# IT5001 Practical Exam

## Instructions:

- If you failed to keep your proctoring camera, your PE will result in **ZERO mark** no matter what you submitted onto Coursemology.
- You ***cannot import*** any additional packages or functions, or you will get zero mark.
- This PE consists of **THREE** parts and each part should be submitted separately in Coursemology. It's better for you to organize your files into `part1.py`, `part2.py` and `part3.py` and each file is independent, e.g. you cannot expect that the answer in `part2.py` calls a function in `part1.py` that is not in `part2.py`. If you want to reuse some function(s) from another file/part, you need to copy them (as well as their supporting functions) into that file/part. For example, you just need to copy the function you want to call in `part1.py` into `part2.py`.
- You are allowed to write extra functions to structure your code better. However, remember to submit together mentioned in the point above.
- No mark will be given *if your code cannot run*, namely any syntax errors or crashes.
  - We will just grade what you have submitted and we will **not** fix your code.
  - Comment out any part that you do not want.
- Workable code that can produce correct answers in the given test cases will only give you *partial* marks. Only good and efficient code that can pass ALL test cases will give you full marks. Marks will be deducted if your code is unnecessarily long, hard-coded, or in poor programming style, including irrelevant code or test code.
- You must use the same function names as those in the template given.
- Your code should be efficient and able to complete *each* example function call in this paper within 2 seconds.
- In all parts, you should **return** values instead of **printing** your output. In another word, you should not need any "`print()`" in this entire PE for submission.
- You should either delete all test cases before submission or comment them out by adding # before the test cases. If you submit more code than required, it will result in **penalty** because your code is "unnecessarily long".
- Reminder: Any type of plagiarism like copying code with modifications will be caught, that include adding comments, changing variable names, changing the order of the lines/functions, adding useless statements, etc.

## Constraints for Part 1

You cannot use the following Python ***built-it*** functions/methods in Part 1:

- `encode, decode, replace, partition, rpartition, split, rsplit, translate, map, filter, join.`

Also, your code cannot be too long. Namely, the function in *each* task must be fewer than 400 characters and 15 lines of code including all sub-functions that called by it.

You cannot use any list, tuple, dict or set and their functions in this part.

## Part 1 DNA Comparison (15 + 15 = 30 marks)

DNA can be represented by a string with the four letters ‘A’, ‘G’, ‘C’ and ‘T’. We can compare the DNAs of two living organisms by locating the difference in their DNAs. Here are two DNA examples and their difference

```
      DNA1: 'AGTCATATAC'

      DNA2: 'ACTCATGTAA'

Difference: '*.****.**.'
```

The difference string is the comparison of the two DNAs. If the two letters in the same position are the same, it will be a `'*'` and `'.'` otherwise.

### Task 1 Iterative DNA Comparison (15 marks)

Write an ***iterative*** version of the `compareDNA_I(dna1,dna2)` to compare two strings of DNAs. You can assume that they are in the same length and with the four letters only. In this task, you cannot use any recursion. Here is some sample output.

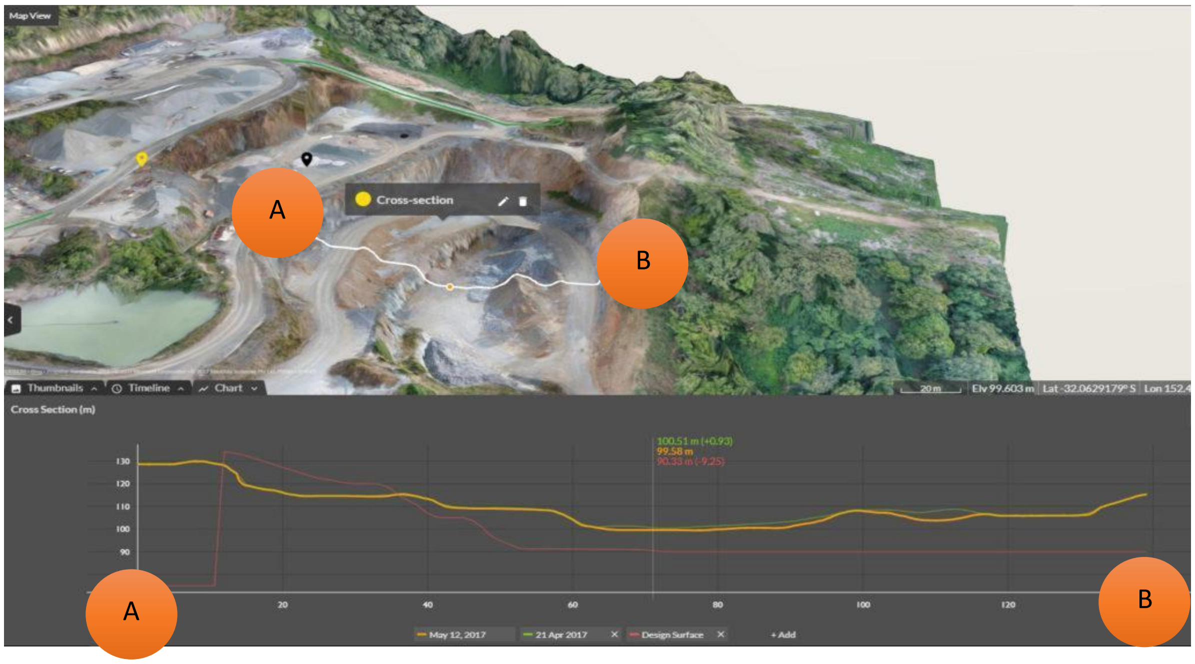
```
>>> print(compareDNA_I('AGTCATATAC','ACTCATGTAA'))
*.****.**.
>>> print(compareDNA_I('AATAC','GAGAA'))
.*.*.
```

### Task 2 Recursive Run-length Decoding (15 marks)

Write a ***recursion*** version of the function `compareDNA_R(dna1,dna2)` with the same functionality in Part 1 Task 1. However, you cannot use any loops or list comprehension in this task.

## Part 2 Local Peaks Searching (15+15+10 = 35 marks)

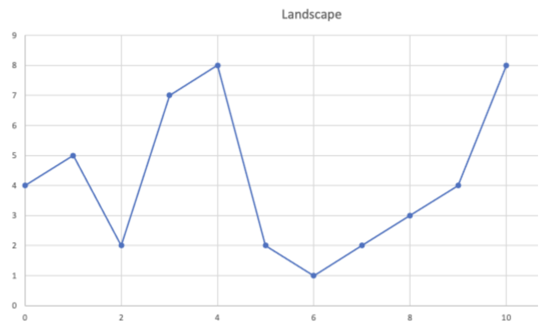
We want to use drones to make maps. We want to do a cross section survey by sending a drone "walking" from Point A to Point B in a straight line, and record its ups and downs every 1 m.



The drone flies in a straight line and it can survey the height of the land surface for every 1m. For example, the data (in meters) sent by the drone is a list of size N:

```
[4,5,2,7,8,2,1,2,3,4,8]
```

You can visualize the landscape as



A local maximum is the point of the landscape that its neighbors are strictly lower than it. For the above landscape, the local maximums are Positions 1, 4 and 10. Note that

- We can have more than 1 local maximum, and you just need to find one
- Positions 10 is deemed as a local maximum because we treat Position -1 and Position N (before and after the array) are negative infinity.

Your job in this part is to return the **position** of ***any*** local maximum. For example, for the given input landscape given above, either 1, 4, or 10 all can be correct answers, and you only need to return either one of them.

### Task 1 Array Version (15 marks)

Given the input as the drone survey data, write a function `local_peak_array_version(survey)` to take in a list of integers survey and output any of the local maximum. Here is an example below, but the answer is also correct if it your function returns 4 or 10.

```
>>> print(local_peak_array_version([4,5,2,7,8,2,1,2,3,4,8]))
1
```

We can assume the data list sent by the drone is

- All positive integers
- The size of the data list = N > 1
- No consecutive two numbers in the survey is the same

## Task 2 Function Version (15 marks)

Instead of using a list, we use a function as an input to the local peak searching. For example, we can have a **survey function** to return the height at position i, e.g. `survey0(i)` that returns `4` when input `i=0` similar to the above array.

```
>>> survey0(0)
4
>>> survey0(3)
7
>>> survey0(6)
1
```

And because we won't know what is the range for the survey function, we have two more input parameters a and b. It means that the survey function takes the range from a to b inclusively. You can assume the function given will return integer values within the range from a to b.

```
>>> survey1 = lambda i: [4,5,2,7,8,2,1,2,3,4,2][i]
>>> print(local_peak_function_version(0,10,survey1))
1
>>> survey3 = lambda i: [1,2,3,4,5,6,5,4,3,2][i%10]
>>> print(local_peak_function_version(1,9,survey3))
5
>>> print(local_peak_function_version(11,19,survey3))
15
```

Your task is, to write the function `local_peak_function_version(a,b,survey)` to find any peak from the survey function between the range of a and b, inclusively.

## Task 3 Big Function Version (10 marks)

You will get the marks for this section if your function in Task 2 can handle very large area of survey.

```
>>> survey2 = lambda i: i if i <= 1234567890 else (1234567890*2) - i - 1
>>> print(local_peak_function_version(0,12345678900,survey2))
1234567890
```

## Part 3 Fibonacci Password (10+20= 30 marks)

In order to protect the data of the Fibonacci Bank, the bank has its password for its main database in the following manner. The password only consists of two alphabets ‘F’ and ‘B’. And the password will change every day since the first day of the bank.

- The first day, the password is ‘F’
- The second day, the password is ‘B’
- The third day, the password is ‘FB’, which is the concatenation of the password of the previous two days
- The fourth day, the password is ‘BFB’, which is the concatenation of the second and third day passwords
- The fifth day, the password is ‘FBBFB’, which is the concatenation of the third and fourth day passwords
- … etc.

Here are the first 9 passwords:

| N | The Nth Day Passwords |
|---|---|
| 1 | F |
| 2 | B |
| 3 | FB |
| 4 | BFB |
| 5 | FBBFB |
| 6 | B**F**BFBBFB |
| 7 | FBBFBBFBFBBFB |
| 8 | BF**B**FBBFBFBBFBBFBFBBFB |
| 9 | FBBFBBFBFBBFBBFBFBBFBFBBFBBFBFBBFB |

(The underlined letters are for the last task for this part.)

### Task 1 The nth Password (10 marks)

Write the function `nth_day_pw (N)` to return the Nth day password as a string. You can assume that the input N is always an integer and greater than zero. You will only get a full mark if you can handle N >= 40 within 1 second.

```
>>> print(nth_day_pw(10))
BFBFBBFBFBBFBBFBFBBFBFBBFBBFBFBBFBBFBFBBFBFBBFBBFBFBBFB
>>> print(nth_day_pw(40)[-20:])
FBFBBFBFBBFBBFBFBBFB
```

## Task 2 The kth Letter of the nth Day Password(20 marks)

Somehow, a less important part of the system of the Fibonacci Bank needs a weaker password. In fact it only needs the kth Letter of the nth day. For example, the second letter of the 6th day password is ‘F’ and the third letter of the 8th day password is ‘B’. Your job is, write a function `kth_letter_nth_day_pw(k,n)` to return the kth letter of the nth day password.

```
>>> print(kth_letter_nth_day_pw(2,6))
F
>>> print(kth_letter_nth_day_pw(3,8))
B
>>> print(kth_letter_nth_day_pw(123456789,99999))
B
>>> print(kth_letter_nth_day_pw(123456789,9875))
B
```

You will only get full marks if your code can handle some very big number of N.

- If you can handle n < 30, you will get 5 marks
- If you can handle 30 < n < 50, you will get another 5 marks
- If you can handle n > 100, you will get the last 10 marks.

--- END OF PAPER ---