Checking qualifications

Demonstrate your knowledge of:

- ➔ **the time module commands;**
- ➔ **ways of working with text.**

Checking qualifications

Which command returns the current time?

How do we calculate how long the program has been running for?

Checking qualifications

The time module from the standard library contains tools for working with time

Function	*Purpose*
`time.time()`	Returns the number of seconds since the beginning of the epoch *(for UNIX systems, this is January 1, 1970)*

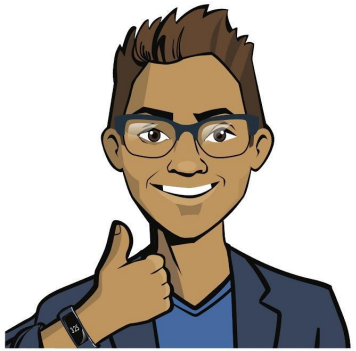
You can log the time at the beginning and at the end of the program!

Checking qualifications

The time module from the standard library contains tools for working with time

Function	*Purpose*
`time.time()`	Returns the number of seconds since the beginning of the epoch *(for UNIX systems, this is January 1, 1970)*

```
import time

start_time = time.time()
#program body
end_time = time.time()
period = end_time - start_time
```



Checking qualifications

Let's have the Area and Label classes defined in the game

How do we create a text object ?

Checking qualifications

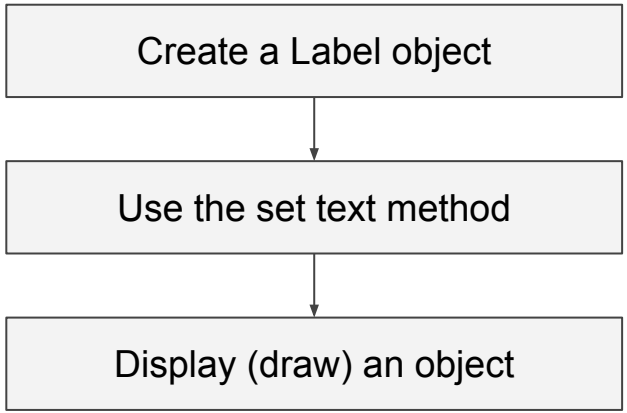
If necessary, use the Fast Clicker project code.

A text object can be created as an instance of the Label class

Arguments are passed to the constructor:

- ➔ the x and y coordinates;
- ➔ the width and height of the text object;
- ➔ the text color

The text is displayed in three steps:



Checking qualifications

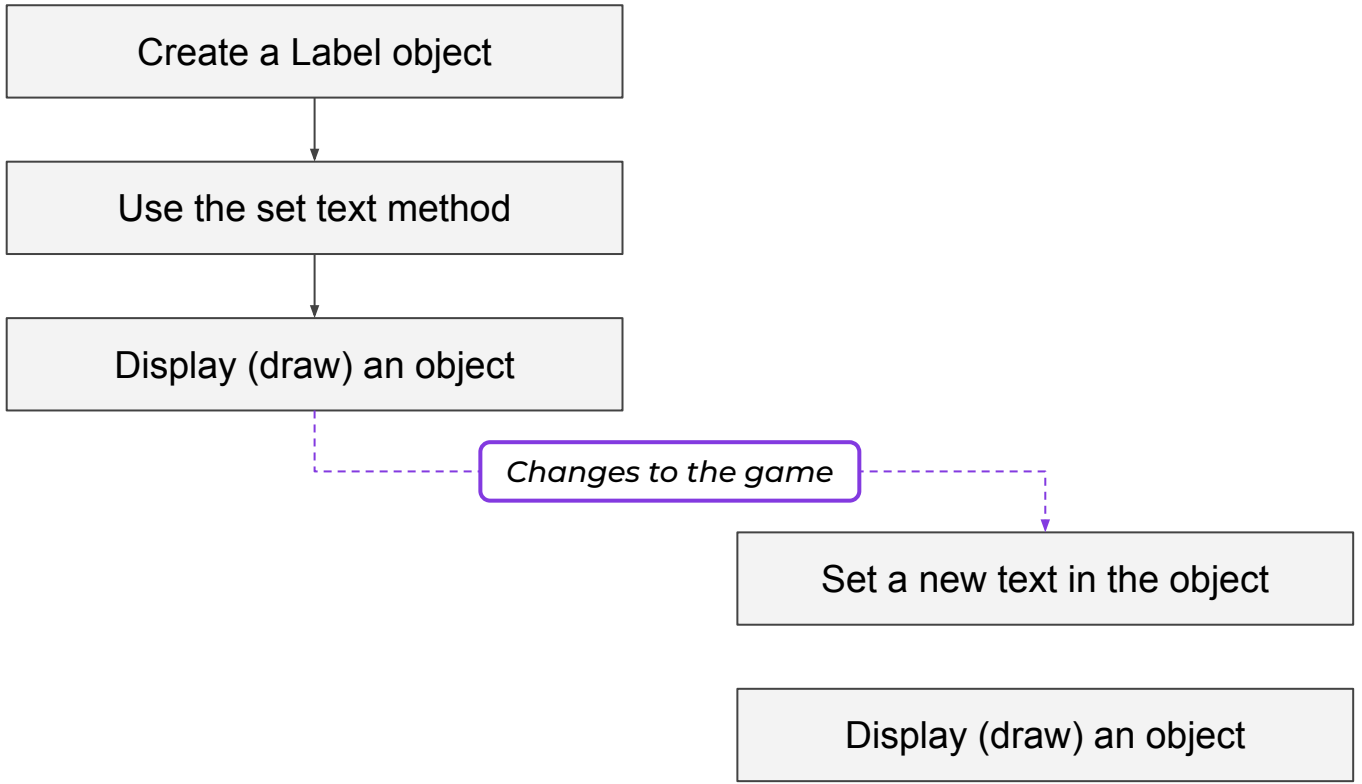
Suppose some event has occurred, after which the label text needs to be changed.

How do we replace the text with a new one?

Checking qualifications

There's no need to create a new text object.

It's enough to set a new text for the object and redraw it:



Checking qualifications

Let's say we know that the program worked for 4.153520787 seconds. The time is displayed in a text object counter.

How do we display a text with the number of seconds rounded to 4?

Checking qualifications

Let's divide the task into two steps:

1) **Making the number of seconds a round number.**
2) **Display the seconds in the counter.**

You can perform rounding using the functions int() or round().

You can display the number in the counter using the previous algorithm.

```
period = end_time - start_time

current_sec = int(period)

timer.set_text(str(current_sec), 40, DARK_BLUE)

timer.draw(0,0)
```

The text object is a counter.

Checking qualifications

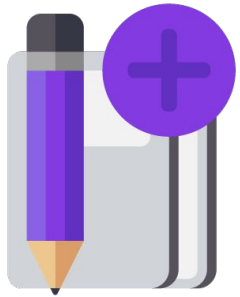
Qualifications confirmed!

Great, you are ready to brainstorm and complete your work task!

Checking qualifications

Brainstorm:

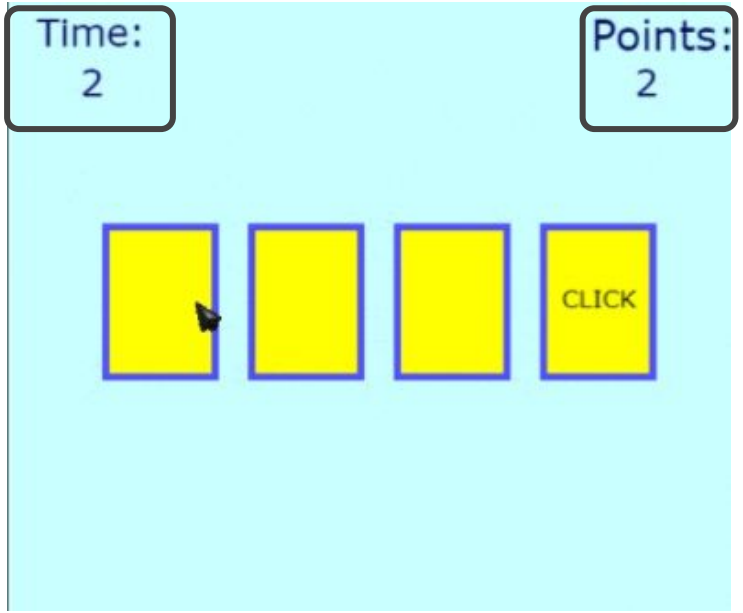
Statistics counters



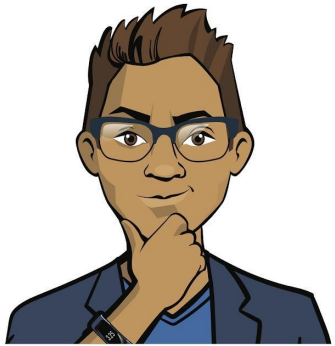
Statistics counters

You need to create two counters in the Fast Clicker game space:

- ➔ seconds since the launch of the game;
- ➔ points scored.



How do we display these labels?

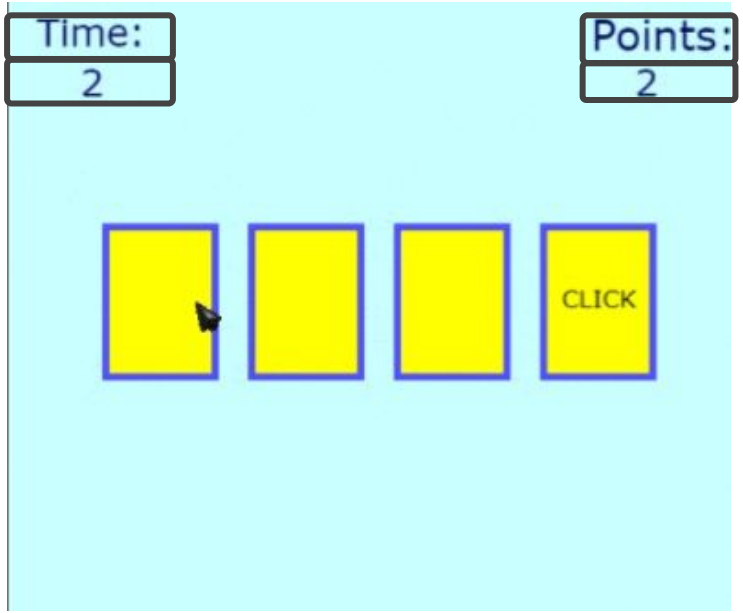


Brain storm

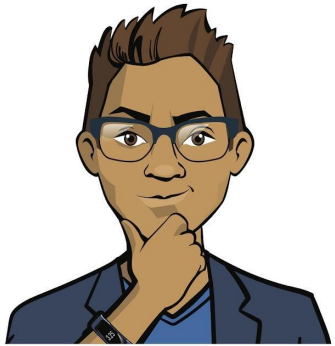
Statistics counters

Possible solution.

Set each counter with two labels: a permanent one (name) and one that updates (number).



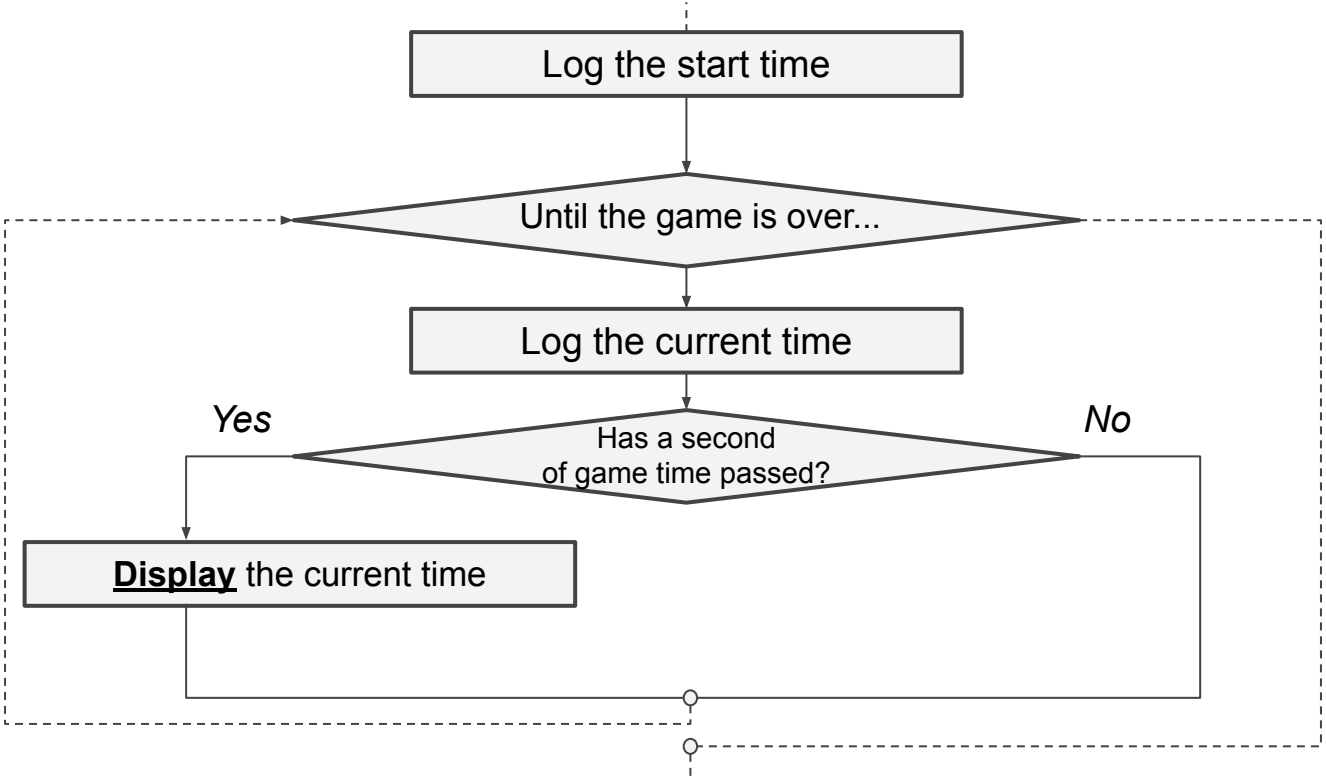
How do we organize counting the time and points in the program?



Brain storm

1. Counting the seconds since the game was launched

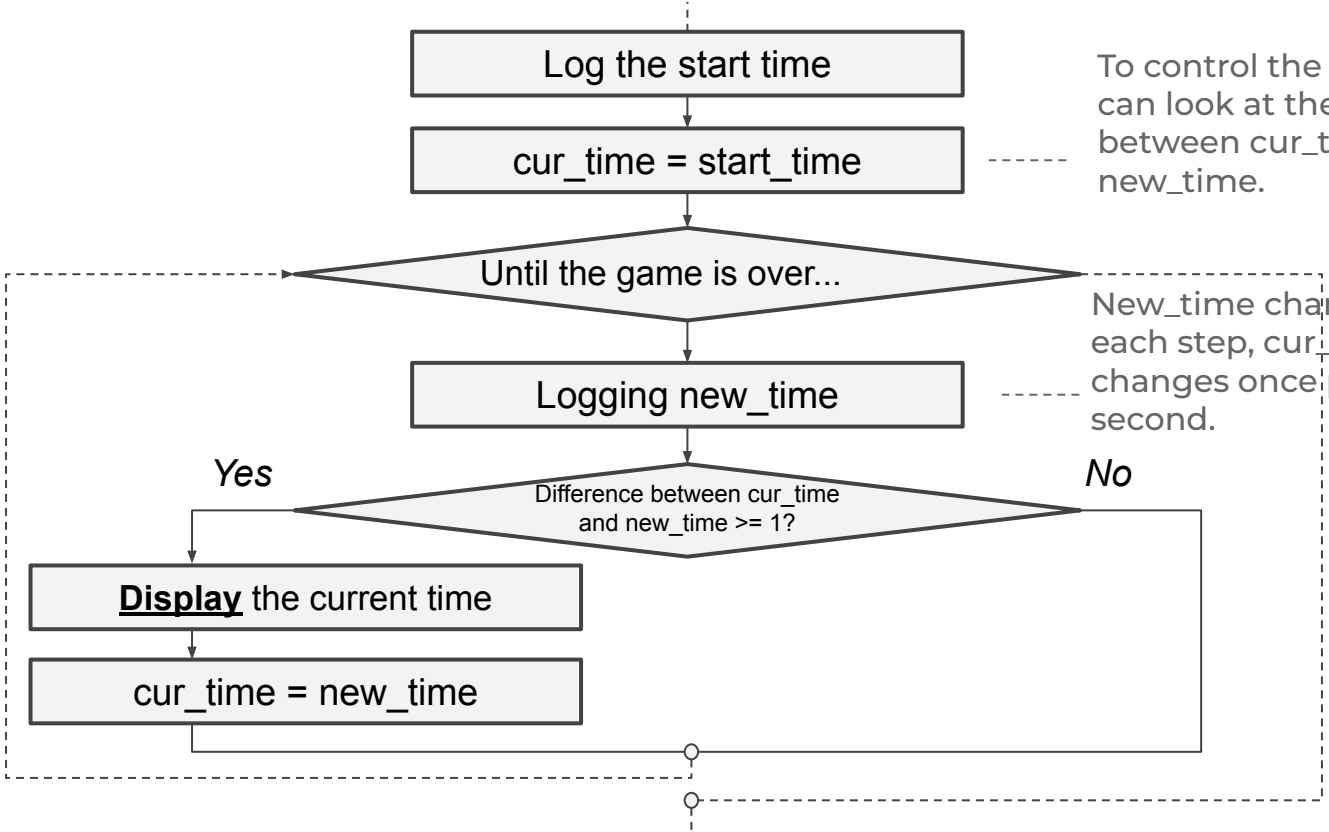
The seconds will be rounded up for the benefit of the user.
Then updating the counter is required once per second.



Brain storm

1. Counting the seconds since the game was launched

The seconds will be rounded up for the benefit of the user.
Then updating the counter is required once per second.



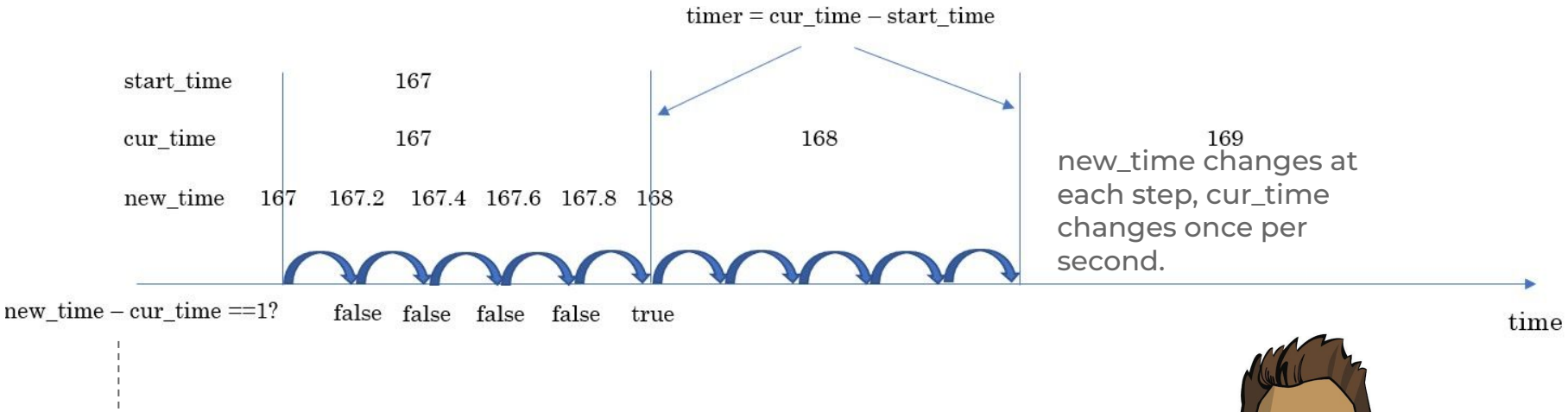
To control the time, you can look at the difference between cur_time and new_time.

New_time changes at each step, cur_time changes once per second.

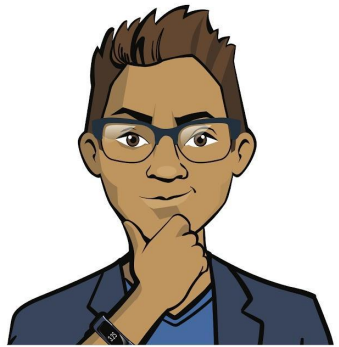
Brain storm

1. Counting the seconds since the game was launched

The seconds will be rounded up for the benefit of the user.
Then updating the counter is required once per second.



To control the time, you can look at the difference between cur_time and new_time.



1. Counting the seconds since the game was launched

The seconds will be rounded up for the benefit of the user.
Then updating the counter is required once per second.

Possible code for the timer:

```
#start and current time
start_time = time.time()
cur_time = start_time

time_text = Label(0,0,50,50,back)
#...
timer = Label(50,55,50,40,back)
#...

while True:
    #current time
    new_time = time.time()

    if new_time - cur_time >= 1:
        timer.set_text(str(int(new_time - start_time)), 40, DARK_BLUE)
        timer.draw(0,0)
        cur_time = new_time
```

Brain storm

1. Counting the seconds since the game was launched

The seconds will be rounded up for the benefit of the user.
Then updating the counter is required once per second.

Possible code for the timer:

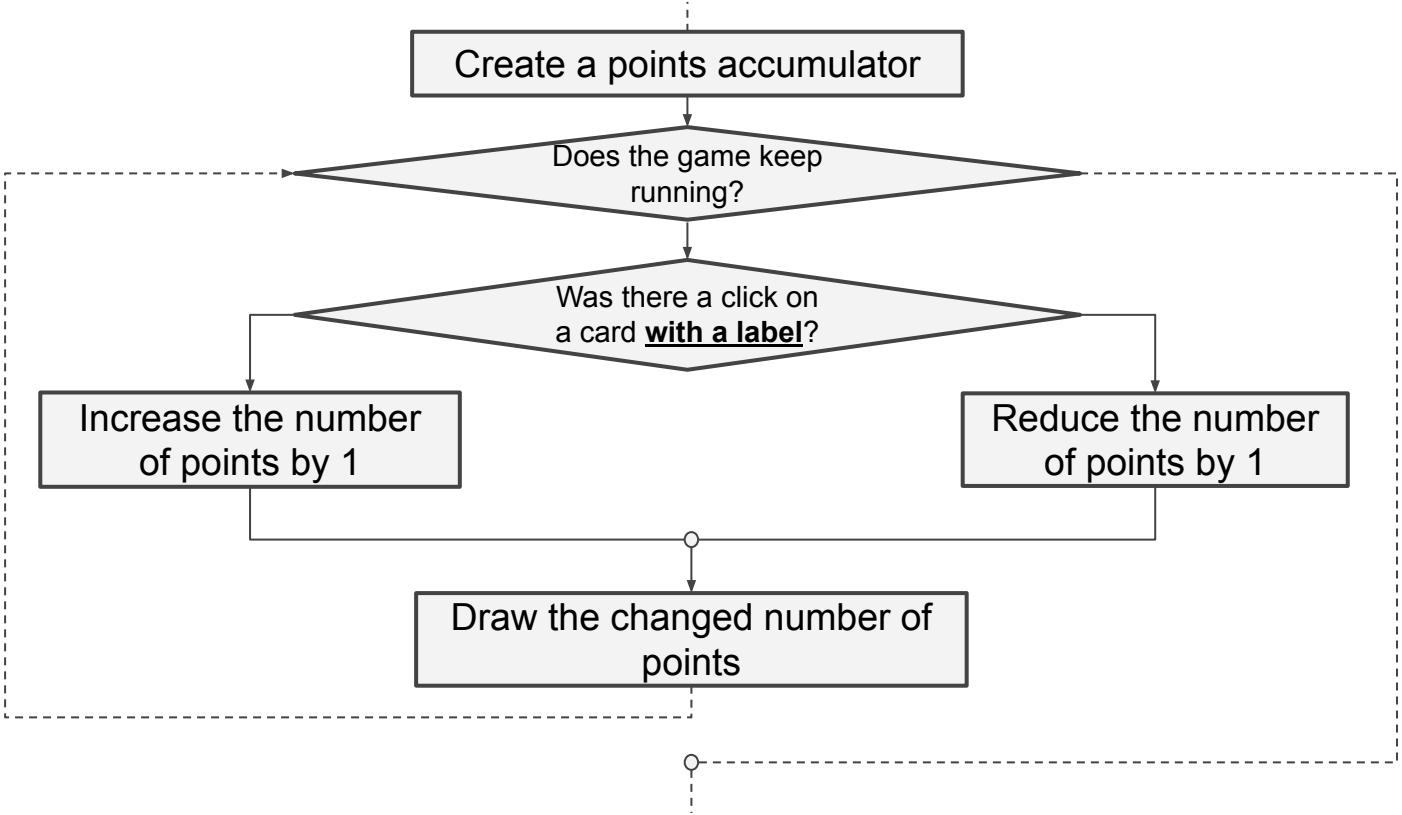
```
#start and current time
start_time = time.time()
cur_time = start_time

time_text = Label(0,0,50,50,back)
#...
timer = Label(50,55,50,40,back)
#...

while True:
    #current time
    new_time = time.time()

    if new_time - cur_time >= 1:
        timer.set_text(str(int(new_time - start_time)), 40, DARK_BLUE)
        timer.draw(0,0)
        cur_time = new_time
```

We will update the counter once every 1 second so we don't create a stressful situation.

Brain storm

2. Counting scored points

Let's consider a complicated version of the game, where incorrect clicks deduct one point.



Brain storm

2. Counting scored points

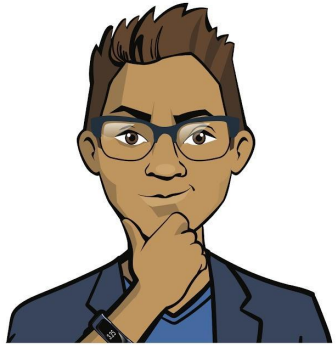
Let's consider a complicated version of the game, where incorrect clicks deduct one point.

```
points = 0
while True:
    If there was a mouse click...
    if cards[i].collidepoint(x,y):
        if i + 1 == click:
            cards[i].color(GREEN)
            points += 1
        else:
             cards[i].color(RED)
            points -= 1
        #rendering the counter
```

Brain storm

Your tasks:

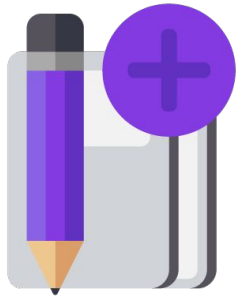
1. Create text objects to display statistics and counter variables.
2. Implement the calculation and display of elapsed time and points scored.



Brain storm

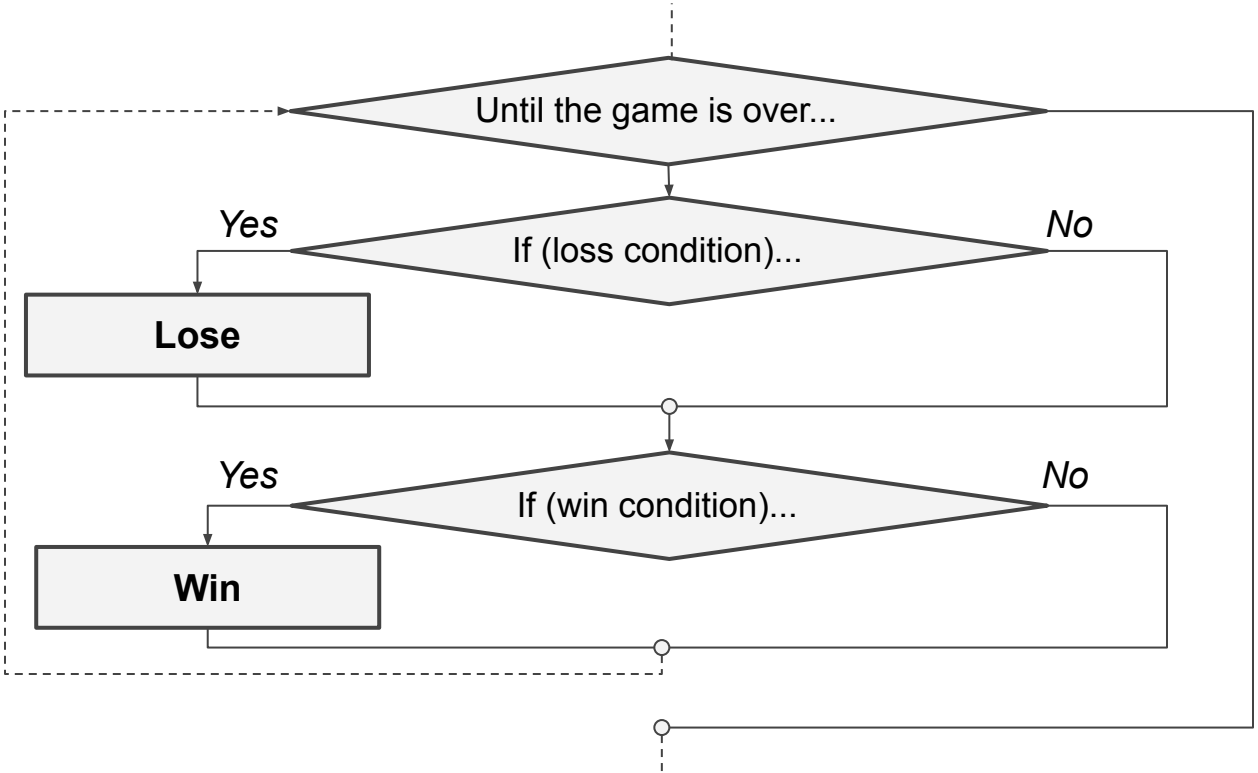
Brainstorm:

Winning and losing



Let's program winning and losing!

Whether a win or loss has occurred should be determined at each step of the game loop:



Brain storm

Let's program winning and losing!

The conditions may be as follows:

What	*Trigger condition*
Lose	11 or more seconds have passed since the start of the game
Win	5 or more points have been scored

These expressions are enough!

Brain storm

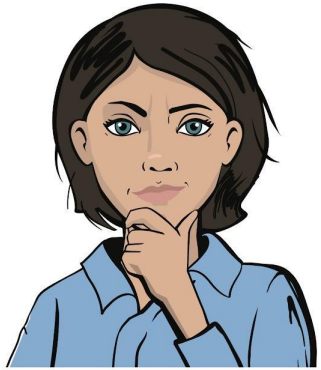
Let's program winning and losing!

The conditions may be as follows:

What	*Trigger condition*
Lose	11 or more seconds have passed since the start of the game
Win	5 or more points have been scored

Judge for yourself:

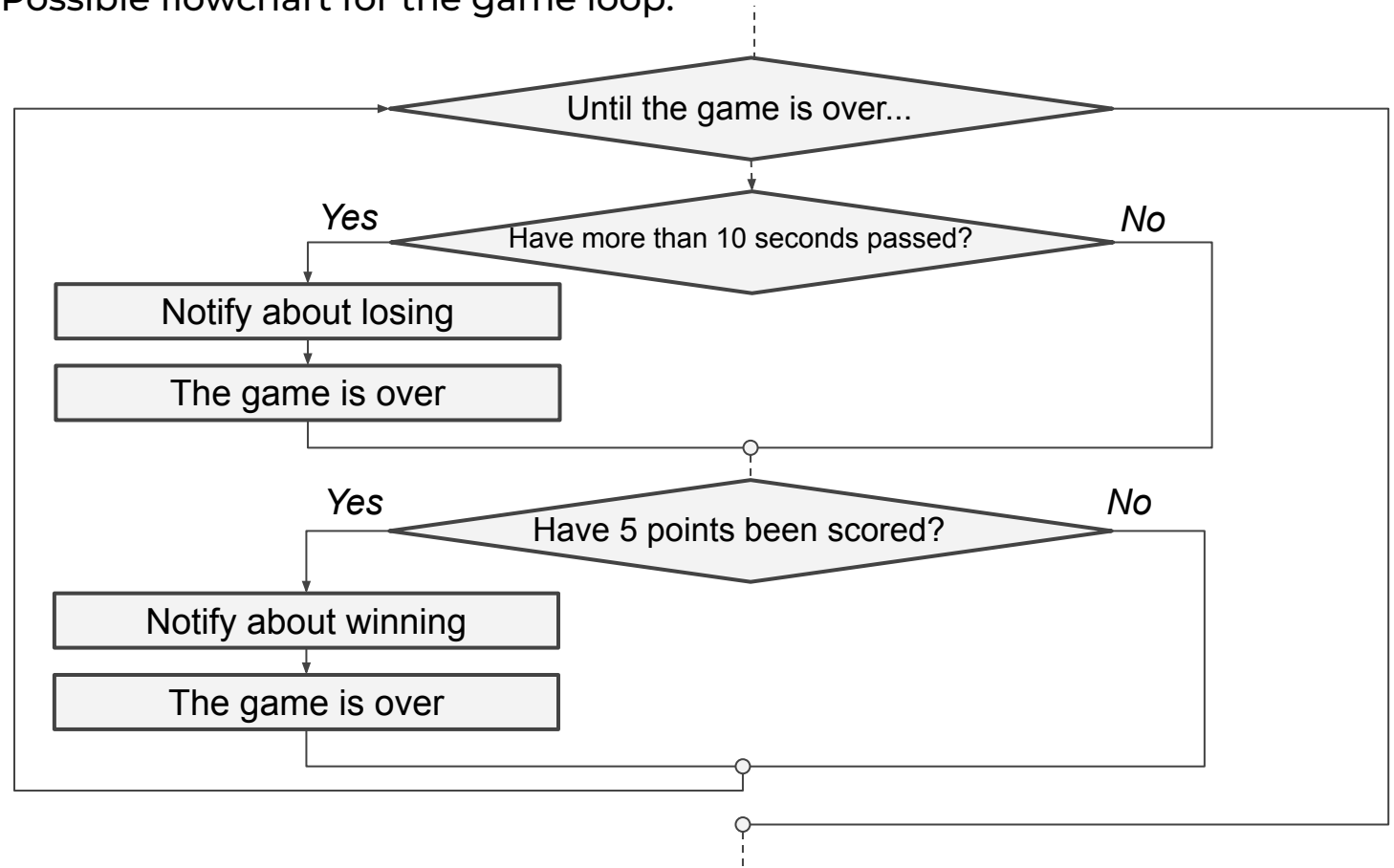
- ➔ You ***don't need time control*** to win. If the time has expired, it will automatically lead to a loss.
- ➔ The ***number of points does not matter*** for a loss. If there is a loss, it means that the win condition did not activate in the allotted time.



Brain storm

Let's program winning and losing!

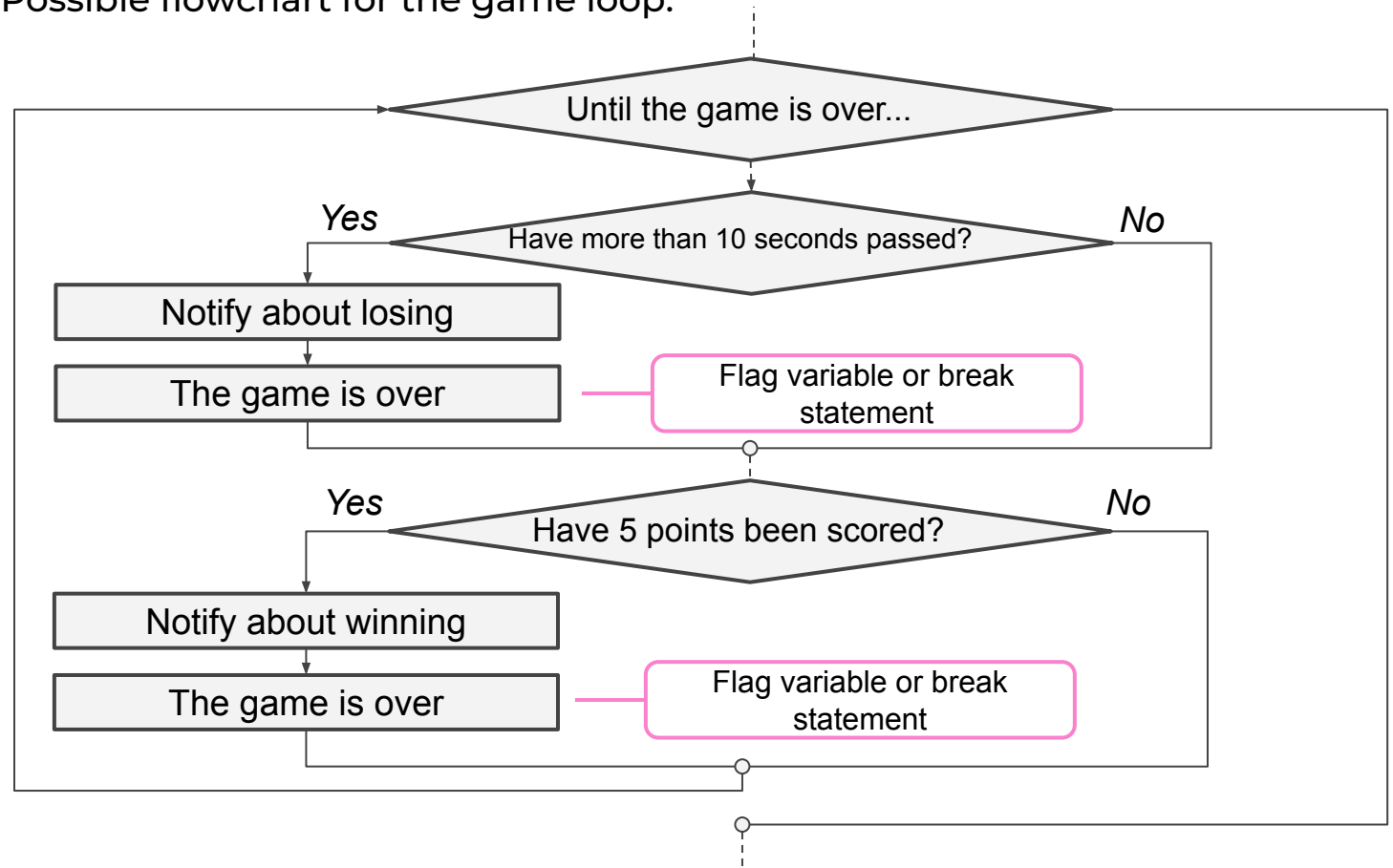
Possible flowchart for the game loop:



Brain storm

Let's program winning and losing!

Possible flowchart for the game loop:



Brain storm

Let's program winning and losing!

Game code:

```
while True:
    if new_time - start_time  >= 11:
        win = Label(0, 0, 500, 500, LIGHT_RED)
        win.set_text("Time's up!!!", 60, DARK_BLUE)
        win.draw(110, 180)
        break
```

Lose

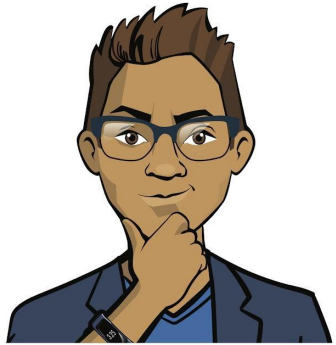
```
    if points >= 5:
        win = Label(0, 0, 500, 500, LIGHT_GREEN)
        win.set_text("You win!!!", 60, DARK_BLUE)
        win.draw(140, 180)
        resul_time = Label(90, 230, 250, 250, LIGHT_GREEN)
        resul_time.set_text(
               "Completion time: " + str(int(new_time - start_time)) + " sec", 40, DARK_BLUE
        )
        resul_time.draw(0, 0)
        break
```

Win

Brain storm

Your task:

- ➔ Complete the game loop with conditional statements responsible for the winning and losing conditions.
- ➔ Launch and test the game. If necessary, adjust its difficulty level.



Brain storm