Instructions:

- Write the quiz solution on blank paper, numbering each of the questions and ideally answering the questions in order. Make sure your name and student number are clearly written at the top of the paper. If you prefer an electronic writing pad to blank to paper, that is fine too.
- When finished with the quiz. Take photos/scans (using your phone is fine) and format the photos in a single document (e.g. paste in a Word document, or anything else). Then create a PDF file not bigger than 10MB and upload to BB using the quiz submission link.
- As with all MATH2504 assessment, create a short voice recording indicating the work is your own and stating that you followed all of the guidelines (if this is the case). Upload the voice recording as an additional file.
- The duration of the quiz is 50 minutes + 10 minutes reading time + 20 more for formatting the solution and creating the voice recording. Thus 80 minutes in total.
- Join the course Zoom link at the start of the quiz (actual quiz is Aug 29, 6:00pm BNE time). A link to the actual quiz will be provided. You can then ask questions (only during) the first 10 minutes (reading time) via private chat on Zoom. It is recommended you stay on Zoom for the duration of the quiz (until final upload at 7:20pm). This is in case there are announcements or comments. There is no need for you to activate your camera or mic. If you are allowed extra time due to special circumstances - communicate this to the lecturer via private zoom chat at the start of the quiz.
- You are not allowed to communicate with **anyone** during the quiz. The only communication allowed is asking questions to the instructor via Zoom private chat during the first 10 minutes.
- You are not allowed to run Julia or any other computational software during the quiz. You may use a hand calculator (or calculator on your phone), but not more.
- You are not allowed to use any material except for the course material directly from the course website (Units 1–3, Practicals A–D, BigHW questions, previous quizzes and their solutions). That is, feel free to use your web-browser to look at the course materials during the practice quiz, however you are not allowed to use **any** other written material or any other material from the web.
- In your voice recording, if it is the case, you should clearly state that you didn't communicate with anyone, that you didn't run Julia or any other software, and that you didn't use any other material except for (perhaps) material from the course website.
- In case of exceptional circumstances (mishap with upload etc...), write the course coordinator. Otherwise, late submissions will not be accepted.

There are 9 items in total with each item worth 12 points.
This yields a total of 108 points total. The maximal grade is 100.

Quiz questions on next page...

Question 1:

Consider the following function designed to return `true` if the input array, `arr` is in "ascending order". That is, for the function to return `true` each element of the array (from the second element onwards) should be equal or greater to the previous element. Otherwise the return value should be `false`.

```
1:  function is_ascending_sorted(arr; by_func = (x)->x)
2:     n = length(arr)
3:     n == 1 && return true
4:     for i in 2:(n - 1)
5:        if by_func(arr[i]) > by_func(arr[i+1])
6:           return false
7:        end
8:     end
9:     return true
10: end
```

For example `is_ascending_sorted([1,10,100,105,105])` should return `true`, and similarly, `is_ascending_sorted([1,10,106,105,105])` should return `false`.

1a: There is a bug in the function which can be fixed by replacing a single character in the code. Show example input for which the bug exists and specify the fix (which character to replace and to what value).

1b: Assume now that with the fixed function (your answer from 1a above) you invoke:

```
is_ascending_sorted(a, by_func = g)
```

where `a` is an array of integer values and `g` is a Julia function that implements a mathematical function that is monotonic decreasing such as for example $g(u) = -u^3$.

What would the function return for the input array `a=[10,5,3]`? Explain your answer.

1c: You now have a recursive variation of the function called `is_ascending_sorted_rec` which is required to operate in the same way and determine if the input is in "ascending order" (note that this version does not have `by_func`):

```
1: function is_ascending_sorted_rec(arr)
2:    length(arr) == 1 && return true
3:    (arr[1] <=  arr[2]) || is_ascending_sorted_rec(arr[2:end])
4: end
```

In its current form, `is_ascending_sorted_rec` does not work. Suggest a minor change to line 3 which will fix the problem.

Question 2:

Assume that you have a function called `fast_sum_1000` which when presented with an input array of `Float64` values with length exactly 1000, produces the sum of the numbers very quickly. There is also the in-built `sum` function which can sum values, but it is not as quick. We wish to create the function `fast_sum` which sums up an input array of any length. Here it is (with `__2a__` and `__2b__`) indicating expressions that you need to fill in:

```
1: function fast_sum(arr)
2:     d = div(length(arr), 1000)
3:     result = 0.0
4:     for i in 1:d
5:         result += fast_sum_1000(arr[__2a__])
6:     end
7:     return result + sum(arr[__2b__])
8: end
```

Note that the `div` function returns the result of integer division. For example `div(10,4)` is 2.

2a: Determine an expression which should go in place of `__2a__` for `fast_sum` to work correctly. The expression can use the loop variable `i`.

2b: Similarly, determine the expression which should go in place of `__2b__`.

2c: Based on your answers above, would the function `fast_sum` still work correctly if `arr` has less than 1000 entries? If so, explain why? If not, describe how to modify the function to fix the problem.

Question 3:

Consider the following function:

```
1: function has_bit_pattern(bits::UInt64, pattern::UInt8)
2:    for i in 0:56
3:       if UInt8((bits >> i) & 0xFF) == pattern
4:          return true
5:       end
6:    end
7:    return false
8: end
```

The function determines if the 8 bits sequence, `pattern`, appears anywhere within the 64 bit sequence `bits`. The code for the function is correct (i.e. the function does what we specify).

3a: What would be the output of `has_bit_pattern(UInt64(0x0280), UInt8(0xA0))`? Briefly explain your answer.

Note that `UInt64()` and `UInt8()` simply ensure that input values are of the types of 64 bits and 8 bits unsigned integers respectively. Recall also that in Julia, expressions such as `0xA0` represent constant numbers (unsigned integers) in hexadecimal form.

3b: We are now interested to enhance the function with a third argument, `num_bits`. This argument determines how many bits out of `pattern` are used to search within `bits`. For example if `pattern` is 01001010 and `num_bits` is 5, then only the 5 right most bits are used, namely: 01010. Also note that the implementation above (without `num_bits` as an argument) is a special case with `num_bits` set to 8.

The new function signature is now

```
has_bit_pattern(bits::UInt64, pattern::UInt8, num_bits)
```

Without seeing the function implementation, assume that now we execute

```
has_bit_pattern(rand(UInt64), UInt8(0b1), 1)
```

What is the probability of the return value being `false`?

Note that as you might expect `rand(UInt64)` returns a uniformly random `UInt64`. Also recall that `0b1` is a binary representation of the number.

3c: Now determine how to implement this feature. For this suggest modifications to lines 2 and 3 in the code above, assuming `num_bits` is available as an input argument and is an integer in the range 0 to 8.