In a sense it allows a proficient reader to see words and know them in the same way that we [see people’s faces and we know them](https://en.wikipedia.org/wiki/Fusiform_face_area) (if we’ve encountered them before). Essentially, phonics is useful, particularly at certain points in our reading development but is rather inefficient and not the work horse of a proficient reader’s reading process. Instant word recognition is required for fluency and comprehension. For additional information on the reading process see the video below.

Cohen, L., Dehaene, S., Naccache, L., Lehéricy, S., Dehaene-Lambertz, G., Hénaff, M. A., Michel, F. (2000). The visual word form area: Spatial and temporal characterization of an initial stage of reading in normal subjects and posterior split-brain patients. *Brain. 123*(2). 291–307. doi:10.1093/brain/123.2.291. ISSN 0006-8950. PMID 10648437

As I prepared for class I wanted to demonstrate two points about the VWFA to students:

1. General shape of words is an attribute used by the VWFA
2. The first and last letter are very important to instant word recognition; the exact ordering of the individual letters (graphemes) of the middle portions of words less important

The first can be evidenced by altering the case of letters within words and seeing it does indeed slow down reading rate. The second can be demonstrated by randomly reordering the inner letters within a word. The amount of case changing or reordering of letters are parameters that can be changed and can slow down reading rate in varied ways. What better way to demonstrate this than using R to reason and programatically allow the testing of theory. The two sections below show R code that tests the (1) altering case theory of the VWFA and (2) the lowered importance of the ordering of the middle letters of words.

First you’ll need to install my [**textshape**](https://cran.r-project.org/web/packages/textshape/index.html) package to get started:

if (!require("pacman")) install.packages("pacman"); library(pacman)

pacman::p\_load(textshape)

**Altering Case Effects**

Mayall, Humphreys & Olson (1997) show that letter case randomization can disrupt the ability to process words. If true, this is evidence that the VWFA (if it exists) uses a general shape attribute for word recognition since mixing case alters shape not letters.

[Mayall, K., Humphreys, G. W., & Olson, A. (1997)](https://www.ncbi.nlm.nih.gov/pubmed/9293635). Disruption to word or letter processing?: The origins of case-mixing effects. Journal of Experimental Psychology: Learning, Memory, and Cognition, 23(5), 1275-1286. 10.1037/0278-7393.23.5.1275

The R script below is a function that takes text and randomly replaces a set proportion of lower case letters with upper case. In the script below I show some text that has 2%, 10%, and 50% (worst case; no pun intended) of lower case letters randomly replaced with upper case letters. The reader can informally see that indeed the letters are the same but the picture quality seen by the brain reduces the ability to process the words. Secondly, 2% change is less disruptive than 50%. This is evidence that there is a VWFA and one of the attributes it uses is word shape.

#' Randomly Change the Case of Letters Within Words

#'

#' Following Mayall, Humphreys, & Olson (1997), this function randomly

#' converts a proportion of lower case letters to upper case.

#'

#' @param x A vector of text strings to upper case.

#' @param prop A proportion of graphemes to change the case of.

#' @param wrap An integer value of how wide to wrap the strings. Using the default

#' \code{NULL} disables this feature.

#' @param \ldots ignored.

#' @return Prints wrapped lines with internal graphemes randomly converted to

#' upper case.

#' @references

#' Mayall, K., Humphreys, G. W., & Olson, A. (1997). Disruption to word or letter

#' processing?: The origins of case-mixing effects. Journal of Experimental Psychology:

#' Learning, Memory, and Cognition, 23(5), 1275-1286. 10.1037/0278-7393.23.5.1275

#' @export

random\_upper 0 & prop <= 1)

## splits each string to a vector of characters

char\_vects <- strsplit(x, '')

## loop through each vector of characters and replace lower with upper case

out <- unlist(lapply(char\_vects, function(chars){

## detect the lower case locations

locs <- grepl('[a-z]', chars)

ilocs <- which(locs)

## sample lower case locations to convert to upper

to\_upper <- sample(ilocs, ceiling(prop \* length(ilocs)))

chars[to\_upper] <- toupper(chars[to\_upper])

## collapse the vector of characters back to its original string

paste(chars, collapse = '')

}))

## optional string wrapping

if (!is.null(wrap) && ![is.na](http://is.na)(as.integer(wrap))) {

invisible(Map(function(x, wrapchar) {

cat(strwrap(x, width = as.integer(wrap)), sep = '\n')

if (wrapchar) cat('\n')

}, out, c(rep(TRUE, length(out) - 1), FALSE)))

} else {

out

}

}

x <- "Many English words are formed by taking basic words and adding combinations of prefixes and suffixes to them."

## 10% random upper

random\_upper(x, .02, 60)

## MAny English words Are formed by taking basic words and

## adding combinations of prefixes and suffixes to them.

## 10% random upper

random\_upper(x, .3, 60)

## Many English words are forMed by taking basic worDs aNd

## adding Combinations of prEFixes and SuffIxeS to them.

## 50% random upper

random\_upper(x, .5, 60)

## MaNy EnglIsH WORDs ARE FoRmED BY taking BAsic WOrDs And

## AddiNg CombinATIoNS OF pRefIXEs AnD sUffIXES to thEM.

**Transposing Internal Letters**

Another interesting phenomenon is the transposing of letters within the middle of words. This was popularized as an Internet meme & hoax about research at Cambridge University:

fqfy4h2

While the claim the meme makes about the research and Cambridge wasn’t true, obviously, there is an element of truth to the inner word transpose effect noted by researchers in the 70s and 80s (e.g., [McCusker, Gough, & Bias, 1981](http://psycnet.apa.org/record/1982-07082-001)). Indeed the reader can still understand the message but there is a cognitive cost to scrambling letters ([Rayner, White, Johnson, & Liversedge, 2006](http://psycnet.apa.org/record/2006-05194-003)).

McCusker, L. X., Gough, P. B., & Bias, R. G. (1981). Word recognition inside out and outside in. Journal Of Experimental Psychology: Human Perception And Performance, 7(3), 538-551. doi:10.1037/0096-1523.7.3.538

Rayner, K., White, S. J., Johnson, R. L., & Liversedge, S. P. (2006). Raeding wrods with jubmled lettres: There is a cost. Psychological Science, 17(3), 192-193. 10.1111/j.1467-9280.2006.01684.x

The R code below allows the user to group the inner portion of words as character [ngrams](https://en.wikipedia.org/wiki/N-gram), reorder within these grams, and optionally reorder the position of the reordered ngram groups. Both the size of the ngrams and the reordering of ngram group position are parameters of the effect that we can alter and informally observe via our self reported effects in our ability to read the strings after altering various parameters. The larger the ngram unit the more the inner portion of words will be scrambled.

The sample.grams parameter allows us to see the effect of keeping scrambled ngram groups in their original position or not. Indeed the longer words are, and the more thorough the remix, the bigger the cost of the letter transpose is. When commonly (or expected) co-occurring ngrams are located randomly (far away from each other) this also may contribute to the cost on scrambling effect. In the final code chunk i allow the first and/or the last letter of words to be scrambled as well. This is evidence that the VWFA is keyed in on the first and last letters and that certain letters are expected to be close to one another.

#' Transpose Internal Letters Within Words

#'

#' Following a famous Internet meme and Rayner, White, Johnson, & Liversedge

#' (2006), this function randomly scrambles the internal (not the first or last

#' letter of > 3 character words) letters.

#'

#' @details Internet meme:

#'

#' It deosn't mttaer in waht oredr the ltteers in a wrod are, the olny iprmoetnt

#' tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be

#' a toatl mses and you can sitll raed it wouthit porbelm. Tihs is bcuseae the

#' huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe.

#'

#' @param x A vector of text strings to scramble.

#' @param gram.length The length of gram groups to scramble. Setting this lower

#' will keep expected graphemes close together. Setting it to a high value (e.g.,

#' 100) will allow the positions of graphemes to deviate farther from the expected

#' clustering.

#' @param sample.grams logical. If \code{TRUE} then the ngram groups don't retain

#' their original location. For example, let's say we had the sequence \code{123456}.

#' Sampling grams of length 3 (\code{gram.length} may produce \code{231564}. Setting

#' \code{sample.grams = TRUE} may further produce \code{564231}.

#' @param wrap An integer value of how wide to wrap the strings. Using the default

#' \code{NULL} disables this feature.

#' @param remix.first logical. If \code{TRUE} the first letter is allowed to be

#' remixed as well.

#' @param remix.last logical. If \code{TRUE} the last letter is allowed to be

#' remixed as well.

#' @param \ldots ignored.

#' @return Prints wrapped lines with internal graphemes scrambled.

#' @references

#' Rayner, K., White, S. J., Johnson, R. L., & Liversedge, S. P. (2006). Raeding

#' wrods with jubmled lettres: There is a cost. Psychological Science, 17(3),

#' 192-193. 10.1111/j.1467-9280.2006.01684.x

#' @export

random\_scramble <- function(x, gram.length = 2, sample.grams = TRUE, wrap = NULL,

remix.first = FALSE, remix.last = FALSE, ...){

if (!is.integer(gram.length)) gram.length 1))

## splits the strings into a list of tokens (words and punctuation)

token\_vects <- textshape::split\_token(x, lower = FALSE)

## loop through the vectors of tokens

out <- unlist(lapply(token\_vects, function(tokens){

## loop through the tokens within each vector

out 1) win <- sample(gram.length, 1) else win 3 because you can't transpose words less than 2 internal characters

## Note: the value of 3 depends if the first and last letters are allowed to vary

len <- nchar(token)

if (len < 4 - (remix.first + remix.last)) return(token)

## split tokens into characters and compute location of internal letters

chars <- strsplit(token, '')[[1]]

locs <- (2 - remix.first):(len - (!remix.last))

## If the length of the internal letters is <= the ngram window randomly

## sample internal letters, collapse characters, and return

if (length(locs) <= win) {

return(

paste(

c(

if (remix.first) '' else chars[1],

paste(sample(chars[locs]), collapse = ''),

if (remix.last) '' else chars[length(chars)]

),

collapse = ''

)

)

}

## Make gram groupings for all grams that match gram.length

## (exclude < gram.length char groups)

locs2 <- rep(1:floor(length(locs)/win), each = win)

## add in the < gram.length groups and store as list of lengths

## and group assignments

locs3 <- rle(c(locs2, c(rep(max(locs2) + 1, length(locs)%%win))))

## Resample the lengths to allow the odd group out (if there is one)

## to be located randomly rather than always at the end

locs4 <- rep(locs3$values, sample(locs3$lengths))

## split the vector of chars into the groups, loop through, sample

## within each group to scramble and collapse the group characters

rands <- unlist(lapply(split(locs, locs4), function(grams){

paste(sample(chars[grams]), collapse = '')

}))

## optionally scramble the group location as well

if (sample.grams) rands <- sample(rands)

## collapse the group gram strings together

paste(

c(

if (remix.first) '' else chars[1],

rands,

if (remix.last) '' else chars[length(chars)]

),

collapse = ''

)

}))

## Paste tokens back together with single space and attmpt to strip out

## inappropriate spaces before punctuation. This does not guarentee

## original spacing of the strings.

trimws(gsub("(\\s+)([.!?,;:])", "\\2", paste(out, collapse = ' '), perl = TRUE))

}))

## optional string wrapping

if (!is.null(wrap) && ![is.na](http://is.na)(as.integer(wrap))) {

invisible(Map(function(x, wrapchar) {

cat(strwrap(x, width = as.integer(wrap)), sep = '\n')

if (wrapchar) cat('\n')

}, out, c(rep(TRUE, length(out) - 1), FALSE)))

} else {

out

}

}

# example text from: <https://www.mrc-cbu.cam.ac.uk/personal/matt.davis/Cmabrigde/>

x <- c(

"According to a study at an English University, it doesn't matter in what order the letters in a word are, the only important thing is that the first and last letter be at the right place. The rest can be a total mess and you can still read it without problem. This is because the human mind does not read every letter by itself but the word as a whole.",

"A vehicle exploded at a police checkpoint near the UN headquarters in Baghdad on Monday killing the bomber and an Iraqi police officer",

"Big council tax increases this year have squeezed the incomes of many pensioners",

"A doctor has admitted the manslaughter of a teenage cancer patient who died after a hospital drug blunder."

)

## Bigram & retain mixed ngram group locations

random\_scramble(x, gram.length = 2, wrap = 70)

## Aroccnidg to a sduty at an Esngilh Uevrstiniy, it deosn't mteatr in

## what order the ltteers in a word are, the only ipmantrot tnihg is

## that the first and lsat letetr be at the rghit place. The rset can be

## a tatol mses and you can stlil read it wuoihtt pbleorm. Tihs is

## becuase the hamun mnid does not raed eevry lteetr by itself but the

## word as a whole.

##

## A vehicle exploedd at a polcie cinheckopt naer the UN hrtaueaerdqs in

## Baghadd on Monady kilnlig the bomber and an Iaqri police oceffir

##

## Big cncouil tax iescnreas tihs yaer hvae sezeequd the iecnmos of mnay

## pneerisons

##

## A docotr has aitmdetd the mhgtenalsuar of a teegane cnacer pneitat

## who died aetfr a hiptaosl durg bulednr.

## Bigram and reorder the mixed ngram groups

random\_scramble(x, gram.length = 2, sample.grams = FALSE, wrap = 70)

## According to a sutdy at an Enlgish Univesrity, it deosn't matetr in

## waht order the lteters in a word are, the olny impotrant thing is

## that the first and last letter be at the right plcae. The rset can be

## a toatl mess and you can still read it wihtuot problem. Tihs is

## because the hmuan mnid deos not raed eevry letetr by istlef but the

## wrod as a whole.

##

## A vehilce epxoledd at a police checkponit near the UN haedquarters in

## Bgahdad on Monady killing the bmoebr and an Iraqi polcie officer

##

## Big council tax icnreases this yaer have suqeezed the incoems of mnay

## penisonres

##

## A dcootr has amditetd the mnaslaughetr of a tenegae cnacer ptaeint

## who deid afetr a hosipatl drug blunder.

## Gram length randomly between 2-5 retain mixed ngram group locations

random\_scramble(x, gram.length = 2:5, wrap = 70)

## Accroindg to a sdtuy at an Esilgnh Unveisirty, it d'onset mtetar in

## waht oerdr the lerttes in a wrod are, the only iantormpt tihng is

## taht the frist and last lteetr be at the rhigt plcae. The rset can be

## a taotl mess and you can stlil read it wtuioht pelrobm. This is

## bacusee the haumn mnid deos not raed eervy letter by ieltsf but the

## wrod as a wolhe.

##

## A viclhee eplodxed at a picloe conikpehct near the UN huaqdearters in

## Bgadhad on Mnoady kililng the bbemor and an Iqari picole ofefcir

##

## Big cionucl tax ieeascnrs this year hvae seqeuezd the imeocns of mnay

## pnseiorens

##

## A dotocr has atedmitd the mslautenahgr of a tgaenee cecanr ptianet

## who died after a hatspiol drug bdnelur.

## 5-gram retain mixed ngram group locations

random\_scramble(x, gram.length = 5, wrap = 70)

## Accdnirog to a sdtuy at an Elisgnh Usrietnivy, it dnsoe't matetr in

## what oerdr the lrettes in a word are, the olny ianmrotpt tinhg is

## taht the frist and lsat leettr be at the rghit pcale. The rset can be

## a ttoal mses and you can sitll read it woutiht peborlm. Tihs is

## bceause the hmaun mind does not read evrey letetr by ilstef but the

## word as a wolhe.

##

## A vichlee exolpedd at a piolce conkipehct near the UN hrtearuqdeas in

## Bhagdad on Mndoay knliilg the boembr and an Iqrai poicle ofceifr

##

## Big cniuocl tax iseenracs this year have seeuqzed the ioncems of mnay

## pernsoines

##

## A docotr has atemtidd the mlsaanheugtr of a tgeenae canecr pteinat

## who died afetr a haptoisl durg beunldr.

let's ramp it up a bit more and see the effect when we allow the first and last letter to be remixed as well.

## Bigram & retain fixed ngram group locations & remix last letter

random\_scramble(x, wrap = 70, remix.last = TRUE)

## Angdiorcc to a sutyd at an Elignsh Uityinsrev, it dns'toe mretta in

## wtha oerrd teh lteters in a wdro aer, teh olny iorpmntat tngih is

## ttah teh frist adn ltsa lertte be at teh rgiht plaec. The rset can be

## a ttoal mses and yuo can sllti rdea it withotu porlbme. Tihs is

## bceesau the hnaum mdin dsoe nto rdae eveyr lttree by ilftse but teh

## wdro as a wohel.

##

## A velheci eedlodxp at a piloce chekctnipo nera teh UN heartqdrsaue in

## Bgaaddh on Modnay klingil teh berbom and an Iraiq pcieol ociffre

##

## Big cliuocn txa incasrese tsih yare heva sezqeued the iesomnc of mnya

## pissnoneer

##

## A dotroc has aittmdde the merhtnagusla of a teeegan cancre patient

## who ddie after a hitspoal durg berlund.

## Bigram & retain fixed ngram group locations & remix first letter

random\_scramble(x, wrap = 70, remix.first = TRUE)

## drinoccAg to a tsduy at an Ensiglh inUverstiy, it esod'nt tetamr in

## waht roder the reletts in a rowd are, hte lony rtminapot thing is

## that hte rsift and last eelttr be at hte ghrit aclpe. hTe rest can be

## a taotl sems and oyu acn still eard it thouwit obrpelm. hiTs is

## ebacuse hte uhamn mind deos ont eard ervey etletr by itself but the

## owrd as a hwloe.

##

## A hievcle olpxeded at a cilpoe nicepkohct aenr hte UN adtrehaureqs in

## daBaghd on ndaoMy kiinllg the bbemor and an aqIri polcie ofcefir

##

## iBg ocuncil tax eainrcess ihts eyar vahe ezsqueed hte ocinmes of namy

## isprenoens

##

## A dootcr has tdamited hte utelaamhgnsr of a gaentee caecnr entiapt

## who died teafr a taiphosl urdg deblunr.

## Bigram & retain fixed ngram group locations & remix first + last letters

random\_scramble(x, wrap = 70, remix.first = TRUE, remix.last = TRUE)

## Acicongdr to a yduts ta na Enlihsg tysiivUnre, it t'nsoed amtter ni

## what redro the terstel ni a owrd rae, the noly aimtroptn htign is

## htat the trsif dna stla tterel be ta eht igthr pleca. ehT erts cna eb

## a latot mess nad yuo nca illst daer it wihttuo mprleob. hTis si

## aubcese the hunam mind does otn ader every eltter by tiesfl but eht

## word sa a hwole.

##

## A ehvleci pldoxede ta a poilce ecntoikpch arne eht UN daehsrrauqte ni

## hdagBda on yaMond illking the rebmbo and na Iqiar poliec ofrecfi

##

## gBi unocilc txa niesscrae thsi arey have zedeueqs het nimocse fo amyn

## peiorssnne

##

## A roctod sah adimedtt the ugerlasnmaht fo a teeneag cncare patietn

## how ided etafr a hospitla gudr rnuedbl.

## 5-gram & retain fixed ngram group locations & remix first + last letters

random\_scramble(x, wrap = 70, gram.length = 5, remix.first = TRUE, remix.last = TRUE)

## dingrccoA to a duyst ta na gEinlsh nievUsrity, ti denso't meratt in

## twha drreo the rtestle in a owrd aer, eht ynol nattmiopr tgnhi is

## atth eth isfrt nad salt rtlete be at het trghi claep. Teh tser nac be

## a tlota mses nda ouy nac ltsli read ti witothu rpleobm. Tish si

## ebseuac teh hmaun imdn sdoe otn edra reeyv tetelr by tsilef but hte

## owdr as a ewohl.

##

## A lehviec epxdldeo ta a oplcie toipnehkcc nrae eth NU auqhedaesrrt ni

## dahgdBa no Mandoy lnlgiki het rombbe dna na rqaIi pcolie ofciefr

##

## gBi lnciuco txa eassencir tish raey ehav zdeeeqsu hte comnise fo mayn

## osnerniesp

##

## A doctro hsa edttmdia teh resmnlaathug fo a tneaege cancer nettipa

## woh eddi fatre a hosipatl rdug blnerdu.

This post showed how I recently used R for some quick theory testing and demonstration. Of course the code could be optimized but the point is quick exploration of concepts I'm reading in the literature.

Similar, quick, iterative, testing with R could be done by researchers/teachers across many fields. The field of visualization comes to mind. This could be made into a shiny app to allow non-technical users to still interact with the code. It is my hope that both the content and code is of interest.