Counting the number of odd numbers:

oddcount <- function(x) {

# Set k to be 0

k <- 0

for(n in x) {

# %% finds remainder on division

if(n %% 2 == 1) k <- k + 1

}

return(k)

}

oddcount(c(1,3,5))

oddcount(c(1,2,3,7,9))

Counting the number of prime numbers in a vector:

primecount <- function (x){

# set k to zero

k <- 0

for (n in x) {

# checking divisors

if(all (n %% seq (2, n-1) != 0)) k <- k +1

# dealing with 1 and 2

else if(n == 1 | n == 2) k <- k + 1}

return (k)

}

primecount(c(1,2,3,44))

primecount(c(1,2,3,44,30,7,7,7,6))

(to be improved) finding runs of 1 given a vector and the length of run:

findruns <- function(x, k) {

n <- length(x)

runs <- NULL

for(i in 1:(n - k + 1)) {

if(all(x[i:(i + k - 1)] == 1)) runs <- c(runs, i)

}

return(runs)

}

y <- c(1, 0, 0, 1, 1, 1, 0, 1, 1)

findruns(y, 3)

findruns(y, 1)

**simple function with return:**

u <- 1:5

w <- function(x) {return(x + 3)}

w(u)

**# I can't believe this works!!! removes all numbers divisible by 4 from a vector:**

**x4 <- 2 4 6 8 10 12 14**

**x4 <- x4[-c(which(x4 %% 4 == 0))]**

better predict function

predb <- function(x, k) {

n <- length(x)

k2 <- k / 2

pred <- vector(length = n - k)

**sm <- sum(x[1:k])**

if(sm >= k2) pred[1] <- 1 else pred[1] <- 0

if(n - k <= 2){

for(i in 2:(n - k)){

sm <- sm + x[i + k - 1] - x[i - 1]

if(sm >= k2) pred[i] <- 1 else pred[i] <- 0

}

}

return(mean(abs(pred - x[(k + 1):n])))

}

->>> **sm <- sum(x[1:k])**

this creates an ibject with a sum of the first k elements of x

**third version of predict function with cumsum()**

predc <- function(x, k) {

n <- length(x)

k2 <- k/2

pred <- vector(length = n - k)

csx <- c(0, cumsum(x))

for(i in 1:(n - k)){

if(csx[i + k] - csx[i] >= k2) pred[i] <- 1 else pred[i] <- 0

}

return(mean(abs(pred - x[(k + 1):n])))

}

**# prime count function applying square root rule**

primecount2 <- function (x){

# set k to zero

k <- 0

for (n in x) {

# checking divisors

if(all (n %% seq (2, sqrt(n)) != 0)) k <- k +1

# dealing with 1 and 2

else if(n == 1 | n == 2) k <- k + 1}

return (k)

}

**vector append comparison:**

z <- NULL

for(i in 1:5) z <- c(z, i)

this results in: [1] 1 2 3 4 5

# compared to:

z2 <- NULL

for(i in 1:5) z2 <- c(i)

this results in: [1] 5

word concordance:

# reads text, separates into vector of words

# create a list to store the words and their positions

#list of positions of words with the words as tags

findwords2 <- function(tf){

txt <- unlist(strsplit(tf, ' '))

word\_list <- list()

for(i in 1:length(txt)){ # loop through each word

word <- txt[i] # get the current word

word\_list[[word]] <- c(word\_list[[word]], i)

}

return(word\_list)

}

text corcondance

# we want to create a function which lists thge different words used in

# a text and their position (non letter characters are removed)

firstpart <- "It was a bright cold day in April and the clocks were striking thirteen Winston Smith his chin nuzzled into his breast in an effort to escape the vile wind slipped quickly through the glass doors of Victory Mansions though not quickly enough to prevent a swirl of gritty dust from entering along with him"

firstpart

# findwords function

findwords2 <- function(tf){

**txt <- unlist(strsplit(tf, ' '))** # reads text, separates into vector of words

word\_list <- list() # create a list to store the words and their positions

for(i in 1:length(txt)){ # loop through each word

word <- txt[i] # get the current word

word\_list[[word]] <- c(word\_list[[word]], i) #**list of positions of words**

**# with the words as tags**

}

return(word\_list)

}

i in the loop are numbers (as many as words in the list)

**# find words in alphabetical order (to be used by previous function)**

alpha\_word\_list <- function(word\_list){

nms <- names(word\_list) # find the tags of the list

sort\_names <- sort(nms) # sort the words (tags) alphabetically

return(word\_list[sort\_names])

}

alpha\_word\_list(findwords2(firstpart))

**# sorting by frequency**

freq\_word\_list <- function(word\_list){

freq\_word\_list <- sapply(word\_list, length) # find the length of each list element i.e. how many entries for each tag (tags=words, entries=frequencies)

return(word\_list[order(freq\_word\_list)]) # sort the words (tags) in order

}

freq\_word\_list(findwords2(firstpart))

fun1 <- **function**(x, printx = TRUE, printy = TRUE) { y <- x**^**2

**if**(printx) **print**(x) *# if(printx) is the same as if(printx == TRUE)*

**if**(printy) **print**(y) *# if(printy) is the same as if(printy == TRUE)*

**return**(y)

}

# Given a vector, e.g. (5, 4, 12, 13, 3, 8, 88) this algorithm compares

# the first element 5 with the others and produces two sub-vectors; the first

# with all the elements smaller than it 4, 3, the second with all the elements

# bigger than it 12, 13, 8, 88 and re-runs quicksort on these two sub-vectors.

**quicksort** <- function(x) {

if(length(x) <= 1) return(x)

pivot <- x[1] # take the first value as pivot

therest <- x[-1] # create a new vector with all values but the pivot

sv1 <- therest[therest < pivot] # subset of values smaller than the pivot

sv2 <- therest[therest >= pivot] # subset of values greater than or = the pivot

sv1 <- quicksort(sv1) # quicksort the values smaller than the pivot

sv2 <- quicksort(sv2) # quicksort the values smaller than the pivot o

return(c(sv1, pivot, sv2))

}

recursive **factorial** function

fact <- function(x) {

if(x == 1) return(x)

return(x \* fact(x - 1))

}

• What is the probability of getting at least 4 heads from 5 tosses of a coin?

1 - pbinom(3, 5, 0.5)

# Pr(X >= 4) = 1 - Pr(X <= 3)

## [1] 0.1875

# or

pbinom(3, 5, 0.5, lower.tail = FALSE) # Pr(X >= 4) = Pr(X > 3)

Suppose we want to find the mean of the maximum of two standard normal random variables.

x <- rnorm(nreps)

y <- rnorm(nreps)

maxxy <- pmax(x, y)

mean(maxxy)

sample(x, size, replace = FALSE, prob = NULL)

sample takes a sample of the specified size from the elements of x using either with or without replacement.

**# greatest common divisor of 2 numbers**

gcd <- function(a,b){

if (b == 0) return(a)

else return(gcd(b, a %% b))

}

**derivatives and curves of functions and derivatives**

function(x) x^2 \* sin(x)

D(expression(x^2 \* sin(x)), 'x') # derivative

x <- 5 # we want to substitute 5 for x, to evaluate at this value, use eval()

eval(D(expression(x^2 \* sin(x)), 'x'))

curve(x^2 \* sin(x), from= 0, to= 10) # graph of function

curve(eval(D(expression(x^2 \* sin(x)), 'x')), from=0, to= 10, add=T)

# graph the derivative

**integrating pnorm function:**

integrate(dnorm, -1.96, 1.96, mean=2, sd=1)

integrate(dnorm, -Inf, 2, mean=0, sd=1)

**ordering a matrix**

d <- data.frame(kids = c('Jack', 'Jill', 'Billy'), ages = c(12, 10, 13))

d[order(d$kids),]

d[order(d$ages),]

sd() function gives sample sd, how to get population sd?

n <- length(student.height)

sd(student.height) \* sqrt((n-1)/n)

r <- runif(10000)

meanr <- mean(r)

sdr <- sd(r)

sample(r, 4)

cl4 <- mean(sample(r, 4)) # mean of one sample

for (i in 1:1000){

cl4 <- c(cl4, mean(sample(r, 4)))

} # drawing 1000 samples and appending their means to the vector

hist(cl4)

taking samples to demonstarte central limit theorem – **appending** vectors

finding min and max value in a table:

Office Mean\_age

1 Boston 41.50000

2 Harvard 41.89474

3 Providence 45.75000

t3[which.min(t3[,2]),1]

t3[which.max(t3[,2]),1]

# Example 1: automatically forming files and file names

for(i in 1:5) {

fname <- paste('N(0,', i, ').pdf', sep = '')

pdf(fname)

hist(rnorm(100, sd = i))

dev.off()

}

# Example 2: find files in a directory ending in a certain type:

list.files(getwd(), '\\.txt$')

# Example 3 web scraping and then cleaning up

library(RCurl)

oct17 <- getURL("https://stat.ethz.ch/pipermail/r-help/2017-October/date.html")

webpage <- strsplit(oct17, "\n")[[1]] # download the webpage see slide 21

# get the lines that contains the authors

# [the authors are in italic, so html code starts with "<I>"]:

authorsraw <- grep("<I>", webpage, value = TRUE)

authors <- gsub("<I>", "", authorsraw, fixed = TRUE) # only keep the names

# create a table that contains the number of contributions for each author,

# and sort it in decreasing order:

author\_counts <- sort(table(authors), decreasing = TRUE)

author\_counts[1:5]