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.
- 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 30, 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 turn on your camera or mic.
- 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). 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.

Each item is worth 15 points.
There are 105 total points total. The maximal grade is 100.

Quiz questions on next page...

Question 1:

A palindrome is a string that equals its reverse. Here are examples of palindromes: `ABBA`, `ANNA`, `KAYAK`, `ROTATOR`, or `STEP ON NO PETS`. As another example, `DOG` is not a palindrome. The empty string is also a palindrome and so is any string with a single character.

For the purpose of this question, let us assume that all strings have upper case letters or other characters, but no lower case letters.

The following one line function, `is_palindrome()`, returns `true` if the input is a palindrome, and `false` otherwise.

```
is_palindrome(str::String) = str == reverse(str)
```

Here is an attempt for an implementation that does not rely on the `reverse()` function:

```
1:   function is_palindrome(str::String)
2:       i_b, i_e = 1, length(str)
3:       while i_b < i_e
4:           if str[i_b] == str[i_e]
5:               return true
6:           end
7:           i_b += 1
8:           i_e -= 1
9:       end
10:     return false
11: end
```

1a: There are mistakes in the implementation above and it does not work. Suggest what code to replace in lines 4, 5, and 10 to fix the function.

1b: Here is an attempt at recursive version of the function:

```
1: function is_palindrome(str::String)
2:     (length(str) == 0 || length(str) == 1) && return true
3:     str[1] == str[end] && return is_palindrome(str[1:end-1])
4:     return false
5: end
```

Does this recursive version correctly determine if a string is a palindrome or not? If not, suggest a simple fix in a single line of the code which will make it correct. If yes, argue in a few sentences why it is correct.

Note that in Julia, indexing into an array or string, `arr`, with `arr[i:j]` and with $j < i$ returns an empty array or string.

Question 2:

The following function computes the factorial of a non-negative integer `n`.

```
1: function my_fact(n::Int)
2:     n < 0 && throw(ArgumentError("n must be non-negative"))
3:     n == 0 && return 1
4:     return n*my_fact(n-1)
5: end
```

2a: Say you evaluate `my_fact(3)`. How many times is the multiplication in line 4 carried out?

2b: Recall now that the mathematical gamma function, denoted $\Gamma(z)$. In general z may be complex valued, but for the purpose of this question we focus on positive real valued z. It is known that the gamma function satisfies $\Gamma(z+1) = z\,\Gamma(z)$. For example $\Gamma(2.7) = 1.7 \times \Gamma(1.7)$.

Assume you are given a Julia function `gamma_in_01()` which when given an input argument `z` in the range $(0, 1]$ evaluates the gamma function at `z` (you do not have the source code for this function but you can use it and you know it is working correctly). Consider now the Julia function (with missing code),

```
1: function my_gamma(z::Float64)
2:     z <= 0 && throw(ArgumentError("z must be positive"))
3:     z <= 1 && return gamma_in_01(z)
4:     return ____________________
5: end
```

Fill in code for this missing expression in line 4 so that `my_gamma()` implements the mathematical gamma function.

2c: It is also possible to create a non-recursive function `my_gamma_non_recursive()` with output identical to `my_gamma()`. Here it is (with missing code):

```
1: function my_gamma_non_recursive(z::Float64)
2:     z <= 0 && throw(ArgumentError("z must be positive"))
3:     g = gamma_in_01(z - floor(z))
4:     while z > 1
5:         g *= z-1
6:         ---------------
7:     end
8:     return g
9: end
```

Fill in the code for the missing statement in line 6.

Note that the Julia function `floor` computes the floor of number so for example `floor(1.7)` returns `1.0`.

Question 3:

Consider a data format called *Binary Coded Decimals* (BCD) for numbers such as $d_4d_3d_2d_1$ where d_i are decimal digits. This can encode integers in the range $\{0, 1, \ldots, 10^4 - 1\}$ where for example, the number 975 has $d_4 = 0, d_3 = 9, d_2 = 7, d_1 = 5$.

BCD allocates 4 bits per decimal digits d_i, such that $d_4d_3d_2d_1$ uses two bytes in total. Each decimal digit occupies 4 adjacent bits. Within its 4 bits, each digit is coded as a binary number. So for example the number 975 is coded as follows:

$$\underbrace{0000}_{d_4=0}\underbrace{1001}_{d_3=9}\ \underbrace{0111}_{d_2=7}\underbrace{0101}_{d_1=5}$$

Not all possible 16 bit combinations are allowed for BCD. As one example, the 16 bit value `0xffff` (all bits are 1), is illegal. There are many other illegal cases.

The following function converts a BCD encoded number to an integer (in the standard way integers are represented in Julia):

```
1:  function bcd2int(bcd::UInt16)
2:      x_value = 0
3:      pow = 1
4:      for _ in 1:4
5:          d = bcd & 0x000f
6:          false && throw(ArgumentError("Not a valid BCD"))
7:          x_value += pow*d
8:          bcd >>= 4
9:          pow *= 10
10:     end
11:     return x_value
12: end
```

3a: At the moment, the function works correctly for valid BCD inputs, yet does not throw an error for invalid inputs. For example the input `0x3BF2` is an illegal BCD entry, yet `bcd2int()` would return a numerical value which is wrong. How would you modify the `false` in line 6 so that the function throws an error for such invalid inputs?

3b: After your modification you execute the following:

```
1: candiate_bcd = rand(UInt16) #This generates some uniformly random 16 bits
2: bcd2int(candiate_bcd)
```

What is the probability that an error is thrown?