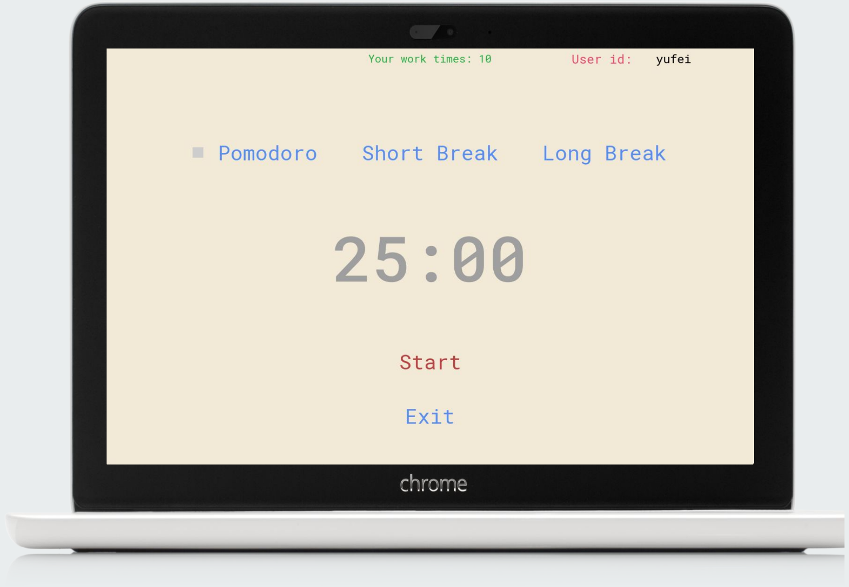
Pomodoro Timer Project

CS5001 Yufei Wu

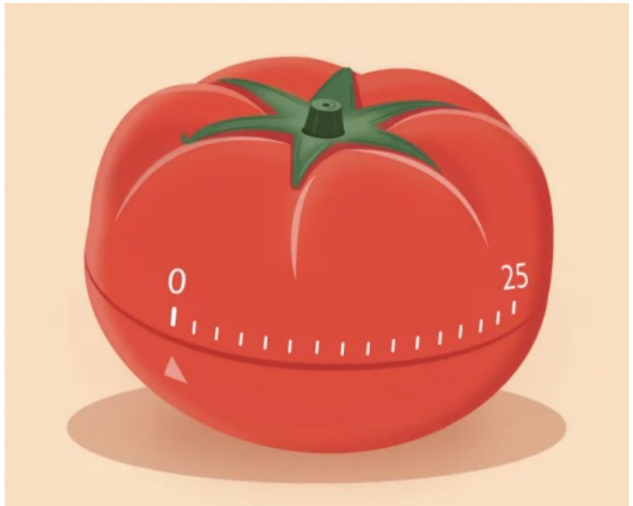


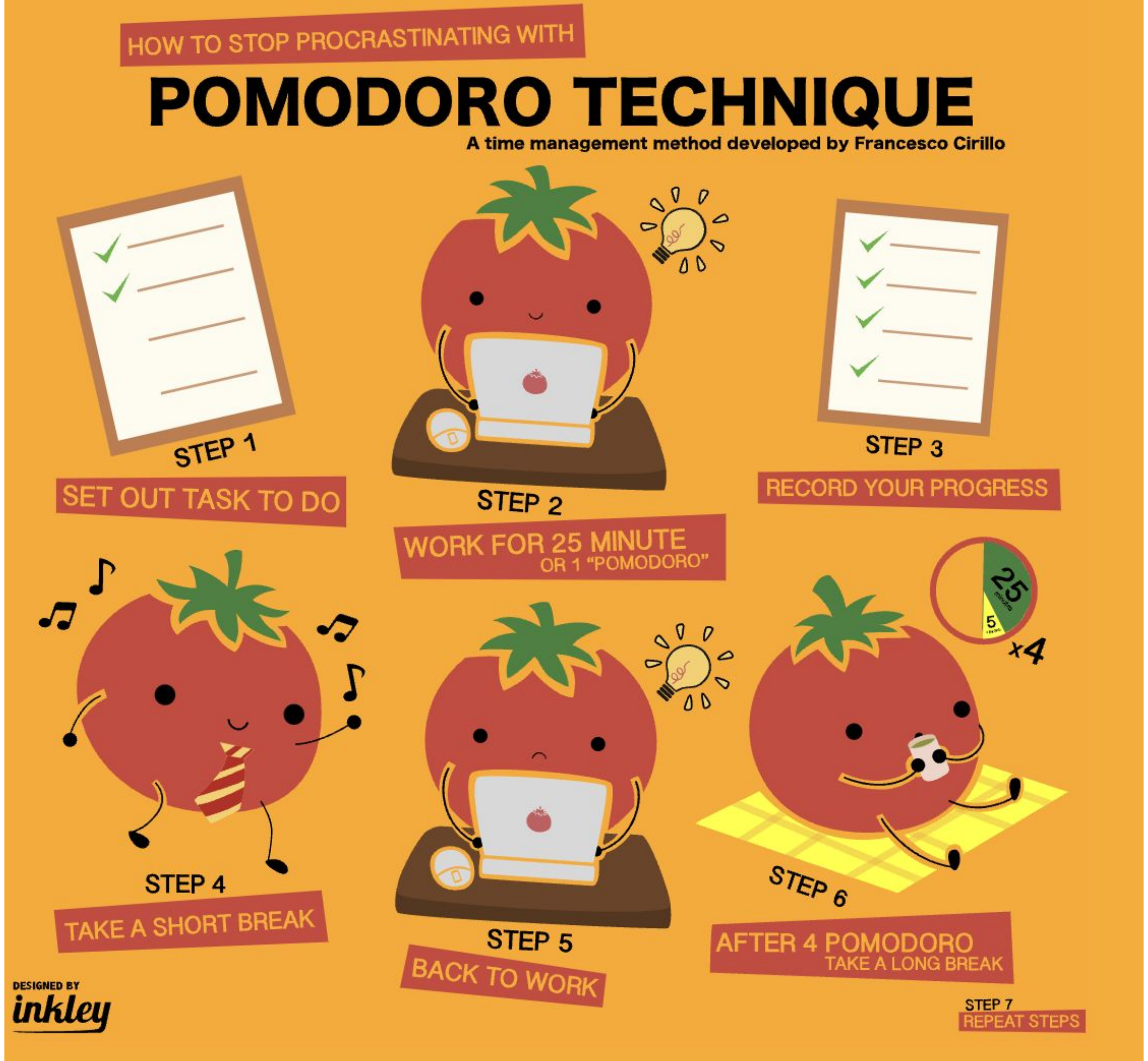
Outline

Project Goals

What is Pomodoro?

- A time management method that is an effective approach to enhancing productivity and managing tasks.
- Pomodoro(in Italian) = Tomato!!




HOW TO STOP PROCRASTINATING WITH
POMODORO TECHNIQUE
A time management method developed by Francesco Cirillo
STEP 1
SET OUT TASK TO DO
STEP 2
WORK FOR 25 MINUTE OR 1 "POMODORO"
STEP 3
RECORD YOUR PROGRESS
STEP 4
TAKE A SHORT BREAK
STEP 5
BACK TO WORK
25
5
x4
STEP 6
AFTER 4 POMODORO TAKE A LONG BREAK
STEP 7
REPEAT STEPS
DESIGNED BY
inkley

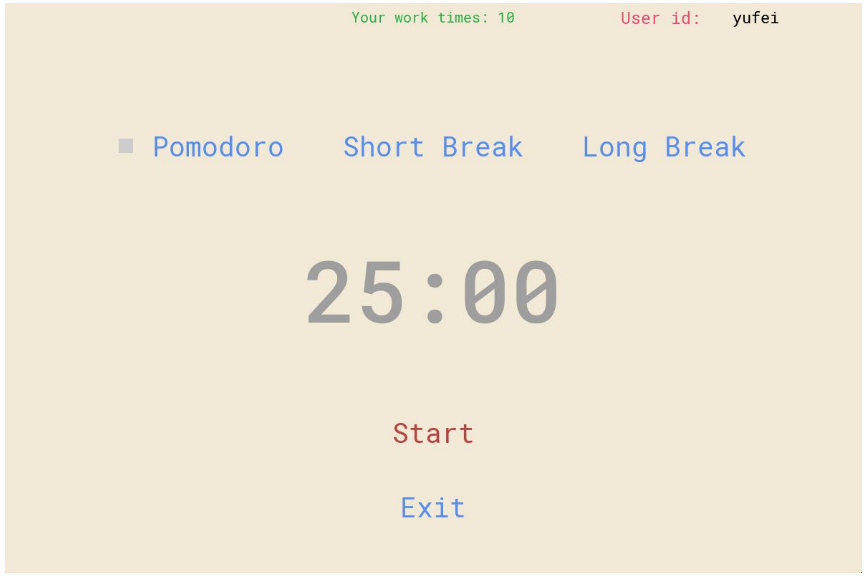
more info

Goals

1. Set three cycles: Pomodoro, short break, and long break
2. Get the timer
3. Reset the timer when click the section buttons
4. Add start/pause functions in each section
5. Give feedback(sound effects) when clicking the button
6. Add an input box for user to enter their user names
7. Use database to track the user_id and their worktimes

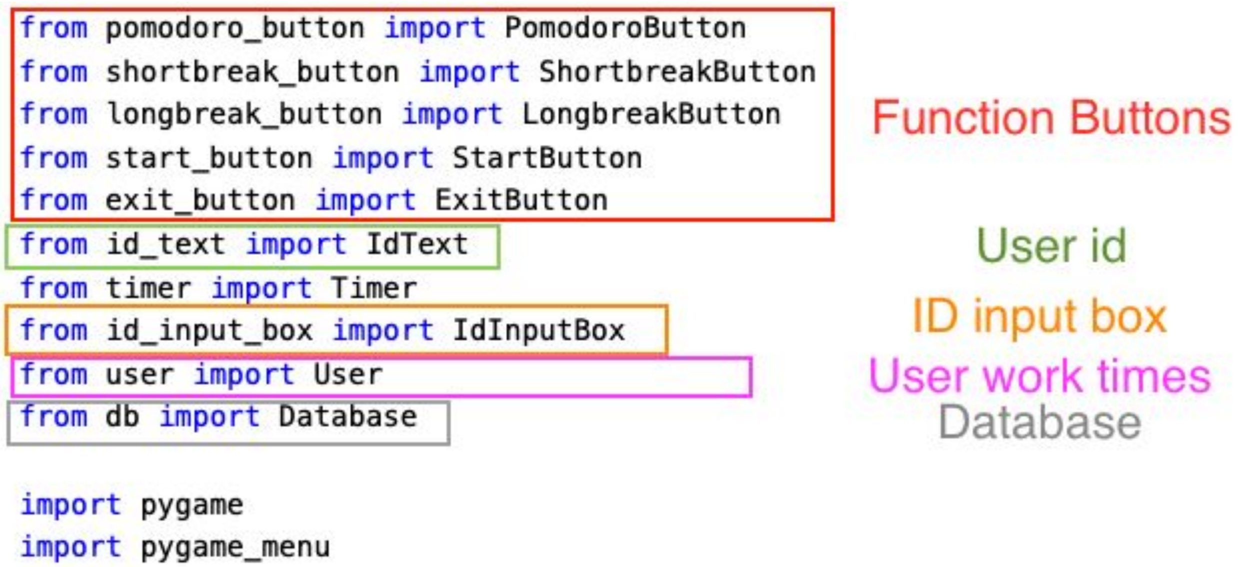
Demo

Show you my code



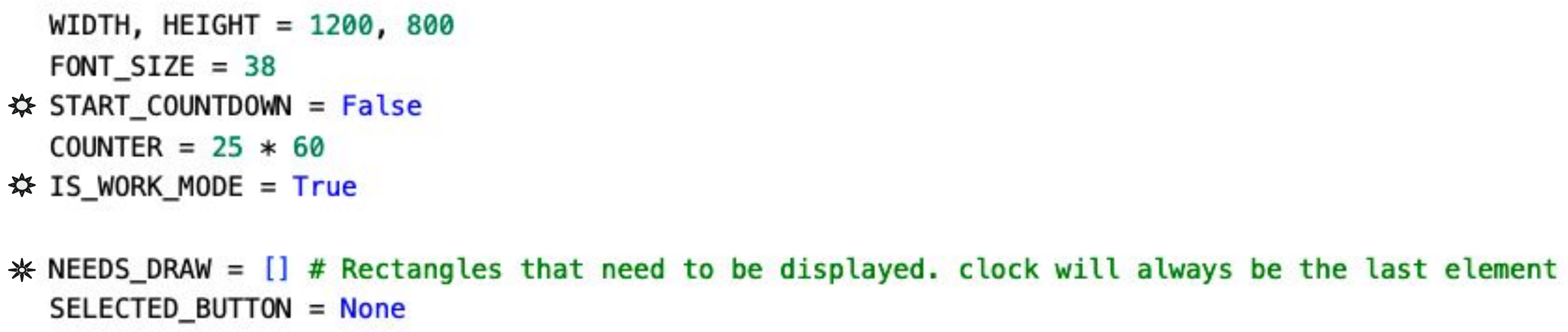
Code Structure

Code Structure: Classes of Objects



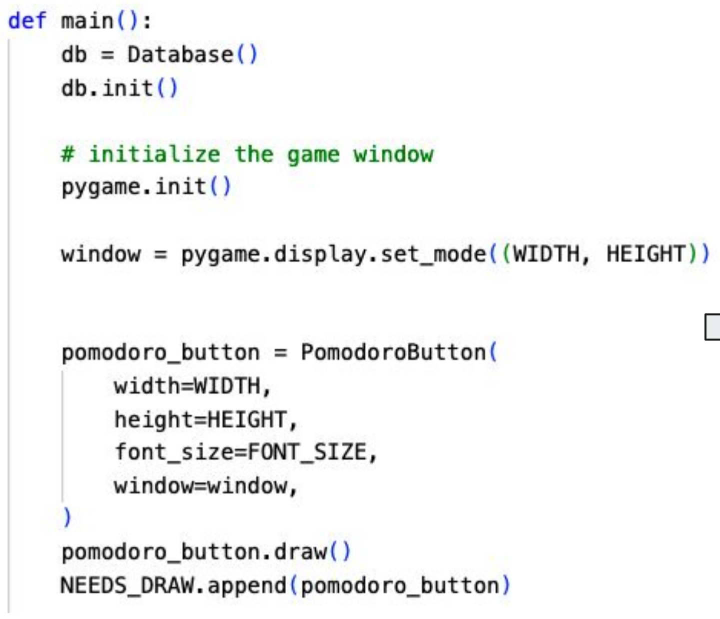
```
from pomodoro_button import PomodoroButton
from shortbreak_button import ShortbreakButton
from longbreak_button import LongbreakButton
from start_button import StartButton
from exit_button import ExitButton
from id_text import IdText
from timer import Timer
from id_input_box import IdInputBox
from user import User
from db import Database

import pygame
import pygame_menu
```

Code Structure: Global Variables



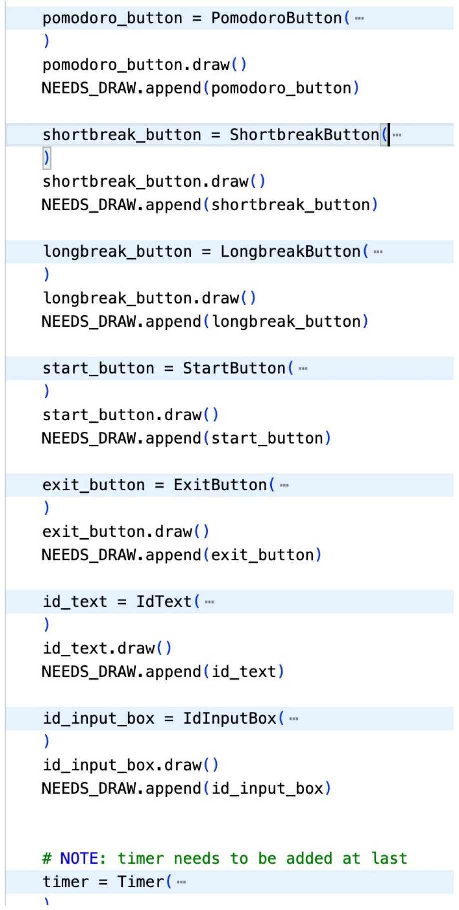
```
WIDTH, HEIGHT = 1200, 800
FONT_SIZE = 38
START_COUNTDOWN = False
COUNTER = 25 * 60
IS_WORK_MODE = True

NEEDS_DRAW = [] # Rectangles that need to be displayed. clock will always be the last element
SELECTED_BUTTON = None
```

Main Loop



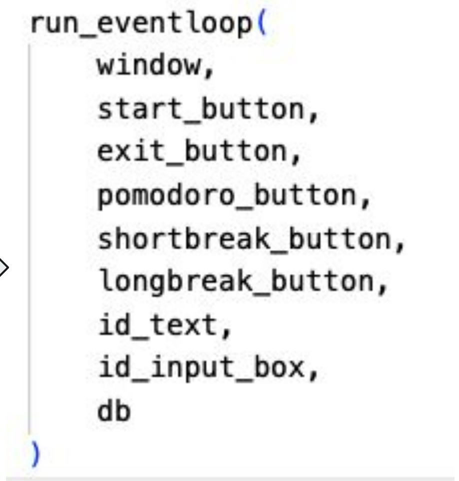
```python
def main():
    db = Database()
    db.init()

    # initialize the game window
    pygame.init()

    window = pygame.display.set_mode((WIDTH, HEIGHT))


    pomodoro_button = PomodoroButton(
        width=WIDTH,
        height=HEIGHT,
        font_size=FONT_SIZE,
        window=window,
    )
    pomodoro_button.draw()
    NEEDS_DRAW.append(pomodoro_button)
```

Create buttons &
Add all objects into the list
NEEDS_DRAW()



```python
pomodoro_button = PomodoroButton(…
)
pomodoro_button.draw()
NEEDS_DRAW.append(pomodoro_button)

shortbreak_button = ShortbreakButton(…
)
shortbreak_button.draw()
NEEDS_DRAW.append(shortbreak_button)

longbreak_button = LongbreakButton(…
)
longbreak_button.draw()
NEEDS_DRAW.append(longbreak_button)

start_button = StartButton(…
)
start_button.draw()
NEEDS_DRAW.append(start_button)

exit_button = ExitButton(…
)
exit_button.draw()
NEEDS_DRAW.append(exit_button)

id_text = IdText(…
)
id_text.draw()
NEEDS_DRAW.append(id_text)

id_input_box = IdInputBox(…
)
id_input_box.draw()
NEEDS_DRAW.append(id_input_box)


# NOTE: timer needs to be added at last
timer = Timer(…
)
```

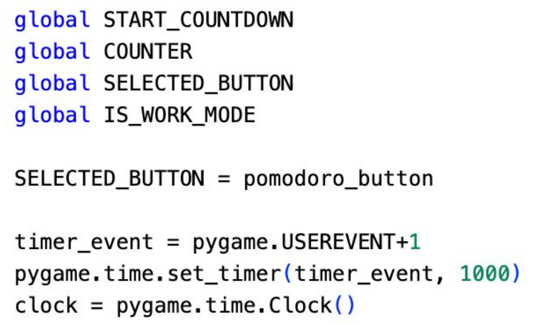
……after creating all buttons



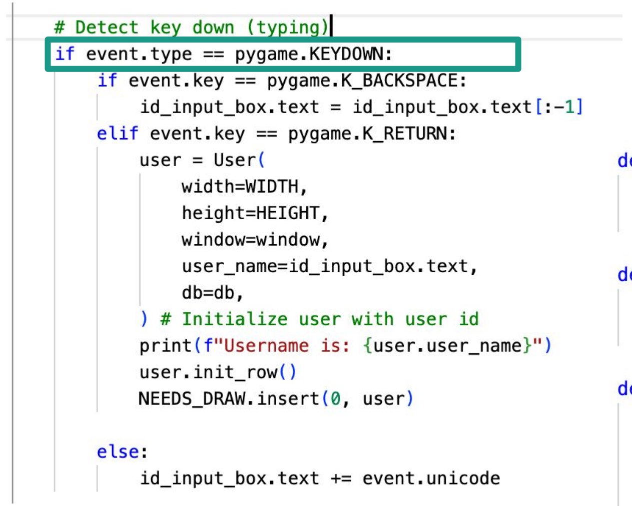
```python
run_eventloop(
    window,
    start_button,
    exit_button,
    pomodoro_button,
    shortbreak_button,
    longbreak_button,
    id_text,
    id_input_box,
    db
)
```

Timer is always the last element since we need to use it to update the remaining time. Otherwise, the new timer rect will overlay the old one.

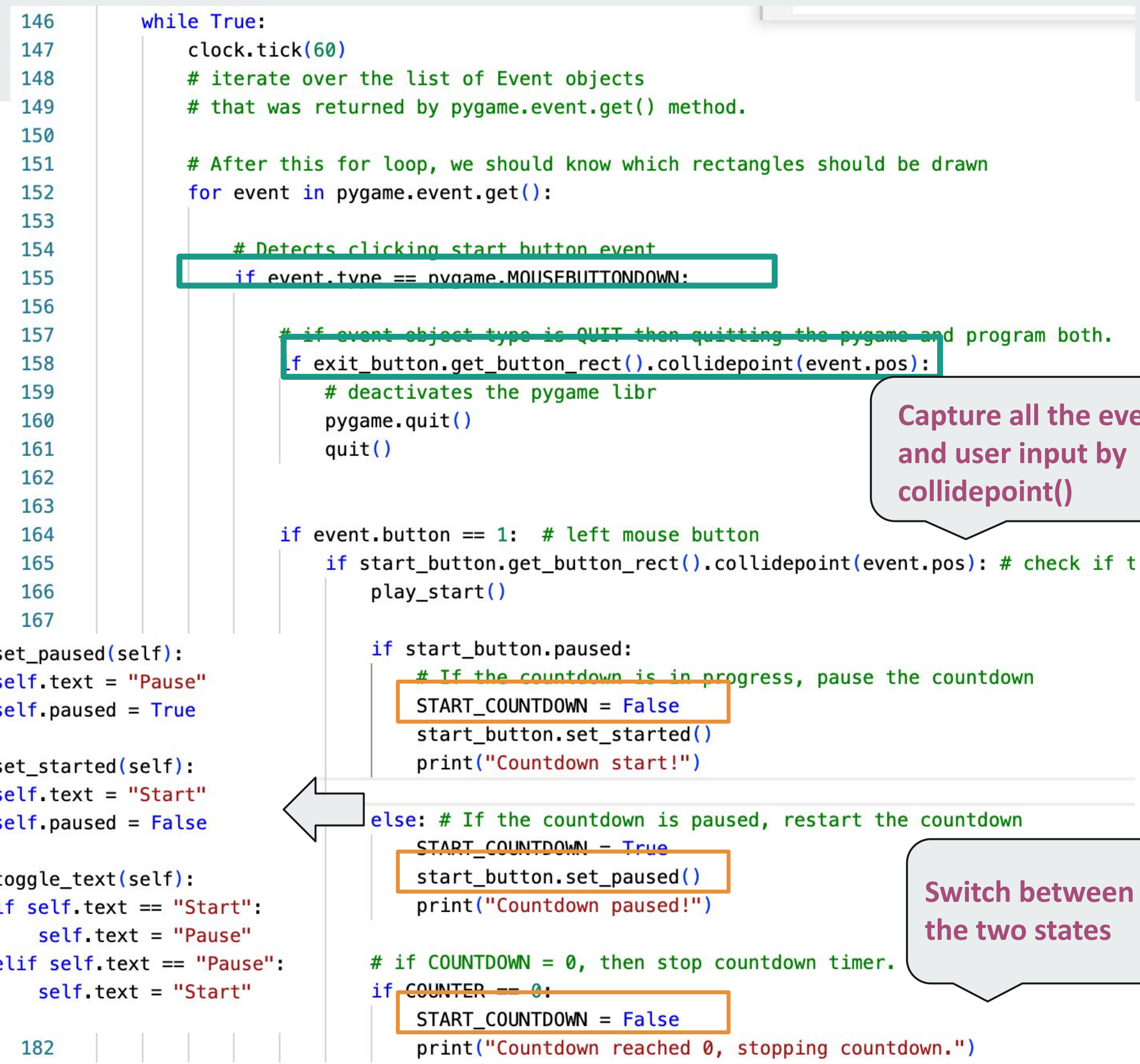
Event Handling Loop



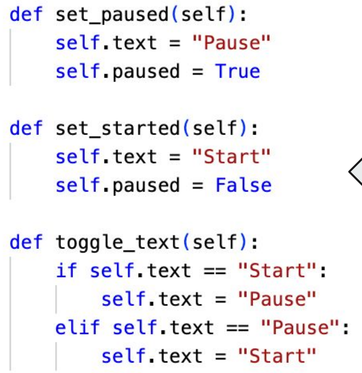
```
global START_COUNTDOWN
global COUNTER
global SELECTED_BUTTON
global IS_WORK_MODE

SELECTED_BUTTON = pomodoro_button

timer_event = pygame.USEREVENT+1
pygame.time.set_timer(timer_event, 1000)
clock = pygame.time.Clock()
```



```
    # Detect key down (typing)
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_BACKSPACE:
            id_input_box.text = id_input_box.text[:-1]
        elif event.key == pygame.K_RETURN:
            user = User(
                width=WIDTH,
                height=HEIGHT,
                window=window,
                user_name=id_input_box.text,
                db=db,
            ) # Initialize user with user id
            print(f"Username is: {user.user_name}")
            user.init_row()
            NEEDS_DRAW.insert(0, user)

        else:
            id_input_box.text += event.unicode
```



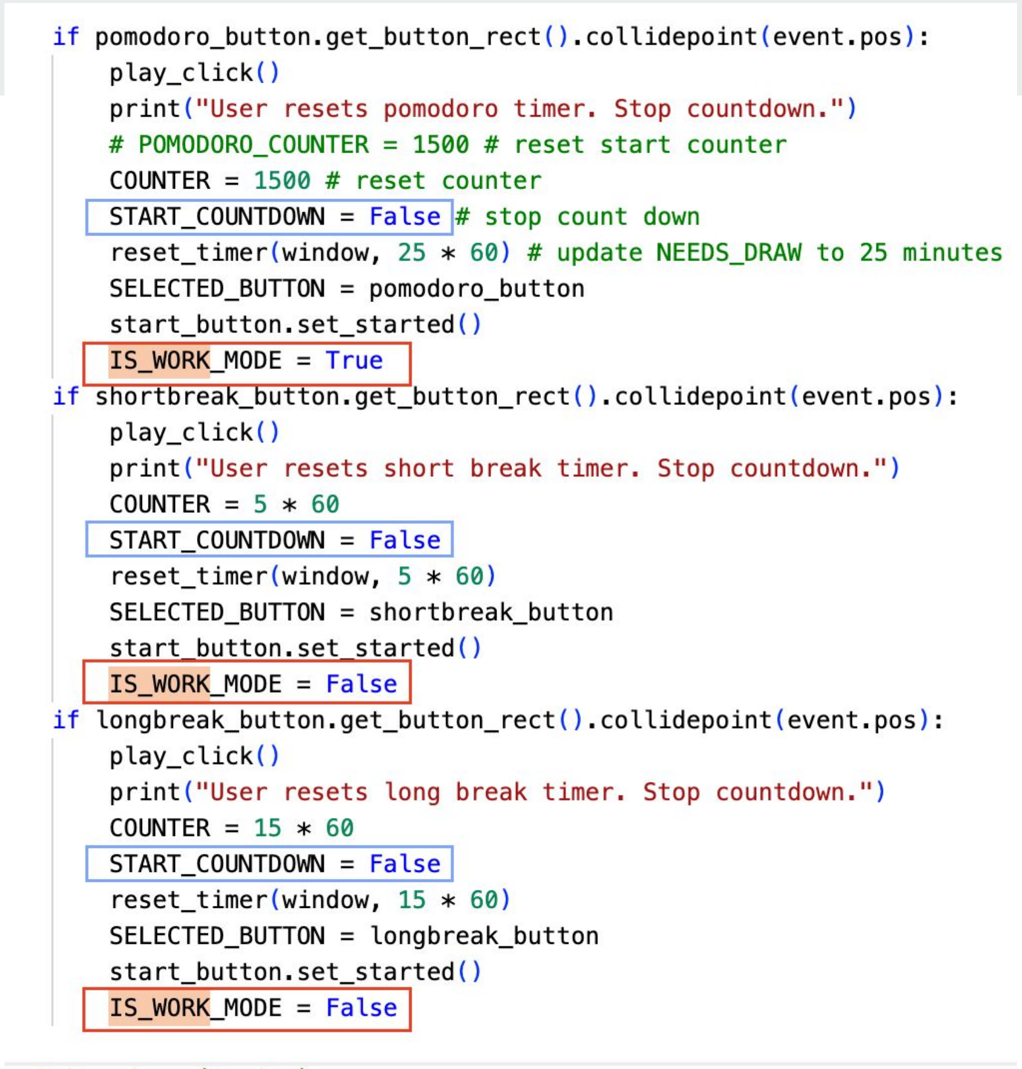
```
    while True:
        clock.tick(60)
        # iterate over the list of Event objects
        # that was returned by pygame.event.get() method.

        # After this for loop, we should know which rectangles should be drawn
        for event in pygame.event.get():

            # Detects clicking start button event
            if event.type == pygame.MOUSEBUTTONDOWN:

                # if event object type is QUIT then quitting the pygame and program both.
                if exit_button.get_button_rect().collidepoint(event.pos):
                    # deactivates the pygame libr
                    pygame.quit()
                    quit()


                if event.button == 1:  # left mouse button
                    if start_button.get_button_rect().collidepoint(event.pos): # check if t
                        play_start()

                        if start_button.paused:
                            # If the countdown is in progress, pause the countdown
                            START_COUNTDOWN = False
                            start_button.set_started()
                            print("Countdown start!")

                        else: # If the countdown is paused, restart the countdown
                            START_COUNTDOWN = True
                            start_button.set_paused()
                            print("Countdown paused!")

                        # if COUNTDOWN = 0, then stop countdown timer.
                        if COUNTER == 0:
                            START_COUNTDOWN = False
                            print("Countdown reached 0, stopping countdown.")
```



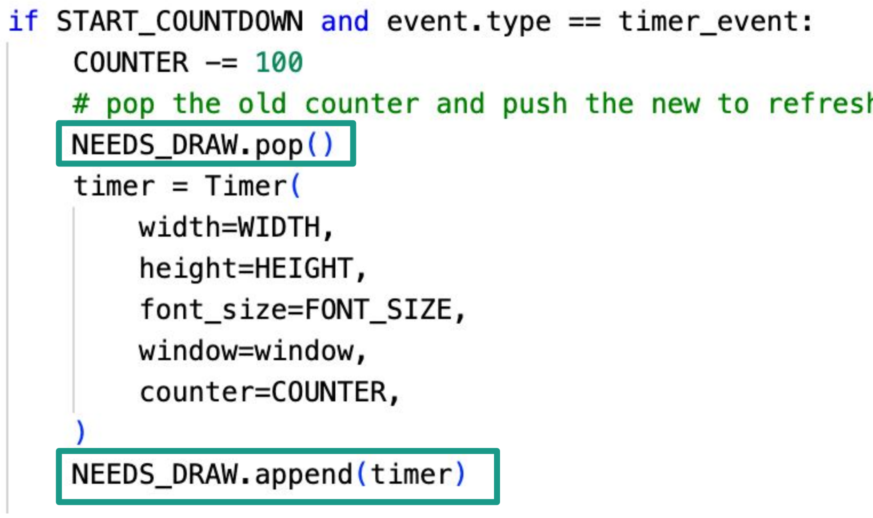
```
def set_paused(self):
    self.text = "Pause"
    self.paused = True

def set_started(self):
    self.text = "Start"
    self.paused = False

def toggle_text(self):
    if self.text == "Start":
        self.text = "Pause"
    elif self.text == "Pause":
        self.text = "Start"
```

Capture all the events and user input by collidepoint()

Switch between the two states

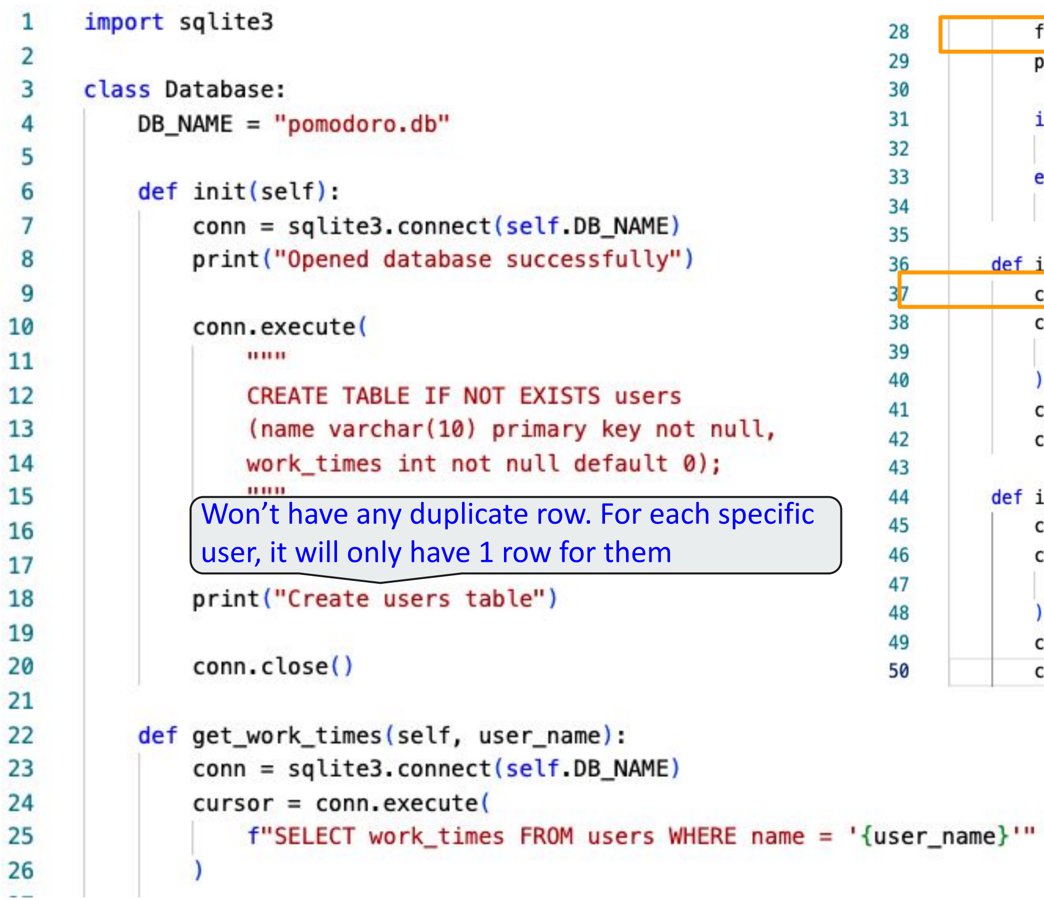


```python
if pomodoro_button.get_button_rect().collidepoint(event.pos):
    play_click()
    print("User resets pomodoro timer. Stop countdown.")
    # POMODORO_COUNTER = 1500 # reset start counter
    COUNTER = 1500 # reset counter
    START_COUNTDOWN = False # stop count down
    reset_timer(window, 25 * 60) # update NEEDS_DRAW to 25 minutes
    SELECTED_BUTTON = pomodoro_button
    start_button.set_started()
    IS_WORK_MODE = True
if shortbreak_button.get_button_rect().collidepoint(event.pos):
    play_click()
    print("User resets short break timer. Stop countdown.")
    COUNTER = 5 * 60
    START_COUNTDOWN = False
    reset_timer(window, 5 * 60)
    SELECTED_BUTTON = shortbreak_button
    start_button.set_started()
    IS_WORK_MODE = False
if longbreak_button.get_button_rect().collidepoint(event.pos):
    play_click()
    print("User resets long break timer. Stop countdown.")
    COUNTER = 15 * 60
    START_COUNTDOWN = False
    reset_timer(window, 15 * 60)
    SELECTED_BUTTON = longbreak_button
    start_button.set_started()
    IS_WORK_MODE = False
```



```python
if START_COUNTDOWN and event.type == timer_event:
    COUNTER -= 100
    # pop the old counter and push the new to refres
    NEEDS_DRAW.pop()
    timer = Timer(
        width=WIDTH,
        height=HEIGHT,
        font_size=FONT_SIZE,
        window=window,
        counter=COUNTER,
    )
    NEEDS_DRAW.append(timer)
```

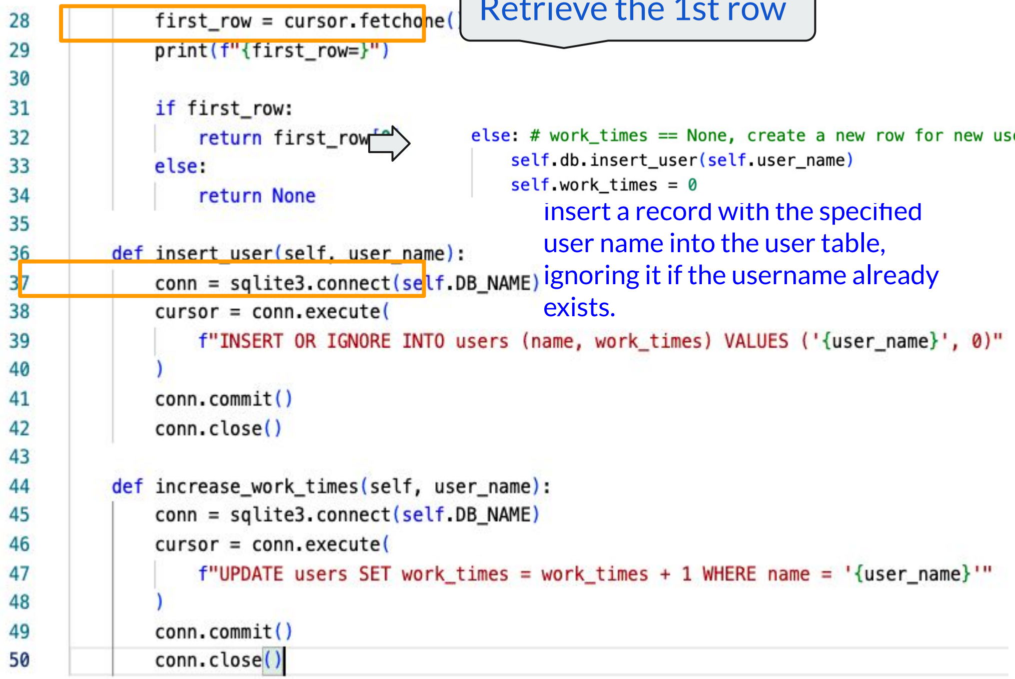
Code Structure: Database



```
import sqlite3

class Database:
    DB_NAME = "pomodoro.db"

    def init(self):
        conn = sqlite3.connect(self.DB_NAME)
        print("Opened database successfully")

        conn.execute(
            """
            CREATE TABLE IF NOT EXISTS users
            (name varchar(10) primary key not null,
            work_times int not null default 0);
            """
```

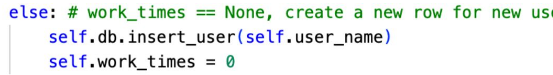
Won't have any duplicate row. For each specific user, it will only have 1 row for them

```
        print("Create users table")

        conn.close()

    def get_work_times(self, user_name):
        conn = sqlite3.connect(self.DB_NAME)
        cursor = conn.execute(
            f"SELECT work_times FROM users WHERE name = '{user_name}'"
        )
```



```
        first_row = cursor.fetchone(
        print(f"{first_row=}")

        if first_row:
            return first_row[
        else:
            return None

    def insert_user(self, user_name):
        conn = sqlite3.connect(self.DB_NAME)
        cursor = conn.execute(
            f"INSERT OR IGNORE INTO users (name, work_times) VALUES ('{user_name}', 0)"
        )
        conn.commit()
        conn.close()

    def increase_work_times(self, user_name):
        conn = sqlite3.connect(self.DB_NAME)
        cursor = conn.execute(
            f"UPDATE users SET work_times = work_times + 1 WHERE name = '{user_name}'"
        )
        conn.commit()
        conn.close()
```

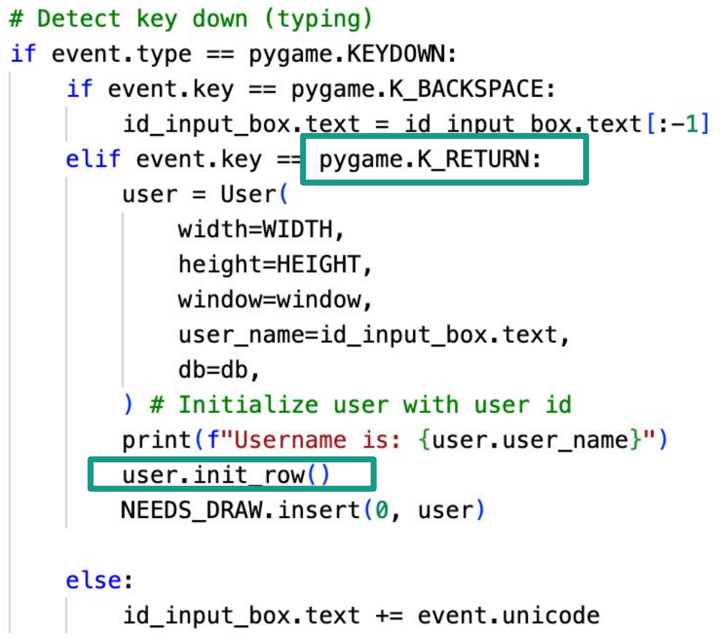
Retrieve the 1st row



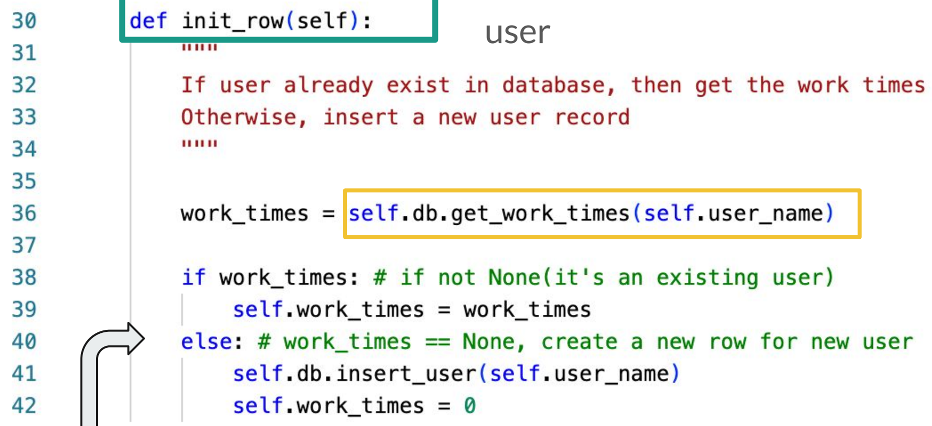
```
else: # work_times == None, create a new row for new us
    self.db.insert_user(self.user_name)
    self.work_times = 0
```

insert a record with the specified user name into the user table, ignoring it if the username already exists.

Code Structure: KEYDOWN() & Database

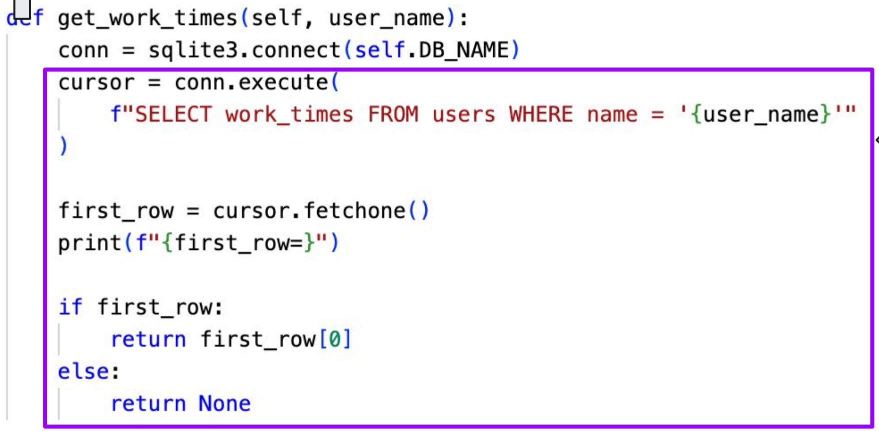


```python
# Detect key down (typing)
if event.type == pygame.KEYDOWN:
    if event.key == pygame.K_BACKSPACE:
        id_input_box.text = id_input_box.text[:-1]
    elif event.key == pygame.K_RETURN:
        user = User(
            width=WIDTH,
            height=HEIGHT,
            window=window,
            user_name=id_input_box.text,
            db=db,
        ) # Initialize user with user id
        print(f"Username is: {user.user_name}")
        user.init_row()
        NEEDS_DRAW.insert(0, user)

    else:
        id_input_box.text += event.unicode
```



user

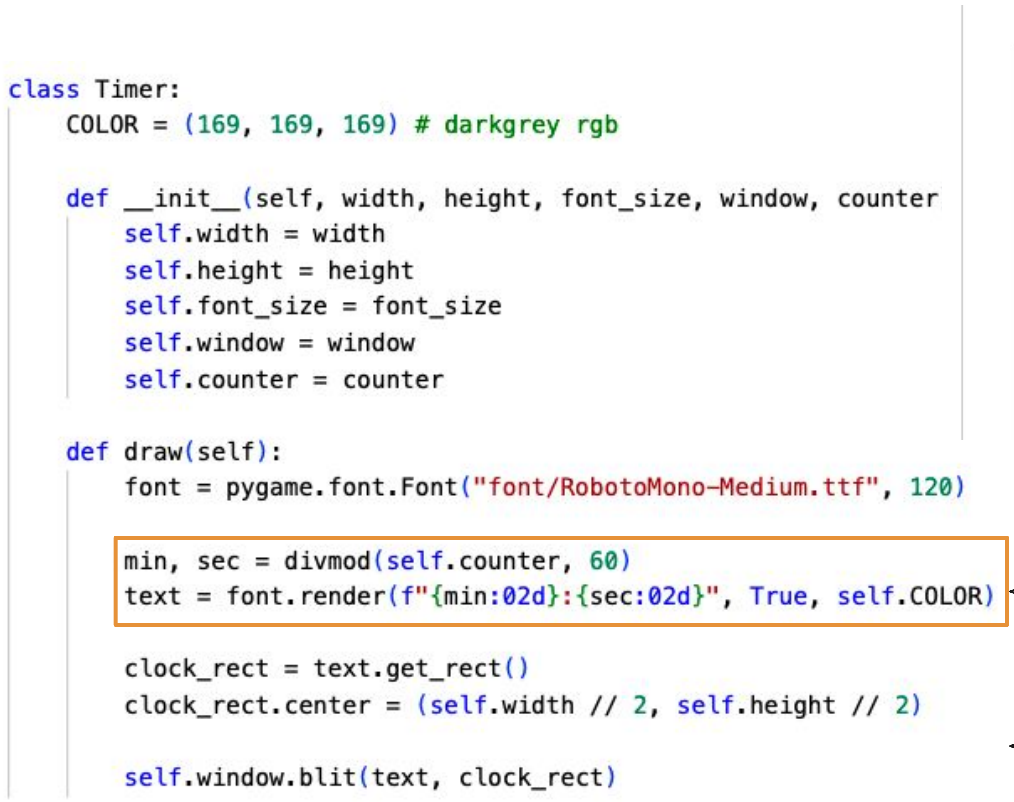
```python
def init_row(self):
    """
    If user already exist in database, then get the work times
    Otherwise, insert a new user record
    """

    work_times = self.db.get_work_times(self.user_name)

    if work_times: # if not None(it's an existing user)
        self.work_times = work_times
    else: # work_times == None, create a new row for new user
        self.db.insert_user(self.user_name)
        self.work_times = 0
```



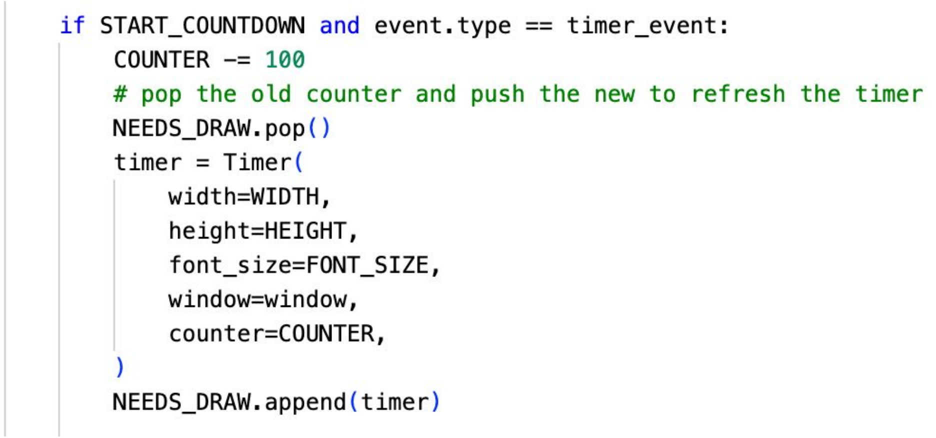
```python
def get_work_times(self, user_name):
    conn = sqlite3.connect(self.DB_NAME)
    cursor = conn.execute(
        f"SELECT work_times FROM users WHERE name = '{user_name}'"
    )

    first_row = cursor.fetchone()
    print(f"{first_row=}")

    if first_row:
        return first_row[0]
    else:
        return None
```

Check whether the user name had already existed in the database. If exists, we just update the worktimes to db. Otherwise, we insert a new row in table and set worktimes to 0.

Code Structure: Timer Class



```
class Timer:
    COLOR = (169, 169, 169) # darkgrey rgb

    def __init__(self, width, height, font_size, window, counter
        self.width = width
        self.height = height
        self.font_size = font_size
        self.window = window
        self.counter = counter

    def draw(self):
        font = pygame.font.Font("font/RobotoMono-Medium.ttf", 120)

        min, sec = divmod(self.counter, 60)
        text = font.render(f"{min:02d}:{sec:02d}", True, self.COLOR)

        clock_rect = text.get_rect()
        clock_rect.center = (self.width // 2, self.height // 2)

        self.window.blit(text, clock_rect)
```

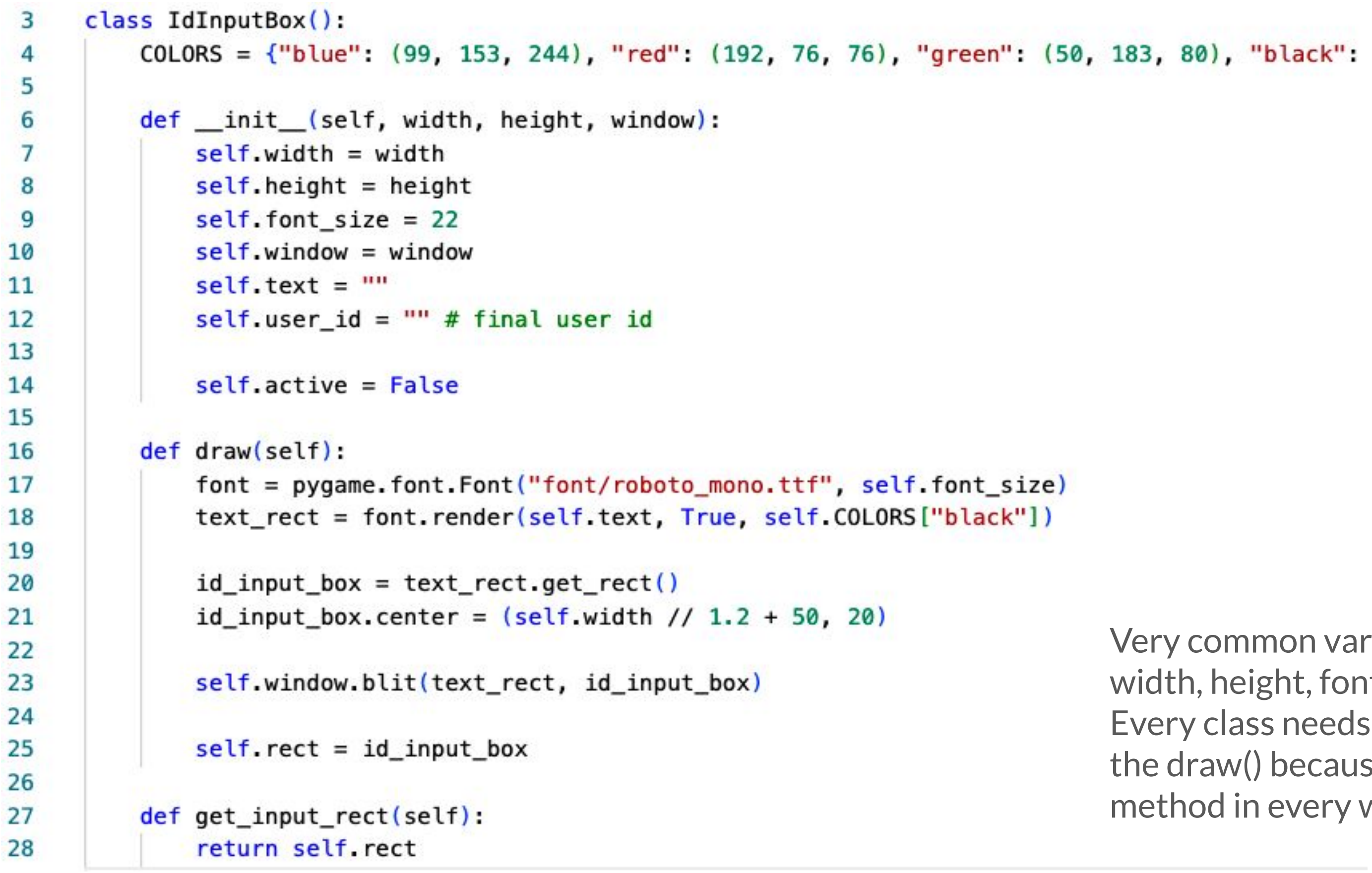


```
if START_COUNTDOWN and event.type == timer_event:
    COUNTER -= 100
    # pop the old counter and push the new to refresh the timer
    NEEDS_DRAW.pop()
    timer = Timer(
        width=WIDTH,
        height=HEIGHT,
        font_size=FONT_SIZE,
        window=window,
        counter=COUNTER,
    )
    NEEDS_DRAW.append(timer)
```

☼ **divmod** takes two arguments and returns a tuple containing the quotient and remainder of dividing the first argument by the second.

☼ **:02d** ensures that the number is formatted with at least two digits, and leading zeros are added if necessary.

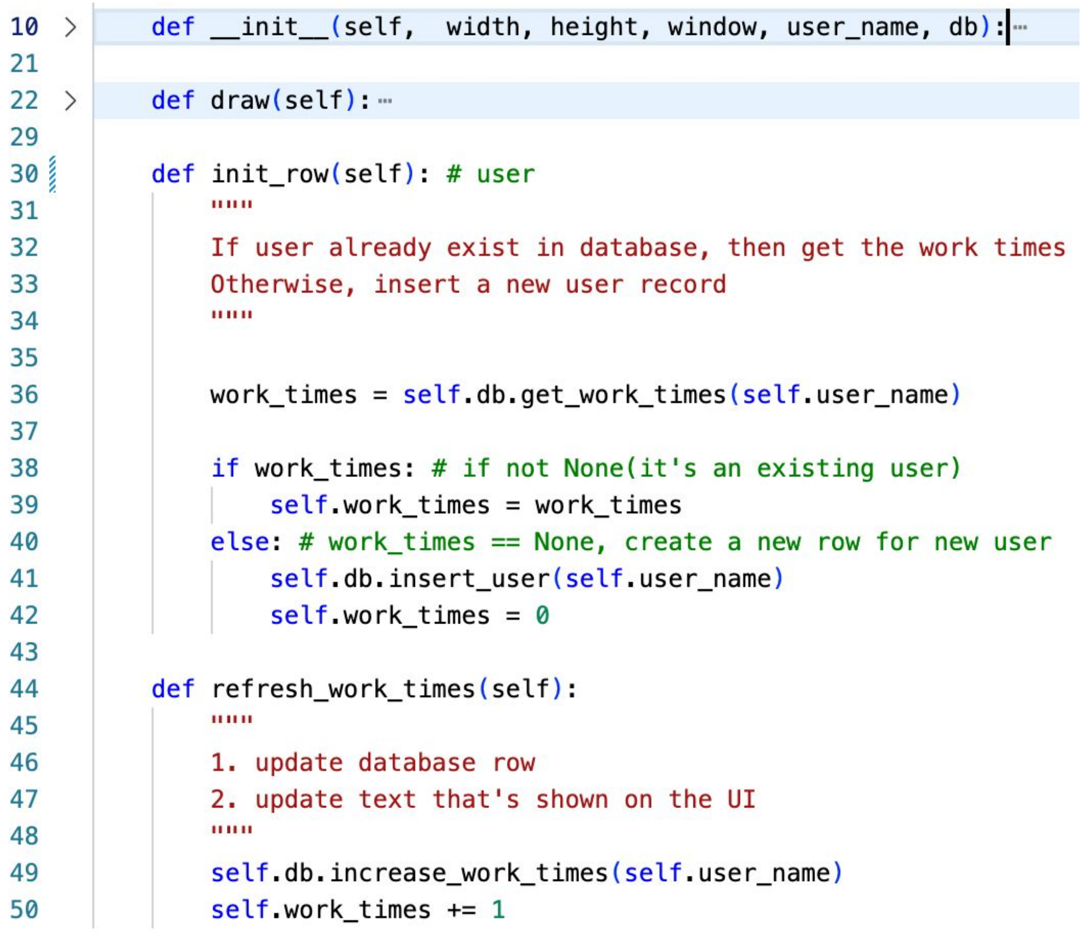
Code Structure: ID Input Box Class



```python
class IdInputBox():
    COLORS = {"blue": (99, 153, 244), "red": (192, 76, 76), "green": (50, 183, 80), "black":

    def __init__(self, width, height, window):
        self.width = width
        self.height = height
        self.font_size = 22
        self.window = window
        self.text = ""
        self.user_id = "" # final user id

        self.active = False

    def draw(self):
        font = pygame.font.Font("font/roboto_mono.ttf", self.font_size)
        text_rect = font.render(self.text, True, self.COLORS["black"])

        id_input_box = text_rect.get_rect()
        id_input_box.center = (self.width // 1.2 + 50, 20)

        self.window.blit(text_rect, id_input_box)

        self.rect = id_input_box

    def get_input_rect(self):
        return self.rect
```

Very common variables like width, height, font and window. Every class needs to implement the draw() because we call this method in every whileloop.

Code Structure: User Class



```
    def __init__(self,  width, height, window, user_name, db):⋯

    def draw(self):⋯

    def init_row(self): # user
        """
        If user already exist in database, then get the work times
        Otherwise, insert a new user record
        """


        work_times = self.db.get_work_times(self.user_name)

        if work_times: # if not None(it's an existing user)
            self.work_times = work_times
        else: # work_times == None, create a new row for new user
            self.db.insert_user(self.user_name)
            self.work_times = 0

    def refresh_work_times(self):
        """
        1. update database row
        2. update text that's shown on the UI
        """
        self.db.increase_work_times(self.user_name)
        self.work_times += 1
```

Limitations

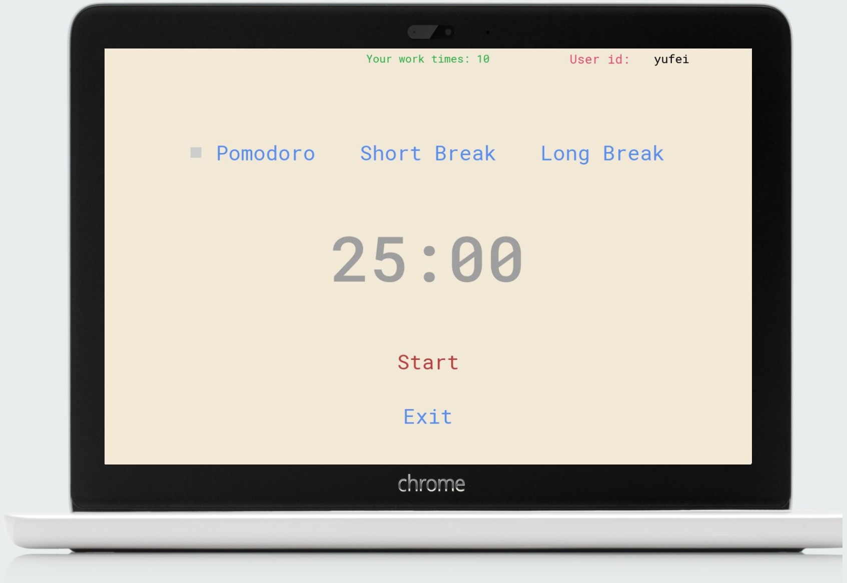
Limitations

1. **Timer State Preservation**: The current program does not save the state of the timer (e.g., whether it is in work or rest mode) when the program is closed or restarted. Consider adding state saving and restoration functionality.
2. **More Unit Customization Options**: Provide users with more customization options, such as adjusting the length of work and break periods, customizing notification sounds, etc.
3. **User Switching Within Program Execution:** Lacks the functionality to facilitate user switching without restarting the program
4. **Richer Statistics and Reports**: While basic work time statistics are implemented, there is room for expansion to generate detailed work time reports, providing users with more information and insights.

Questions?

Doctors: Googling stuff online does not make you a doctor.
Programmers:

Thanks for your listening!



References

ChatGPT

https://stackoverflow.com/questions/12105198/sqlite-how-to-get-insert-or-ignore-to-work

https://dev.to/ra1nbow1/8-ways-to-add-an-element-to-the-beginning-of-a-list-and-string-in-python-925#:~:text=The%20insert()%20method%20takes,as%20the%20first%20parameter%201%20.

https://mariadb.com/kb/en/create-table/#:~:text=mode%20to%20STRICT%20.-,CREATE%20TABLE%20IF%20NOT%20EXISTS,will%20be%20triggered%20by%20default.

https://github.com/baraltech/Pomodoro-Timer-PyGame/blob/main/youtubemain.py