ASKNBID DATA SCIENCE INTERNSHIP

NAME : PRAKHAR SHUKLA (11602506)

Email : prakharshukla20160@gmail.com

Institution : LOVELY PROFESSIONAL UNIVERSITY

Profile: DATA SCIENCE

ASKNBID ASSIGNMENT (DATA SCIENCE) Solution:-

Task 1 :-

NOTE:- Please do read this..

- I have made this **Hangman program using R language**.
- I have tried to develop my program as near possibly correct.
- I have tried my best and put my best efforts.

****

- In my program if we comment the lines 96,105-111.
- My program will always end up in finding the correct word , and it will always win.
- But if we keep these lines 96,105-111 uncommented, the program might win/lose in some cases.
- start()---> this will begin the program just write start() on console and then next line give input.

****

Program:-

```
consonants <- c("b","c","d","f","g","h","j","k","l","m","n","p","q",
               "r","s","t","v","w","x","y","z")
vowels <- c("a","e","i","o","u")
catn <- function(s) #this function is used for printing a variable and for next line
{
 cat(s,"\n",sep="")
}
start<-function() #main function ---->this starts the game ---takes an input first, give your input in the console
{
  inputword<-readline() # reading the string from console (user input)
  inputword<-tolower(inputword)
guessed_string<-NULL #guessed string--->will be storing the correctly guessed characters
missed<-NULL  #missed---> stores the wrong guessed characters
  x <- nchar(inputword)
len <- nchar(inputword)
#printing the empty base before the program makes a guess.
while (x != 0)
 {
   guessed_string=paste(guessed_string,'_',sep="")
   x = x - 1
   if (x == 0) {
     missed<-paste(missed,"missed :")
   }
 }
catn(c(guessed_string,missed))
attempt=0
# from here program starts guessing the letters, after 6 wrong attempts it will stop.
```

```
while(attempt!=6)
{
 val<-sample(vowels,size=1)
 val1<-sample(consonants,size=1)
 i=1
 if(str_detect(inputword,val)) #checks if the letter exists in the given string
 {
    cat("guess :",val,"\n\n")
    index<-str_locate_all(pattern=val,inputword)
    while(i<=len)
    {
     if(is.element(i,index[[1]]))
     {
       substr(guessed_string,i,i)<-val
     }
     i=i+1
    }
    if(guessed_string==inputword) #if matched print the output and program terminates here.
    {
     catn(c(guessed_string,missed))
     break;
    }
    catn(c(guessed_string,missed))
    val=""
  }
  else if(str_detect(inputword,val1))
  {
    cat("guess :" ,val1,"\n\n")
    index<-str_locate_all(pattern=val1,inputword)
    while(i<=len)
    {
```

```
      if(is.element(i,index[[1]]))
      {
        substr(guessed_string,i,i)<-val1
      }
      i=i+1
    }
    if(guessed_string==inputword)
    {
      catn(c(guessed_string,missed))
      break;
    }
    catn(c(guessed_string,missed))
    val1=""
  }
 else                                  #letters which do not match are stored in missed.
 {
   re<-str_locate_all(pattern=val,missed)
  re1<-str_locate_all(pattern=val1,missed)  #----------->line mentioned in the note at starting
   if(length(re[[1]])==0)  #condition makes sure if the letter is not repeated. If yes then attempt is not
counted.
    {
     cat("guess :",val,"\n\n")
    missed<-paste(missed,val)
     catn(c(guessed_string,missed))
     attempt=attempt+1
   }
   else if(length(re1[[1]])==0)  #----------->lines mentioned in the note at starting
   {
   cat("guess :",val1,"\n\n")
     missed<-paste(missed,val1)
     catn(c(guessed_string,missed))
```

```
    attempt=attempt+1
  }
 }
}}
start()   # ←-------- calls all the functions and initiates the program
```

****

Outputs generated while testing---------->

OUTPUT 1: ------status: program loses

```
> start()
hello
_____ missed :
guess : j
_____ missed : j
guess : l
__ll_ missed : j
guess : o
__llo missed : j
guess : a
__llo missed : j a
guess : o
__llo missed : j a
guess : u
__llo missed : j a u
guess : q
__llo missed : j a u q
guess : e
_ello missed : j a u q
guess : x
_ello missed : j a u q x
guess : k
_ello missed : j a u q x k
```

OUTPUT 2: ------status: program wins

> start()

go

__ missed :

guess : t

__ missed : t

guess : a

__ missed : t a

guess : y

__ missed : t a y

guess : o

_o missed : t a y

guess : u

_o missed : t a y u

guess : z

_o missed : t a y u z

guess : g

go missed : t a y u z

OUTPUT 3: ------status: program wins

>start()

prakhar

_____ missed :

guess : a

__a__a_ missed :

guess : a

__a__a_ missed :

guess : a

__a__a_ missed :

guess : r

_ra__ar missed :

guess : p

pra__ar missed :

guess : r

pra__ar missed :

guess : o

pra__ar missed : o

guess : a

pra__ar missed : o

guess : a

pra__ar missed : o

guess : l

pra__ar missed : o l

guess : j

pra__ar missed : o l j

guess : w

pra__ar missed : o l j w

guess : k

prak_ar missed : o l j w

guess : a

prak_ar missed : o l j w

guess : k

prak_ar missed : o l j w

guess : a

prak_ar missed : o l j w

guess : a

prak_ar missed : o l j w

guess : f

prak_ar missed : o l j w f

guess : p

prak_ar missed : o l j w f

guess : r

prak_ar missed : o l j w f

guess : h

prakhar missed : o l j w f

****

Task 2 - Accuracy

In this I have printed the accuracy for each word,for every word my program guesses the %accuracy by the formula --->

(correct_guess/lenght_of_the_word) * 100 = x%

eg:-

hello //--->word_string

correct guess=3

total lenght=5

acccuracy = (3/5)*100= 60%

Program :-

```
getwd()
words<-read.delim("words.txt",header=F)
consonants <- c("b","c","d","f","g","h","j","k","l","m","n","p","q",
               "r","s","t","v","w","x","y","z")
vowels <- c("a","e","i","o","u")
catn <- function(s) #this function is used for printing a variable and for next line
{
  cat(s,"\n",sep="")
}
inputword<-""
x=0
len=0
```

```
start<-function(extracted_word) #main function ----this starts the game ---takes an input first, give your input in the console
{
 inputword<-extracted_word# reading the string from console (user input)
 inputword<-tolower(inputword)
 x <- nchar(inputword)
 guessed_string<-"" #guessed string--->will be storing the correctly guessed characters
 missed<-""  #missed---> stores the wrong guessed characters
 len <- nchar(inputword)
 #printing the empty base before the program makes a guess.
 missed<-paste(missed,"missed :")
 attempt=0
 cal=0
 count=0
 # from here program starts guessing the letters, after 6 wrong attempts it will stop.
 while(attempt!=6)
 {
    val<-sample(vowels,size=1)
  val1<-sample(consonants,size=1)
  if(str_detect(inputword,val))  #checks if the letter exists in the given string
  {
    if(str_detect(guessed_string,val)==F){
    count=count+1
      guessed_string<-paste(guessed_string,val)
    }
    if(nchar(guessed_string)==nchar(inputword)) #if matched print the output and program terminates here.
    {
      break;
    }
    val=""
  }
```

```
   else if(str_detect(inputword,val1))
   {
     if(str_detect(guessed_string,val1)==F){
     count=count+1
     guessed_string<-paste(guessed_string,val1)
     }
   if(nchar(guessed_string)==nchar(inputword)) #if matched print the output and program
terminates here.
   {
   break;
   }
     val1=""
   }
   else  #letters which do not match are stored in missed.
   {
     re<-str_locate_all(pattern=val,missed)
       re1<-str_locate_all(pattern=val1,missed) #---------->line mentioned in the note at starting
if(length(re[[1]])==0) #condition makes sure if the letter is not repeated. If yes then attempt is not
counted.
     {
        missed<-paste(missed,val)
        attempt=attempt+1
     }
     else if(length(re1[[1]])==0)  #---------->lines mentioned in the note at starting
     {
       missed<-paste(missed,val1)
       attempt=attempt+1
     }
   }
}
```

```
  cal=(count/len)*100
  cal=round(cal,0)
  cat(c(inputword,"--->",cal,"%","\n"))
}
j=1
while(j<=nrow(words))
{
  start(words[j,1])
  j=j+1
}
```

****

Output:-

```
> j=1
> while(j<=nrow(words))
+ {
+   start(words[j,1])
+   j=j+1
+ }
aaronic ---> 71 %
abbatie ---> 43 %
abbott ---> 17 %
abbreviators ---> 42 %
abisia ---> 33 %
aboon ---> 40 %
aborting ---> 50 %
abortive ---> 50 %
abraded ---> 57 %
absarokite ---> 50 %
absquatulate ---> 50 %
abstr ---> 20 %
abusage ---> 57 %
```

abuta ---> 40 %

abuttals ---> 50 %

acanthodei ---> 50 %

acaroid ---> 43 %

acates ---> 50 %

accademia ---> 56 %

accent ---> 50 %

accentor ---> 50 %

acceptancies ---> 50 %

accesses ---> 38 %

accumulative ---> 33 %

accumulatively ---> 29 %

ace ---> 67 %

acephalocyst ---> 50 %

acerli ---> 50 %

acetoxyls ---> 56 %

acetylsalol ---> 45 %

acidulate ---> 67 %

ackerman ---> 50 %

acor ---> 50 %

acquent ---> 57 %

acquire ---> 86 %

acrobatic ---> 56 %

acromyotonus ---> 33 %

acrostics ---> 44 %

actinocarpous ---> 38 %

actinozoa ---> 67 %

actory ---> 33 %

acylamino ---> 56 %

adachi ---> 50 %

add ---> 0 %

addedly ---> 29 %

addlings ---> 50 %

adelantados ---> 36 %

adelges ---> 43 %

adenocele ---> 33 %

adenological ---> 50 %

adeptness ---> 56 %

adherence ---> 33 %

adiaphoretic ---> 42 %

adits ---> 80 %

adjuring ---> 50 %

adlare ---> 50 %

admirers ---> 25 %

admissibility ---> 38 %

adonias ---> 43 %

adoptively ---> 50 %

adpress ---> 43 %

adrenalize ---> 50 %

adron ---> 60 %

adscriptitius ---> 54 %

adterminal ---> 30 %

advertising ---> 36 %

advisers ---> 50 %

aeroview ---> 50 %

aeruginous ---> 50 %

aesir ---> 60 %

afb ---> 67 %

aflicker ---> 50 %

afret ---> 60 %

aftosa ---> 33 %

agana ---> 20 %

agars ---> 20 %

agavose ---> 57 %

aged ---> 50 %

agenesic ---> 50 %

agenting ---> 38 %

aggrieved ---> 56 %

aggro ---> 60 %

aghastness ---> 40 %

aglucone ---> 50 %

agrah ---> 20 %

agronomical ---> 55 %

aguste ---> 50 %

ailina ---> 33 %

ainsley ---> 71 %

airbrushing ---> 27 %

airdate ---> 71 %

airedale ---> 38 %

alamiqui ---> 38 %

alamoth ---> 29 %

alantolactone ---> 31 %

alay ---> 0 %

alberta ---> 43 %

albuminose ---> 50 %

alcyone ---> 14 %

aldide ---> 50 %

alek ---> 50 %

alephs ---> 50 %

algidness ---> 67 %

alias ---> 40 %

alkylated ---> 56 %

allagite ---> 50 %

alle ---> 50 %

allegheny ---> 44 %

alleyite ---> 50 %

alliaceous ---> 50 %

allocator ---> 33 %

allomorph ---> 0 %

allosterically ---> 50 %

allothogenous ---> 46 %

allouez ---> 57 %

allyou ---> 50 %

almaden ---> 43 %

almoner ---> 86 %

alterocentric ---> 46 %

altimeter ---> 22 %

altincar ---> 25 %

alvah ---> 60 %

amalgams ---> 12 %

ambitionless ---> 50 %

amblycephalidae ---> 33 %

amboceptor ---> 50 %

amboinese ---> 78 %

----------my program does not display output after this.