DHQ: Digital Humanities Quarterly
2023
Volume 17 Number 2

ᐊᒐᐦᑭᐯᐦᐃᑲᓇ ᒫᒥᑐᓀᔨᐦᐃᒋᑲᓂᐦᑳᓂᕽ | *acahkipehikana mâmitoneyihicikanihkânihk* | Programming with Cree# and Ancestral Code: Nehiyawewin Spirit Markings in an Artificial Brain

Jon Corbett <joncorbett_at_live_dot_ca>, Simon Fraser University https://orcid.org/0000-0002-3721-1640

Abstract

In this article, I discuss my project “Ancestral Code”, which consists of an *integrated development environment* (IDE) and the Nehiyaw (Plains Cree) based programming languages called Cree# (pronounced: Cree-Sharp) and ᐊᒋᒧ (*âcimow*). These languages developed in response to western perspectives on human-computer relationships, which I challenge and reframe in Nehiyaw/Indigenous contexts.

Introduction

Nohkompan,[1] my paternal grandmother, was Nehiyaw (Cree) and Saulteaux (Chippewa). After her passing and understanding the matriarchal nature of many Cree peoples, I found myself looking at the Nehiyaw culture of her mother, *Nitâpân* ,[2] for inspiration for my creative works. In 2014, I attempted to use *Nehiyawewin* (the Cree language) words as variables using the Unified Canadian Aboriginal Syllabics [Unicode Inc. 2021]. My early attempts were instantly problematic because the development environments (IDEs) I used would render the syllabics, like “ᑖ”, as empty boxes “☒”. So I searched for other solutions. However, at that time, the platforms I was most familiar with, such as Processing, openFrameworks, Java, and C#, all shared this same “problem.” This experience sparked my desire to bring Nehiyawewin, in its syllabic form, into being as a programming language. In this article, I present the physical and abstract challenges of developing my Ancestral Code digital toolkit that allows a programmer to insert code blocks into transcribed Indigenous stories that, once executed, produce a generative visual narrative of the transcribed story. But for you to better understand this desire’s importance, it is customary in my heritage to first introduce myself. 1

I identify as Nehiyaw-Métis and belong to the Métis Nation of Alberta. The Métis in Canada are federally recognized as one of the three Indigenous groups of peoples, the other two being First Nations and Inuit. The Métis have been historically referred to as half-bloods or, more derogatorily, half-breeds. We are a people that came to be from the blending of cultures and traditions of the first European visitors with the numerous Indigenous peoples of the “new world.” Many of these first visitors were not men seeking to settle; they were merely coming to seek a better living as fur traders. They were nomadic and travelled extensively, with many intending to return to their respective homelands when they retired. 2

Nevertheless, these traders married into First Nation and Inuit tribal communities, often for financially strategic reasons. My Great-Grandfather five times over was a historically notable English fur trader named John Sayer and was one of these first visitors. These early relationships between European visitors and Indigenous peoples continued through generations, with subsequent mixed-racial generations raised with a blend of European knowledge and their original Indigenous languages and traditions. This concrescence of cultures and languages is how the Métis developed into a unique culture that privileges the cultural teachings of their respective Indigenous heritages without entirely rejecting their European roots. 3

My research as a Métis scholar and digital media artist has since evolved from using computers and programming as *tools* to generate my artwork to viewing computers as *animate creatures*, digital representations of my Indigenous heritage. Subsequently, I see computer languages as digital extensions of Nehiyaw storywork and ceremonies that reflect the epistemological, ontological, and axiological concerns of my Nehiyaw beliefs and practices. My perspectives on computer programming critique the prevalent use of English in coding languages and the reflection of settler/colonial perspectives in their design. Though I recognize that computer programming languages, like most technologies, are constantly evolving and changing, I maintain that they are also seemingly immutable and typically manifest from a "historically-essential … colonial impulse" [Ali 2014]. Most notably, the culture(s) from which modern programming languages and practices grow "[come] with significant cultural baggage" [Heaven 2013]. Through my own visual media art explorations with the popular new-media-art programming development environment "Processing", I recognized significant challenges to programming in anything other than English. 4

The detrimental legacy of colonial practices on the lives of Indigenous people is well documented, from forced displacement from traditional homelands to attempted erasure of culture, language, and practices [Gradual Civilization Act 1857] [Venne 1981] [Milloy 2008] [Milloy 2017]. As citizens of a globally connected techno-culture, we perpetuate this erasure by embracing a technicism that incorporates a blind acceptance of technologies and the cultural systems from which they are derived as necessary to engage with one another in our digital lives. Yet, critical theories of technologies are continually demonstrating that digital technologies are neither "socially or culturally neutral" [Garneau 2018] nor are they "determinist, but rather [sites] of social struggle" [Warschauer 1998]. Despite this technicism, navigating the modern technology ecosystem remains vital to Indigenous peoples, who actively utilize these new computing domains for "survivance" [Vizenor 2008] and (re)establish both individual and community identities from these technological relationships and their ancestral cultures [Ormond-Parker et al 2013]. 5

By viewing computing through a lens of my Indigenous heritage, my perceptions of computer languages and programming have been dramatically altered. My awareness of programming contexts informed by Nehiyaw language and cultural practices has opened new understandings of how programming languages can facilitate a greater sense of personal and digital identity and cultural belonging that go far beyond the purely functional operations of programming. These experiences are at the heart of what has become my Ancestral Code project. 6

Ancestral Code is a wholistic programming environment built upon my own Indigenous computing design theory. It consists of both hardware, in the form of a specialized keyboard for typing Nehiyawewin syllabics, and software in the forms of a programming IDE and a multi-form programming language. The programming language component of Ancestral Code that is the focus of this article aims explicitly to integrate a "wholistic" [Absolon 2010] braiding of Nehiyaw language and cultural practices within its design. I use *wholistic* here instead of holistic because Indigenous learning interconnects all aspects of being that include mind, body, emotion, and spirit and is not merely referring to a "whole" as a sum of parts. In particular, Ancestral Code is my vision of how Nehiyaw culture and language can be utilized as a primary interface for computer programming and computing design, uniting the values/benefits of Nehiyaw perspectives with western programming practices. To begin this journey, I will first provide a brief background of what I frame as an Indigenous computing design theory, describing the importance of this framework when approaching technology and computing design development with an Indigenous focus. Next, I describe how this design influences a unique perspective that views "Code as Story." And then, I delve into the challenges (and related solutions) to my ongoing efforts to bring the Ancestral Code platform for programming in Nehiyawewin to Nehiyaw communities. I hope this platform can open new opportunities for heritage language use in our modern technological context and further foster Indigenous cultural maintenance. Finally, I surmise and summarize how this work can be used or reproduced by other Indigenous cultures with similar cultural perspectives and language construction. 7

Indigenous Wholistic Computing Design Theory

Computing system design theories continue to evolve and have been described by or compared to a wide range of systems such as mechanical [Worden, Staszewski, and Hensman 2011], biological [Babaoglu et al 2006] [Benenson 2012], and hierarchical [Kleinrock and Kamoun 1980] [Abd-El-Barr 2009], to name just a few. Similarly, the past decade has also seen increased interest in postcolonial computing and colonialism as an influencing or embedded aspect of 8

technological architectures [Irani and Dourish 2009] [Irani et al 2010] [Merritt and Bardzell 2011] [Dourish and Mainwaring 2012] [Philip, Irani, and Dourish 2012] [Abdelnour-Nocera, Clemmensen, and Masaaki 2013] [Ali 2014]. In developing an Indigenous computing theory, I consider common concerns from a general pan-Indigenous perspective, recognizing that research by Indigenous scholars commonly employs cultural practices and knowledge such as ceremony [Wilson 2008] [Cormier and Ray 2018], wholistic practices [Absolon 2010], honouring of oral knowledges [Thomas 2005], and community engagement [Madden, Higgins, and Korteweg 2013].

From these general characteristics I also used Nehiyaw-specific knowledge to help shape my own approach of establishing an Indigenous computing framework. The English term *Cree* is an anglicized form of the French word *cri* meaning to shout or cry aloud, and is how early European settlers came to name the Cree/Nehiyaw people. However, I do not refer to myself as Cree-Métis. I am Nehiyaw-Métis. *Nehiyaw*, in my heritage language, translates as “four-bodied” or “four-spirit” people. We see ourselves as a people composed of four “bodies:” the physical, mental, emotional, and spirit. Therefore, specific to Nehiyaw culture, I use spiritual teachings of the “four-bodies” framework to inform, describe, and highlight the aspects of computing that are inherently Nehiyaw and reflect Nehiyaw understandings and knowledge development. 9

Furthermore, each of these bodies has numerous cultural teachings that exist as living (oral) histories highlighting the cultural significance of their meanings and relationships between humans and the world. In other words, *kinehiyâwiwininaw nehiyawewin / The Cree Language is our Identity* [Wolfart and Ahenakew 1993]. The Cree language, *Nehiyawewin*, is an intricately woven fabric that unites our life, being, and identity, across generations of knowledge and embodies spiritual and sacred knowledge that spans thousands of years. Because Nehiyaw language is so integral to Nehiyaw life, understanding user-computer relationships in a Nehiyaw context requires redefining philosophical understandings of computer system design using Nehiyaw terms. Therefore, using Nehiyawewin in my theoretical designs serves as a ceremonial healing practice in addition to its functional role of creating computational instructions. I find these considerations crucial to the design of computing technologies for Nehiyaw people as they re-envision the computer as a member of the community and an extension of one’s cultural identity. 10

Therefore, placing culture as the driving force behind the development of an *Indigenous Wholistic Computing Design Theory* can, and will, often be at odds with western philosophies on computer function and design. For instance, cultural attitudes towards concepts of time, order/sequencing, and efficiency can be radically different from their computational definitions. For example, I wrote a program to digitally-bead portraits of my family. The first version of this program used a basic loop to place pixel-beads on the screen in a sequence of rows, which it did in a left-to-right fashion. From a computational perspective, this loop was simple and efficient. However, the original computational instructions did not reflect my physical action of beading if I were to bead this image by hand. Nor did it capture the cultural significance of the continuous thread that connects each bead. There is a special meaning in the unbroken thread. As a result, I reformatted my code to reflect my Métis beading practice. The new code created alternating rows of digital beadwork that progress from left-to-right, then right-to-left in a continuous unbroken stream. This updated code better represents my physical process instead of code that created patterns that flowed in one direction, from left-to-right, as multiple individual lines instead of a continuous loop. The *best* or *most efficient* code fractures the image’s construction, resulting in digitally rendered images conforming to the technological tools’ design and language, favouring efficiency over cultural focus. 11

Experiences such as these altered my perspectives on general computing design theories and stimulated my desire to investigate computing design from Indigenous wholistic perspectives. Shifting the source of computing design from one driven by systemic operations to one that is human and cultural requires a certain degree of re-prioritization. I opted to expand existing system design theories by creating an *Indigenous Wholistic Computing Design Theory*. By identifying and privileging Indigenous and Nehiyaw-specific perspectives within existing computational environments, I aim to reevaluate what aspects of computing design are favoured over others. 12

In more concrete terms, systems design processes are vital because they create a clear overview intended to guide the actual development of a given product, whether it is hardware or software. In an Indigenously informed design process, this overview is still as essential but follows principles that promote [Indigenous] community collaboration and 13

engagement over the tools and technologies that will be used; emphasizes interrelationships between components by establishing connections between Indigenous lived experience and the involved digital artifacts and their behaviours; and is open and flexible, or even unconcerned, with time and timelines in solution development. In such a theoretical framework western principles like *Gestalt grouping* or *Fitt's Law* may not be applicable because the effects of cultural knowledge on a design may alter how components are typically created, or how they behave and interact with one another in comparison to systems focused directly at efficiency concerns like those found in data categorization, code flow, and execution and processing times.

Indigenous Story as Code

What is coding if not a story? At a basic level, a story consists of five essential components: character(s), setting, plot, conflict, and resolution. Scholar Annette Vee, known for her study of composition and rhetoric in computer programming literacy, compares narrative writing with computer programming stating that they "are not the same thing [but] have a lot in common and can even merge into each other" [Vee 2017]. The relationship between story and code is particularly evident in esoteric computer languages such as "Inform 7" [Aikin 2009] and "Shakespeare" [Hasselström and Åslund 2001], where code is literally formatted as stories. Moreover, I argue that "normal" programming languages like C/C++, C#, and Java also share storytelling's main elements despite their visual structure, notation, syntax, and semantic constructions. In "normal" programming languages, these story elements exist in more abstract contexts where variables represent the "characters", the "setting" is the programming environment, and the "plot" is the program's function as it operates up until its "resolution" or termination. I first learned how to program in elementary school in 1979 using Applesoft BASIC. My teacher explained BASIC's syntax using story examples. I was so enthralled with the idea of being able to represent the world around me using computer code that I started writing my journal entries in language arts class as BASIC code. For me, coding is very much a form of storytelling. 14

In creating the Ancestral Code programming languages "Cree#" and "ᐊᑎᒼ" (*âcimow*)[3], my vision arrived as a genuine dream. In this dream, I was sitting in a community lodge (i.e., tipi), listening to an Elder tell a story. In this story, an image of *Wîsahkecâhk*, sometimes known as the "Trickster Raven" in Nehiyaw cultural teachings, was animated on the wall behind him. *Wîsahkecâhk* was puppeted through the Elder's words, while a syllabic subtitled transcription of the story ran beneath the imagery. This dream revealed a path for me. It led me to understand that this generative virtual landscape was a relationship between Nehiyawewin and the computer, and though this landscape was digital, it was still a "place." And here, in this place, I experienced my Ancestral Code project as each of the four "spirits:" The physical experience of seeing and being present; the mental experience of processing and parsing the story; the emotional experience of being attached to an Elder and *feeling* the story; and the spiritual experience of the dream state itself. 15

Furthermore, in my dream, the syllabic transcription was an instruction-set that manipulated the vision of *Wîsahkecâhk*, thereby exposing me to the computing code. Thus, though this is a brief origin story of my Ancestral Code project, I have interpreted the role of the four spirits in furthering its creation as a critical component of its development. As a result, I have found new ways of applying Nehiyaw cultural teachings to my computer coding practice. 16

Challenges and Solutions

Though I expected to encounter obstacles in this project, I did not honestly foresee how deep into my coding practice and theory I would be required to go to design the Ancestral Code programming language. In this section, I trace out each of the challenges with working the Nehiyawewin orthography — including its use in the software and hardware, how I incorporated Nehiyaw language structures into the programming language, and how I see culture as being crucial to the programming languages I developed. 17

Orthography

The Nehiyawewin orthography, called ᐊᑌᐦᐸᓇᐦᐃᐧᑲᓇ or *acahkasinahikana*, literally translates into English as "spirit markers." Nehiyawewin also has a *Standard Roman Orthographic* (SRO) writing system in which the language can be written with the standard Latin alphabet to aid English speakers in pronunciation. Though Nehiyawewin learners 18

frequently use SRO, many Nehiyaw first language speakers consider SRO an accommodation or adoption of western processes. They also feel this system voids much of the cultural significance and meaning attached to the “spirit markers”. Though I prefer to use Nehiyawewin spirit markers, most of the Nehiyawewin in this article is written in SRO form for easier readability by English readers. I have included a glossary at the end of this article with definitions and a pronunciation guide using the International Phonetic Alphabet (IPA) phonetic notation system.

19

My original desire to use *acahkasinahikana* exclusively in Ancestral Code was a conscious decision to combat the “accommodation” of English language constructions and representations. However, in investigating my needs as a programmer, these language-based concerns revealed that support for both SRO and *acahkasinahikana* are required to support Nehiyawewin language learners that may come from different dialects or communities. For this reason, programming in Ancestral Code can be performed in its Romanized form that I call Cree# (pronounced Cree-sharp) or its syllabic form ᐊᐞᑎᓄᐣ (*âcimow*). The Ancestral Code IDE allows the programmer to switch between the two styles, where Cree# accommodates a certain level of familiar structure to the C# programming language, separating it from the more story-like narrative structure of ᐊᐞᑎᓄᐣ (*âcimow*). Switching between these modes provides unique alternatives to experiencing written Nehiyawewin, where representing code in multiple cultural forms can be done without necessarily favouring one over the other. Furthermore, the IDE development aspect revealed a couple, albeit minor, concerns, namely:

1. How to use the Unified Canadian Aboriginal Syllabics block of the Unicode Standard, that is, what font family/typeface to use for coding with syllabics.
2. How to address coding without, or with minimal, punctuation and numeracy.

Software: *acahkasinahikana* for coding

20

Using Unicode characters in modern programming environments is possible if the computing system supports the desired character sets. The Unified Canadian Aboriginal Syllabics block of the Unicode Standard occupies 640 code positions from 1400hex to 167Fhex. It includes orthographic glyphs for Blackfoot, Carrier, Cree, Dene, Inuktitut, Ojibwe, and other Canadian Athabascan languages. Unfortunately, the coding environment is often the main obstacle in using Unicode characters, especially as variable names and, more problematically, as keywords or reserved words. Typically, this lack of support is simply because the environment often uses a pre-installed font that does not contain glyph(s) in the desired code points of the font file. By default, the coding environments I am familiar with, such as *Visual Studio* and *Processing*, use a fixed-width font/typeface such as Consolas, Courier, or Monospace. These fonts are common to Windows OS systems but do not have the Canadian Aboriginal Syllabics block of glyphs. Until recently, changing this font was not easy. Although, as of the writing of this article, the settings or preferences support in some IDEs is open to changing the default font, some still limit the fonts that can be used. There also are limited fonts suitable for programming using the Unicode Canadian Aboriginal Syllabics, and I argue there are currently no suitable fonts. Though it is theoretically possible to use syllabics as variable names in those IDEs, reliance on English programmatic tokens remains. The question then becomes, which font(s) can or should be used for programming with *acahkasinahikana*?

Software: *acahkasinahikana* fonts

21

Though this may sound like a trivial topic from a western perspective, as it truly has little to do with actual coding and more to do with aesthetic preference, my critical reflection on coding practice makes this a necessary point of discussion. Remember that Nehiyaw *acahkasinahikana* are visual representations and extensions of being. They are called *spirit markers* for a reason. So, their representation in the computer must be treated with equal consideration.

22

Most programmers would agree that fixed-width or monospaced typefaces are ubiquitous in coding because they align very well in rows and columns and generally provide a greater distinction between similarly shaped characters like “0, o, O,” narrow characters like “I, l, i,” as well as providing more space for syntactical and operational characters “(, {, [,!” [Ardley 2014]. Combining this respect for the visual aspects of language with the need for a monospaced coding font reveals two challenges. The first challenge is that, to my knowledge, only one monospaced font purposefully includes the Canadian Aboriginal Syllabics blocks, Everson Mono.[4] The second challenge is that the physical structure of

syllabics makes using non-fixed-width fonts challenging to navigate as code. The visual structure of the major glyphs (i.e., the glyphs representing a consonant and vowel pair) consists of reflected orientations of a single shape (ex. ᒥ ᒣ ᒪ ᒧ). The resulting printed text is fairly consistently sized but is visually similar to coding in all English caps. This is not necessarily a negative, as my original programs written in BASIC were done in all caps. But a more significant point of consideration is how a culture perceives the visual design of its language. For example, in developing typefaces for Canadian syllabics, Canadian typography designer Kevin King worked directly with Indigenous communities to ensure their languages were written in the respective community's preferred style. He notes that "to ignore this would result in a text that was neither culturally appropriate for local readers nor able to convey adequately the meaning and atmosphere of the text for that readership" [King 2022]. Adding to these, the limited number of monospaced fonts that support Canadian syllabic glyphs encouraged me to develop a new code-friendly font to address these obstacles. This font (Figure 1), I tentatively named "AC Mono," was constructed using my wholistic design framework, and my choices in the font design process are consciously aware of Indigenous visual aesthetics. I want to note here that my design of this font was considered from a more pan-Indigenous perspective and not specific to Nehiyaw culture. This is because the Unified Canadian Aboriginal Syllabics is not Nehiyaw specific – it represents glyphs from various North American Indigenous languages. For example, as seen in the syllabic T-series glyphs like "tâ" (ᑦ) and "te" (ᑌ), I used thicker lines, an emphasis on fluidity in the curves, and rounded features on the start and end points of the strokes. These organic geometries and gestures are found in numerous Indigenous art traditions in North America, including the ovoid traditions of Coast-Salish artforms, Inuit bone and stone carving, and Métis beadwork.

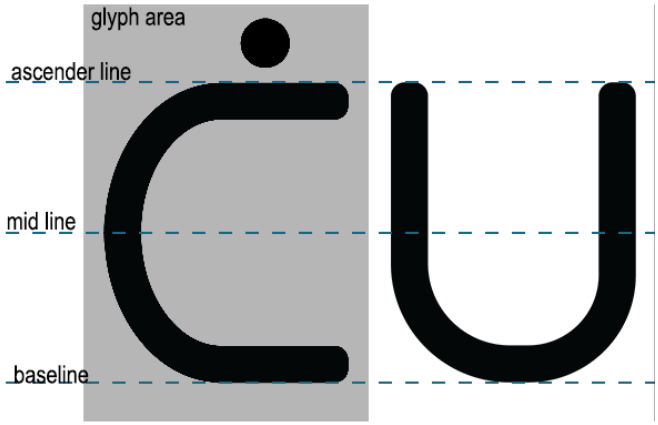

Figure 1. AC Mono Font glyphs for "tâ" (ᑦ) and "te" (ᑌ). Image by Jon Corbett.

In the construction of AC Mono, I used the typefaces of Everson Mono and Consolas only as sizing templates, building my resulting font with the stroke weight of Consolas and the mildly rounded stroke end of Everson Mono. Of particular note, syllabics have no differing case structures. Therefore, no glyphs need to accommodate space for descenders that extend below the baseline. The removal of the descender area results in a noticeably reduced distance between the glyph baseline and the edge of the glyph-space in the AC Mono font compared to ASCII/English fonts (Figure 2), allowing for a more consistent line height, providing better options for vertical spacing between rows when coding.

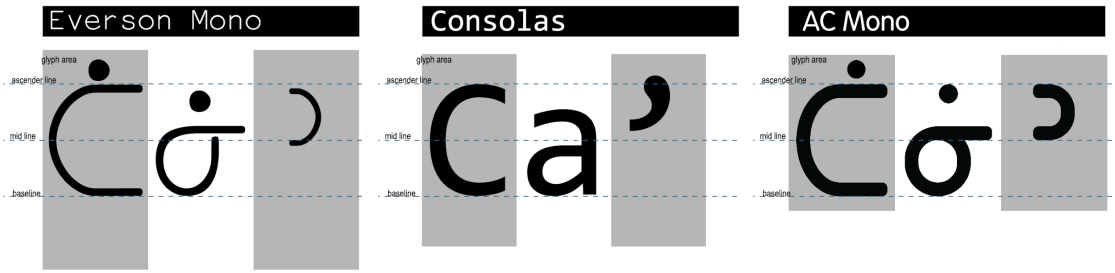

Figure 2. Font design comparison between Everson Mono, Consolas, and AC Mono, the English glyphs selected here are chosen for their visual similarity to the Nehiyawewin syllabics, they are not representative of English pronunciations or linguistic function.

Software: *acahkasinahikana* punctuation and numeracy

A more significant challenge is how to support specific programming tasks using limited punctuation and, preferably, an alternative numeric symbology. Programming nomenclature and notation have had a unique evolution and, as Arawjo notes, have developed in “concert and conflict with discretizing infrastructure[s]” [Arawjo 2020]. Without getting into a long history lesson on programmatic notation, I have seen how my own programming practice evolved from coding in BASIC, using very little punctuation except the double quote and round brackets, to today, where I predominantly use C# and Java that, use nearly every non-numeric and non-alpha character on the standard keyboard. In today’s programming culture, it is rare to see a programming language not use every symbol and character on the keyboard. Notation marks that currently have meaning and function in many modern programming languages can be problematic when developing solutions for Indigenous orthographies that traditionally do not use these marks. 24

As I mentioned earlier, the only punctuation commonly found in Nehiyawewin text is the full stop syllabic that looks like a lowercase “x” but is actually U+166E (᙮); and the hyphen “-” most often used to separate certain morphemes or break up very long sentence-word constructions when reading. My understanding of this lack of punctuation stems from my understanding of Nehiyawewin morpheme structures that provide the necessary context and grammatical positioning in its word construction, making punctuation usage unnecessary, except to indicate the end of thought or discussion [Wolfart 1973] [Okimasis and Wolvengrey 2008]. 25

Another point of consideration between western culture and Nehiyaw culture is numeracy. Much of the world uses the ten symbols of the Hindu-Arabic decimal number system to represent numbers. Though Nehiyaw numeracy has become pretty much forgotten in favour of Hindu-Arabic numerals, Nehiyaw culture did develop a way to represent numbers using the syllabic glyphs (Figure 4). Furthermore, from a language perspective, Nehiyaw culture has developed certain relationships to numbers and numeric meanings that are often associated with Nehiyaw ontologies, as exemplified previously in my explanation of the meaning of Nehiyaw as the four-bodied people. 26

Punctuation Symbols:

A simple but effective example of a *traditional* programming style that would execute a print-to-screen command might look like this: 27

```
print("hello world");
```

In this example, the parentheses mark the start and end of the function contents, and the double quotes mark the entry and exit points of the text to render to the screen.

So how would a programmer make this indication if there are no brackets or quotation marks? Another common symbol in modern programming languages is the semi-colon (;) as an instruction separator or terminator. Again, how does a programmer indicate where to terminate an instruction without this mark? These orthographic punctuation devices have become so commonplace in modern programming that it is difficult to envision not having access to them. It is precisely 28

these kinds of practices that I have attempted to overcome in developing Ancestral Code. My intent here is not a need to use minimal punctuation. Instead, it is a resistance to adopting marks and practices that belong to other languages. To that end, one of the first steps I took was to inventory the Nehiyaw syllabary and functional language glyphs. I then established a list of necessary programmatic functions and decided on the syllabic symbols for my programming language.

29 To start this investigation, I started with the most straightforward programming language I know — BASIC. I used a variety of dialects of BASIC for nearly fifteen years, and upon reflection, it was one of the languages that I could use with the least amount of punctuation. BASIC also provides a good starting point for just "reading" code as if it were a recipe or short story. It also helped me identify the bare necessities I felt Ancestral Code required. In the end, I came to the following general determinations to make language development easier. These initial determinations were primarily based on my *Indigenous Computing Design*'s resistance to western computing reliance on non-alpha characters:

- The Unified Canadian Aboriginal Syllabics block of the Unicode Standard currently contains 640 symbols. However, my dialect of Plains Cree only uses 107 syllabics from a total block of 134 glyphs that *could* be used in Nehiyaw speech. So the remaining 533 symbols from the syllabary became potential candidates to fulfill those programmatic roles where western punctuation is usually employed.
- Ancestral Code will be line-driven. In other words, the linefeed/carriage return is the primary instruction terminator. Using the linefeed, makes parsing of instructions easier.
- The full stop symbol (`x`) is used to terminate *and* exit code (especially within a loop). Being the only punctuation mark, its function and use needed to make sense in the narrative flow of the source code.
- Language-based reserved words are used to mark code start and end positions instead of braces or other symbols. For example, in Visual Basic `IF` and `END IF` mark the start and endpoints of a conditional code block; similarly, *sipiy* and *âniskôsîpiy* perform the same roles in Ancestral Code.
- In Ancestral Code, Nehiyawewin syllabic-numerals replace Hindu-Arabic numbers. Hindu-Arabic numerals will be allowed, but only in SRO mode, and a built-in calculator will convert Hindu-Arabic numbers to their appropriate syllabic representations when in syllabic mode.[5]

30 With this software side of the orthography resolved, I looked at the next obstacle: how should the user input code?

Hardware: the *acahkasinahikana* keyboard

31 The history of print cultures has led to privileging western "stories, voices, and values" [Risam 2018, p. 89], and modern coding cultures and computer language development have naturally adopted. Though Nehiyawewin can be typed on "standard" keyboard layouts using a Romanized orthography, this seemingly innocuous accommodation is probably the number one contributor to the continued erosion of Indigenous culture and language in the digital age.

32 Initially designed for English typewriters in the mid-to-late 19th century, the International Standards Organization officially adopted the QWERTY keyboard layout in 1971, becoming "the de facto Standard layout for Communications and computer interface keyboards" in 1972 [Noyes 1983]. Over the last 50 years, the ISO Standard has evolved as our communication technologies have, and ISO 9995-9:2016 now includes definitions for multilingual and multiscript keyboard layouts [International Standards Organization 2016]. However, these standards still assume that most languages can be represented with a Latin alphabet. The ISO Standard clearly states that it "is intended to address all characters needed to write all contemporary languages using the Latin script, together with standardized Latin transliterations of some major languages using other scripts" [International Standards Organization 2016]. This assumption that "all contemporary languages" can be written using a Latin character set is the type of colonial coercion that continues to institute western ideologies as dominant.

33 Regarding computing, our experiences are configured and guided by technology design philosophies that do not always include a combined understanding of "people, technology, society, and business" [Norman and Tognazzini 2015]. By working with members of my community, my Ancestral Code project has allowed me to explore keyboard designs that can better represent Nehiyawewin and Nehiyaw culture when entering syllabics into the computer. I aimed my design objectives at challenging assumed western standards by investigating the role of the keyboard as an input device, and

how such devices can support Nehiyaw cultural pedagogy and improve the relationships between the user, their language (i.e., *Nehiyawewin*), and the computer. Though Nehiyaw syllabics are taught in several ways, one popular method is the arrangement of the Nehiyawewin syllabary into a star design, often referred to as a *Cree syllabic star chart* (Figure 3). This design is used often in teaching syllabics and contains several culturally specific teachings. For example, as a student at University nuhelot'ine thaiyots'i nistameyimâkanak Blue Quills, my favourite syllabic "origin-stories" were about the "ᑌ" syllabic, described as being symbolic of the pipe used in Nehiyaw ceremony, "ᐱ" is a medicine lodge (i.e., tipi), and "ᑕ" can be interpreted as the toe of a moccasin. Whether or not these story sources are the true inspirations for the initial syllabic creations, I recognize the importance the visual imagery of syllabics holds and how they are intricately tied to cultural representations and understandings. Furthermore, due to the reflected and rotational arrangement and orientation of Nehiyaw syllabics, these teachings cannot be represented using the modern four-row keyboards used for typing in western computing.

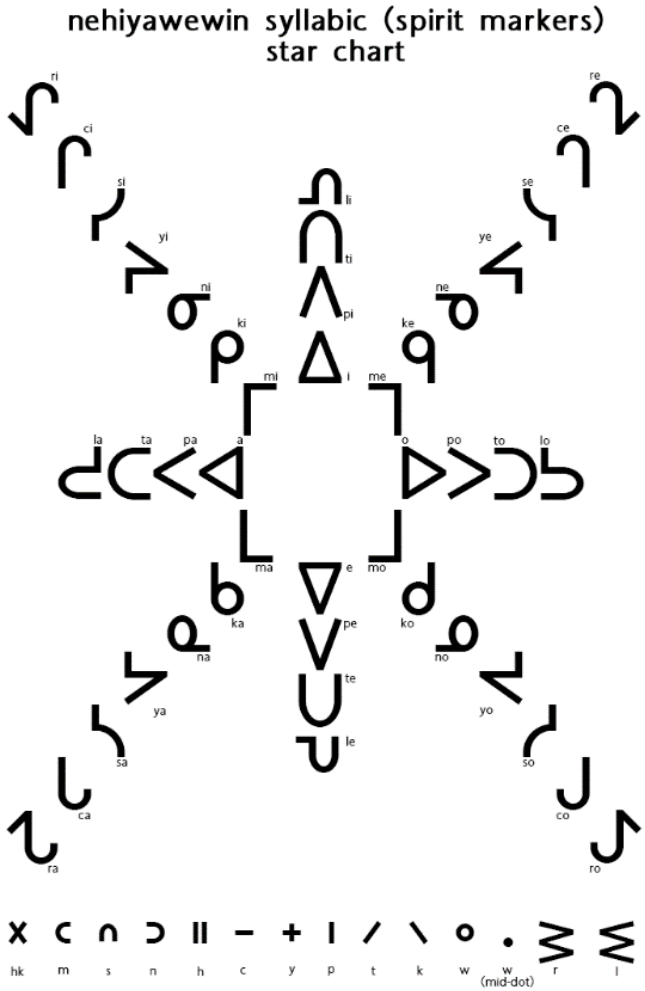

Figure 3. My own Nehiyawewin syllabic star chart.

I feel that a keyboard or input device that uses this Nehiyaw star layout is culturally significant, meaningful to a Nehiyaw user, and therefore appropriate for capturing Nehiyawewin. Through my iterative design process for the Nehiyawewin keyboard, I critically evaluated everything from the keycaps to the printed circuit boards ("PCB"). During this process, I created five distinct designs to retain cultural associations, allow efficient syllabic entry, and have practical usability for typing everyday documents. I found the star design keyboard was best suited as the primary interface for coding with Nehiyawewin in the Ancestral Code programming environment (Figure 4). Similarly, testing these designs with Nehiyawewin language-learners, users found typing syllabics with a "star" keyboard easier than QWERTY layouts because of the grouping of syllabic sounds in each of the four quadrants establishes a mental-map making locating syllabics by sound easier. For example, all the "i" syllabics are located in the keyboard's top left and top vertical. This

relationship between spoken language, computer language, and teachings of the syllabic orthography is meaningful and is one that I feel is better supported by a device derived from Nehiyawewin pedagogy.

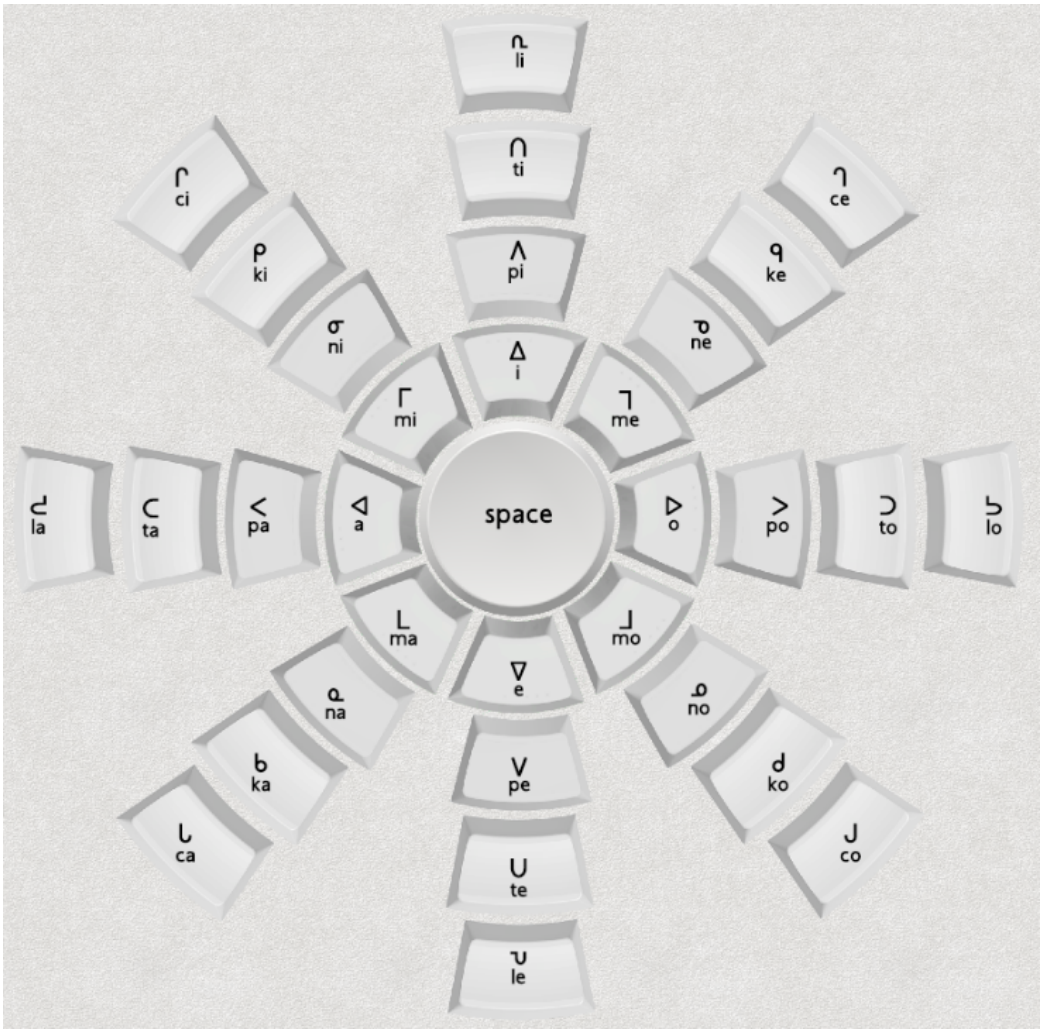

Figure 4. One of my proposed Nehiyawewin “Star Chart Keyboard” designs.

Bringing Nehiyawewin characteristics to a programming language

With the formalities of the orthography addressed, I progressed to investigating how to engage with Nehiyawewin programmatically. I initially approached this project with a very naive and western mindset. I considered common modern computing ideas and abstractions that included *variables*, *data types*, *loops*, *conditional branching*, and *linear/sequential instructions* (i.e., lines of code), and embarked on a journey to convert their English versions to Nehiyawewin. Thinking that some of these concepts would easily convert from English to Nehiyawewin, I quickly realized that this approach of language substitution in Nehiyawewin would not work. As I discovered on my first day as a Nehiyawewin student, Nehiyawewin technically does not have a word for “computer,” let alone any of the programming concepts I was hoping to capture. Technological words in English like *programming*, *network*, and *protocol* I found were, for the most part, “non-translatable” to Nehiyawewin, at least not a way I could use in developing a programming language. Today, finding appropriate Nehiyaw cultural meanings that can map to technological terminology remains the biggest challenge in developing the Ancestral Code project as a fully Nehiyawewin-privileged computing platform. Nevertheless, I took these challenges with language and formulated an approach that involved finding easily translatable concepts (if they existed), and borrowing from Nehiyaw language construction and word forms to create a unique coding paradigm.

Translating Nehiyawewin

What is a computer? I asked the most knowledgeable person in my Nehiyaw class, the Elder. That conversation went something like this:

Me	How do you say *computer*?

Elder	Well, that depends. There really is no word for computer, but some people use *masinatakan cikastepayicikanis*.

Me	What does it mean?

Elder	*Masinatakan* means to type or write, but actually comes from words that mean “using a tool to create or stamp a mark” and *cikastepayicikanis* is the word we use for TV, it comes from a word about *shadows* so something like “making a shadow, or full of shadows, or a box of shadows.” You can interpret these words together as “making marks in a box of shadows without a pen.”

Me	Uh-huh… *Ayhay.*

I note that this was not an isolated occurrence. “Well, that depends. There really is no word for [insert word],” was a very common response to almost anything “western” in my classes and applied to most modern technologies like radios, televisions, and cell phones. Depending on whom you asked, there are as many ways to describe these modern contraptions as there are dialects of Nehiyawewin. However, I accept and now use *mâmitoneyihcikanihkân* [6], suggested by Wayne Jackson, an esteemed Nehiyaw language expert and Nehiyawewin professor at University Blue Quills. As explained to me, this word is understood to mean “artificial thought/brain.” I choose this definition over *masinatakan cikastepayicikanis* because I feel it better conveys the essence of what a computer is. I find that *masinatakan cikastepayicikanis* [7] is more about describing the computer as a physical, non-animate object of utility than the more conceptual or abstract and humanized idea of what a computer is. After all, we as a species have a considerably long history of relating technologies to aspects of being human [Travers 1996, p. 57], and *mâmitoneyihcikanihkân* suits this tradition.

Finding suitable replacements for some of the more programming-specific lingo such as *integer*, *float*, *string*, *array*, *variable*, *subroutine/function*, *do/while*, *for/next*, and *if/then* proved to be highly problematic. These concepts either do not translate easily, can only be translated in a general sense, and/or require a broader context. And, in most cases, they cannot be translated without simplifying their meaning. Additionally, these concepts can only be translated by conversation and engagement with community members, fluent speakers, and Elders with the required experience and knowledge.

My solution to this particular challenge was one of metaphoric application rather than translation. I credit Hawai’ian game developer and computer programmer Kari Noe for introducing me to this philosophical change. Noe was a member of a Ōlelo Hawai’ian programming team engaged with translating C# into Hawai’ian [Muzyka 2018]. She described the “if/then/else” statement as an example where the “if/then/else” statement does not make sense when translated from English to Hawai’ian. The programming team consulted with native Hawai’ian speakers and community members to find meaningful terms that could be both culturally appropriate and programmatically practical. The result for “if/then/else” becomes *muliwai*, the Hawai’ian word for “river”. The idea of rivers being able to branch from the main waterway and eventually rejoin the main river later provided a better conceptualization of a Hawai’ian context than the English “if/then/else” statement. This culturally-aware solution in their language was not only inspirational but fundamentally altered how I was approaching the relationships between computing concepts and cultural relevance. Though I recognize that Hawai’ian peoples’ relationship with water is considerably different from those of the Indigenous peoples of the Americas, water is no less important. In Nehiyaw culture, water is a medicine, a provider of life, and is sacred. Therefore, I have come to use “river” as a conditional branch command. In Nehiyawewin river is *sîpiy*, and with this root I can then branch an “if” into *sîpîsis*, a “small river” or “creek” in English. Therefore, a code example might look

like the following examples:

```
ᐊᓯᐘᐦᐁᐤ ᐃᔨᓂᒥᐣ ᒦᓂᓯᐘᐟ
   ᓰᐱᕀ ᑭᑾᕀ ᐲᐦᒋᔨᕽ ᒦᓂᓯᐘᐟ
      ᓰᐲᓯᐢ ᐃᔨᓂᒥᐣ
         ᐊᓯᐘᐦᐁᐤ ᐊᔫᐢᑲᐣ ᒦᓂᓯᐘᐟx
      ᓰᐲᓯᐢ ᐊᔫᐢᑲᐣ
         ᐊᓯᐘᐦᐁᐤ ᐃᔨᓂᒥᐣ ᒦᓂᓯᐘᐟ
         ᐋᓂᐢᑰᓰᐱᕀ
   ᓰᐱᕀ ᑮᓯᐸᔨᐤ
```

Example 1. A series of conditional branch commands in ᐊᑎᒧ

```
asiwahew iyinimin mînisiwat
   sîpiy kikway pîhciyihk mînisiwat
      sîpîsis iyinimin
         asiwahew ayôskan mînisiwat.
      sîpîsis ayôskan
         asiwahew iyinimin mînisiwat
         âniskôsîpiy
   sîpiy kîsipayiw
```

Example 2. Example 1 in SRO (Cree#)

```
put the blueberries in the berry bag
   [start] a river, what is [in] this bag?
      [start] a creek [for] blueberries
         put a single raspberry in the berry bag.
      [start] a creek [for] a raspberry
         put the blueberries in the berry bag
         join the river
   the river ends
```

Example 3. The literal English translation of the code block from Example 1 or Example 2

In this example, the first "creek" ends with *mînisiwat*x[8] where "x" terminates and subsequently ends the "river-if" statement. The second "creek" ends with *âniskôsîpiy*[9] or "rejoin the river," and in this case, it would continue to the following statement in the "if" code block. In this way, the "if" statement is fluid. It can branch, terminate, or rejoin the original, reflecting a natural flow or progression. 45

Understanding computer functions in these cultural terms provides a more Indigenous method of relating to the machine. In addition, defining programmatic operations using culturally meaningful metaphoric terminology changes the process of keyword *translation* to a process of *knowledge adaptation*, ensuring that a Nehiyaw programmer does not need to language-switch between Nehiyawewin and English to have the computer perform its tasks. Looking at a computer's code from this metaphoric perspective can also be extended to the binary level of the machine, where a Nehiyaw worldview can reframe the binary understanding of 1 and 0 as animate and inanimate. 46

Nehiyaw language construction

I felt that morphemes and free word order were characteristics of Nehiyawewin that offered strong potential for programmatic versatility though they did offer some unique challenges. 47

Morphemes

Nehiyawewin is intensely metaphoric and descriptive and is a polysynthetic language, meaning the language combines many morphemes, often resulting in lengthy sentence-word constructions [Shirt and Wellman 2022]. A morpheme is a single linguistic unit of meaningful speech. For an English example, without focusing too much on etymology, the word "code" from the Latin *codex* can add different suffixes to alter its meaning. One such ending is "-ing" to change it to *coding*, representing an active verb. Another is "-er" to change it to *coder,* which is a noun meaning "a person who codes", and both could be structured in a sentence as "the coder is coding." These suffixes are morphemes. 48

A fun example that illustrates how Nehiyawewin morphemes are stringed together is the word for "hippopotamus." Nehiyawewin does not have an actual word for hippopotamus because this animal is not native to the Americas, and therefore it is described in language using its most prominent features. In Nehiyawewin, as described by Nehiyaw Elder Solomon Ratt, a hippopotamus is called *kihci-kispakasakewi-mistipwâmi-mahkitôni-nîswâpitewi-atâmipeko-pimâtakâwi-kohkôs* or in English as "great, thick-skinned, big-thighed, big-mouthed, two-toothed, underwater, swimming, pig " [Ogg 2019]. 49

Another interesting example is the Nehiyawewin translation for the English word "pony". In Nehiyawewin, the word *atim* [10] is the word for dog. When you prefix *atim* with *mist-* from the word *mistahi-*, which means "[something] is great/large/big," the resulting word *mistatim* is formed, which can mean "big dog," but is more commonly used to mean "horse." Adding the diminutive suffix *–osis,* meaning "[something] is small or little," to *atim*, the result is *acimosis* meaning "little dog" or "puppy." We can go one step further and combine all three of these ideas to get *mistacimosis*. The resulting understanding is a "small horse" or "pony." Though one could argue it also means a "very big puppy," the context would clarify what the speaker is referring to. 50

For Ancestral Code, I used these bound morphemes of *mistahi-* and *-osis* as ways to make variables increase or decrease in value (i.e., size), even though they may not be semantically or syntactically correct in Nehiyawewin speech. So, for example, using a numeric variable called *atim*, you can use the following statement: 51

```
atimosis
```

This is equivalent to the more traditional coding representation of:

```
atim = atim – 1;
```

This usage carries the meaning of "increase the size of 'the dog' and reduce the size of 'the dog'" or "make 'the dog' bigger then smaller." In this example, the variable "atim" has "some value" manipulated through a single morpheme-constructed token instead of a more *common calculation assignment* in other programming languages delimited by spaces to separate each programmatic token. Not to mention the problem with translating the syntax of `atim = atim + 1` as an assignment command and not a logical statement into Nehiyawewin.

Free Word Order

Free word order means words in a sentence do not necessarily need to be in a rigid sequence. This ordering is especially apparent in something like "the cat sees the dog"; whereas, in English, you cannot swap "the cat" and "the dog" without changing its meaning, as in "the dog sees the cat." However, this is not the case in Nehiyawewin, where *minôs wâpamew atimwa*[11] and *atimwa wâpamew minôs* mean the same thing. This language use is because there is a third-person obviative *-wa* attached to *atim*, indicating the dog *atim* is the object being acted on regardless of where it appears in the sentence in relation to the cat. To say this sentence in English requires modifying the verb to clarify who or what is seeing and identifying who or what is being seen. This sentence is a simplistic example, but as the coding environment for Ancestral Code was designed to be written as a narrative, having this flexibility is something I wanted to be able to use because it is non-linear and changes our perceptions of coding in line-by-line formats. A simple coding example would be the completion command for *sîpiy*, `sîpiy kîsipayiw`[12]. In this case, *sîpiy kîsipayiw* and *kîsipayiw sîpiy* mean the same thing. Although, as a coder, I can use "end river" or "river ends," it does not matter the order of the 52

tokens as they are not fixed. A more expanded example uses this same idea but applies to whole lines or activities, as in a looping construction where a character such as Wîsahkecâhk might walk, hop, and talk:

```
ᐱᐳᐣ
   ᐃᐧᓴᑫᒐᕽ ᐱᒧᐦᑌᐤ
   ᓇᐸᑌᑳᐧᐢᑯᐦᑎᐤ ᐃᐧᓴᑫᒐᕽ
   ᐃᐧᓴᑫᒐᕽ ᐄᑭᐢᑫᐧᐤ
```

Example 4. Free word order in in ᐊᒋᒧ

```
pipon
   Wisakecahk pimohtew
   napate-kwâshohtiw Wisakecahk
   Wisakecahk pîkiskwew
```

Example 5. Example 4 in SRO (Cree#)

```
for one winter
   Wisakecahk walks
   hop Wisakecahk
   Wisakecahk speaks
```

Example 6. The literal English translation of the code block from Example 4 or Example 5

53 When this block of code runs, the lines that follow *pipon* are "randomized" as it is not essential which order those actions execute, as long as the "Wîsahkecâhk" object variable performs all three actions in this block. By comparison, an equivalent representation of this code in C# might look like this:

```
int winter = 1;
do {
   Random r = new Random();
   string[] actions = new string[] { "walk", "hop", "speak" };
   actions = actions.OrderBy(x => r.Next()).ToArray();
   foreach (string doAction in actions) {
      makeWisakecahk(doAction);
   }
   winter--;
} while (winter == 1);
```

54 The Ancestral Code keyword for a "do loop" is `pipon`[13], which translates as "a winter." In the next section, I will elaborate on the meaning and cultural significance of the reserved word *pipon*. But, for this example, if I wanted to perform these actions in sequence and not a random order, I would not bother using the `pipon` loop. You may also notice that the lines that direct the "Wîsahkecâhk" variable to do an action (see Example 4, Example 5, or Example 6) are in free word order in the Ancestral Code example (as in `Wîsahkecâhk walks` and `hop Wîsahkecâhk`). Unlike many programming languages that must start with a command or keyword followed by variables or parameters, this is not necessary for Ancestral Code, as long as the variable and command are on the same executing line. The programming parser, the part of the programming environment responsible for separating lines of code into smaller

elements and individual instructions, will still be able to discern the token from the data, regardless of the position of each in the line. In the next section, I will detail the reasons behind this built-in randomization. Still, in terms of programming language construction, using features of morphemic commands and free word order, are a means to describe the relationships between the programmatic structures of Ancestral Code and Nehiyaw culture.

Culture and language == code && code == culture and language

Language plays a significant role in the enterprise of computer programming, and these activities are still heavily informed by western culture. Consequently, it is challenging to envision programming as anything but a socio-technical subculture populated with hordes of Benjamin Nugent's *Lewis Skolnick-esque* Type-1 "nerds" [Nugent 2008, p. 5].[14] This stereotyped and generalized view of programmers and programming was the last challenge I aimed to "reformulate," to introduce Indigenous cultural practices as digital metaphorical structures that view programming from Indigenous epistemological and ontological concerns that alter how programming is perceived in its current western context. For example, I admire how Ramsey Nasser describes the algorithms depicted in the Arabic calligraphic forms of his قلب programming language "as high poetry" [Nasser 2012]. I also found that describing western computing ideas in Nehiyawewin often resulted in algorithmically poetic word creations, as evidenced with the concept of "river" to represent "if/then" logic. I expand on this analogy-based token use for representing programmatic instructions by applying the same metaphoric treatment to culturally specific understandings as a way of genuinely viewing computer programming as a non-western endeavour. Through this reimagining of computing terms, I highlight how computing concepts already reflect Indigenous cultural teachings, practices, especially ceremony, as I demonstrate in the following specific examples of *miyâhkasike*[15], *pipona*[16], and *waniyaw* [17]. 55

Miyâhkasike and Tisamân

One of the most common and essential cultural practices in Nehiyaw culture is *the smudge*. A *smudge* is a small, personal ceremonial practice where the burning of an Indigenous medicinal herb such as sweetgrass or sage is used to "cleanse" and "purify" the individual. When *smudging*, people pass their hands or objects to be "blessed" through the rising smoke trails. In a normal smudge, you use your hands to draw the smoke towards you – blessing your head, ears, eyes, mouth, heart, and body with the smoke. And then, you "bless" anything else you wish to be cleared of negative energies, such as food, tobacco, eyeglasses, or even your laptop. Essentially, smudging is responsible for blessing anything that can affect any of your four spirits. I have even heard stories from several Nehiyaw and Métis Elders that have physically smudged their laptop to purify it before "Googling." In the context of ceremony, this idea of "cleansing" is something that computers do regularly. Whether it is emptying the trash bin, clearing memory, resetting the graphics display, or deleting a browser's cache, the intent of all these activities is to remove items that can negatively affect the system's operation. Therefore, the first command in an Ancestral Code program is `miyâhkasike` which is "to smudge with sage/sweetgrass," or `tisamân`[18], which is "to smudge (in general)," both serve the same purpose of preparing the system. The choice of which smudge command to use is up to the programmer. However, I personally feel that a program that relates a sacred story or contains culturally specific or significant knowledge in the code would start with `miyâhkasike`, being more purposeful than the more generic `tisamân`. These "smudging" operations include, but are not limited to, clearing any current output on the screen, clearing and readying the program's libraries and variables, and clearing any cache from a previous execution. 56

Miyâhkasike is an essential piece of code because not only does it have a very real programmatic purpose, but to have the system digitally mirror a user's physical ceremonial practice transcends the system. It also symbolically provides the computer a "spirit" that the user can relate to as more of a living being instead of seeing the computer as a subordinate spiritless instrument. From an Indigenous perspective, this kind of human-machine connection is one of collaboration and kinship, and has been explored by several Indigenous scholars and artists [Noori 2011] [L'Hirondelle 2014] [Lewis et al 2018]. 57

Pipon

Pipon literally means "winter." It, and its plural form *pipona,* along with the lexically related words *pipohki* (next winter), 58

awasipipon (last winter), and *mesakwanipipon* (every winter)[19] are used in Ancestral Code as "for loops." `Pipon` describes the single execution of a group of lines, and `pipohki`, `awasipipon`, and `mesakwanipipon` are sub-functions that can only occur inside a repeating `pipona` loop.

You may ask why use *pipon* as a metaphor for the programmatic "loop?" Could you not use *nîpin* (summer) since it also occurs annually? What makes *pipon* important is its significance to aging and identity in Nehiyaw culture. For example, in Nehiyawewin, I say, "I am currently 49 winters old." As heard from respected Nehiyaw culture and language instructor Reuben Quinn, we use "winter" and not another season because back in the days before "comfortable housing" in the northern climes of what is now Canada, surviving winter signified your resilience and survival of the most extreme elements of Earth Mother [Quinn 2021]. Surviving winter is a valued and personal accomplishment. The symbolism in "winter" as a programming keyword lies in its representation as a repeatable cycle and in its relationships to aging and resilience that naturally imply increased experience and knowledge. So, similar to when a western-formatted computer program loops through a series of instructions, it often builds upon previously executed statements — the loop "ages" as it progresses and continues until the loop conditions are met or terminate. 59

Waniyaw

Waniyaw is a word that can be used for meaning "at random." In Ancestral Code, it is a way to simulate the dynamic and unpredictable forces of the natural world. Randomization is fundamental to Ancestral Code because of the generative nature of the outputs it creates. From a cultural perspective, I want the visual outputs to have aspects that are arbitrary and not controllable by the programmer. This environment of chance reflects the aspects of nature we cannot control. Therefore, giving the system some autonomy and decision-making is one way to prevent the programmer from always having complete control. In Nehiyaw culture, it is necessary to allow nature to run its course or recognize that there are elements in our world beyond the individual's control. 60

Randomization

In addition to the reserved keyword `waniyaw`, the words `pipon`, `mihcecis`, `mihcet`, `mihcetinwa`[20], and the bound morpheme *misi-* incorporate randomization events in one form or another. 61

> `pipon` – is the instruction for "do once" or literally "for one winter," and all lines of code that proceed it are executed in random order until the "`x` " full stop is encountered.
>
> `mihcecis` – means "small many" and is used to produce a random number between 100 and 1,000.
>
> `mihcet` – means "many" and is used to produce a random number between 1,000 and 100,000.
>
> `mihcetinwa` – means "numerous" and is used to produce a random number between 100,000 and 1 million.
>
> `waniyaw` – is used in the context of the entire program or within a `pipona` loop block.

So, for example, if the programmer wants a statement or series of statements to execute randomly, they would write it like this: 62

```
waniyaw Wisakecahk pimohtew
```

Ancestral Code interprets this instruction as "have Wîsahkecâhk walk at a random interval." If this instruction is in the main body of the code, it will execute for random intervals from that point forward. If this instruction is inside a `pipona` block, it executes randomly only for as long as that loop is active and ends when the loop ends. In this use, the computer takes on the responsibility of "nature," and each time the program is executed, the randomness introduced with `waniyaw` in the code guarantees the output will always be unique. This uniqueness is similar to how a story is never quite the same when repeated, even by the same storyteller.

Conclusion

When it comes to coding in any of the thousands of computer programming languages available, a programmer is obliged to subscribe to and accept the social, technological, and cultural attitudes that created that language. Ancestral Code, is no exception, in that it is formulated to be more accessible to Nehiyaw users. However, in contrast to other (i.e., more common/traditional) computing languages, Ancestral Code is built on specific Nehiyaw cultural principles and not necessarily the lineal or logical requirements defined by the system. This difference means that Ancestral Code's model and programming paradigm can alter computing philosophies and create new opportunities and avenues for Indigenous computer programming pedagogy. 63

"Survivance," as defined by distinguished Indigenous cultural theorist Gerald Vizenor, is "an active sense of presence, the continuance of native stories, not a mere reaction, or a survivable name. Native survivance stories are renunciations of dominance, tragedy and victimry" [Vizenor 2008, p. 1]. As a project, my intention in creating Ancestral Code was to make a system capable of collaborating with Indigenous knowledges to create a uniquely Indigenous experience within a digital space born from western computational sciences. Through a wholistic and Indigenous approach to computer programming, I have revealed that there can be a deep connection in the human-computer relationship paradigm, one that can advance programming practices to be more culturally informed while remaining relevant and critical to the survivance of Indigenous language and culture. 64

Using theoretical wholistic Indigenous design frameworks and culturally-determined computer programming language like the ones I described in this article, I am encouraging deeper critical discussions on the socio-technical philosophies of computer programming. Ancestral Code can be used as a template that seeks to harmonize cultural epistemologies and ontologies with computing by redefining computing philosophies through a cultural lens. It is a project meant to take a user on a voyage through Nehiyaw knowledges that have developed over millennia and have those knowledges define the relationships and models of modern computing. This journey then changes the relationship from one of "human-and-computer" to one that is "culture-and-computer." 65

I feel this change in philosophy and approach to computer programming rewards both Indigenous and western computing cultures. From my perspective, a programmer's identity is heavily imbued with western computing practices and personally meaningful relationships with software and the interaction with computing devices. This broadening and augmenting of software and hardware architectures are worthy of further investigation, especially for the potential benefits they can provide as a template for other Indigenous communities who wish to advocate and explore their cultural languages and teachings as programmatic interfaces. 66

Glossary 67

Standard Roman Orthography	IPA (Pronunciation)	Meaning
âcimow	ʌtʃɪmow	[s/he] narrates [her/his] own story
acimosis	ʌtʃɪmʊsɪs	small dog; puppy
âniskôsîpiy	aːnɪskoːsiːpij	following the river; rejoin the river
atim	ʌtɪm	a dog
atimwa wâpamew minôs	ʌtɪmwʌ waːpʌmew mɪnoːs	the dog + is seen by + the cat
awasipipon	ʌwʌsɪpɪpʊn	last winter
kîsipayiw	kiːsɪpʌyiw	[something] ends or terminates
mâmitoneyihcikanihkân	maːmɪtonejɪhtʃikanɪhkaːn	computer; artificial brain
masinatakan cikastepayicikanis	mʌsɪnʌtʌgʌn tʃɪkʌsteːpajɪtʃikʌnɪs	computer; box of shadows
mesakwanipipon	meːsʌkwʌnɪpɪpʊn	every winter or every year
mihcecis	mɪhtʃeːtʃɪs	several
mihcet	mɪhtʃeːt	many
mihcetinwa	mɪhtʃeːtɪnwʌ	a lot; numerous
mînisiwat	miːnɪsɪwʌt	a bag used for berry picking
minôs	mɪnoːs	a cat
mistacimosis	mɪstʌtʃɪmʊsɪs	a pony
mistahi	mɪstʌhɪ	[something] is big or large
mistatim	mɪstʌtɪm	a horse; or a large dog
miyâhkasike	mijaːhkʌsɪgeː	[s/he] smudges with sweetgrass
nehiyaw	neːhɪyʌw	a Cree person; Cree culture
nehiyawewak	neːhɪyʌweːwʌk	Cree people (plural)
nehiyawewin	neːhɪyʌweːwɪn	Cree Language
nîpin	niːpɪn	summer time
nitâpân	nɪtaːbaːn	my great grandparent (grandmother)
nohkompân	nʊhkʊmbaːn	grandmother + passed on
pipohki	pɪpʊhkɪ	next winter
pipon	pɪpʊn	winter
pipona	pɪpʊnʌ	winters (plural)
sîpihkomipit	siːpɪhkʊmɪpɪt	bluetooth
sîpîsis	siːpiːsɪs	creek (small river)
sîpiy	siːpij	river
tisamân	tɪsʊmaːn	smudge
waniyaw	wʌnɪyʌw	at random; at a random time
wâpamew	waːpʊmew	[s/he] sees [her/him]
wîsahkecâhk	wiːsʌhkeːtʃaːhk	cultural teacher and legendary figure in Nehiyaw culture

Table 1. Standard Roman Orthography, IPA (Pronunciation), and Meaning of Nehiyawewin

Appendix

Hindu-Arabic Numbers	Nehiyawewin Syllabic-Numerals									
1 - 10	ᐧ	ᐧᐧ	ᐳ	ᐧᐳ	ᐧᐧᐳ	ᐳᐸ	ᐸ	ᐸᐧ	ᐸᐧᐧ	ᒥ
11 - 20	ᒥᐧ	ᒥᐧᐧ	ᒥᐳ	ᒥᐧᐳ	ᒥᐧᐧᐳ	ᒥᐳᐸ	ᒥᐸ	ᒥᐸᐧ	ᒥᐸᐧᐧ	ᓀ
21 - 30	ᓀᐧ	ᓀᐧᐧ	ᓀᐳ	ᓀᐧᐳ	ᓀᐧᐧᐳ	ᓀᐳᐸ	ᓀᐸ	ᓀᐸᐧ	ᓀᐸᐧᐧ	ᓂ
31 - 40	ᓂᐧ	ᓂᐧᐧ	ᓂᐳ	ᓂᐧᐳ	ᓂᐧᐧᐳ	ᓂᐳᐸ	ᓂᐸ	ᓂᐸᐧ	ᓂᐸᐧᐧ	ᓄ
41 - 50	ᓄᐧ	ᓄᐧᐧ	ᓄᐳ	ᓄᐧᐳ	ᓄᐧᐧᐳ	ᓄᐳᐸ	ᓄᐸ	ᓄᐸᐧ	ᓄᐸᐧᐧ	ᓇ
51 - 60	ᓇᐧ	ᓇᐧᐧ	ᓇᐳ	ᓇᐧᐳ	ᓇᐧᐧᐳ	ᓇᐳᐸ	ᓇᐸ	ᓇᐸᐧ	ᓇᐸᐧᐧ	ᐧ
61 - 70	ᐧᐧ	ᐧᐧᐧ	ᐧᐳ	ᐧᐧᐳ	ᐧᐧᐧᐳ	ᐧᐳᐸ	ᐧᐸ	ᐧᐸᐧ	ᐧᐸᐧᐧ	ᐳ
71 - 80	ᐳᐧ	ᐳᐧᐧ	ᐳᐳ	ᐳᐧᐳ	ᐳᐧᐧᐳ	ᐳᐳᐸ	ᐳᐸ	ᐳᐸᐧ	ᐳᐸᐧᐧ	ᐸ
81 - 90	ᐸᐧ	ᐸᐧᐧ	ᐸᐳ	ᐸᐧᐳ	ᐸᐧᐧᐳ	ᐸᐳᐸ	ᐸᐸ	ᐸᐸᐧ	ᐸᐸᐧᐧ	ᐱ
91 - 100	ᐱᐧ	ᐱᐧᐧ	ᐱᐳ	ᐱᐧᐳ	ᐱᐧᐧᐳ	ᐱᐳᐸ	ᐱᐸ	ᐱᐸᐧ	ᐱᐸᐧᐧ	ᐧᒥᑕ

Table 2. Syllabic Numeracy Appendix (Numbers 1 to 100)

Notes

[1] *Nohkompan* – grandmother who is passed on

[2] *Nitâpân* – great-grandmother

[3] ᐊᑎᒧ *âcimow* – story

[4] The font can be found at https://www.evertype.com/emono/

[5] See the Syllabic Numeracy Appendix for more

[6] *mâmitoneyihcikanihkân* – one Nehiyaw word for computer, meaning artificial brain.

[7] *masinatakan cikastepayicikanis* – another Nehiyaw word for computer. Roughly, a book or writing in a box of shadows.

[8] *mînisiwat* – a berry bag.

[9] *âniskôsîpiy* – where a river converges.

[10] *atim, mistatim, acimosis, mistacimosis* – dog, puppy, horse, pony; respectively

[11] *minôs wâpamew atimwa* – the cat sees the dog.

[12] *sîpiy, sîpiy kîsipayiw* – river, and the river ends

[13] *pipon* – winter

[14] Lewis Skolnick, portrayed by Robert Carradine, is one of the primary characters from the 1984 film *Revenge of the Nerds*.

[15] *Miyâhkasike* – to smudge with sweetgrass or sage.

[16] *pipona* – winters (plural of *pipon*).

[17] *waniyaw* – random, or randomly.

[18] *tisamân* – to smudge (in general).

[19] *pipohki, awasipipon, mesakwanipipon* – next winter, last winter, every winter; respectively.

[20] *mihcecis, mihcet, mihcetinwa* – a few, many, a lot; respectively.

Works Cited

Abd-El-Barr 2009 Abd-El-Barr, Mostafa. (2009) “Topological Network Design: A Survey.” *Journal of Network and Computer Applications* 32 (3): 501–9.

Abdelnour-Nocera, Clemmensen, and Masaaki 2013 Abdelnour-Nocera, José, Torkil Clemmensen, and Masaaki Kurosu. (2013) “Reframing HCI Through Local and Indigenous Perspectives.” *International Journal of Human–Computer Interaction* 29 (4): 201–4. https://doi.org/10.1080/10447318.2013.765759.

Absolon 2010 Absolon, Kathy. (2010) “Indigenous Wholistic Theory: A Knowledge Set for Practice.” *First Peoples Child & Family Review* 5 (2): 74–87. https://doi.org/10.7202/1068933ar.

Aikin 2009 Aikin, Jim. (2009) “The Inform 7 Handbook.” https://www.musicwords.net/if/I7Handbook8x11.pdf.

Ali 2014 Ali, Mustafa. (2014) “Towards a Decolonial Computing.” In *Ambiguous Technologies: Philosophical Issues, Practical Solutions, Human Nature*, 28–35. Lisbon, Portugal: International Society of Ethics and Information Technology.

Arawjo 2020 Arawjo, Ian. (2020) “To Write Code: The Cultural Fabrication of Programming Notation and Practice.” In *CHI ’20: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–15. Honolulu, HI. https://doi.org/10.1145/3313831.3376731.

Ardley 2014 Ardley, Sean. (2014) “Why Are Monospaced Typefaces Used for Programming? (Answer (1 of 2)).” *Quora*. https://www.quora.com/Why-are-monospaced-typefaces-used-for-programming/answer/Sean-Ardley.

Babaoglu et al 2006 Babaoglu, Ozalp, Geoffrey Canright, Andreas Deutsch, Gianni A Di Caro, Frederick Ducatelle, Luca M Gambardella, Niloy Ganguly, Márk Jelasity, Roberto Montemanni, and Alberto Montresor. (2006) “Design Patterns from Biology for Distributed Computing.” *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 1 (1): 26–66.

Benenson 2012 Benenson, Yaakov. (2012) “Biomolecular Computing Systems: Principles, Progress and Potential.” *Nature Reviews Genetics* 13 (7): 455–68.

Cormier and Ray 2018 Cormier, Paul, and Lana Ray. (2018) “A Tale of Two Drums: Kinoo’amaadawaad Megwaa Doodamawaad – ‘They Are Learning with Each Other While They Are Doing.’” In *Indigenous Research: Theories, Practices, and Relationships*, edited by Deborah McGregor, Jean-Paul Restoule, and Rochelle Johnston, 112–25. Toronto, Canada: Canadian Scholars’ Press.

Dourish and Mainwaring 2012 Dourish, Paul, and Scott D Mainwaring. (2012) “Ubicomp’s Colonial Impulse.” In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 133–42. Pittsburgh, PA: Association for Computing Machinery, New York, NY.

Garneau 2018 Garneau, David. (2018) “Electric Beads: On Indigenous Digital Formalism.” *Visual Anthropology Review* 34 (1): 77–86.

Gradual Civilization Act 1857 Gradual Civilization Act. (1857) *An Act to Encourage the Gradual Civilization of the Indian Tribes in This Province, and to Amend the Laws Respecting Indians*.

Hasselström and Åslund 2001 Hasselström, Karl, and Jon Åslund. (2001) “The Shakespeare Programming Language.” 6/6/2018. http://shakespearelang.sourceforge.net/.

Heaven 2013 Heaven, Douglas. (2013) “One Minute with...Ramsey Nasser.” *New Scientist* 217 (2909): 03–03.

International Standards Organization 2016 International Standards Organization. (2016) “ISO/IEC 9995-9:2016 - Information Technology — Keyboard Layouts for Text and Office Systems — Part 9: Multi-Lingual, Multiscript Keyboard Layouts.” ISO. 2016. https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/43/54374.html.

Irani and Dourish 2009 Irani, Lilly C, and Paul Dourish. (2009) “Postcolonial Interculturality.” In , 249–52.

Irani et al 2010 Irani, Lilly, Janet Vertesi, Paul Dourish, Kavita Philip, and Rebecca E Grinter. (2010) “Postcolonial Computing: A Lens on Design and Development.” In *CHI ’10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1311–20. Atlanta, GA, USA: Association for Computing Machinery.

King 2022 King, Kevin. (2022) “Typotheque: Syllabics Typographic Guidelines and Local Typographic Preferences by Kevin King.” *Typotheque* (blog). January 24, 2022. https://www.typotheque.com/articles/syllabics_typographic_guidelines.

Kleinrock and Kamoun 1980 Kleinrock, Leonard, and Farouk Kamoun. (1980) “Optimal Clustering Structures for

Hierarchical Topological Design of Large Computer Networks.” *Networks* 10 (3): 221–48.

Lewis et al 2018 Lewis, Jason Edward, Noelani Arista, Archer Pechawis, and Suzanne Kite. (2018) “Making Kin with the Machines.” *Journal of Design and Science*. https://doi.org/10.21428/bfafd97b.

L’Hirondelle 2014 L’Hirondelle, Cheryl. (2014) “Codetalkers Recounting Signals of Survival.” In *Coded Territories: Tracing Indigenous Pathways in New Media Art*, edited by Steven Loft and Kerry Swanson, 147–68. Calgary, AB: University of Calgary Press.

Madden, Higgins, and Korteweg 2013 Madden, Brooke, Marc Higgins, and Lisa Korteweg. (2013) “‘Role Models Can’t Just Be on Posters’: Re/Membering Barriers to Indigenous Community Engagement.” *Canadian Journal of Education* 36 (2): 212–47.

Merritt and Bardzell 2011 Merritt, Samantha, and Shaowen Bardzell. (2011) “Postcolonial Language and Culture Theory for HCI4D.” In *CHI’11 Extended Abstracts on Human Factors in Computing Systems*, 1675–80.

Milloy 2008 Milloy, John. (2008) “Indian Act Colonialism: A Century Of Dishonour, 1869-1969.” Research Paper. Canada: National Centre for First Nations Governance.

Milloy 2017 Milloy, John S. (2017) *A National Crime: The Canadian Government and the Residential School System*. Vol. 11. Univ. of Manitoba Press.

Muzyka 2018 Muzyka, Kyle. (2018) “A Hawaiian Team’s Mission to Translate Programming Language to Their Native Language | CBC Unreserved Radio.” CBC Unreserved Radio. November 30, 2018. https://www.cbc.ca/radio/unreserved/indigenous-language-finding-new-ways-to-connect-with-culture-1.4923962/a-hawaiian-team-s-mission-to-translate-programming-language-to-their-native-language-1.4926124.

Nasser 2012 Nasser, Ramsey. (2012) “قلب (‘Qalb’).” 2012. https://nas.sr/%D9%82%D9%84%D8%A8/.

Noori 2011 Noori, Margaret. (2011) “Waasechibiiwaabikoonsing Nd’anami’aami," Praying through a Wired Window": Using Technology to Teach Anishinaabemowin.” *Studies in American Indian Literatures* 23 (2): 1–23.

Norman and Tognazzini 2015 Norman, Don, and Bruce Tognazzini. (2015) “How Apple Is Giving Design A Bad Name.” *Fast Company* (blog). November 10, 2015. https://www.fastcompany.com/3053406/how-apple-is-giving-design-a-bad-name.

Noyes 1983 Noyes, Jan. (1983) “The QWERTY Keyboard: A Review.” *International Journal of Man-Machine Studies* 18 (3): 265–81.

Nugent 2008 Nugent, Benjamin. (2008) *American Nerd: The Story of My People*. Simon and Schuster.

Ogg 2019 Ogg, Arden. (2019) “Hippopotamus in Cree: Solomon Ratt (y-Dialect).” *Cree Literacy Network* (blog). December 16, 2019. https://creeliteracy.org/2019/12/16/hippopotamus-in-cree-solomon-ratt-y-dialect/.

Okimasis and Wolvengrey 2008 Okimasis, Jean, and Arok Wolvengrey. (2008) *How to Spell It in Cree: The Standard Roman Orthography*. misāskwatōminihk (Saskatoon): Houghton Boston, miywāsin ink.

Ormond-Parker et al 2013 Ormond-Parker, Lyndon, Aaron David Samuel Corn, Kazuko Obata, and Sandy O’Sullivan. (2013) *Information Technology and Indigenous Communities*. AIATSIS Research Publications Canberra.

Philip, Irani, and Dourish 2012 Philip, Kavita, Lilly Irani, and Paul Dourish. (2012) “Postcolonial Computing: A Tactical Survey.” *Science, Technology, & Human Values* 37 (1): 3–29.

Quinn 2021 Quinn, Ruben. (2021) “Intermediate ᓀᐦᐃᔭᐠ Language Lessons.” Zoom Course.

Risam 2018 Risam, Roopika. (2018) *New Digital Worlds: Postcolonial Digital Humanities in Theory, Praxis, and Pedagogy*. Evanston, Illinois, US: Northwestern University Press.

Shirt and Wellman 2022 Shirt, Marilyn, and Tina Wellman, eds. (2022) *Tânisîsi Kâ-Ôsîtahk Pîkiskwêwinisa : Morphology Dictionary*. St. Paul, AB: University nuhelot’ine thaiyots’i nistameyimâkanak Blue Quills.

Thomas 2005 Thomas, Robina Anne. (2005) “Honouring the Oral Traditions of My Ancestors through Storytelling.” In *Research as Resistance: Critical, Indigenous and Anti-Oppressive Approaches*, edited by Leslie Brown and Susan Strega, 237–54. Toronto, ON: Canadian Scholars’ Press/Women’s Press.

Travers 1996 Travers, Michael David. (1996) “Programming with Agents: New Metaphors for Thinking About Computation.” Doctor of Philosophy, Cambridge, MA: Massachusetts Institute of Technology.

Unicode Inc. 2021 Unicode, Inc. (2021) “The Unicode Standard, Version 14.0.” Unicode, Inc. https://unicode.org/charts/PDF/U1400.pdf.

Vee 2017 Vee, Annette. (2017) *Coding Literacy: How Computer Programming Is Changing Writing*. Mit Press.

Venne 1981 Venne, Sharon. (1981) “Indian Acts and Amendments, 1868-1975.” An indexed collection. University of Saskatchewan Native Law Centre, Saskatoon.

Vizenor 2008 Vizenor, Gerald Robert. (2008) *Survivance: Narratives of Native Presence*. U of Nebraska Press.

Warschauer 1998 Warschauer, Mark. (1998) “Technology and Indigenous Language Revitalization: Analyzing the Experience of Hawai’i.” *Canadian Modern Language Review* 55 (1): 139–59.

Wilson 2008 Wilson, Shawn. (2008) *Research Is Ceremony: Indigenous Research Methods*. Halifax, NS: Fernwood Publishing.

Wolfart 1973 Wolfart, H Christoph. (1973) “Plains Cree: A Grammatical Study.” *Transactions of the American Philosophical Society* 63 (5): 1–90.

Wolfart and Ahenakew 1993 Wolfart, H.C., and Freda Ahenakew, eds. (1993) *Kinêhiyâwiwininaw Nêhiyawêwin. The Cree Language Is Our Identity: The La Ronge Lectures of Sarah Whitecalf*. Winnipeg, MB, CA: University of Manitoba Press.

Worden, Staszewski, and Hensman 2011 Worden, Keith, Wieslaw J Staszewski, and James J Hensman. 2011. “Natural Computing for Mechanical Systems Research: A Tutorial Overview.” *Mechanical Systems and Signal Processing* 25 (1): 4–111.