class06

Dalena (PID: A17327787)

Table of contents

1. Function basics 1
2. Generate DNA sequence 2
3. Generate Protein function 3

1. Function basics

Let's start writing our first function to add some numbers:

Every R function has 3 things:

- name (we get to pick this)
- input arguments (there can be loads of these separated by a comma)
- the body (the R code that does the work)

```
add <- function(x,y=10,z=0){
  x + y + z
}
```

I can just use this function like any other function as long as R knows about it (i.e. run the code chunk)

```
add(1, 100)
```

```
[1] 101
```

```
add(x=c(1,2,3,4),y=100)
```

```
[1] 101 102 103 104
```

```r
add(1)
```

```
[1] 11
```

Functions can have “required” input arguments and “optional” input arguments. The optional arguments are defined with an equals default value (ex: `y=10`) in the function definition.

```r
add(x=1,y=100,z=10)
```

```
[1] 111
```

> Q. Write a function to return a DNA sequence of a user specified length? Call it `generate_dna()`

The `sample()` function can help here

```r
#generate_dna <- function(size=5){}

students <- c("jeff","jeremy","peter")

sample(students, size = 5, replace = TRUE)
```

```
[1] "peter" "peter" "peter" "jeff"  "peter"
```

2. Generate DNA sequence

Now work with `bases` rather than `students`

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

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

Now I have a workign `snippet` of code. I can use this as the body of my first function version here:

```r
generate_dna <- function(size=5){
  bases <-c("A","C","G","T")
sample(bases, size=size, replace=TRUE)
}
```

```r
generate_dna(100)
```

```
  [1] "T" "C" "G" "G" "T" "T" "G" "C" "C" "G" "C" "C" "G" "A" "C" "G" "C" "A"
 [19] "G" "G" "G" "C" "C" "A" "T" "T" "C" "G" "T" "A" "C" "T" "A" "A" "G" "A"
 [37] "T" "A" "G" "T" "A" "C" "G" "T" "A" "A" "A" "G" "C" "G" "C" "T" "G" "G"
 [55] "A" "C" "A" "G" "T" "G" "T" "C" "C" "C" "T" "T" "T" "G" "T" "A" "C" "A"
 [73] "A" "A" "A" "C" "G" "T" "A" "C" "A" "A" "A" "C" "C" "T" "G" "G" "A" "A"
 [91] "C" "T" "A" "C" "G" "G" "T" "G" "C" "C"
```

```r
generate_dna()
```

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

I want the ability to return a sequence like "AGTACCTG" i.e. a one element vector where all the baes are all together.

```r
generate_dna <- function(size=5, together=TRUE) {
  bases <-c("A","C","G","T")
  sequence <- sample(bases, size=size, replace=TRUE)

  if(together){
    sequence<-paste(sequence, collapse ="")
  }
  return(sequence)
}
```

```r
generate_dna()
```

```
[1] "ACGAG"
```

```r
generate_dna(together=FALSE)
```

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

3. Generate Protein function

We can get the set of 20 natural amino-acids from the **bio3d** package.

```
aa <- bio3d::aa.table$aa1[1:20]
```

> Q. Write a protein sequence generating function that will return sequences of a user specified length.

```
generate_protein <- function(size=7, together=TRUE) {
  ## Get the 20 amino-acids as a vector
  aa <-bio3d::aa.table$aa1[1:20]
  aa_sequence <- sample(aa, size, replace=TRUE)
  ## Optionally return a single element string
  if(together){
    aa_sequence<-paste(aa_sequence, collapse="")
  }
  return(aa_sequence)
}
```

```
generate_protein()
```

```
[1] "TGHGHHC"
```

> Q. Generate random protein sequences of length 6 to 12 amino acids.

```
## generate_protein(size=6:12)
### this code does not work to answer this question
```

We can fix this inability to generate multiple sequences by either editing and adding to the function body code (e.g. a for loop) or by using the R **apply** family of utility function.

```
## Use sapply(list/x, FUN)
ans<-sapply(6:12, generate_protein)
ans
```

```
[1] "HNARPD"        "ACFLKKR"       "RRVESNYE"      "CQMIIDTGD"     "ASPTKCSCFN"
[6] "RMSHADMGQMY"   "WHFTKCRFADQW"
```

It would be cool and useful if I could get FASTA format output.

I want this to look like:

```
>ID.6
HMMNMN
>ID. 7
FAKKAWL
>ID. 8
WQTSTTPD
```

The functions paste and cat can help here. Paste allows you to combine strings. Cat also concatenates (but useful for user-defined functions, converting to characters, appends using sep):

```
cat(ans,sep="\n")
```

```
HNARPD
ACFLKKR
RRVESNYE
CQMIIDTGD
ASPTKCSCFN
RMSHADMGQMY
WHFTKCRFADQW
```

```
cat(paste(">ID.",7:12,ans,"\n",sep=""),sep="\n")
```

```
>ID.7HNARPD

>ID.8ACFLKKR

>ID.9RRVESNYE

>ID.10CQMIIDTGD

>ID.11ASPTKCSCFN

>ID.12RMSHADMGQMY

>ID.7WHFTKCRFADQW
```

```
## We can condense this
```

```r
id.line <-paste(">ID.",6:12,sep="")
seq.line <-paste(id.line,ans,sep="\n")
cat(seq.line, sep="\n")
```

```
>ID.6
HNARPD
>ID.7
ACFLKKR
>ID.8
RRVESNYE
>ID.9
CQMIIDTGD
>ID.10
ASPTKCSCFN
>ID.11
RMSHADMGQMY
>ID.12
WHFTKCRFADQW
```

> Q. Determine if these sequences can be found in nature or are they unique? Why or why not?

I BLASTp searched my FASTA format sequences against NR and found that lengths 6, 7,& 8 are not unique and can be found in databases with 100% coverage and 100% identity.

Random sequences of length 9 and above are unique and can't be found in the databases.

IDs used in BLASTp search:

> ID.6 MSGPYT ID.7 EQCGPKS ID.8 RTTYQNAQ ID.9 DLIEEYVLD ID.10 VMNLIKYFKN ID.11 YQYSETLWVEI ID.12 QFDTEKYWPAHG"