Metaphors Be With You: A Tireless Work On Play On Words

Table of Contents

Copyright © 2015 Richard Madsen Neff
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being "(", with the Front-Cover Texts being "Metaphors Be With You: A Tireless Work On Play On Words", and no Back-Cover Texts.
A copy of the license is included in the section entitled ‘‘GNU Free Documentation License’’.

-~-~-~-~-~-

To my Dad
for inspiring me to get 'er done.

-~-~-~-~-~-

1 (

Groucho Marx once quipped, "Outside of a dog, a book is a man's best friend. Inside of a dog, it's too dark to read." This book is not about that kind of word play. (These are not the droids you're looking for. Move on.) But it *will* have something to say about *Outside* versus *Inside* in a logical context later on.
Kith and kin to word play is the idea of transformation, new from old. Old is known, new unknown. The unknown is daunting unless and until it becomes known. And the becoming process is called learning, a transformation from a old state of (less) knowledge to a new state of (more) knowledge.

1.1 Topics To Be Explored

The field of knowledge called discrete mathematics is **HUGE** — and no wonder; it is like a catch-all net. Anything not contemplated by mathematics of the continuous variety (calculus and company) is considered discrete (but not discreet). So narrowing the scope is

a challenge of near epic proportions — met by allowing the following topic pairs to serve as guideposts as we venture out into this vast field.

1 Sets and Logic
2 Functions and Relations
3 Combinatorics and Probability
4 Number Theory and Practice
5 Trees and Graphs
6 Languages and Grammars

Functions (which build on sets (which build on logic)) embody transformations. These are natural, foundational topics in a study of discrete mathematics. And to provide a transformative learning experience, functional programming (the oft-neglected (by most students) odd duck of the programming world) will be a particular focus of this work.

There are two main reasons to gain experience with the functional paradigm. One, functional programming rounds out the dyad of procedural and object-oriented programming, presumably already in the typical programmer's toolbox. Two, learning functional programming is a natural way to learn discrete mathematics, especially since functions are a key aspect, and illuminate many other aspects of discrete mathematics.

For reasons revealed à la Isaiah 28:10, the language of choice for learning functional programming will be the **lisp** programming language, specifically the Emacs lisp (elisp) dialect. Why lisp, certainly, but especially why elisp? To forestall unbearable suspense, let's dispense with the knee-jerk objection that elisp is not a **pure** functional programming language. With discipline it can be used as such, and the effort will be instructive.

A quick review and preview:

Procedural	**O**
Variable values vary.	
Collections are mutable.	

Code is stateful (function calls can have side effects).	Functions are partial (not defined on all inputs, so exceptions can be thrown).

The above table doesn't capture all the distinctions in this three-way comparison, but one thing is obvious. When looked at through the functional lens, procedural and object-oriented are the same. What then distinguishes them? Mainly what types can be the values of variables. Object-oriented adds user-defined types to the standard primitives and composites of procedural.

P-L and O-O also differ on how they treat nouns and verbs. It gets pretty topsy-turvy when normal everyday distinctions between nouns (things) and verbs (actions) are blurred. When not only do nouns get verbed, but verbs get nouned, we have a double play on words!

Speaking of double, combinatorics, the study of combining, or counting, and number theory, the study of integers (useful for counting — especially the positive ones) are an integral ☺ part of discrete math. How many ways are there of combining things in various combinations? Variations, arrangements, shufflings — these are things that must be accounted for in the kind of careful counting that is foundational for discrete probability theory. But they also come into play in modern practices of number theory like cryptology, the study of secret messages.

Double encore: two more discrete structures, with a cornucopia of applications, are trees and their generalization, graphs. Trees and graphs are two of the most versatile and useful data structures ever invented, and will be explored in that light.

Teaser begin, a **binary** tree — perhaps the type easiest to understand — can represent any conceptual (as opposed to botanical) tree whatsoever. How to do so takes us into deep representational waters, but we'll proceed carefully. And because graphs can represent relations, listed above in conjunction with functions, they truly are rock solid stars of math and computer

science. The relationship between functions and relations is like that of trees and graphs, and when understood leads to starry-eyed wonder, end teaser.

Last double: finishing off and rounding out this relatively small set of topics is the dymanic duo, languages and grammars. Language is being used, rather prosaically, as a vehicle for communication, even as I write these words, and you read them. There is fascinating scientific evidence for the facility of language being innate, inborn, a human instinct. Steven Pinker, who literally wrote the book on this, calls language a "discrete combinatorial system", which is why it makes total sense that it be in this book and beyond.

And what is grammar? Pinker in his book is quick to point out what it is not. Nags such as "Never end a sentence with a preposition," "Don't dangle participles," and (my personal favorite) "To ever split an infinitive — anathema!" Such as these are what you may have learned in "grammar school" as teachers tried their best to drill into you how to use good English. No doubt my English teachers would cringe at reading the last phrase of the last sentence of the last paragraph, where "it" is used in two different ways, and "this book" can ambiguously refer to more than one referent — think about it.

No, in the context of discrete mathematics, grammar has nothing to do with proper English usage. Instead, a grammar (note the indefinite article) is a **generator** of the discrete combinations of the language system. These generated combinations are called **phrase structures** (as opposed to word structures), hence, in this sense grammars are more precisely known as **phrase structure grammars**. But we're getting ahead of ourselves — PSG ISA TLA TBE l8r.

1.2 Autobiographical Aside

I am writing this book for many reasons, which mostly relate to the following language-ish math-ish relationships and transformations: Words have Letters, Numbers have Digits, Letters map to Digits,

hence Words map to Numbers.

I love words. I love numbers. Does that mean English was my first choice of discipline, Math second? No! English is my native tongue, but Math to me feels PRE native, or the language I came into this world furnished with. Not that I lay claim to being a mathematical prodigy like Fredrich Gauss, John Von Neumann, or any extraordinary human like that. Not even close!

A first recollection of my inchoate interest in math is from third grade. There I was, standing at the blackboard, trying to figure out multiplication. The process was mysterious to me, even with my teacher right there patiently explaining it. But her patience paid off, and when I "got it" the thrill was electrifying! Armed with a memorized times table, I could wield the power of this newly acquired understanding. Much later in life came the realization that I could immensely enjoy helping others acquire understanding and thus wield power!

Back to words. As I said, I love them. I love the way words work. I love the way they look. I love the way they sound. Part of this stems from my love of music, especially vocal music. Singers pay attention to nuances of pronunciation, dipthongs being but one example. Take another example. The rule for pronouncing the e in the. Is it ee or uh? It depends not on the look of the next word but its sound. The honorable wants ee not uh because the aitch is silent. The Eternal, but the One — ee, uh. Because one is actually won.

The attentive reader may have noticed that, among other infelicities, the preceding paragraph plays fast and loose with **use** versus **mention**. There is a distinction to be made between the use of a word, and the mention of a word.

For example, the word **Idaho**:

Idaho has mountains. (A use of the word.)

'Idaho' has five letters. (A mention of the word.)

This distinction will come to the fore when expressions in a programming language like lisp must be precisely formed and evaluated. The quote marks are all-important. Small marks, big

effect.
Analogously, what importance does a simple preposition have? How about the crucial difference between being saved **in** your sins versus being saved **from** your sins! Small words, enormous effect.
I have a well-nigh fanatical fondness for the English alphabet with its 26 letters. 26 is 2 times 13. 2 and 13 are both prime, which is cool enough, but they're also my birth month and day. And it's unambiguous which is which. 2/13 or 13/2 — take your pick. Because there's no thirteenth month, either way works. Not every birth date is so lucky! Not every language's alphabet is so easy, either.
Despite the coolness of twice a baker's dozen, it is deficient by one of being an even cooler multiple of three, and that's a state of affairs we'll not long tolerate. But which non-alphabetic (and non-numeric) symbol is best pressed into service to bring the number to three cubed?
Twenty-seven is three times nine. Nine give or take three is the number of main topics. Three chapters (not counting opening and closing bookends), with each chapter having at least three sections, each section having at least three subsections, each subsection having at least three subsubsections — perhaps bottoming out at this level — this seems to me to be a pretty good organization.
Slice it. Dice it. No matter how you whatever ice it — notice it's not ice (but is it water — whatever hve u dehydrated?!) — 'alphabet' is one letter shy of being a *concatenation* (smooshing together) of the first two letters of some other (which?) alphabet. So correct its shyness (and yours). Say alphabeta. Go ahead, say it! Now say ALP HAB ETA (each is pronounceable) — but don't expect anyone to understand what you just said!

1.3 TOP is the TLA

The Organizing Principle I have chosen for this book is the Three-Letter Acronym.
Why? Well, it fully fascinates me that "TLA ISA" is a perfectly grammatical sentence if interpreted just so. It expresses concisely

the truth: 'TLA' "self-describes" (which is another intriguing saying).
But TLA can also stand for Three-Letter Abbreviation, of which there are many, `(JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC)` being but one example.
Replacing the 'A' in TLA with a 'W' gives TLW, an acronym for Three-Letter Word — `(TOP TOE TON TOT TOY)` for a butcher's half-dozen of which.
Note that we could generate this list by simply prefixing `TO` to each letter in `PENTY`, a carefully chosen way to code-say Christmas.
Note too that neither this list of five TLWs nor the first list of twelve TLAs has commas, those typical list item separators. The comma-free rendering is deliberate and significant, as will be seen.
Back to playing with words. If you can eat your words, shouldn't you be able to smell and taste them too?! Or play with words like you play with your food? (These are rhetorical questions — don't try to answer them!)
Scrambled `EGGS` — `(GSGE GESG SGEG ...)`
Can all 12 (not 24! (which is NOT to say 24 factorial (which equals 620448401733239439360000, IYI))) arrangements be put into one tetragramamalgam? What's a tetragramamalgam? (These are NOT rhetorical questions — DO try to answer them!)
So there you have it. An assorted set of words and "words" to guide discovery. Many to unpack meaning from. If the metaphor is a good one, that meaning gets packed into words, (or encodings thereof (another layer of packing material)), then unpacking can result in rich interconnected structures that strengthen learning, that make it stick in the brain. And that that permanently (it is hoped) stuck learning will serve you well is my high hope.

2 ONE

He awoke with a start. What time was it? … Did his alarm not go off? … No, it was 3:45, he just woke up too early. He lay there, mind racing, trying to figure out what had woken him up. His wife was breathing deeply, she was sound asleep, so it wasn't her

stirring. No … it was that dream. That awesome, awefull dream where he was outside staring into the clear night sky, basking in "the icy air of night" when suddenly he felt a very real vertigo, as if he were falling *up* into that endless star-sprinkled void.
Falling up. A contradiction in terms. Yes, but one to be embraced, not shunned, because it gives one an EDGE. No, this is the title of a book. Yes, that treasured book he had acquired years ago, and still enjoyed. Clever poems and cute drawings. It was because the author was so playful with words and the thoughts they triggered that he liked it, and his other book, what was it? Where The Sidewalk Ends. Yes, well, enough of that. He would have to get up soon. Today he would meet his two new tutees. TWOtees for short. A male and a female had seen his ad and responded. Was he expecting more than two to? Yes … but two would be fine. Two would be enough to keep him busy.
-~-~-~-~-~-
"What I'd like to do today is just make introductions and see where we're at," he said. "My name is Tiberius Ishmael Luxor, but you can call me Til. T-I-L, my initials, my preference." While escorting his two visitors into his study, his mind wandered, as it always did when he introduced himself. Tiberius, why had his parents, fervent fans of Captain Kirk and all things Star Trek, named him that? And Ishmael? Was Moby Dick also a source of inspiration? Or the Bible? He had never asked them, to his regret. Luxor, of course, was not a name they had chosen. He rather liked his family name. The first three letters, anyway. Latin for light. He often played with Luxor by asking himself, Light or what? Darkness? Heavy? Going with the other meaning of light seemed frivolous.
"Uh, my name's Atticus. Atticus Bernhold Ushnol," his male visitor said, noticing that Til looked distracted. "What do you prefer we call you?" asked Til, jerking himself back to full attention. "Well, I like Abu, if that's okay with you!"
Til smiled. A man after his own heart. "How acronymenamored of you!" he said.
"Is that even a word?" asked his female guest, addressing Til.

Feeling just a little annoyed, she turned to Abu and said "You really want us to call you the name of the monkey in Aladdin?"
Abu smiled, a crooked grin. "Well, I wouldn't know about that — I never saw Aladdin."
"How could you not have seen Aladdin? What planet are you from?" she said, completely incredulous.
"Hold on a minute," said Til. "Let's discuss the finer points of names and origins after you introduce yourself."
"Ila. My middle name is my maiden name, Bowen. My last name is my married name, Toopik, which I suppose I'm stuck with." Ila always felt irritated that marriage traditionally involved changing the woman's name, not the man's. "And yes, it's Eee-la, not Eye-la. And I especially prefer you don't call me Ibt, if you get my drift!"
Til laughed. Abu was still grinning but said nothing, so Til went on. "We need variety, so Ila is fine. Not everyone needs to be acronymized!"
-~-~-~-~-~-
Later that day as Til reflected on their first meeting, he saw himself really enjoying the teacher-learner interactions to come. His ad had had the intended effect. Abu and Ila were motivated learners, albeit for different reasons. He was somewhat surprised that not more interest had been generated. But he knew it was unlikely up front. So two it would be. Two is a good number. Three is even better (odd better?) if he counted himself as a learner. Which he always did. Learning and teaching were intricately intertwined in his mind. Teaching was like breathing, and both teaching and learning were exhilarating to Til.
His wife interrupted his reverie. "Til, dear, what did you learn today from your first meeting with — what did you call them — Abu and Ila?"
"Interesting things," said Til. "Abu is the general manager at a landscaping company (I think he said nursery as well). Ila works for some web design company (I forget the name) doing programming. She wants to learn discrete math because she's been told it will make her a better programmer!"

"However, Abu's not a programmer, and not really all that techy, I gathered, but says he *loves* learning, practical or not. I predict this may put him at odds at times with Ila, who said she *only* wants practical learning. I told her Euclid would mock that attitude, and she was surprised. But she agreed to look into that. She didn't know who Euclid was, if you can imagine!" Til winked his eye, and his wife rolled hers.
"Anyway, Abu's pretty laid back, so her barbs won't bother him — much I hope," he said.
"Are they married, any kids?" she said.
"Abu is single, but would like to be married." Til rubbed his chin. "Ila is married, but she didn't mention her husband's name, or really anything about him, except that he's not a nerd like her. Before they even warmed up to my affinity for acronyms, Ila confessed to being a DINK, and Abu immediately responded, 'I guess that makes me a SINKNEM!' So he's pretty quick on the uptake."
"That's good!" she said. "So Ila's husband is not nerdy like her, but …?"
"But she also didn't say what he does," he said. "She taught herself programming, how cool is that?!"
"Very cool, Mr. Luxor," she said affectionately, gently rubbing his shoulders.
"Mrs. Luxor, I'll give you 20 minutes to stop that!" he said.
"What are their interests other than discrete math?" she said, ignoring his pathetic ploy.
"Well, let's see, Abu rattled off his other interests as flowers, stars and poetry!" he said. "And Ila likes sports and the outdoors. Hopefully that includes stars!"
"That would be great if you all have stars in common," she said.
Til nodded, but his eyes unfocused as his thoughts started to wander. His wife withdrew, wordlessly worrying that Til was thinking about sharing his dream about falling up into the stars with Abu and Ila. That never seemed to go well. But Til was remembering the exchange he had with Abu, who lingered a few

minutes after Ila left.
He had said, "Abu, since you like flowers, you must be okay with the STEMless reputation they have!" Without waiting for him to answer he went on, "Since I gave Ila an invitation to learn something about Euclid, I invite you to look up what Richard Feynman said about flowers and science, specifically whether knowing the science made him appreciate the flowers more or less?"
Well, enough musing for now. His two tutees were eager and grateful, and not just because they thought the fee he had set was reasonable. Little did they know the price of knowledge he would exact of them.
-~-~-~-~-~-

2.1 ABC

When you read you begin with A, B, C; when you sing you begin with Do, Re, Mi; when you count you begin with 1, 2, 3 — or is it 0, 1, 2?! Mathematicians do both, and have logical reasons for each. Computer scientists mainly do zero-based counting, as coaxed to do by both hardware and software. 0-1-2; ready-set-goo! So let's begin with the set of positive integers, also known as the counting numbers — except, wait. How do you count the number of elements in the set with no elements, AKA the empty set? You need zero, which for centuries was not even considered a number. Lucky for us, these days zero is a full-fledged member of the set of **natural** numbers, another name for the counting numbers. But some still see zero as UNnatural and thus exclude it from the set. Right away, ambiguity rears its hoary head, and definitions must needs be clarified before crafting arguments that depend on whether zero is in or out.
VTO: a **set** is just a **collection** of **objects**, where objects are individual, discrete, separate things that can be named or identified

somehow. A collection that serves the mathematical idea of a set is an **unordered** one — the elements can be listed in any order and it doesn't change the set.
Speaking of lists, LISP originally stood for LISt Processor (or Processing), so let's list some simple sets in lisp style, and show a side-by-side comparison with math style:

Lisp Style	Math Style
() or nil	{} or ∅
(A B C)	{A, B, C}
(Do Re Mi)	{Do, Re, Mi}
(1 2 3)	{1, 2, 3}
(0 1 2)	{0, 1, 2, …}

Other than dispensing with commas and replacing curly braces with parentheses, lisp is just like math — except, wait. What are those three dots saying? In math, the ellipsis (…) is short for "and so on" which means "continue this way forever". Since lisp is a computer programming language, where doing things forever is a bad idea, we truncated the set **N** after listing its first three members.
Also, no duplicates are allowed in sets. This requirement guarantees element uniqueness, a desirable attribute strangely yoked with element unorderliness, the other main (anti-)constraint.
Here are three lispy sets (really lists):
`(A B C) (A A B C) (A A A B C C)`
Here are three more:
`(B A C) (C A B) (B C A)`
But by definition these are all the same set — `(A B C)` — so let's just make it a rule to quash duplicates when listing sets. And since order is implied by the normal left-to-right listing order, we have a contradictory situation where order is implied yet irrelevant. That really implies (linked) lists are not the ideal data structure for

representing sets, and (linkless) vectors in elisp are no better — they are simply indexable arrays of (heterogenous or homogeneous) elements — but let's say our rule also prefers vectors to lists. So the rule transforms a mathy set like {A, C, C, C, B} into a lisp list `(A C B)` and thence to the vector `[A C B]`, which is now in a form that can be evaluated by the lisp interpreter built into emacs (a later lesson shows how lists can be evaluated too):
VTO: an **expression** expresses some value in some syntactic form, and to **evaluate** means to reduce an expression to its value (for the purpose of handing it off to some further processing step).

2.1.1 ACE

So far so good. We have learned a lot (in theory). Now it's time to exercise (put in practice) that learning to help it grow.

2.1.1.1 ADG

What is the result of evaluating the following expression in elisp?

```
[A C B]
```

2.1.1.2 BEH

Predict/verify the first element of

```
[A C B]
```

2.1.1.3 CFI

Predict/verify the second element of

```
[A C B]
```

2.1.1.4 JMP

Predict/verify the third element of

```
[A C B]
```

2.1.1.5 KNQ

Predict/verify the zeroeth element of

```
[A C B]
```

2.1.1.6 LOR

Predict/verify the fourth element of

```
[A C B]
```

2.1.2 GIK

We next turn our attention to a way to view statements about sets as **propositions** with Boolean values (true or false). In lisp, true is abbreviated **t**, and false **nil**. With some help from a pair of friendly global variables, we can be traditional and just use true and false.
VTO: a **proposition** is a sentence (or statement) that is either true

or false. The sentence can be about anything at all (not just sets), as long as ascertaining its truth or falsity is feasible. The study called **propositional logic** deals with standalone and compound propositions, which are propositions composed of other propositions, standalone or themselves composite. How to compose or combine propositions to create these compounds is coming after a few exercises in model building. Do start the warmup by thinking up three examples and three non-examples of propositions. Make them pithy.

True propositions are AKA **facts**. A set of true propositions is called a knowledge base, which is just a database of facts. This set represents what we know about the world (or a subset of the world) or, in other words, the meaning or interpretation we attach to symbols (such as sentences — strings of words). Take a simple two-proposition world:

- It is true that snow is white.
- It is false that grass is white.

Given our propensity for abbreviation, let's assign propositional variables `p` and `q` to make it easier to manipulate this world:

- `p` = "snow is white"
- `q` = "grass is white"

`p and not q` represents our interpretation of this world. We could imagine a stranger world — stranger than the **real** one where we live — where `p and q` (i.e., where snow is white and so is grass) or even `not p and q` (where snow is green, say, and grass is white) is the real model.

VTO: a **model** is (roughly speaking) the meanings (interpretations) of symbols. In general, a model is all over the map, but in propositional logic, models are assignments of truth values to propositions (conveniently identified as variables). As aforementioned, truth values are Boolean values, true or false. Period. No partly true or somewhat false, nor any other kind of wishy-washiness.

Decision-making ability is an important component of any programming language. Lisp is no exception. The conditional logic

decidering of one of the basic logical connectives will be revealed in the following exercise that invites you to explore the logical connectives `and` (∘), `or` (∨), `xor` (⊕), and `not` (¬). Naturally, `not` is not a connective in the sense of connecting two things together, but it's a logical operator nonetheless. For `xor`, the 'exclusive' or, use the `if` form to do a conditional expression.

2.1.2.1 SVY

In the form `(if p (not q) q)` `p` is `true` or `q` is `true`, but not both — check it out:

```
; let p be false and q be false
(if false (not false) false)
; now let p be false and q be true
(if false (not true) true)
; now let p be true and q be false
(if true (not false) false)
; finally let both be true
(if true (not true) true)
```

2.1.2.2 TWZ

What other special forms for doing **cond**-itionals does lisp have?

2.1.3 MOQ

We can compose or combine propositions to create compound propositions by applying the following rules again and again (consider this a 'grammar' for generating these propositional 'molecules' from propositional 'atoms'):

1 A `proposition` is a variable, that is, a single element from the set `[p q r s t ...]`;

2 alternatively, a `proposition` is a `proposition` preceded by a `not`;

3 alternatively, a `proposition` is a `proposition` followed by a `connective` followed by a `proposition`.

4 A `connective` is a single element from the set `[and or xor]`.

Rules 2 and 3 have a *recursive* flavor, so said because they refer back to themselves; that is, they define a proposition in terms of itself, and simpler versions of itself.

We actually need more syntactic structure to make sense of certain compounds. For example, building a proposition from scratch by

starting with
`proposition;`
from thence applying rule 1 (choosing `p`) then rule 3 yields
`p connective proposition;`
from thence applying rule 4 (choosing `and`) yields;
`p and proposition;`
from thence applying rule 3 again yields
`p and proposition connective proposition;`
from thence applying 1 twice and 2 once yields
`p and q connective not r;`
from thence one more application of 4 (choosing `or`) yields the not-further-expandable:
`p and q or not r`
Which is ambiguous. Does it mean `p` and `s` where `s` is `q` or `not r`? Or does it mean `s or not r` where `s` is `p and q`? In other words, which is the right interpretation, 1 or 2?
1 `p and (q or not r)`; associating from the right.
2 `(p and q) or not r`; associating from the left.
The mathematically astute reader will recognize that the parentheses are the extra syntactic sugar we need to sprinkle on to disambiguate (choose 1 or 2 (but not both)).
The next question is, does it matter? Try it both ways with the truth-value assignment of `p` to `true`, `q` to `false`, and `r` to `true`:
Replacing `p`, `q`, and `r` with `true`, `false`, and `true`, respectively, in
`p and (q or not r)`
yields
`true and (false or not true)`
Before simplying this expression, a quick review of how these connectives work:
The simplest one first: `not` flips truth values — when notted (or negated — `not` is AKA negation) true becomes false, false becomes true. To be true, `or` requires at least one of the two propositions appearing on its left-hand-side (LHS) and its right-hand-side (RHS) to be true. If both of them are false, their disjunction (a fancier name for `or`) is false. To be true, `and` requires

both propositions (LHS and RHS) to be true. If even one of them is false, their conjunction (a fancier name for `and`) is false.
Back to the task: the above expression simplifies to
```
true and (false or false)
```
from thence to
```
true and false
```
from thence to
`false`.
Replacing propositional variables with their values in
```
(p and q) or not r
```
yields
```
(true and false) or not true
```
which simplifies to
```
(true and false) or false
```
from thence to
```
false or false
```
and finally
`false`.
So they're the same. But wait! What if we try a different assignment of truth values? With all of `p`, `q` and `r false`, here is the evolving evaluation for 1 `(p and (q or not r))`:
1 `false and (false or not false)`
2 `false and (false or true)`
3 `false and true`
4 `false`
And here it is for 2 `((p and q) or not r)`:
1 `(false and false) or not false`
2 `false or not false`
3 `false or true`
4 `true`
So it's different — it DOES matter which form we choose. Let's tabulate all possible truth-value assignments (AKA models) for each, and see how often they're the same or not:
First, how do we generate all possible assignments? Let's use a decision tree:
```
       p     q     r
   / false false false
```

```
  /\ false false true
 /\/ false true  false
/  \ false true  true
\  / true  false false
 \/\ true  false true
  \/ true  true  false
   \ true  true  true
```

In this binary tree, each **node** (wherever it branches up or down) represents a decision to be made. Branching up represents a choice of false, branching down true. Going three levels (branch, twig, leaf) gives us the eight possibilities.

VTO: a tabulation of possible models (essentially picking off the leaves of the tree from top to bottom (making each leaf a row)) is called a **truth table**. Here's an example:

p	**q**	**r**	**not r**	**(q or not r)**	**(p and (q or not r))**
false	false	false	true	true	false
false	false	true	false	false	false
false	true	false	true	true	false
false	true	true	false	true	false
true	false	false	true	true	true
true	false	true	false	false	false
true	true	false	true	true	true
true	true	true	false	true	true

Numerically it makes sense to have, and indeed most other languages do have, 0 represent `false`. Lisp is an exception — 0 evaluates to non-nil (which is not nil — not `false`). Having 1 represent `true` has advantages, too, although MOL treat anything

non-zero as `true` as well. Still, it's less typing, so we'll use it to save space — but keep in mind, the *arithmetization* of logic is also instructive because of how it enables certain useful encodings. Here's the same truth table in a more compact form:

(	**p**	**∘**	**(**	**q**	**∨**	**¬**	**r**	**)**	**)**
0	0	0	0	1	1	1	0		
0	0	0	0	0	0	1	1		
0	0	1	0	1	1	1	0		
0	0	1	0	1	0	1	1		
1	1	0	0	1	1	1	0		
1	0	0	0	0	0	1	1		
1	1	1	0	1	1	1	0		
1	1	1	0	1	0	1	1		

2.1.3.1 UXA

Build a truth table of expression 2 and compare it with the truth table of expression 1.

2.1.3.2 BEH

Experiment with the Truth Table Generator in the course code repository. What did you learn?

2.1.4 SUW

Let it be. Let it go. Let it go so. Let it be so. Make it so.

Let `p` be prime. In lisp, something like `p` is called a symbol, and you can treat a symbol essentially like a variable, although it is much more than that.

Let something be something is a very mathy way of speaking. So lisp allows somethings to be other somethings by way of the keyword `let`.

Like God, who said let there be light, and there was light, the programmer commands and is obeyed. The keyword `let` binds symbols to values, fairly commanding them to stand in for values. Values evaluate to themselves. Symbols evaluate to the values they are bound to. An unbound symbol cannot be evaluated. If you try it, an error results.

Before doing some exercises, let's look at `let` syntactically and abstractly:

```
(let <frame> <body>)
```

The `<frame>` piece is a model in the form of a list of bindings. Just as we modeled truth-value assignments as, e.g., `p` is `true` and `q` is `false`, in list-of-bindings form this would be `((p true) (q false))`. A binding is a two-element list `(<var> <val>)`.

Variables can be left unbound, so for example, `(let ((a 1) (b 2) c) ...)` is perfectly acceptable. It assumes **c** will get a value later, in the `<body>`.

The `<body>` is a list of forms to be evaluated — forms that use the bindings established in the frame model.

The frame list can be empty, as in:

```
(let () 1 2 3)
; alternatively
(let nil 1 2 3)
```

The value of this **symbolic expression** (lispers shorten this to an **s-expression**, or **sexp** for shorter) is 3, the value of the last form before the closing parenthesis.

2.1.4.1 CFI

Predict/verify the result of evaluating the following sexp:

```
(let ((a 1) (b 2) (c 3)) (+ a b c))
```

2.1.4.2 DGJ

Predict/verify the result of evaluating the following sexp:

```
(let ((a 1) (b 2) (c 3)) (+ a b c d))
```

2.1.4.3 KNQ

Predict/verify the result of evaluating the following sexp:

```
(let ((a 1) (b 2) (c 3)) [+ a b c d])
```

2.1.4.4 LOR

Predict/verify the result of evaluating the following sexp:

```
(let ((a 1) (b 2) (c 3) (+ 4)) (+ a b c +))
```

2.1.4.5 MPS

Describe the more-than-syntactic difference between the starless form of let and the star-suffixed form.

```
(let ((a 1) (b 2) (c 3) (d (+ a 4))) (+ a b c d))
; versus
(let* ((a 1) (b 2) (c 3) (d (+ a 4))) (+ a b c d))
```

Could you get by without this star-suffixed form? How?

2.1.5 YAC

The rule for dealing with lisp expressions is to look for lists, and when found, evaluate them. Casual observation reveals the bulk of lisp code is in fact lists, and lists of lists nested to many levels. Evaluating a list entails evaluating each of its elements, except the first, which is treated differently. For now, think of the first element as an operator that operates on the values of the rest of the elements. So `(+ 1 2 3 4)` adds together 1 (the value of 1 — numbers self-evaluate), 2, 3 and 4 to produce 10. Note the economy of this so-called *prefix* notation: only one mention of the '+' sign is needed. The more familiar *infix* notation requires redundant '+' signs, as in `1+2+3+4`.

2.1.5.1 TWZ

Do the other three basic math operators work as expected?
How about when the number of operands (what operators operate on) is:

- Greater than two?
- Less than two?

Predict/verify the result of evaluating the following sexp, and in the process, think about 'list' as an 'operator' (on what?).

```
(let ((a 1) (b 2) (c 3) (d 4)) (list (+ a b) (/ d b) (-
d a) (* c d)))
```

2.1.6 EGI

CSP a description of set operations, union, intersection, difference, symmetric difference, subset-of, superset-of, disjoint, mutually or not; De Morgan's laws for logic and sets.

p	q	p ∘ q	p ∨ q	p ⊕ q
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

2.1.6.1 UXA

Investigate Venn Diagrams.

2.1.6.2 VYB

Venture into the 'quote' "macro" in lisp, used to "suppress" evaluation of symbols, lists and other things. Explore the difference between *set*, *setq* and *setf* — *setf* being a Common Lisp macro (hence the need to `(require 'cl)`) — and note which feature of functional programming is being flouted in this code:

```
(require 'cl)
(set 'a 'one)
(setq a (quote one))
(setf b '(one two))
(setf a (first b))
(setf (first b) a)
```

2.2 DEF

Til has assigned a LOT of homework. Ila complains to Abu about it. Abu is sympathetic.

"I mostly get the logical connectives, which why we don't just call them operators is what I don't get. They're the same as the logical operators in JavaScript, and …" Abu interrupted Ila's rant. "What's JavaScript?", he asked. "It's a programming language — the one I mostly program in. Anyway, what I also don't get is the conditional operator. It just seems illogical that it's defined the way it is. How did Til put it?"

"I believe he said 'p only if q is true, except when p is more true than q'," said Abu.

"That's what I'm talking about, what does 'more true than' mean? Something is either true or it's not. How can it be 'more true'? Truer than true??"

"Well, here's how I see it," said Abu. "When they have the same truth value, both true or both false, neither is more or less than the other, they're equal."

Ila felt her face go red. "Oh, now I get it — a true p is greater than a false q. And the last case, when p is false and q is true, p is less true than q. Way less, because it has zero truth."

"Irrelevant to the truth of q," said Abu. "I drew its truth table, like Til asked, and only one row is false, the rest true. Perhaps you drew it this way too?"

---

p q p →

	q
0 0	1
0 1	1
1 0	0
1 1	1

Ila frowned. "Mine has the last column centered under the arrow, but otherwise it's the same as yours."
"I like centered better," said Abu. "Til will be glad we get it, I think, but do you think he was going to explain it some more?"
"He will if we ask, but I don't want to ask him — will you?" said Ila. "No problem," said Abu. "I don't mind appearing ignorant, because appearances aren't deceiving in my case — I really AM ignorant!" "Well, I hate asking dumb questions," said Ila. "Never been a problem for me," said Abu with a slow exhalation.
Ila voiced her other main complaint. "And I'm still floored that he said, go learn lisp, it will do you good." Abu nodded. "I agree that seems like a lot to ask, but I'm enjoying the challenge. Lisp is cool!" Ila sighed. "It certainly lives up to its name: Lots of Irritatingly Silly Parentheses, and if you ask me, it's just confusing." Abu smiled. "I found this xkcd comic that compares lisp to a Jedi's lightsaber. See, I *got* the Star Wars reference. And by the way, just to show you what planet I'm from, I went and watched Aladdin. So let's see, Chewbacca is Abu to Han Solo's Aladdin, right!" "Hardly," said Ila. "So you saw the movie and you still want us to call you Abu?!"
-~-~-~-~-~-
"Think of it this way," said Til. "Inside versus Outside. With your pads draw a couple of circles, two different sizes, the smaller inside the larger. Make them concentric. Okay? Now consider the possibilities. Can something be inside both circles simultaneously?"
Abu and Ila both nodded simultaneously.
"That's your both p and q true case. Now, can something be outside both circles? Yes, obviously. That's your both p and q false case.

And can something be inside the outer circle without being inside the inner one? Yes, and that's when p is false and q is true."
Ila was feeling a tingle go up her spine. She blurted, "I see it! The only impossibility is having something inside the inner circle but not inside the outer — p true and q false. Makes perfect sense!"
Til beamed. "You got it, Ila! How about you, Abu?"
Abu just smiled and nodded his head.
"Would you please explain 'let' more?" said Ila. She was getting past her reluctance to ask questions, and it felt great!
"Think cars," said Til. "A car has a body built on a frame, just like a let has a body built on a frame. But without an engine and a driver, a car just sits there taking up space. The lisp interpreter is like the engine, and you, the programmer, are the driver."
Abu had a thought. "A computer also has an engine and a driver, like processor and programmer, again. But cars *contain* computers, so I doubt *equating* a car with a computer makes sense."
Til agreed. "You see a similarity but also a big difference in the two. A car is driven, but the only thing that results is moving physical objects (including people) from here to there. That's all it does, that's what it's designed to do. But a computer is designed to do whatever the programmer can tell it to do, subject to the constraints of its design, which still — the space of possibilities is huge."
Ila was puzzled. "Excuse me, but what does this have to do with 'let'?"
Til coughed. "Well, you're right, we seem to be letting the analogy run wild. Running around inside a computer are electrons, doing the "work" of computation, electromagnetic forces controlling things just so. It's still electromagnetics with a car, for example, combustion of fuel. But these electromagnetic forces are combined with mechanical forces to make the car go."
"Just think of 'let' as initiating a computational process, with forces defined by the lisp language and interacting via the lisp REPL with the outside world. Then *letting* computers be like cars makes more sense. Do you recall that a lisp program is just a list?"

Abu nodded, but interjected. "It's really a list missing its outermost parentheses, right?"
Til's glint was piercing. "Ah, but there's an *implied* pair of outermost parentheses!"
Ila fairly stumbled over her tongue trying to blurt out. "And a 'let nil' form!"
Now it was Abu's turn to be puzzled. He said, "Wait, why nil?" Ila rushed on, "It just means that your program can be self-contained, with **nil** extra environment necessary to run. Just let it go, let it run!"

2.2.1 KMO

Let's leave 'let' for now, and look at lisp functions via the *defun* special form.

```
(defun <parameter-list> [optional documentation string]
<body>)
```

Examples of /def/ining /fun/ctions via `defun` are forthcoming. First note that this language feature is convenient, but not strictly necessary, because any function definition and call can be replaced with a `let` form, where the frame bindings are from the *caller's* environment.

For example, the function `list-some-computations-on`, defined and invoked (another way to say called) like this:

```
(defun list-some-computations-on (a b c d)
  "We've seen this one before."
  (list (+ a b) (/ d b) (- d a) (* c d)))

(list-some-computations-on 1 2 3 4)
```

This definition/call has exactly the same effect as:

```
(let ((a 1) (b 2) (c 3) (d 4))
  (list (+ a b) (/ d b) (- d a) (* c d)))
```

Of course, any procedural (or object-oriented) programmer knows why the first way is preferable. Imagine the redundant code that would result from this sequence of calls if expanded as above:

```
(list-some-computations-on 1 2 3 4)
(list-some-computations-on 5 6 7 8)
; ...
(list-some-computations-on 97 98 99 100)
```

Functions encapsulate code modules (pieces). Encapsulation enables modularization. The modules we define as functions become part of our vocabulary, and a new module is like a new word we can use instead of saying what is meant by that word (i.e., its definition) all the time.

But stripped down to its core, a **function** is just an object that takes objects and gives other objects. Speaking completely generally, these are objects in the 'thing' sense of the word, not instances of classes as might be assumed if object-oriented programming were the context. Are there any restrictions on what objects can be the inputs to a function (what it takes) and what objects can be the outputs of a function (what it gives)? Think about and answer this question before looking at the endnote.

VTO: again, a **function** is simply an object, a machine if you will, that produces outputs for inputs. But simply amazingly, there is a plethora of terms describing various kinds and attributes of functions that have been designed and pressed into mathematical and computer scientific service over the years:

- **domain** is the term for the **set** of all *possible* inputs for a **function**. (So functions depend on sets for their complete and rigorous definitions.)
- **codomain** is the set of all *possible* outputs for a function.
- **range** is the set of all *actual* outputs for a function (may be different — smaller — than the codomain).
- **onto** describes a function whose range is the same as its codomain (that is, an **onto** function produces (for the right inputs) all possible outputs).
- **surjective** is a synonym for onto.
- **surjection** is a noun naming a surjective function.
- **one-to-one** describes a function each of whose outputs is generated by only one input.
- **injective** is a synonym for one-to-one.
- **injection** is a noun naming an injective function.
- **bijective** describes a function that is both one-to-one and onto.
- **bijection** is a noun naming a bijective function.

- **recursive** describes a function that calls itself. Yes, you read that right.
- **arguments** are the inputs to a function (argument "vector" (argv)).
- **arity** is short for the number of input arguments a function has (argument "count" (argc)).
- **k-ary function** — a function with **k** inputs.
- **unary – binary – ternary** — functions with 1, 2 or 3 arguments, respectively.
- **infix notation** describes a *binary* function that is marked as a symbol *between* its two arguments (rather than before).
- **prefix notation** describes a *k-ary* function that is marked as a symbol (such as a word (like list)) *before* its arguments, frequently having parentheses surrounding the arguments, but sometimes (like in lisp) having parentheses surrounding symbol AND arguments.
- **postfix notation** describes a function that is marked as a symbol *after* its arguments. Way less common.
- **image** describes the output of a function applied to some input (called the **pre-image** of the output). (The image of 2 under the 'square' function is 4. The square function's pre-image of 4 is 2.)

Since the square function (a unary, or arity 1 function) is a handy one to have around, we defun it next:

```
(defun square (number)
  "Squaring is multiplying a number by itself, AKA
raising the number to the power of 2."
  (* number number))
```

There are many one-liners just itching to be defuned (documentation string (doc-string for short) deliberately omitted):

```
(defun times-2 (n)
  (* 2 n))

(defun times-3 (n)
  (* 3 n))
```

This one deserves a doc-string, if only to remind us what the 'x' abbreviates:

```
(defun xor (p q)
  "Exclusive or."
  (if p (not q) q))

(defun not-with-0-1 (p)
  (- 1 p))
```

As written, all of these one-liner functions are problematic; take, for example, `not-with-0-1`. Its domain is exactly the set `[0 1]`, and exactly that set is its codomain (and range) too. Or that's what they *should* be. Restricting callers of this function to only pass `0` or `1` is a non-trivial task, but the function itself can help by testing its argument and balking if given something besides `0` or `1`. Balking can mean many things — throwing an exception being the most extreme.

2.2.2 QSU

We've been viewing a function a certain way, mostly with a programming bias, but keep in mind that functions can be viewed in many other ways as well. The fact that functions have multiple representations is a consequence of how useful each representation has proven to be in its own sphere and element. A function is representable by each of the following:

- assignment
- association
- mapping (or map)
- process
- rule
- formula
- table
- graph

and, of course,

- code

CSP examples and exercises.

The bottom line: mathematical functions are useful abstractions of the process of transformation. Computer scientific functions follow suit.

2.2.3 WYA

Let's now define a **sequence** and distinguish it from a **set**. The difference is that a sequence is an **ordered** collection. So a list or a vector is just a sequence, and truth be told, a sequence is really just a special kind of function.

VTO: a finite **sequence** of elements of a set **S** is a function from the set `{1, 2, 3, ..., n}` to **S** — the number **n** is the **length** of the string. Alternatively, the domain of the sequence function can be `{0, 1, 2, ..., n - 1}` (or with `n` concretized as 10, `[0 1 2 3 4 5 6 7 8 9]` in lisp vector style). The sequence length is still **n** (ten) in this case.

Generically, $\mathbf{a_n}$ denotes the image of the (nonnegative) integer **n**, AKA the $\mathbf{n^{th}}$ term of the sequence. An *infinite* sequence of elements of **S** is a function from {1, 2, 3, …} to **S**. Shorthand notation for the entire sequence is $\mathbf{\{a_n\}}$. Again, the domain can be either $\mathbf{Z^+}$ or **N**.

For example, $\mathbf{\{a_n\}}$, where each term $\mathbf{a_n}$ is just three times **n** (i.e., **a(n) = 3n**, or in lisp, `(defun a (n) (times-3 n))`), is a sequence whose list of terms (generically $\mathbf{a_1, a_2, a_3, a_4, \ldots}$) would be 3, 6, 9, 12, … (all the multiples of three — infinite in number).

Another term for a *finite* **sequence** is a **string**, that venerable and versatile data structure available in almost all programming languages. Strings are always finite — if it's an infinite sequence, it is most *emphatically* not a string.

With lisp symbols there must automatically be support for strings. Every symbol has a unique string naming it, and everywhere the same symbol is used, the same string name (retrieved via the function `symbol-name`) comes attached.

2.2.3.1 CFI

What do the following "atoms" and "molecules" evaluate to?

```
"abc"
;
"123"
;
(let ((letters "abc"))
  (list (symbol-name 'letters) (symbol-value 'letters)
letters))
```

We've seen this single quote before — in 'letters it means a mention of that symbol, not a use, because using it means evaluating it (as in the third element in the above list).

```
(let ((letters "abc") (numbers "123"))
  (list (concat letters numbers) (concat numbers
letters)))
```

2.2.3.2 DGJ

Does the sum of three numbers ever equal their product? If so, replace the numbers with those that would make the following evaluate to `t`:

```
(let ((x 2)
      (y 3)
      (z 7))
  (= (+ x y z) (* x y z)))
```

2.2.3.3 EHK

What kind of operator/function is '='?

```
(let ((a 1)
      (b 2)
      (c 1))
  (list (= 1 2) (= a b) (= ?a ?A) (= a c)))
```

2.2.4 CEG

Functions that return false (`nil`) or true (`t` — actually anything that evaluates to **non** `nil`) are ubiquitous. So much so that they have a special name — **predicate**.

VTO: a **predicate** is a function whose codomain is the set `[true false]`, and is AKA a **property**. The domain of a predicate can be any conceivable set. We could also call these functions **deciders** (short for *decision-makers*) that ascertain whether or not a given object has a specified property.

2.2.5 LOR

Investigate set builder notation, and discuss why it's problematic.

2.2.6 IKM

"Applying the plus function to sequences yields summations."

Let's unpack that sentence. To apply a function to a sequence we need that function's symbol, the sequence as a list, and the function `apply`. Here's where we start to see the power of the functional programming paradigm.

CSP applying '+ (yes, no closing quote is correct) to sequences to get summations.

2.2.7 OQS

Continuing to mind our `p`'s and `q`'s, it's time to talk predicates and quantifiers.

In our neverending quest for succinctness, we desire a shorter way to say, for example, all the colors of the rainbow are beautiful. Shorter, that is, than just listing all true propositions, with `B` for Beautiful as predicate:

- `B(red)`
- `B(orange)`
- `B(yellow)`
- `B(green)`
- `B(blue)`
- `B(indigo)`
- `B(violet)`

We know all the colors in the domain, so why not just say, for all the colors in the rainbow, each is beautiful. Or even more succinctly, every rainbow color is beautiful. These *quantifiers* "all" and "every" apply to the whole of our universe of discourse.

VTO: a **universe of discourse** (AKA **domain of discourse**) is the set of all things we've set forth to talk about, i.e., *all things under consideration*. It must be specified (unless the real universe is the universe of discourse) to make sense of propositions involving predicates. The **universal quantifier** ∀ says something is true for all members of this universal set, no exceptions.

Saying something like

- ∀ `x B(x)`

rather than

- `B(red)` ∘ `B(orange)` ∘ `B(yellow)` ∘ `B(green)` ∘ `B(blue)` ∘ `B(indigo)` ∘ `B(violet)`

(because, of course, each and every one is true) is much shorter, which is especially important when the domain is large (or infinite).

- ∀ `x Even(x)`

is true when `x` comes from `[2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42]`, but false when the domain is `[2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 41]`, instead. And indeed

• ∀ `x Even(x)`

is infinitely shorter than a conjunction of unquantified propositions `Even(2)` ∘ `Even(4)` ∘ …, one for each member of the set of even integers.

2.2.8 UWY

Another easy example:

Let P = "is a ball".

So P(x) means **x** is a ball. Is P(x) true?

We can't say. A predicate (or more precisely, a propositional function (or formula) consisting of a predicate and some variable(s)) is **not** a proposition.

∃ x P(x)

Meaning: There exists an **x** such that **x** is a ball.

∃ is the **Existential** quantifier.

Is this true?

It depends.

What is the Universe of Discourse?

Let it be the set [markers, erasers, whiteboards] (in a classroom, for example).

¬ ∃ x P(x) means "There does not exist an **x** such that **x** is a ball". (True, in this Universe.)

∃ x ¬ P(x) means "There exists an **x** such that **x** is not a ball".

This is the same as saying "Something is not a ball."

But it's not the same as saying "Everything is not a ball."

∀ x ¬ P(x)

Meaning: For every **x**, it is not the case that **x** is a ball.

Remember, ∀ is the **Universal** quantifier.

¬ ∀ x P(x) means "Not everything is a ball".

Let R(y) = "y is red".

∀ y R(y) means "Everything is red".

¬ ∀ y R(y) means "Not everything is red".

∃ y ¬ R(y) means "There is some non-red thing."
∃ y R(y) means "There is something that is red" or "There exists some red thing".

2.2.9 BDF

Let's examine a "Sillygism":

Object-oriented is good.
Ada is good.
∴ Ada is Object-oriented.

Very *wrong*! Better as a "Syllogism":

Object-oriented is good.
Ada is object-oriented.
∴ Ada is good.

That's right. In general:

- G(x) = x is good.
- O(x) = x is object-oriented.

∀ x (O(x) → G(x).
O(Ada)
∴ G(Ada)

A Well-Known Syllogism:

All men are mortal.
Socrates is a man.
∴ Socrates is mortal.

A Slightly Less Well-Known Syllogism:
• S(x) = x is a student of Discrete Mathematics.
• T(x) = x has taken Data Structures.
∀ x (S(x) → T(x))
Or more succinctly:
∀ x T(x)
(if the Domain of Discourse is decided on in advance to be "students in this class").

2.2.10 HJL

The Gospel as Domain of Discourse is an abundantly fruitful mindscape.
In his teachings, Paul the apostle uses conditionals (and their variations) and syllogistic reasoning **a lot**. For example: I Corinthians 15:29
Else what shall they do which are baptized for the dead, if the dead rise not at all? why are they then baptized for the dead?
All things that are done by people are done for a reason (i.e., they have a point). Baptisms for the dead are done by people. Therefore, baptisms for the dead have a point. If the dead rise not (i.e., there is no resurrection) then there would be no point to do baptisms for them. Thus, if baptisms for the dead have a point, then there is a resurrection of the dead. Therefore, the resurrection of the dead is real.

2.2.10.1 MPS

Investigate the conditional variations called *converse*, *inverse* and *contrapositive* (but not *intrapositive*).

2.2.11 NPR

As a *rule of thumb*, **universal** quantifiers are followed by implications. For example, the symbolic form of "Every prime > 2 is odd" is
∀ x ((Prime(x) ◦ GreaterThanTwo(x)) → Odd(x))
or
∀ x > 2 (Prime(x) → Odd(x))
but *not*
∀ x > 2 (Prime(x) ◦ Odd(x)).

As a *rule of thumb*, **existential** quantifiers are followed by conjunctions. For example, the symbolic form of "There exists an even number that is prime" is
$\exists$ x (Even(x) $\circ$ Prime(x)),
but *not*
$\exists$ x (Even(x) $\rightarrow$ Prime(x)).
Remember to change the quantifier when negating a quantified proposition.
Take another example, the negation of the statement that "some cats like liver". It is *not* the statement that "some cats do not like liver"; it is that "no cats like liver," or that "all cats dislike liver."
So with "cats" as the Universe of Discourse and letting L = "likes liver";
$\exists$ x L(x) says "some cats like liver."
$\neg \exists$ x L(x) $\equiv \forall$ x $\neg$ L(x)
"All cats dislike liver", or "No cats like liver".
$\neg \forall$ x L(x)
"It is not the case that all cats like liver", or "Not all cats like liver".
$\neg \forall$ x $\neg$ L(x)
"Not all cats dislike liver", or "Some cats like liver".

2.2.12 TVX

There are two important concepts to keep in mind when dealing with predicates and quantifiers. To turn a first-order logic formula into a proposition, variables must be **bound** either by
- assigning them a value, or
- quantifying them.

What means Freedom Versus Bondage?
In formulas, variables can be *free* or *bound*. Bondage means being under the control of a particular quantifier. If ϕ is some Boolean formula and *x* is a variable in ϕ, then in both $\forall$ *x* (ϕ) and $\exists$ *x* (ϕ) the variable *x* is *bound — free* describes a variable in ϕ that is not so quantified.
As in propositional logic (AKA the *propositional* (or *predicate*) *calculus*), some predicate formulas are true, others false. It really

depends on how its predicates and functions are interpreted. But if a formula has any free variables, in general its truth is indeterminable under *any* interpretation.
For example, let the symbol 'L' be interpreted as *less-than*.
Whether the formula
∀ x L(x, y)
is true or not is indeterminable. But inserting an ∃ *y* right after the ∀ *x* makes it true. Yes, we just nested one quantifer inside another. And importantly, when nesting *different-flavored* quantifiers, the order in which they appear matters (but not if they are the same flavor).
Using a predicate, e.g., if (Q x y) is "x + y = x - y"
∀ x ∀ y (Q x y) is false;
∀ x ∃ y (Q x y) is true;
∃ x ∀ y (Q x y) is false; and
∃ x ∃ y (Q x y) is true.

2.2.12.1 NQT

Consider Doctrine and Covenants 130:20-21

1 There is a law, irrevocably decreed in heaven before the foundations of this world, upon which all blessings are predicated.

2 And when we obtain any blessing from God, it is by obedience to that law upon which it is predicated.

Given the predicate P(x, y) = "blessing x is predicated on law y", these two verses are best expressed as a quantified statement by which of the following?

A.
∀ x ∀ y P(x, y)
B.
∀ x ∃ y P(x, y)
C.
∃ x ∀ y P(x, y)
D.
∃ x ∃ y P(x, y)

2.3 EN1

So, ONE more time: why learn a lovely little language like lisp? Because it's cool!

You are invited to *creatively* explore the cool meaning of *binary* predicates and *nested* quantifiers. This invitation to exploration is codenamed *Coolness*.

Using the supplied sample code as a guide and a starting point, implement these four functions, shown side-by-side with their symbolic logic equivalents:

Function Name	In Symbols
for-all-for-all	$\forall x \forall y$
for-all-for-some	$\forall x \exists y$
for-some-for-all	$\exists x \forall y$
for-some-for-some	$\exists x \exists y$

Let's examine these quantifications in the context of loops and a generic predicate `P`. These will be *nested* loops, an outer one wrapping an inner one, because *nested* quantification is what is being expressed. The `(P x y)` (predicate) function call goes in the inner loop, which controls the `y`, while the outer loop controls the `x`.

"For all `x` for all `y`" wants to find `(P x y)` always true. That's what it means for the nested quantification to be true, and naturally, this only works if the domains of `x` and `y` are finite. Even then, it really only **works** if these domains are reasonably finite — not *too* big. Iteration is serial, after all, and time is short.

So, `for-all-for-all` loops through `x`'s domain, and for each `x` loops through each `y` in `y`'s domain. On each inner-loop iteration it calls `(P x y)` and checks the value. If the value is ever false, then `for-all-for-all` is false — immediately — no need to check any further. Some `y` has been found for some `x` where the predicate is false. If both loops finish with nary a false evaluation, `for-all-`

`for-all` is ultimately true. There is no `x` for which, for any `y`, `(P x y)` is false.
The other function with similarly relatively simple logic is `for-some-for-some`. This function loops through `x`'s domain, and for each `x` loops through each `y` in `y`'s domain. On each inner-loop iteration it calls `(P x y)` and checks the value. If a true value is found, then `for-some-for-some` is true — immediately — no need to check any further. If both loops finish never having triggered true, `for-some-for-some` is ultimately false. There is no `x` for which there is some `y` for which `(P x y)` is true.
The other two are trickier. "For all `x` for some `y`" wants `(P x y)` to always be true sometimes, and "for some `x` for all `y`" wants `(P x y)` to sometimes be true always. AGP to find the best, most elegant way to implement for-all-for-some and for-some-for-all.
Test your evolving implementations of these four functions using a suitable binary (AKA two-argument, 2-ary, or arity 2) predicate, for example:

```
(defun greater-than (x y)
  "A function of arity 2 that wraps the one already
bound to '>."
  (> x y))
```

When you believe your loop logic is solid with `greater-than`, your further creative task is to invent your own binary predicates and domains, and get them to work as well.

2.4 GHI

"I hate that there are so many different ways to say the same thing," said Ila, her complainer imp feeling especially strong today.
Abu said, "I know what you mean. But I can see having more than one representation enhancing my understanding. Just as Feynmann's knowledge of chemistry and physics enhanced his understanding and appreciation of flowers, so my puny discrete mathematical mind is enriched, not impoverished, by having more ways of seeing. I really am starting to see the beauty of it. The more representations, the clearer our understanding."
Ila replied, "Maybe, but I'm not convinced yet. Til, you no doubt

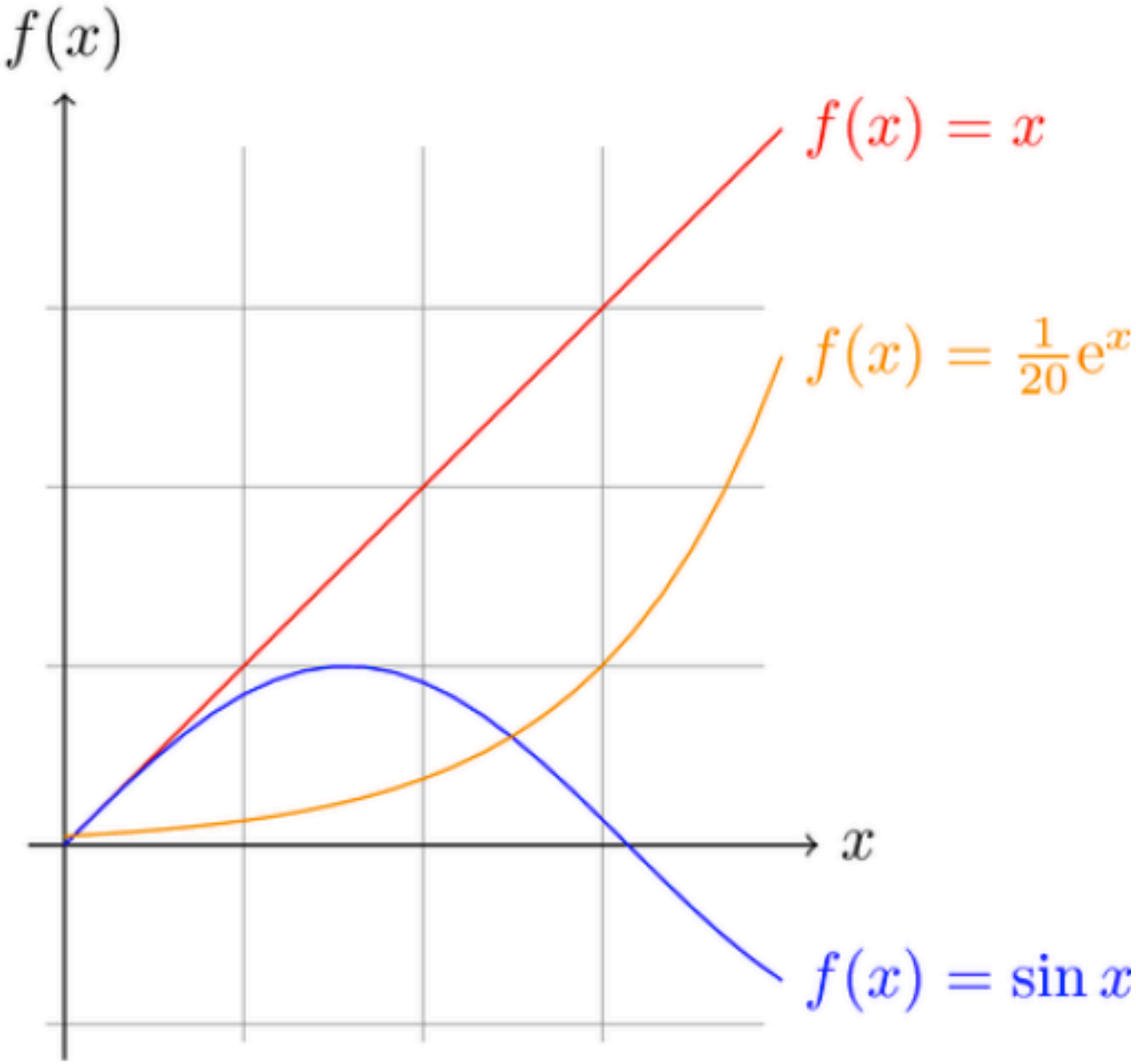
agree with Abu, right?"
Til cautioned, "Yes, but we must be careful. Each representation has its own precision … and its own vagueness."
"How so?" asked Abu.
"We use words, symbols, pictures, etc., to understand our world and achieve mutual understanding amongst ourselves," said Til. "But each representation of an idea we seek to illuminate is just a facet of the whole, and subject to misinterpretation if solely focused on. We need to be holistic AND reductionistic, both — not one at the exclusion of the other."
Abu and Ila both looked mildly puzzled, but Til went on. "Look at how we write 1/2 versus ½ or

$$\frac{1}{2}$$

. The first suggests one divided by two, the second or third the fraction one-half, what we get when we divide one by two. The process versus the result. Neither is more important than the other."
"Representational space deals with relations between ideas, concepts, aspects of reality. Representational fluency speaks to how nimbly we deal with aspects of reality like time and space, and our movement through each. Causality, cause and effect — three-dimensional spatial relationships, above, below, right, left, before or behind — proximity and locality, are you with me here?" said Til. He had noticed Ila starting to drift, although Abu was still focused.
Ila replied, "Yes, sorry. Could you explain more what you meant by, if there's any such thing as a *central* idea in math, representational fluency is it?"
"It's central," said Til, "because it's *key* to a mathematician's success. The successful ones work for and attain the ability to move fluently among various referent representations, while allowing a given representation to stand for different referents depending on the context. Especially when referring to functions,

it's crucial to be able to move comfortably between equations, tables, graphs, verbal descriptions, etc."

x	**.05** $\mathbf{e^x}$
0.00	0.0 50
0.10	0.1 36
0.20	0.3 69
0.30	1.0 04



"Speaking of graphs," said Abu, "this picture you're showing us is what I've thought were graphs ever since I took algebra."
"Ila, do you agree this is the everyday ordinary meaning of "graphs"?" asked Til.
"Yes, except I know they're also these webby-looking things with dots and lines," said Ila. "Just another example of confusing terminology, this time using the same word to refer to two different things."

"Right, and we'll get to those types of graphs, important discrete structures as they are, later," said Til, ignoring her complaint. "But for now, here's what I want you to do. I've just sent to your pads a high-level summary of how mathematicians from Euclid (at least) on down work at the task of understanding. Take this list and find examples in math history of each step:

- Think about how understanding will be assessed.
- Decide that an important measure of understanding is easy, fluid translation (movement) between representations.
- Make in-the-large pattern recognition easier because of fluency of representations. (Expert mathematicians perceive big, meaningful patterns. Novices see little, not so meaningful patterns.)
- Gain expertise by lots of problem solving practice.
- Evolve representations via problem solving practice.
- Introduce, test and refine representations in this evolutionary process.
- Nurture evolving representations as they grow progressively more sophisticated." -~-~-~-~-~-

2.4.1 ZBD

How have mathematicians generalized functions? They defined *relations*. With one big difference, relations are essentially functions. The difference is, there is no "fan-out" constraint on relations as there is on functions. What that means is that an element of the domain of a relation can map to *any number* of (zero, one or more) elements in its codomain.

VTO: A *binary relation* **R** from a set **A** to a set **B** is a subset: $\mathbf{R} \subseteq \mathbf{A} \times \mathbf{B}$. A binary relation **R** *on a set* **A** (a **self relation**) is a subset of $\mathbf{A} \times \mathbf{A}$, or a relation from **A** to **A** (from the set to itself).

VTO: $\mathbf{A} \times \mathbf{B}$, the **Cartesian product** (or just product) of set **A** with set **B** is the set of all ordered pairs **(a b)** where $\mathbf{a} \in \mathbf{A}$ and $\mathbf{b} \in \mathbf{B}$. This generalizes to **n** sets in the obvious way.

For an $\mathbf{A} \times \mathbf{B}$ example:

- **A** = [a b c]

• **B** = [1 2 3]
• **R** = [(a 1) (a 2) (b 2) (b 3) (c 3)]

For a self relation example:

• **A** = [a b c]
• **R** = [(a a) (a b) (a c)]

2.4.1.1 UXA

How many possible binary relations are there on a set with 3 elements? Replace the … to answer the question in the general case:

```
(defun number-of-binary-relations-on (set)
  "Computes the number of binary relations on a set
with *n* elements."
  (let ((n (size set))) ...)
```

2.4.2 FHJ

VTO extravaganza of special **properties of binary relations** — given:

• A Universe **U**
• A binary relation **R** on a subset of **U**
• A shorthand notation **xRy** for "x is related to y" — shorter than saying "the pair (x y) is in the relation R" (in symbols, (x y) $\in$ **R**), and shorter too than "x is related to y (under the relation R)". Since "(under the relation R)" is the assumed context, choose succintness.

R is **reflexive** iff $\forall$ x [x $\in$ U $\rightarrow$ xRx].

If **U** = $\varnothing$ then the implication is vacuously true — so the void relation on a void Universe is reflexive. Reflect on that. If **U** is not void, then to be a reflexive relation, *all* elements in the subset of **U** must be present.

R is **symmetric** iff $\forall$ x $\forall$ y [xRy $\rightarrow$ yRx].

R is **antisymmetric** iff $\forall$ x $\forall$ y [xRy $\circ$ yRx $\rightarrow$ x = y].

From this definition, using logic, you should be able to show that if xRy and x $\neq$ y then it is false that yRx.

R is **transitive** iff $\forall$ x $\forall$ y $\forall$ z [xRy $\circ$ yRz $\rightarrow$ xRz].

2.4.2.1 VYB

Take some examples of these properties. Here are four relations

over the set A = [1 2 3 4]:

- R = [(1 1)]
- S = [(1 2) (2 3) (3 2)]
- T = [(1 3) (3 2) (2 1)]
- U = [(1 4) (2 3)]

Using definitions and logic, argue for or against the correctness of these classifications:

- R is not reflexive, symmetric, antisymmetric and transitive.
- S is not reflexive, not symmetric, not antisymmetric and not transitive.
- T is not reflexive, not symmetric, antisymmetric and not transitive.
- U is not reflexive, not symmetric, antisymmetric and transitive.

2.4.3 LNP

CSP naturally, binary relations readily generalize to **n**-ary relations.

2.4.4 RTV

CSP a very brief treatment of equivalence relations.

2.4.5 XZB

To end this foray into all things relational, it is illuminating to investigate how mathematicians think and make sense of the world. However, this investigation will be limited for now to one mathematician in particular — Roger Penrose.

So this is how one world-class mathematician (who is also a mathematical physicist) thinks. In this excerpt from his book *The Large, the Small and the Human Mind*, with abundant intellectual humility, Penrose writes:

What is *consciousness*?

Well, I don't know how to define it.

I think this is not the moment to attempt to define consciousness, since we do not know what it is. I believe that it is a physically accessible concept; yet, to define it would probably be to define the wrong thing. I am, however, going to describe it, to some degree. It seems to me that there are at least two different aspects to consciousness. On the one hand, there are *passive* manifestations

of consciousness, which involve *awareness*. I use this category to include things like perceptions of colour, of harmonies, the use of memory, and so on. On the other hand, there are its *active* manifestations, which involve concepts like free will and the carrying out of actions under our free will. The use of such terms reflects different aspects of our consciousness.

I shall concentrate here mainly on something else which involves consciousness in an essential way. It is different from both passive and active aspects of consciousness, and perhaps is something somewhere in between. I refer to the use of the term *understanding*, or perhaps *insight*, which is often a better word. I am not going to define these terms either — I don't know what they mean. There are two other words I do not understand — *awareness* and *intelligence*. Well, why am I talking about things when I do not know what they really mean? It is probably because I am a mathematician and mathematicians do not mind so much about that sort of thing.

They do not need precise definitions of the things they are talking about, provided they can say something about the *connections* between them.

The first key point here is that it seems to me that intelligence is something which requires understanding. To use the term intelligence in a context in which we deny that any understanding is present seems to me to be unreasonable. Likewise, understanding without any awareness is also a bit of nonsense. Understanding requires some sort of awareness. That is the second key point. So, that means that intelligence requires awareness. Although I am not defining any of these terms, it seems to me to be reasonable to insist upon these relations between them.

3 TWO

"Let's shift gears for a moment," said Til.

"Let's talk about predicting what comes next in a string of words to motivate our first look at basic probability theory," said Til. "What do you mean?" said Ila. "Take what you just said," said Til. "Had

you started to ask 'What do you …' and the question was interrupted there, only some words make sense coming next."
"So, what is the likelihood that 'mean' comes next, versus 'want' or 'do', or et cetera," said Abu. "Right!", said Til.
"Take your cell phones out, and start composing a text message. What do you see?"
-~-~-~-~-~-

3.1 JKL

CSP more fun with TAI interactions.

3.1.1 DFH

CSP Basic counting, EEC (Elementary Enumerative Combinatorics — contrast with EGC Elementary Generative Combinatorics)

- The Addition and Multiplication Principle.
- Permutations and Combinations without repetition.

3.1.2 JLN

CSP Basic Probability Theory.

3.2 MNO

"Euclid proved there are infinitely many primes. Can you?!" Til said. His words were met with utter silence. Dropping that bombshell meant his two pupils had to readjust to the darkness of uncertainty while trying to process what Til just said so nonchalantly. Without waiting for the daze to subside, he went on.
"Before the end of today's session, you will have learned the classic method he used to accomplish this amazing feat."
"Another way to say there are infinitely many primes is to say there is no largest prime," said Til. "Suppose not …" he trailed off, pausing for dramatic effect.
"Wait, what do you mean, suppose not?" said Ila. "I thought we were trying to prove a negative anyway, that there is no largest prime?"
"What I mean is, suppose, in contradiction to what we seek to establish, that there **is** a largest prime. Call it **x**."
"Hold on," said Ila. "I don't see how that helps us prove that there

is no largest prime?"
"What we're going to do is show that it is **logically impossible** for this **x**, this so-called largest prime to exist!" said Til.
Abu chimed in. "This sounds like some Latin phrase I read somewhere — I believe it's *reductio ad absurdum* — RAA to TLA it."
Til replied, "Right As Always!" To which Ila takes umbrage. "Hey, Abu's not **always** right. I'm right too sometimes."
"I was just kidding," said Til. "But you realize that saying Abu is always right says nothing about how often you're right, right?"
"I know," said Ila. "It's just that he's always so smiley and self-assured and smug. Like this stuff is a piece of cake to him."
"Hey, it **is** a piece of cake — and a delicious one at that!" said Abu.
"Can we get on with the proof?" said Til. Both Ila and Abu nodded yes, Abu still grinning — bad enough, thought Ila, but worse still, he winked at me as Til looked down at his pads. Grrr!
"So, **x** is the greatest prime. Now follow these seven steps," said Til.

1 "First form the product of all primes less than or equal to **x**, and then add 1 to the product.
2 This yields a new number **y**. To concretize, **y** = (2 × 3 × 5 × 7 × … × x) + 1.
3 If **y** is itself a prime, then **x** is not the greatest prime, for **y** is obviously greater than **x**.
4 If **y** is composite (i.e., not a prime), then again **x** is not the greatest prime. That's because being composite means **y** *must* have a prime divisor **z**; and **z** must be different from each of the primes 2, 3, 5, 7, …, up to **x**. Hence **z** must be a prime *greater than* **x**.
5 But **y** is either prime or composite, there's no third option.
6 Hence **x** is not the greatest prime.
7 Hence there is no greatest prime."

"Whoa, back up, please!" said Ila. "I'm doing this and I don't see where **y** is ever composite:"

• 2 + 1 = 3

• $2 \times 3 + 1 = 7$
• $2 \times 3 \times 5 + 1 = 31$
• $2 \times 3 \times 5 \times 7 + 1 = 211$
• $2 \times 3 \times 5 \times 7 \times 11 + 1 = 2311$

"All of these, 3, 7, 31, 211, 2311, … are prime," said Ila. "Abu, don't you agree?"

"Well, actually, look at the very next one in the sequence." said Abu. "$2 \times 3 \times 5 \times 7 \times 11 \times 13 + 1 = 30031$, which is 59 (prime) × 509 (also prime), if I calculate correctly."

Ila was completely taken aback. Why she didn't think to go one step further caused her jaw to drop with a thud.

Noticing Ila's nonplussitude, Til reassured her. "Abu calculated correctly, but you would have too, and the fact you stopped one short is just another testimony to the all too human tendency to leap to conclusions with inductive assertions."

"So, yeah, I see it now," said Ila, recovering. "That special number **y** — made by adding 1 to the product of all primes in a **supposedly complete** list of primes — always gives rise, one way or another, to a prime not in the list!"

"You got it!" said Til. "Me too," grinned Abu. "It's a weird way to prove something, though. It just seems backwards."

"It **is** backwards," said Til. "It's called proof by contradiction, or reducing to the absurd, to translate your Latin phrase. It's a wonderfully respectable, nay, even venerable, proof technique."

"Right!" said Ila. "After all, it goes all the way back to Euclid!"

3.2.1 PRT

Til's aside, we fire our opening salvo into "mathemagics land" by way of distinguishing Elementary Number Theory from Advanced (or Analytic) Number Theory (ANT)). ANT uses the techniques of calculus to analyze whole numbers, whereas ENT uses only arithmetic and algebra. What more primordial concept to begin with than those multiplicative building blocks, the **atoms** of numbers, primes. Let's examine several ways to say the same thing:

VTO: a **prime** is a number (integer) greater than 1 that is divisible

only by 1 and itself.

- A prime is a positive integer > 1 with exactly 2 divisors (those being 1 and itself).
- A positive integer > 1 that is not prime is called composite.
- In first-order logic, prime(n) ↔ ∀ x ∀ y [(x > 1 ◦ y > 1) → xy ≠ n].
- A **composite** is a positive integer with 3 or more divisors (because 3 is the first number greater than 2, and a number cannot have fewer than two divisors).
- 1 is neither prime nor composite — call it **unit**.
- 2 is the only even prime: ∃ ! n [even(n) ◦ prime(n)]. The "∃ !" construct is short for "there exists a unique" or "there exists one and only one".

3.2.1.1 WZC

Finish writing in first-order logic (predicates and quantifiers) the RHS of composite(p) <–> …

3.2.1.2 DGJ

Experiment to see that there are gaps of many (with one exception, even-numbered) sizes between consecutive primes.

3.2.1.3 EHK

Write some lisp code to facilitate this — like a frequency table and average gap size in the first 10 million primes.

3.2.1.4 FIL

Also with code, investigate the (in)famous Twin Prime Conjecture and its generalized version.

3.2.1.5 MPS

Also with code, check and make sure you understand that when factoring an integer **n**, you need do trial divisions only by integers less than or equal to the square root of **n**. Why is that the case?

3.2.2 VXZ

What's so important about primes?

The Fundamental Theorem of Arithmetic, that's what. The FTA says that every positive integer can be written as the product of prime numbers in essentially one way. (So, 2 × 2 × 2 × 3 × 5 × 7 = 840 = 7 × 5 × 3 × 2 × 2 × 2 is two ways in one — the order of the

factors doesn't matter.)
Sometimes uniqueness is guaranteed by phrasing it as "every integer greater than 1 can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of nondecreasing size". But what about 1? The first statement didn't exclude it, why did the second?
Determining if a given integer is prime is a **key** component to modern cryptology — and **crucial** to the success of the enterprise of cryptology is the difficulty of factoring large integers into their prime factors.
More prime lore can be perused and pursued by the interested reader, but here's a playful smattering:

1 What's the smallest three-digit number that remains prime no matter how its digits are shuffled?
2 Why are the primes 2, 5, 71, 369119, and 415074643 so special?
3 Cicadas of the genus Magicada appear once every 7, 13, or 17 years; whether it's a coincidence or not that these are prime numbers is unknown.
4 Sherk's conjecture. The $\mathbf{n}^{\mathbf{th}}$ prime, if **n** is even, can be represented by the addition and subtraction of 1 with all the smaller primes. For example, 13 = 1 + 2 - 3 - 5 + 7 + 11.
5 You look up one on your own so sexy can be 6 (because the Latin for *six* is *sex*).
6 **Sexy primes** are such that **n** and **n** + 6 are both prime. The first few such pairs are 5-11, 11-17, and 13-19.

3.2.3 ADG

There are many more prime puzzles for our pondering pleasure provided also courtesy David Wells. A random sample will suffice for now.
Is there a prime between n and 2n for every integer $n > 1$? Yes, provably so.
Is there a prime between n^2 and $(n + 1)^2$ for every $n > 0$? No one knows.
Many primes are of the form $n^2 + 1$ (e.g., 2, 5, 17, 37, 101, …). Are there a finite or infinite number of primes of this form? No one

knows.
Is there an even number greater than 2 that is **not** the sum of two primes? No one knows. (That there is not is Goldbach's Conjecture.)
Is there a polynomial with integer coefficients (e.g., $n^2 - n + 41$) that takes on only prime values at the integers? The next exercise explores this question.

3.2.3.1 NQT

See if this function computes $n^2 - n + 41$.

```
(defun poly (n)
  (- (* n n) n -41))
```

3.2.3.2 ORU

Write another function to see how often (for which values of n) `poly` is prime. Try at least 10 million candidates.

3.2.3.3 VYB

Explore with code and answer, true or false? If **m** is a factor of **a** and **n** is a factor of **a**, then **mn** is a factor of **a**.

3.2.3.4 WZC

The number $2^3 \times 3^2 \times 5^1$ has a certain number of factors. How does knowing 360's prime factorization allow you to immediately say it has 24 factors?

3.2.3.5 XAD

The number 627023653815768 is divisible by 3, 4, 6, 8, 11 and 24, but not by 9 or 16. Without doing trial division, how do you quickly verify these facts?

3.2.4 JMP

One very special kind of prime is the so-called **Mersenne prime**, which is one of the form $2^p - 1$ where **p** is prime. Here are a few examples:

- $2^2 - 1 = 3$
- $2^3 - 1 = 7$
- $2^5 - 1 = 31$
- $2^7 - 1 = 127$
- $2^{11} - 1 = 2047 = 23 \times 89$ — $\therefore$ *not* prime.
- …
- $2^{31} - 1 = 2147483647$ (prime or not? How do you tell, easily?)

• …

The Great Internet Mersenne Prime Search (GIMPS) can be googled to examine the 48th known Mersenne prime, a number with over 17 million digits (do not even THINK about looking at them all).

3.2.5 SVY

How many primes are there up to a given number n? **The Prime Number Theorem** reveals in a sense how the primes are distributed among the composites. But how does one define a function that measures the distribution of prime numbers?

VTO: Define the function **π(n)** as the number of primes not exceeding (less than or equal to) the number **n**. Check it out:

n	**π(n)**	**n / π(n)**	**⌈ n / log n ⌉**
10	4	2.5	5
100	25	4.0	22
1000	168	6.0	145
10,000	1,229	8.1	1,086
100,000	9,592	10.4	8,686
1,000,000	78,498	12.7	72,383
10,000,000	664,579	15.0	620,421
100,000,000	5,761,455	17.4	5,428,682
1,000,000,000	50,847,534	19.7	48,254,943
10,000,000,000	455,052,512	22.0	434,294,482

This seems to show that

$$\pi(n) \approx \frac{n}{\log n}$$

.

More formally,

$$\lim$$

$n \to \infty$
$\pi(n)$
$(n/\log n)$

$= 1$

.
The discovery of [this theorem] can be traced as far back as Gauss, at age fifteen (around 1792), but the rigorous mathematical proof dates from 1896 and the independent work of C. de la Vall\'{e}e Poussin and Jacques Hadamard. Here is order extracted from confusion, providing a moral lesson on how individual eccentricities can exist side by side with law and order.
— from *The Mathematical Experience* by Philip J. Davis and Reuben Hersh.
So
$n/\log n$
is a fairly simple approximation for
$\pi(n)$
, but it is not especially close. Much closer is an approximation that uses the reknowned Riemann zeta function:
$\zeta(z) = 1 +$
$\frac{1}{2^z}$

$+$
$\frac{1}{3^z}$

$+$
$\frac{1}{4}$

z

$+\cdots$

$R(n)=1+$

$\sum$

∞

$k=1$

1

$k\zeta(k+1)$

(log$n$

)

k

$k!$

Let's see how much better R(n) approximates π(n):

n	**π(n)**	**R(n)**
100,000,000	5,761,455	5,761,552
100,000,000	5,761,455	5,761,552
200,000,000	11,078,937	11,079,090
300,000,000	16,252,325	16,252,355
400,000,000	21,336,326	21,336,185
500,000,000	26,355,867	26,355,517
600,000,000	31,324,703	31,324,622
700,000,000	36,252,931	36,252,719
800,000,000	41,146,1	41,146,2

	79	48
900,000,000	46,009,215	46,009,949
1,000,000,000	50,847,534	50,847,455

3.2.5.1 EHK

More prime pursuits for the intrepid: a famous theorem states that there are infinitely many primes of the form ak + b whenever a and b are coprime. Try to find a counterexample to get a feel for this. Use lots of different k's.

3.2.5.2 FIL

A famous *conjecture* (not theorem) states that *every even* number appears infinitely often as a gap between consecutive primes. This statement has **not** been proved. Nor has it been disproved. As hinted above, even proving infinitely many twin primes (gap of 2) remains elusive. Indeed, the Twin Prime Conjecture is an active area of mathematical research, both in finding twins and in attempting to prove (or disprove) they never run out. How does this grab you?

3.2.5.3 GJM

Are there arbitrarily large gaps between successive primes, or in other words, for any natural number r, is there is a sequence of (at least) r consecutive composite numbers?

3.2.5.4 NQT

What is the first CCS of length 2? 3? 4? … 10? Not the first, nor the only one, but this provably is one such sequence:

```
(defun factorial (n)
  (if (zerop n)
      1
    (* n (factorial (1- n)))))

(defun consecutive-composite-numbers-sequence-of-length
(r)
   (let ((r+1-factorial (factorial (1+ r))))
     (loop for n from 1 to r
        collect (+ r+1-factorial n 1))))
```

3.2.5.5 ORU

Write code (e.g., flesh out)

```
(defun all-composite (r) ...)
```
to verify that the above defined sequence contains naught but composites. Note that in a sequence of consecutive numbers, every other one is even, hence composite, every third one is a multiple of 3, hence composite, etc. But is there a better way to see this than sifting through the sequence looking for primes somehow hiding among all those composites? How would you prove it?

3.2.6 BEH

There is an **astounding** proof that there exist *arbitrarily long* sequences of *highly composite* consecutive integers! (That means having as many prime factors as we want.) In other more precise words:

Let r and n be positive integers. Then there exist r consecutive numbers, each divisible by *at least* n distinct primes.

Actually, we would need a little more knowledge before looking at the proof of this amazing (if you think about it) theorem. But as persistent investigation will reveal, what this proof demonstrates (like others) is that these multiplicative building blocks called the prime numbers have a gamut of surprises.

3.2.7 KNQ

Concepts known to ancient Greek mathematicians and no doubt before then too, Greatest Common Divisors (GCD) and Least Common Multiples (LCM) have an interesting interplay.

VTO: In lisp style, the Greatest Common Divisor `(gcd a b)` is the largest integer that *divides* both **a** and **b**, and its counterpart, the Least Common Multiple `(lcm a b)`, is the smallest positive integer that *is divisible by* both **a** and **b**. Integers whose gcd is 1 are called **coprime** (or relatively prime), while **pairwise coprime** (or pairwise relatively prime) describes a set of integers every pair of which is coprime.

3.2.7.1 PSV

Verify the following:

```
(= (gcd 56 8) 8)
(= (gcd 65 15) 5)
(= (lcm 5 7) 35)
(= (lcm 4 6) 12)
```

Note that two composites can be coprime.

```
(= (gcd 28 65) 1)
```

3.2.7.2 WZC

Write a function to check whether or not any given set (e.g., [2 3 4 35]) is a set of coprime integers.

3.2.7.3 XAD

The GCD of $2^3 * 3^2 * 5$ and $2^2 * 3^3 * 7$ is $2^2 * 3^2$, or more helpfully, $2^2 * 3^2 * 5^0 * 7^0$. Why more helpfully?

3.2.7.4 YBE

The prime factorization of 126 is $2^1 * 3^2 * 7^1$. Write this in the more helpful (*canonical*) form.

3.2.7.5 FIL

Verify that for many pairs of integers a and b:

```
(= (* (gcd a b) (lcm a b)) (* a b))
```

Can you prove this equality necessarily holds for **all** integer pairs?

3.2.8 TWZ

Six more topics and then we'll be ready to practice all this number theory:

1 Alternate Base Representation
2 The Division Theorem
3 Modular Arithmetic
4 Modular Inverse
5 Modular Exponentiation
6 Euclidean GCD, plain versus extended

Infinite Representational Fluency. Sounds crazy, no? But representations of integers *are* endless, because the choice of **base** to use is infinite.

Alternate Base Number Representation

$n = (a_k\ a_{k-1}\ \ldots\ a_1\ a_0)_b$

b	Name for ABR	Example n = (expanding above form making b explicit)
2	binary	$a_6\ 2^6 + a_5\ 2^5 + a_4\ 2^4 + a_3\ 2^3 + a_2\ 2^2 + a_1\ 2^1 + a_0\ 2^0$
3	ternary	$a_5\ 3^5 + a_4\ 3^4 + a_3\ 3^3 + a_2\ 3^2 + a_1\ 3^1 + a_0\ 3^0$
8	octal	$a_4\ 8^4 + a_3\ 8^3 + a_2\ 8^2 + a_1\ 8^1 + a_0\ 8^0$
1	hexadecimal	$a_2\ 16^2 + a_1\ 16^1 + a_0\ 16^0$
6		

6 base-64 (imagine $a_2\ 64^2 + a_1\ 64^1 + a_0\ 64^0$
4 that)

And so forth, ad infinitum.

Algorithm for Alternate Base Representations

Repeat the three steps until zero appears on the left side of the equation

$n = \ldots + a_6\ b^6 + a_5\ b^5 + a_4\ b^4 + a_3\ b^3 + a_2\ b^2 + a_1\ b^1 + a_0\ b^0$.

1 The right-most $\mathbf{a_i}$ is the remainder of the division by **b**.

2 Subtract this remainder from **n**.

3 Divide both sides of the equation by **b**.

Each iteration yields one $\mathbf{a_i}$ coefficient. Simple algebra guarantees the correctness of this algorithm.

For example, take **n** =97 and **b** = 3.

---

97 $= a_4 \times 3^4 + a_3 \times 3^3 + a_2 \times 3^2 + a_1 \times 3^1 + a_0 \times 3^0$

1) 97 $= 32$, remainder $= 1 = a_0$
/3

2) 96 $= a_4 \times 3^4 + a_3 \times 3^3 + a_2 \times 3^2 + a_1 \times 3^1$

3) 32 $= a_4 \times 3^3 + a_3 \times 3^2 + a_2 \times 3^1 + a_1 \times 3^0$

1) 32 $= 10$, remainder $= 2 = a_1$
/3

2) 30 $= a_4 \times 3^3 + a_3 \times 3^2 + a_2 \times 3^1$

3) 10 $= a_4 \times 3^2 + a_3 \times 3^1 + a_2 \times 3^0$

1) 10 $= 3$, remainder $= 1 = a_2$
/3

2) 9 $= a_4 \times 3^2 + a_3 \times 3^1$

3) 3 $= a_4 \times 3^1 + a_3 \times 3^0$

1) 3/ $= 1$, remainder $= 0 = a_3$
3

2, 3 $= a_4 \times 3^1$, $1 = a_4 \times 3^0$
3)

1) 1/ $= 0$, remainder $= 1 = a_4$
3

2, 0 $= 0$, so

3)

$$97 = 1 \times 3^4 + 0 \times 3^3 + 1 \times 3^2 + 2 \times 3^1 + 1 \times 3^0$$
$$= 81 + 9 + 6 + 1 = 97, \therefore 97_{10} = 10121_3$$

3.2.8.1 GJM

Make sure you understand how this recursive version of the above iterative algorithm works:

```
(defun alternate-base-representation (n b)
  (let ((d (div n b))
        (m (mod n b)))
    (if (zerop d)
        (list m)
      (append (alternate-base-representation d b) (list
m)))))
```

Try it with some random n and b:

```
(alternate-base-representation 987654321 127)
```

3.2.9 CFI

CSP The Division Theorem

3.2.10 LOR

CSP Modular Arithmetic

3.2.11 UXA

CSP Modular Inverse

3.2.12 DGJ

Modular Exponentiation: as powerful as simple as algorithms go. Say we want to compute, for example, $893^{2753}\%4721$. (Why we would want to do that is a mystery, but let it go for now.)

Doing this the straightforward way — raising 893 to the power 2753 and then reducing that huge number modulo 4721 — is difficult and slow to do by hand. An easy, fast way to do it (by hand and even faster by computer) first expands the exponent in base 2:

$2753 = 101011000001_2 = 1 + 2^6 + 2^7 + 2^9 + 2^{11} = 1 + 64 + 128 + 512 + 2048$

In this table of powers of 2 powers of 893, all instances of $\equiv$ should technically be $\equiv_{4721}$, as all intermediate values are reduced modulo 4721:

$893^{1} \equiv 893$

$893^{2} \equiv 432$

1

$893^{4} \equiv 432 \equiv 42$

$1^{2} \quad 07$

$893^{8} \equiv 420 \equiv 45$

$7^{2} \quad 41$

$893^{16} \equiv 454 \equiv 40$

$1^{2} \quad 74$

$893^{32} \equiv 407 \equiv 31$

$4^{2} \quad 61$

$893^{64} \equiv 316 \equiv 22$

$1^{2} \quad 85$

$893^{128} \equiv 228 \equiv 45$

$5^{2} \quad 20$

$893^{256} \equiv 452 \equiv 26$

$0^{2} \quad 33$

$893^{512} \equiv 263 \equiv 22$

$3^{2} \quad 61$

$893^{1024} \equiv 226 \equiv 39$

$1^{2} \quad 99$

$893^{2048} \equiv 399 \equiv 19$

$9^{2} \quad 74$

We can then exploit this table of powers of 893 to quickly compute $893^{2753} \% 4721$:

$\equiv 893^{1 + 64 + 128 + 512 + 2048}$

$\equiv 893^{(1 + 64 + 128 + 512 + 2048)}$

$\equiv 893^{1} \cdot 893^{64} \cdot 893^{128} \cdot 893^{512} \cdot 893^{2048}$

$\equiv 893 \cdot 2285 \cdot 4520 \cdot 2261 \cdot 1974$

$\equiv 1033 \cdot 4520 \cdot 2261 \cdot 1974$

$\equiv 91 \cdot 2261 \cdot 1974$

$\equiv 2748 \cdot 1974$

$\equiv 123$

= 123

3.2.13 MPS

Euclidean GCD Algorithm

one for the book!

Euclid's famous algorithm to find the GCD of two numbers (positive integers) goes like this. Divide the larger number by the smaller, replace the larger by the smaller and the smaller by the remainder of this division, and repeat this process until the remainder is 0. At that point, the smaller number is the greatest common divisor.

3.2.13.1 HKN

Verify the correctness of this implementation of Euclid's algorithm

```
(defun gcd-candidate (a b)
  (let ((x a) (y b) r)
    (while (not (zerop y))
      (setq r (mod x y) x y y r))
    x))
```

3.2.13.2 ORU

Compare `gcd-candidate` with the built-in gcd function. Is it faster or slower?

3.2.13.3 PSV

Try implementing gcd-a-la-Euclid (like Euclid did, only not in lisp) **without** using mod.

3.2.13.4 QTW

Try implementing a recursive version of gcd.

3.2.14 VYB

CSP the Extended Euclidean GCD algorithm.

3.3 PQR

"Solving a system of simultaneous congruences is what we're going to tackle today," said Til, synching his pad to Abu's and Ila's. "Let's say we're given three congruences, which I've numbered 1-3. This seeds our knowledge base with facts about an unknown quantity `x` that we want to solve for. We'll use helper variables `t`, `u`, and `v`; some basic facts about congruences modulo `m` (that you two will verify); and some eyeballing of modular inverses — altogether, 31 Facts of arithmetic and algebra. We'll record how we

arrive at each fact that follows from previous facts."

Fact	Fact Description	How Arrived At?
1.	$x \equiv_7 4$ (equivalently, $x \% 7 = 4$).	Given congruence.
2.	$x \equiv_{11} 2$ (equivalently, $x \% 11 = 2$).	Given congruence.
3.	$x \equiv_{13} 9$ (equivalently, $x \% 13 = 9$).	Given congruence.
4.	If $a \equiv_m b$ then $a = mk + b$ for some k.	By definition of $\equiv_m$.
5.	If $a \equiv_m b$ then $a + c \equiv_m b + c$.	Easily verified by Ila.
6.	If $a \equiv_m b$ then $a - c \equiv_m b - c$.	Easily verified by Ila.
7.	If $a \equiv_m b$ then $ac \equiv_m bc$.	Easily verified by Ila.
8.	If $a \equiv_m b$ then $a \equiv_m b \% m$ and $a \% m \equiv_m b$.	Easily verified by Abu.
9.	If $a \equiv_m b$ then $a \equiv_m b + m$.	Easily verified by Abu.
10.	$x = 7t + 4$	Fact 1 used with Fact 4.
11.	$7t + 4 \equiv_{11} 2$	Fact 10 substituted into Fact 2.
12.	$7t \equiv_{11} -2$	Fact 6 used to subtract 4 from both sides of Fact 11.
13.	$7t \equiv_{11} 9$	Fact 9 used to add 11 to the RHS of Fact 12.
14.	Need to find an MMI mod 11 of 7.	Looked for a multiple of 7 that is 1 more than a multiple of 11.
15.	$8 \times 7 = 56 = 1 + 5 \times 11$.	Eyeballed it. So 8 is TUMMI mod 11 of 7.
16.	$56t \equiv_{11} 72$	Used Fact 7 to multiply both sides of 15.
17.	$t \equiv_{11} 6$	Used Fact 8 twice with $m = 11; 56 \% 11 = 1; 72 \% 11 = 6$.
18.	$t = 11u + 6$	Used Fact 4 applied to Fact 17.
19.	$x = 7(11u + 6) + 4 = 77u + 46$	Substituted Fact 18 into Fact 10 and simplified.

20.	$77u+46 \equiv_{13} 9$	Substituted Fact 19 into Fact 3.
21.	$12u+7 \equiv_{13} 9$	Used Fact 8 with $m=13;77 \% 13=12,46 \% 13=7$.
22.	$12u \equiv_{13} 2$	Used Fact 6 to subtract 7 from both sides of Fact 21.
23.	Need to find an MMI mod 13 of 12	Looked for a multiple of 12 that is more than a multiple of 13.
24.	$12\times12=144=1+11\times13$	Eyeballed it. So 12 is TUMMI mod 13 of 12.
25.	$u \equiv_{13} 24$	Used Fact 7 to multiply both sides of Fact 22 by 12 from Fact 24.
26.	$u \equiv_{13} 11$	Used Fact 8 with $m=13;24 \% 13=11$.
27.	$u=13v+11$	Used Fact 4 applied to Fact 26.
28.	$x=77(13v+11)+46$	Substituted Fact 27 into Fact 18.
29.	$x=1001v+893$	Simplified Fact 28.
30.	$x \equiv_{1001} 893$	Reverse-applied Fact 8 to Fact 29.
31.	$x=893$ is the simultaneous solution.	Double-checked: 893 % 7 = 4, 893 % 11 = 2, 893 % 13 = 9.

"Why must 893 necessarily be the solution?" asked Abu. Til answered, "Examine this next table and arrive at your own conclusions. We're out of time for today."

N	**N %**	**N %**	**N %**	**N %**	**N %**
	3	**5**	**8**	**4**	**6**
0	0	0	0	0	0
1	1	1	1	1	1
2	2	2	2	2	2
3	0	3	3	3	3
4	1	4	4	0	4
5	2	0	5	1	5

6	0	1	6	2	0
7	1	2	7	3	1
8	2	3	0	0	2
9	0	4	1	1	3
10	1	0	2	2	4
11	2	1	3	3	5
12	0	2	4	0	0
13	1	3	5	1	1
14	2	4	6	2	2
15	0	0	7	3	3
16	1	1	0	0	4
17	2	2	1	1	5
18	0	3	2	2	0
19	1	4	3	3	1
20	2	0	4	0	2
21	0	1	5	1	3
22	1	2	6	2	4
23	2	3	7	3	5
24	0	4	0	0	0

3.3.1 EHK

The following puzzle should throw some light on the table just seen.

3.3.1.1 XAD

Explain what this code does after trying it on a few tests.

```
(defun mystery (divisors)
   (loop for n from 0 to (apply '* divisors) do
      (printf "%3d <--> %s\n" n
       (loop for m in divisors collect (mod n m)))))
(mystery '(3 5))
(mystery '(4 6))
```

Replace '* with 'lcm and do it again:

```
(mystery '(4 6))
```

What's a better name for this function?

3.3.2 NQT

We have been investigating a modular arithmetic scenario known to the first century Chinese and probably before that too. But the name has stuck to them: the Chinese Remainder Theorem. In a nutshell:

Given the remainders from dividing an integer n by an *arbitrary* set of divisors, the remainder when n is divided by the *least common multiple* of those divisors is *uniquely* determinable.

If the divisors are pairwise coprime, their LCM is their product, which is slightly easier to compute, which is why the CRT is usually presented in this more restrictive form (less arbitrary). A proof by construction is typical — show how to find the unique solution — and prove the construction correct.

Here's an example of this method using the system Til had Abu and Ila solve:

N	Congruence form	Equivalent modulus form
1.	$x \equiv_7 4$	$x \% 7 = 4$
2.	$x \equiv_{11} 2$	$x \% 11 = 2$
3.	$x \equiv_{13} 9$	$x \% 13 = 9$

3.3.2.1 YBE

```
(let* ((r1 4)
       (r2 2)
       (r3 9)
       (m1 7)
       (m2 11)
       (m3 13)
       (m (* m1 m2 m3))
       (o1 (/ m m1))
       (o2 (/ m m2))
       (o3 (/ m m3))
       ...
       (x (mod (+ (* r1 o1 y1) (* r2 o2 y2) (* r3 o3
y3)) m)))
    (if (< x 0) (+ x m) x))
```

3.3.2.2 ZCF

What goes where the … is? And why the final if?

3.3.3 WZC

$143 \equiv_{7} 1 \Rightarrow y_1 = -2$

$91 \equiv_{11} 1 \Rightarrow y_2 = 4$

$77 \equiv_{13} 1 \Rightarrow y_3 = -1$

So $x = 4 \times 143 \times -2 + 2 \times 91 \times 4 + 9 \times 77 \times -1$

$= 4 \times -286 + 2 \times 364 + 9 \times -77$

$= -1144 + 728 + -693$

$= -1109$

Now add 1001 twice to get 893. Why 1001? Why twice?

Because we want TUMMI, so it must be a number between 0 and 1001 (actually 1000).

This construction works because of the following facts:

$-28 \equiv_{7} \mathbf{1} \equiv_{11} 0 \equiv_{13} 0$

6

$$36_{4} \equiv 0_{7} \equiv \mathbf{1}_{11} \equiv 0_{13}$$

$$-77 \equiv 0_{7} \equiv 0_{11} \equiv \mathbf{1}_{13}$$

If you've studied any linear algebra, and have solved a system of simultaneous equations using a method like Gaussian Elimination, the 0-1 matrix embedded above should ring a bell.

3.3.4 FIL

There are two main applications of the CRT — computer arithmetic with large numbers, and the RSA Cryptosystem.

3.3.4.1 GJM

Research and write lisp to play with the idea of doing arithmetic with large integers using the CRT.

3.3.5 ORU

There is another theorem that, together with the CRT, underpins the RSA cryptosystem. Both that theorem (hint: FLT) and RSA are GPA fodder.

3.4 EN2

You are invited to implement and exercise a simplified version of the RSA cryptosystem to understand how it works. In the process, it would be *well* for you to grasp *why* it works as well. This invitation to exploration is codenamed *Wellness*.

For in-class practice in performing encryption/decryption with RSA, we will write code that is *adequate* for solving exercises using two-digit primes (and completely INadequate for doing ecommerce or ebanking).

For doing both encryption and decryption we will need a faster, better modular exponentiation algorithm than the `mod-pow` function supplied. Just for decryption we will also need a better way to compute an MMI (Multiplicative Modular Inverse), as the supplied `find-inverse` function is in very brute-force form.

The modus operandi for this code is to accept all required input via command-line arguments — no file I/O or the like. The code also

has output statements to make it clear what it's doing for every step in the encryption/decryption process.

Your task is to further implement functions to enable the program to deal with larger primes, hence larger messages. One such function is named `find-p-and-q`. Its job is to find a pair of prime numbers that are *just* big enough to be able to encrypt/decrypt your name (first and last), converted to a base-27 number using only uppercase alphabetic letters and the @ character (standing in for the space character). (Now you know why alternate base representations were deemed worthy of explication.)

For example, my name (RICK@NEFF) treated as a base-27 number using the mapping

@ = 0	I = 9	R = 18
A = 1	J = 10	S = 19
B = 2	K = 11	T = 20
C = 3	L = 12	U = 21
D = 4	M = 13	V = 22
E = 5	N = 14	W = 23
F = 6	O = 15	X = 24
G = 7	P = 16	Y = 25
H = 8	Q = 17	Z = 26

converts to base-10 (decimal) as

$18 \cdot 27^8 + 9 \cdot 27^7 + 3 \cdot 27^6 + 11 \cdot 27^5 + 0 \cdot 27^4 + 14 \cdot 27^3 + 5 \cdot 27^2 + 6 \cdot 27^1 + 6 \cdot 27^0$

which equals
5083731656658 + 94143178827 + 1162261467 + 157837977 + 0 + 275562 + 3645 + 162 + 6 = 5179195214304.
To encrypt/decrypt this 13-digit number requires two 7-digit primes (just enough) — but most of your names will convert to bigger numbers. Encrypt your name-number and then decrypt the result. Convert the decrypted number back to base-27 to see if your name is recovered or not. The satisfaction you will have on getting this to work is worth plunging deep into the well to get there!

4 THR

CSP more high-level TAI interaction.

4.1 STU

CSP more interaction with TAI and Trees.

4.1.1 XAD

CSP Tree terminology, examples, VTOs.

4.1.1.1 HKN

4.1.2 GJM

CSP examples of BSTs (Binary Search Trees) and Huffman Trees.

4.1.2.1 ILO

4.2 VWX

CSP more high-level TAI interaction.

4.2.1 PSV

CSP the statement: TLA VWX has degree sequences such and such.
Unpack that statement, and in the process, learn graph modeling and terminology.

4.2.2 YBE

4.3 EN3

You are invited to discover how to deal computationally and cognitively with graphs, and become *aware* of certain graph properties and metaproperties. This invitation to exploration is codenamed *Awareness*.
Now that you know what K_n = complete graphs are and what subgraphs are, it's time to apply that knowledge. Another name for

a complete graph is a word you're all familiar with, except in high school you might have pronounced it as *click* instead of the correct way that rhymes with the second syllable of *antique*. The word, of course, is *clique*.

Your task is to write code that *verifies* this graph property. To save you the trouble of looking up an operational definition, a clique is a subgraph of a given graph in which every two nodes are connected by a link. An **anti**-clique is a subgraph in which every two nodes are **not** connected by a link. (Note that this is the same as saying that **no** two nodes in this subgraph are connected. Or in other words, they form an *independent set* of nodes — nodes that are all independent of each other.) Searching through a specified graph, your code will check the alleged "clique-ness" or "anti-clique-ness" of a given list of nodes.

Graphs will be read in from various files containing up to one million (1,000,000) links. Each line of each input file represents one link, and consists of two nonnegative integers that label the nodes being linked. All graphs will be connected, and the node numbers will be contiguous from 0 to `n`, where `n` is some number less than or equal to 60,000. For example, a file will not have nodes 1, 2, 3, 4, 5 and 10. The nodes may not be in order in the file, however. But the graphs are guaranteed to have no links from a node to itself (self-loops, e.g., **3 3**) nor redundant (directed) links (e.g., both **3 5** and **5 3** are considered the same link). These guarantees are enforced so that error checking can be ignored.

Use the supplied code as a guide and a starting point. Functions already exist to handle reading an input file (the preceding paragraph just identified some issues to consider when specifying a graph in a file). Test input files are supplied via the functions `graph-n` where `n` is in `[1 2 3 4 5 6]`, `n` getting bigger representing successively bigger and more complex graphs.

Your specific task is to flesh out the function named `check-clique-or-anti-clique` per the instructions (see the `TODO`) in the code. The output as rendered by the supplied `output-results` function will be in the following format (not the right numbers, just

an example):

```
Graph 1 does contain a clique of size 6 with vertices
3 5 9 30 100 129

Graph 2 does contain an anti-clique of size 10 with
vertices
2 4 6 8 10 12 14 16 18 20

Graph 3 DOES NOT contain a clique of size 6 with
vertices
3 5 9 30 100 129

Graph 4 DOES NOT contain an anti-clique of size 10 with
vertices
2 4 6 8 10 12 14 16 18 20
```

4.4 YZ@

CSP more high-level TAI interaction.

4.4.1 HKN

CSP definitions, examples, VTOs.

4.4.2 QTW

CSP a discussion of the Chomsky Hierarchy.

5)

Where the Book Ends is a starting point for Further Learning Adventures (FLAs).

This book is not about that kind of word play

According to http://www.urbandictionary.com/define.php?term=metaphors%20be%20with%20you, the phrase I'm using for this book's title has been around awhile. I honestly do not know whether I heard or read it online first, or if I thought it up independently. No matter. Come day number four of month number five, all things Star Wars come to mind. ☺

The debate rages on whether the adjective in the subtitle should be *tireless* or *timeless*. Perhaps in a future printing the cover will have a hologram showing it both ways, depending on the angle of view. Perhaps not. Ascribing the timelessness quality to one's own work is the height of hubris. The pinnacle of presumption. The acme of

arrogance. The depth of debauchery. The very idea. (Wait, that breaks the pattern. ☺)
So it is *tireless* — leaving the r-to-m transformation to its own devices. Perhaps in that iffy future holographic printing the transformation (transfiguration?) can be an animation that looks like a natural process, like wax melting under a flame, or gravity dictating the bounce of a ball. In the meantime, it is what it is — an obvious ploy for humor's sake? Or perhaps just a PWCT?
Parallel Word-Chain Transformations chain a sequence of word transformations in lock-step. So maybe the 'a'-less subtitle could move from "tireless work on play on words" to "tineless fork in clay of woods" to "tuneless dork is claw if goods" and from thence (some would say already) into madness and nonsense. ☺
Just say no. See word/phrase/sentence transformations for what they are — discrete mathematical operations on strings and sequences of strings. No deeper meaning to them need be linked.

rounds out the dyad

Something consisting of two elements or parts is called a *dyad*.
End of freebies. Upon encountering an unfamiliar word, from now on the reader is encouraged to apply a principle introduced in http://emp.byui.edu/neffr/mymanifesto.html. So what is a monad? A triad? Tetrad? Pentad? What comes next in this progression?

Isaiah 28:10

Read this verse at https://www.lds.org/scriptures/ot/isa/28?lang=eng#10 and then map the acronym-expand function to [LUL PUP HAL TAL].

A quick review and preview

Thanks to the last responder at https://www.quora.com/How-should-I-go-about-switching-from-an-object-orientated-way-of-thinking-to-a-more-functional-way-of-thinking

any conceptual (as opposed to botanical) tree

The reader is presumably familiar with hierarchical organization charts, family trees, tournament trees, perhaps even decision trees, to name a few.

PSG ISA TLA TBE l8r

To elabor8, TBA and TBD and TBE and TBR and the like are just TLAs interpreted by concatenating Arranged and Determined and Examined and Revealed and et cetera to To Be.

dipthongs

How laaaa-eekly is it that a random reader knows what these are?

Not every language's alphabet is so easy

But the Hawaiian alphabet is even easier (smaller at least) — half less one, letters number but twelve: (A, E, H, I, K, L, M, N, O, P, U, and W). Aloha. Mahalo.

alphabet

What can happen when an Internet company uses a new Internet TLD (Top-Level Domain) name with three letters to accompany a three-letter host name: https://abc.xyz/

PENTY, a carefully chosen way to code-say Christmas

`PLENTY` minus the `L` — better as `ABCDEFGHIJKMNOPQRSTUVWXYZ` as the ultimate in Noel `silLliness`. ☺

IYI

If You're Interested. I picked up this particular TLA by reading David Foster Wallace's *Everything and More: A Compact History of Infinity*. HRR, BTW.

is my high hope

My maybe-not-quite-so-high hope is that you will enjoy reading this book as much as I have enjoyed writing it. I've done my best to conquer my bad habit of overcomplificationaryizing everything. ☺

He awoke with a start

This character is my surrogate. Pedagogically speaking, my using his voice is a choice of a nice device. A sleek technique. A slick trick. A cool tool. A … … (Here we go again.)

EDGE

Elphaba Defying Gravity Emphatically, for example. (Note the FLA. Four-Letter Acronym. Which ISN'T. Ambiguous, too. Could mean Five-Letter Acronym.)

he had acquired years ago

If you take the time to enjoy it, http://www.amazon.com/Falling-Up-Shel-Silverstein/dp/0060248025 is a sweet treat. And wouldn't

you know it, on page 42 is a poem that uses lisp.

DINK

Dual Income No Kids

SINKNEM

Single Income No Kids Not Even Married

STEM

Science Technology Engineering Mathematics

appreciate the flowers more or less?

http://www.goodreads.com/quotes/184384-i-have-a-friend-who-s-an-artist-and-has-sometimes

ABC

Always Begin Carefully — https://en.wikipedia.org/wiki/ABC

AKA

Also Known As — but you already knew that, NEH?

VTO

Vocabulary Time Out. There will be many of these.

the mathematical idea of a set

A mathematician/computer scientist wrote the following intriguing words: It is impossible to improve upon [Georg] Cantor's succinct 1883 definition: *A set is a Many which allows itself to be thought of as a One*. One of the most basic human faculties is the perception of sets. […] When you think about your associates and acquaintances you tend to organize your thought by sorting these people into overlapping sets: friends, women, scientists, drinkers, gardeners, sports fans, parents, etc. Or the books that you own, the recipes that you know, the clothes you have — all of these bewildering data sets are organized, at the most primitive level, by the simple and automatic process of set formation, of picking out certain multiplicities that allow themselves to be thought of as unities. Do sets exist even if no one thinks of them? The numbers 2, 47, 48, 333, 400, and 1981 have no obvious property in common, yet it is clearly *possible* to think of these numbers all at once, as I am now doing. But must someone actually think of a set for it to exist? […] The basic simplifying assumption of Cantorian set theory is that sets are there already, regardless of whether

anyone does or could notice them. A set is not so much a "*Many* which allows itself to be thought of as a *One*" as it is a "Many which allows itself to be thought of as a One, if someone with a large enough mind cared to try." For the set theorist, the bust of Venus is already present in every block of marble. And a set **M** consisting of ten billion random natural numbers exists even though no human can ever see **M** all at once. A set is the form of a possible thought, where "possible" is taken in the broadest possible sense. — Rudy Rucker in *Infinity and The Mind: The Science and Philosophy of the Infinite*

dispensing with commas

IMO, commas are the ugly ducklings in the world of programming language syntax.

we truncated the set N

The set **N** of natural numbers (including zero) is AKA the nonnegative numbers. Clarification: in this context numbers mean integers, numbers with no fractional parts, not even decimal points followed by nothing (or nothing but zeros). The set **P** is *not* the set of positive integers, that's called $\mathbf{Z}^+$. **P** names the **prime** numbers (but not always). $\mathbf{Z}^-$ is the negative integers, but there's no abbreviation for the nonpositive integers, which is $\mathbf{Z}^-$ together with 0. All together, **Z** is the short name for all integers, positive, negative, and zero. **Z** is short for **Zahlen**, which is the German word for numbers. Now you know.

our rule also prefers

The vectors-over-lists preference has a special reason, revealed IYI. But it's not just its indexability. A lisp list can also be indexed, but the operation is more expensive than a vector index (a quick add offset and pointer dereference), as it must walk the list, counting as it goes. Before we delve here into a bit of algorithm analysis, what is an algorithm? The recipe analogy is often used, but recipes for cooking are not the only kind. For another, far more weighty example, see https://www.lds.org/general-conference/2008/04/the-gospel-of-jesus-christ?lang=eng — but it's interesting that Elder Perry ties the gospel recipe back to the

cooking kind. This kind features an implicit "Step 0" that could be made explicit as: "Acquire all required ingredients in the proper quantities." The "List of Ingredients" is usually the first component of a recipe, with "Directions" second.

handing it off to some further processing step

How the emacs elisp Read-Evaluate-Print Loop (`REPL`) works is discovered via GPA.

Boolean values

George Boole is who this data type is named for, mostly because of his book *The Laws of Thought*.

just use true and false

Ensure these global variables are defined before trying to use them by putting `(require 'ment)` at the top of your `.el` source code file. This will work provided you have provided the package named `ment` to your emacs lisp environMENT. A **package** in lisp is a module (library, body of code) that is defined in a global namespace. The package name is a symbol (making it unique) and the named package is found and loaded on two conditions: one, it is in a file named <package-name>.el[c] that has the form `(provide '<package-name>)` in it (typically as the last line of the file), and two, the file is somewhere in the load path (in emacs, `load-path` is an elisp global variable, but other lisps have something equivalent). When does a package get loaded? The `(require '<package-name>)` special form sees to it. (The single quote preceding <package-name> suppresses evaluation of the <package-name> symbol.)

kind of wishy-washiness

I used to think I was indecisive, but now I'm not so sure. ☺

the 'exclusive' or

Contrasted with the 'inclusive' or — "p or q" is true if either p is true or q is true, OR BOTH. For 'exclusive' or, replace "OR BOTH" with "BUT NOT BOTH".

simpler versions of itself

There's a different kind of recursive flavor to this never-ending "dialogue" that leads to less and less simple versions of itself:

- Sue: I'm happy.
- Tom: I'm happy you're happy.
- Sue: I'm happy you're happy I'm happy.
- Tom: I'm happy you're happy I'm happy you're happy.
- …

eight possibilities

This is because eight equals two to the third power. Just like two choices (2^1) expand to four (2^2) when for each of two choices we must make another binary choice. Thence to 8, 16, 32, … or 2^n where `n` is the number of true-or-false yes-or-no in-or-out on-or-off … choices made.

it enables certain useful encodings

The *arithmetization* of logic is used to operationalize logic, and make it more like doing arithmetic. For example, `not` (as in `not 1 = 0, not 0 = 1`) becomes the subtraction-from-one operation. Compound propositional formulas can be encoded in useful ways as well. One useful encoding turns a Boolean formula ϕ into a polynomial equation **E** in many variables. This clever encoding is such that ϕ is satisfied **if and only if E** has integral roots.

Truth Table Generator

This app (written by students) is found at http://firstthreeodds.org/mebewiyo/code/tru

Make it so.

Captain Picard really started something here: https://www.youtube.com/watch?v=RrG4JnrN5GA

much more than that

You will find a nice metaphor for a symbol here: https://www.gnu.org/software/emacs/manual/html_node/eintr/Symbols-as-Chest.html#index-Chest-of-Drawers_002c-metaphor-for-a-symbol (which is not a bad introduction to programming in elisp: https://www.gnu.org/software/emacs/manual/html_node/eintr/index.html#SEC_Contents).

If you try it, an error results

When the elisp debugger's *Backtrace* buffer pops up with the

following information:
Debugger entered–Lisp error: <a lispy description of the error>
<a stack trace showing the error's code path>
you should probably just quit, and not worry just yet about learning the intricate ins and outs of elisp debugging. The `q` key is your friend.

numbers self-evaluate

You know, like TLA self-describes. What other data types do you think self-evaluate in lisp?

DEF

Do Everything First — https://en.wikipedia.org/wiki/DEF

xkcd comic that compares lisp

http://www.explainxkcd.com/wiki/index.php/297:_Lisp_Cycles. The really deep dive into lisp lore is found at http://www.buildyourownlisp.com/ — but it's not for the faint of heart.

With your pads

Or rather, PADS — Personal Assistive Device Stylus — imagine the possibilities …

any restrictions on what objects

Not in lisp. This powerful language allows even functions *themselves* to be inputs and outputs to/from other functions.

non-trivial task

Made more so by lisp's dynamic (as opposed to static) typing, which means variables don't have declared types, they can store *any* type during their lifetime.

can be viewed in many other ways

Such as a machine, a formula, a table, or a graph — check out this beautiful app (also written by students):
http://firstthreeodds.org/mebewiyo/code/fun

they have a special name

When talking about logic, the noun **predicate** — to quote from the dictionary — "is that part of a proposition that is affirmed or denied about the subject." For example, in the proposition "We are mortal," "mortal" (or "is-mortal" or "are-mortal") is the predicate.

Any statement about something could "predicate" (the verb form) some general term of that thing; however, convolution may result. For example, "The main purpose of the Doctrine and Covenants […] is to implement the law of consecration." It's not easy to pull out a term which, predicated of "the Doctrine and Covenants", has the same meaning. So if we're saying `P(x)`, where `x` stands for "the Doctrine and Covenants", what does `P` stand for? How about this? `P` stands for "to-implement-the-law-of-consecration-main-purpose-of". Just saying `P(x)` (or lisp-style, `(P x)`) is easier. Let it stand for any desired sentence with an `x` in it. It's not necessary (nor even desirable in most cases) to say what `P` stands for by itself.

supplied sample code

For exploration liftoff see
http://firstthreeodds.org/mebewiyo/code/one and
http://firstthreeodds.org/mebewiyo/code/two and
http://firstthreeodds.org/mebewiyo/code/thr.

how mathematicians think

See the fascinating book by William Byers at
http://www.amazon.com/How-Mathematicians-Think-Contradiction-Mathematics/dp/0691145997

"mathemagics land"

I believe it was in fifth grade that I first saw this tribute to the beauty and power of math: http://donald-in-mathemagics-land-find-real-link-please/.

multiplicative building blocks

It is interesting to compare these building blocks with the *additive* building blocks, the powers of 2. Sums of 1, 2, 4, 8, etc., in all their combinations, suffice to generate all the nonnegative integers. (Zero, of course, is the combination that takes none of these powers of 2.)

1 is neither prime nor composite

See http://numberphile.com/videos/1notprime.html (5:22) if you need convincing that 1 is indeed the *loneliest* number.

2 is the only even prime

Which makes it the oddest prime of all, according to an old math

joke. ☺

can be perused and pursued

From the book *Prime Numbers, The Most Mysterious Figures in Math* by David Wells. http://find-the-url-for-this-one-too/.

appears infinitely often

It was Alphonse de Polignac in 1849 who put forth this conjecture, which generalizes the TPC.

finding twins

As of December 2011 the pair $3756801695685 \times 2^{666669} \pm 1$ holds the twin-prime record. Each is a prime with 200,700 digits!

6 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright (C) 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

1 PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We

recommend this License principally for works whose purpose is instruction or reference.

1 APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover

Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document

whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

1 VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

1 COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally

prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.
If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.
If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.
It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.
1 MODIFICATIONS
You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:
A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions

(which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. C. State on the Title page the name of the publisher of the Modified Version, as the publisher. D. Preserve all the copyright notices of the Document. E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice. H. Include an unaltered copy of this License. I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications

given therein. L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

1 COMBINING DOCUMENTS

You may combine the Document with other documents released

under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

1 COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

1 AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume

of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.
If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.
1 TRANSLATION
Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.
If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.
1 TERMINATION
You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and

will automatically terminate your rights under this License. However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.
Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.
Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.
1 FUTURE REVISIONS OF THIS LICENSE
The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.
Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes

you to choose that version for the Document.

1 RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Author: Rick Neff

Created: 2015-09-28 Mon 21:21

Validate