Class06WBray

WBray

```
ADD <- function(x,y){x+y
}
ADD(24.15,111)
```

```
[1] 135.15
```

you have to run the function before calling it; will show up in the functions tab

```
ADD(1,1)
```

```
[1] 2
```

```
ADD(x=1, y=100)
```

```
[1] 101
```

```
ADD(x=c(100,1,100),1)
```

```
[1] 101   2 101
```

fOR the last example there, x is a vector, and y is simply 1, as a result, 100+1, 1+1, 100+1 are the result.

```
Barry <- function(x, y=1) {x + y}
Barry(13)
```

```
[1] 14
```

By setting the y=1, this is setting the DEFAULT value, while if you include a y value directly, it will overwrite.

If you put a ? in front of a function and run it (ideally in the console), R studio will show output in the help section to the right...

```
Barry(1,1)
```

```
[1] 2
```

Make a function "generate_dna" that makes a random nucleotide sequence of a specified length

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

```
generate_dna(12.45)
```

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

```
amino <- unique(bio3d::aa.table$aa1)[1:20]
```

This is using the bio3d package to look only at the contents of the aa table, have looked only at aa1, and one entry per amino acid, stopping after the first 20

```
generate_pro <- function(length){
 amino <- unique(bio3d::aa.table$aa1)[1:20]
sequence <- sample(amino, size=length, replace=TRUE)
sequence <- paste(sequence, collapse = "")
return(sequence)
}
```

The second sequence assignation here is to use the paste function Adding the return sequence section here ensures that this is explicitly the last thing that the code returns.

```
generate_pro(28)
```

```
[1] "DENCMNCYPTMQNFYEHPTALVRDKPEG"
```

appears functional!

```
generate_pro(6)
```

```
[1] "QGFEQF"
```

```
generate_pro(7)
```

```
[1] "EKNCGRT"
```

```
generate_pro(8)
```

```
[1] "NGDPNIET"
```

```
generate_pro(9)
```

```
[1] "MSYKCKHIH"
```

```
generate_pro(10)
```

```
[1] "LVYKVVAFMS"
```

```
generate_pro(11)
```

```
[1] "SAERMPQFMCE"
```

```
generate_pro(12)
```

```
[1] "PMFTSADWYIGK"
```

```
generate_pro(13)
```

```
[1] "KMTPYRSYKCHVW"
```

Alternatively, we could use a set of super useful functions called "Apply()" or "SAPPLY()", so to generate sequences of length 6 to 12 using SAPPLY! X is a vector, and fun is a function applicable… Apply is used for matrices etc

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

```
[1] "GMMFYK"       "HIRMGFS"      "VPQNYIRW"     "YIMHTKPYS"    "CHIYMPCSRF"
[6] "GEGDIRGERLT"  "GQTMGWQYMQYD"
```

```
answer <- sapply(6:12, generate_pro)
```

```
cat(paste(">id.", 6:12, "\n", answer, sep=""), sep="\n")
```

```
>id.6
PFVYIV
>id.7
WVIDWEE
>id.8
CMTVLFWK
>id.9
PTVMPNPHI
>id.10
DEDAWVGWAT
>id.11
FFKHRGQHSME
>id.12
IQCGTLAVYSTY
```

check for 100% identical, 100% coverage But we'd like the letters to not have quotes! Let's revisit the paste function

ex: paste(c("barry,"Alice", "amy", "Chandra""),"loves r", sep=" ")

```
# Can you improve this analysis code?
library(bio3d)
s1 <- read.pdb("4AKE") # kinase with drug
```

```
  Note: Accessing on-line PDB file
```

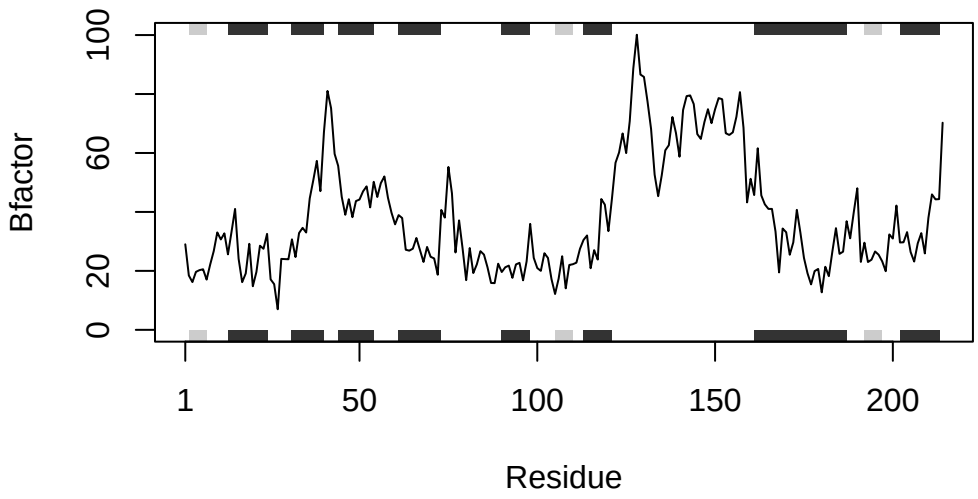
```
s2 <- read.pdb("1AKE") # kinase no drug
```

```
  Note: Accessing on-line PDB file
   PDB has ALT records, taking A only, rm.alt=TRUE
```

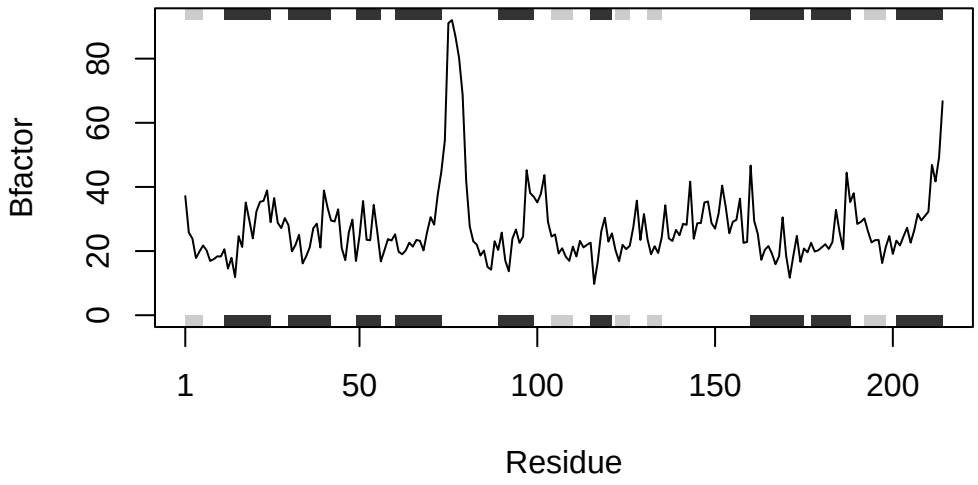
```
s3 <- read.pdb("1E4Y") # kinase with drug
```

```
  Note: Accessing on-line PDB file
```

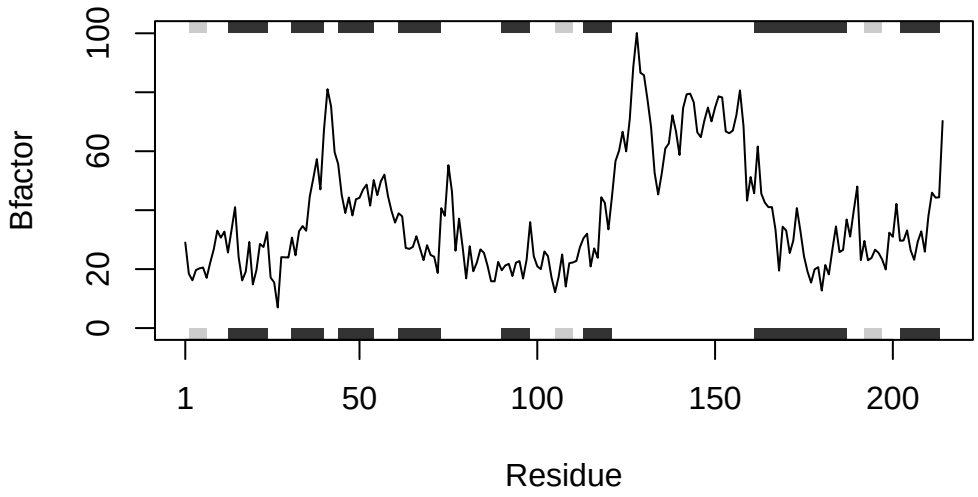
```
s1.chainA <- trim.pdb(s1, chain="A", elety="CA")
s2.chainA <- trim.pdb(s2, chain="A", elety="CA")
s3.chainA <- trim.pdb(s1, chain="A", elety="CA")
s1.b <- s1.chainA$atom$b
s2.b <- s2.chainA$atom$b
s3.b <- s3.chainA$atom$b
plotb3(s1.b, sse=s1.chainA, typ="l", ylab="Bfactor")
```



```
plotb3(s2.b, sse=s2.chainA, typ="l", ylab="Bfactor")
```




```
plotb3(s3.b, sse=s3.chainA, typ="l", ylab="Bfactor")
```



```
cat(paste(">id.", 1:3, "\n", c("barry","chris","alice"), "\n",sep=""))
```

```
>id.1
barry
 >id.2
chris
 >id.3
alice
```