𒅴𒋃𒀀

First Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Second Author
Affiliation / Address line 1
Affiliation / Address line 2
Affiliation / Address line 3
email@domain

Abstract

This paper announces the discovery of the use of neural nets almost 4,000 years before their use in the modern era. Newly discovered tablets preserve a perceptron used for calculating the numbers on Plimpton 322, the most important object in the history of mathematics. The native programming language used by the ancient Babylonian "cuneogrammers" uses sexagesimal numbering leading to some "weirdness".

[illegible]

[illegible]

1 Introduction

The history of math is long, but the history of programming is longer. Cuneiform, arguably the first writing in the world, is known for its sheep receipts and beer ration lists as well as copper complaints (Oppenheim, 1954). This article adds neural network programming to that vaunted list of human achievements. A set of newly discovered cuneiform tablets preserve the mechanism for performing simple neural network calculations. These methods, it seems, were used to calculate the lengths of triangles; a well known exercise preserved on the tablet known as Plimpton 322[1]. It is remarkable that no historian of math or cuneiform scholar has ever consider this possibility. This paper covers the background, a description of the cuneogramming language, and includes a facsimile copy of the most important tablet as an appendix.

Unfortunately, the hardware required to execute these programs (i.e. a living Babylonian mathematician) has not been adequately preserved, but we have managed to write a Python library which emulates it.[2] The assumption is that these calculations were done by hand in their copious free time between inventing the wheel and the concept of zero. While the actual output of these tablets is relatively simple by modern standards, the implications of this discovery are profound. Future work will explore how these techniques could have been used in the realm of astronomical calculation and elucidate the full extent of the Babylonian's computational prowess.

2 Description of the Language

Programs in 𒅴𒋃𒀀 (EME.ŠID.A "language of counting") follow a tabular structure with three main sections: (i) a header, denoted by 𒁾 (DUB "tablet"), (ii) a sequence of instructions, and (iii) a colophon detailing the tablet's authorship. Each instruction spans four columns, which we have taken to calling the *arguments*, *opcode*, *destination*, and *line number*. These columns are usually tab-separated, though in a few documents they are TAB-separated (using the cuneiform sign TAB 𒋰). Instructions are grouped into blocks by means of horizontal lines.

Arguments An instruction's arguments may be numbers, register addresses, or a combination of the two. Numbers are encoded following standard Babylonian conventions (**?**), with [illegible] denoting the radix point which separates the integer part from a following fraction. There is an explicit representation for zero ([illegible]), making these tablets some of the earliest unambigiuous examples of the mathematical concept of zero.

[1] And important and real introduction into this interesting object is found in Robson (2002).

[2] github.com/MrLogarithm/emeszida

A register address is denoted by the phrase NID2.KAS7 *n*-KAM 𒃻𒋃*n*𒄰 (“thing.account *n*-th”, “the item in the *n*-th place”), where *n* is any number. 𒃻𒋃 expressions can be nested to perform a kind of pointer dereferencing: for example, if 𒃻𒋃𒁹𒄰 (register 1) contains the value 𒐈 (3), and 𒃻𒋃𒐈𒄰 (register 3) contains the value 𒐌 (7), then 𒃻𒋃𒃻𒋃𒁹𒄰𒄰 will evaluate to 7 (the value in the register pointed to by 𒃻𒋃𒁹𒄰).

If an instruction has multiple arguments, these must be delimited by a wide space (distinct from the short space used to separate groups of digits within a number), or by one of the phrases *a-na* 𒀀𒈾 “to” or *i-na* 𒄿𒈾 “from”. By convention, the choice of delimiter depends on the instruction’s opcode (see below), with multiplication operations preferring space delimiters, addition preferring 𒀀𒈾, and subtraction preferring 𒄿𒈾. There is no mechanism to enforce these conventions, but we recommend following them because 𒊩𒋃𒀀 is hard enough to read at the best of times.

Destination In most cases, the destination column of an instruction will be a register address where the output is to be stored, e.g. 𒃻𒋃𒁹𒄰. Some control-flow instructions (see below) instead expect the destination to be a line number. 𒋤 SUD (“distant, remote”) can be used as a null destination, for statements which produce no output.

Opcodes Each instruction has a single opcode belonging to the following vocabulary:

- 𒁕𒄩 DAḪ.ḪA, “add”
- 𒁀𒍣 BA.ZI, “tear out”
- 𒀀𒊏 A.RA , “multiply”
- 𒅆 IGI, “reciprocal”
- 𒈨 ME, “to be”
- 𒅆𒈥 *ta-mar*, “you will see”
- 𒉏𒈾 NIGIN.NA, “start again”
- 𒋾𒌆𒁉 𒋛𒀀 TUKUM.BI DIRIG, “if it exceeds”
- 𒋾𒌆𒁉 𒋝 TUKUM.BI SIG, “if it it is weak”

The first three of these are binary operators for addition, subtraction, and multiplication respectively. The subtraction operator deserves special attention for its use in constructions of the following shape, which appear hundreds of times throughout the Babylonian programming corpus:

𒑱𒑱𒐈𒌍𒄿𒈾𒑱 𒁀𒍣 𒃻𒋃𒁹𒄰 𒑱

This instruction subtracts 𒑱𒑱𒐈𒌍 from zero and stores the result in 𒃻𒋃𒁹𒄰. This effectively negates the first argument, and seems to have been the primary way by which Babylonian cuneigrammers represented negative numbers, as their primitive and archaic notation otherwise lacked a means to encode such values. This curious practice gives definitive proof that the invention of negative numbers occurred centuries earlier than previously believed.

The language does not appear to have any kind of binary division operator. Rather, a unary 𒅆 operator was used to find the reciprocal of the denominator, which was then multiplied by the numerator using the binary 𒀀𒊏 operator.

𒈨 is a unary assignment operator which stores a value in a destination register. 𒅆𒈥 functions as a unary print operator.

𒊩𒋃𒀀 includes three types of control-flow instructions. 𒉏𒈾 functions like GOTO, and jumps the program counter to the specified line number. 𒋾𒌆𒁉𒋝 functions like the x86 jz instruction, and jumps to the specified line number if and only if its argument is zero. 𒋾𒌆𒁉𒋛𒀀 is similar, and jumps if the argument is greater than zero.

Line Numbers Every line of an 𒊩𒋃𒀀 program ends in a mandatory line number. However, these numbers are not generally sequential, and need not even be distinct. For example, most lines of the perceptron tablet are labeled with the number 𒑱 (zero); only lines that are the destination of some control-flow instruction receive non-zero identifiers.

2.1 Fractional Indexing

Both line numbers and register addresses in 𒊩𒋃𒀀 can have fractional parts. The original scribes seem to have exploited this fact to establish non-overlapping “namespaces” for the different parts of their code. For example, in the perceptron tablet, all of the model

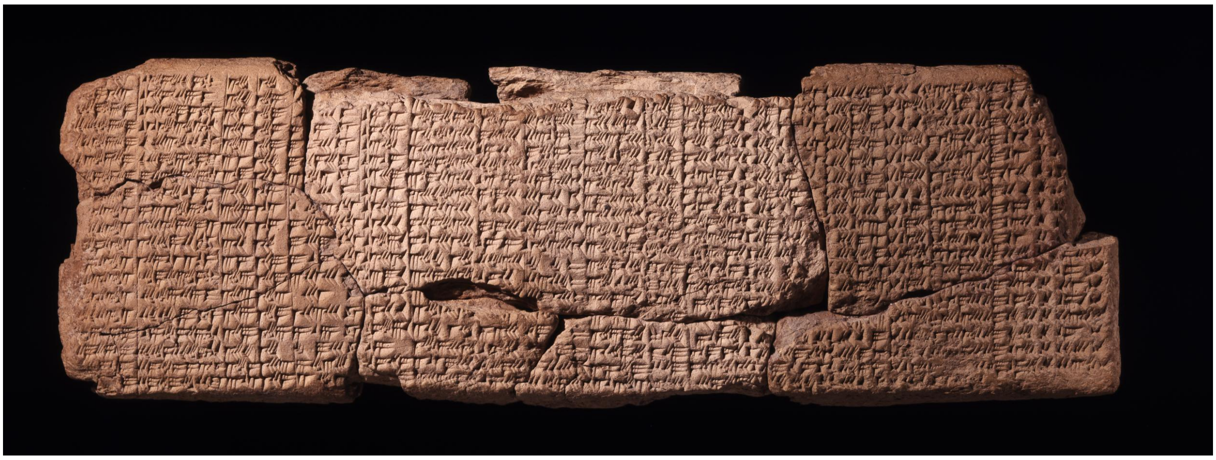

Figure 1: BM 34580, courtesy of the Trustees of the British Museum, CC BY-NC-SA 4.0

parameters are stored in adresses with integer part 1; the program inputs all have integer part 2; the matrix multiplication subroutine uses addresses with integer part 3; and so on. The fractional parts of register addresses also appear to follow some standard conventions, with the X;0 register typically storing a subroutine's return address, while X;1 onward were used for its arguments.

The perceptron tablet also uses fractional register addresses to perform a kind of multi-dimensional array indexing. As an example, the first layer of the perceptron has a 50×2 weight matrix, and this is stored in registers 1;0,0,0 through 1;0,49,1. The integer part of these addresses denotes the "data" portion of memory; the first digit after the radix point identifies this as the 0th model parameter; and the second and third digits can be treated as a pair of indices ranging from 0–49 and 0–1 respectively. To access a specific element in this matrix, the scribes use repeated division by 60 to implement a kind of "bit-shift" instruction, in order to shift integer indices into the correct positions after the radix point. By adding the bit-shifted element indices (e.g. 0;0,4,7 for the element in row 5, column 8) to a pointer to the top corner of the matrix (1;0,0,0) they obtain the address of the desired element (1;0,4,7).

Notably, this practice limits the size of their model parameters to at most 60×60, as for larger values the addresses would carry over to higher digits and thus begin to overwrite one another. This limitation may explain why AI never made waves in Babylonian society, as their models were all too small to be truly revolutionary.

3 Description of the Texts

The tablets preserve a number of methods for performing simple neural network calculations. Simple array operations as well as the calculation of the dot product are included in the corpus. All of this comes together in the large perceptron tablet. It is clear that the cuneogrammers were slowly developing their skills as some of the earlier and simpler methods date as far back as the Old Akkadian period (c. 2300–2150 BCE) while the perceptron tablet is from the Old Babylonian period (c. 1900–1600 BCE). The longest tablet, included in the appendix, shows a fully featured perceptron ADD MORE HERE Interestingly, the long weights at the end of the tablet very closely resemble the parameters of later astronomical calculations (see Figure 1).

- add graph plotting perceptron output vs true trigonometric ratios from Plimpton

4 Implications

This completely rewrites the history of modern computing...

Other ancient corpora which have resisted decipherment, and which boast a similar numeric component, may represent additional examples of ancient programming traditions.

- compare the table of params at the end of the file to tables of astronomical parameters from genuine tablets

Acknowledgments

The search for these tablets and our effort to understand them was inspired by Ramsey Nasser's قلب programming language (https://github.com/nasser/—).

References

A. L. Oppenheim. 1954. The seafaring merchants of ur. *Journal of the American Oriental Society*, 74(1):6–17.

Eleanor Robson. 2002. Words and pictures: New light on plimpton 322. *The American Mathematical Monthly*, 109(2):105–120.

A Example Appendix

- reproduce entire perceptron-full.eme in the text, without comments:

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

𒑱 𒐌 𒐏𒐉 𒎙𒐍 𒎙𒐌 𒐌 𒎙𒐉 𒌍 𒌍𒁹 𒌋𒈫 𒐌 𒌍𒐎 𒌍𒐉 𒎙𒁹 𒐏 𒐐𒐋 𒌍𒐈 𒌍𒐋
𒑱 𒌋𒐈 𒌋𒁹 𒌍𒐊 𒎙𒈫 𒌍𒐎 𒐐𒐌 𒌍𒐎 𒎙𒈫 𒌍 ⊨◈
𒑱 𒐌 𒎙𒐍 𒐏𒐍 𒐐𒈫 𒎙𒐈 𒐈 𒐐𒐋 𒐏𒐉 𒐐𒐋 𒐐𒐎 𒎙𒐈 𒐏𒐊 𒐏 𒐋 𒐏𒐎 𒐐𒐊 𒌋𒈫 ⊨◈
𒑱 𒐌 𒌋𒐌 𒎙𒐌 𒎙𒐋 𒎙𒁹 𒐍 𒐐𒐈 𒌋𒁹 𒎙𒐍 𒌋𒐋 𒌍𒐈 𒌋𒈫 𒈫 𒌋𒐋 𒌋𒐉 𒎙𒐉
𒑱 𒐌 𒐍 𒎙𒐎 𒌍𒈫 𒐐𒐎 𒐐𒐉 𒐏𒁹 𒐌 𒌋𒐍 𒌍𒐈 𒐌 𒐌 𒁹 𒎙𒐉 𒐊 𒐏𒐊 𒌍𒐋
𒑱 𒐎 𒐐𒐊 𒌋𒐉 𒌍𒐊 𒌍𒁹 𒎙𒐍 𒌋𒐋 𒎙 𒌍𒁹 𒎙𒐎 𒐐𒐍 𒎙𒐋 𒐐𒐋 𒎙 𒐐𒐊 𒐏 𒐏𒐍 ⊨◈
𒑱 𒐊 𒐉 𒐏𒐋 𒌍𒐈 𒎙𒐊 𒌋𒐋 𒐈 𒐊 𒎙 𒎙𒐉 𒌋𒈫 𒐏𒐋 𒎙𒐉 𒐎 𒌋𒈫 𒐐𒐎 𒌍𒐋
𒑱 𒐎 𒐐𒐊 𒐐𒐎 𒐐𒐊 𒁹 𒌍𒐌 𒌍 𒌋𒐈 𒌋𒁹 𒐉 𒐐𒐎 𒌋 𒌋𒁹 𒐏𒈫 𒐏𒐈 𒌋𒈫
𒑱 𒌋𒁹 𒐍 𒎙𒐍 𒌍𒐋 𒐉 𒌍𒁹 𒐏 𒐏𒁹 𒎙𒈫 𒐐𒐈 𒎙𒐋 𒎙𒐉 𒐏𒐉 𒌋𒐉 𒌋𒈫 𒎙𒐍 𒐏𒐍
𒑱 𒌋𒈫 𒌍𒐉 𒐊 𒐌 𒐐𒈫 𒐐𒐋 𒌍𒐎 𒎙 𒐉 𒐐𒐎 𒐊 𒐐𒐎 𒐋 𒐊 𒌋𒁹 𒈫 𒎙𒐉 ⊨◈
𒑱 𒌋𒐌 𒌍 𒎙𒐌 𒌋 𒌋 𒐐𒐋 𒐎 𒌍 𒎙𒐉 𒌍𒐋 𒐐𒐋 𒌋𒐋 𒐌 𒁹 𒌍𒐌 𒐐𒐊 𒌋𒈫 ⊨◈
𒑱 𒌋 𒌋𒐉 𒎙𒐍 𒌋𒁹 𒎙𒐊 𒈫 𒌋𒐎 𒌍𒁹 𒌍𒐉 𒐐𒐈 𒎙𒈫 𒌍𒈫 𒎙𒐎 𒌍𒐍 𒐏𒁹 𒌋𒐋 𒐏𒐍 ⊨◈
𒑱 𒐏 𒌋𒐎 𒐐𒐋 𒌍𒁹 𒌋𒈫 𒌋𒐈 𒐏𒐎 𒐏𒐋 𒐏𒈫 𒌍𒁹 𒌍𒐊 𒎙𒐋 𒐐𒐋 𒌍𒐌 𒎙𒐍 𒐏𒈫 𒌋𒐉 𒎙𒐉 ⊨◈
𒑱 𒐈 𒌋𒐎 𒌍𒁹 𒐋 𒐐𒁹 𒌍 𒎙𒐌 𒈫 𒎙𒐉 𒎙𒁹 𒌍𒐊 𒐐𒐈 𒁹 𒐐𒐌 𒐏𒐉 𒐈 𒐐 𒎙𒐉
𒑱 𒌋𒐊 𒎙𒐋 𒐏𒈫 𒌍𒁹 𒌍𒐎 𒎙𒐊 𒌍𒐉 𒌍𒐉 𒌋𒈫 𒐐𒐉 𒌋𒐋 𒎙𒈫 𒐐𒐈 𒌋𒁹 𒈫 𒎙𒐉
𒑱 𒐉 𒌋𒐋 𒐌 𒐐𒈫 𒎙𒐎 𒌋𒐌 𒐈 𒁹 𒐌 𒐐𒈫 𒌍𒐋 𒌋𒈫 𒌋𒈫 𒌋𒈫 𒐏 𒌋𒐎 𒌋𒈫 ⊨◈
𒑱 𒐈 𒎙𒁹 𒌍𒈫 𒐏𒐌 𒐐𒐌 𒐐𒐊 𒐏𒐎 𒌍𒐊 𒌋𒐌 𒐏𒐈 𒌍𒈫 𒌋𒐉 𒐐𒁹 𒌋𒁹 𒌋𒐈 𒐐𒐊 𒌋𒈫
𒑱 𒐋 𒌍 𒐐𒐉 𒌍𒐈 𒐐𒐌 𒌍𒐋 𒎙𒐉 𒌍𒁹 𒌋𒐌 𒎙𒁹 𒌋𒐌 𒌋 𒐐𒁹 𒎙 𒎙𒐋 𒐐𒈫 𒐏𒐍 ⊨◈
𒑱 𒐊 𒎙𒐉 𒐏 𒌍𒁹 𒁹 𒐏𒐉 𒌋 𒐏 𒌋𒐋 𒐐𒐎 𒈫 𒐋 𒐐𒁹 𒎙𒐎 𒌍𒐎 𒐐 𒎙𒐉
𒑱 𒌋𒐈 𒐐𒐈 𒐏𒐎 𒌍𒐋 𒐈 𒐐𒐋 𒐐𒐍 𒌍 𒎙𒁹 𒐋 𒌍𒐈 𒌋𒐈 𒐏𒐌 𒎙𒁹 𒐐𒐎 𒈫 𒎙𒐉
𒑱 𒌋 𒎙 𒌍𒈫 𒐐𒈫 𒌋𒁹 𒌍 𒌍𒁹 𒐊 𒎙𒐍 𒌍𒐋 𒎙𒐈 𒐏𒐎 𒎙𒐉 𒐊 𒐐𒐊 𒌍𒐋
𒑱 𒁹 𒎙𒈫 𒐈 𒎙𒐋 𒌍 𒌍𒁹 𒎙𒐍 𒌍𒐎 𒌋 𒐊 𒐉 𒎙 𒎙𒐈 𒐐𒐈 𒐊 𒌋𒐋 𒐏𒐍
𒑱 𒌋𒐋 𒐐𒐍 𒎙𒐌 𒐏𒈫 𒐊 𒌍𒈫 𒌍𒐍 𒌋𒐍 𒐏𒐋 𒐏𒐍 𒌋𒐈 𒐐𒐋 𒐉 𒌍𒁹 𒐐𒈫 𒌋𒐎 𒌋𒈫 ⊨◈
𒑱 𒐍 𒐐𒐋 𒐈 𒌋𒐊 𒐏𒐌 𒐊 𒐏𒐎 𒌋𒐊 𒌋𒐎 𒎙𒐈 𒎙 𒐏𒁹 𒌍𒐌 𒌍𒈫 𒐎 𒌍𒐋 ⊨◈
𒑱 𒌋𒐊 𒎙𒐊 𒌋𒈫 𒎙𒐎 𒐏𒐎 𒐐𒐎 𒌍𒈫 𒌍𒐍 𒌍𒐍 𒐏𒐌 𒎙 𒐊 𒐏 𒐐𒐎 𒌍𒁹 𒌋𒈫
𒑱 𒌋 𒌋𒐊 𒌍𒁹 𒐐𒐋 𒐏𒐈 𒐎 𒐐𒐋 𒌋𒐍 𒐐𒐊 𒌍𒐉 𒌋𒐋 𒌍𒐋 𒐐𒐈 𒐏 𒐐𒐎 𒌍𒁹 𒌋𒈫 ⊨◈
𒑱 𒐏 𒌍𒁹 𒌍𒐊 𒌋𒐉 𒐏 𒌍𒐈 𒌋𒐊 𒐊 𒁹 𒐐𒈫 𒎙𒐊 𒎙𒐌 𒐐𒐉 𒐌 𒐐𒐋 𒐐𒐊 𒐏 𒐏𒐍
𒑱 𒐋 𒐐𒐈 𒐐 𒌋𒐌 𒎙𒐈 𒌋𒈫 𒐏𒈫 𒐐𒐍 𒎙𒈫 𒐐𒐈 𒌍𒐋 𒐊 𒎙 𒐏𒐈 𒐏 𒎙𒐍 𒐏𒐍 ⊨◈
𒑱 𒐍 𒎙𒐎 𒁹 𒐏𒐉 𒎙 𒎙𒁹 𒌋𒁹 𒐌 𒐊 𒎙𒁹 𒁹 𒐏𒁹 𒐐𒈫 𒐐𒐍 𒎙𒈫 𒐉 𒐏𒐍 ⊨◈
𒑱 𒐏 𒌋𒐉 𒐐𒐎 𒌍𒐌 𒁹 𒌍𒐎 𒌍𒐎 𒐐 𒎙𒐍 𒎙𒐋 𒌋𒐋 𒐌 𒌍𒈫 𒎙𒐎 𒐐𒈫 𒌍 𒐏𒐈 𒌋𒈫
𒑱 𒐊 𒌍𒐊 𒐐𒐎 𒐎 𒐌 𒐊 𒐏𒐍 𒌍𒐉 𒐌 𒎙𒈫 𒐐𒐊 𒌍𒈫 𒈫 𒐏𒁹 𒌋𒐋 𒐏𒐍 ⊨◈
𒑱 𒌋𒐋 𒐏𒐉 𒌍𒐉 𒐋 𒌋𒁹 𒎙𒐈 𒌍𒐉 𒐎 𒌍𒐋 𒌍𒐈 𒌋 𒐐𒐎 𒌋𒐊 𒐐𒐎 𒌍𒐋 𒐐𒐊 𒌍𒐋 ⊨◈
𒑱 𒈫 𒌋𒐎 𒌍𒐈 𒐐𒐉 𒎙𒐎 𒎙𒁹 𒐐𒐎 𒌋𒐉 𒌋𒐎 𒐏𒐍 𒐐𒐋 𒎙 𒌋𒐉 𒐏 𒐌 𒐏 𒐏𒐍
𒑱 𒐍 𒐐 𒐌 𒐏𒐍 𒐏 𒐐𒐌 𒐐𒐌 𒎙𒐋 𒐐𒁹 𒈫 𒐏𒐎 𒐏𒐌 𒌋 𒐐𒐊 𒎙𒐎 𒌋𒐋 𒐏𒐍 ⊨◈
𒑱 𒁹 𒌋𒐌 𒐏𒁹 𒎙𒐎 𒎙𒐊 𒐋 𒐎 𒐋 𒎙𒈫 𒌍𒐎 𒎙𒐊 𒐐𒐌 𒐋 𒎙𒐌 𒐏𒐊 𒐏𒐌 𒌍𒁹 𒌋𒈫
𒑱 𒐌 𒌍𒐈 𒐏 𒌋𒐊 𒐏𒈫 𒐐𒐌 𒌋𒁹 𒌋𒁹 𒐏𒈫 𒌋𒐋 𒐏𒐌 𒐋 𒎙𒁹 𒐐𒐋 𒎙𒈫 𒐉 𒐏𒐍 ⊨◈
𒑱 𒐈 𒎙𒐌 𒎙𒐊 𒌍𒁹 𒐐𒈫 𒐐𒐊 𒌍𒈫 𒌍𒐊 𒌍𒐈 𒐐𒐋 𒐉 𒌋𒐈 𒐏𒈫 𒐐𒐎 𒌋𒐎 𒐏 𒐏𒐍 ⊨◈
𒑱 𒐎 𒐏 𒐏 𒌍𒁹 𒐐𒐊 𒐐𒐎 𒐐𒐉 𒎙𒐌 𒐋 𒐏𒐎 𒐏𒈫 𒌍𒐋 𒌍𒐎 𒎙𒐈 𒐐𒐉 𒌋𒐉 𒎙𒐉
𒑱 𒌋𒐋 𒐐𒐉 𒐐𒈫 𒐍 𒐐𒐉 𒌍𒐉 𒐐𒁹 𒌍𒐉 𒌍𒐌 𒐐𒁹 𒎙𒐈 𒐉 𒎙𒐋 𒎙𒐉 𒐐𒐌 𒌍𒐋 ⊨◈
𒑱 𒐌 𒎙𒐍 𒐊 𒌋𒐉 𒌍𒐌 𒐏𒐌 𒐐𒐋 𒌍𒁹 𒎙𒈫 𒐎 𒐊 𒐐𒐌 𒎙𒁹 𒁹 𒎙𒐋 𒎙𒐉 ⊨◈
𒑱 𒐋 𒐏𒐊 𒐏𒐎 𒌍𒐈 𒁹 𒐏𒐉 𒐐𒐈 𒌍𒁹 𒐊 𒐉 𒐏𒐎 𒎙𒐊 𒐏𒈫 𒌍𒁹 𒐏 𒐏𒐍
𒑱 𒐊 𒌋𒈫 𒎙𒐉 𒐏𒐌 𒌋𒈫 𒈫 𒐏𒐎 𒎙𒐎 𒎙𒐎 𒐏𒐈 𒌍𒐍 𒐈 𒌋𒐈 𒐉 𒌍 𒐏𒐈 𒌋𒈫
𒑱 𒁹 𒐏𒐋 𒎙 𒌋𒁹 𒎙𒈫 𒌋 𒌍𒐈 𒐍 𒐐𒐋 𒌋𒐉 𒌋𒐌 𒐉 𒌋 𒎙𒐋 𒐏𒁹 𒌋𒐋 𒐏𒐍
𒑱 𒐌 𒐏𒐌 𒌍𒐊 𒎙𒐍 𒌋 𒐏 𒌍𒈫 𒌍𒐍 𒐐𒁹 𒐏𒁹 𒌋𒐌 𒐌 𒎙𒁹 𒌋𒈫 𒐐𒐌 𒌍𒐋
𒑱 𒁹 𒌋𒐍 𒌋𒐌 𒐏𒐈 𒎙𒐊 𒐐𒐉 𒐐𒐋 𒐐𒐎 𒐏𒐎 𒎙𒐋 𒐐𒐊 𒐐𒐋 𒐏 𒌋𒁹 𒐏𒐎 𒌍𒐌 𒐐𒐊 𒌋𒈫
𒑱 𒁹 𒐏𒐈 𒁹 𒐐𒐋 𒌋𒐌 𒐐𒐈 𒐐𒐎 𒌋𒐉 𒌍𒐊 𒐐𒐊 𒐏𒐌 𒐏𒐉 𒎙𒈫 𒐏𒐈 𒐐𒐋 𒐉 𒐏𒐍
𒑱 𒐌 𒐏𒐊 𒌋𒐎 𒌋𒐈 𒌋𒐊 𒌋𒐎 𒌍𒐎 𒐐𒐊 𒐐𒐈 𒐐𒐉 𒎙𒐍 𒌋 𒐌 𒎙 𒐈 𒐐 𒎙𒐉 ⊨◈
𒑱 𒎙 𒌋𒐊 𒎙𒐎 𒌋𒈫 𒎙𒐋 𒎙𒈫 𒌋𒁹 𒐏𒐊 𒐏𒐊 𒐏𒁹 𒎙𒐍 𒐏𒐉 𒐉 𒐐𒐎 𒌍𒁹 𒌋𒈫 ⊨◈
𒑱 𒌋𒐈 𒐐𒐊 𒎙𒐊 𒐏𒈫 𒁹 𒌋𒐋 𒌍𒐊 𒌋𒐍 𒎙𒐈 𒐉 𒐐𒐈 𒌋𒐊 𒐐𒐌 𒐌 𒌋𒈫 ⊨◈
𒑱 𒐋 𒎙𒐈 𒎙𒐋 𒐐𒐈 𒐌 𒐏𒁹 𒌋𒐌 𒐏𒐈 𒐍 𒐏𒐊 𒐐𒐈 𒐏 𒌋𒈫 𒐏 𒌋𒐎 𒌋𒈫
𒑱 𒐊 𒐏𒐋 𒐏𒐎 𒈫 𒐐𒐋 𒌍𒐎 𒐊 𒌍𒐎 𒐏𒐍 𒎙𒐌 𒐐𒈫 𒐏𒈫 𒌋𒐉 𒎙𒐉 ⊨◈
𒑱 𒐋 𒐌 𒐏𒁹 𒌋𒐌 𒌍𒐍 𒐐𒐈 𒐐𒐉 𒐉 𒎙𒈫 𒎙 𒌋𒐈 𒎙𒐎 𒎙 𒐐𒐋 𒐏𒐎 𒐐𒐊 𒌋𒈫 ⊨◈
𒑱 𒁹 𒌋𒐎 𒐏𒐍 𒌍𒐉 𒌍𒐈 𒌋𒐍 𒌋 𒌍𒐉 𒌋𒐌 𒐏𒐉 𒌍 𒌋𒐌 𒌍 𒎙 𒐐𒁹 𒐉 𒌋𒐎 𒌋𒈫
𒑱 𒐊 𒎙𒈫 𒎙𒐍 𒌋𒐌 𒐐𒐈 𒎙𒐎 𒎙𒐍 𒌍𒐍 𒈫 𒎙𒐎 𒎙𒐈 𒌋𒐍 𒐐𒐋 𒌍𒐈 𒐏𒐌 𒌍𒁹 𒌋𒈫
𒑱 𒐋 𒌍𒐌 𒌍𒐊 𒐏 𒎙𒐈 𒁹 𒐏𒈫 𒐍 𒐏𒐈 𒎙𒐈 𒐈 𒌋𒐋 𒌍𒁹 𒐐𒈫 𒌋𒐎 𒌋𒈫
𒑱 𒈫 𒌋𒐎 𒌍𒐉 𒐉 𒐐𒐍 𒌍𒁹 𒎙𒁹 𒌋𒐉 𒌍𒐍 𒁹 𒐐𒁹 𒐐𒐉 𒈫 𒎙𒐊 𒐎 𒐌 𒌋𒈫 ⊨◈
𒑱 𒐊 𒎙𒁹 𒌍𒐋 𒐏𒈫 𒐏 𒐏 𒐎 𒐐𒐈 𒌋𒐊 𒐏𒁹 𒎙𒐍 𒐏𒐉 𒐉 𒐐𒐎 𒌍𒁹 𒌋𒈫
𒑱 𒐏 𒐐𒐉 𒐐𒐍 𒌋𒐎 𒐏𒐉 𒐋 𒐐𒐍 𒐏𒐉 𒌋𒐈 𒌋𒐍 𒎙𒁹 𒌍 𒌍𒐈 𒌋𒈫 𒈫 𒌋𒐍 𒌋𒐉 𒎙𒐉
𒑱 𒐈 𒐏𒐈 𒌍𒐊 𒎙𒁹 𒎙𒁹 𒌋𒐍 𒐐𒐈 𒎙𒐌 𒌍𒐉 𒐊 𒐐𒐈 𒎙𒐋 𒌋𒐉 𒌋𒐈 𒐐𒁹 𒐏𒐉 𒌍𒐍 𒎙𒐉 ⊨◈
𒑱 𒐍 𒌍𒐉 𒌋𒐊 𒌋𒐉 𒐐𒐉 𒎙𒐊 𒎙𒐌 𒐐𒐍 𒐏𒈫 𒌍𒐍 𒐏𒈫 𒐏𒐎 𒐍 𒌍𒐎 𒌍𒐈 𒐌 𒌋𒈫 ⊨◈
𒑱 𒐋 𒐌 𒐏𒐊 𒐈 𒁹 𒐊 𒌍𒐎 𒐏 𒌍𒐍 𒌋𒈫 𒐏𒐎 𒐏𒐊 𒎙𒐊 𒐐𒁹 𒐏𒐉 𒌍𒐍 𒎙𒐉
𒑱 𒐏 𒌍𒈫 𒎙𒈫 𒎙𒐌 𒐉 𒐋 𒌍𒐌 𒎙𒐈 𒐐𒐍 𒌋𒐋 𒌋𒐋 𒎙 𒐏𒐍 𒌋𒈫 𒐐𒐉 𒐍 𒌍𒐍 𒎙𒐉 ⊨◈
𒑱 𒐈 𒐐𒐎 𒎙𒐋 𒐐 𒐏𒐈 𒌍𒈫 𒌍𒐎 𒐋 𒌍𒐉 𒐎 𒈫 𒐊 𒐋 𒎙𒐊 𒐐𒐊 𒌋𒈫
𒑱 𒐏 𒌋𒐋 𒐏𒁹 𒐌 𒐏𒐌 𒐌 𒌍𒐋 𒐈 𒌋𒐊 𒐏𒐋 𒌍𒁹 𒐈 𒌍𒐋 𒌋𒐊 𒌋𒈫 𒎙𒐈 𒈫 𒎙𒐉
𒑱 𒈫 𒐌 𒁹 𒐐𒐌 𒐏𒁹 𒌋𒐍 𒎙𒐍 𒎙𒐎 𒐐𒐋 𒐉 𒌍𒐌 𒐏𒐍 𒐐𒐍 𒐏𒐊 𒐌 𒌋𒈫
𒑱 𒐊 𒌋𒁹 𒐏𒐉 𒐏𒐎 𒌍𒐌 𒐏 𒈫 𒐏𒐉 𒌋𒐉 𒐏𒐋 𒎙 𒐊 𒐐𒐉 𒐏𒐍 𒐐𒐌 𒌍𒐋 ⊨◈
𒑱 𒐌 𒐋 𒌋𒐈 𒎙𒐈 𒎙𒁹 𒌍𒐎 𒐐𒐈 𒐏𒐈 𒐏𒐈 𒐋 𒌍𒁹 𒌍𒐎 𒐋 𒎙𒐍 𒁹 𒐐𒐊 𒌋𒈫 ⊨◈
𒑱 𒐊 𒐏𒐎 𒐏𒐈 𒐈 𒐏𒐈 𒐐𒐋 𒐐𒁹 𒐏𒐋 𒎙𒐋 𒌋𒐋 𒐏𒐎 𒈫 𒎙𒐎 𒐉 𒐌 𒐏 𒐏𒐍
𒑱 𒈫 𒐐𒁹 𒐋 𒌍𒐍 𒌍𒐊 𒎙𒐈 𒐐𒐍 𒎙𒐈 𒐋 𒈫 𒌋𒐈 𒌍 𒎙𒐈 𒐐𒐎 𒐉 𒐏𒈫 𒌋𒐉 𒎙𒐉
𒑱 𒐈 𒐏𒐊 𒐏 𒐐𒁹 𒈫 𒎙𒐊 𒎙𒈫 𒐉 𒌋𒐊 𒌍𒈫 𒎙𒐉 𒐏𒈫 𒐐𒁹 𒌋𒐊 𒐐 𒎙𒐉
𒑱 𒐎 𒌋𒐎 𒐋 𒎙𒐊 𒌍𒐉 𒐏𒐌 𒐎 𒌋𒐍 𒌍𒈫 𒐏 𒌍 𒐐 𒐐𒐋 𒎙𒐎 𒌋𒁹 𒈫 𒎙𒐉 ⊨◈
𒑱 𒐏 𒐎 𒌋𒐉 𒐐𒐉 𒌍𒐍 𒎙𒐍 𒐐𒐌 𒐌 𒁹 𒌋𒈫 𒐐𒐋 𒎙𒐌 𒐏 𒐊 𒐐𒐋 𒌍𒐎 𒌍𒐈 𒐌 𒌋𒈫
𒑱 𒐎 𒌋𒁹 𒐌 𒌍𒐋 𒐏𒐎 𒐏𒐎 𒎙𒐌 𒐏𒐉 𒌋𒈫 𒌍𒐈 𒌋𒐍 𒐏𒐈 𒐐𒐊 𒌋 𒌍𒐎 𒎙𒁹 𒌍𒐋 ⊨◈
𒑱 𒈫 𒌋𒐉 𒎙𒐊 𒌋𒐍 𒎙 𒎙𒐋 𒐐𒐈 𒌋𒐋 𒎙𒐊 𒐈 𒐐 𒌍𒐉 𒌍 𒎙𒈫 𒎙𒐎 𒐐 𒎙𒐉 ⊨◈
𒑱 𒐏 𒌍𒐉 𒎙𒐊 𒎙𒐉 𒎙𒐈 𒐐 𒁹 𒐏𒈫 𒐏𒐈 𒌋𒐎 𒐏𒐉 𒐐𒐌 𒐈 𒌋𒁹 𒎙𒐎 𒐏𒐉 𒌍𒐋 𒎙𒐉 ⊨◈
𒑱 𒐋 𒐐𒐌 𒐐 𒎙𒐊 𒌋𒐊 𒌋𒐊 𒐏𒐍 𒎙𒐈 𒐏𒁹 𒌋𒁹 𒐐𒐉 𒎙𒈫 𒌋𒐉 𒐏 𒐐𒐌 𒌍𒐋 ⊨◈
𒑱 𒐍 𒎙𒐉 𒐍 𒎙𒐊 𒎙𒐉 𒐏 𒐐𒐎 𒌍𒐎 𒎙 𒐏𒐋 𒌋𒐍 𒐎 𒐏𒐌 𒌍𒁹 𒌋𒈫 ⊨◈
𒑱 𒐏 𒐈 𒎙𒁹 𒐈 𒐏 𒌍𒐎 𒌋𒐌 𒐉 𒌍𒐈 𒎙 𒐏𒐍 𒐏 𒌍𒐍 𒐐𒐊 𒐏𒈫 𒌋 𒐐𒐋 𒌍𒐍 𒎙𒐉
𒑱 𒐌 𒌍𒐌 𒐐 𒐐𒐉 𒎙𒐍 𒌍𒈫 𒐍 𒌋𒈫 𒌍𒐌 𒌍𒐎 𒐐𒐋 𒌋𒁹 𒌋𒐊 𒌍𒁹 𒐐𒐍 𒐉 𒐏𒐍
𒑱 𒐌 𒌋𒐍 𒌋𒈫 𒌋𒈫 𒌋𒐊 𒐌 𒎙𒐌 𒐏𒐎 𒐏𒐍 𒐏𒐉 𒎙𒐉 𒌋𒐎 𒌍𒐌 𒐏𒐍 𒌋𒐌 𒌋𒐋 𒐏𒐍
𒑱 𒐋 𒐏𒈫 𒈫 𒁹 𒐐𒐊 𒌍𒐉 𒐐𒈫 𒐏𒐊 𒎙𒐎 𒎙𒐋 𒐏𒐌 𒐉 𒌍𒐋 𒐏𒈫 𒌍𒐌 𒎙𒐋 𒎙𒐉 ⊨◈
𒑱 𒈫 𒐎 𒌋𒈫 𒌋𒐋 𒌋𒐍 𒎙𒈫 𒌍𒐌 𒐏𒐉 𒐏𒐈 𒌋𒐌 𒎙𒐎 𒌍 𒌍𒁹 𒐐𒐉 𒌍𒐌 𒎙𒐋 𒎙𒐉 ⊨◈
𒑱 𒐌 𒌍𒐋 𒌍𒈫 𒐉 𒌋𒐈 𒐈 𒎙𒈫 𒐏𒐍 𒎙𒐌 𒎙𒐉 𒐐𒐈 𒌋𒐉 𒎙𒐋 𒐐𒐎 𒐏𒈫 𒐏𒐈 𒌋𒈫 ⊨◈
𒑱 𒐋 𒎙𒐋 𒌋𒐋 𒎙𒈫 𒐐𒈫 𒎙𒐊 𒐏𒐊 𒎙𒐈 𒌍 𒌍𒐍 𒐏𒐋 𒐏𒁹 𒎙𒐈 𒌋𒐊 𒐉 𒌋𒐎 𒌋𒈫 ⊨◈
𒑱 𒈫 𒌋𒐎 𒌍𒐊 𒌍𒐉 𒎙𒐎 𒐎 𒐏𒐌 𒐐𒐌 𒌍𒐍 𒐐𒐌 𒌋𒐎 𒐐𒐈 𒎙𒐊 𒌍𒐈 𒌋𒐍 𒐏𒐈 𒌋𒈫 ⊨◈
𒑱 𒐌 𒌋𒐍 𒐌 𒎙𒐌 𒎙𒐎 𒐐𒈫 𒐐𒐊 𒌍𒐎 𒌋𒁹 𒐉 𒌍𒐌 𒐏𒐍 𒐐𒐍 𒐏𒐊 𒐌 𒌋𒈫
𒑱 𒈫 𒎙𒐉 𒎙𒐌 𒐐𒐎 𒎙𒐈 𒐐𒐎 𒐏𒐌 𒐐𒐊 𒐐 𒌋𒐎 𒐏 𒐏𒐍 𒌋𒐋 𒎙𒐉 𒐏𒐈 𒐏𒐋 𒌍𒐈 𒌍𒐋 ⊨◈
𒑱 𒐏 𒌍𒐈 𒌍𒐈 𒌍𒐍 𒐍 𒌋𒐊 𒌍𒐌 𒐐𒐉 𒐏𒐌 𒌋𒐋 𒐏 𒐐𒁹 𒎙 𒐏𒐈 𒎙𒐍 𒐌 𒐏 𒐏𒐍 ⊨◈
𒑱 𒐌 𒐊 𒌋𒁹 𒐉 𒎙𒈫 𒐐𒐎 𒎙𒐊 𒎙𒁹 𒐐𒐋 𒐌 𒌋 𒌋𒁹 𒐉 𒐏 𒐏𒐋 𒐉 𒐏𒐍
𒑱 𒐏 𒌍 𒌋𒁹 𒐐𒐈 𒐍 𒐐𒐊 𒌍𒐋 𒌋𒐉 𒌋𒐊 𒐏𒐋 𒐐𒐌 𒐈 𒌍𒐋 𒐐𒐈 𒐐𒐉 𒐏𒐍 𒐐𒐌 𒌍𒐋
𒑱 𒐋 𒁹 𒐐𒐈 𒌍𒐊 𒐊 𒌍𒐍 𒎙𒈫 𒐏𒐉 𒐈 𒐏𒁹 𒌋𒐈 𒌋𒐊 𒐋 𒌍𒐌 𒎙𒐋 𒎙𒐉
𒑱 𒐉 𒐐𒐊 𒎙𒐍 𒐏𒁹 𒐈 𒐏𒐈 𒐏 𒐋 𒐏𒈫 𒐊 𒎙𒐎 𒐐𒐈 𒐏𒐊 𒎙𒈫 𒌋 𒌍𒐈 𒌍𒐋
𒑱 𒐌 𒐎 𒌋𒐈 𒐐𒁹 𒐐 𒐎 𒁹 𒐐𒐉 𒈫 𒌋𒐋 𒌍𒐊 𒐋 𒐎 𒌍𒐋
𒑱 𒐏 𒐐𒐌 𒌋𒐊 𒐋 𒌍𒐊 𒐏𒁹 𒌋𒐌 𒐏 𒌋𒐍 𒁹 𒐋 𒐏𒐊 𒌋𒁹 𒐐𒐎 𒌍𒈫 𒎙𒁹 𒐌 𒌋𒈫
𒑱 𒐌 𒎙𒐍 𒌋𒐊 𒐏𒐉 𒌍𒐊 𒐈 𒌍𒐈 𒌍𒈫 𒐏𒐍 𒌋𒐋 𒎙𒐊 𒐏𒐎 𒁹 𒌍𒁹 𒐏 𒎙𒐍 𒐏𒐍 ⊨◈
𒑱 𒐌 𒐏 𒐏𒁹 𒌍𒐈 𒌍𒐍 𒐐𒁹 𒌋𒐋 𒌋𒐋 𒐐𒐉 𒌋𒐋 𒐏𒐈 𒌋𒐉 𒐌 𒌋 𒐐 𒐐𒈫 𒐏𒐍 ⊨◈
𒑱 𒐉 𒐏 𒌍𒈫 𒐎 𒐐𒐍 𒌋𒐍 𒐋 𒐋 𒌍 𒌋𒐋 𒐐 𒐐𒐍 𒌍𒐋 𒎙𒁹 𒐐𒐈 𒌋𒐋 𒐏𒐍 ⊨◈

[illegible]

[illegible]

[illegible]

[illegible]

𒐉 𒎙𒐌𒑲𒑲	𒈦	𒅁𒌓𒁹𒑲𒌋𒐌𒁹𒄭	𒑲
𒌍𒐌 𒐏𒁹𒑲𒑲	𒈦	𒅁𒌓𒁹𒑲𒌋𒐌𒑲𒄭	𒑲
𒐐𒐈 𒎙𒐌𒑲𒑲	𒈦	𒅁𒌓𒁹𒑲𒌋𒐌𒁹𒄭	𒑲
𒐐𒐋𒑲𒑲	𒈦	𒅁𒌓𒁹𒑲𒎙𒑲𒄭	𒑲
𒁹 𒎙𒐋𒑲𒑲	𒈦	𒅁𒌓𒁹𒑲𒎙 𒁹𒄭	𒑲
𒈫	𒂍𒆠	𒈾𒄀	𒑲
𒁹	𒁾𒊬𒆳𒁲𒆕	𒈾𒄀	𒌋

colophon