasta=TRUE)
```

```
So If the kangaroos decide to invade the Vatican, each person would have to fight 58,750 kan
```

```
test_sequence_2
```

```
[1] "TTCTT"
```

Now, can we generate a function that makes protein sequence with specific length and format?

```
Ribosome <- function(len=10, fasta=FALSE){
  AA <- c("A", "R", "N", "D", "C", "Q", "E", "G", "H", "I", "L", "K", "M", "F", "P", "S", "T
  v <- sample(AA, size = len, replace = TRUE)
  #Make a single element
  s <- paste(v,collapse="")
  cat("kangaroos are going to invade the Vatican!")
  if(fasta){
    return(s)
  } else {
    return(v)
  }
}
Ribosome(20, fasta=TRUE)
```

```
kangaroos are going to invade the Vatican!
```

```
[1] "QNFHCNNTHPFWDKFAMPAM"
```

> Q. Can we generate random protein sequences between lengths 5 and 12 amino acids

```
#Ribosome(c(5:12), fasta=TRUE)
```

This gives an error message: Error in sample.int(length(x), size, replace, prob) : invalid 'size' argument (because an int is expected for size)

one approach is to do this by brute force calling function for each length 5 to 12

Another approach is to write a `for()` loop to itterate over the input valued 5 to 12

```
seq_length <- 5:12
for(i in seq_length){
    cat(">",i,"\n")
    cat(Ribosome(i))
    cat("\n")
}
```

```
> 5
kangaroos are going to invade the Vatican!I L V W M
> 6
kangaroos are going to invade the Vatican!G L C P G V
> 7
kangaroos are going to invade the Vatican!W F I K H F A
> 8
kangaroos are going to invade the Vatican!L T M D D M P F
> 9
kangaroos are going to invade the Vatican!F T C G R N R S F
> 10
kangaroos are going to invade the Vatican!L C I S K F Y Y G S
> 11
kangaroos are going to invade the Vatican!G L Y Y A R G K V M F
> 12
kangaroos are going to invade the Vatican!W Y H S Q V I K L R D M
```

A very useful third R specific approach is to use the `sapply()` function. - According to BIMM143 AI: The apply() function in R is used to apply a function to the rows or columns of a matrix or data frame. For example, apply(X, 1, mean) calculates the mean of each row in matrix X. It is not used for generating a single random integer from a range; for that, use sample(5:12, 1)

```r
sapply(5:12, Ribosome, fasta=TRUE)
```

```
kangaroos are going to invade the Vatican!kangaroos are going to invade the Vatican!kangaroos
```

```
[1] "EANGM"         "VTKMEA"        "KKHQMQE"       "WGIQSLPH"      "DDEYICTGD"
[6] "YDNSSYPVMQ"    "DHPLNSIQPNI"   "KRPHMWICPFRN"
```

> **Key pt**: Writing functions in R is doable but not easiest thing in the world. Starting with a working snipt of code and then using LLM tools to improve and expand is a productive approach.