Dis-encapsulation: Object-Oriented Programming and the Phenomenology of Experience

Anthony Curtis Adler

ABSTRACT

Starting out from a general consideration of the concepts of code, abstraction, and programming paradigm, and of how computers compel us to reconceive of thinking as *technē* rather than as the realization of a given natural endowment, this essay argues that object-oriented programming, through the concepts of encapsulation and interface, offers powerful resources for reorienting ontology. Just as the interface, which mediates between a human agent and a hidden (encapsulated) mechanism, is itself a technical accomplishment, the philosophical text, rather than serving as a representation of reality, can itself be understood as the production of a kind of interface (with beings, with Being, or even with that which is beyond Being)—or, in other words, as a script whose interpretation yields a concrete system of encapsulation and dis-encapsulation. This, in turn, offers a new way of approaching the central concern of Martin Heidegger's later philosophy: the relation of technology and truth, and ultimately also politics and political subjectivity.

KEYWORDS computer, object-oriented programming, Heidegger, ontology, technology

Ex-position, Issue No. 44, December 2020 | National Taiwan University
DOI: 10.6153/EXP.202012_(44).0001

Anthony Curtis ADLER, Professor, Underwood International College, Yonsei University, Korea

Introduction

The computer was a dream before it became a reality. The famous automata of the eighteenth century, whether genuine or fraudulent, testified to the human obsession with producing a mechanism that would emulate the capacities of the human mind and even perhaps attain self-consciousness. As the theoretical and technical means of automated computation fell into place, making possible the powerful transistorized machines of the present, this obsession has only grown in intensity, despite the fact that AI research, after experiencing setbacks in the pursuit of the "strong" goal of emulating general human intelligence, has for the most part redirected itself toward the "weaker" task of mastering more narrowly defined tasks such as pattern-recognition or game strategy.[1] Yet our fascination with artificial intelligence in its most dramatic forms—strong AI, the "hard problem" (in David Chalmer's words) of consciousness, and "singularity"—has tended to obscure the fact that, with respect to certain tasks traditionally included in the scope of human intelligence (memory as the storage and retrieval of information, calculation), computers, almost from the beginning, have far exceeded human capacities. While of course many, even seemingly trivial, human accomplishments still surpass the capacities of digital brains, and while self-consciousness remains as elusive as ever, computers have been, almost from the outset, capable of mind-boggling feats of memory and rapid calculation, feats that indeed far exceed human capacity.[2] This turn has given new impetus to the philosophical inquiry into the nature of human intelligence, one which, in the main, has taken two different directions. For some, the capacities of computers offer new resources for pursuing the mechanistic reduction of human intelligence, an approach already taken by John von Neumann in his 1956 Silliman Lectures at Yale University, whose unfinished manuscript was published under the title "The Computer and the Brain." Drawing at once on his own work as a pioneer in computing as well as early research into neurons, von Neumann describes both the similarities and dissimilarities between the computer and the brain. Ultimately, however, even the most striking differences suggest not that the brain *isn't* a computer, but only that it is of a radically different kind, above all since its primary language is not the

[1] Regarding the distinction between Strong AI and Weak AI, see Searle.

[2] The ENIAC, widely regarded as the first Turing-complete electronic general-purpose digital computer, was by current standards extremely slow, running at a mere 100 khz; nevertheless, it already far exceeded human capacity, being able to perform "5000 additions/subtractions and 357 multiplications per second" (Van der Spiegel et al. 173).

"language of *our* mathematics" (von Neumann 82). Even Norbert Wiener's cybernetics, while rejecting mechanistic reduction in favor of an understanding of order in terms of negentropy and self-regulatory feedback loops, and while profoundly aware of the need to defend humanistic values against technocracy, nevertheless moves in this same direction.[3] For others, following a script that can itself be traced back to the German Idealist critique of mechanistic reduction and to Kant's second critique or even to Descartes, the prospect of a computer simulation of complex human behavior—of thinking "man" as a "machine"—makes it all the more important to locate the essence of the human as a mysterious principle of consciousness and free will.

Nor do we need to imagine fantastic fusions of organic and silicon neurons or microchips embedded within the brain; if we understand the cyborg not just as the sci-fi fantasy of an organism in which biological and mechatronic parts have been unified into a single contiguous body, but as a functional synthesis of biological and mechatronic systems each possessing the capacity for complex behavior and cybernetic self-regulation, then we are already cyborgs whenever we post to Twitter or Instagram, do a Google search, or access an online library catalog—even when we write a conference paper on our laptop. Computers, laced together into massive networks, have become prosthetic extensions of the organic brain.[4] The result, moreover, is that to an ever greater degree human beings—and also human consciousness—have access to a realm of truth that has only been disclosed to them through these digital extensions. Telling, in this regard, is the gracious response of Go master Ke Jie to his defeat by Google's AlphaGo: "After humanity spent thousands of years improving our tactics, computers tell us that humans are completely wrong. . . . I would go so far as to say not a single human has touched the edge of the truth of Go" (qtd. in Gerrish 231).[5] Rather than simply lament the defeat as an affront to human dignity, Ke Jie affirms the "truth" of the game as a truth that, while existing *for* humans, nevertheless may

[3] For an accessible introduction to the program of cybernetics, see Wiener.

[4] It could be objected, of course, that this is true, in a banal sense, of even the simplest tool. How is the pen different than the word processor? How is the electronic missile control system different from the sword? Were cavemen already cyborgs? It is important, against this, to stress that a cyborg requires the augmentation of the biological body with mechatronic "self-regulating" systems allowing complex behavior rather than with simple mechanical tools. A powerful argument, however, could be made, along the lines of Lewis Mumford, that complex social organizations, such as were already found in early civilizations, can already be seen as cybernetic machines. Yet such systems are not, in the first instance, extensions of the power of the individual human being, but political/governmental regimes to which the individual is compelled to submit, and through which he is integrated into a *common body*. It is not a question of the cyborg but the leviathan, though significantly, as is already clear in Hobbes, the organization of the leviathan is itself necessarily mechanical, and one might also speak, with the advent of the Internet, of a *cyborg-leviathan*.

[5] See also Dou and Geng.

well be revealed in novel and powerful ways, far beyond unaccompanied human capacity, through the computational devise. It is only the still rather clunky modes of access—mediated through our sensomotoric apparatus—that allow us to convince ourselves that we exist independent of the electronic systems which, to an ever greater degree, we enrich our lives with and submit our lives to. One could even say that the typical sci-fi vision of the cyborg—a single eye glowing red, skin opening up to a macabre mix of veins, muscles, wires, and transistors—is more regressive than radical; it holds on to the fantasy that the human being exists as a discrete body, an island unto itself.

The consequences of this, however, are to be found not just in the quantity of information suddenly made available to us—a vast externalized memory—or the computational feats that we can now almost effortlessly carry out. Computers present a new way of thinking about, and existing in relation to, thinking itself. They not only make it possible, but compel us, to conceive of thinking itself no longer simply as the accomplishment of the mind (merely finite or perhaps infinite and divine) whose essential nature has been settled in advance and can serve as the *a priori* condition of cognition, but as a vastly complex technical accomplishment. It could, nevertheless, be argued that the *a priori* view of the mind, given most powerful articulation by Kant's *Critique of Pure Reason* but already implicit in Aristotle's account of the soul, had already been displaced by the "historicism" and "evolutionism" of German Romantic philosophy, and in particular by the significance attached to language as the medium of thought, which in turn paves the way for the explicit "linguistic turn" in philosophy and the Sapir-Whorf hypothesis. Moreover, with Darwin and Marx, the Idealist account of evolution gives way to a more materialistic account; the human brain can now be seen as an evolved "technology" responding to specific environmental challenges, whereas the multitude of different cultures and languages are no longer, as they were for Herder, "spiritual essences" rooted in the life of the nations through which the human race reveals itself in the course of historical time, but a "superstructure" built on the foundation of concrete forces and relations of production. These philosophical developments certainly lay the groundwork for the transformation in the conception of thinking brought about through computers, in the same way that Platonism lays the groundwork for the Christian doctrine of the soul. Yet if it had long been difficult to think of human intelligence as an eternal, unchanging *a prior* disposition, it was also not yet possible, beyond a very limited and esoteric sphere, to envision a concrete technical invention of the technology of thinking itself. This is, above all, because a chasm still seemed to separate the tools of thinking, which may be freely invented, from thinking itself, which still is oriented

around a mysterious ensemble of traits (consciousness, self-reflective awareness, imagination, freedom) that ultimately serve to constitute the "end," the "for the sake of which," through which the use of these tools is organized. Even Ludwig Wittgenstein, while going so far in the *Philosophical Investigations* as to understand thought as grounded in language, and language as rooted in concrete "forms of life," never ceases to think of language as either a *tool* or a *game*, or indeed as both *tool* and *game*, and in this way continues to refer language to a "for the sake of which" beyond itself while leaving unanswered the question, already raised by Plato, as to whether man, the "puppet of the Gods," is "put together . . . for their play or for some serious purpose" (Plato 644d). With the computer, this divide is breached: thinking itself (rather than just the language in which it takes place) is a technical accomplishment. But if the *computer tool* is itself thinking, then it is no longer clear, as it once had been, that thinking is that which *deploys* tools, and not rather a tool, a *technology*, that deploys itself. Thinking collapses into pure instrumentality, "means without ends": it has become a kind of pure *technē*, "art for the sake of art" in a far more radical sense than that envisioned by Romantic aesthetes—since the art in question is no longer submitted to the "humanistic" end of beauty. This is already intimated by the curious status of computer science as a science of the artificial rather than the natural.[6] Because of this transformation in thinking, the grounding of philosophy in self-reflection, and ultimately in the self-contemplation of the divine mind, appears more questionable than ever. It is not merely that we have cause to question the logical consistency of such grounding or even their ideological presuppositions: a new and affirmative model of thinking has emerged. If one could still speak of an *a priori*, it cannot be, as for Kant, an *a priori* specific to human cognition but an *a priori* that applies to any possible mechanism of thought.

The implications of this are immense. Nor are they restricted only to those philosophical tendencies, such as Cartesian Rationalism or German Idealism, where self-reflection plays an explicit role, or even to those which, following Socrates, seek self-knowledge and insight into the nature of the human. The very idea that philosophy somehow concerns *what is given*—that it takes its departure from some kind of original evidence—must be called into question. And indeed, the obsession with artificial intelligence might itself be regarded not as a premoni-

[6] For a summary of the development of computer science as an academic field, see Peter J. Denning and Craig H. Martell 3-10. As Denning and Martell note, while "computer science" originally struggled against the common perception that science was concerned with nature and had to gain legitimacy for a "science of the artificial," in recent decades, "information" has itself emerged as a primary paradigm for understanding nature.

tion of the future but as the last residue of a philosophy modeled on self-reflection. Self-replication is perhaps all that remains possible for self-reflection in the wake of the computer's rise, the last refuge for those who would deny that computers have introduced a change in the very nature of human cognition. Perhaps the obsession with an artificial intelligence conceived *after* the model of organic intelligence reveals nothing else than the desire, at the very point when the essential nature of human intelligence is itself changing, to preserve it by way of replicating it.

Logotechnicians

When philosophy was understood, implicitly or explicitly, in terms of self-reflection, then the distinction between the philosopher and the non-philosopher rested ultimately in the capacity of thought to turn away from the world, away from the immediate apprehension of the given, and toward thought itself as the element through which the given is given. Just as for Socrates the philosopher is the one enjoined to know himself, for Hegel the grandeur of speculative thought unfolds as the identity of substance and subject. But if thinking is no longer the element, whether passive or active, through which the given is given, but rather that which is produced through a *technē* that, without replacing organic human intelligence and consciousness, nevertheless not only supplements and augments its capacities but transforms its very habitus, then the distinction between the philosopher and the non-philosopher, if such a distinction remains relevant, must assume an altogether different aspect. This is not to say that philosophy in the prior sense ceases to exist as an institutional reality or intellectual discipline. But another stance has become possible; another distinction comes into view. This distinction concerns the relation not to *thinking* as the given (organically embodied) element through which the given presents itself in its truth, but to the technical apparatus of thinking.

The information-technical apparatus presents itself first of all as one tool among others, yet with fixed, if incessantly expanding and indeed uncannily pluripotent, capacities. Unlike a more ordinary tool such as a hammer, this tool is composed of many different, complexly interrelated, components: the physical hardware of a single computer, the operating system, software running on top of the operating system, and the networks binding computers together into systems of staggering complexity. It is not, however, merely its complexity that distinguishes it from other tools. A modern jet airplane, for example, is bewilderingly complex, not least since it includes advanced computational systems. And yet in one regard it is simple: even simpler, perhaps, than a hammer. The use for which it is deployed is very narrowly circumscribed, and while certain misuses may be

possible—turning it into a hotel, using it as a weapon—these misuses, if they do not result in the destruction of the tool itself, remain prohibitively wasteful so long as the airplane is still salvageable, and if it is not salvageable, they no longer involve putting the airplane itself to use *as* an airplane but merely its parts. The airplane's complexity results, in other words, from a specialization of its function, as is typically the case when tools are combined together into a machine; a gasoline engine, for example, can serve many purposes, but when it is joined with wheels and a steering mechanism, it becomes a means of transportation.

The computer, in contrast, does not present itself to us as a single discrete object with a well-defined range of normal uses together with a more open-ended set of deviant uses, but as a general-purpose tool that can be adapted to an unlimited number of different purposes, and whose concretely actualized capacities are constantly developing and expanding. While it is possible to establish absolute limits to these capacities—this is the subject of much work in theoretical computer science—this does nothing more than establish the outermost frontier for a world that remains constantly growing and evolving. This aspect of the computer reveals itself with particular force in the use of general-purpose computers that have been in effect down-purposed: turned into more mundane tools. While in actuality most electronic devises contain general-purpose computers programmed to perform a limited range of functions, these are hidden behind an interface. There is, however, something slightly disturbing—even uncanny—about an iPad at the local café carapaced in a plastic shell and reduced to a credit card reader.

For the most part, however, the relation that people have to these general-purpose tools, even if not quite so dramatically as with the credit-card-reader-encased iPad, ends up approximating itself to an ensemble of more ordinary tools: word-processing, email, games, spreadsheets, and media players. While the computer itself remains a general-purpose devise, we relate to it only through a specific range of preprogrammed functionalities. And just as the philosopher for the most part exists in the mode of ordinary everydayness—it was the great accomplishment of Martin Heidegger to challenge the tendency of philosophical reflection to abstract away from everyday being-in-the-world; from the mode of existing in which we relate to things not as objects of theoretical attention but as tools—nearly everyone for much of the time relates to the information-technical apparatus in the manner sketched above. But a few, nevertheless, also enter into a different kind of relation. The computer reveals itself not just as that which is in any given instance a fait accompli—a finished artifact, whose manifold potencies have been wrapped together into a unity—but as a mechanism that can itself be transformed, built upon, "hacked into": a mechanism whose general computational

powers can be actively and creatively exploited. While it still remains a tool, the very mode of its tool-being as well as the relation we have to it has changed decisively. It shows itself not simply through its effects but in its inner workings: a network of processors, data links, and storage devices working in concert to perform simple logical operations at a bewildering speed. Moreover, though, it presents itself as the theoretical model for a mode of thinking, the universal Turing machine, that is concretized and embodied, and in this sense realized (while only approximately) through a technical devise, and that, through a combined understanding of the theoretical model itself and its technical realization, can be put to use—not only for certain fixed tasks, but for an open range of purposes, including ultimately the replication of fluid intelligence.

Just as one had previously distinguished between the non-philosopher and the philosopher, between the one who merely thinks and the one who thinks about thinking, we now might distinguish between the multitude, who either use or play around with the tools of thinking—who merely pick them up and put them down and employ them—and those who, by way of their own thoughts, not only have entered into a thoughtful relation to these tools, reflecting on their use and becoming aware, as it were, of their tool-being as such, but are indeed playing with them and working with them from the inside, fashioning from within them the tools and toys that they want or need. There are, this is to say, *logotechnicians* and *non-logotechnicians*.

The Ubiquity of Coding

Such a distinction might at first glance seem questionable—as questionable as that which it seeks to replace. Just as there is no thinking that is not given over to the infinite task of self-reflection—this was the great discovery of the Jena Romantics—there is no relation to the information-technical apparatus, however crude, that is entirely free of "coding." To use a computer, even if just for Snapchat, involves mastering a kind of code. Code in this most general sense refers to any semantic relation that is, on the one hand, distinct from the "natural" language of oral communication and, on the other hand, not immediately bound up with and comprehensible through the physical relations constituting the natural world.

The simplest example of "coding," in this sense, is using an on/off switch. The relation between the switch-state and the outcome (light-on/light-off) is arbitrary. To use a switch properly, you must possess a kind of abstract knowledge: Does being-flipped-up mean "on" or does it mean "off"? Such knowledge is not fundamentally different from the more explicitly technical knowledge of the

voltages that correspond to different binary values in a transistorized logic-gate. In contrast, the use of a steering wheel does not present itself as a "code," since there seems to be a "natural," motivated relation between the use of the wheel and the action (turning to a certain degree either left or right). If there were a significant number of steering wheels that behaved in the opposite way, as is sometimes the case with paddleboats when the tiller is disguised as a steering wheel, then it would become a "code"; one would have to know a seemingly arbitrary fact before one could use the devise properly. And of course one might still argue that knowing that the use of the wheel "makes sense" in this way is itself a kind of code; it could be the opposite, as happens when one drives backwards, or, as with certain toy cars, when the wheel lacks the capacity to modulate the degree of turning. Indeed, what prevents it from appearing as a code is a more-or-less intuitive knowledge, either learned from an early age or "hard-wired" into the brain, of basic physical relations between things, such as, for example, allows us to anticipate the trajectory of a ball that has been tossed or even sense when a tree branch will collapse under our weight.

Of course, if we could actually peek inside the light switch and the electrical network, then its semantics would no longer appear arbitrary; it would cease to be a "code." This is itself instructive, however: what makes a code a code is the hiddenness of its interior mechanism. The code, in this sense, must always be understood as the consequence of a lack of understanding: if the real relations constituting the physical world (assuming, as one might ultimately not assume, a "classical" determinate system) were understood exhaustively, as would only be possible for the infinite divine intellect of the sort envisioned by Leibniz, then the "code" would "disappear" into the real relations, and a special knowledge of the "code" as such would cease to be necessary.

This concept of a code might seem quite strange. We might think of a code as just a system of translating one symbol (say: one letter of an alphabet) into another (say: a number); it is a theoretical construct whose material embodiment is irrelevant. A code, then, is simply a transformative function. Yet here we must be careful: a code is not just a transformative function, but a transformative function that is computable, a transformative function that, at least in theory, can be carried out by a physical mechanism. There are, in fact, many mathematically conceivable functions that are incomputable, such as, most famously—assuming the Turing machine as the only physically realizable model of computation—the halting function.[7] Therefore, we might say that a code is only possible if there exists a mechanism

[7] For an accessible account of the "halting problem," see Copeland, especially 39-45.

that *could* carry it out, and that it only actually exists if there *is* a mechanism that carries it out. This apparatus may be nothing more than a human agent who follows certain mechanical operations, such as, in the simplest case, substituting one letter for another letter in accordance with a table. Or it may be a supercomputer chaining together hundreds of trillions of transistors.[8] This changes nothing. The actual existence of "code" consists in the achievement, through a "mechanical" process, of a transformation from one value to another.

Whereas a hammer or a painter's brush (both of which are basically simple machines, whose operation depends on a mechanical transformation) serves as an immediate extension of the powers of the human body, which is in turn applied toward the end of bringing forth physical objects that are themselves material realizations of *forms* and *purposes* existing in the mind of the creator, the code—even if ultimately applied to human purposes (such as concealing information from an enemy)—itself involves an "inner mechanism" that, while at first only theoretically articulated, already demands a machine to bring it forth, and, hence, in the absence of sufficient technical means, turns the human being himself into such a machine. This "inner mechanism," which awaits mechanical realization, is already a kind of "mechanistic" thinking, a technical thinking—a thinking through *technē*.

This "mechanistic" thinking, moreover, seems in a certain sense to belong to all thinking, and in particular to all forms of language, whether human or animal, natural or artificial. Perhaps, indeed, the only thing that distinguishes conscious mental life from mechanistic thought (which already allows self-relatedness in the form of recursion) is that its mechanism has been concealed from it. Whereas a machine *is* its mechanism, consciousness hides the mechanism in such a way that a new system of entities emerges which, within the new field of "experience" that opens, relates to each other as such; conscious experience is simply the possibility of such relations, which seem at least, *for* experience itself, to take on a life of their own. There can be no hiddenness without the possibility of that "for" which the hidden hides by appearing *as* hidden; this "for," as Hegel already recognized, is equivalent to the experience of consciousness.

Natural language, in this sense, could then itself also be understood as a kind of code. Consider, for example, the "language games" described by Wittgenstein in the *Philosophical Investigations*. Insofar as each of these involves an ensemble

[8] It is significant in this regard that the term *computer*, as N. Katherine Hayles notes, was originally applied to human beings, primarily woman, carrying out what was regarded as a kind of clerical work (Hayles 1). The modern computer, this suggests, only became possible once computation, as a human activity, was isolated through the division of labor.

of observable behavior, rather than some mysterious complex of "private" mental states, then it would be possible (or, at least, this can be imagined as a kind of thought experiment, just like the "language games" themselves) to program a computer to simulate any language game, from the simplest to the most sophisticated. If we could know the script of the computer, and the language in which this script was written, then we could recognize its behavior as deterministic. But this determinism would only be relative to the determinism of the mechanism, which itself appears as mysterious. We would still only understand it in terms of the abstraction of code rather than its simple mechanism. In the case of "language games" of actual human beings, the "code"-like nature of the language is only increased, since the "mechanism" in question is even more opaque. It is significant, moreover, that in the case of so-called "neural networks," the foundation of so-called weak AI, the difference in opacity between a deterministic computer and a biological organism becomes less and less clear. The sophisticated neural networks of AI involve computers and programs that, at base, are fundamentally deterministic: no different, in this regard, from any other kind of software and hardware. And yet, once the process of training is complete, it is no longer possible to really understand how the computer gets to a certain result; to the extent that the neural net carries out something analogous to a process of reasoning, the mechanism of reasoning—in contrast to the mechanism of the "neural network" itself—is opaque.[9]

But if it is true that it is impossible to use a computer—or a digital alarm clock, or a simple telephone—without some coding, the converse is no less true. Not even the most technically sophisticated, "intensive" user of the information-technical apparatus ever "touches" the inner workings of the computer save through layers of mediation. Even someone programming in assembly language—almost unheard of today except in the most demanding embedded applications or when creating devise drivers or building compilers—would still approach the "machine language," the instruction set that the CPU itself understands, through some

[9] It seems, nevertheless, that to recognize natural language as a code demands that one occupy a position outside the immanence of communication. Such an "exteriority" emerges the moment when oral communication is brought into relation to writing, or when one language appears as the translation of another language, or even when some sort of split emerges between the sound of a word and a sense. This is not to argue that a pure moment of immanence, free of all exteriority, could exist, but only that human language does not *always* show itself as a code, and indeed, proximally and for the most part, *does not*. At the same time, though, in the case of human natural language, the physical system (not just one human brain, but the entire nexus of organic human life in its sociality and its relation to its environment) is so complex as to be practically impenetrable, while at the same time the experience of language is an experience of the hiddenness of this hiddenness, of a perfect comprehensibility of words that presents itself whenever we use them without reflection.

minimal degree of mediation, including not only the expression of the on-off states of transistors as hex code or decimal representations when indicating memory locations or values, but also a direct translation of the instruction set of the processor into a more human-readable symbolic representation or even some macros to reduce the amount of tedious boilerplate. And even the "machine language" itself consists in a set of instructions that regulate the bewilderingly complex labyrinths of logic and memory gates—transistorized valves—that constitute the actual "hardware" of a computer. And indeed, the switch-like operation of each single transistor itself rests on physical processes involving the flow of myriad electrons.[10]

In a "low-level" language like C, which was first conceived as a kind of "universal" assembly language, additional levels of mediation appear. These new forms of mediation more clearly assume the form of abstraction, symbolic representations of functional aspects of a computer's operation. Some of these abstractions, such as the primitive data types *char*, *int*, and *float*, conceptualize data types that are natively represented across a range of computer architectures. But there are also more complex data types, such as arrays and structs, which build on existing elements. Other abstractions, in turn, identify aspects of the program flow whose ultimate possibility rests on the ability of computer circuits to control program flow through conditional statements.[11] These include not only the basic elements of imperative programming, such as code blocks, numerical and logic operators (and, or, not), conditional operators (if/then/else), and basic loops as well as the much-maligned and largely expunged "goto" statement (which allows a direct leap from one line in the code to another), but also more sophisticated organizational structures such as functions, subroutines that are called within the execution of the main routine. As one moves on to "higher level" programming languages, however, not only do new levels of abstraction emerge, but the abstraction itself functions in a new way: it offers a generalized paradigm for thinking about what a computer is doing, and in turn conceptualizes the act of programming itself. It is striking, indeed, that the names of these new kinds of abstractions ("objects," "classes," "multiple inheritance," "reflection," "introspection," "ontologies") are generally drawn not from a purely technical domain but from the realm of superstructure: social organization, communication, and even philosophy.

[10] For an excellent nontechnical introduction to the physical principles underlying digital electronics, see Steiglitz 22-61.
[11] See Steiglitz 139.

The Computer as Interface: Minimal Phenomenology

It would thus seem that no absolute but only a relative distinction can be drawn between the logotechnic and non-logotechnic users. Considered from a purely technical perspective, there is only a continuum. Yet this is not the only perspective through which the information-technical apparatus presents itself and in terms of which it must be understood. The computer does not exist as a purely technical object, but as an interface; it exists as the threshold of encounter between embodied human sensomotoric experience (or we might even say: being-in-the-world) and the technical apparatus.[12]

This interfacing, in turn, assumes two qualitatively distinct modes. On the one hand, the interface may present itself as a horizon filled out with beings in such a way that the meaningfulness of every being and relation between beings within the horizon is fully "satisfied" within the horizon itself, with all access beyond this horizon blocked off. On the other hand, the interface may present itself as that which exists in relation to an inner mechanism that is concealed, hidden away behind the interface, and yet at the same time rendered accessible through the interface.

Drawing a rigorous distinction between these two modes of presentation—modes of the *phenomenality of the phenomena*, or, in other words, ways in which the thing, as that which shows itself, shows itself—is one of the great philosophical accomplishments of the late nineteenth and early twentieth centuries. The distinction itself, however, not only preexists the advent of modern computing but traces back to the earliest roots of Western philosophy. Intimated already in the opposition that the Socrates of Plato's *Phaedrus* draws between materialistic and telic explanations, this distinction gains even more clarity with Aristotle's description, in the first chapter of the *Nicomachean Ethics*, of the *telos*. Yet it is suppressed by the hegemonic metaphysics that emerges through the Scholastic reception of Aristotle: by seeking a causal explanation of physical reality in terms of a *telic* first principle, Scholastic metaphysics offers a kind of détente between the two modes

[12] For a powerful account of the computer as "interface," see Alexander R. Galloway. Galloway argues: "[I]nterfaces are not simply objects or boundary points. They are autonomous zones of activity. Interfaces are not things, but rather processes that effect a result of whatever kind. For this reason I will be speaking not so much about particular interface objects (screens, keyboards), but *interface effects*. And in speaking about them I will not be satisfied just to say an interface is defined in such and such a way, but to show how it exists that way for specific social and historical reasons" (vii). It would be impossible, in the scope of this essay, to engage with Galloway's complex and rich account of the interface; I would only state one key point of difference. I believe that the "mediation argument" that he quickly rejects—the claim that the computer "remediates" an obsolescent metaphysics—needs to be taken seriously, though perhaps in a different sense than understood by Marshall McLuhan and Friedrich A. Kittler.

of inquiry; the formal and the material are fused together into a unity. It is only with the emergence of the modern scientific method that materialist explanation again came to the fore, and this in turn brought a crisis to metaphysics, culminating in psychologism's attempt to elevate the empirical explanation of psychological phenomena to the status of *proto philosophia*. The rise of psychologism, however, brought resistance from many different fronts: logicism, phenomenology, neo-Kantianism, and even neo-Aristelianism. In this way, the alternative mode of presentation was brought sharply into focus, freed from its metaphysical interpretation, as a field of meaning, sense, and signification irreducible to material or psychological reality. Even so, the polemical character of these philosophical discussions, and the internecine conflicts that broke out between the different parties of opposition, obscured the fact that what was at stake was not identifying the one true method of philosophy but distinguishing between qualitatively distinct *modes of encounter*. It is a question, one might say, of the most fundamental and basic distinction between different modes of phenomenological givenness—*a minimal phenomenological distinction*, in other words.

The distinction between the logotechnic and non-logotechnic users comes down to how the interface presents itself. But at the same time—and this is the decisive point—the qualitative distinction in the mode of interfacing is also a technical achievement. Just as what had once been esoteric and exotic scientific, military, and industrial technologies developed into the "personal" computer, the interface, which in the first computers involves physical rewiring of the circuits, underwent a series of transformations. Not only did it develop more and more into a kind of simulacrum of ordinary everyday experience, but the interior workings of the computer became shielded more and more behind the interface, culminating in tablet operating systems such as iOS. This technical development, which originated with the research done at the Xerox PARC laboratory, satisfied an obvious commercial need, though a need which belonged not to the present but to the future, and which therefore took the visionary genius of Steve Jobs to realize, if imperfectly.[13] The personal computer, however, realized not merely a certain fantasy of the imagination but the very fantasy of the imagination itself—of images bound up with images through a law of pure association. The personal

[13] The iOS-based tablet might be seen as a realization of the concept of a "personal computer" vividly described by Alan Kay and Adele Goldberg in their essay "Personal Dynamic Media" (1977), yet in certain respects it deviated quite significantly from their program: programming or coding as a mode of interaction with the machine has been suppressed in favor of a pure interface, and therefore the interactive pedagogical role of the computer, envisioned by the PARC researchers and exemplified in the Smalltalk language—which began as a way of teaching children programming—has been almost entirely suppressed, giving way to largely passive forms of entertainment.

computer is, one could even say, the Romantic medium *par excellence*. This is felt whenever one falls into the strange rhapsody of "web surfing."

Thus, while the distinction between the logotechnic and non-logotechnic users remains merely a matter of degree when regarded from the technical perspective, according to which the information-technical object always presents itself as a relation between the physical mechanism and the code, it constitutes an absolute, qualitative distinction when regarded from the perspective of the interface. The interface is neither merely technical nor experiential, but is, as it were, the *technical accomplishment within experience of the very relation between two modes of interfacing—two modes of phenomenality*. The information-technical apparatus not only exists in this duality, as—in another sense—does human existence itself, but exists in this duality *as* a duality that, in a certain way, it has itself accomplished and that it could even be said to have a mastery over, though a very tenuous mastery—a mastery that is of the "hacker" and not of the "engineer," a mastery that exhausts itself in interventions that cannot take command over the technical apparatus as a whole.

Encapsulation

In the section "The Ubiquity of Coding," I spoke of abstraction in programming languages as a mediation of the inner workings of the computer. One of the highest levels of abstraction possible is the idea of the computer-language paradigm. Computer languages can facilitate a certain paradigm, and they may also compel the computer programmer into a certain paradigm.[14] Yet paradigms are far more than merely features of a given language; rather, they are ways of thinking abstractly about what a computer, as a theoretical object, is, and what we are doing when we program it.

The two dominant paradigms, whose various strengths and weaknesses remain subject to endless debate, are called *functional programming* and *object-oriented programming*. Both of these, I would now suggest, involve an abstract representation of the computer *as* interface. More specifically, they bring into view *encapsulation*: the concealment of the "interior" properties of an entity, the hiddenness of "information." In the case of functional programming, what distinguishes a function from the more general concept of a procedure or subroutine is that the function does not have any "side-effects": the function accepts an argument and

[14] Thus Alan Kay, one of the principle innovators of the Smalltalk language and "personal computing," distinguishes between two kinds of languages: those that are an "agglutination of features" and those that are a "crystallization of style," or, in other words, a paradigm (Kay 512).

returns a value, and it is exclusively the relation of the former to the latter that constitutes the behavior of the function. The function is neither affected by, nor capable of changing, system states external to the function. In this way, the inner mechanism of the function—the actual program code that transforms the input into the output—is hidden; the function is a "black box." The object-oriented programming paradigm, on the other hand, treats a program as an ensemble of objects, which communicate to each other by "passing messages." In the case of Smalltalk, for example, even a simple arithmetic calculation involves a message passed between objects: in the case of "1+2," for example, the object "1" (belonging to the class "integer," which determines how it behaves) receives the message "+" together with the object "2," and then sends its result (the object "3") as another message. Each object, in other words, consists in an "interface" which determines its relation to other objects, and in this sense, moreover, every "object" is itself a "microcosmic" representation of the computer itself, which, in turn, may be regarded as an "object" with an interface, passing messages back and forth with other entities.[15] The program relates to the object through the interface; the particular code that implements the functionality of the interface is itself off-limits to the outside.

Both the functional and object-oriented paradigms, this is to say, are abstract representations of encapsulation. But there is no encapsulation without dis-encapsulation: the significance of encapsulation lies in the fact that not everything is encapsulated—not everything is hidden away behind the object or function. A perfectly black box would not be anything at all. From a technical perspective, encapsulation has a strict and concrete meaning. In a strictly object-oriented language, attributes of an object, such as variables defined within its lexical scope, cannot be accessed from the outside unless one creates special methods ("getters" and "setters"). In a function, on the other hand, it is the operational states that are "encapsulated," in the sense that, issues in performance notwithstanding, the "meaning" of the function in the given computational context consists in the specific transformation from input to output that it executes, while everything else is irrelevant—which would not be the case if the function has "side effects." Yet from the non-technical perspective—and, as noted, both perspectives must coexist insofar as the computer itself exists as an interface between the technical and the non-technical—encapsulation has a much broader meaning. Or indeed, it first becomes meaningful rather than merely having a finite reference to a certain technical feature of a

[15] For a rather different account of the philosophical implications of object-oriented programming, see Evens. Given the recent vogue of "object-oriented ontology," surprisingly little has been written on this subject.

programming language that in turn must be implemented by its compiler or interpreter. The meaning of encapsulation, indeed, is metaphysical, in the strict sense that it refers to an aspect of thing-hood that is true not only of spatiotemporal, materially embodied things, but also of things in the most general sense, including *thoughts*, *numbers*, *spiritual entities*, the *mind* or *soul*, as well as various kinds of physical things, such as *living organisms*. *Encapsulation* is what it means for a thing to be a thing: what makes a thing a thing. It is nothing less than the meaning of Being.[16] A thing *is what it is* because it not only shows itself in a certain way but hides what it otherwise might be behind a certain interface or façade. These hidden aspects also belong to what a thing is, but as hidden, and it is only through a certain transformation, in which the entity (so far as it reveals itself as an object of reference) may remain the same, that the sense of its Being changes. Thus, for example, a wild horse can be made not only into a living tool (a domesticated animal) but also into a body to be operated on, or even a corpse to be buried and honored as a beloved member of the family, or turned into meat or rendered into glue.

This might, at first glance, seem like a preposterously bold claim, and it does not seem possible to justify it completely in the context of this essay. Let it suffice to show, through a relatively simple example, how one might begin to approach metaphysics in terms of encapsulation. Consider again, for example, a living organism like a horse. What is it that makes it alive? For Aristotle and his followers, it is the soul, which he understood as an active power of organization explaining the various capacities (nutrition, sensation and movement, thought) belonging to a living being. Such an explanation seems mystifying, of course, but it draws attention to something of the greatest importance. Organic life becomes possible by creating a boundary between inside and outside. This boundary, in turn, allows the organism, having constituted itself as a kind of interior, to interact with the exterior through a limited set of potencies. Each of the three basic capacities that Aristotle identifies can then be seen as offering more sophisticated forms of the

[16] The metaphysical possibilities of the concept of encapsulation are suggested by Alan Kay when he writes, "Philosophically, Smalltalk's objects have much in common with the monads of Leibniz and the notions of 20th century physics and biology. Its way of making objects is quite Platonic in that some of them act as idealizations of concepts—*Ideas*—from which *manifestations* can be created. That the Ideas are themselves manifestations (of the Idea-Idea) and that the Idea-Idea is a kind of Manifestation-Idea—which is a-kind-of itself, so that the system is completely self-describing—would have been appreciated by Plato as an extremely practical joke" (Kay 512-13). The deepest paradox confronting the Platonic theory of forms becomes a practical reality in Smalltalk, such that the concept of recursion—"[i]n computer terms," Kay goes on to explain, "Smalltalk is a recursion on the notion of the computer itself," meaning that every "object" is a "recursion of the entire possibilities of the computer"—is the basis of a self-describing computer language (513).

encapsulation that is at play. In the case of the nutritive, it is merely a question of metabolic exchange, whereas through the sensomotoric apparatus the living being becomes an agent interacting with the world. And indeed, perception itself involves encapsulation. The senses do not so much present the world to the organization as collapse it into an ensemble of objects and relations that are themselves encapsulated in various ways, offering certain modalities of interaction—or as we might say, a certain set of methods. And finally, thought itself involves a further encapsulation, one which indeed brings the encapsulated structure of the perceptual world into view. The animal already relates to the world as an ensemble of things that have various potencies, various "methods" attached to them. This piece of furniture, for example, offers this cat specific possibilities of engagement which, through its proverbial curiosity, it discovers and exploits. And it is likely that a cat will draw on experience when confronted with things similar to what it has already encountered. What the human mind alone is capable of, however—at least according to a long-standing metaphysical dogma—is recognizing the thing *as* this thing. But this very act of recognition amounts to nothing else than hiding the specificity of the thing behind the general type or class to which it belongs while attaching a name to that class. *It is now a chair*. Here a kind of encapsulation—as well as dis-encapsulation—takes place. It is not only the *thisness* of this and every other chair that is hidden through the word "chair," but every particular differentiating feature of it, and indeed the sheer fact of its existence.

Metaphysics as the Encapsulation of Encapsulation

The advantage of the concept of encapsulation, however, consists in bringing together different kinds of entities into a single analysis without losing sight of the "concreteness" of the thing. This concreteness, not to be confused with spatiotemporal existence, involves the concreteness of the relations through which it exists in relation to other things. A number, or even a far more abstract mathematical property (such as primality), is concrete insofar as what it is must be understood in terms of the mathematical context to which it belongs, which, in the case of single natural numbers, consists in the entire set of natural numbers and the elementary operations that range over them. But of course the conceptualization of the context is always a kind of abstraction from the larger mathematical context, and hence itself an encapsulation. It may, for example, consider integers as a group in terms of the operation of addition or multiplication while abstracting away from those operations, such as division, that would violate the principle of closure—potentially resulting in a value that is not itself an

integer.[17] Likewise, in the case of a simple single-celled organism, the concreteness involves the manifold relations existing between the organism and its environment. The organism is not a thing because of the merely abstract, strictly logical property of being-a-unity; rather, what makes *this* thing the thing that it is are the various ways that it is able to preserve an interior stasis in relation to the outside. The thing is a thing as a relatively closed system.[18]

As soon as we claim, however, that encapsulation is the meaning of Being—that encapsulation is what it means for a thing to *be* the thing that it *is*—then we are compelled to ask: What is the status of traditional ontology, of metaphysics as customarily understood? Is it a mere error? A delusion? We have, of course, already treated language itself as a kind of encapsulation, but what about the special terminology that arose to talk about the nature of beings in the most general sense, and in particular the concept of substance and essence? When the question is posed in this way, it becomes immediately clear that ontology in the sense of *metaphysica generalis* is itself a kind of encapsulation, and indeed an encapsulation of the various *concrete forms of encapsulation* of which I have spoken. Substance, for example, may be understood as that which *hides* the manifold differences of the thing behind the unity of its self-identity, which in turn is what shows itself to us through the essence, and the attributes belonging to the essence, in contrast to the accidents. A substance, in effect, is that which is "closed" under a set of transformations (variations of its accidental properties such as changing the color of Socrates's skin), though there are other transformations (such as changing the "essence" of Socrates) that will violate this closure. When something shows itself as a substance, in other words, it shows itself through its identity to itself, and its difference from itself is "encapsulated" behind this self-identity, to the extent that it can no longer be accessed immediately, but only through the mediation of its relation to the self-identity of the substance, such that a radical and originary difference—the difference of the thing from itself—comes to present itself as accidents belonging to a substance.

It is clear, moreover, that metaphysical encapsulation happens through language—indeed through logical and grammatical structures that are themselves self-encapsulations of language insofar as, through them, language presents itself

[17] A given set (say: natural numbers) is *closed* under a given two-term operation (say: addition) if, for any two members of that set, the operation yields a member of the set. The sum or product of any two natural numbers is always a natural number, whereas natural numbers are not closed under division since, say, 3 divided by 2 yields 1.5, which is not a natural number.

[18] As David West observes, considerations of both biological systems and Heideggerian hermeneutics played a role in the early development of object-oriented programming (59).

as a simplified model of its own operations. Language may, for example, understand its own operations in terms of the *naming* of objects; the grammatical concept of subject and the metaphysical concept of substance reinforce each other, such that ultimately the two converge. It is on this basis that we can speak of onto-logy. But it is no less clear, from all that we have said, that ontology in this sense not only is a special kind of encapsulation but stands in a peculiar relation to the "concrete" forms of encapsulation we have already identified. The "abstract" ontological encapsulation, first of all, exists in a necessary relation to the "concrete" encapsulation; that which can be spoken of *as* a substance must somehow exist and show itself in its existence. For precisely this reason, the attempt to critique metaphysics by way of a "linguistic turn"—as, for example, when Wittgenstein begins his *Philosophical Investigations* by arguing that Augustine's concept of language as denotative, along with the substance ontology that it underwrites, reduces the rich texture of language to a simple primitive kind of language game—comes at the price of phenomenological blindness (2-3).

This existence, which need not be thought of only in spatiotemporal terms, involves both encapsulation and dis-encapsulation. Yet in the case of the concretely pre-existent thing which gets shown as a substance, encapsulation and dis-encapsulation belong together and existence takes place as the interplay between them. The cell is at once always communicating with its environment but at the same time gathering itself back into itself, preserving its interior stasis: encapsulating itself in relation to its environment. With the ontological revelation of the concrete thing, however, a decisive transformation in the relation of encapsulation and dis-encapsulation takes place. What is already dis-encapsulated in the concrete pre-existing thing—the quality of enduring in self-sameness over time—becomes revealed anew through the explicit relation of substance and accidents, whereas what is hidden "behind" and "by" the self-same thing is banished into the absolute obscurity of a pure materiality that does nothing more than "obscure" the thing itself, and which, as the realm of pure exteriority and non-self-identity, must be banished for the thing to *truly* become the thing that it is. In this way, the being presents itself only as dis-encapsulated, or, rather, no longer even as dis-encapsulated but as pure presence. The fundamental trait of metaphysics, as Heidegger noted, is the privilege of presence.

Ontological Dis-encapsulation

In the last section, I started out from the distinction between the logotechnic and non-logotechnic users. This distinction may now be reframed. The non-logotechnic

user exists in the medium of *ontological dis-encapsulation,* in which the encapsulated has disappeared so much from view as to disappear into absolute nothingness, a nothing that manifests itself only through the spectral intrusion of "bugs," ghosts in the machine. Ultimately, the machine itself only shows itself through glitches. The non-logotechnic user, in other words, remains within the horizon of onto-logy. The ground of ontology, to which it must return to and renew itself through, is phenomenality. The phenomenality of the phenomena is its appearing in absoluteness and immediacy, which is to say: simply as what it is. The phenomenological perspective is one in which the tree appears simply as a tree—in its original givenness as a tree—and not as a "tree-sensation" or "tree-perception" or a "tree-idea" or even as a physical entity. Yet the phenomenality from which ontology takes its departure is necessarily a mode of phenomenality in which concealment—encapsulation—is itself suppressed so much so that it cannot appear at all save as a *limit* of appearance. The logotechnic user, on the other hand, is the one for whom the distinction between the encapsulated and the dis-encapsulated shows itself as something that is itself achieved through the technical apparatus.

Human being-in-the-world involves a fundamental and originary encapsulation, which takes place through sensation, perception, thought, language, and indeed through all the complex forms of cultural life and being-with-others. Metaphysics, in turn, encapsulates this encapsulation behind pure presence, pure manifestation, and hence renders it inaccessible. To exist within the horizon of metaphysics therefore is to exist in a surface for which there is no outside. But it is precisely this mode of existing—this mode of being in the truth—that the technical apparatus itself replicates for the non-logotechnic user. For the logotechnic user, the dis-encapsulation of the information-technical apparatus presents itself as an accomplishment of the technical apparatus; the depth of the machine reveals itself behind the surface of windows, menus, icons, and links.

Dis-encapsulation and Unconcealment

In the preceding pages, I have done little more than reframe Heidegger's account of truth as *un-concealment, a-letheia*. Heidegger's "Origin of the Work of Art" presents a revolutionary rethinking of the nature of truth that is already underway in his *Being and Time*. Originary truth is not mere "correctness," not merely the correspondence between language or thought and reality. Not only does propositional truth, as he already insists in *Being and Time*, already depend on the prior disclosure of beings, but disclosure is only possible on the basis of a

prior concealment ("Origin" 115). Originary truth, then, is not simply the truth of pure disclosure, *the lumen rationis*, but is the play, and even the altercation, between concealment and unconcealment. Originary truth is an un-concealment, a removing of hiddenness from out of hiddenness.

And it may also be clear, as the word "framing" suggests, that another of Heidegger's seminal texts is also in play here: "The Question Concerning Technology." "The Question Concerning Technology," which belongs to the beginning of the last phase of Heidegger's philosophical career, reprises the question of truth. Yet whereas "The Origin of the Work of Art" takes its departure from the artwork, which, marked by the conflictual play of concealment and unconcealment, itself sets the truth to work in the work and thus offers an originary manifestation—exemplified by the Greek temple—of originary truth as the opening up of a historical world, "The Question Concerning Technology" concerns not so much truth's archaic past—its originating event—as its catastrophic future. The *Gestell*, as the essence of modern technology, is also a way in which truth happens. Yet it is a *mode* of truth that threatens the very possibility of truth itself as an opening to *Being* ("Question" 236).

It is in light of both these texts, brought into a constellation, that the significance of thinking of un-concealment as dis-encapsulation reveals itself. For if the above account of dis-encapsulation converges with Heidegger's thinking of originary truth as un-concealment, it also diverges from it in certain crucial respects. Dis-encapsulation presents the question of truth from the side of technology. Encapsulation and dis-encapsulation is a technical accomplishment, and in this sense even the biological organism, even inorganic entities such as the elementary molecules and particles themselves, may be regarded as techniques of dis-encapsulation. And this implies that, as for Heidegger, technology is not simply a human doing; it is a mode of truthing, a mode of disclosure ("Question" 222). Yet at the same time—and this is the decisive point—the technical mode of truthing, which Heidegger characterizes as the *Gestell*, also cannot be situated at the tail end of the history of Being, as the catastrophic danger of an almost absolute eclipse of the truth of Being. Not only does the technical, as Heidegger might himself admit, belong to the interplay of truth as un-concealment from the beginning, but the revelation of truth is in a way entirely technical, and therefore modern technology is not merely the danger showing the way to salvation but is itself salvation ("Question" 236-37). This is not to deny that there is still danger. But the danger consists not in the relentless hegemony of the technical, but in the refusal to concretely enter into the technical, to occupy, or indeed, ex-ist in *technē*. The danger consists in retreating from the technical into the phenomeno-

logical surface of ordinary experience, as if this surface were not itself an effect of technology.

What makes an engagement with technology impossible is, as already intimated, a refusal to recognize the concreteness of the thing—the concreteness of dis-encapsulation. Precisely because metaphysics encapsulates encapsulation, Being can itself emerge as the refuge for the very hiddenness that metaphysics banishes. In seeking to overcome metaphysics as a philosophy of presence, Heidegger draws on Being, which itself serves as the "standing reserve" of concealment. But in just this way, Being can only appear as that which has always already retreated, and continues to retreat, into silence, negativity, and mystery. The very concreteness of unconcealment remains unthinkable; at most we are given the artwork.

Scripting Philosophy

For the dominant metaphysical tradition, philosophy aspires to resemble as much as possible the divine mind contemplating its own perfection. Against this, it is necessary to insist that philosophy is a *technē*: it brings forth new possibilities of thinking—possibilities that are themselves always technical. It is an intervention in the technical apparatus of thought itself. This does not mean that philosophy should be replaced by programming, but rather that we should move beyond thinking of philosophy either as pure contemplation or as a representation of some sort of reality or as logical deduction and proof, and conceive of it instead as the technology of producing and interpreting a certain kind of writing. We might even think of the philosophical work, the philosophical text, as a kind of script. It is, just as in a computer program, not a representation of some sort of eternal reality; it is a set of symbols awaiting interpretation. This interpretation, of course, is not merely mechanical, but it is also not exactly hermeneutic. The concept of hermeneutics must always have recourse to a model of thinking as the immanent event of meaning. Rather, the interpretation in question consists in bringing into view, by at once encapsulating and dis-encapsulating, a certain concrete manner of encapsulation/dis-encapsulation.

The particular strength of this seemingly strange reconceptualization of philosophy is that it is, to use the jargon of computing, "backwards compatible": any philosophical text from the history of philosophy can be understood in this way. Moreover, it makes it possible to appreciate metaphysics without buying into its epistemic and methodological presuppositions. Even if we reject metaphysics as a path to representational knowledge of the true nature of things,

we can nevertheless appreciate it as a technology of thinking. This might seem paradoxical: If metaphysics is a technology of thinking, then isn't it a broken technology? And yet thinking as a technology is never anything but broken—it doesn't "correspond" to anything outside of itself. But, here, of course, the programming analogy also breaks down. If a program is broken, it either does not compile, or it crashes during "run time," or it fails to produce the desired result. The strange grandeur of human thought, and the basis for every technology of thinking, for every script, is that human thought never quite crashes, never fails—never dramatically at least. It may not lead to the expected or desired outcome, it may lead us hopelessly astray, or in circles—it may lead us nowhere—but interpretation always takes place. Thus, a certain mode of thinking can only be said to fail when it is measured against the expectation of a certain kind of success. The same cannot be said, however, of political life—or of thought applied to the problem of life. Political life and political thought have failed when either life as such or a certain quality (or kind) of life has become impossible for a community of beings that has come to understand itself as such.

If there is still need for a philosophical meta-critique, it can no longer consist in establishing the limits of human reason and human thought, but in differentiating between different kinds of philosophical scripts and establishing *how* they can be interpreted. This means above all else establishing the technologies of thinking that they prescribe. As a very tentative step in this direction, bringing this speculative essay to an even more speculative conclusion, I will propose a basic classification of philosophical scripts, using "philosophical" in the most general sense—a sense that, I believe, we would do well to recover.

1) Natural-scientific scripts, which show the thinking interface as derived from "encapsulated" functionality. The psychoanalytic theory of the unconscious, as well as the Marxist theory of substructure, may be seen as variants of the "natural-scientific script"; that these might seem to violate a certain norm of "positivist" empirical science, or that they are connected with therapeutic or revolutionary practice, changes nothing. Or, indeed, precisely because psychoanalytic therapy and revolutionary praxis offer a different mode of access to the "encapsulated" content than the technical manipulation of nature, they also allow for a different norm of scientific praxis.

2) Phenomenological scripts, which present the thinking interface as interface, in its immanently satisfied meaningfulness. Phenomenological scripts uncover a *surface* without depth, a world in which the meaning of everything presents

itself exhaustively. Thus, for example, Jean-Paul Sartre's account of *bad faith* compels the entire realm of the unconscious into the light of perspicacious motivations, of reason and will (Sartre 50-51). Mathematico-logical scripts, which disclose the properties of purely formal systems whose conceptualization is possible by the thinking interface, may be seen as a *formal* and *abstract* form of a phenomenological script.[19] Both psychoanalysis and Marxism can also be refashioned in phenomenological terms, such as happens, in the case of the latter, when the vulgar-materialist opposition of substructure and superstructure gives way to more subtle accounts of ideology.

3) Speculative scripts. These are scripts that may be seen as "extensions" of the phenomenological script rather than "hacks" into encapsulated functionalities. Speculative scripts *interpret* the phenomenological script in terms of a "higher"-order instance, such as God. The most common kinds of speculative scripts are *onto-theological*. Any theory of the ultimate *truth* that is sought through revolutionary, therapeutic, or scientific praxis involves a speculative extension of a phenomenological script.

To these we must add a fourth: metascripts, which concern the relation of different scripts to each other and the notion of "script" as such. This essay itself is an example of a metascript, and should be read as such. It is, in other words, self-interpreting.

While I have mentioned several kinds of scripts that have a *political* or *ethical* sense, I have not spoken of political or ethical scripts as such. Nor have I directly addressed the question of subjectivity. Is it a certain kind of script that serves to organize modes of ethical or political subjectivity? It might seem, indeed, as if to speak of "scripts," as we have, merely reframes the post-structuralist critique of subjectivity. Against this, however, I would argue that subjectivity, far from simply being dissolved into "code" or "scripturality," should in fact be understood as the point of confluence between different scripts; as an interface between interfaces. Subjectivity, this is to say, cannot be adequately reduced to a *single* kind of script—be it natural-scientific, phenomenological, or speculative. Rather, it is the point of contact among all three: the interface at which the *pure surface* of things as they show themselves to us touches the depths of *hidden mechanisms* and reaches into the heights of *truth* and *meaning*. Yet if subjectivity is not just

[19] It is not surprising, in this regard, that both Husserl's phenomenology and Frege's logicism take their departure from the rejection of psychologism.

one kind of script, it is still a script rather than a simple given: it is the scripting of the interface of scripts. Or, in other words, a metascript. The metascripts that script subjectivity, moreover, have both an ethical and a political sense; ultimately, they constitute the possibility of community most broadly understood. It is in this way, moreover, that thinking, already a *technē*, passes over into praxis.

A metanarrative, in the sense employed by Jean-François Lyotard in *The Postmodern Condition*, is a kind of metascript, but not every metascript is a metanarrative (*Postmodern Condition* xxiv). A metanarrative sutures the three different levels of script into a fully "totalitarian" system. Yet while a metascript *may* refuse such totalization, it is not necessary to conceive of the metascript as primarily seeking, in the words of Lyotard, to "bear witness to the differend" (*Differend* xiii). Accordingly, the "stakes" of regarding philosophy as a question of "scripts" rather than "phrases" is something else than the rejection of "humanism," and, likewise, a thoroughgoing rejection of "subjectivity" is not called for. Philosophy need not collapse into the abyss of merely registering difference; it can work toward a concrete, substantive, if fragile, sense of the world.

Appendix: A Very Brief Introduction to Programming Paradigms

Suppose we are faced with the simple arithmetic task: multiplying together two numbers. Most children are taught this as an algorithmic procedure whose underlying logic remains mostly obscure. If they had a competent teacher, they will learn not only a mechanical sequence of actions but an intuitive geometric model to help them understand what they are actually doing when they carry out the algorithm. And if they go on to study advanced mathematics, they will begin to understand even this simple operation in subtler and more abstract ways: in terms of the set-theoretical construct of natural numbers or as a generalized operation with an inverse and an identity operation. Even so, they will continue to avail themselves of the algorithm throughout their lives, using it whenever faced with the task of multiplying two sufficiently large numbers together by hand. In this they are acting not so differently than a computer: their capacity to produce the correct result depends on a mechanical procedure rather than a genuine understanding of the meaning of the symbols.

This example can help conceptualize different programming paradigms. Each paradigm consists in a different way of thinking about what the computer is doing, indeed what a computer is. The result of the operation and its mathematical meaning remain the same; what changes, however, is the way in which the

computation, as well as the relation we have to it as mediated by the computer, is conceptualized.

The Imperative Paradigm

Most programmers will seldom have to worry about multiplying two numbers together. In higher-level computer languages, multiplication almost always involves a simple function call (often conveyed through standard mathematical notation) interpreted or compiled into instructions carried out by the processor, which, after converting the entered values from decimal to binary representation, follows a series of steps to yield the correct result. Nowadays, these simple arithmetic operations are highly optimized, performed by special numerical processors, with memory tables employed for low values—the equivalent of a child memorizing a multiplication table. Let us imagine, though, that we, for whatever reason, wish to write a program that actually "spells out" the sequence of steps needed to yield the result. And let us further imagine that, rather than drawing on the native arithmetic functions offered by nearly all actual programming languages, we will utilize only a very small repertoire of simple operations: incrementing a number by one, decreasing a number by one, and reporting whether a given number is equal to zero. Such a program will, no doubt, be completely useless: incredibly slow for all but the most trivial calculations. But it is a useful exercise in breaking down a complex task into its simplest components.

The primary operations we will allow are as follows:

Add1(x), which returns the value x incremented by 1
Sub1(x), which returns the value x reduced by 1
Is0 (x), which returns TRUE if x is equal to 0, and FALSE if it is not.

We further assume the existence of a few variables, w, x, y, and z, that can take as their value integers greater than equal to 0, and basic *INPUT* and *OUTPUT* routines that, respectively, fetch a positive integer from the user and display a positive integer. We might even think of this according to a simple physical representation: there are four empty bowls with the letters w, x, y, and z written on them; there is an unlimited supply of black beads; and there is a "operator" capable of the following operations: placing a single bead in one bowl (Add1); taking a single bead from one bowl (Sub1); testing to see if a bowl is empty (Is0); filling a bowl up with all the beads placed in a special input bin (INPUT); and pouring all the beads from one of the bowls into a special output bin (OUTPUT).

Our first program, written following an imperative paradigm, will consist in a series of commands given to the computer to execute:a kind of recipe. Each individual command has a unique number assigned to it. The computer reads these commands starting from the first, either advancing to the next line in the sequence or deciding, based on the evaluation of a Boolean expression, to leap to another place in the program. (# is used for comments. The program itself passes over these.)

```
# An imperative program for multiplying two positive integers
1: x = INPUT
2: y = INPUT
3: z = 0                        # for the result of the calculation
4: if Is0 (x) go to 11          # if x = 0 then terminate
5: x = Sub1(x)                  # main loop begins; z = x-1
6: w = y                        # sub loop begins; reset w to y
7: w = Sub1(w)                  # w = w-1
8: z = Add1(z)                  # z = z+1
9: if not Is0(w) go to 7        # if w > 0, repeat sub loop
10: go to 4                     # repeat main loop
11: OUTPUT z
```

The imperative paradigm thus expresses a very elemental insight into what a computer is: an infinitely docile, mindless servant that carries out a sequence of instructions with perfect accuracy. This is usually how children are taught to code.

The Procedural Paradigm

The problem with the above program, however, is that it lacks an evident structure. It is just a sequence of steps. To understand what the algorithm is actually doing, we have to look very closely at each of these steps. There is little in the structure of the program—in the way in which it is written out—to help us. For a simple program, this is fine, but as the program grows in size, it becomes less and less legible and more and more difficult to manage. The second paradigm, called *procedural programming*, tries to remedy this by analyzing the program into discrete components, that is, distinct procedures that are carried out. This is an abstraction from the simplest "imperative" model. The code lines 6-9 are now understood as a subroutine that consists in adding a positive integer to z.

```
# A procedural program multiplying two positive integers
SUBROUTINE AddToZ:
    w = y
    while not Is0(w):
        z = Add1(z)
        w = Sub1(w)
PROGRAM:
    x = INPUT
    y = INPUT
    z = 0
    while not Is0(x):
        x = Sub1(x)
        AddToZ
    OUTPUT z
```


The Functional Paradigm

While the above program is more elegant than the first—it is certainly easier to figure out what it is doing from looking at it—it remains problematic. The above "procedure" is entangled with the main program and is only comprehensible in terms of it, since the variables that it uses (the bead-filled bowls) are only defined within the domain of the main program. Abstracted from this context, the procedure becomes incomprehensible.

It is also possible, however, to think of procedures as functions in a strictly mathematical sense. In this case, the output is strictly determined by the input; we could even understand the function as a mapping between one list of values and another. Unlike a procedure, a function cannot have side-effects, and it cannot invoke variables that are not defined within its domain. We are now, indeed, forced to add to and rename our bowls, since each of the two functions must have its own special bowls.[20]

[20] Strictly speaking, functional programming is a form of declarative programming (see below), since in a pure functional language the program consists solely of declarations or expressions rather than statements. The example I gave above is misleading, in this sense, as it employs functional constructs within a more or less imperative idiom. It is nevertheless helpful for our purposes, which is to treat these different paradigms from the perspective of the interface, rather than from a purely formal or technical perspective, in which case very different considerations would come into play.

```
# A functional program multiplying two integers
FUNCTION AddXY (x,y):
    if not Is0(x):
        x = Sub1(x)
        y = Add1(y)
        x,y = AddXY (x,y)          # recursive function call
        return x,y
    else:
        return x,y
FUNCTION MultXY (x,y,z):
    if not Is0(x)
        x = Sub1(x)
        z,z = AddXY(y,z)
        x,y,z = MultXY (x,y,z)     # recursive function call
        return x,y,z
    else:
        return x,y,z
PROGRAM:
    x = INPUT
    y = INPUT
    z = 0
    x,y,z = MulyXY (x,y,z)
    OUTPUT z
```

The Object-Oriented Paradigm

Let us now conceive of the entire computation as involving objects that interact with each other individually according to certain rules to yield a result.

This makes most sense when one thinks of numbers in terms of their binary representations:

e.g. $4_{10} = 0100_2$
eg. $3_{10} = 0011_2$

$3_{10}+4_{10} = 0100_2+0011_2 = 0111_2$

We now, indeed, have a rather different physical apparatus. We can get rid of our bowls, and replace them with a large but finite mat with a grid of lines, with each square only large enough for a single bead. The board is set up in the following

way. With the rows and columns of the mat labeled from 0 to N starting from the upper right corner, lay out beads for the binary representation of the first number on top of the grid, going from right to left, and for the second number on the right of the grid, going from top to bottom. This is equivalent to "inputting" values into a program. Then, starting from the top-right, move down the side one square at a time. Whenever you find a bead on the side, lay out the same binary representation found on top of the current row, but starting from a position that is as far to the left of the rightmost column as the present row is below the topmost row. (The starting point, in other words, is determined by a diagonal drawn from the top-right corner.)

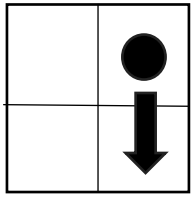
Once the multiplication has been "set up" in the right way, then the operation itself proceeds by having the beads on the grid interact with each other according to the following rules:

(1) If the square below is free, move down one square.

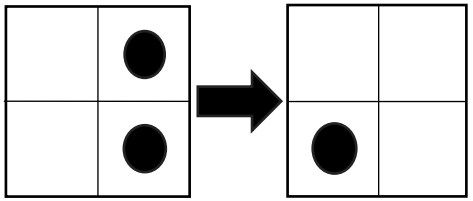
(2) If the square below is occupied, move to the closest available position to the left, and send an instruction to the cell below to destroy itself.

Assuming that the field is of sufficiently large yet finite dimensions, then, when no moves are possible, the result can be read off at the bottom of the field.

This is, of course, an abstruse way to perform a simple mathematical operation, though it is in fact algorithmically efficient and indeed similar to the method that is actually implemented in circuitry. The point is merely to give a vivid example of the perspectival shift implied in object-oriented programing. Although, of course, the operation still has to be set up and the result retrieved from the top down, the actual calculation proceeds not through the centrally orchestrated execution of a series of steps, but through the transformation and interaction of the individual "objects" into which the operation has been dissolved. These "objects" change their internal state (the position which they occupy), send messages to each other (for example, "die!"), and are also capable, on receiving such a message, of self-destruction—removing themselves from the board.

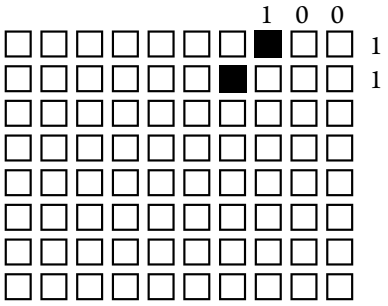


RULE # 1

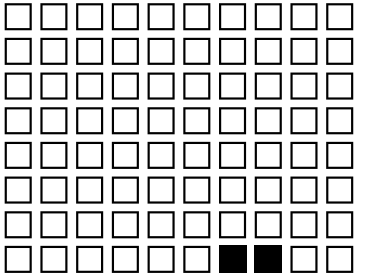


RULE #2

EXAMPLE 1: $100_2 * 11_2 = 1100_2$



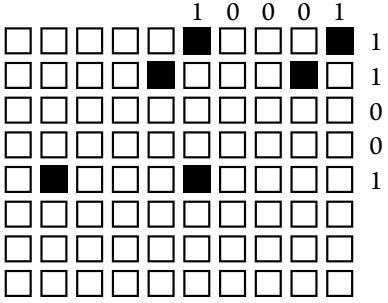
INITIAL SETUP



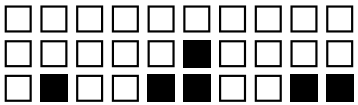
FINAL STATE

=> 1100_2 (12_{10})

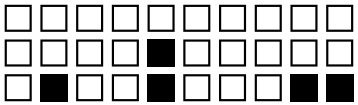
EXAMPLE 2: $10001_2 * 10011_2 = 101000011_2$



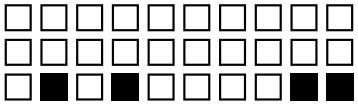
INITIAL SETUP



APPLYING RULE #1 REPEATEDLY



APPLYING RULE #2



APPLYING RULE #2

=>101000011_2

The Declarative Paradigm

In all of the above cases, we are thinking of the program in terms of the details of its execution. But, as already seen, the result of a program is independent of these details to the extent that the same results can be achieved through many different means. Suppose, then, that we abstract away from all of these details. If a program is understood as a script that *determines* a certain relation between input and output, then the very statement of the problem (say, what is 10*20?) may itself be understood as a kind of programming. This is the case, for example, with a simple desk-top calculator: we "program" it simply by *declaring* the problem that we want solved. But it is also the case with many very sophisticated languages, such as SQL (for querying databases) and Prolog.

WORKS CITED

Chalmers, David J. "Facing up to the Problem of Consciousness." *Journal of Consciousness Studies* 2.3 (1995): 200-19.

Copeland, B. Jack. "Computable Numbers: A Guide." *The Essential Turing: The Ideas That Gave Birth to the Computer Age*. Ed. B. Jack Copeland. Oxford: Oxford UP, 2004.

Denning, Peter J., and Craig H. Martell. *Great Principles of Computing*. Cambridge, MA: MIT P, 2015.

Dou, Eva, and Olivia Geng. "Humans Mourn Loss after Google Is Unmasked as China's Go Master." *Wall Street Journal* 5 Jan. 2017. Web. Accessed 3 Mar. 2020.

Evens, Aden. "Object-Oriented Ontology, or Programming's Creative Fold." *Angelaki: Journal of Theoretical Humanities* 11.1 (2006): 89-97.

Galloway, Alexander R. *The Interface Effect*. Cambridge, UK: Polity, 2012.

Gerrish, Sean. *How Smart Machines Think*. Cambridge, MA: MIT P, 2018.

Hayles, N. Katherine. *My Mother Was a Computer: Digital Subjects and Literary Texts*. Chicago: U of Chicago P, 2010.

Heidegger, Martin. "The Origin of the Work of Art." *Basic Writings*. Routledge Classics Edition. Ed. David Farrell Krell. New York: Routledge, 2011. 89-139

---. "The Question Concerning Technology." *Basic Writings*. Routledge Classics Edition. Ed. David Farrell Krell. New York: Routledge, 2011. 217-38.

Kay, Alan C. "The Early History of Smalltalk." *History of Programming Languages II.* New York: Association for Computing Machinery, 1996. 511-98.

Kay, Alan, and Adele Goldberg. "Personal Dynamic Media." *Computer 3* (1977): 31-41.

Lyotard, Jean-François. *The Differend: Phrases in Dispute*. Trans. Georges Van Den Abbeele. Minneapolis: U of Minnesota P, 1988.

---. *The Postmodern Condition: A Report on Knowledge*. Trans. Geoff Bennington and Brian Massumi. Manchester: U of Manchester P, 1984.

Mumford, Lewis. *The Myth of the Machine: Technics and Human Development*. New York: Harcourt Brace Jovanovich, 1967.

von Neumann, John. *The Computer and the Brain*. New Haven: Yale UP, 1958.

Plato. *Laws*. Trans. Thomas L. Pangle. Chicago: U of Chicago P, 1988.

Sartre, Jean-Paul. *Being and Nothingness: A Phenomenological Essay on Ontology*. Trans. Hazel E. Barnes. New York: Pocket Books, 1966.

Searle, John R. "Is the Brain a Digital Computer?" *Proceedings and Addresses of the American Philosophical Association* 64.3 (1990): 21-37.

Steiglitz, Ken. *The Discrete Charm of the Machine: Why the World Became Digital*. Princeton: Princeton UP, 2019.

Van der Spiegel, Jan, James F. Tau, Titiimaea F. Ala'ilima, and Lin Ping Ang. "The Eniac: History, Operation, and Reconstruction in VLSI." *The First Computers: History and Architecture*. Ed. Raúl Rojas and Ulf Hasgagen. Cambridge, MA: MIT P, 2000. 121-78.

West, David. *Object Thinking*. Redmond WA: Microsoft Press, 2004.

Wiener, Norbert. *The Human Use of Human Beings: Cybernetics and Society*. London: Free Association Books, 1989.

Wittgenstein, Ludwig. *Philosophical Investigations*. 3rd ed. Trans. G. E. M. Anscombe. Oxford: Blackwell, 2001.

***Manuscript received 10 Mar. 2020, accepted for publication 30 Nov. 2020*