Coursework commentary 2017–2018

CO1112 Creative computing 1

Coursework assignment 1

Introduction

This coursework assignment provided an opportunity to practise research and writing in an academic context, an exploration of the use of randomness and colour in digital art, and a chance to write a *Processing* sketch based upon the idea of randomness.

The overall performance on this coursework assignment was generally good, with approximately 70% of submissions being graded as *Excellent*, *Very Good* or *Good* (*i.e.* grades A, B or C). A further 10% were graded as *Acceptable*, but 19% received a *Fail* grade. Those who failed generally did so because they had not completed every part of the assignment.

General comments

Even in cases where you are unable to complete some part of the coursework assignment, marks could have still been gained by discussing what problems you faced and how you tried to overcome them, even if unsuccessful. Some candidates did not submit anything for Part 2 and/or Part 3 of the coursework assignment, and in these cases the examiner did not know whether they had tried and failed to complete the question, or whether they had simply run out of time.

Part 1

For many candidates, this was their first experience of researching and writing an academic essay. The examiners were looking for a coherent flow and well-designed structure to the essay, for knowledge of relevant work with appropriate use of citation and referencing, and for the expression of the candidate's own ideas. Doing all these things well takes practice. Nevertheless, many candidates made a good attempt at this part.

The key to writing a good essay is to read and understand various relevant sources (other papers, books, web sites, *etc*.), and having done that, to decide what *you* think about the topic, and to write about that in your own words. A common weakness in the essays was when they repeated what the candidate had read elsewhere. Another (related) weakness was that many essays read as a random list of facts recounted from a variety of different sources with no coherent flow or structure. To address this, you should think about the main point(s) you want to make in your essay: start by saying what these are, and try to link everything you say together so that the ideas in the essay flow naturally from one idea to the next.

Most candidates made a good attempt at citation and referencing of the sources they discussed. However, a common problem was to include a reference list at the end of the essay, but not to use in-text citations. A citation – *e.g*. (Taylor, 2018) – is used in the main body of the essay at the point where you are talking about a particular source, and refers to a corresponding entry

in the reference list where full publication details of that source are provided. Both citation and referencing are required to let the reader know when you are referring to someone else's work, and exactly what it is.

A related issue is the use of quotations from other sources. If you quote text from another source, you must make it clear what text has been quoted. The usual way to do this is to use 'quotation marks' around the quoted text. Of course, citation and referencing are also required here, to indicate exactly where the quoted text has come from. Try to avoid the over-use of long quotes from other sources, as this can harm the flow of your essay.

Regarding the actual content of the essays, some candidates discussed randomness without ever actually providing a definition or discussion of what it is. Likewise, some drew a distinction between randomness and chance without clearly explaining what the difference is. These are quite general concepts where it might be hard to find a satisfactory definition that everyone would agree with in every context; nevertheless, it is useful to state your preferred definition in cases such as this, so that the reader is sure what you are talking about when *you* discuss the concept.

Some candidates only described the work of Pollock, or LeWitt. While the wording of the question was such that you didn't necessarily have to discuss both of these artists. The examiners were nevertheless expecting that if you chose not to discuss one or both of them, you would discuss a suitable alternative artist(s) in their place.

It was pleasing to see that some candidates made a good effort at structuring their essays, clearly stating in the first paragraph what they intended to discuss, and drawing the essay to a close with a clear conclusion in the final paragraph.

Part 2

The successful completion of this part of the coursework assignment required three tasks to be completed.

The first task was to add comments to the code provided to describe what it does. The majority of candidates did this well. The most common problem was to provide an inappropriate level of detail in the comments – either too little (few or no additional comments) or too much. Examples of the latter category include commenting every single line of code (*e.g.* "`x += 1; // add one to x`"). You should assume that the reader of a comment understands the basic syntax of the language (hence comments like this one are not required). Comments should describe the overall design of the code in terms of how it solves a particular problem or achieves a particular output behaviour.

The second task of Part 2 was to write a short discussion of the way in which randomness is used in the sketch. This was used several times in the sketch, including in both of the source files (Polygon.pde and PolygonPShapeOOP3.pde). A few answers missed some of these cases (a quick search for the keyword "random" would have found them). A small number of candidates did not submit anything in their submitted PDF file for this part.

The final task for Part 2 was to add colour to the sketch, in a way that made use of randomness in some way. In addition to extending the sketch in this way, the question also specified that comments should be added to the new code as appropriate, and that the candidate submit a discussion of what they did in their PDF document. Some candidates chose to generate colours at random; allowing each shape to be a completely random colour was somewhat uninteresting – a better approach adopted by some was to choose one colour at random and then use it as the basis for a colour palette by using some ideas

from colour theory as discussed in the subject guide. Other candidates used a fixed (hand coded) colour palette but introduced randomness in other aspects of the sketch such as how colours were assigned to shapes. Some submissions used the `noise()` function instead of `random()`. The best answers included a clear discussion of what the candidate was trying to achieve and why, preferably with reference to some aspect of colour theory and/or the properties of randomness.

Part 3

The final part of the coursework assignment was generally not done as well as the first two parts. The question focussed on a variety of specific issues that should be discussed in the submitted PDF file, but many candidates omitted some of these discussion points. Successful submissions were those that started with some clearly stated idea relating to randomness, and developed that idea in an interesting way. Many of the poorer submissions had very little to do with randomness, or only used it as a minor aspect of the sketch.

The examiners graded this part according to a number of different criteria, including the quality of the submitted code, the effectiveness of the output, the creativity of the idea, the technical challenge, the discussion of the idea and motivation behind the sketch, the critique of the outcome, and discussion of possible improvements. Candidates who achieved high marks were those who addressed all these aspects to some degree – not necessarily those who submitted the longest or most technically challenging code by itself.

Some candidates submitted very technically impressive sketches, and yet did not score highly because the theme of randomness was unclear. Others made too much use of code obtained from other sources – while it is fine to use someone else's code in your answers (as long as the source is clearly acknowledged), the majority of submitted code in these coursework assignments should be your own.

In a few cases examiners saw multiple candidates submit answers to Part 3 that did very similar things. While it seems that the candidates had coded their own solutions, it was apparent that some level of discussion had occurred between candidates about what to implement for this part. While some level of discussion is fine, there is a fine line to be drawn here. Given that the main thrust of this part was to develop a creative idea based upon randomness, we were looking for candidates to come up with their *own* ideas; an idea that has been implemented very similarly by a group of candidates is inevitably less creative than one that a candidate has come up with as an individual.

Having said that, it was gratifying to see that a variety of very original and impressive sketches were submitted.

Coursework commentary 2017–2018

CO1112 Creative computing 1

Coursework assignment 2

Introduction

This coursework assignment explored the use of L-Systems, genetic algorithms and object-oriented programming in *Processing*.

The overall performance on this coursework assignment was generally very good, with approximately 75% of submissions being graded as *Excellent*, *Very Good* or *Good* (*i.e.* grades A, B or C), with a fairly even split between these three categories. A further 10% were graded as *Acceptable*, but 15% received a *Fail* grade. Those who failed generally did so because they had not completed every part of the assignment.

General comments

Even in cases where you are unable to complete some part of the coursework assignment, marks can still be gained by discussing what problems you faced and how you tried to overcome them, even if unsuccessful. Some candidates did not include such a discussion in places where they couldn't complete part of the assignment, and in these cases the examiners did not know whether they had tried and failed to complete the question, or whether they had simply run out of time.

In terms of the quality of the submitted code, although some candidates submitted very neat, well designed and well commented code, in many of the submissions there was plenty of room for improvement in coding standards. In addition to a lack of useful comments, there were many examples of code with inconsistent indenting and spacing. People new to programming may regard the neatness of their code as quite inconsequential, but as you get more experienced with writing code – and, in particular, with *reading* code (possibly written by other people) – you realise that having consistent standards of code presentation greatly helps the reader in understanding how the code is structured. In Processing in particular, there is little excuse for producing poorly formatted code, because there is an *Auto Format* function in the *Edit* menu, which will tidy up your code for you!

Part 1

The first part of the coursework assignment was straightforward and most candidates achieved high marks. For full marks the examiners were expecting to see a diagram of the output pattern specifying the lengths of the lines drawn and the angles between lines. The question asked the candidate to "explain the meaning" of each of the symbols; some candidates lost marks for merely stating the specific *Processing* code that was executed for a symbol. For example, for the "[" symbol we were looking for an answer such as "*saves the current position and orientation of the drawing point"* rather than "*pushMatrix()*".

Part 2

The first two subsections of Part 2 were done well by most candidates. However, some candidates had problems with the second subsection, which involved generating a string of length between 1 and 8 characters. These candidates used the following code to determine the length of the string to be produced:

```
int(random(1,8))
```

While this might look sensible, it is in fact incorrect for this purpose: `random(1,8)` will return a float between 1.0 and up to (*but not including*) 8.0, *i.e.* it will return values in the range 1.0 to

7.9999... When 7.9999 is converted to an integer, the end result is 7, because the `int()` function in *Processing* always rounds down. So the code shown above will only return integers between 1 and 7 inclusive. To return numbers between 1 and 8 inclusive, you should use:

```
int(random(1,9))
```

This subtle problem highlight the importance of proper testing of your code to ensure that it is behaving as expected.

The third subsection asked about any potential problems with the strings generated with this method. One of the most obvious problems is that the string might not contain any F or H characters; these are the only two characters that actually result in a line being drawn on the screen, so strings that do not contain a single instance of either of these characters will not result in anything being drawn on the screen. While the majority of candidates spotted this problem, a significant minority did not.

The final subsection of Part 2 involved adding the "[" and "]" characters to the alphabet of allowable elements in the string. As noted in the question, this is not straightforward, because care must be taken to ensure that each "[" is followed by a matching "]", and a "]" cannot appear before a corresponding "[". To solve this problem in its most general form, allowing multiple sets of matching brackets which may or may not be nested, is a significantly harder challenge than the preceding subsections. Few candidates submitted a completely general solution, but most submitted partial solutions that still gained decent marks. The most trivial partial solution was just to allow a single "[" in the string followed by a single "]". Better solutions allowed for multiple sets of brackets. The most common technique in coding these solutions involved implementing a counter to keep track of the number of currently open brackets requiring the addition of an additional close bracket.

One problem with the final subsection that was spotted by a few candidates was that it was possible to generate a bracket pair without any characters in between them, "[]", which would not have any effect on the string's output. Some of the best responses to this part included code that would check for this and only generate bracket pairs that contained printable characters in between them.

Part 3

Most candidates managed to produce 12 images in their submissions. However, in many submissions there was a disappointing lack of variety in the 12 images displayed, suggesting that the candidate had spent little time investigating the capabilities of the system. The code provided is actually capable of generating a surprising variety of patterns.

While some candidates successfully managed to give each saved image file a unique name, this was a more complicated task than it might first

seem. Some candidates appended a random number or a timestamp to the filename. However, the most common approach was to add a loop in the question3setup() method to run the code 12 times, and use the loop index in the filename so that each filename was unique (*e.g.* "image11.png"). This might seem like a straightforward approach, but various issues arise from it. The loop would have to be placed in the question3setup() method, because if it was in the draw() method it would draw all 12 trees on top of each other. But the save() method to save the image to file has to be called from the draw() method, after the drawing has actually occurred. The problem is how to allow the draw() method to access the loop index as set up in the question3setup() method, so that the filename can be set appropriately within the draw() method. The easiest solution – which many candidates implemented – is to create a global variable at the start of the sketch which will be used as the loop index in question3setup() but will also be accessible in draw().

It was disappointing to see that many candidates lost out on marks in this part by not providing any discussion of their observations about the output, as requested in the question.

Part 4

Responses to this part were mixed. While some were done very well, many candidates gave few specific details of how they might implement a suitable mutate() method. Many candidates discussed the possibility of mutating individual characters in the string, but other possibilities that were not discussed in many submissions include the ability to insert new characters into the string and the delete existing characters. These insertion and deletion operators will allow the length of the string to change over time – this is not only permissible but also desirable, to allow the user to explore more complicated patterns and the system evolves.

Part 5

Of the candidates who completed Part 5, most submissions were competent and achieved average marks, although very few were outstanding. One of the most common weaknesses was a failure to clearly state an aim of what the candidate was trying to achieve, and (just as importantly), *why* they were trying to achieve it.

When working on creative artefacts like this, it is advisable to start off with an idea that you would like to express - the sketch should be an expression of your idea. You should discuss the idea, and how you have chosen to express it, in your submitted discussion. Many submissions were technically proficient, but lacked an interesting idea. Quite a few submissions were based on very similar ideas – it was apparent that these candidates had discussed the work together before completing the coursework assignment. While this was not regarded as copying as such (each candidate had produced their own code), the end results were rather unexciting as they were all based upon an agreed idea arrived at "by committee" rather than a unique idea developed by each individual candidate.

A surprising number of candidates lost out on marks by not including the requested evaluation of what they thought are the strengths and weaknesses of what they had created.

Some candidates used code they had obtained from elsewhere as part of their submitted work. It is fine (and quite normal) to use someone else's code as a starting point, but if you do so, you *must* clearly state where the code has come from (*e.g.* in a comment at the start of the code, and/or in your submitted discussion). This is very important – you must make it very

clear to the examiners which parts you have written yourself and which have come from elsewhere. Failure to do so, and to properly give details of where the code has come from, runs the risk of being considered plagiarism. The examiners regard plagiarism in code just as seriously as plagiarism in written coursework assignments, so be sure to acknowledge your sources properly.

Overall, this coursework assignment was generally done well. The main sources of weakness were in failing to provide answers for everything requested in each part of the question, and in not attempting all parts of the question.