# Project 1
# COMP301 Fall 2020
**Deadline: Nov 1, 2020 - 23:59 (GMT+3)**

In this project, you will work in groups of two. To create your group, use the Google Sheet file in the following link:

*Link to Google Sheets for Choosing Group Members*

Please use the code boilerplate and submit your Racket file to BlackBoard. Name your submission file as *p1_yourIDno_username.rkt* (scm extension is also fine).

Example: *p1_0011221_asabuncuoglu13.rkt.*

The deadline for this project is Nov 1, 2020, 23:59 (GMT+3 : Istanbul Time). **Read your task requirements carefully. Good luck!**

**Problem Definition:** To represent a certain set of quantities in a particular way, we are defining a new data type. In PS1, you created two different representations (unary and bignum) of natural numbers.

Another example is representing all the integers (negative and non-negative) as diff-trees, where a diff-tree is a list defined by the grammar as the book's Exercise 2.3 (page 34) which is defined with the following grammar.

```
Diff-tree ::= (one) | (diff Diff-tree Diff-tree)
```

These examples show how data abstraction can be managed with different interfaces and implementations. In this project, you will create a new data-type to represent a quantity of your choice. You can select any quantity to represent such as Natural Numbers, Rational Numbers or Cities on a Map...

**Part A.** Similar to how natural numbers are represented in Unary and BigNum Representations, you will define two new types to represent your selected quantity. In this part, define the grammar definition of your data-types.

**Part B.** Implement these representations in Scheme. For each of these representations, implement the following procedures below:

- `create`: gets an input and creates the new data-type.
- `is-empty?`: returns `#t` if the representation has no value, otherwise returns `#f`.
- `successor`: gets a representation a small building block of your data-type and adds it to defined data-type.

**Part C.** Please explain what are Constructors, Observers, Extractors and Predicates. For each procedure explained in Part B, please indicate if they are Constructors, Observers, Extractors or Predicates.

**Part D.** Create 3 new test cases for the following procedures of both representations.

- `create`: Write one case.
- `is-empty?`: Write two test cases for returning one true and one false.
- `successor`: Write one case.