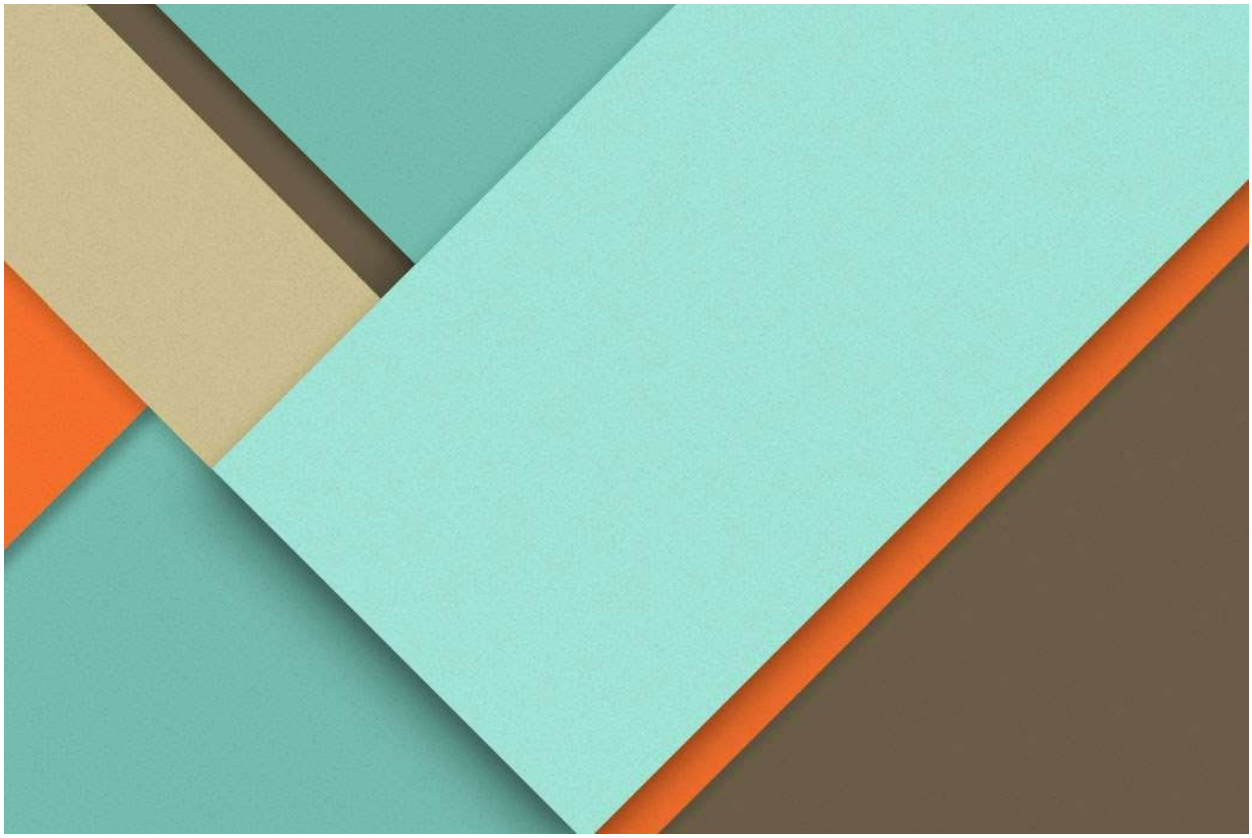


# Typing Speed Test

07.06.2021

ARMAAN SHARMA , AMIT KUMAR
190040020 , 190040013
Team NO # 75
Team PythonDUO

## Introduction

Typing accurately and at an excellent pace is desirable to everyone and highly useful in the contemporary world where it is ever needed in the increasingly digitized world. With this fact in mind , we found it desirable to create a system which can allow individuals to test their typing speeds and accuracy ; and at the same time allow them to improve their typing speed and accuracy as well. The “Typing Speed Test” , as the name suggests is a python project which intends to do the aforementioned work in order to help improve and test the skills of various individuals who may find it useful in their application.

The python file upon being opened will display a sentence and the user is expected to type it out on their keyboard, the input taken will then be computed to give the results in the form of their accuracy & “words per minute” , as mentioned in a comment earlier on our abstract by the TAs, the project will also allow people to type out random words & measure their WPM (words per minute) rate.

## Implementation

1. The user is expected to open the project and type out the given sentence in order to allow it to compute the results and display them on the screen.
2. The python script begins with defining a game class which would then be utilized to implement all functions associated inside it.
3. Firstly we will define the __init__ function, herein we will define and initialize all the important variables which will be used in writing the program. We will also initialize the pygame window in which our program will be run.
4. Then we will use a text_drawing function, the idea of this function is to create a black box and display the text which the user inputs & display it inside the created black box. This function will take itself , the message input by the user, the font and colour of the text, the pygame screen at which it will be drawn & the defining height on the pygame screen as it’s arguments
5. We have created a fillers.txt file in our directory, this txt file stores a series of sentences which can be used to test the typing speed of our user. The sentences in this file can be extracted, sorted & displayed on the game screen.

6. The fillers.txt file will be used in the get_sentences() function now, this function will extract the sentences input inside the txt file. This will then display the sentence taken at random from the many sentences present in the txt file.
7. Now is the results_display() function, which will use itself and the pygame screen as the arguments. It will initialize a counter which would take an increase for each matching input position with the displayed sentence character position. This will become the accuracy of the user and it will then be utilized to compute the WPM rate of the player, the “words per minute” rate can be computed through a simple formula of “correct inputs *60/ 5 * total time”.
8. The results_display() function will also be used to compute the time taken by the user in writing the sentence , herein some variable defined in the __init__ function would be programmed to turn their opposite value ( ie true becomes false or vice versa) at the end of the writing, when we will end the timer. The timer would begin when the user inputs the first letter from his keyboard. This difference would be utilized to compute the total time taken which will be used to measure the accuracy as described previously.
9. Finally after the computation of all the important results, the results will be displayed on the screen, which too is accommodated in the same results_display() function’s snippet.
10. The main part of the game program is the run() function, this function will help run the program by taking input, displaying it on the screen, allowing the user to change their input by using the backspace key & allowing the user to input special character or to reset the program with the reset key which will run through the reset_game() function & to allow the user to enter their entire input and allow them to see their results.
11. The run() function initializes the clock when a certain variable as described in the __init__() function is implemented in order to help compute time as described earlier. This function takes input and displays it on the screen with the help of the previous functions utilized. It is responsible for creating the exit sequence in case the user wants to quit the game & also responsible for creating the reset box which will utilize the reset_game() function in case the user wants to play the game again or wants to reset it.
12. Whether a person wants to reset or exit or play the game, how would the program decide that ? The answer to this question is solved by the run() function as well, the function takes the response of the mouse click button and utilizes it in order to figure out how the user wants to proceed further. The mouse click position on the pygame box can be stored and compared against the position of other functions box space as an acceptable argument.

13. Another problem is how to take input on the game screen, unlike the python input() command, we can't give it to the notebook or command prompt window directly. In order to solve this problem , the run() function gives one more utility. Every time a key is pressed down , it will accept it as input and display it on the screen. Additionally , in order to know when the user has started typing the first key pressed command after the mouse click touching the entry bar can be accepted as an argument to run the timer. This will thus, help us compute time as well for other functionalities.
14. Another problem is what if a user has mistyped something and wants to correct it ? For such a scenario, we have introduced the backspace key recognition system in our run() function. This would recognize whenever a backspace key is pressed and at the same time remove one character from the input string and then update the screen to show one lesser character on the entry bar.
15. The final problem is the use of special characters, something which is not accepted by compilers usually, in order to solve that , we will introduce a unicode function which is available in standard pygame libraries. This would help us accept special characters like &*@$%@#% as inputs and allow us to display them on the entry bar whenever inputted by the user
16. Finally is the reset_game() function, it will initialize all the variables as they were in the beginning , declared in the __init__() function. This will allow the user to play the game continuously without opening and closing the file over and over again for a smooth experience.
17. I have given a rough implementation idea in the above paragraphs, furthermore I am attaching herein the code with line by line comments for further enquiry of any sorts of functions, objects etc etc used in the given script.

```
import sys
import time
import random
import pygame
from pygame.locals import *


# In[2]:


class Game:
```

```
    def __init__(self):
        self.w = 750      #width of game box is 750
        self.h = 500      #height of game boc is 500

        self.inputs = ''   #taking input
        self.words = ''

        self.start_time = 0   #measuring time at beginning
        self.total_time = 0   #measuring end time

        self.end = False     # to show results at end and calculate time
(total_time-start_time if end=true)
        self.reset=True      # will be used for resetting
        self.active = False  #used to reset the timer and click

        self.accuracy = '0%'                          #intializing accuracy
(count/len(self.words)*100)
        self.results = 'Time:0 Accuracy:0 % Wpm:0 ' #will display results
        self.wpm = 0                                  #initializing wpm =
correct charx100/total chars


        self.chead = (255,213,102)  # the tuples assigned are rgb values of
the respective colours
        self.ctext = (240,240,240)
        self.cresult = (255,70,70)


        pygame.init()               #initialize all imported pygame modules

        self.open_img = pygame.image.load('opening.png')
#opening image & transform size
        self.open_img = pygame.transform.scale(self.open_img,
(self.w,self.h))

        self.bg = pygame.image.load('bkg.jpg')               #game
background image
        self.bg = pygame.transform.scale(self.bg, (500,750))

        self.screen = pygame.display.set_mode((self.w,self.h))  #This
function will create a display Surface
```

```
        pygame.display.set_caption('Typing Speed Time')   #creates a
caption on the display window




    # In[3]:


    #in order to draw further on the scren we will use a text_drawing
method
    #the argument takes the input screen, message, font size, y cordinate,
colour of font

    def text_drawing(self, screen, msg, yaxis, font_s, color):

        font = pygame.font.Font(None, font_s)  #create a new  Font object

                                                    # create a text
surface object,
                                                    # on which text is
drawn on it.
        text = font.render(msg, 1,color)


        text_rect = text.get_rect(center=(self.w/2, yaxis)) # put text
rectangle at desired centre

        screen.blit(text, text_rect)                    # draw text at
text_rect place


        pygame.display.update()                        #update and show the
following changes on the screen


    # In[4]:


    #The get_sentence() method will open up the file and return a random
sentence from the list
```

```
    def get_sentence(self):


        #split the whole txt file sentence list through the \n character to
get sentence


        froms= open('fillers.txt').read()
        fillers = froms.split('\n')
        sentence = random.choice(fillers)
        return sentence




    # In[5]:


    def results_display(self,screen):

        if(not self.end):      #self.end initialized as false, ie not end =
true, till that time calculate

            self.total_time = time.time() - self.start_time  #calculating
time

            #Calculating accuracy

            counter = 0                                 #initialize a counter
            for i,tu in enumerate(self.words):       # we will use words to
get the sentence, then enumerate it
                                                        # from 0-n , then check
whether letter i of input matches
                                                        #the letter c , ie of
c=i position of enumerated sentence.


                #since the user may type an incorrect character at desired
```

```
place, we will use the try and except
                #command in order to make sure that we bypass all errors,
otherwise 'if' would be fine

                try:
                    if self.inputs[i] == tu:
                        counter += 1              #increase counter for
each instance when i and c matches
                                                  #if all are correct then
counter=len(self.words)
                except:
                    pass
            self.accuracy = counter/len(self.words)*100    #accuracy
formula

            #Calculating wpm = words per minute

            self.wpm = len(self.inputs)*60/(5*self.total_time)  #this is
the given formula
            self.end = True             #end was initialized as false
earlier,by setting it true, means that it has taken time
            print(self.total_time)

            #making the standard results format ie:: 'Time:0 Accuracy:0 %
Wpm:0 '

            self.results = 'Time:'+str(round(self.total_time)) +" secs
Accuracy:"+ str(round(self.accuracy)) + "%" + ' Wpm: ' +
str(round(self.wpm))


             # drawing the icon image

            self.newimg = pygame.image.load('icons.png')
            self.newimg = pygame.transform.scale(self.newimg, (150,150))


            screen.blit(self.newimg, (self.w/2-75,self.h-140))
            self.text_drawing(screen,"Reset", self.h - 70, 26,
(100,100,100))

            #update and display the results
```

```
            print(self.results)
            pygame.display.update()




    # In[6]:


    # we will create a method here called reset_game in order to reset all
the variables used here
    # we would also need to run a loop in order to capture events committed
by mouse and keyboards
    def run(self):

        self.reset_game()    # we will call this method right here in order
to reset all variables at the beginning
                             # the reset_game() method will be defined in
the next block

        self.running=True    #initializing variable for the 'while' loop

        while(self.running):

            clock = pygame.time.Clock()  #initialize the clock

            self.screen.fill((0,0,0), (50,250,650,50))
            pygame.draw.rect(self.screen,self.chead, (50,250,650,50), 2)
#drawing the header


            # updating user input based text
            self.text_drawing(self.screen, self.inputs, 274,
26,(250,250,250))
            pygame.display.update()


            for event in pygame.event.get():

                if event.type == QUIT:     #mode of exiting the sequence
                    self.running = False
                    sys.exit()
```

```
                elif event.type == pygame.MOUSEBUTTONUP:   #receives the
position of the box of mouseclick
                    x,y = pygame.mouse.get_pos()

                    # position of input box

                    if(x>=50 and x<=650 and y>=250 and y<=300):  #check
whether mouseclick pos lies in input box position
                        self.active = True
                        self.inputs = ''
                        self.start_time = time.time()

                     # position of reset box

                    if(x>=310 and x<=510 and y>=390 and self.end): #check
whether mouseclick pos lies in reset box position
                        self.reset_game()
                        x,y = pygame.mouse.get_pos()


                elif event.type == pygame.KEYDOWN:  #checks whether a key
has been pressed

                    if self.active and not self.end: #self.active= true
means that typing has begun and
                                                        #self.end= false means
that typing has not ended.

                        if event.key == pygame.K_RETURN:  #if a key is
pressed then print the result on screen

                            print(self.inputs)
                            self.results_display(self.screen)
                            print(self.results)
                            self.text_drawing(self.screen,
self.results,350, 28, self.cresult) #call function to display text
                            self.end = True
#self.end = true means timer ends here

                        elif event.key == pygame.K_BACKSPACE:      #if
backspace is pressed then inputs will go one lesser
```

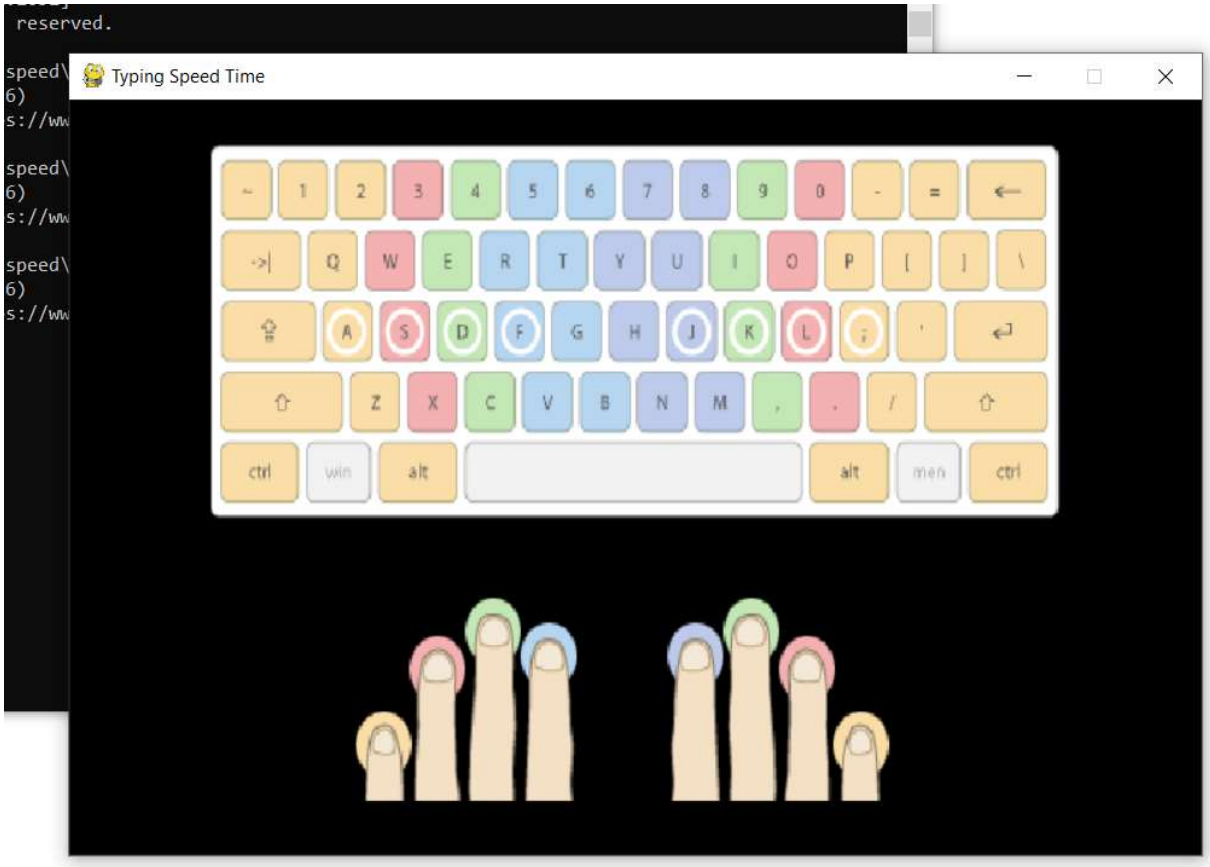
```
                            self.inputs = self.inputs[:-1]

                        else:                               #using try and
pass inorder to bypass any erros and allow smooth run
                            try:
                                self.inputs += event.unicode  #recognize
special characters like ',".;@#$% etc
                            except:
                                pass

            pygame.display.update() #after taking above commands, allows
screen to be available to user

        clock.tick(60)        #allows 60 frames per second


    # In[7]:


    def reset_game(self):

        self.screen.blit(self.open_img, (0,0)) #reopen opening image
        pygame.display.update()                #readjust display
        time.sleep(1)                          # delays further execution
by one second, ie allows opening image for 1 sec

        self.inputs=''                            # initializing inputs the
way it was in __init__(self)
        self.words = ''

        self.start_time = 0                  # intializing time the way it
was in __init__(self)
        self.total_time = 0
        self.wpm = 0


        self.reset=False
        self.end = False
        self.active=False
```

```
        # Getting a random sentence again
        self.words = self.get_sentence()
        if (not self.words): self.reset_game()

        #drawing the heading again


        self.screen.fill((0,0,0))
        self.screen.blit(self.bg,(0,0))


        msg = "Typing Speed Test"

        self.text_drawing(self.screen, msg,80, 80,self.chead)

        # draw the rectangle for input box
        pygame.draw.rect(self.screen,(255,192,25), (50,250,650,50), 2)

        # draw the sentence string
        self.text_drawing(self.screen, self.words,200, 28,self.ctext)
        pygame.display.update()


    # In[8]:


Game().run()
```

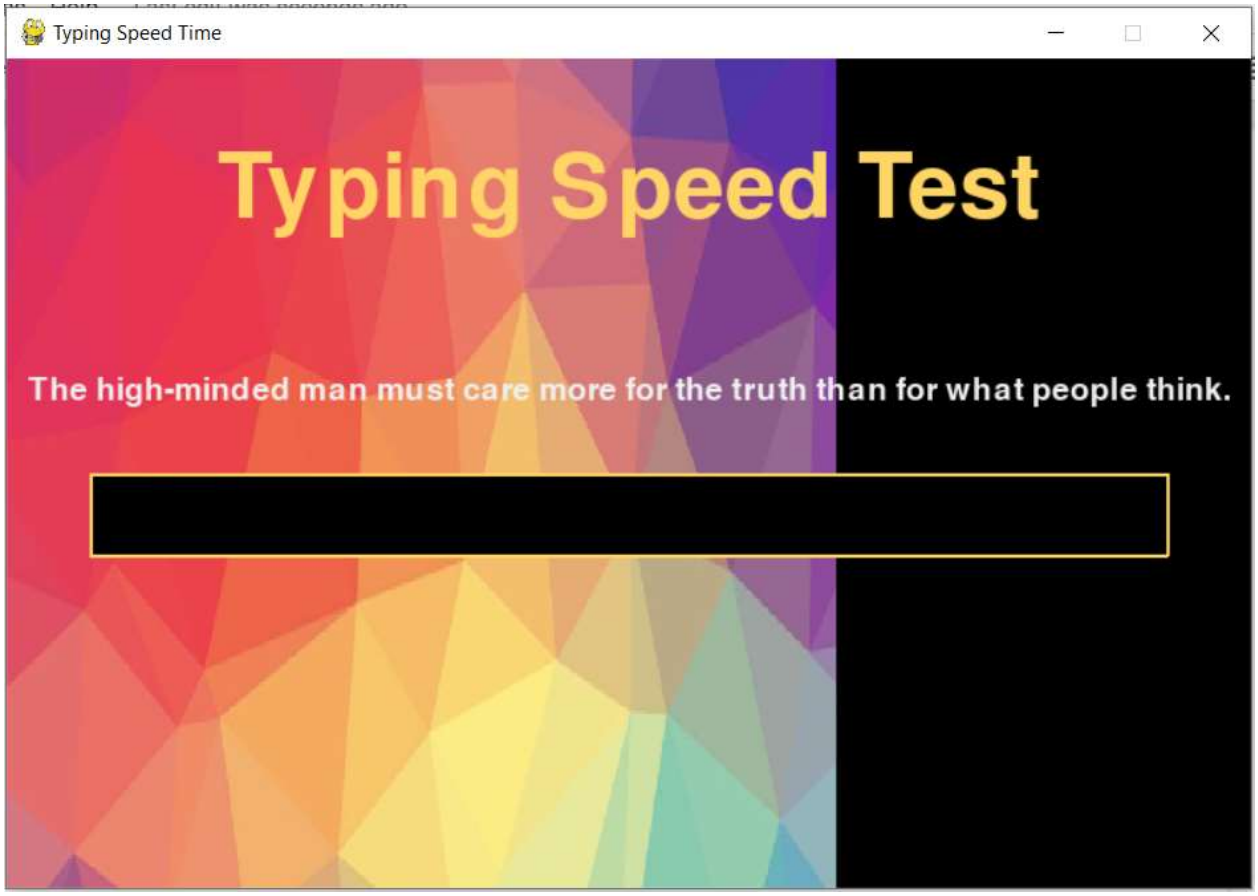
18. Apologies for the bad formatting, we are unable to put the code as it is due to the size of the comments.

## Screenshots

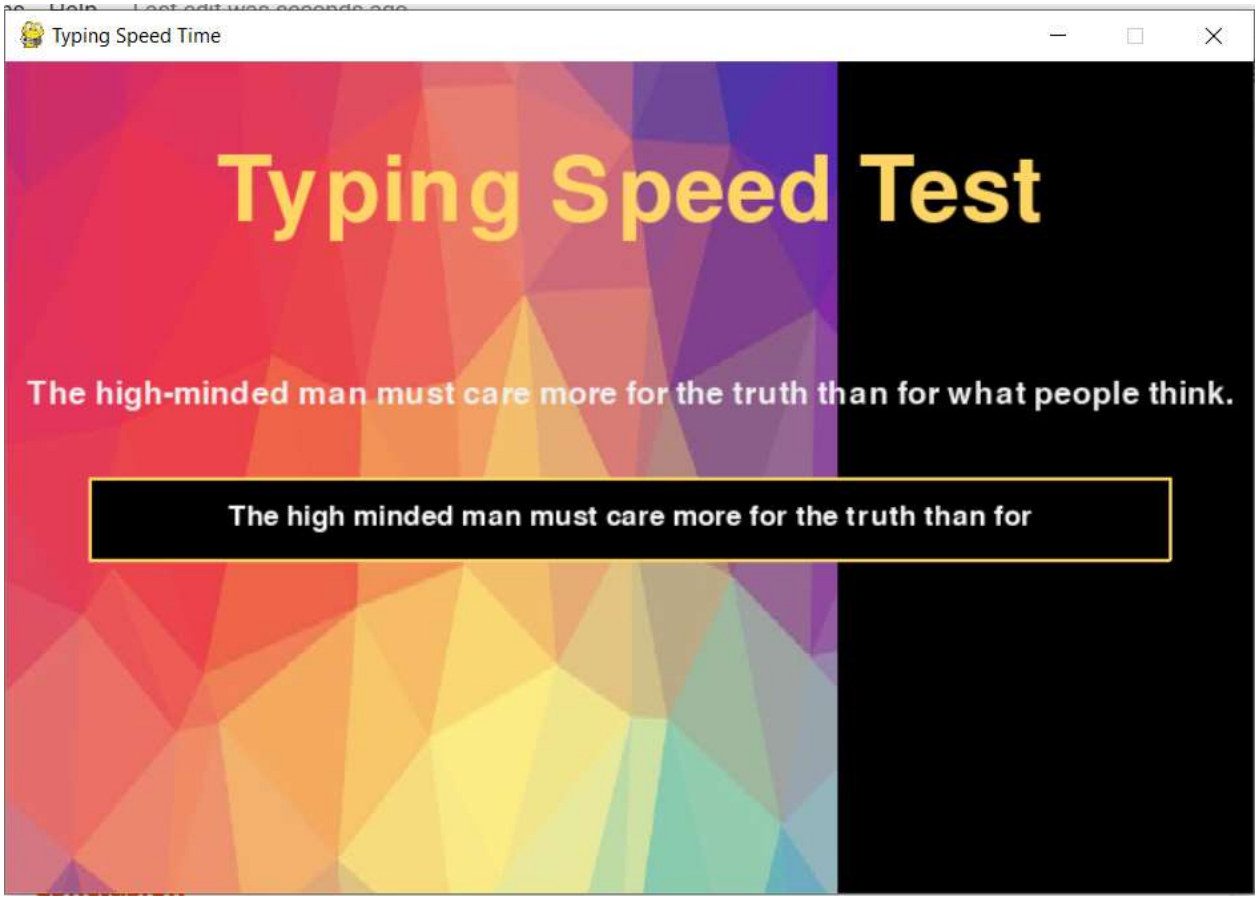
- Opening screen



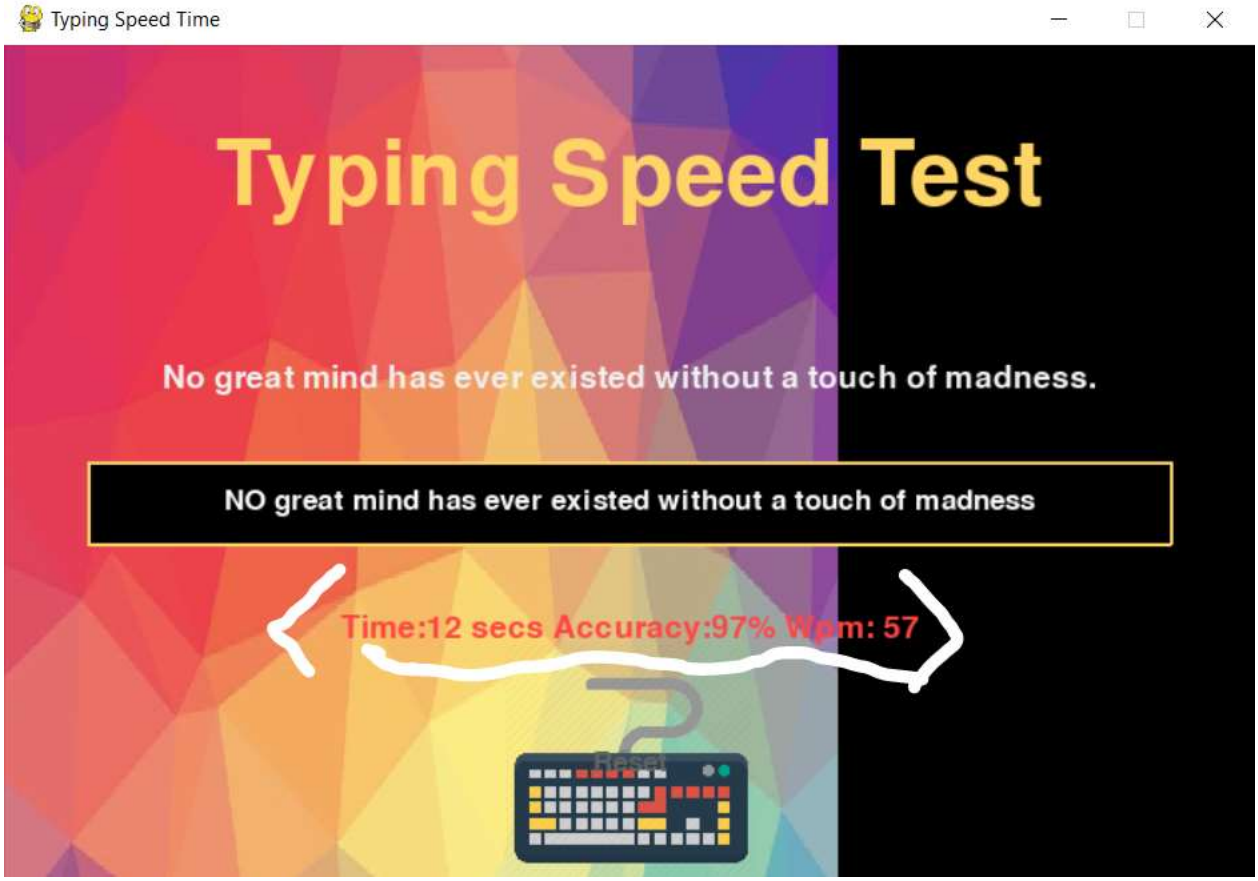
- Running game



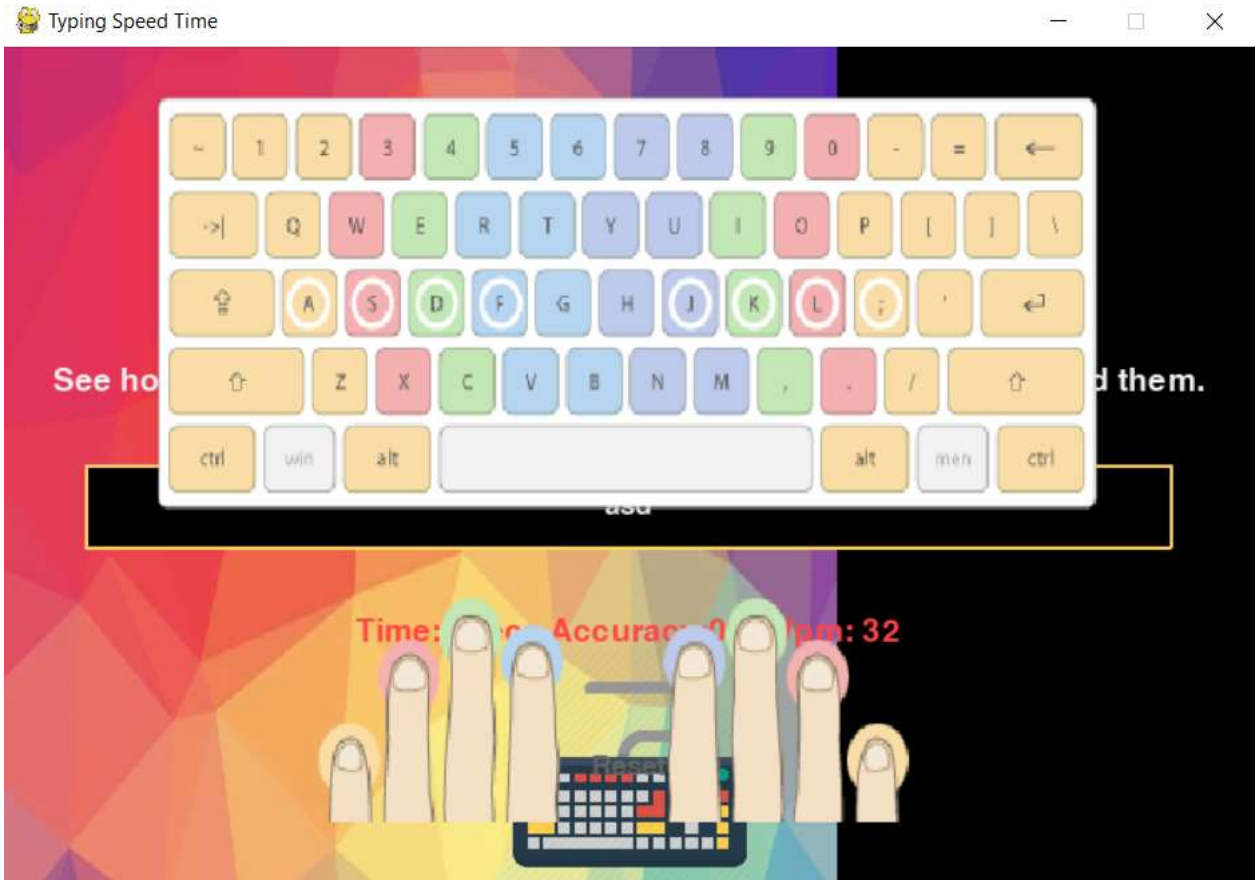
- Typing in progress after the game has begun running.



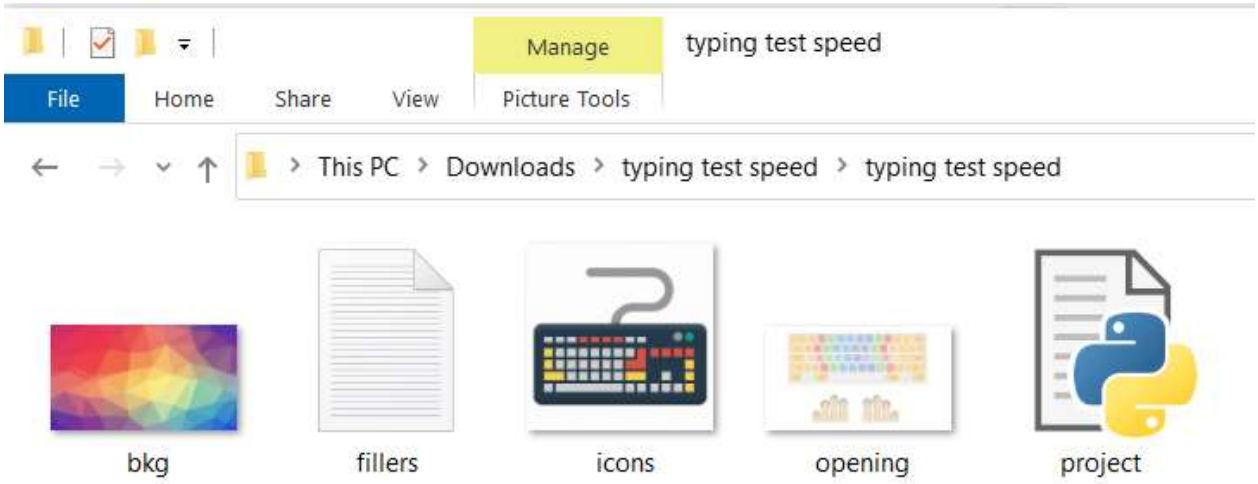
- Results being displayed



- Game reset , transition screen




- The directory file folder



# Conclusion

At the beginning of the course we barely had any idea about computer programming & the summer course had assisted us tremendously. However, regardless of that, this project was quite a bit difficult which required a lot of additional learning which we are happy to have undergone.

The project has worked out correctly, there have been no visible errors or problems of any sorts with it either, with it being capable of computing the WPM rate and the accuracy of the user.  It will hopefully help enhance the typing pace and accuracy of all those who may use it. Furthermore, it can help improve people's skills and help save them a lot of time while working on their computers and laptops or other devices which require a keyboard to take input for typing.