Chapter 4 Class Exercises

**CE 4.1: Elegant loop solutions**

Go back to the pumpkin you made earlier in Chapter 2 (or use the [example code](https://drive.google.com/file/d/1Aj5VOxnbR_O8MjDfHpzVWHxBzQDY9_pG/view?usp=sharing) we placed online). You/we used to use multiple flip() statements to let the message *Happy halloween!* flash on and off on the screen. Now try to achieve the same goal in a more elegant way using loops. What type of loop did you use and why?

**CE 4.2: Connecting back to PsychoPy Builder**

Think back to the routine and flow interfaces in the [PsychoPy Builder](https://docs.google.com/document/d/1TCojdpHBt02YbNTfnrlZe9TkAMTgirVWaCpgPbkYtBo/edit?usp=sharing) (see Figure 1.2 and 1.9). How do they relate to the loop infrastructure we got to know in Chapter 4?

**CE 4.3: Stroop task - trial loop**

Time to check your progress by advancing our [Stroop task exercise from last week](https://drive.google.com/file/d/1XYPPykz82yQici8dj6sQiEEHqqL6IR-t/view?usp=sharing).

So far, you have been able to display the messages and stimuli on the screen. This is equivalent to the construction of routines and adding in the stimulus components (we'll learn how to add response components in Chapter 5).

Chapter 4 is all about control and you can probably already guess that this refers to the loop infrastructure we saw in the Flow panel of the PsychoPy Builder.

So let’s apply this loop infrastructure to the Stroop task from last week by adding a trial loop in which all of the 16 stimuli are displayed (for half a second to speed things up a bit). Note that they will appear in order, we’ll take care of the random order in a later chapter.

What type of loop did you use and why?

**CE 4.4: Stroop task - feedback**

For now, we can’t register the responses from the participant yet (you’ll learn how to do this in Chapter 5). So let’s assume that the participant always responds with the *d* key for now (i.e., they always answer *red*).

Can you use this information to display a feedback message on the screen on each trial, telling the participant whether they were correct (Correct) or not (Wrong answer!)? Apply it to your code for CE 4.3! Let’s also store that information in the data structure with trial info.

What control structure will you need to use? Where will you insert it?

**CE 4.5: Stroop task - nested loops**

In the PsychoPy Builder we already had some examples of nesting. Can you recall where we applied nesting?

If you are thinking about the block loop and the trial loop of the Stroop task, that is exactly what we will translate to code now.

First, think for a second how you would translate a trial loop nested inside a block loop into code. Maybe it helps to sketch this out on a piece of paper first.

Next, return to your code from CE 4.4 and implement a block loop. Basically, you want to perform 2 blocks of 16 trials each. What type of loop will you use and why? Where do you insert the loop? What needs to happen to the trial loop?

Add a message at the start of each block, announcing the block that is about to start (e.g., Block 1 will start now, Block 2 will start now). Where will you place this message? Use the iteration index of the loop to automatically adapt the message to the block number.

**CE 4.6: Stroop task - reversed instructions**

Can you make sure that participants with an odd number have to respond to the ink color, while the participants with an even number have to respond to the word meaning?

What type of control structure will you need? What will need to be adjusted?

*Hint: did you think about the feedback?*

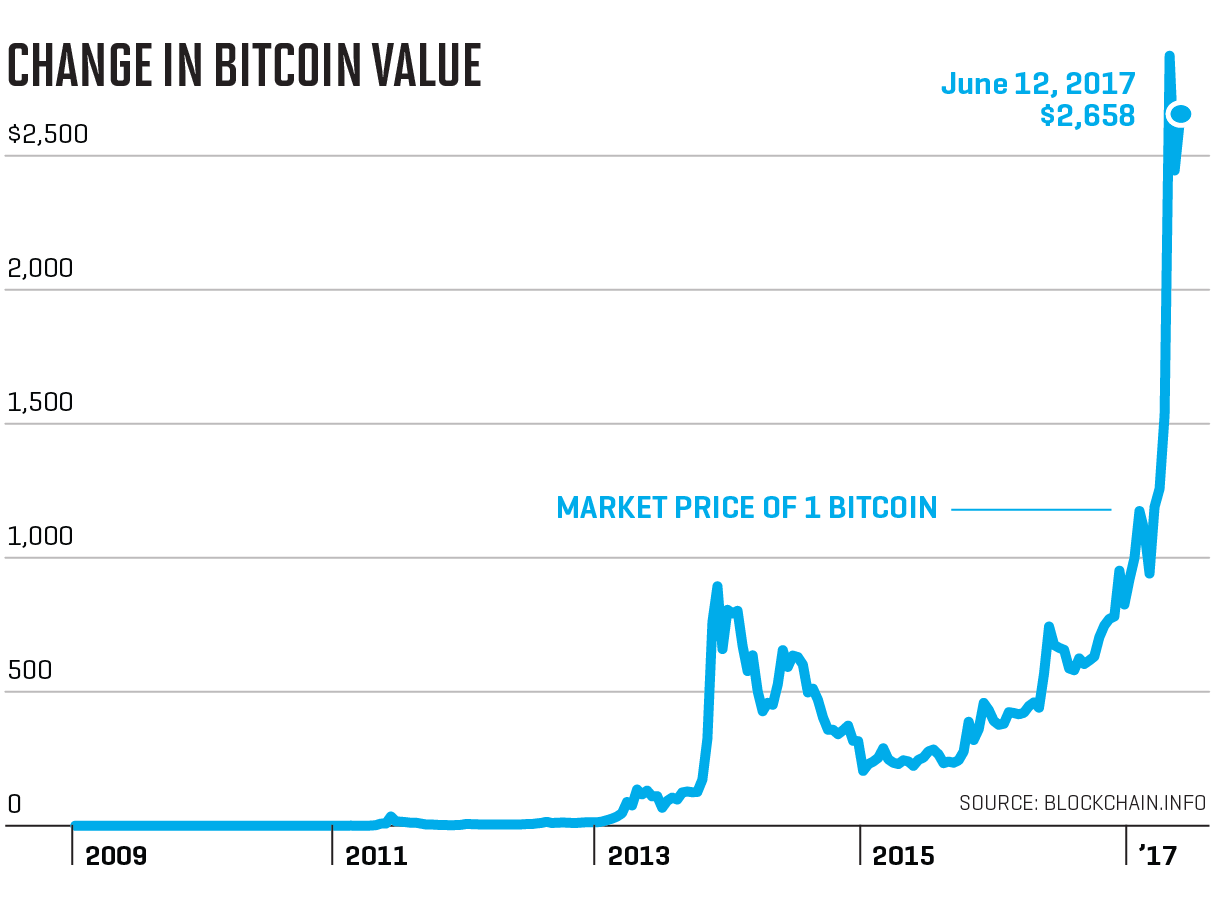
Save this script because we will continue to build on it in the following chapters.

**CE 4.7: Bitcoin investment**

So far we have guided you through exercises with a limited scope (first implement a trial loop, then the feedback message, next the block loop and finally the reversed instructions). In real life and on Test 2 you will get a more complex problem to solve and it will be up to you to disentangle the complex problem into various smaller challenges and determine what control structure you need to tackle those smaller challenges.

We will now practice this skills of recognizing when a specific control structure in needed.

In this exercise you’re going to program the evolutions of the value of a bitcoin investment. Suppose you have bought 1 euro worth of bitcoins. You want to visualise the evolution of the price with a typical stock market graph. Once your investment has increased its value to 100 euro, you stop the simulation.



Assume that the average increase of the bitcoin value is 10%, with a standard deviation of 2.5 (just fictional of course). Sample from this normal distribution (with a mean of 10 and a standard deviation of 2.5) to simulate the evolution of the value of your bitcoin.

*How do you draw from a distribution in Python? Google it!*

*Will you update your value or will you also keep track of the evolution over time? What data type fits best with this scenario?*

On top of the screen, display the updating value of the bitcoin (e.g. “3 euro”) as it increases from 1 to 100 euro. The color of the value should scale along with the value, starting at white when it’s only worth 1 euro and gradually becoming more red until it’s fully red at 100 euro. You might want to use a black background to ensure visibility.

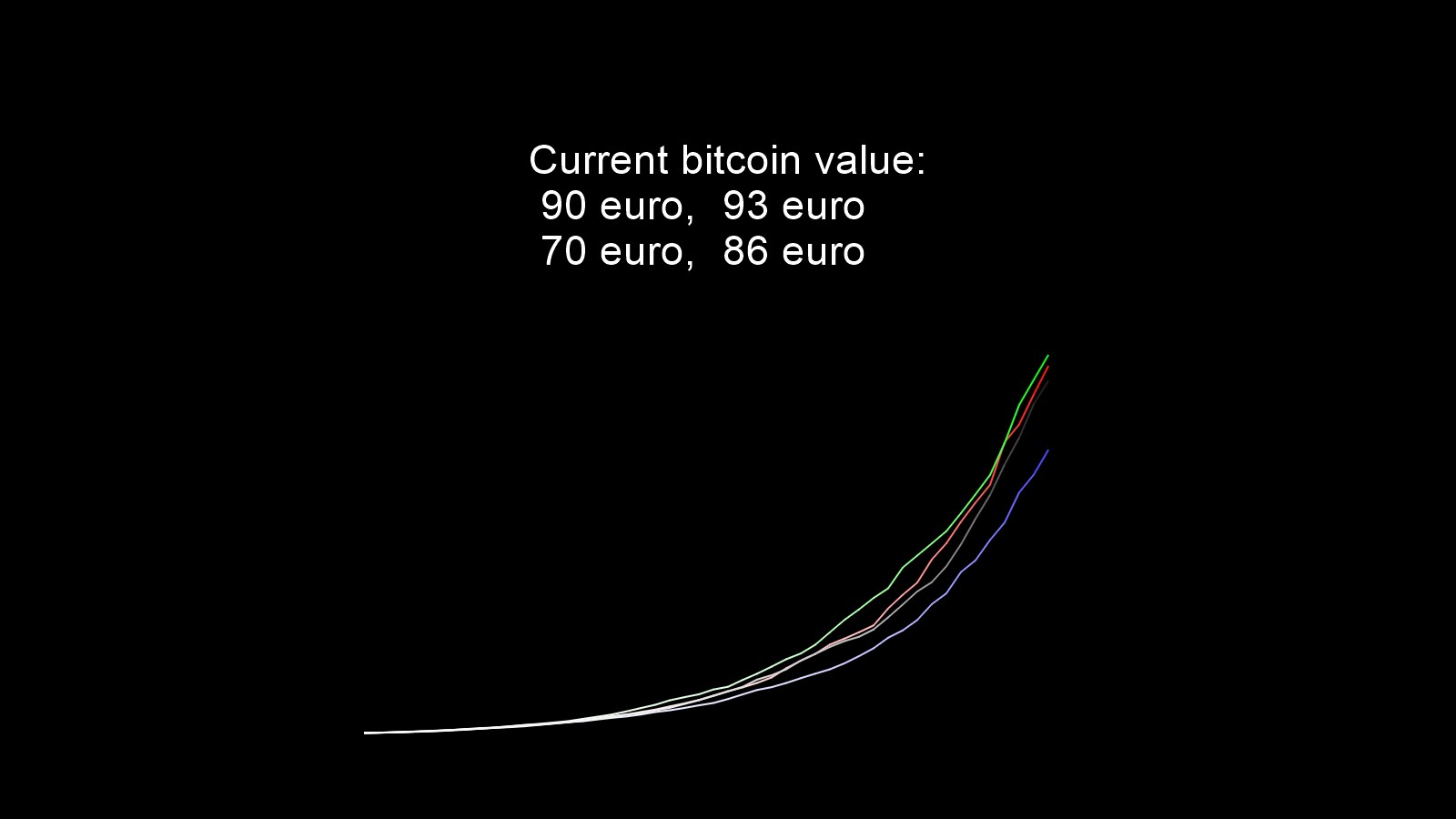
*What are the edge cases that might cause problems with this approach? How can you build in security measures to handle these edge cases?*

Below the value of the bitcoin, plot a line that displays the evolution of the price as it progresses (see example image above).

*Again, what precautions should you take to prevent any problems with edge cases?*

Stop the simulation when you reach 100 euro, at which point you sell your bitcoins and collect your profit.

Extra: now try to run four simulations simultaneously and follow along which simulation hits the target of 100 euro first. You might want four different colors for the four simultaneous simulations.



| Some additional tips Here is an overview of some general tips for the script for Class Exercise 4.7.  It’s usually a good practice to **group code** into the following sections (in order):   * Importing all the modules you need. * Initialize all the variables that you will use in the code (everything that stays constant). * Initialize the data structures you will use to store information (everything that varies). * Initialize/construct all the graphic elements that you will use. * The main loop(s) that display the stimuli on the screen etc.   There are multiple ways to implement this exercise and there is **no objectively optimal script**. Most scripts vary in the following characteristics:   * Some like a bit more white space in between sections to increase readability. * Some have the good habit of adding a clear description to the start of each chunk of code. * Some like to spread out a thought process (complex calculations) across different steps, while some prefer to implement a complex statement in one line.   A bit more difficult is to think about all the possible **edge cases**:   * Will the size of my window be a problem for any screen? If so, can I build in a safety measure to make sure problems are avoided? For example: check whether the dimensions of the window you want to open exceed the actual width and height of the screen of the user (google how you can ask Python to give you that information). Make an if-statement to check whether this criterion is violated and foresee an alternative (e.g., a full screen version of the window). * Will the spatial position of my stimuli possibly exceed the window? If this is possible, can I build in a safety measure to make sure problems are avoided? For example: check whether the maximum or minimum possible value is violated via an if-statement. If the criterion is violated, bring that value back to the maximum or minimum. However, you don’t necessarily need to use an if-statement to do this. Can you find another way (that solves this in one line of code)?   **Avoiding clutter**:   * A variable that will change on each pass through the loop doesn’t necessarily require to be initialized before the loop. Another line of code to get rid of. * Likewise, a constant that does not vary inside the loop should only be initialized once, outside of the loop. Another calculation to get rid of. * You might want to round the value of the bitcoins because you’re probably not that interested in what is happening far beyond the decimal point. * You don’t necessarily need to set any properties for which the default value agrees with your chosen values (e.g., lineColor='white'). * While programming you might want to include print() statements in your loop to check whether everything is going according to plan. Maybe removing these from your code will make it easier to read (or not, it depends).   When you use not so trivial algorithms or solutions to edge cases or complex criteria… Make sure the code stays **readable for others and your future self**:   * Use variable names that clearly define their purpose. * Explain algorithms with a block of comments. * Try to solve problems locally instead of via mechanisms that span over large parts of your code   The hardest part is knowing **what problems to tackle first** and what to do later. To give you an example, I’ve saved my code during four programming steps:   * A first step in which I just implemented the incrementing value by drawing from a normal distribution. Notice that I stick to the output window at this point. * A second step in which I implement the color information and visualise the bitcoin value on the screen instead of in the output window. * A third step will then take care of the increasing line. I’ve also added a more fancy version that adds a color gradient to the line. * The fourth and final step takes a bit of creative thinking with data types to generate a concise script for drawing four simulations at the same time. |
| --- |