class06

Hailey Heirigs (PID: A16962278)

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

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

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
```

1. R functions

Every R function has 3 things. - name (we get to pick this) - input arguments - the body

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

```
add(1)
```

```
[1] 11
```

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

```
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

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

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

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

```
[1] "peter"  "peter"  "jeremy" "jeremy" "peter"
```

2. Generate DNA sequences

Now work with `bases` rather than `students`

```
bases <- c("A", "C", "G", "T")

sample(bases, size = 10, replace = TRUE)
```

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

Now I have a working snippet of code & the body of my first function is 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" "T" "C" "C" "C" "A" "T" "G" "A" "T" "T" "G" "T" "G" "A" "C" "A"
 [19] "T" "C" "A" "A" "G" "A" "A" "C" "G" "G" "A" "G" "T" "T" "T" "G" "C" "A"
 [37] "C" "T" "A" "T" "C" "A" "C" "T" "A" "G" "G" "A" "A" "C" "T" "C" "T" "G"
 [55] "G" "C" "G" "A" "C" "A" "G" "C" "C" "C" "G" "G" "A" "A" "T" "T" "A" "C"
 [73] "C" "C" "C" "G" "T" "T" "G" "A" "C" "A" "C" "C" "A" "A" "G" "T" "G" "A"
 [91] "T" "C" "G" "T" "T" "C" "G" "T" "T" "C"
```

```r
generate_dna()
```

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

I want the ability to return a sequence like "AGTACCTG" i.e. a one element vector where the bases 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] "GTGTC"
```

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

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

3. Generate Protein function

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

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

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

```
aa
```

```
 [1] "A" "R" "N" "D" "C" "Q" "E" "G" "H" "I" "L" "K" "M" "F" "P" "S" "T" "W" "Y"
[20] "V"
```

and use this in our function

```
generate_protein <- function(size=6, together=TRUE) {
  ## Get the 20 amino-acids as a vector
  aa <- bio3d::aa.table$aa1[1:20]
  sequence <- sample(aa, size, replace=TRUE)

  ## Optionally return a single element string
  if(together){
    sequence <- paste(sequence, collapse = "")
  }
  return(sequence)
}
```

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

```
generate_protein(7)
```

```
[1] "KDWLFFG"
```

```
generate_protein(8)
```

```
[1] "QREKQIME"
```

```
generate_protein(9)
```

```
[1] "QHMKQDANK"
```

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 functions.

```
sapply(6:12, generate_protein)
```

```
[1] "FFKENL"       "KAAVIDR"      "NNKYIFEW"     "ATCVTYWVD"    "GFKDPHVTDF"
[6] "DAYSESDSQRS"  "RYGNFTYDSQHA"
```

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

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

```
ans <- sapply(6:12, generate_protein)
ans
```

```
[1] "EYHIYR"       "NPYWAFL"      "PKYEHRNW"     "WQMFSYHSY"    "GFDQLGMFTP"
[6] "GPPQACRQCNC"  "ITKWNIYGDNWG"
```

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

```
EYHIYR
NPYWAFL
PKYEHRNW
WQMFSYHSY
GFDQLGMFTP
GPPQACRQCNC
ITKWNIYGDNWG
```

I want this to look like

```
>ID.6
QCRAWKLHID
>ID.7
TYNVGCMDFNT
>ID.8
EVLPAFWHLWIV
```

The functions `paste()` and cat()‘ can help us here…

```
ans2 <-paste(">ID.", 7:12, "\n",ans, sep="")
cat(ans2, sep = "\n")
```

```
>ID.7
EYHIYR
>ID.8
NPYWAFL
>ID.9
PKYEHRNW
>ID.10
WQMFSYHSY
>ID.11
GFDQLGMFTP
>ID.12
GPPQACRQCNC
>ID.7
ITKWNIYGDNWG
```

> 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 the databases with 100% coverage and 100% identity.

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

You can add options to executable code like this

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).

```
1+1
```

```
[1] 2
```