24

Quiz Games

Introduction

Today, we are going to write a different type of game — a quiz game.
We will start with the design of the screen and dealing with events (player clicked on correct or wrong answer, time ran out, etc) but later we will focus on ways to organise and manage question banks. We will use external files to store our questions and so we will need to learn how to find, read and write files in python.

As in our earlier games we will develop multiple versions as we add more and more features.

① **Basic quiz game** .... `quiz_basic.py`

Our starting version of the game consists is again taken from excellent *Coding Games with Python* book.

We will use this game as a starting point for dealing (reading/writing) files.

② **Adding sounds** .... `quiz_sound.py`

Go to www.zapsplat.com, and find some sounds that we can use
when you click on the correct answer, when you click on an incorrect answer, and when you run out of time. We will cover how to download and convert in order to use in our game.

For background music have a look at www.melodyloops.com
This has a good selection of tracks and can cut a track to whatever length you want — you could set the length to match the time given to complete the level.

③ **Question manager/generator** .... `quiz_generator.py`

Currently we only have a small number of questions and the question data is stuck in the middle of our program code. This is not great. A better approach would be to store questions in external files. Once we do that we could then start thinking about other improvements:

- we could organise questions by topic.
- we could organise questions by difficulty.
- we could keep track of how often a question is answered correctly and use this to update how difficult a question is ...

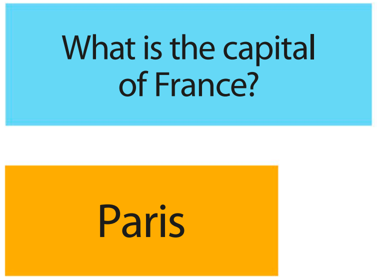
① quiz_basic

How to build Big Quiz

Put your coding skills to the test and create a quiz game to challenge your friends. You're the quizmaster, so you can make the questions about any topic you like.

What happens

When the game begins, the first question is shown on the screen along with four possible answers. The player has ten seconds to click on an answer box. If they get the right answer, the game moves on to the next question. If the player chooses a wrong answer, or if the time runs out, the game ends and the final score is displayed on the screen.

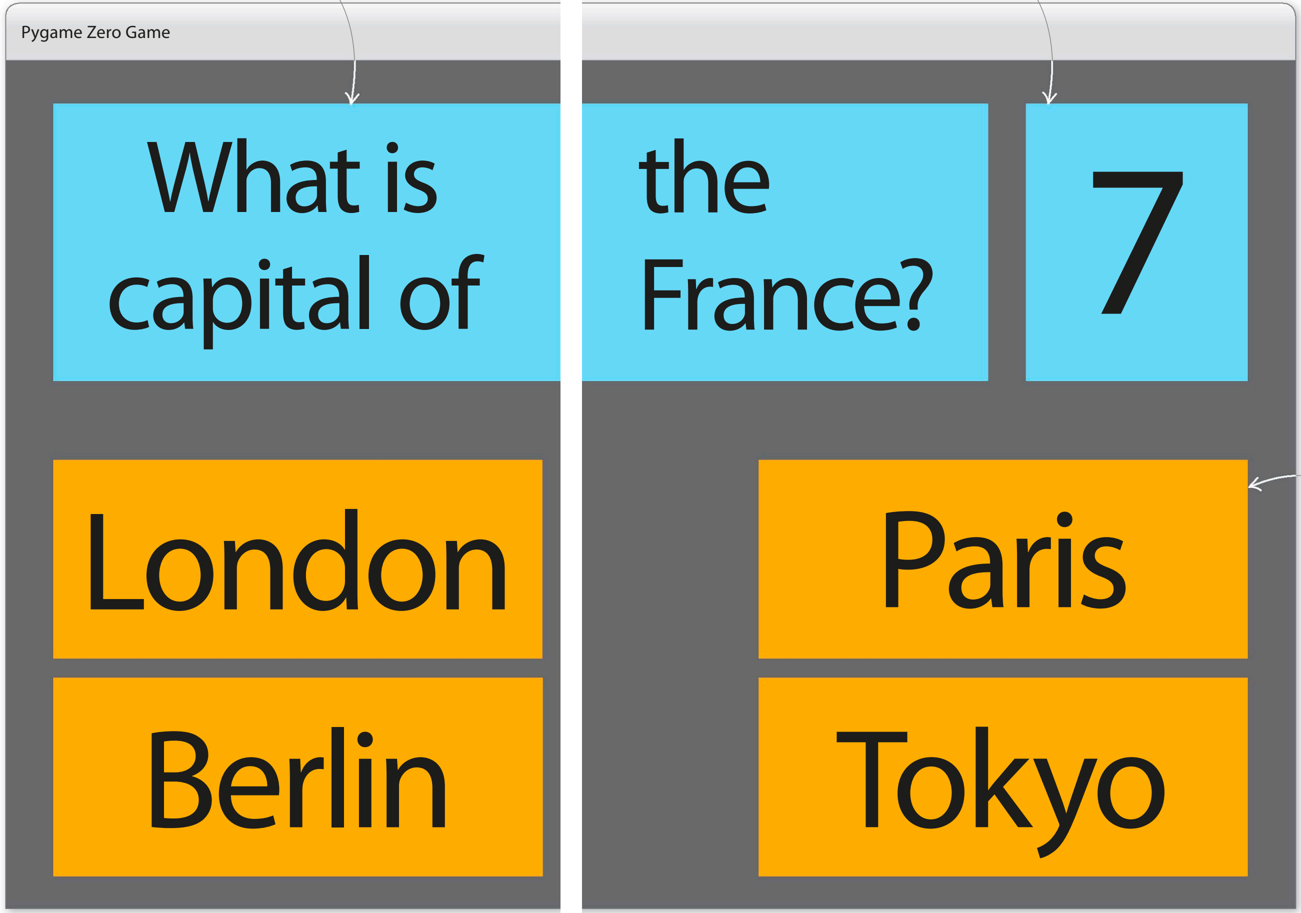

What is the capital of France?

Paris

△ **Boxes**
This game doesn't use images. Instead, the questions, answers, and the timer are displayed in colorful boxes that you create using code.

Each question is displayed here.

The game starts with ten seconds on the timer. If it reaches zero, the game ends.


Pygame Zero Game

What is the capital of France?

7

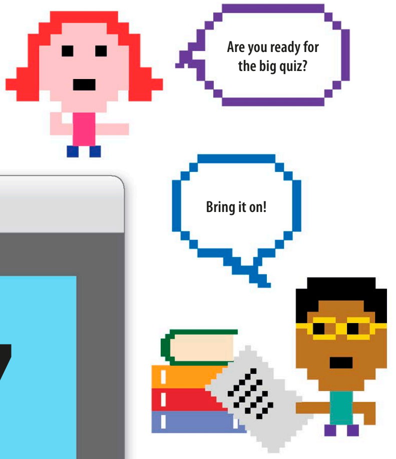
London

Paris

Berlin

Tokyo

The possible answers are displayed in orange boxes. You can change them to a different color if you like.

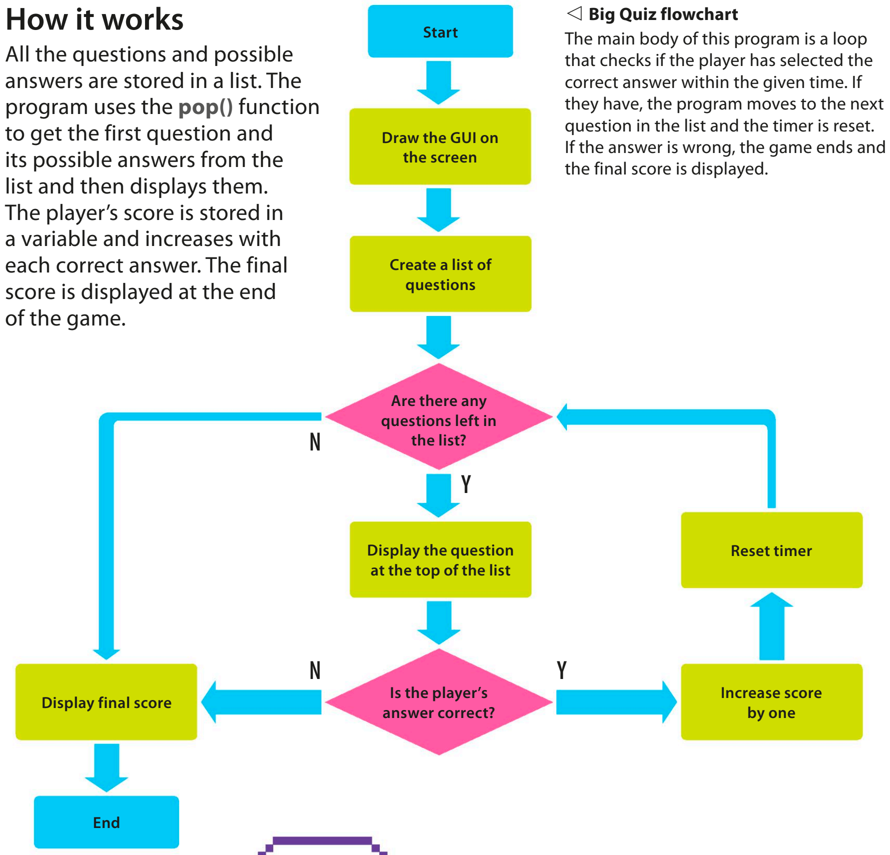


◁ **Quiz time!**
This program uses a Graphical User Interface (GUI, pronounced "goo-ey"), which is the visible part of a program that a user interacts with. In this game, you'll build a GUI using Pygame Zero.

① quiz_basic

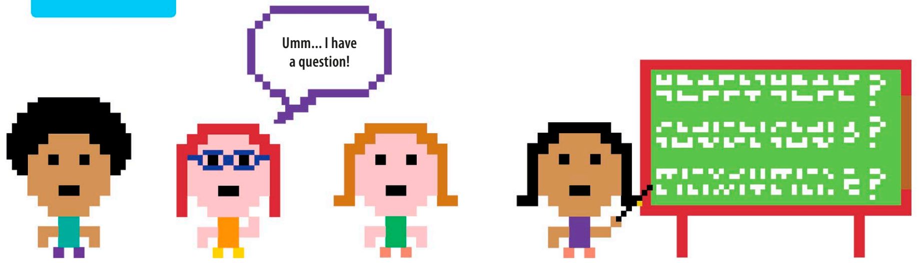
How it works

All the questions and possible answers are stored in a list. The program uses the **pop()** function to get the first question and its possible answers from the list and then displays them. The player's score is stored in a variable and increases with each correct answer. The final score is displayed at the end of the game.



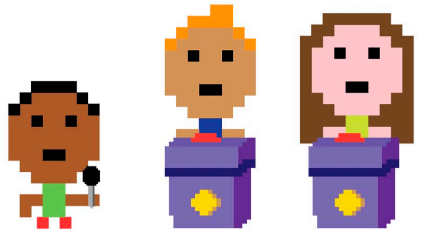
Start
Draw the GUI on the screen
Create a list of questions
Are there any questions left in the list?
N
Y
Display the question at the top of the list
Is the player's answer correct?
N
Y
Increase score by one
Reset timer
Display final score
End

◁ **Big Quiz flowchart**
The main body of this program is a loop that checks if the player has selected the correct answer within the given time. If they have, the program moves to the next question in the list and the timer is reset. If the answer is wrong, the game ends and the final score is displayed.



Thinking caps on!

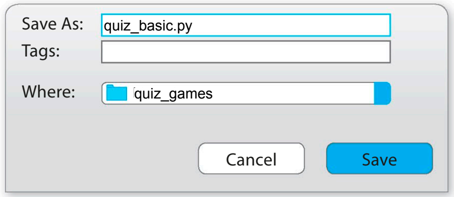
There may be a time limit to answer the questions, but not to build the game! Follow these steps carefully to build your own quiz show to play with your friends and family.



Set it up

1 As usual we are going to create a separate folder to hold our quiz based games.

- In Mu editor, start a new file. and click on Save.
- Change directory to `coderdojo_tramore`.
- Inside folder `coderdojo_tramore`, create a new subfolder called `quiz_games`.
- Set file name to `quiz_basic.py`.



Save As: quiz_basic.py
Tags:
Where: quiz_games

2 **Set the screen size**
Next you need to define the size of the playing area. Add this code to the very top of your program to set the width and height of the game screen.

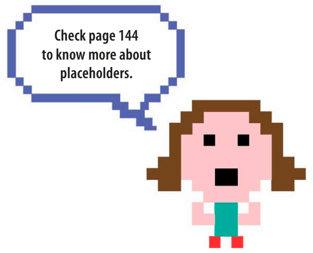
```
WIDTH = 1280
HEIGHT = 720
```

These values are in pixels.

3 **Create the stubs**
You don't need any images for this game, so you can jump straight into writing the code. First create placeholders for the functions you'll need to build the quiz.

```
def draw():
    pass

def game_over():
    pass

def correct_answer():
    pass

def on_mouse_down(pos):
    pass

def update_time_left():
    pass
```

Remember, **pass** is used as a placeholder for functions that you don't want to define right away.

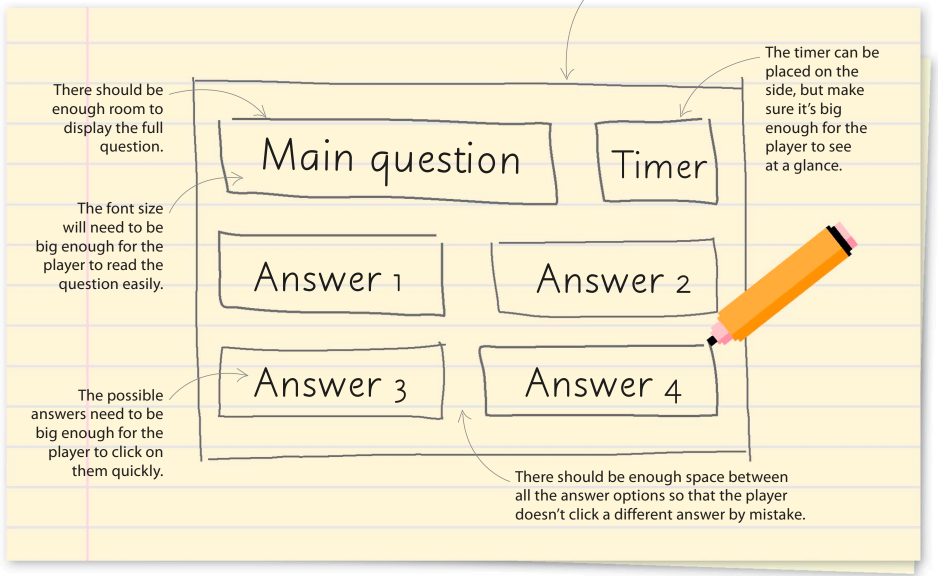


① quiz_basic

4 Plan the interface

When building this game, you need to think about the way it looks, or its "interface." The player needs to be able to see the question, its possible answers, and a timer that shows how long they've got left. Here's a sketch of how you might want the interface to look.

When planning the interface of a game, try sketching it on paper before writing any code.



There should be enough room to display the full question.

The font size will need to be big enough for the player to read the question easily.

The possible answers need to be big enough for the player to click on them quickly.

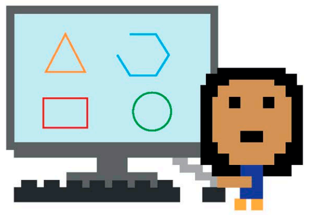
The timer can be placed on the side, but make sure it's big enough for the player to see at a glance.

There should be enough space between all the answer options so that the player doesn't click a different answer by mistake.

LINGO

Wireframes

Computer game designers can plan their game interfaces using wireframes. These are diagrams that show the different parts of an interface that the player sees on screen. They can be drawn by hand or made using a simple drawing tool on a computer. By doing this, the interface can be tested, and any changes to the design can be made before writing the code.



5 Create a box for the interface

Now that you've planned what the interface will look like, you can create the rectangular boxes that will make up the GUI. Type this code below what you typed in Step 2 to create a box for the main question.

```
WIDTH = 1280
HEIGHT = 720

main_box = Rect(0, 0, 820, 240)
```

This function takes four parameters. The first two numbers are the coordinates of the top-left corner of the box, and the last two numbers are the coordinates of the bottom-right corner of the box.

This sets the box size to 820 pixels wide and 240 pixels high.

6 Make the other boxes

You now need to make a box for the timer and four separate boxes for each of the possible answers. Type this code under what you wrote in Step 5.

```
main_box = Rect(0, 0, 820, 240)
timer_box = Rect(0, 0, 240, 240)
answer_box1 = Rect(0, 0, 495, 165)
answer_box2 = Rect(0, 0, 495, 165)
answer_box3 = Rect(0, 0, 495, 165)
answer_box4 = Rect(0, 0, 495, 165)
```

The timer box is a square 240 pixels wide and 240 pixels high.

All the answer boxes are the same size.

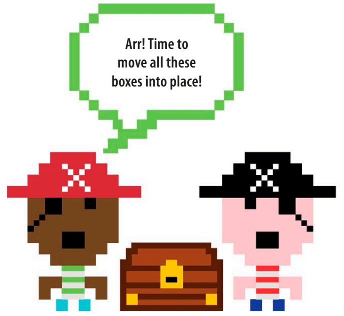
7 Move the boxes

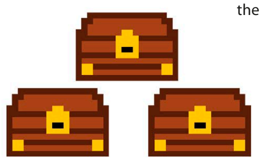
At the moment, all the boxes will be drawn on top of each other in the top-left corner. You need to add some code to move them to their correct positions on the screen. Type this code immediately after the code from Step 6.

```
answer_box4 = Rect(0, 0, 495, 165)

main_box.move_ip(50, 40)
timer_box.move_ip(990, 40)
answer_box1.move_ip(50, 358)
answer_box2.move_ip(735, 358)
answer_box3.move_ip(50, 538)
answer_box4.move_ip(735, 538)
```

move_ip() moves each rectangle to the place you want it on the screen.

The top-left corner of each box will be placed at the coordinates in the brackets.





① quiz_basic

8 Create a list of answer boxes

This game uses four boxes to show the possible answers to each question. You can keep track of these boxes by using a list. Add this code immediately after what you typed in Step 7.

```
answer_box4.move_ip(735, 538)
answer_boxes = [answer_box1, answer_box2, answer_box3, answer_box4]
```

This list holds all the answer boxes.

9 Draw the boxes

Now that you've created the boxes, it's time to add some code to draw them on the screen. Replace **pass** under **def draw()** from Step 3 with this code.

```
def draw():
    screen.fill("dim gray")
    screen.draw.filled_rect(main_box, "sky blue")
    screen.draw.filled_rect(timer_box, "sky blue")

    for box in answer_boxes:
        screen.draw.filled_rect(box, "orange")
```

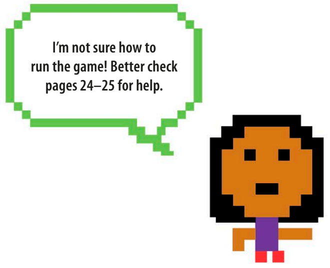
This sets the background to a dim gray color.

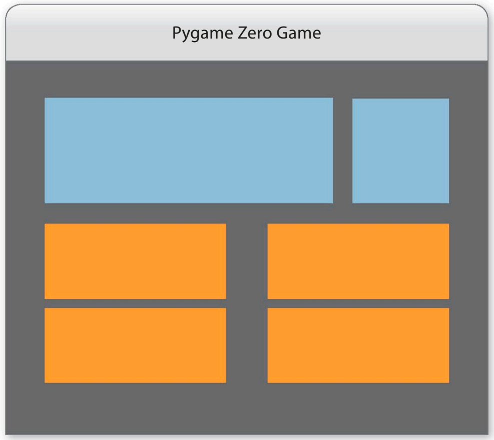
These lines draw the main box and the timer on the screen and colors them sky blue.

This draws every box in the **answer_boxes** list on the screen and colors them all orange.

10 Try it out

Save your file and run it from the command line in the Command Prompt or Terminal window. You should be able to see your GUI, ready to be filled with quiz questions. If your program fails to run successfully, go back to your code and try to catch those bugs!


I'm not sure how to run the game! Better check pages 24–25 for help.



11 Set the score

Now that the interface is ready, you need to start thinking about how the game will work. Create a variable to hold the score and set it to zero. Type this after the code you wrote in Step 8.

```
answer_boxes = [answer_box1, answer_box2, answer_box3, answer_box4]

score = 0
```

Don't forget to save your work.

12 Set the timer

You also need to create a timer that will hold the number of seconds the player has left to answer each question. You can give them ten seconds to answer by setting the variable to **10**.

```
score = 0
time_left = 10
```

This is the number of seconds the player has to answer each question.

13 Add the first question

It's time to create the first quiz question. All the questions will be multiple choice, which means there are several possible answers, but only one of them is correct. You can use a list to store the information about each question. Type this code next.

This is the name of the list. Here it means question 1.

This question is the first item in the list.

```
time_left = 10

q1 = ["What is the capital of France?",
      "London", "Paris", "Berlin", "Tokyo", 2]
```

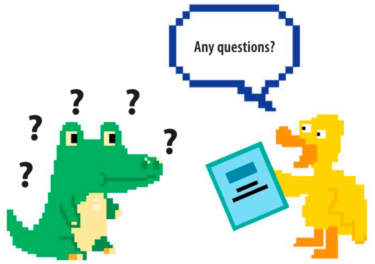
These are all the possible answers to the question.

This number indicates the position of the correct answer. Here it's **2**, which means **Paris** is the correct answer.

14 More questions

Let's add some more questions by typing the code shown in black below the lines from Step 13. Remember, you can create your own sets of questions if you like. You could base them on your favorite sports team, or show off what you know about your favorite series of books.

```
q1 = ["What is the capital of France?",
      "London", "Paris", "Berlin", "Tokyo", 2]

q2 = ["What is 5+7?",
      "12", "10", "14", "8", 1]

q3 = ["What is the seventh month of the year?",
      "April", "May", "June", "July", 4]

q4 = ["Which planet is closest to the Sun?",
      "Saturn", "Neptune", "Mercury", "Venus", 3]

q5 = ["Where are the pyramids?",
      "India", "Egypt", "Morocco", "Canada", 2]
```



① quiz_basic

15 Create a list for the questions
Next you need to add some code to keep the questions in order. You can do this using a list, just like you did for the answer boxes in Step 8. Add this line under the code from Step 14.

```
q5 = ["Where are the pyramids?",
      "India", "Egypt", "Morocco", "Canada", 2]

questions = [q1, q2, q3, q4, q5]
```

This list holds all the questions.

16 Add a function
In a real-life quiz, the quizmaster begins by picking up the first question from the top of a list. In Python, you can do the same thing by using the **pop()** function. This function removes the first item from the list, which makes the second item move to the top of the list. So in your code, **pop()** will remove **q1**, and **q2** will take its place. Type this code next.

```
questions = [q1, q2, q3, q4, q5]
question = questions.pop(0)
```

This gets the first question from the **questions** list and stores it in a variable called **question**.

EXPERT TIPS

Pop the stack

When you place items in a list, Python stacks them on top of each other. The first item in the list appears at the top of the stack. Using the **pop()** function removes an item from the top of the stack.

17 Display the boxes
Now you need to update the **draw()** function to display the questions and the timer on the screen. Use a **for** loop to draw the possible answers inside the answer boxes. Add this code to the **draw()** function under what you typed in Step 9.

```
    screen.draw.filled_rect(main_box, "sky blue")
    screen.draw.filled_rect(timer_box, "sky blue")

    for box in answer_boxes:
        screen.draw.filled_rect(box, "orange")

    screen.draw.textbox(str(time_left), timer_box, color=("black"))
    screen.draw.textbox(question[0], main_box, color=("black"))

    index = 1
    for box in answer_boxes:
        screen.draw.textbox(question[index], box, color=("black"))
        index = index + 1
```

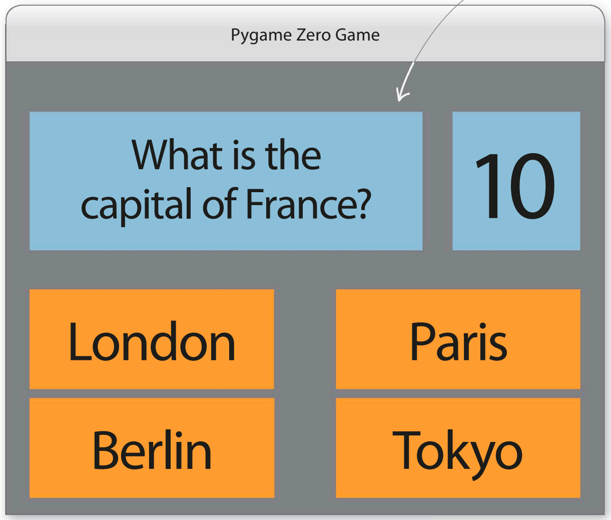
This line displays the number of seconds remaining in the timer box.

This displays the question in the main box.

These lines draw each possible answer in an answer box.

18 Run the code again
Save your code and run it from the command line again. You should see the first question and its four possible answers. At the moment, you can't click on any of the options, and the timer is also fixed at ten. You'll add the code to do these things soon.

The first question is displayed on screen.



What is the capital of France?

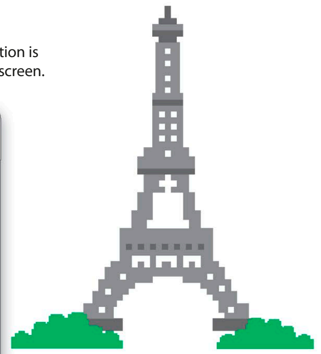
10

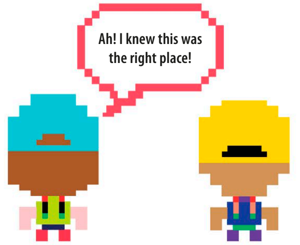
London

Paris

Berlin

Tokyo





19 Set up the final screen
It's time to think about how the game should end. Write some code that displays the final score when the game ends. Replace **pass** under **def game_over()** from Step 3 with this code.

```
def game_over():
    global question, time_left
    message = "Game over. You got %s questions correct" % str(score)
    question = [message, "-", "-", "-", "-", 5]
    time_left = 0
```

This creates a message that will show the player's final score.

Since there's no correct answer here, it's set to **5**, which isn't on the list.

This sets the time to zero when the game ends.

The final message is displayed instead of another question. This will set all possible answers to a dash because you don't want the player to be able to answer.

① quiz_basic

20 Correct answers

Now you need to tell Python what you want the program to do if the player gets an answer correct. You need to increase the current score, and then get the next question. If there aren't any questions left, the game should end. Replace **pass** under **def correct_answer()** from Step 3 with this code.

```
def correct_answer():
    global question, score, time_left

    score = score + 1
    if questions:
        question = questions.pop(0)
        time_left = 10
    else:
        print("End of questions")
        game_over()
```

This increases the score by one.

This gets the next question if there are any-more questions left in the list.

This resets the timer back to ten seconds.

This block runs if there are no more questions in the list.

This displays a message in the Command Prompt or Terminal window.

21 Answering questions

Next add some code that will run when the player clicks on an answer box. This code will check to see which box has been clicked on, and then print the result in the Command Prompt or Terminal window. Replace **pass** under the **on_mouse_down(pos)** function from Step 3 with this code.

This line checks which box has been clicked on.

The variable **index** holds a number that represents the position of the answer box in the list.

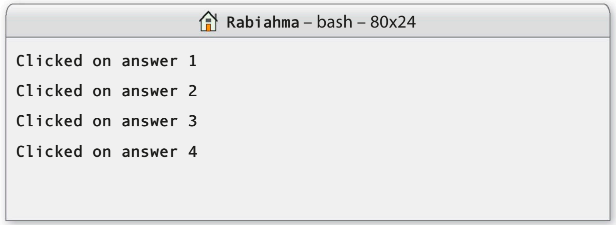
```
def on_mouse_down(pos):
    index = 1
    for box in answer_boxes:
        if box.collidepoint(pos):
            print("Clicked on answer " + str(index))
        index = index + 1
```

The variable **index** increases by one and moves to the next answer box in the list.

This displays a message in the Command Prompt or Terminal window.

22 Click the boxes

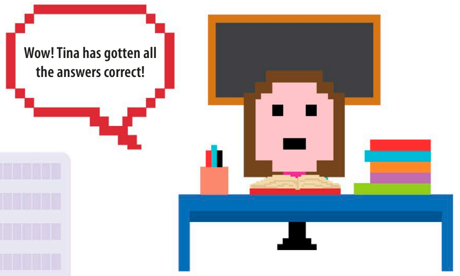
Run your code again and click on each answer box that appears on the screen. You should see messages in the Command Prompt or Terminal window telling you which box you clicked on.


Rabiahma – bash – 80x24

```
Clicked on answer 1
Clicked on answer 2
Clicked on answer 3
Clicked on answer 4
```

23 Check the answer

Now you need to update the body of the **on_mouse_down(pos)** function from Step 21. Add the following code that will run if the player clicks on a box with the correct answer.



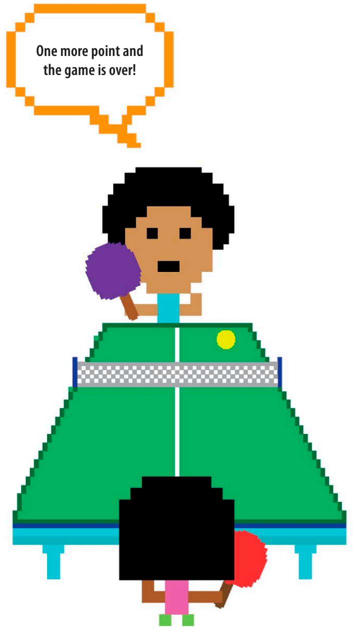
```
def on_mouse_down(pos):
    index = 1
    for box in answer_boxes:
        if box.collidepoint(pos):
            print("Clicked on answer " + str(index))
            if index == question[5]:
                print("You got it correct!")
                correct_answer()
        index = index + 1
```

The item at position five in each question list is the number that corresponds to the correct answer.

This checks if the player has clicked on the correct answer box.

24 End the game

If the player clicks on a wrong answer, the game should end. Update the code under **def on_mouse_down(pos)** one last time and use an **else** statement to run the **game_over()** function if the player selects a wrong answer. Add the code shown in black below.



```
def on_mouse_down(pos):
    index = 1
    for box in answer_boxes:
        if box.collidepoint(pos):
            print("Clicked on answer " + str(index))
            if index == question[5]:
                print("You got it correct!")
                correct_answer()
            else:
                game_over()
        index = index + 1
```

This block runs if the box clicked on doesn't hold the correct answer.

① quiz_basic

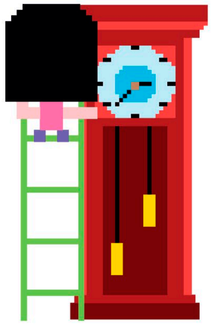
25 Update the timer

Now you need to update the code under **def update_time_left()** from Step 3. This will decrease the number of seconds by one every time the function is run. Type the following code.

```
def update_time_left():
    global time_left

    if time_left:
        time_left = time_left - 1
    else:
        game_over()
```

If there's still time left on the timer, this decreases it by one.

This ends the game when the time runs out.



26 Schedule the timer

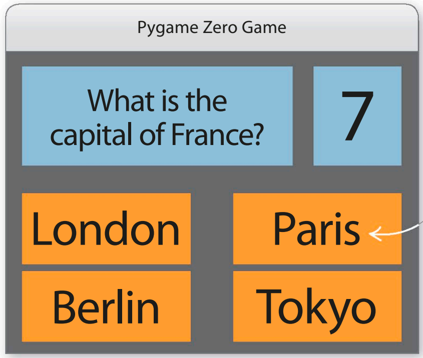
Finally, you need to update the **update_time_left()** function so that it runs automatically once every second. You can use Pygame Zero's clock tool to do this. Add this line of code to the very bottom of your program.

```
    global time_left

    if time_left:
        time_left = time_left - 1
    else:
        game_over()

clock.schedule_interval(update_time_left, 1.0)
```

This calls the **update_time_left()** function once every second.

27 Get quizzing!

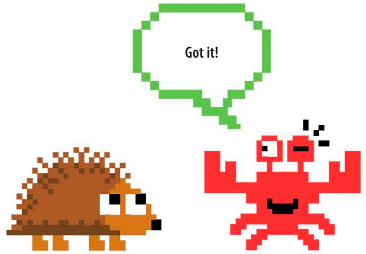
That's it! Run your game and try it out. Hopefully, you'll be able to answer all the questions correctly. Remember, if your screen doesn't look right, you'll have to go back to your code and debug it. Read every line carefully and make sure your code matches the steps exactly. Have fun quizzing your friends!



The player needs to click on the correct answer before the time runs out.

Hacks and tweaks

You've built a great game, but don't limit yourself to just five questions. Take Big Quiz to the next level by tweaking some of its rules. Here are some suggestions to get you started.



▷ **Take the hint**

You can give the player a hint by displaying the correct answer box in the Command Prompt or Terminal window if they press the **H** key. Here is some code you could use to do this.

```
def on_key_up(key):
    if key == keys.H:
        print("The correct answer is box number %s " % question[5])
```

◁ **Skip a question**

You could add some more code to the **on_key_up(key)** function that allows the player to skip a question by pressing the **Space bar**. Skipping a question means they move on to the next question, but without scoring a point. Here's one way of doing it.

```
def on_key_up(key):
    global score
    if key == keys.H:
        print("The correct answer is box number %s " % question[5])
    if key == keys.SPACE:
        score = score - 1
        correct_answer()
```

This block first decreases the player's score by one and then runs the **correct_answer()** function, which increases it by one, keeping the score the same.

▷ **More questions**

You can play this game over and over again, but to keep it interesting, you can change the questions or even add some more. Here are a few examples of questions that you might want to use. You can always add some of your own! If you add extra questions, what else in the code will you need to update?

```
q6 = ["What is a quarter of 200?",
      "50", "100", "25", "150", 1]

q7 = ["Which is the largest state in the USA?",
      "Wyoming", "Alaska", "Florida", "Texas", 2]

q8 = ["How many wives did Henry VIII have?",
      "Eight", "Four", "Six", "One", 3]
```