# 2WF90 Algebra for Security

Software Assignment 1:
Integer and Modular Arithmetic

2022 – 2023

Andreas Hülsing
Responsible Lecturer
a.t.huelsing@tue.nl

Benne de Weger
Lecturer
b.m.m.d.weger@tue.nl

Matthias Meijers
Teaching Assistant
m.c.f.h.p.meijers@tue.nl

July 27, 2023

# Table of Contents

# 1 Introduction

This document describes the first software assignment of the course. In this assignment, you are tasked with constructing software that can perform certain integer and modular arithmetic operations on "arbitrarily" large integers in different radices. Naturally, you are also expected to provide corresponding documentation. The ensuing sections provide all the details necessary to carry out the assignment.

*Note: Before you start working on the assignment, make sure to carefully read the "Software Assignments Overview and Guideline" document available on the course site on Canvas. (In addition to, of course, carefully reading this document.)*

# 2 Software

The first deliverable of this assignment is a piece of software that conforms to certain restrictions (see Section 2.5) and is capable of reading and deserializing input exercise data from a file, performing the operation(s) indicated by this data, and serializing and writing the answer to a file.

## 2.1 Expectations

For the software deliverable of this assignment, you are expected to submit a Python 3 source code file called `solve.py`. This file must contain a function called `solve_exercise` that takes two `string` parameters: the first parameter takes the path to a file from which you read the input exercise data, and the second parameter takes the path to a file to which you write the output answer data; for clarity, Listing 1 provides the signature of `solve_exercise`. Given these parameters, `solve_exercise` should figure out which exercise it is asked to solve, solve the exercise, and write the answer in the desired format to the expected location. Here, it is important to note that that you are restricted in the use of the built-in features of Python 3, the use of the available libraries for Python 3, and the required execution time (for a precise description of these restrictions, see Section 2.5); this is to ensure that your software actually constitutes a self-made implementation of multi-precision integer and modular arithmetic that is of sufficient quality. Failing to conform to these restrictions may result in a significant loss of points.

```
1 def solve_exercise(exercise_location : str, answer_location : str)
```

Listing 1: Signature of the `solve_exercise` function that must be present in `solve.py`

An example `solve.py` file is available on the course page on Canvas. This file exemplifies, among other things, the input and output handling for this assignment. Feel free to use this file as a template or to draw inspiration from.

Notice that you may actually submit multiple files, as long as the set of files you submit contains a file called `solve.py` with a function called `solve_exercise` that satisfies the signature provided in Listing 1. You can use this to modularize the code, which may facilitate the development process.

## 2.2 Types of Exercises/Operations

The types of exercises your software should be able to solve are basic integer and modular arithmetic operations. More specifically, regarding integer arithmetic, your software should be capable of performing the following operations.

- **Addition**
  Given $x, y \in \mathbb{Z}$, compute $x + y$.
- **Subtraction**
  Given $x, y \in \mathbb{Z}$, compute $x - y$.
- **Multiplication (Primary School Method)**
  Given $x, y \in \mathbb{Z}$, compute $x \cdot y$ using the primary school method.
- **Multiplication (Karatsuba Method)**
  Given $x, y \in \mathbb{Z}$, compute $x \cdot y$ using the Karatsuba method.
- **Extended Euclidean Algorithm**
  Given $x, y \in \mathbb{N}$ (not both 0), compute $a$, $b$, and $d$ such that $a \cdot x + b \cdot y = d$ and $d = \gcd(x, y)$ using the extended Euclidean algorithm.

Concerning modular arithmetic, your software should be able to perform the following operations.

- **(Modular) Reduction**
  Given $x \in \mathbb{Z}$ and $m \in \mathbb{N}$, compute the smallest nonnegative integer congruent to $x$ modulo $m$. That is, compute $x \bmod m$.
- **(Modular) Addition**
  Given $x, y \in \mathbb{Z}$ and $m \in \mathbb{N}$, compute the smallest nonnegative integer congruent to $x + y$ modulo $m$. That is, compute $(x + y) \bmod m$.
- **(Modular) Subtraction**
  Given $x, y \in \mathbb{Z}$ and $m \in \mathbb{N}$, compute the smallest nonnegative integer congruent to $x - y$ modulo $m$. That is, compute $(x - y) \bmod m$.
- **(Modular) Multiplication**
  Given $x, y \in \mathbb{Z}$ and $m \in \mathbb{N}$, compute the smallest nonnegative integer congruent to $x \cdot y$ modulo $m$. That is, compute $(x \cdot y) \bmod m$; you may do so using any method.
- **(Modular) Inversion**
  Given $x \in \mathbb{Z}$ and $m \in \mathbb{N}$, compute the smallest nonnegative integer congruent to $x^{-1}$ modulo $m$.

Although your software may assume that the values in the exercises come from the above-specified domains, notice that this does not automatically rule out all possibilities for invalid inputs (i.e., inputs for which certain computations are impossible/undefined). For example, in any modular arithmetic operation, the provided modulus may still be 0.

Now, your software is expected to be able to perform the above-mentioned operations on large integer values. For the extended Euclidean algorithm and modular inversion, "large integer values" means "integer values in the range $[-10^{250}, 10^{250}]$"; for the other

operations, "large integer values" means "integer values in the range $[-10^{500}, 10^{500}]$". Furthermore, these integer values may be represented in any radix from the integer range $[2, 16]$; so, your software is additionally expected to be able to deal with values represented in these different radices[1].

### 2.3 Input

The format in which the input exercise data is formatted is JavaScript Object Notation (JSON). More precisely, an exercise is described in a single JSON object that *always* contains the following keys[2].

- `"type"`
  The value of this key indicates the type of the exercise, i.e., whether the exercise is an integer arithmetic exercise or a modular arithmetic exercise.
  This key has one of the following values.
  - `"integer_arithmetic"` (type: `string`)
    Indicates that the exercise is an integer arithmetic exercise.
  - `"modular_arithmetic"` (type: `string`)
    Indicates that the exercise is a modular arithmetic exercise.
- `"operation"`
  The value of this key indicates the operation to be performed for this exercise. The interpretation of the value of this key depends on the type of the exercise (as indicated by the value of the `"type"` key).
  This key has one of the following values.
  - `"addition"` (type: `string`)
    Indicates that the operation to be performed is addition. If the value of the `"type"` key is `"integer_arithmetic"`, this refers to (non-modular) integer addition; else, if the value of the `"type"` key is `"modular_arithmetic"`, this refers to modular addition.
  - `"subtraction"` (type: `string`)
    Indicates that the operation to be performed is subtraction. If the value of the `"type"` key is `"integer_arithmetic"`, this refers to (non-modular) integer subtraction; else, if the value of the `"type"` key is `"modular_arithmetic"`, this refers to modular subtraction.
  - `"multiplication"` (type: `string`)
    Indicates that the operation to be performed is multiplication. Only possible if the value of the `"type"` key is `"modular_arithmetic"`.

[1] To clarify: Although the radix in which values are represented may differ between exercises, it is the same for all values within a single exercise.

[2] All keys are of type `string`.

  - `"multiplication_primary"` (type: `string`)
    Indicates that the operation to be performed is multiplication using the primary school method. Only possible if the value of the `"type"` key is `"integer_arithmetic"`.
  - `"multiplication_karatsuba"` (type: `string`)
    Indicates that the operation to be performed is multiplication using the Karatsuba method. Only possible if the value of the `"type"` key is `"integer_arithmetic"`.
  - `"extended_euclidean_algorithm"` (type: `string`)
    Indicates that the operation to be performed is the extended Euclidean algorithm. Only possible if the value of the `"type"` key is `"integer_arithmetic"`.
  - `"reduction"` (type: `string`)
    Indicates that the operation to be performed is modular reduction. Only possible if the value of the `"type"` key is `"modular_arithmetic"`.
  - `"inversion"` (type: `string`)
    Indicates that the operation to be performed is modular inversion. Only possible if the value of the `"type"` key is `"modular_arithmetic"`.

- `"radix"`
  The value of this key denotes the radix in which the integer values of this exercise are represented.
  This key has a value of type `number` that is in the integer range $[2, 16]$.

- `"x"`
  The value of this key denotes the value of the left-hand side operand of the operation.
  This key has a value of type `string` that contains between 1 and (approximately) 1660 characters. Each character, except for potentially the first one, denotes a digit from the set of valid digits determined by the value of the `"radix"` key[3]. The first character of the string may be a `"-"`, indicating that the value is negative[4]. (In case the value is positive, the first character is simply a digit.) This string is to be interpreted as an integer in the radix indicated by the value of the `"radix"` key.

- `"points"`
  The value of this key indicates the number of points this exercise is worth. This key/value pair is only used for grading; you can safely ignore this.

The remainder of the keys present in a JSON object that describes an exercise is

[3] For example, if the `"radix"` key has value 2, each character in the string (except for potentially the first one) is either `"0"` or `"1"`.

[4] For instance, if the value of the `"radix"` key is 16, `"-FFFF"` denotes the (decimal) value -65535.

dependent on the values of the "type" and "operation" keys. Indeed, this is because different exercises may be defined by different values. The following is a list of all keys that *may* be present in a JSON object that describes an exercise.

- "y"
  The value of this key denotes the value of the right-hand side operand of the operation. This key is only (but always) present if the "operation" key has value "addition", "subtraction", "multiplication", "multiplication_primary", "multiplication_karatsuba", or "extended_euclidean_algorithm".
  This key has a value of type `string` that contains between 1 and (approximately) 1660 characters. Each character, except for potentially the first one, denotes a digit from the set of valid digits determined by the value of the "radix" key. The first character of the string may be a "-", indicating that the value is negative. (In case the value is positive, the first character is simply a digit.) This string is to be interpreted as an integer in the radix indicated by the value of the "radix" key.

- "modulus"
  The value of this key denotes the value of the modulus. This key is only (but always) present if the "type" key has value "modular_arithmetic"
  This key has a value of type `string` that contains between 1 and (approximately) 1660 characters. Each character denotes a digit from the set of valid digits determined by the value of the "radix" key. This string is to be interpreted as an integer in the radix indicated by the value of the "radix" key.

Exemplifying the above-specified description of exercises, Listing 2, Listing 3, and Listing 4 respectively provide the descriptions of an integer arithmetic addition exercise, an integer arithmetic extended Euclidean algorithm exercise, and a modular arithmetic reduction exercise. Of course, these examples are intentionally made simple for instructional purposes. Additional examples of input exercise descriptions (and corresponding output answer descriptions), both simple and realistic, can be found on the course site on Canvas.

```
{
    "type": "integer_arithmetic",
    "operation": "addition",
    "radix": 3,
    "x": "-1020",
    "y": "1102",
    "points": 0.5
}
```

Listing 2: Example Description of Integer Arithmetic Addition Exercise.

```json
{
    "type": "integer_arithmetic",
    "operation": "extended_euclidean_algorithm",
    "radix": 10,
    "x": "96",
    "y": "40",
    "points": 0.5
}
```

Listing 3: Example Description of Integer Arithmetic Extended Euclidean Algorithm Exercise.

```json
{
    "type": "modular_arithmetic",
    "operation": "reduction",
    "radix": 16,
    "x": "1F",
    "modulus": "E",
    "points": 0.5
}
```

Listing 4: Example Description of Modular Arithmetic Reduction Exercise.

### 2.4 Output

As the input exercise description data, the output answer data is (to be) formatted in JSON. An answer is described in a single JSON object that either contains the three keys `"answer-a"`, `"answer-b"`, and `"answer-gcd"`, or the single key `"answer"`. The following elaborates on these keys.

- `"answer-a"`
  The value of this key denotes the first coefficient of Bézout's identity (i.e., the $a$ in $a \cdot x + b \cdot y = \gcd(x, y)$) for the operand values (i.e., the integers denoted by the values of keys `"x"` and `"y"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"operation"` key in the description of the corresponding exercise has value `"extended_euclidean_algorithm"`.
  This key has a value of type `string` that contains at least one character. Each character, except for potentially the first one, denotes a digit from the set of valid digits determined by the value of the `"radix"` key in the description of the

corresponding exercise. The first character of the string may be a `"-"`, indicating that the value is negative. (In case the value is positive, the first character is simply a digit.) This string is interpreted as an integer in the radix indicated by the value of the `"radix"` key in the description of the corresponding exercise.

- `"answer-b"`
  The value of this key denotes the second coefficient of Bézout's identity (i.e., the $b$ in $a \cdot x + b \cdot y = \gcd(x, y)$) for the operand values (i.e., the integers denoted by the values of keys `"x"` and `"y"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"operation"` key in the description of the corresponding exercise has value `"extended_euclidean_algorithm"`.
  This key has a value of type `string` that contains at least one character. Each character, except for potentially the first one, denotes a digit from the set of valid digits determined by the value of the `"radix"` key in the description of the corresponding exercise. The first character of the string may be a `"-"`, indicating that the value is negative. (In case the value is positive, the first character is simply a digit.) This string is interpreted as an integer in the radix indicated by the value of the `"radix"` key in the description of the corresponding exercise.

- `"answer-gcd"`
  The value of this key denotes the greatest common divisor of the operand values (i.e., the integers denoted by the values of keys `"x"` and `"y"`) provided in the description of the corresponding exercise. This key is only (but always) present if the `"operation"` key in the description of the corresponding exercise has value `"extended_euclidean_algorithm"`.
  This key has a value of type `string` that contains at least one character. Each character denotes a digit from the set of valid digits determined by the value of the `"radix"` key in the description of the corresponding exercise. This string is interpreted as an integer in the radix indicated by the value of the `"radix"` key in the description of the corresponding exercise.

- `"answer"`
  The value of this key denotes the answer to the corresponding exercise.
  This key has a value of type `string` that contains at least one character. Each character, except for potentially the first one, denotes a digit from the set of valid digits determined by the value of the `"radix"` key in the description of the corresponding exercise. The first character of the string may be a `"-"`, indicating that the value is negative. (In case the value is positive, the first character is simply a digit.) This string is interpreted as an integer in the radix indicated by the value of the `"radix"` key in the description of the corresponding exercise.

As alluded to before, it may occur that an exercise description describes an exercise that amounts to performing an operation that is impossible/undefined (e.g., a modular arithmetic operation with a modulus of 0). In such a case, the keys in the answer

description should be the same as in a case where the operation to be performed is well-defined; however, all keys should be assigned the special `null` value.

Illustrating the above-specified description of answers in JSON, Listing 5, Listing 6, and Listing 7 respectively provide the descriptions of the answers to the exercises described in Listing 2, Listing 3, and Listing 4.

```
{
    "answer": "12"
}
```

Listing 5: Description of Answer to Integer Arithmetic Addition Exercise Described in Listing 2.

```
{
    "answer-a": "-2",
    "answer-b": "5",
    "answer-gcd": "8"
}
```

Listing 6: Description of Answer to Integer Arithmetic Extended Euclidean Algorithm Exercise Described in Listing 3.

```
{
    "answer": "3"
}
```

Listing 7: Description of Answer to Modular Arithmetic Reduction Exercise Described in Listing 4.

## 2.5 Remarks and Restrictions

In order to ensure that your software actually comprises a self-made implementation of multi-precision integer and modular arithmetic, it should satisfy the following requirements.

- Your software may not store and/or operate on values larger than what fits in 32

bits (Python 3's built-in integer type allows for storing and operating on nearly arbitrarily large integer values). To assist you in not accidentally storing and/or operating on values larger than what fits in 32 bits, you may use the built-in `ctypes` library and/or the external `fixedint` library.

- Your software may not use any libraries other than the `json`, `ctypes`, and `fixedint`. If you want to use any other libraries, you can ask for permission to do so via an e-mail to `m.c.f.h.p.meijers@tue.nl`. Naturally, any libraries that make the assignment significantly easier will be rejected.
- Your software may not use built-in features that perform radix conversions. In particular, your software cannot use the `int()` function with two arguments[5].

Then, concerning the quality of your software, the ensuing requirement should be met.

- Your software should, for any exercise, complete within 5 seconds. If your software exceeds this limit for an exercise, you will not be granted any points (for that particular exercise).

For further general software requirements that apply to both software assignments, please make sure to carefully read the "Software Assignments Overview and Guideline" document available on the course site on Canvas.

[5] However, your software may use the `int()` function with a single argument to, e.g., truncate and cast floats to integers.

# 3 Documentation

The second deliverable of this assignments is documentation for the first deliverable, i.e., your piece of software.

## 3.1 Expectations

You are expected to hand in a single PDF file containing the documentation for your constructed software. This documentation should, at least, contain the following.

- A title page with an appropriate title (e.g., "2WF90 Software Assignment 1"), as well as the names and student IDs of the submission's group members.
- Table of contents.
- A clear statement specifying the version of Python 3 in which the submitted software is written.
- In case you have been given the permission to use additional libraries (in addition to the explicitly permitted libraries in the relevant assignment description), a clear statement specifying the additional libraries that are used in the submitted software. Moreover, if necessary, provide clear installation and setup instructions for each of these libraries.
- A concise explanation of the purpose of your software; that is, a short problem description.
- A comprehensive explanation of the approach your software takes to serve its purpose/solve the considered problem. This includes a mathematical description of what your software is capable of doing and how it does so.
- A discussion on the difference between multiplication using the primary school method and multiplication using the Karatsuba method based on your software. This should include a comparison between the number of elementary (addition and subtraction) operations performed during multiplication using these methods.
- An explanation of the limitations of your software.
- Illustrative examples covering all operations with different operand sizes and radices; these examples should convince the reader that your software works as expected (i.e., provide examples of both regular and special/edge cases).
- A description of each group member's contribution to the submitted work.
- Where relevant, proper references to lecture material and/or literature.
- In case any references are used throughout the document, a proper list of references at the end of the document.

### 3.2 Remarks and Restrictions

For general documentation requirements that apply to both software assignments, please make sure to carefully read the "Software Assignments Overview and Guideline" document available on the course site on Canvas.