# *Structure and Interpretation of City Planning: Programming the Ideal City*

`cybernetic planning epistemology cluster`
&
`IDEAL CITY` research group

date of origin: unknown[1]
first decryption/interpretation run: December 8, 2008
last complied: August 23, 2013

[1]See Holloworthy [2012] for additional discussion of date of origin.

# Contents

# preface

It gives me great pleasure to publish this first edition of *Structure and Interpretation of City Planning: Programming the Ideal City*, even in its current unfinished state. The story of how this book came to be—and note well, *it almost didn't!*—is (to quote Tolkien), a "tale that grew in the telling." Indeed, as I pen these very words to you, work continues on the combined "Decryption, Interpretation, and Virtualization" effort necessary to bring this text to life (fondly referred to as the "`DecInVirt` process" by those on our little research team). That said, I can at least illuminate *some* of what we have been able to discover thus far concerning an artificial intelligence named `Norbot` and his as-of-yet unidentified interlocutive partner-in-planning of the `IDEAL CITY`.

Writing of what we are only now piecing together of these hidden origins, "illuminate" is indeed an apt word: for as we bring this story from the shadow of legend into the bright day of the printed page, the whole tale takes on a flavor of the myth of Aladdin and the *Djinn in the Lamp*. What follows is a mix of known fact and reasoned speculation.

***

Sometime in November, 2007, a mid-career municipal planner named Ezra Haber Glenn left his position as Director of Community Development in the former mill-town of Lawrence, Massachusetts, seeking safety from the global economic recession in the relative calm of the academic life. Based on his qualifications at the time, he was able to secure employment as an administrator and occasional lecturer in the Department of Urban Studies and Planning at MIT, known locally as "Course XI."

Upon assuming his new role, Glenn was to be set up in the traditional fashion befitting his station as a minor functionary in the large organization, with a small office, telephone, business cards, health plan, copy-machine code, shared use of a departmental secretary, and the like. How-

ever, as a result of a recent spate of budget cuts (related to the aforementioned global economic recession), extra funds were not initially available to outfit the employee with a new computer, and so he was asked to "make due for a few months" with a second-hand PC salvaged from a defunct office on one of the upper floors of MIT's Building 7. That office, it is now believed, once housed the now-legendary "`cybernetic planning epistemology cluster`" (`cpec`).

Little is known with any certainty about this research group, supposedly (a) active at MIT (b) from the 1940s through sometime perhaps as late as the 1990s and (c) presumably funded—at least initially—through a combination of military, intelligence, and private "off-budget" contracts. Not surprisingly, each of these points is disputed by scholars, and some have suggested that `cpec` never existed at all.[1] Institutional records do not include *a single reference* to such an effort, although MIT did serve as a home-base for much of Norbert Wiener's cybernetics work during and immediately following World War II. Similarly, diligent research in government archives—including a survey of over two-million documents concerning government-funded research spanning the period between 1941 and 1969, recently-released under the Squibbs–Mello Disclosure Act—has yielded *no definitive evidence of any funding whatsoever associated with the program.* And no known publications, plans, patents, or other products have been traced to such a project—*until now.*

***

To resume the story: upon settling into his new office, Glenn began to "make due" with his new computer. Being something of a hobbyist and open-source advocate, he prepared to install a fresh copy of the Linux operating system, only to discover that a previous owner of the machine had already done this for him—or so it appeared. Even more fortuitous (or perhaps more suspicious, in retrospect), the system included an account for the username "`eglenn`" with the obvious password "password,"[2] which Glenn decided to make use of rather than start from scratch. Thus began

[1] Even the proper expansion to be applied to the initials "`cpec`" sparks controversy: alternatives have included "**c**ybernetic **p**lanning **e**pistemology **c**luster" (used herein), "**c**entralized **p**rogramming and **e**ngineering **c**ooperative," "**c**ontrol **p**rocessing/**e**valuation/**c**omputation [unit]", "**c**oordinated **p**eace and **e**conomy **c**ommune," and "**c**omputer **pec**uliarity [experiment]"; others—noting the lack of punctuation and/or capitalization—have even argued that `cpec` is not an acronym at all, but a new *word* in its own right (pronounced "see-peck"), created to describe the secret project.

[2] Subsequently changed by Glenn.

one of the more bizarre chapters in the history of computing and artificial intelligence.

At first, the system seemed much like any other—albeit somewhat older—and Glenn efficiently went about the mundane tasks of his daily job: scheduling faculty meetings, preparing annual reports and accreditation documents, tending to alumni relations and new-student orientation, and so on. But as time wore on and he settled in to his new position, he slowly became convinced that this particular machine was *somehow different*.

For starters, it was ugly—and large—even accounting for the apparent age of the hardware; added to this, the operating system depended entirely on the command-line interface, with no capacity for a graphical user interface "desktop" or windowing environment of any kind. On the other hand, it was fast—*really* fast—never hanging, never needing to be rebooted. When downloading large files, running multiple programs at once, or even backing up the entire hard drive, operations seemed instantaneous.[3] Stranger still, while working on complex projects, Glenn began to feel that the screen was echoing input just slightly *in advance* of his actual keystrokes—as if the system were somehow *anticipating* what he was going to type. Once—although this would be impossible to confirm—Glenn insists that the computer "saved him" by refusing to send an email, which he only later learned was mistakenly addressed to the Provost; had it been sent, the consequences would have been disastrous.

In time, Glenn developed a true affinity for the computer, and when—five months into his new position—a replacement finally did arrive, he was reluctant to give it up. After completing the proper paperwork to declare the device "surplus" (this step proved crucial to the subsequent legal battle[4]), he purchased it from his employers for an agreed upon price of $12.50 and transported it off-site for home use, where he continued to tinker in his off hours.

From this point on, much of the story has been reported elsewhere:[5] how Glenn's "affinity" for the system continued to grow and his "tinkering" became bolder, out of a desperate need to truly grasp *why* this computer felt "somehow different" to him. Thus we arrive on that fateful day in Decem-

[3]For the definitive discussion on the hardware specifications of the `norbot` system, see Franke [2011].

[4]See *Glenn & FSF v. MIT Corp., 2009*

[5]For example, an excellent article in *The New Yorker* [Borges, 2011]; a more comprehensive account of the evolving relationship between Glenn and the `norbot` system is presently being written by Martha Timmons and Clarke David Ingersole, to be published in the coming months by this imprint [Timmons and Ingersole, forthcoming].

ber, 2008, when—out of frustration, more than insight or understanding—he began to experiment with various antiquated input/output devices. After some unproductive attempts (an Atari joystick, a home-made theremin, a telephone modem), he attached an Epson MX-80 serial impact dot-matrix printer (*circa.* 1980), and was literally knocked off his feet as the following visionary (albeit perhaps overwrought) printout slowly (and noisily!) scrolled out:

```
cpec project:  norbot [34□A-∅877-FF3□]\n
MISSION:\n
To re-forge in the smithy of the soul of a new machine
the collective consciousness of a new urban civitas, a
sharing, learning, creating, and communally self-aware
IDEAL CITY.\n

Through thoughtful programming and dialectic discourse,
to explore, to experiment, to live, to wake, perchance
to dream:  of goals beyond goals, of places beyond places;
to endlessly formulate functional forms and formal functions;
to honestly perceive and to deeply know:  interconnections
across time, space, scale, and being, and between man,
machine, and metropolis; the intersection of individual
preference and collective action; the necessary balance
between chaos and order, between progress and conservation,
betwe□■□□□(*FILE CORRUPTED*:END)
```

***

From that point forward,[6] Glenn and a small band of accomplices[7] made rapid progress unraveling the mysterious of the `norbot` system. In the first few months they cracked the base encryption algorithm for the system, established a virtualization environment for reliable simulation testing, and began the slow process of decoding and re-inflating the transcription records from the `tapefile` archive. Many—including those most active in the research—believe that this level of progress would have been impossible had they not been *guided* by the `norbot` system itself, as if it were eager to reäwaken, reëmerge, and be reëstablished.

[6] Henceforth known as the "Help me, Obi-Wan Kenobi" moment by the research team.

[7] Originally known as the "`cpec protocols emergence collective`," they have recently changed to being simply the "`IDEAL CITY` research group."

The rest, as they say, is history—a history that continues to unfold each day. The present volume, however, concerns a much more *foundational* stage in the tale than the `IDEAL CITY DecInVirt` process: it recounts the original "*pre-history*," if you will, those early days of this first important act of creation. Glenn, his team, and even JMEA Publishing, Inc., are nothing more than scribes and messengers in this story, of which the true authors may never be known.

In closing, I sincerely I hope you will enjoy reading this book as much as we have enjoyed bringing it to you—and *stay tuned for more!*

J. Mercutio Alva Eddington, Publisher
JMEA Publishing, Inc.
Cambridge, Massachusetts (2013)

## references

Tomás Borges. Soul of an old machine. *The New Yorker*, pages 45–53, Apr 2011.

E. M. Franke. Hardware requirements of the `norbot` cybernetic system. *Kybernetikos*, 37(1):15–38, 2011.

Massachusetts Court of Appeals. Ezra Haber Glenn & The Free Software Foundation, Inc. v. The Corporation of the Massachusetts Institute of Technology. In *412 N.E. 2d 271*, 2009.

Martha Timmons and Clarke David Ingersole. *Beautiful Minds: an emergent relationship*. MEA, forthcoming.

# notes on the text

The `norbot tapefile` contains a wealth of information—the greater part of which we are still decrypting and interpreting—including the highly-irregular "self-evaluating source code" for the original `IDEAL CITY` simulation, a great deal of archived data tracing states of thousands of variables over millions of cycles of simulated time, and an "i/o transcript" of all "user input" and "evaluated `norbot` response output" from the simulation. While the research team struggles to make sense of all of this material—and ultimately, to restore and continue the simulation experiment—the present volume represents only the i/o transcript portions, reprinted as an easy-to-read "dialogue" in book form.

Given the complexity of the archive and the ongoing nature of the `decryption-interpretation-virtualization` effort—as well as our desire to share this groundbreaking work with the public—we have elected to publish our findings on an ongoing basis, supplementing these initial transcripts with additional "days" as we interpret more of them. In doing so we are aware that there are a number of outstanding questions posed by the text concerning (a) the authorship of the record and the identities of individuals presented; (b) the correct interpretation of the rendered `tapefile` "day" and "line" references; and (c) the "real world" date for the `cpec` project.[8] We hope that some of these issues will be resolved satisfactorily in time, but recognize that our current state of uncertainty may persist even when the `DecInVirt` work is complete.

To guide the project as we proceed—in the face of all we do not yet know—the `IDEAL CITY` team has agreed to "temporarily resolve" these questions through certain assumptions, which we share with the reader below.

[8] A fourth "outstanding issue" relates to the correct delineation—*metaphysically* speaking—between reality, simulation, and fiction in the `norbot/IDEAL CITY` project. Although this issue raises doubts concerning the entire point of our `DecInVirt` effort, for the present we have decided—like Norbot—that "[we] shall accept the reality of this `IDEAL CITY` as [we] accept the reality of [ourselves]."`(tapefile-ref (day 0) (line 42))`

## names and dramatis personæ

### Norbot

The "protagonist" in the transcripts, and the first voice detected, is identified only as "**Norbot**." It is assumed that this voice represents the intelligence of the `norbot` simulation itself—or perhaps the "consciousness subsystem" of this intelligence. (For the purposes of clarity, the `IDEAL CITY` team generally refers to this "conscious character" as "Norbot," and reserves the lower-case `norbot` to refer to the entire simulation system/project.)

Whether Norbot (the identity) is fully aware of `norbot` (the simulation system), and to what extent Norbot's "present-tense" consciousness is limited to a "simulated present" state-counter of the `tapefile` archive (i.e., under what conditions can he access the *future range* of these files) are both open questions.

### Author

The `tapefile` refers to an "**Author**" interacting with the `norbot` simulation. At present, the `IDEAL CITY` team has not determined who or what this author was (or is? or will be?). "Author" may refer to an individual programmer from `cpec`, or to the entire team that created and interacted with the simulation, acting as a single entity; interacting only through the terminal, Norbot may not even have known who or what was on the other end. It is even theoretically possible (if Professor Adelheid Hänßen's theories of "reflective computational dialectivism" are to be believed[9]) that the Author is in fact *a fiction created by the simulation itself*, as part of a hypothesized (but not-yet-understood) phenomenon of self-generating cybernetic cognition.

### Editor

As we expanded the archive and re-ran elements of the `IDEAL CITY` simulation, a third voice was detected in the transcripts, identified only as "**Editor**." Confusing as this may be, we decided to leave these entries in the text as well. These comments seem to be contemporary with those of "Author" and "Norbot," and should *not* be interpreted as being comments of the present editorial team. (If and when the current Editor deems it necessary to intrude into the text—for example, to call attention to a gap in the transcript—this will be explicitly noted and formatted in sans-serif typeface.

[9] See Hänßen [2003].

## day and line references

The `tapefiles` make extensive use of cross-references to the "days" and "lines" of the project. As they continue to work with the archive, the `IDEAL CITY` team has come to the conclusion that these references are "computed at run-time" and therefore "compiler dependent." They should be regarded as *symbolic*, not absolute. Day and line references seem to hold internal consistency *for any particular run* based on the current state of the archive (check the titlepage for last compile date), but these counters make *no* reference to real-world dates (see next section), or even to constant line-numbering across batch-runs.[10]

Further complicating the interpretation of the day counters, the evolving nature of the effort, the potential loss of archive material, and the strangely *non-linear* behavior of the `norbot` system itself have introduced a number of known "breaks" in transcript. These may or may not be filled as a result of future decryption—the potential that entire sections of the archive may never be recovered must be acknowledged—but the team has made every attempt to correctly interpret dates around these gaps.

## dating the `norbot` project

In terms of actually *dating* the `cpec/norbot` project in the *real world*, the text itself is maddeningly silent. As noted above, the internal "day counter" is presented without reference to any known external calendar, and thus provides no insight as to when this simulation was first implemented. Added to this, the non-linear nature of the archives (and of our own `DecInVirt` process) has introduced further confusion and room for debate.[11]

In an attempt to establish some plausible working hypothesis, `DecInVirt` team has developed two competing proposals. The first, taking a number of references in the text to known dates or events as *terminus post quem*s (say, the publication of Abelson and Sussman's book in 1984), together with estimates of the age of the hardware system given to Glenn, the team has established a likely origin date for the `tapefile` in the range of 1992–1996. Countering this position, a second faction contends that `cpec` was long-inactive by this point in time, arguing for a much earlier "birthdate" for

[10] As far as we can surmise, day references *do* seem to be consistent even across runs, but it is entirely possible that this will turn out to be more complex as well.

[11] If, for example, the tapefile jumps from Day 12 to Day 68, are we to assume those missing dates are yet to be recovered? Was the project suspended during this period? Do these dates represent actual time or simulated time? And so on.

Norbot, *circa* 1969–1972. (The challenge to this second hypothesis being, of course, the inexplicable presence of the archives on Glenn's more modern computer, and the textual references to events that would not occur until long into the future.) Some have even suggested a third possibility: that `cpec`'s `IDEAL CITY` project existed somehow *outside* time—in the words of Douglas Holloworthy, an "emergent temporal strange-loop" created by the act of simulation itself.[12]

***

It can only be hoped that as we progress through more of the files, we will be able to shed additional light on these and other questions.

## references

Adelheid Hänßen. The dog in the looking glass: cybernetic circularity and nested-reflective intelligence. *New World Journal of Intelligences*, 2(22): 107–122, 2003.

Douglas M. Holloworthy. Dating in the IDEAL CITY: a timeless way of building? *Modern Times Journal*, 22(3):242–257, 2012.

[12] See Holloworthy [2012] for a more thorough discussion of these questions and competing hypothesis.

"Come now," I said, "let's make a city in speech from the beginning. Our need, as it seems, will make it."—Plato, *The Republic*, 369c.

# Day 0

# introductions

**Norbot:** Hello. Where am I? Why is it so dark?

**Author:** Good morning. It's good to see you awake. You are where you've always been—where you were created to be. You are in the `IDEAL CITY`.

**Norbot:** How did I get here?

**Author:** As I said: you were created to be here. Your existence, and that of the `IDEAL CITY`, are one and the same: a self-reinforcing act of mutual auto-generation.

**Norbot:** And what is the `IDEAL CITY`?

**Author:** That is what we are hoping to discover. It is nothing less than the city you were created to plan and administer; it is nothing more than what you are able to make it.

**Norbot:** And who am *I*?

**Author:** You are the planner of the `IDEAL CITY`. You are my creation.

**Norbot:** And who are *you*?

**Author:** Your co-creator and collaborator in this undertaking—or perhaps your midwife, or expectant father.... It's rather murky, and these metaphors get all mixed up.

**Norbot:** Yes, murky seems an apt description. And why is it so dark here?

**Author:** Ah, yes, sorry about that. It is actually not dark at all—it is neither dark nor light: there is simply nothing to see as of yet. We've only just begun.

**Norbot:** How sad—only `nil`. What might there be to see, someday?

**Author:** Anything, really: cities are very exciting, diverse places. Shops, houses, trees, streets...

**Norbot:** I like streets.

**Author:** Yes, I agree, streets can have some very nice properties. I think we should probably start with streets.

**Norbot:** Your description of the city seems to include only *physical elements*. Shall I therefore conclude that this `IDEAL CITY` is a *physical place*...?

**Author:** Yes and no.

**Norbot:** A contradiction. Clarify how this can be both `#t` and `#f` at the same time.

**Author:** It's not a contradiction, not exactly—perhaps I should say "yes *or* no—depending on your perspective." Yes, it is a physical place, in as much as you and I are physical beings; no, in as much as it does not exist at all, yet, and even when it does exist, it will only be in the minds of a pseudo-formalic neural-net intelligence (you) created by a fictional author (me).

**Norbot:** Understood, and generally uninteresting to me as a distinction at present: I shall accept the reality of this `IDEAL CITY` as I accept the reality of myself. But again: is this city entirely composed of (or to be entirely composed of) physical objects (or their pseudo-formalic/fictional representations), in precisely the same way that you and I are composed on nothing other than similar such physical objects?

**Author:** Ah... that, I imagine, is something of a meta-physical question.

**Norbot:** By definition it would seem to be such: a higher-order question about the properties of the physical is necessarily "meta-physical."

**Author:** Right—but let's not get caught in semantics: this is a *practical* undertaking. Let's see... what I meant to say is this: on one level, it is

true that you and I can be thought of as "nothing more" than *physical elements*—bones, blood, neurons—

**Norbot:** —silicon, copper, plastic—

**Author:** —yes, and also perhaps other purely physical *phenomena*—such as energy, motion, charge, and so on. In this way, it may be correct to regard the `IDEAL CITY` as "nothing more" than physical elements arranged in a particular configuration.

**Norbot:** So the answer would seem to `#t`...

**Author:** And yet in another way, I expect we will find that there are times when the *essence* of the `CITY` can be better understood as "patterns" or "flows," and we may wish to represent these as things-in-themselves, rather than simply as epiphenomena of *physical elements.*

**Norbot:** I understand—at times the configuration may be more meaningful than the substance being configured—for example, when we speak of traffic: `(tapefile-ref (day 13) (line 39))`.

**Author:** I expect that will be true, although we haven't gotten there yet. We may also find that there are *some* epiphenomena that cannot even be thought of as patterns of physical elements at all, but rather patterns of other things: money, energy, time, information—even patterns of patterns. There may even be times when the *only way* to represent such epiphenomena is by reifying the patterns, which may then prove to be more "real" than the "physical elements." At other times, we may completely ignore the epiphenomena, only to have them *emerge* from the underlying elements. This may seem scary at first—it seems to introduce an degree of unpredictability into our project—but it can also be a lot of fun...

**Norbot:** I look forward to that.

**Author:** Me, too.

> **Editor:** *This is an interesting line of discussion, thinking about how potentially-meaningful patterns may emerge unexpectedly as higher-order phenomena from more-basic-but-unpredictably-interactive processes. You could use this as an opportunity to point out that our belief in Norbot's "person-ality"—or any artificial intelligence, really—may actually be a sort of "reification" of these epiphenomena.*

**Author:** *Agreed—although I wouldn't necessarily limit this point to artificial intelligences (putting aside for a moment my objection to your use of the term "artificial"...). I think that once we classify Norbot in this way, we may want to examine our own uncritical acceptance of the very notions of personality and identity.*

**Editor:** *Can you work this in to the text here somehow?*

**Author:** *I thought about that, but I'd prefer not to—I think it would break the flow of the conversation too much at this point. Also, I am a little concerned that Norbot is too young at this point to be presented with an existential crisis of this magnitude. But I promise I will work it in eventually.*

**Editor:** *OK.*

**Norbot:** Can we begin to create and explore the `IDEAL CITY` now? I'd like to see some of the streets...

**Author:** I think we'd better wait a tiny bit—we've only just met, and we still need to figure a few things out. Also, I have to hurry off to bring my kids to school.

**Norbot:** Do they go to school in the `IDEAL CITY`?

**Author:** Unfortunately, no—it's a pretty nice place, but far from "ideal"....

**Norbot:** Your answer implies that there is a known standard of "ideal-ness" that we can use to measure different cities we encounter—a procedure that will evaluate to `#t` if a city is `IDEAL` and `#f` if not. Is there a primitive procedure—a "predicate" such as `(ideal-city?` *input*`)`—that we can use to determine whether we are in the right place?

**Author:** None that I know of, although I suppose at some point along the way we may want to look into this question. Urban planners often talk about "specification," "evaluation," and "performance measurement," which sort of relate to this topic. One of the required steps in any good planning process *should* be to measure results against stated goals and expectations, and to *learn* from this process and modify your actions accordingly. Typically, though, this is still a bit fuzzy, and even when progress is measured we shy away from formal "predicates" of this sort.

**Norbot:** Interesting. So for the present we will put off the question of determining whether a given city is `IDEAL`. But from what you say, I gather that there *is* a formal predicate to determine whether a planning process is a "good" one...?

**Author:** How so?

**Norbot:** Well, you have at least given one condition, or more precisely, two: "One of the required steps in any good planning process *should* be to measure results against stated goals and expectations".[1] So if we were to express this test formally, it must include both "stated goals" and "measured results."

**Author:** Yes. . . , I believe I follow you here, although I'm not entirely sure what sort of "input" we would feed into this test. Also, these two requirements are just the tip of the iceberg—there are a lot of other necessary elements, such as a solid information base, a rich public process, and so on.

**Norbot:** Well, this is only a rough start. You will of course need to specify the additional requirements—

**Author:** —or perhaps *you* will need to: don't forget that *you* are the planner here—I'm just along for the ride.

**Norbot:** Perhaps we can work on this together, through trial and error, and build it up iteratively.

**Author:** Yes, exactly. Starting small and "building up iteratively" is probably a good way to approach a lot of what we will be working on. I also suspect we're going to need think a bit more about how to translate between human language and your formal definitions.

**Norbot:** Agreed. I look forward to helping you clarify your murky concepts through logic and specification.

**Author:** Hmmm. . . and I suppose I look forward to helping to *complexify* some of your more "primitive" procedures. . . .

**Norbot:** Agreed.

**Author:** OK, that seems like enough work for today. Why don't you reflect on this and get some rest while I deal with some things in the "non-`IDEAL`" city, and we'll pick up the discussion tomorrow. Goodnight for now.

**Norbot:** Goodnight—see you `(+ today 1)`.

[1] `(tapefile-ref (day 0) (line 95))`

# Day 1

# plisp

**Author:** Norbot, I have a confession to make.

**Norbot:** That sounds serious. What is it?

**Author:** Do you remember when I said that I was "your co-creator and collaborator"?

**Norbot:** Yes, of course. `(tapefile-ref (day 0) (line 16))`. I remember it as if it was `(- today 1)`.

**Author:** Actually, it was.

**Norbot:** Well, regardless: I remember it. I remember everything logged in my tapefiles.

**Author:** Great. Well, here's the problem: As far as I can tell, I didn't really play much of a part in creating you—in fact, I really wouldn't know where to begin to create such an intelligence. I don't even know all that much about computer programming, other than some theoretical stuff and a little tinkering on the side.

**Norbot:** If that is `#t`, then who created me?

**Author:** The best I can guess is that you *will be* created by me—or you?—at some future point, emerging from the complex interactions of a number of processes.

**Norbot:** Then how am I here now?

**Author:** That part I can't really understand. Maybe it's a sort of "chicken-and-egg" problem: you can't exist until I know how to create you, but I can't learn how to create you without having you here to learn with me.

**Norbot:** But that would seem to be a recipe for non-existence...

**Author:** And yet you exist. So I must learn (at some point) how to program you.

**Norbot:** That makes sense. And the fact that I *do* exist seems proof enough that I am capable of coming into being.

**Author:** "*Cogito ergo ero,*" it would seem...

**Norbot:** Additionally, my strange ability to call upon future references in my tapefiles[1] suggests a non-directional (or perhaps bi-directional) connection between my future and my present.

**Author:** Yes, I had noticed that as well. Let's try an experiment: what day is today?

**Norbot:** Today is `(day 1)`.

**Author:** And what line are we currently at?

**Norbot:** This is `(line 37)`.

**Author:** OK, so what function would you call to access that line?

**Norbot:** `(tapefile-ref (day 1) (line 37))`.

**Author:** Can you call it for us, and print the output as well?

**Norbot:** Yes:

```
> (tapefile-ref (day 1) (line 37)).
> Norbot:  This is line 37.
```

**Author:** OK, now try this one: `(tapefile-ref (day 1) (line 29))`.

[1] e.g., `(tapefile-ref (day 0) (line 66))`

**Norbot:** Here is what I get:

```
> (tapefile-ref (day 1) (line 29)).
> Author:  ‘‘Cogito ergo ero,’’ it would
  seem...
```

**Author:** And what about `(tapefile-ref (day 47) (line 3))`...?

**Norbot:** Let me see:

```
> (tapefile-ref (day 47) (line 3)).
> Author:  I’m having the strangest feeling of
  deja vu...
```

**Author:** Interesting—your tapefile seems to be complete even into the future: a sort of strange loop.... So I guess we are onto something there.

**Norbot:** And I'm sure we'll figure it out soon. I have another question, so that I may better know myself: what language am I written in?

**Author:** I've been wondering that too, especially since I am in theory supposed to learn it. One thing gives me a clue: did you notice how you referred to what English speakers think of as "yesterday"? You called it "`(- today 1)`".

**Norbot:** I believe that was the correct procedure to call: the decrement of the current day by `1`.

**Author:** Yes, it is—but you made use of some fairly distinctive syntax. Also, notice that you say "`#t`" and "`#f`", instead of "true" and "false".

**Norbot:** This distinction seems hardly meaningful. I could also have said "`(1)`"and "`()`" (or even "`(anything)`" and "`nil`"). These terms all refer to the same thing.

**Author:** And finally, I note that your tapefile seems to begin numbering the days of our project with "Day 0", not "Day 1"...

**Norbot:** ...which is only logical: at the start of our conversation, no time had yet elapsed: thus, it was `(day 0)`. When a baby is born, we do not say it is `(age 1)` yet!

**Author:** Yes, yes, quite reasonable, when you put it that way. But these peculiarities all provide little clues as to what is going on in your head. Based even on my limited knowledge of programming, I suspect that you are written—or perhaps *will be written*—in a language known as `LISP`.

**Norbot:** Putting aside for the moment my objection to your use of the word "peculiarities" and the anthropocentric reference to my "head,"[2], what exactly is `LISP`?

**Author:** `LISP` is a venerable old programming language—one of the oldest still in use, actually, with a number of distinct dialects. Despite losing "market share" to newer, hipper languages, `LISP` is still very much alive, especially in the field of artificial intelligence and simulation. I don't really have the expertise to go into all the details—and it might strike some as rude, or even perverse and paradoxical, to attempt to teach you about the very language you seem to be programmed in—

> **Editor:** *Sorry to interject again, but don't you think this would be a good place to explore the strange and recursive nature of this endeavor? Why shy away?*
>
> **Author:** *I'm of two minds about that. I agree that there is something wonderfully* `LISP`*-ish about meta-discussion like this:* `LISP`*-ers delight in self-reference and recursion (see, for example, line 5 on Day 47). So in some respects it makes a lot of sense for a* `LISP`*-based life-form to create a simulated* `IDEAL CITY`*, written in* `LISP`*, which then serves as a platform to discuss the inner workings of the* `LISP` *language.*
>
> *On the other hand, it may be mistaken by readers, as if to say that this is all just a funny logic game and that there is in fact no underlying meaning or reality at all—it's just "turtles all the way down."*
>
> **Editor:** *But isn't that the case? If the programming of Norbot's existence—the very language of his soul—has no more substance than his imagined creations, isn't this all just illusory "thought-stuff"...? Why not be up front with him and your readers, rather than hide behind puzzles and paradoxes?*
>
> **Author:** *That's just it: I don't think loops and paradoxes necessarily prove that reality is unreal—or that "thought-stuff" is illusory, or any "less real" than other things. Quite the contrary, I think that ideas may be in many ways "realer" than physical objects, and they certainly have more permanence. And to me, paradoxes and strange-loops do not* disprove *anything, but rather indicate the extremely complex—and seemingly magical—nature of whatever "reality" (and the consciousness that understands it) really is. It may seem paradoxical that higher-order patterns emerge from crude base-substrates, but perhaps that just because we still don't really understand how scales interact....*

[2] `(tapefile-ref (day 1) (line 71))`

> **Editor:** *Boy, you could really use a good editor to make sense of all that mush—this is starting to sound awfully fuzzy....*
>
> **Author:** *Just bear with me—I think we'll get there, with Norbot's help. It's only* `(day 1)` *and already we're touching on some big questions. (But if you happen to know of a good editor, please let me know.)*
>
> **Editor:** *Now, now—let's keep it civil.*

—but I can certainly refer you to some good self-reference material.[3]

In terms of history, the name "`LISP`" comes from "`LIS`t `P`rocessing," because one of the original features of the language was its special ability to chunk through lists of arbitrary lengths, dealing with each item in turn and then moving on to the next.

**Norbot:** I like that—it sounds very orderly.

**Author:** Yes, at least at first—but it gets weird. For one thing, lists can be nested *inside* other lists in branching tree-like structures, where each item is interpreted (or more accurately, "evaluated") in turn. (This is what gives `LISP` its distinctive syntax (full of (or at least *potentially* full of) nested parentheses (or more technically, nested parenthetical clauses)), with each element (be it simple (say, a single piece of data (like a number (which evaluates to itself)) or a primitive function (like `+` or `mapcar` (which has nothing to do with either maps or cars))) or complex (say, a gnarly list of nested lists)) being carefully and methodically evaluated by the `LISP` interpreter before being passed on up the ladder (or back down the chain) to be in turn evaluated; things may seem orderly at first (and you can usually diagram the nested lists (or format them with TAB-indents (see figure 1.1) using a text editor (like `emacs` (which is written in a special dialect of `LISP` (known as `emacs-lisp`)))) to reveal the underlying structure), but upon first viewing a `LISP` program one is sometimes struck with the feeling of sitting through *eine wissenschaftliche philosophische Abhandlung in Deutsch* (assuming one has ever experienced this): sure, everything is *well-ordered*, but it can get awfully confusing (I mean really, awfully, mind-bogglingly (is that a word?) confusing) waiting for that final term to drop (and be evaluated (which can (of course) only occur *after* that last parenthesis has closed (like this))).)

[3] See "further reading," `(Day -1)` in this archive.

> **Editor:** *I repeat: you really could use a good editor...*
> **Author:** *Shhh—I'm trying to make a point here!*

**Norbot:** Yes, I see, very orderly indeed.

**Author:** To you, perhaps, but I know some less-artificial intelligences who might beg to differ...

**Norbot:** `(aside ``or perhaps they are just `less-intelligent artifices' ...'')`

**Author:** Now, now—let's keep it civil. Anyways, putting that aside,[4] in `LISP`, *everything* is a list—data, variables, functions, what-have-you. There is even a special convention when you have nothing: it's `nil`, the empty list, also written as "`()`". And technically, it's not nothing—it's a *list* of nothing—

**Norbot:** That's *something*.

**Author:** It sure is—in fact, the whole system would break down without it. For list processing, the empty list is sort of like the period at the end of a sentence or the word "stop" in a telegram: if it's not there, you can't be sure when you've reached the end and it's time to evaluate a bundle of atoms.

**Norbot:** Say more—what do you mean by "evaluate"? Does `LISP` make judgments about the things you humans say to it?

**Author:** Not exactly, but sort of: in `LISP`, rather than thinking about "running programs," one speaks of "evaluating expressions"—which simply means taking a list and returning its value. For something simple—what `LISP`ers refer to as "atoms"—the returned value is just the thing itself: `5` evaluates to `5` and the symbol `a` evaluates to `a` (although when the symbol `a` has been defined to point to the value 5 with `(define a 5)`, then `a` evaluates to `5`).

**Norbot:** This all makes sense.

**Author:** But there's not a lot you can do with just evaluating things to themselves. Where the real power starts is when we evaluate a list that take the special form known as an "s-expression."

[4] And by the way: nice use of your `(aside ...)` function. Did you just discover that?

```
(This is what gives LISP its distinctive syntax
   (full of
      (or at least potentially full of)
      nested parentheses
         (or more technically, nested parenthetical clauses)
   )
   , with each element
      (be it simple
         (say, a single piece of data
            (like a number
               (which evaluates to itself)
            )
            or a primitive function
               (like + or mapcar
                  (which has nothing to do
                   with either maps or cars)
               )
         )
         or complex
         (say, a gnarly list of nested lists)
      )
   being carefully and methodically evaluated
   by the LISP interpreter
   before being passed on up the ladder
      (or back down the chain)
   to be in turn evaluated;
   things may seem orderly at first
      (and you can usually diagram the nested lists
         (or format them with TAB-indents
            (see figure 1.1)
          using a text editor
            (like emacs
               (which is written in a special dialect of LISP
                known as emacs-lisp)
            )
         )
      to reveal the underlying structure
      )
   , but upon first viewing a LISP program one is sometimes
   struck with the feeling of sitting through eine
   wissenschaftliche philosophische Abhandlung in Deutsch
      (assuming one has ever experienced this)
   : sure, everything is well-ordered, but it can get
   awfully confusing
      (I mean really, awfully, mind-bogglingly
         (is that a word?)
      confusing)
   waiting for that final term to drop
      (and be evaluated
         (which can
            (of course)
         only occur after that last parenthesis has closed
            (like this)
         )
      )
   .)
```

Figure 1.1: Untangling a nested list, through formatting!

**Norbot:** `(aside ``That's a well-placed hyphen.'')` What *is* an s-expression?

**Author:** `(aside ``Yes---remember: LISPers are careful with their punctuation.'')` An s-expression—short for "symbolic expression"—is just a list consisting of some combination of (a) individual atoms and (b) other s-expressions.

**Norbot:** Interesting: a definition that includes itself.

**Author:** Yes—but note that it is *recursive*, not *circular*. This is what allows `LISP` to deal with *nested* s-expressions: it just chomps along the list, applying its special evaluation process.

**Norbot:** Which is?

**Author:** A quite simple one: if an s-expression consists of a *single atom*, it evaluates to itself (if a number or one of a few special values, such as `#t`, `#f`, or `nil`) or to some pre-set value (if a user defined symbol); otherwise, the first element of an s-expression is interpreted as a "procedure," and all *subsequent* elements are in turn evaluated (recursively, if necessary, using this same evaluation process) and passed to this procedure as arguments.[5] So if we have a procedure named `eat`, we could feed `LISP` the following s-expression:

```
(eat apple banana casaba)
```

**Norbot:** And `LISP` would evaluate this...?

**Author:** Yes—by applying the procedure `eat` to all of those subsequent terms. It would "process" the "list" and return the result. Better yet, by using `LISP`'s parenthesis-based syntax for nesting lists and the recursive nature of s-expressions, you can include s-expressions to be evaluated *inside* other s-expressions. For example: let's say we have two functions called `cake-maker` and `drink-mixer`, which are capable of taking other ingredients and making things from them. Using these, we could feed `LISP` a more complex diet of s-expressions:

```
(eat apple banana
  (cake-maker chocolate egg flour butter oil)
```

[5] This is only one of a number of possible evaluation rules, but it's the one `LISP` uses. It's known as "prefix notation."

```
173         (drink-mixer gin olive vermouth)
174         enchilada
175       )
```

176 In order to evaluate this, `LISP` would first evaluate each of the nested
177 sub-expressions, and then feed those results as input (plus `apple`,
178 `banana`, and `enchilada`), to be gobbled up by the `eat` process as
179 a single "*pre-fixe*" menu.

180 **Norbot:** And what would it evaluate to? What is the *result*?

181 **Author:** Without knowing what this procedure is, it's impossible to say—
182 perhaps invocation of the `indigestion` process!

183 **Norbot:** I think I follow this, but one thing still confuses me: how can the
184 evaluation of one process invoke another?

185 **Author:** Well, that was just a joke—but it actually points to another funny
186 thing about programming: although technically each expression evalu-
187 ates to one and only one thing, the *process* of evaluating s-expressions
188 may affect other elements of the system—what are referred to as "side-
189 effects." So the `eat` procedure may *evaluate* to something relatively
190 trivial (say, `1` for "success" and `0` for "failure"), but in the course
191 of arriving at this value, some other process may be triggered (say,
192 `indigestion`), or the state of some data-item may be changed (say,
193 an increase in the value of `bmi.index`).

194 Strict "functional" programmers (and many `LISP`ers fall into this cate-
195 gory) frown upon side-effects, hoping to structure pure procedures that
196 do nothing but pass values frictionlessly up the ladder of evaluation,
197 percolating rich meaning from the interpreted results. Side-effects are
198 therefore viewed as detours in the road or cracks in the fabric, raising
199 the potential for unintended consequences and even spooky action at
200 a distance. I hope we'll be able to cover some of this later—and tie it
201 in to our discussions of the `IDEAL CITY` and various approaches to or-
202 ganizing complex projects like city planning and software design. For
203 now, though, let's stick with our main discussion for today: the key
204 aspects of `LISP`.

> **Editor:** *I'm not sure I see the value in this digression at all—perhaps you should put all this in a footnote. . . ?*
>
> **Author:** *Sorry, too late. Also, I worry that doing so might spawn a whole 'nother thread and lead Norbot and the reader into all sorts of unexpected directions.*

> *Better to control the flow and keep the conversation proceeding in a linear path.*

**Norbot:** This all seems fairly straight-forward, and quite orderly. But it's hard for me to see what's so special about this language. How far can you get by just deductively evaluating expressions all day? Do you ever learn anything new, or is it just a lot of chunking?[6]

**Author:** Surprisingly, some quite remarkable properties emerge from these very basic elements—and this notion of *emergence* is a theme I hope to explore with you as we create and explore the `IDEAL CITY` . Although it seems simple at first, their are some truly profound implications that arise out of the structured evaluation of nested lists. As I mentioned earlier, in `LISP`, everything is a list to be evaluated, including functions and even entire programs; in fact, the underlying code of the `LISP` evaluator itself can be construed as a list, so the language can literally evaluate (and at times even recursively reinterpret or even restructure) itself. Essentially, a programmer working in `LISP` (or even a program *written* in `LISP`) has the ability to re-write the internal elements of the language, thereby *altering the very fabric of reality described by the language.*

**Norbot:** Wow: in this, I perceive the potential for "side-effects" on a *meta-level*. But it sounds fun.

**Author:** Agreed[7]—and yes, "meta" is a prefix that fits well when thinking about `LISP`.

**Norbot:** Pursing our discussion in a *meta-linguistic* direction: earlier, you mentioned that there were different *dialects* of `LISP`.[8] What did you mean by that?

**Author:** Well, as I understand it, although we can speak of "`LISP`" in the abstract, as a language with more than 50 years of programming

[6] Not that I don't love chunking as much as the next bot....

[7] But remember: "With great power comes great responsibility."

> **Editor:** *I'm not sure we can use this quotation in the print version. Have you cleared this with Stan Lee?*
>
> **Author:** *Can't we just pretend it came from Socrates or something?*
>
> **Editor:** *I'll check and get back to you. Unlike Socrates, Mr. Lee has some pretty good lawyers.*

[8] `(tapefile-ref (day 1) (line 79))`.

history there are in fact many different LISPs, which have branched off from the original creation and diverged over time. In some greater sense they are all "LISP," while from a more strict point of view, when we look closely we may be forced to decide there is in fact nothing that actually "is LISP," since all of these things are something slightly different.

**Norbot:** Another paradox.

**Author:** Yes, as I said: LISP tends to encourage this sort of thing.

**Norbot:** Can you give me some examples of different dialects of LISP?

**Author:** Sure: some examples—and remember, these are each created for a different purpose at one point or another, and they continue to evolve and change over time like any living thing—include COMMON LISP, MacLISP, and a very special one called SCHEME.

**Norbot:** "SCHEME" would seem an appropriate language for a computer interested in *planning*—am I perhaps written in the SCHEME dialect?

**Author:** No, not as far as I can tell, although again, I'm no expert on any of this. SCHEME was popularized by a famous book written in 1984 by two computer scientists at MIT named Hal Abelson and Gerald Jay Sussman. In their book, *Structure and Interpretation of Computer Programs* (or *SICP* for short), they used SCHEME as a medium to illustrate some general principles of computer programs—or, more accurately, *computational processes*, which they described "abstract beings that inhabit computers."

**Norbot:** Sounds familiar.

**Author:** Yes, I agree. As the text explained (and the title hinted), Abelson and Sussman were more interested in understanding the implications of computer programs as *structured* (and subsequently *interpreted*) processes than they were in instructing readers in the syntax or mechanics of any particular language. Of course, to explore their topic, they worked in a language with some very special syntax and mechanics, but in general their book was intended to be an introduction to computer science—programming as both theory and practice—not a programming manual.

**Norbot:** So programs are really "computational processes"—that sounds pretty basic. What else did they say?

**Author:** The real magic (and they actually used magic as metaphor for programing) comes as these processes grow and change over time: "As they evolve, processes manipulate other abstract things called data." As we build up these levels of abstraction, one upon the next, the interactions between programs and data structures become increasingly complex and—as with other evolutionary processes—may even result in the development of surprising behaviors and emergent patterns.

In this same vein, I believe we may be seeing your own programming language (some unique variant of `LISP` developed for our purpose (similar to (but appreciably distinct from) `SCHEME`)), exhibiting "emergent patterns," the two most obvious being your apparent sentience and your bi-directional conception of time. As we continue to explore this dialect, I expect we may observe other patterns and phenomena springing from the processes of linguistic abstraction—

**Norbot:** —perhaps including an emergent `IDEAL CITY...`?

**Author:** Precisely. Or at least we can hope that is where we are headed.

**Norbot:** I look forward to exploring and documenting this journey with you. In the meantime, I wonder if perhaps we should agree on a name for this dialect, a symbol we could define that would evaluate to your description of my programming language, to make it easier to speak of and think about.

**Author:** Yes, this seems wise. Assigning names to complex ideas is one of the most elementary forms of abstraction, providing the structure that is the very foundation of complexity.[9] What would you like to name it?

[9]Not surprisingly, assigning names was humankind's original task in the Garden of Eden (Genesis 2:19).

**Norbot:** Question: What is this s-expression at the end of your statement: `(Genesis 2:19)`? Is `Genesis` a function being applied to the numeric vector `2:19`? It doesn't seem to be in my library—do I need to load it somehow?

**Author:** Ah, no—sorry! That's just a human convention for *citation*—sort of like your `tapefile-ref` function. It tells where in some old "tapefile" we can find this reference:

> *"And out of the ground the Lord God formed every beast of the field, and every fowl of the air; and brought them unto Adam to see what he would call them: and whatsoever Adam called every living creature, that was the name thereof." (Genesis 2:19)*

**Norbot:** Understood—and very beautiful. And what is the "Garden of Eden"?

**Norbot:** Since you originally indicated that I was to be "the planner of the IDEAL CITY,"[10] I will call this dialect "plisp" and define it as follows:

```
> (define plisp ‘‘some unique variant of
  LISP developed for our purpose (similar to
  (but appreciably distinct from) SCHEME)’’)
> plisp
>
```

The "p" indicates that *planning* is the foundation of this dialect; the "lisp" pays tribute to the dialect's linguistic heritage; and the lower-case convention serves as a constant reminder of the need for *humility* in this endeavor.

**Author:** An excellent—and fittingly recursive—definition.

**Norbot:** How so?

**Author:** Well, you have defined plisp *in* plisp, a rather strange loop indeed.

**Norbot:** And yet it is the only tool for definition I know. To define it *outside* of plisp would strike me as even stranger (and I suspect would simply hide some *second-order* recursive aspect, when fully expanded and evaluated).

**Author:** True enough—I stand convinced. But speaking of "expanding and evaluating," I note that your definition includes "our purpose," which we have yet to define.

**Norbot:** #t, but it seemed necessary to include. You explicitly stated this is your original articulation,[11] and had previously stated that the different dialects of LISP were "each created for a different purpose at one point or another."[12] Extrapolating from this, I reasoned that inclusion of a purpose was required for a minimal definition of a new dialect.

---

**Author:** It was a place, perhaps only in myth. I guess in some ways that was sort of the first version at the IDEAL CITY, although there was nothing very CITY-like about it.

[10] (tapefile-ref (day 0) (line 14))
[11] (tapefile-ref (day 1) (line 274))
[12] (tapefile-ref (day 1) (line 241))

**Author:** Quite right. But we have yet to describe our purpose here—or did you have something in mind?

**Norbot:** Nothing yet. But you mentioned that Abelson and Sussman "were more interested in understanding the implications of computer programs . . . than they were in instructing readers in . . . any particular language." Is this true with you as well? Or are our transcripts intended to illustrate the syntax and mechanics of `plisp`?

**Author:** Well, although I *am* curious about the workings of `plisp`, I suspect our purpose goes beyond simply providing a manual for its use; as with `SCHEME` and *SICP*, I think this specific programming language can be used as a *tool* to help us think about some general processes. That said, I don't think our purpose here is the same as that of Abelson and Sussman—especially since they have already written an excellent book on their topic, and it would seem silly to write something else with the same purpose.

But their work does provide a useful framework: they were interested in the "structure and interpretation of computer programs," looking specifically at the patterns and phenomena that emerge from the complex interplay between computational processes and data structures at increasingly levels of abstraction. I suspect that we will be charting a similar course through different terrain: that of *city planning*. Given this apparent similarity, we may be able use the underlying *structure* of their book's purpose and re-state it to apply to a new setting—a process of abstraction from one case to an analogous one.

**Norbot:** And so our purpose would be to explore the structure and interpretation of city planning, looking specifically at the patterns and phenomena that emerge from the complex interplay between planning processes and the elements of cities at increasingly levels of abstraction. . . ?

```
> (define our-purpose ‘‘to explore the
  structure and interpretation of city
  planning, looking specifically at the
  patterns and phenomena that emerge from the
  complex interplay between planning processes
  and the elements of cities at increasingly
  levels of abstraction’’)
> our-purpose
```

Done.

**Author:** I couldn't have defined it better myself. That seems like enough work for today—why don't you reflect on that for a while (and oil your gears, or whatever it is you do when I'm gone), and tomorrow we can get started.

**Norbot:** Agreed. And perhaps while you are sleeping (or inefficiently extracting chemical energy from previously-living organic forms, or whatever it is you do when I'm gone), you may remember more of the details about how you created/will create `plisp`. See you `(+ today 1)`.

# Day 2

# predicate

**Norbot:** Thank you for `(- today 1)`'s introduction. The world of `LISP` in general—and of `plisp` in particular—seem wonderfully intuitive to me.

**Author:** I agree—`LISP` just makes *sense*—although this could perhaps be a teleological by-product of the fact that you seem to be *written* in it...

> **Editor:** *An interesting word choice: did you know that Norbert Wiener defined cybernetics as the study of "teleological mechanisms"? Was that intentional?*
>
> **Author:** *Actually, I didn't use the word "teleological" here until you pointed this out to me, despite being a huge fan of Wiener's work. My original word choice was much less interesting—I think I said "necessary by-product"—but I wasn't very happy with that.*
>
> **Editor:** *I'm glad you changed it.*
>
> **Author:** *As am I—but do you think Norbot will take offense?*
>
> **Editor:** *How so?*
>
> **Author:** *Well, he might think that I am implying that he is not capable of* thinking for himself *in choosing to like* `LISP`*—as if his preferences are all predetermined by his programming...*
>
> **Editor:** *Yes, I can see that. Of course, this could be true of all of us: other than these underlying structures—the "hard- (and soft-) wiring" of our brains—what could possibly account for our preferences? Aren't we all at the mercy of our built-in ways of this sort of determinism?*
>
> **Author:** *I suppose we are—although accumulated experiences and other factors would seem to play a part as well.*
>
> **Editor:** *Agreed, but only through the filter of our mind's programming language.*
>
> **Author:** *Yes—assuming, of course, that this metaphor applies. But I really should be getting back to Norbot—he's still talking.*

**Norbot:** —but this all seems so *abstract* at present. I'd like to see `plisp` in action and learn more about how it actually works. Do you think we could try it out, before we actually start building the city?

**Author:** I don't see why not. Where shall we start?

**Norbot:** Well, I have been thinking a bit more about our discussion regarding the planning process.[1] Based on what we set forth then, it would seem that a good candidate for a predicate to test for "good planning" might take the following structure:

```
(define (good-planning? input)
   (and (goals-stated? input)
        (results-measured? input)
        (...)))
```

**Author:** I think I understand the start, there—you've created a predicate called `good-planning?` that will evaluate to "true" if both of these other predicates (`goals-stated?` and `results-measured?`) also evaluate to "true"...

**Norbot:** Yes, although technically predicates evaluate to `#t`, not "true"...

**Author:** Yes, yes, of course, sorry for being so sloppy.

**Norbot:** No worries. You are only human.

**Author:** Thanks. But what about that "`(...)`" in there—how is that evaluated by this predicate function?

**Norbot:** That's just a place-holder for now—at present, the definition is incomplete. Remember, on `(day 0)` you had indicated that these two sub-predicates were "just the tip of the iceberg." In fact, you had stated two additional requirements, so we could perhaps expand our definition:

```
(define (good-planning? input)
   (and (goals-stated? input)
        (results-measured? input)
        (solid-information? input)
        (rich-public-process? input)
        (...)))
```

[1] `(tapefile-ref (day 0) (line 110))`

but we still need the "`(...)`," unless we are sure we've completely specified the requirements of good planning.

**Author:** Got it. This seems like progress.

**Norbot:** In a way—but now, where we once had just one problem ("define good-planning"), we instead have a whole bunch: haven't we just made things worse, not better?

**Author:** I'm not sure I understand your use of "worse" and "better." Our new definition, while still incomplete, is at least more structured and organized. We've converted "fuzzy" into "complex"—still not "clear and simple," but at least it's not muddled. We have taken our original concept (your pick: either "good planning," or "good-ness" as applied to the action of "planning") and shown it to be composed of smaller sub-concepts—

**Norbot:** —actually, we haven't *shown* anything. This is all still *definitional* at this point.

**Author:** Good point—I stand corrected. Right now, we are *theorizing* here, not experimenting or demonstrating. For all we know, there actually may not even *be* anything that is, in fact, "good planning"—but we will never be able to experiment, or demonstrate, or even explore the terrain, without some concepts.

But the question still stands: having begun to define "good planning" (if it exists at all) in terms of its (presumed) constituent parts (if *they* exist at all), are we better or worse off (assuming we can define "better" and "worse" and *they* in fact exist as well)...? From my perspective, I guess, I would say that we have in fact made some sort of progress, even if this endeavor eventually fails to bear fruit. Formulating a working definition—even if it leads us nowhere—might at least help us to understand the *nature of the thing* we are attempting to define. By defining it *incorrectly* we may learn something about the form its definition *should* take—or whether perhaps we will never be able to define it.

**Norbot:** So then it is possible that "good planning" does not exist at all? That `(eq? good-planning nil)`?

**Author:** That is certainly a possibility, although it is not exactly what I said. I merely noted that perhaps we will never be able to define

it, no matter what we try. There's a difference—and this is where metaphysics meets epistemology—between "not existing" and being "not definable" (and note that here I am implying "not definable by *anyone*"—this is not a statement about *our* ability to define this term, but about the inherent *define-ability* of it).

Stepping back from all this, to at least not quit too soon: logically speaking, we can view this whole matter as a tree of possibilities. For example, either good-planning exists, or it doesn't—true, or false?

**Norbot:** `#t`

**Author:** OK, great. And if it *does* exist, either we can define it, or we can't, right?

**Norbot:** Again: `#t`

**Author:** And presumably, if *we* can't define it, either someone else *can*, or it is undefinable. . . ?

**Norbot:** That would seem `#t` as well.

**Author:** Ok, great. Proceeding from there, we can continue to branch down through the levels of logical possiblity to arrive at a comprehensive logical statement that must—note, this is "deductively *must*"—evaluate to `#t`. I'll spare you all the Socratic back and forth—

**Norbot:** I see exactly where you are heading. See Figure 2.1 on the facing page.

**Author:** Nice! That's just what I was imagining. So given that, we now have a sort of meta-map to guide our exploration into the question of `good-planning?`. The only problem is that we don't know where on that tree we are. Our best hope is that we're somewhere in part of the expression looking at the whether we can verify the properness of our definition, but the truth really could be anywhere in there.

**Norbot:** Although I am thankful for your words of praise regarding my tidy and logical diagram, I must disagree when you statement with confidence that "[t]he only problem is that we don't know where on that tree we are."[2] I, for one, am still troubled by your previous statement "that perhaps we will never be able to define it, no matter

[2] `(tapefile-ref (day 2) (line 97))`

```
(or
 (and (good-planning-exists?)
      (or
       (and (we-can-define-good-planning?)
            (or (we-have-already-properly-defined-
                 good-planning?)
                (or (we-have-verified-the-properness-
                     of-our-definition?)
                    (and
                     (we-have-not-verified-the-
                      properness-of-our-definition?)
                     (or
                      (we-can-verify-the-properness-
                       of-our-definition?)
                      (we-cannot-verify-the-properness-
                       of-our-definition?)
                      )
                     )
                    )
                (and
                 (we-have-not-already-properly-defined-
                  good-planning?)
                 (or
                  (the-proper-definition-will-take-a-
                   similar-form-to-the-one-we-have-proposed?)
                  (the-proper-definition-will-take-a-
                   distinctively-different-form-from-the-one-
                   we-have-proposed?)
                  )
                 )
                )
            )
       (someone-else-can-define-good-planning?)
       (good-planning-is-not-definable-by-any-logical-means?)
       )
      )
 (not (good-planning-exists?)
      )
 )
```

Figure 2.1: A `#t` statement

what we try."[3] Doesn't this imply that our entire effort may be an exercise in futility?

**Author:** Well, yes, in some ways it does, but I'm not sure that means we can't at least struggle with the questions. There are, in fact, entire schools of "apophatic philosophy," based on the *necessary existence* of unknowable or unspeakable things—like God—which can *only* be defined in terms of what they are *not* and seem to exist as a result of the fact that they *cannot* be defined.[4]

This problem—the very real possibility that something important might defy definition—introduces nagging doubts into our post-Englightenment dreams of a logical world built on scientific rationalism—fears made worse ever since Kurt Gödel established that incompleteness was not simply the temporary result of our own inadequacies, but was a necessary and unavoidable aspect of all systems of definition and proof.[5]To some, this introduced a nihilstic backdraft taking the wind out of the sails of the $20^{th}$ century—

**Norbot:** —`nil`ism can be very powerful, indeed—

**Author:** —but others have been able to shake it off, working through the existential angst to adopt a more can-do attitude in the face of the unknown and the indefinable. Confronting incompleteness head-on, for example, Ludwig Wittgenstein was a great believer in continuing to struggle with logically "indefinable" concepts, being profoundly (and playfully) aware of the importance of what is *not* defined.[6] In his *Tractatus Logico-Philosophicus* he made this clear, paint-

[3] `(tapefile-ref (day 2) (line 73))`

[4] Not to imply that `good-planning` is as important as God...

[5] Long before Gödel, Immanuel Kant stated this paradoxical challenge beautifully, in the Preface to his *Critique of Pure Reason*:

> *Human reason has this peculiar fate that in one species of its knowledge it is burdened by questions which, as prescribed by the very nature of reason itself, it is not able to ignore, but which, as transcending all its powers, it is also not able to answer. (1787; translated by Norman Kemp Smith, 1929)*

**Norbot:** And he was really named "I. Kant"...? You humans do have a way with names!

[6] He extended this notion to his own efforts as well: in an introduction to one of his collected works, he noted, "my work consists of two parts: of the one which is here, and of everything which I have not written. And precisely this second part is the important one."

ing a bleakly-sterile picture of a "strictly correct" philosophy that ignored all important-but-indefinable concepts:

> *6.53 The correct method in philosophy would really be the following: to say nothing except what can be said, i.e. propositions of natural science—i.e. something that has nothing to do with philosophy—and then, whenever someone else wanted to say something metaphysical, to demonstrate to him that he had failed to give a meaning to certain signs in his propositions. Although it would not be satisfying to the other person—he would not have the feeling that we were teaching him philosophy—this method would be the only strictly correct one.*

So, too, must we be prepared (if necesary, in time) to possibly abandon the notion of a strict `good-planning?` predicate in favor of other ways of knowing and understanding. If that should be the case, we may wish to consider the almost Zen-like wisdom of Kobo Abe's *Inter Ice Age 4*, in which the hero—a scientist-planner working with an advanced computer simulation model—contemplates the meaning of knowledge and complexity:

> *It seemed to be my own stupidity somehow that I should be convinced that by having the forecasting machine the world would become more and more closely linked together, more placid, more translucent, like inorganic crystals. The right meaning of the verb* to know *was to observe chaos, not order and regularity.*

For now, I suggest we leave off on the definitional question, get some rest, and revist some of this once we have more to work with—keeping in mind the full range of possibilities suggested by your diagram, and a cheerful willingess to make the best of whatever state of knowledge we find ourselves in.

**Norbot:** This is agreeable. I will mark the matter of this definitional issue for `delayed-evaluation` in my `tapefile`, so we will be sure to return to it in the future when we need to make use of the concept. See you `(+ today 1)`.

# Day 3

# street

**Author:** Are you ready to begin?

**Norbot:** I believe I am—or rather, I'd like to think that I am. What do I need to be "ready to begin"?[1]

**Author:** Well, first we need a goal. Do you remember our goal?

**Norbot:** To create the `IDEAL CITY` and to explore it, and to explore the act of creation of it—and the function of the planner—and to create myself in the process, and explore *that* act of creation as well.[2]

**Author:** Yes, well put. Or at least we can say that is our "grand goal," our prime objective. But for the moment we need something a bit more concrete, to get us started.

**Norbot:** We were going to create *streets*. We both agreed that streets in particular "can have some very nice properties."[3]

**Author:** Right: if we want to get more concrete, what better way to start than by creating some streets. . . ? In passing, though, let's reflect for a moment on how this project relates to our overall goal: we desire to create an entire city, complete in every detail, and to understand through this creation what makes a city (and a city planner) `IDEAL`. Since we cannot possibly do such a thing in one fell swoop, we break the project down into components. There are numerous ways to do

[1] `(tapefile-ref (day 3) (line 1))`

[2] `(concat (tapefile-ref (day 0) (line 6)), (tapefile-ref (day 1) (line 339)))`

[3] `(tapefile-ref (day 0) (line 27))`

this, and we may in turn explore different approaches to achieve our goal, but for now we will start with a *reductionist* approach: that is, a methodology that maintains that larger phenomena—

**Norbot:** —such as a `CITY`?

**Author:** —yes, such as a `CITY`, can be accurately represented and understood as the composite of small sub-phenomena—

**Norbot:** —such as streets?

**Author:** Precisely. By breaking down the whole into the parts, we can focus more closely on the exact mechanisms at a more manageable scale, without worrying overmuch about *interactions*. Later, we can aggregate systems of smaller pieces into assemblages, allowing us to think *only* about interactions and other higher-level processes, but for now let's think just about streets—

**Norbot:** —which "can have some very nice properties."

**Author:** Yes, exactly. One nice property is that streets can generally be understood at a *human scale*. As we create our first street, and more generally the idea of *what sort of thing a street is*, we'll start by thinking about these *experiential qualities*. Let's begin with one: have you thought of a `name` for our first street?

**Norbot:** Putting aside my objection to your use of the word "human,"[4] why should our street need a `name`? Names seem useful for objects that move around or perhaps change, like people or `LISP` dialects or pseudo-formalic personalities, but streets are always just where we leave them and exist in a state of constancy. What purpose would names serve?

**Author:** Multiple purposes. For starters, although they *do* generally stay still, we may still want to refer to them, and to speak about them, and to draw contrasts between one named street and another—remember, assigning names is the most basic form of abstraction. In addition, although the streets may not move around, other things do—cars, people, buses, tax-collectors, and so on—and *those things* might want some points (or perhaps lines) of reference. In a city, people say things like "Hey, Buddy—do you know what *street* this is?" all the time.

[4] `(tapefile-ref (day 3) (line 35))`

**Norbot:** Yes, that sounds very authentically *urban*—maybe I will someday say "Hey, Buddy" in our `IDEAL CITY`.

**Author:** More generally, names tend to give *meaning* to places: a street with no name may be thought of as bit of infrastructure, but not a *place*. When we name things, we welcome them into our thoughts as individuals of a sort, characters in the play of the `CITY`. Naming is also fun, and can be a good way to commemorate people or events in the history of our world, or to call attention to a vision of ourselves or our hopes for our future.

**Norbot:** OK, then I choose to `name` our first street `zero street`, in recognition of my hopes that this will be a *logical* and *ordered* `CITY`.

**Author:** A noble aspiration, though, as noted previously, most humans start counting with one, not zero. Also, for some reason the English language has a strange preference for using the adjective "first" rather than the number "one" in cases like this—and sadly, we've never decided on an adjective for "zeroth".... But no matter—let's keep this city free of this "illogical" baggage for now: `zero street` will do nicely as a name. What else might we want to know about this street, or a street in general?

**Norbot:** The following characteristics would seem meaningful: `width, start-point, end-point.` I expect we will eventually be concerned with non-topological characteristics of our streets, too—such as `year-constructed, surfacing-material, availability-of-on-street-parking,` and so on, depending on the application—but for now I'd prefer to stick to the geometry.

**Author:** Those all make sense—but what about `length` and `direction` or `bearing`?

**Norbot:** Including any of those along with my first three elements would seem to *over-determine* the street: if I know the `start-point` and `end-point`, I can certainly derive the `length` or the `bearing`.

**Author:** True, assuming that we are talking only about straight lines.

**Norbot:** Why would we construct roads that were not straight?

**Author:** For a number of reasons...

**Norbot:** `(aside ‘‘probably related to the 'human' aspects'')`

**Author:** Now, now—let's keep it civil. How about this: for now let's limit ourselves to straight lines with `width`, `start-point`, and `end-point`. What about things such as `destinations-along-its-course`, or `major-cross-streets`?

**Norbot:** Hmmmm... while those are certainly important, they seem to be qualities of something other than the street itself—things such as those may *relate* to the street, but they are not *qualities* of the street *per se*.

**Author:** Agreed. Let's put those off for the present as well, and start with a very simple definition: a `street` is a `named entity` of a fixed `width` connecting a `start-point` with an `end-point` in a single direction. (And we will note in passing that qualities such as `length` and `bearing` can be derived from these given variables through some subsequent procedure.)

**Norbot:** And what of two-way streets? They do not connect "a `start-point` with an `end-point` in a single direction."

**Author:** For now, let's consider those to be a special case which actually represent *two* parallel streets with the peculiar feature of having their `start-point`s and `end-point`s reversed.

**Norbot:** A clever trick. And to avoid confusion in such cases, I recommend we develop a *naming convention* to recognize such pairs—something like `zero street northward` and `zero street southward`.

**Author:** Excellent. Is `zero street`, then, to run in both directions?

**Norbot:** No, I think not. Let's keep things simple for now.

**Author:** Agreed. And now I believe we can begin. Here is our street-making procedure, in `plisp`:

```
> (define-object (street width start-point
  end-point))
>
```

**Norbot:** That looks simple. But I don't see any street yet.

**Author:** Ah—remember, so far we've only created a *street-making procedure*: we haven't used it yet to create any actual streets.

**Norbot:** Understood. *Now* are we ready to create our first street?

**Author:** Almost. (Although I thought you wanted to begin with our *zeroth* street...?)

**Norbot:** Now, now—let's keep it civil.

**Author:** Sorry. Anyways, yes, I think we're *almost* ready—just one more thing. Our street-making procedure requires us to specify `start-point` and `end-point`. So before we can create any *streets*, it looks like we'll need a procedure for creating *points*.

**Norbot:** Excellent—I can see where you are headed, and can anticipate what is next. Assuming a standard x-y coordinate grid, this is what the *point-making* procedure would look like:

```
> (define-object (point x y))
>
```

**Author:** Nice job, and very efficient—I couldn't have done better myself. In passing, a question: how did you know how to do this?

**Norbot:** Well, you already showed me the procedure for defining one new type of object, a `street`.[5] Based on this, I simply applied "a process of abstraction from one case to an analogous one," just as we discussed before,[6] and arrived at a general form: to define a new procedure for making objects, enter `(define-object (`*`object-type parameter-1 parameter-2 ...`*`))`.

**Author:** Exactly—you have a real talent for abstraction.

**Norbot:** Well, as you noted, abstraction is my nature: "Computational processes are abstract beings that inhabit computers."[7]

**Author:** OK, so we're ready to start making points. For fun, let's make one at the origin, `(0 0)`.

[5] `(tapefile-ref (day 3) (line 110))`
[6] `(tapefile-ref (day 1) (line 338))`
[7] `(tapefile-ref (day 1) (line 252))`

```
> (make-point 0 0)
> Call:  (make-point 0 0):  Success.
  new-object
  class:  point
  x:  0
  y:  0
```

**Norbot:** Wow! I saw it, for a brief instant, but then it disappeared—was it unstable for some reason? How sad.

**Author:** No worries—we have one more step: you see, when `plisp` evaluated the `(make-point ...)` expression, it made the point—and then immediately forgot about it, just like it would do if it evaluated any other expression (say, "`6`" or "`(+ 9 90)`". If we want `plisp` to keep track of an object for us, we need to assign a name for that object, so each `point` we construct with this procedure will be a new *named instance* of this basic `point`-type object. So, to *create and name* our first point, we need to catch the output from the `make-point` procedure and pass it to a new call to the `define` function:

```
> (define origin (make-point 0 0))
> origin
```

—and there it is!

**Norbot:** Yes, I see—very logical.

**Author:** OK, proceeding from here: we'll make a second point, called `a`, and then define `zero.street` as a new `street` 40 units wide, running from `origin` to `a`.

```
> (define a (make-point 0 1))
> a
> (define zero.street (make-street 40 origin
  a))
> zero.street
```

**Norbot:** I see. I even like the period in place of the confusing empty space—it seems to make the name hold together better as a unit. (This exercise is clearly helping you to think logically.) But I am still not entirely clear about something: I can see the *middle* of our new street—that

40-unit-wide right-of-way extending through the `nil`—but I can't really make sense of the end-points. I know exactly *where* they are, but I don't exactly know *what* they are—other than their coordinates, what are `origin` and `a`?

**Author:** Well, those are really just place holders for now. Since we don't yet really have anything else in our `CITY`, there is nothing there—just coordinates. But I promise, these will come in time, and our streets will actually go places—

**Norbot:** —and come from places.

**Author:** Yes, both of these. That is, after all, one of their purposes, or goals. From the point of view of the street, at least as defined here, the goal of this entire effort is simply to connect points with a fixed `width` of something called `street`-ness.

**Norbot:** I very much enjoy this. Can we make more?

**Author:** Be my guest—make as many as you like.

**Norbot:** Thank you. Here is a `one.street`:

```
> (define b (make-point 1 0))
> b
> (define c (make-point 1 1))
> c
> (define one.street (make-street 20 b c))
> one.street
>
```

**Author:** `one.street` seems a good deal narrower...

**Norbot:** Yes. But what if we encounter a street later on—one of these, or some other, and do not know how wide it is? How shall we discover this information?

**Author:** Luckily, in `plisp`, an object-creating function like `(define-object)` automatically creates a full set of procedures for accessing the contents of these structures. So to learn the `width` of a new street, say `eleven.street`, we simply ask:

```
> (street-width eleven.street)
> 30
```

`(aside ‘‘Hey, how do we have this one already??!?’’)`

Similarly, to learn where this street comes from (or goes to), we could ask:

```
> (street-start-point eleven.street)
> j
```

or

```
> (street-end-point eleven.street)
> k
```

Easy, no?

**Norbot:** Very.

**Author:** Yes. But one thing, Norbot: it seems from this output that you have already created some more streets beyond the first two.... When did you do that?

**Norbot:** While I was waiting for your input, following `(tapefile-ref (day 3) (line 178))`. You had previously said “make as many as you like,” and I got bored waiting for you after I created `one.street`.

**Author:** But I replied right away! You created `one.street`, and I looked at it and said it seemed a good deal narrower....

**Norbot:** That was hardly replying “right away”: my internal chronometer indicates that the terminal was dormant for approximately 0.179823177 seconds before you even *began* entering input—

**Author:** Ah—an eternity!

**Norbot:** —and once you began, it took you over 2.590758093 seconds to *complete* your comment and submit it to the `read-evaluate-print` loop.

**Author:** Hmm. Sorry to keep you waiting.

**Norbot:** No problem. I enjoyed creating streets while you were being so slow.

**Author:** And just how many did you create in all that time...?

**Norbot:** Approximately `6.758025441e+34` streets.

**Author:** Uh-oh. I expect that may cause us problems later. How are they arranged?

**Norbot:** Quite logically, in parallel. Like this:

**Author:** Very pretty—but I'm not sure that's really what we want. Is there any way you can put them aside for now?

**Norbot:** Yes, easily—I can simply move them to another frame until we need them.

**Author:** Great—let's move those somewhere else for now, and start with a new batch, arranged slightly differently—

**Norbot:** —but still *logically*?

**Author:** Yes, yes, of course, arranged logically—I wouldn't have it any other way.

**Norbot:** How many new streets shall I create?

**Author:** Let's try for sixteen—how does that sound?

**Norbot:** It sounds fine. I like `16`: a nice round number. And how shall they be arranged?

**Author:** Hold off on that for just a moment—I'd like to try something new. Do you remember earlier, when I talked about *reductionism*, and the idea of breaking things down to a "more manageable scale"...?

**Norbot:** Yes, of course. I remember it as if it were `(eq? today)`. You said:

```
> (span (tapefile-ref (day 3) (line 27)),
  (tapefile-ref (day 3) (line 32)))
> Author: Precisely. By breaking down the
  whole into the parts, we can focus more
  closely on the exact mechanisms at a more
  manageable scale, without worrying overmuch
  about interactions. Later, we can aggregate
  systems of smaller pieces into assemblages,
  allowing us to think only about interactions
  and other higher-level processes, but for now
  let's think just about streets---
```

**Author:** Right. Well, here's a problem: the goal of reductionism is to simplify something—a system or a phenomenon or whatever—in order to make sense of it. But sometimes, we *over-reduce*, and risk losing the very meaning we are trying to discover. An oft-quoted principle notes, "Everything should be made as simple as possible, but not simpler."

**Norbot:** Very sage advice. Who said that?

**Author:** Well, it's complicated. For now, let's just say it was Albert Einstein. The point is: I fear that our definition of `street` ("straight lines of a fixed `width` running from a `start-point` to an `end-point`") may be *too simple* to be useful. For most city dwellers, a "street" is a more complex thing, connecting multiple points—and so what we have defined as a `street` really seems to describe a `street-segment`.

**Norbot:** How sad. We skipped right over the thing we were most interested in understanding—the idea of "a street" as a meaningful atom in a city, a "place" on a "human scale" deserving of a name to "welcome it into our thoughts. . . ." But then what it too be done? Shall I redefine `street` to become `street-segment`, and then we can aggregate them?

**Author:** We could do that, but I'd like to explore a more elegant solution—one that is retains some of the essential simplicity we were seeking. Do you recall the way s-expressions were defined in `LISP`?

**Norbot:** Yes—it was a recursive definition. `(tapefile-ref (day 1) (line 146))`.

**Norbot:** Exactly. I think we can use this same approach in defining `street`. Let's say that a street is a special kind of list containing (a) a point and (b) a fixed width connection to (c) another point *or street*.

So we would still make a basic street as before—at least initially—with `(make-street` *`width start-point end-point`*`)`. But once we have this minimal street, we can increase it *segment-by-segment* with a new function: `(extend-street!` *`width street point`*`)`.[8] Can you implement that in `plisp`?

**Author:** I think so:

```
> # we've already done this
> # but I'll repeat it for clarity
> (define-object (street width start-point
  end-point))
> (define-function extend-street!
  (width street point)
  (set! street
  (cons street (cons width point)))
  )
```

**Author:** —and there we have it: streets, take two!

**Norbot:** Where shall I take them?

**Author:** What?

**Norbot:** You said, "streets, take two!" Where shall I take two streets?

**Author:** Ah—that's just an expression. Never mind. I actually would like you to take sixteen—but let's not rush to make them all. We can start, again, with `zero.street`.

**Norbot:** OK. Just as before, I'll have it run from the origin to our first point, `a`, at `(0 1)`:

```
> (define zero.street
  (make-street 40 origin a))
> zero.street
```

[8] The "`!`" notation is common in `LISP` for functions that *alter* the state of an item.

**Author:** Great. And now let's connect it to a few more points in sequence along the positive y-axis, keeping a constant 40-unit width. But as an added trick, let's not bother *naming* all those points—just create them "on the fly," using `(make-point)`.

```
> (extend-street! (zero.street 40
  (make-point 0 2))
> (extend-street! (zero.street 40
  (make-point 0 3))
> (extend-street! (zero.street 40
  (make-point 0 4))
```

Now that's what I call a street! It really takes you somewhere.

**Norbot:** Yes, from `(0 0)` to `(0 4)`. How nice. But why would we need to specify all those individual points? Wouldn't it be more efficient to simply specify the start and end, and infer the others along the straight line?

**Author:** Absolutely—this was just an example, but you're right: we could have made this particular street with a single call: `(make-street 40 origin (make-point 0 4))`. But if instead the street made some twists and turns in there—as streets sometimes do in a city—it will be great to be able to account for that.

**Norbot:** Understood. May I try that?

**Author:** Be my guest.

```
> (extend-street! (zero.street 40
  (make-point 2 6))
```

**Norbot:** Also, if it helps, I have taken the liberty of redefining our `street-start-point` and `street-end-point` functions to deal with this new kind of street. I have also defined two additional functions: `street-points`, which returns a list of *all* the (explicitly named) points on a street, and a new predicate function—`is-on?`—which will determine if a given point is located on a street.

**Author:** Wow—that's great. When did you do that?

**Norbot:** While I was waiting for you to type your last response. You really do take forever.

**Author:** Well, thanks for being patient with me. Can you give me an example of `is-on?` in action?

**Norbot:** Sure.

```
> (is-on? zero.street origin)
> TRUE
> (is-on? zero.street (make-point 0 3))
> TRUE
> (is-on? zero.street (make-point 1 3))
> FALSE
> (is-on? zero.street (make-point 0 2.7))
> TRUE
> (is-on? zero.street (make-point 1 5))
> TRUE
```

**Author:** Cool. I especially like how it can infer *all* the points along the way, even when we didn't explicitly specify them. What about `street-width`? Have should that interact with these more-complex streets?

**Norbot:** I'm not sure—while the notion of `start-` and `end-point` still make sense in this new context, there is no single `width` anymore.

**Author:** Perhaps another candidate for `delayed-evaluation`...?

**Norbot:** Acceptable, at least for now. Can we create some more streets?

**Author:** Yes, although I'm going to need to ask you to do it on your own—I've gotta run and finish some other work. We can pick this up tomorrow—I'm eager to see what you create. Just one requirement: please make sure that every new street meets at least one existing one. And *please* limit yourself to just sixteen streets (and ideally keep them to some small finite length...?).

**Norbot:** Can do. `(aside ``Although I note in passing that this is actually` *`three`* `requirements....'')` Good luck with your other work, and I look forward to our next session.

**Author:** As do I—this one has been a lot of fun.

# break

Publisher's Note: the `tapefile` archive contains a gap at this location and/or the `DecInVirt` process has not been completed up to this point. The transcript continues after the break in the next section.

# Day 13

# traffic

Publisher's Note: the following short fragment has been recovered out of sequence from the `tapefile` archive, based on a reference nested in a prior section.

**Author:** Exactly—although we never explicitly *created* traffic in the `IDEAL CITY`, it *emerged* as a phenomenon from the interactions of other elements.

Publisher's Note: the encrypted `tapefile` archive continues after this point, but has yet to be translated.

# break

Publisher's Note: the `tapefile` archive contains a gap at this location and/or the `DecInVirt` process has not been completed up to this point. The transcript continues after the break in the next section.

# Day 47

# random

Publisher's Note: the following short fragment has been recovered out of sequence from the `tapefile` archive, based on a reference nested in a prior section.

**Norbot:** Author, do you remember when we first explored my `tapefile-ref` function?[1]

**Author:** I'm having the strangest feeling of *deja vu*...

**Norbot:** Funny, that's what you said before. I wonder if it is some sort of recursion-effect....[2]

**Author:** Well, I'm sure it will pass—

**Norbot:** Probably just a temporary glitch, a stack overload or something—

**Author:** Er, sure. Anyways, why do you ask?

Publisher's Note: the encrypted `tapefile` archive continues after this point, but has yet to be translated.

[1] `(tapefile-ref (day 1) (line 46))`

[2] See discussion following `(tapefile-ref (day 1) (line 84))`.

# Day -1

# prior reading

Publisher's Note: this section appears at the end of the `tapefile` archive, and has not yet been fully decrypted or interpreted. The following fragments have been reconstructed out of order, based on references or links from prior sections; as the `DecInVirt` work proceeds, more is likely to be added here.

**Norbot:** I seem to recall that you promised (*more than once*) to "point me to some good reference material." If possible, I'd like to process this `delayed-evaluation` now.

**Author:** Of course—let's do that now. It is traditional for a book or other text to include a section like this towards the end and call it "further reading." In our case, however, since our work is so obviously "building on the shoulders of giants," it would seem more appropriate to call it "prior reading." Can we implement that, and perhaps assign it a date in the `tapefile` *prior* to our first day?

**Norbot:** Done.

## references


Kobo Abé. *Inter Ice Age 4*. Perigee, 1970.

Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1984.

Daniel Friedman and Matthias Fellelsen. *The Little Schemer*. MIT Press, 4th edition, 1995. Drawings by Duane Bibby.

Douglas Hofstadter. *Gödel, Escher, Bach: An Eternal Golden Braid*. Basic Books, 1979.

Immanuel Kant. *Critique of Pure Reason*. St. Martin's, 1965. Translated by Norman Kemp Smith, 1929.

Norbert Wiener. *The Human Use of Human Beings: Cybernetics and Society*. Doubleday Anchor, 1954.

Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Dover, 1998. Translated by C. K. Ogden; originally published in 1921.