CODING AS A PLAY-GROUND

Graphic Design ; Post-Digital Age ; Learn to Code vs. Code to Learn: Creative Coding Beyond the Economic Imperative

Essay by Silvio Lorusso Demian Conrad Onomatopee, 2022

LEARN TO CODE

In 2021, the economic imperative to train and retrain has never been so strong. After the pandemic's dramatic impact on artists' economies, a skepticism about "creative" work is emerging, portraying it as unproductive daydreaming, and a wholly unessential industry. The emphasis is now on hard labor and effectiveness. The fundamental idea of Learn to Code is that the ability to program is a historical necessity for people working at a useless or obsolete job, and that these people must serve the economic imperatives of capitalism. This servitude is referred to as "retraining". Perhaps it is useful at this point to briefly reiterate the distinction between coders and programmers. While programmers are recognized as having an acknowledged and relatively arcane expertise with a correspondingly high salary, coding is increasingly perceived as semi-skilled labor. The programmer belongs to a profession, the coder to a workforce.

Back to design. Currently, public recognition of graphic designers is not so different from that of journalists. Both are now perceived as entire professions that it would be good to automate once and for all. Jobs meant to become buttons. According to this scenario, coding emerges as a professional panacea linked to the rhetoric of skill obsolescence and employability. Coding becomes a skill, in the most reductive sense of the term: something to add to your CV, better if exploded into discrete units. HTML and CSS: good, JavaScript: basic.

The book Graphic Design Surveyed shows that US and UK students consider coding the third most useful skill to acquire (after networking and idea generation).

In 2014, German media theorist Florian Cramer dissected the various meanings of the term "post-digital". One of them was "the contemporary disenchantment with digital information systems and media gadgets." The Learn to Code meme suggests that disenchantment does not only revolve around the tools of the trade, but also around the trade itself. Coding, in the light of retraining, doesn't seem to be so emancipating.

Another understanding of post-digital Cramer highlighted has to do with the revival of old media. This might be a bit of a stretch, but what is more "old media", more 20th-century, than the idea of a workforce to be forged for the good of the nation? Of course, the Learn to Code narrative hints at the fact that jobs, skills and aspirations do not exist in a vacuum. However, due to a combination of disenchantment with programming and old-media labor rhetoric, coding emerges as a post-digital manifestation of capitalist realism, forcing graphic designers, journalists and coal miners alike to deal with their situations. All of them must go through mandatory updates, just like software. Is programming itself immune to this logic? Not really, it would seem, as the angelus novus of AI promises or threatens (depending on whom you ask) to automate the coder as well.

CODE TO LEARN

Ted Nelson once said: "The computer is as inhuman as we make it."

The fact that coding is a cognitive activity does not make it intrinsically humane. It can also become a tedious, repetitive, industrialized exercise. This is what the jobs of a "new collar" workforce might look like, especially if such a workforce were a coercively retrained one. What then is coding beyond the not very empowering economic imperative of "learning to code"? Coding—whether it consists of writing HTML and CSS, posting a Javascript snippet on Github Gist, tweaking a Processing sketch, or publishing a Python module—can be not merely the content of a learning process, but its very medium.

Thus, it can be a craft and a culture, or even better, a cultural meeting point. Who would we find there? A community of practice.

LEARNING WITH COMPUTERS

In a time when much is being said about the creativity of autonomous AI-powered machines, it is good to reconsider Licklider's notion of human-computer symbiosis. When you code, you instruct the computer to execute a more or less complex task, which is then immediately performed. You do not always know what to expect: the result might awe or disappoint you, allowing you to reorient or even redefine your initial goal. Part of the symbiosis is intrinsic to the language shared by user and machine—namely, code. Creativity unfolds through this micro-iterative learning process: it is neither in the mind nor in the machine, but rather in the continuous scripted dialogue between the subject and their extensions. The computer is just one of those extensions, but a particularly powerful one, since it is, to use Alan Kay's term, a metamedium, that is, a medium capable of simulating all others.

As such the computer should be a thing that can be shaped and transformed. When it becomes less malleable, the computer is fixed within a stable media, which is perhaps more efficient, but also less surprising, less "creative." You do more but you learn less. Fundamentally, creativity is a question of time. Mostly of our daily activity with computers happens through hopefully speedy but ossified software. We use the computer in "speedrun mode." This is the paradox of creative coding: the coding part is supposed to make things faster, the creative part requires that things go slowly. According to permacomputing principles, one might say that Learn to Code is very "yang", and Code to Learn does also value the "yin": it "accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation."

LEARNING THROUGH COMPUTERS

Coding does not just manifest as a relationship between a user and a computer, but also between users through computers. Users exchanging techniques in real life or on Stack Overflow, appreciating each others' solutions, using coding as an excuse to just hang out, or building upon each other's tools.

The input of this process is patience and a capacity for listening; the output is fun and a sense of belonging. Coding can also be a bridge linking us to past users.

We see this in Ted Davis' assignment to recreate pioneering computer works or with the Re-Programmed Art Project, where a series of contemporary designers reinterpreted "analog" works of the Italian collective Gruppo T, active in the 1960s.

CODING AS A CRAFT

While Learn to Code turns coding into a resumé-ready skill, Code to Learn is about coding as a craft. My understanding of craft is wide-ranging: "a good job well done," as Sennett defines it. A craft is a savoir faire that is capable of stabilizing and consolidating one's identity.

In a time when designers are urged to constantly decorate their bio with strategic labels, a craft is a reflective activity, in the sense that the crafted things and the tools for crafting are a reflection of their maker, who generally recognizes themselves in them. This is also true of coding. As Roberto Arista, creator of the Python for Designers course, puts it: Programming then can become a way to escape [the confinement of desktop publishing software], connecting different regions and patiently rebuilding the workshop within the tools that effectively destroyed these regions.

The craftsperson enters their own physical or digital workshop—a local hackerspace, a custom i3 setup, a DIY CMS—and feels at home. This is where they code and learn, learn and code.

This is where they can forget, for a while at least, if they are lucky, the pressures and economic necessities of daily life.

Without neglecting Learn to Code's stressful refrain of employability and professional obsolescence, Code to Learn helps by considering the coding activity in itself, and not merely as an inevitable destiny. Opting for one model over the other in a graphic design school also means determining how to teach programming. When I speak of coding in and of itself I do not mean it as an index of technical notions (variables, loops, etc.). That is precisely the instrumental reduction of the Learn to Code attitude. Rather, I mean it in a broader sense: coding as a social activity and a cultural domain. This is what Code to Learn is all about.

CODING AS A PLAY-GROUND

Graphic Design ; Post-Digital Age ; Learn to Code vs. Code to

Learn: Creative Coding Beyond the Economic Imperative

Essay by Silvio Lorusso Demian Conrad Onomatopee, 2022

LEARN TO CODE

In 2021, the economic imperative to train and retrain has never been so strong. After the pandemic's dramatic impact on artists' economies, a skepticism about "creative" work is emerging, portraying it as unproductive daydreaming, and a wholly unessential industry. The emphasis is now on hard labor and effectiveness. The fundamental idea of Learn to Code is that the ability to program is a historical necessity for people working at a useless or obsolete job, and that these people must serve the economic imperatives of capitalism. This servitude is referred to as "retraining". Perhaps it is useful at this point to briefly reiterate the distinction between coders and programmers. While programmers are recognized as having an acknowledged and relatively arcane expertise with a correspondingly high salary, coding is increasingly perceived as semi-skilled labor. The programmer belongs to a profession, the coder to a workforce.

Back to design. Currently, public recognition of graphic designers is not so different from that of journalists. Both are now perceived as entire professions that it would be good to automate once and for all. Jobs meant to become buttons. According to this scenario, coding emerges as a professional panacea linked to the rhetoric of skill obsolescence and employability. Coding becomes a skill, in the most reductive sense of the term: something to add to your CV, better if exploded into discrete units. HTML and CSS: good, JavaScript: basic.

The book Graphic Design Surveyed shows that US and UK students consider coding the third most useful skill to acquire (after networking and idea generation).

In 2014, German media theorist Florian Cramer dissected the various meanings of the term "post-digital". One of them was "the contemporary disenchantment with digital information systems and media gadgets." The Learn to Code meme suggests that disenchantment does not only revolve around the tools of the trade, but also around the trade itself. Coding, in the light of retraining, doesn't seem to be so emancipating.

Another understanding of post-digital Cramer highlighted has to do with the revival of old media. This might be a bit of a stretch, but what is more "old media", more 20th-century, than the idea of a workforce to be forged for the good of the nation? Of course, the Learn to Code narrative hints at the fact that jobs, skills and aspirations do not exist in a vacuum. However, due to a combination of disenchantment with programming and old-media labor rhetoric, coding emerges as a post-digital manifestation of capitalist realism, forcing graphic designers, journalists and coal miners alike to deal with their situations. All of them must go through mandatory updates, just like software. Is programming itself immune to this logic?

Not really, it would seem, as the angelus novus of AI promises or threatens (depending on whom you ask) to automate the coder as well.

CODE TO LEARN

Ted Nelson once said: "The computer is as inhuman as we make it."

The fact that coding is a cognitive activity does not make it intrinsically humane. It can also become a tedious, repetitive, industrialized exercise. This is what the jobs of a "new collar" workforce might look like, especially if such a workforce were a coercively retrained one. What then is coding beyond the not very empowering economic imperative of "learning to code"? Coding—whether it consists of writing HTML and CSS, posting a Javascript snippet on Github Gist, tweaking a Processing sketch, or publishing a Python module—can be not merely the content of a learning process, but its very medium.

Thus, it can be a craft and a culture, or even better, a cultural meeting point. Who would we find there? A community of practice.

LEARNING WITH COMPUTERS

In a time when much is being said about the creativity of autonomous AI-powered machines, it is good to reconsider Licklider's notion of human-computer symbiosis. When you code, you instruct the computer to execute a more or less complex task, which is then immediately performed. You do not always know what to expect: the result might awe or disappoint you, allowing you to reorient or even redefine your initial goal. Part of the symbiosis is intrinsic to the language shared by user and machine—namely, code.

Creativity unfolds through this micro-iterative learning process: it is neither in the mind nor in the machine, but rather in the continuous scripted dialogue between the subject and their extensions. The computer is just one of those extensions, but a particularly powerful one, since it is, to use Alan Kay's term, a metamedium, that is, a medium capable of simulating all others. As such the computer should be a thing that can be shaped and transformed. When it becomes less malleable, the computer is fixed within a stable media, which is perhaps more efficient, but also less surprising, less "creative." You do more but you learn less. Fundamentally, creativity is a question of time. Mostly of our daily activity with computers happens through hopefully speedy but ossified software. We use the computer in "speedrun mode." This is the paradox of creative coding: the coding part is supposed to make things faster, the creative part requires that things go slowly. According to permacomputing principles, one might say that Learn to Code is very "yang", and Code to Learn does also value the "yin": it "accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation."

LEARNING THROUGH COMPUTERS

Coding does not just manifest as a relationship between a user and a computer, but also between users through computers. Users exchanging techniques in real life or on Stack Overflow, appreciating each others' solutions, using coding as an excuse to just hang out, or building upon each other's tools.

The input of this process is patience and a capacity for listening; the output is fun and a sense of belonging. Coding can also be a bridge linking us to past users.

We see this in Ted Davis' assignment to recreate pioneering computer works or with the Re-Programmed Art Project, where a series of contemporary designers reinterpreted "analog" works of the Italian collective Gruppo T, active in the 1960s.

CODING AS A CRAFT

While Learn to Code turns coding into a resumé-ready skill, Code to Learn is about coding as a craft. My understanding of craft is wide-ranging: "a good job well done," as Sennett defines it. A craft is a savoir faire that is capable of stabilizing and consolidating one's identity.

In a time when designers are urged to constantly decorate their bio with strategic labels, a craft is a reflective activity, in the sense that the crafted things and the tools for crafting are a reflection of their maker, who generally recognizes themselves in them. This is also true of coding. As Roberto Arista, creator of the Python for Designers course, puts it: Programming then can become a way to escape [the confinement of desktop publishing software], connecting different regions and patiently rebuilding the workshop within the tools that effectively destroyed these regions.

The craftsperson enters their own physical or digital workshop—a local hackerspace, a custom i3 setup, a DIY CMS—and feels at home. This is where they code and learn, learn and code. This is where they can forget, for a while at least, if they are lucky, the pressures and economic necessities of daily life.

Without neglecting Learn to Code's stressful refrain of employability and professional obsolescence, Code to Learn helps by considering the coding activity in itself, and not merely as an inevitable destiny. Opting for one model over the other in a graphic design school also means determining how to teach programming. When I speak of coding in and of itself I do not mean it as an index of technical notions (variables, loops, etc.). That is precisely the instrumental reduction of the Learn to Code attitude. Rather, I mean it in a broader sense: coding as a social activity and a cultural domain. This is what Code to Learn is all about.

CODING AS A PLAY-GROUND Graphic Design ; Post-Digital Age ; Learn to Code vs. Code to Learn: Creative Coding Beyond the Economic Imperative

Essay by Silvio Lorusso
Demian Conrad Ono-matopee, 2022

LEARN TO CODE

In 2021, the economic imperative to train and retrain has never been so strong. After the pandemic's dramatic impact on artists' economies, a skepticism about "creative" work is emerging, portraying it as unproductive daydreaming, and a wholly unessential industry. The emphasis is now on hard labor and effectiveness. The fundamental idea of Learn to Code is that the ability to program is a historical necessity for people working at a useless or obsolete job, and that these people must serve the economic imperatives of capitalism. This servitude is referred to as "retraining". Perhaps it is useful at this point to briefly reiterate the distinction between coders and programmers. While programmers are recognized as having an acknowledged and relatively arcane expertise with a correspondingly high salary, coding is increasingly perceived as semi-skilled labor. The programmer belongs to a profession, the coder to a workforce.

Back to design. Currently, public recognition of graphic designers is not so different from that of journalists. Both are now perceived as entire professions that it would be good to automate once and for all. Jobs meant to become buttons. According to this scenario, coding emerges as a professional panacea linked to the rhetoric of skill obsolescence and employability. Coding becomes a skill, in the most reductive sense of the term: something to add to your CV, better if exploded into discrete units. HTML and CSS: good, JavaScript: basic.

The book Graphic Design Surveyed shows that US and UK students consider coding the third most useful skill to acquire (after networking and idea generation).

In 2014, German media theorist Florian Cramer dissected the various meanings of the term "post-digital". One of them was "the contemporary disenchantment with digital information systems and media gadgets." The Learn to Code meme suggests that disenchantment does not only revolve around the tools of the trade, but also around the trade itself. Coding, in the light of retraining, doesn't seem to be so emancipating.

Another understanding of post-digital Cramer highlighted has to do with the revival of old media. This might be a bit of a stretch, but what is more "old media", more 20th-century, than the idea of a workforce to be forged for the good of the nation? Of course, the Learn to Code narrative hints at the fact that jobs, skills and aspirations do not exist in a vacuum. However, due to a combination of disenchantment with programming and old-media labor rhetoric, coding emerges as a post-digital manifestation of capitalist realism, forcing graphic designers, journalists and coal miners alike to deal with their situations. All of them must go through mandatory updates, just like software. Is programming itself immune to this logic? Not really, it would seem, as the angelus novus of AI promises or threatens (depending on whom you ask) to automate the coder as well.

CODE TO LEARN

Ted Nelson once said: "The computer is as inhuman as we make it."

The fact that coding is a cognitive activity does not make it intrinsically humane. It can also become a tedious, repetitive, industrialized exercise. This is what the jobs of a "new collar" workforce might look like, especially if such a workforce were a coercively retrained one. What then is coding beyond the not very empowering economic imperative of "learning to code"?

Coding—whether it consists of writing HTML and CSS, posting a Javascript snippet on Github Gist, tweaking a Processing sketch, or publishing a Python module—can be not merely the content of a learning process, but its very medium.

Thus, it can be a craft and a culture, or even better, a cultural meeting point. Who would we find there? A community of practice.

LEARNING WITH COMPUTERS

In a time when much is being said about the creativity of autonomous AI-powered machines, it is good to reconsider Licklider's notion of human-computer symbiosis. When you code, you instruct the computer to execute a more or less complex task, which is then immediately performed. You do not always know what to expect: the result might awe or disappoint you, allowing you to reorient or even redefine your initial goal. Part of the symbiosis is intrinsic to the language shared by user and machine—namely, code.

Creativity unfolds through this micro-iterative learning process: it is neither in the mind nor in the machine, but rather in the continuous scripted dialogue between the subject and their extensions. The computer is just one of those extensions, but a particularly powerful one, since it is, to use Alan Kay's term, a metamedium, that is, a medium capable of simulating all others. As such the computer should be a thing that can be shaped and transformed. When it becomes less malleable, the computer is fixed within a stable media, which is perhaps more efficient, but also less surprising, less "creative." You do more but you learn less. Fundamentally, creativity is a question of time. Mostly of our daily activity with computers happens through hopefully speedy but ossified software. We use the computer in "speedrun mode." This is the paradox of creative coding: the coding part is supposed to make things faster, the creative part requires that things go slowly. According to permacomputing principles, one might say that Learn to Code is very "yang", and Code to Learn does also value the "yin": it "accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation."

LEARNING THROUGH COMPUTERS

Coding does not just manifest as a relationship between a user and a computer, but also between users through computers. Users exchanging techniques in real life or on Stack Overflow, appreciating each others' solutions, using coding as an excuse to just hang out, or building upon each other's tools.

The input of this process is patience and a capacity for listening; the output is fun and a sense of belonging. Coding can also be a bridge linking us to past users.

We see this in Ted Davis' assignment to recreate pioneering computer works or with the Re-Programmed Art Project, where a series of contemporary designers reinterpreted "analog" works of the Italian collective Gruppo T, active in the 1960s.

CODING AS A CRAFT

While Learn to Code turns coding into a resumé-ready skill, Code to Learn is about coding as a craft. My understanding of craft is wide-ranging: "a good job well done," as Sennett defines it. A craft is a savoir faire that is capable of stabilizing and consolidating one's identity.

In a time when designers are urged to constantly decorate their bio with strategic labels, a craft is a reflective activity, in the sense that the crafted things and the tools for crafting are a reflection of their maker, who generally recognizes themselves in them. This is also true of coding. As Roberto Arista, creator of the Python for Designers course, puts it: Programming then can become a way to escape [the confinement of desktop publishing software], connecting different regions and patiently rebuilding the workshop within the tools that effectively destroyed these regions.

The craftsperson enters their own physical or digital workshop—a local hackerspace, a custom i3 setup, a DIY CMS—and feels at home. This is where they code and learn, learn and code. This is where they can forget, for a while at least, if they are lucky, the pressures and economic necessities of daily life. Without neglecting Learn to Code's stressful refrain of employability and professional obsolescence, Code to Learn helps by considering the coding activity in itself, and not merely as an inevitable destiny. Opting for one model over the other in a graphic design school also means determining how to teach programming. When I speak of coding in and of itself I do not mean it as an index of technical notions (variables, loops, etc.). That is precisely the instrumental reduction of the Learn to Code attitude. Rather, I mean it in a broader sense: coding as a social activity and a cultural domain. This is what Code to Learn is all about.

CODING AS A PLAYGROUND
Graphic Design ; Post-Digital Age ; Learn to Code vs. Code to Learn: Creative Coding Beyond the Economic Imperative

Essay by Silvio Lorusso Demian Conrad Onomatopee, 2022

LEARN TO CODE

In 2021, the economic imperative to train and retrain has never been so strong. After the pandemic's dramatic impact on artists' economies, a skepticism about "creative" work is emerging, portraying it as unproductive daydreaming, and a wholly unessential industry. The emphasis is now on hard labor and effectiveness. The fundamental idea of Learn to Code is that the ability to program is a historical necessity for people working at a useless or obsolete job, and that these people must serve the economic imperatives of capitalism. This servitude is referred to as "retraining". Perhaps it is useful at this point to briefly reiterate the distinction between coders and programmers. While programmers are recognized as having an acknowledged and relatively arcane expertise with a correspondingly high salary, coding is increasingly perceived as semi-skilled labor. The programmer belongs to a profession, the coder to a workforce. Back to design. Currently, public recognition of graphic designers is not so different from that of journalists. Both are now perceived as entire professions that it would be good to automate once and for all. Jobs meant to become buttons. According to this scenario, coding emerges as a professional panacea linked to the rhetoric of skill obsolescence and employability. Coding becomes a skill, in the most reductive sense of the term: something to add to your CV, better if exploded into discrete units. HTML and CSS: good, JavaScript: basic.

The book Graphic Design Surveyed shows that US and UK students consider coding the third most useful skill to acquire (after networking and idea generation).

In 2014, German media theorist Florian Cramer dissected the various meanings of the term "post-digital". One of them was "the contemporary disenchantment with digital information systems and media gadgets." The Learn to Code meme suggests that disenchantment does not only revolve around the tools of the trade, but also around the trade itself. Coding, in the light of retraining, doesn't seem to be so emancipating.

Another understanding of post-digital Cramer highlighted has to do with the revival of old media. This might be a bit of a stretch, but what is more "old media", more 20th-century, than the idea of a workforce to be forged for the good of the nation? Of course, the Learn to Code narrative hints at the fact that jobs, skills and aspirations do not exist in a vacuum. However, due to a combination of disenchantment with programming and old-media labor rhetoric, coding emerges as a post-digital manifestation of capitalist realism, forcing graphic designers, journalists and coal miners alike to deal with their situations. All of them must go through mandatory updates, just like software. Is programming itself immune to this logic? Not really, it would seem, as the angelus novus of AI promises or threatens (depending on whom you ask) to automate the coder as well.

CODE TO LEARN

Ted Nelson once said: "The computer is as inhuman as we make it." The fact that coding is a cognitive activity does not make it intrinsically humane. It can also become a tedious, repetitive, industrialized exercise. This is what the jobs of a "new collar" workforce might look like, especially if such a workforce were a coercively retrained one. What then is coding beyond the not very empowering economic imperative of "learning to code"? Coding—whether it consists of writing HTML and CSS, posting a Javascript snippet on Github Gist, tweaking a Processing sketch, or publishing a Python module—can be not merely the content of a learning process, but its very medium. Thus, it can be a craft and a culture, or even better, a cultural meeting point. Who would we find there? A community of practice.

LEARNING WITH COMPUTERS

In a time when much is being said about the creativity of autonomous AI-powered machines, it is good to reconsider Licklider's notion of human-computer symbiosis. When you code, you instruct the computer to execute a more or less complex task, which is then immediately performed. You do not always know what to expect: the result might awe or disappoint you, allowing you to reorient or even redefine your initial goal. Part of the symbiosis is intrinsic to the language shared by user and machine—namely, code.

Creativity unfolds through this micro-iterative learning process: it is neither in the mind nor in the machine, but rather in the continuous scripted dialogue between the subject and their extensions. The computer is just one of those extensions, but a particularly powerful one, since it is, to use Alan Kay's term, a metamedium, that is, a medium capable of simulating all others.

As such the computer should be a thing that can be shaped and transformed. When it becomes less malleable, the computer is fixed within a stable media, which is perhaps more efficient, but also less surprising, less "creative." You do more but you learn less. Fundamentally, creativity is a question of time. Mostly of our daily activity with computers happens through hopefully speedy but ossified software. We use the computer in "speedrun mode." This is the paradox of creative coding: the coding part is supposed to make things faster, the creative part requires that things go slowly. According to permacomputing principles, one might say that Learn to Code is very "yang", and Code to Learn does also value the "yin": it "accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation."

LEARNING THROUGH COMPUTERS

Coding does not just manifest as a relationship between a user and a computer, but also between users through computers. Users exchanging techniques in real life or on Stack Overflow, appreciating each others' solutions, using coding as an excuse to just hang out, or building upon each other's tools.

The input of this process is patience and a capacity for listening; the output is fun and a sense of belonging. Coding can also be a bridge linking us to past users.

We see this in Ted Davis' assignment to recreate pioneering computer works or with the Re-Programmed Art Project, where a series of contemporary designers reinterpreted "analog" works of the Italian collective Gruppo T, active in the 1960s.

CODING AS A CRAFT

While Learn to Code turns coding into a resumé-ready skill, Code to Learn is about coding as a craft. My understanding of craft is wide-ranging: "a good job well done," as Sennett defines it. A craft is a savoir faire that is capable of stabilizing and consolidating one's identity.

In a time when designers are urged to constantly decorate their bio with strategic labels, a craft is a reflective activity, in the sense that the crafted things and the tools for crafting are a reflection of their maker, who generally recognizes themselves in them. This is also true of coding. As Roberto Arista, creator of the Python for Designers course, puts it: Programming then can become a way to escape [the confinement of desktop publishing software], connecting different regions and patiently rebuilding the workshop within the tools that effectively destroyed these regions.

The craftsperson enters their own physical or digital workshop—a local hackerspace, a custom i3 setup, a DIY CMS—and feels at home. This is where they code and learn, learn and code.

This is where they can forget, for a while at least, if they are lucky, the pressures and economic necessities of daily life.

Without neglecting Learn to Code's stressful refrain of employability and professional obsolescence, Code to Learn helps by considering the coding activity in itself, and not merely as an inevitable destiny. Opting for one model over the other in a graphic design school also means determining how to teach programming. When I speak of coding in and of itself I do not mean it as an index of technical notions (variables, loops, etc.). That is precisely the instrumental reduction of the Learn to Code attitude. Rather, I mean it in a broader sense: coding as a social activity and a cultural domain. This is what Code to Learn is all about.

CODING AS A PLAYGROUND
Graphic Design ; Post-Digital Age ; Learn to Code vs. Code to Learn: Creative Coding Beyond the Economic Imperative
Essay by Silvio Lorusso Demian Conrad Onomatopee, 2022

LEARN TO CODE
In 2021, the economic imperative to train and retrain has never been so strong. After the pandemic's dramatic impact on artists' economies, a skepticism about "creative" work is emerging, portraying it as unproductive daydreaming, and a wholly unessential industry. The emphasis is now on hard labor and effectiveness. The fundamental idea of Learn to Code is that the ability to program is a historical necessity for people working at a useless or obsolete job, and that these people must serve the economic imperatives of capitalism. This servitude is referred to as "retraining". Perhaps it is useful at this point to briefly reiterate the distinction between coders and programmers. While programmers are recognized as having an acknowledged and relatively arcane expertise with a correspondingly high salary, coding is increasingly perceived as semi-skilled labor. The programmer belongs to a profession, the coder to a workforce.
Back to design. Currently, public recognition of graphic designers is not so different from that of journalists. Both are now perceived as entire professions that it would be good to automate once and for all. Jobs meant to become buttons. According to this scenario, coding emerges as a professional panacea linked to the rhetoric of skill obsolescence and employability. Coding becomes a skill, in the most reductive sense of the term: something to add to your CV, better if exploded into discrete units. HTML and CSS: good, JavaScript: basic. The book Graphic Design Surveyed shows that US and UK students consider coding the third most useful skill to acquire (after networking and idea generation).
In 2014, German media theorist Florian Cramer dissected the various meanings of the term "post-digital". One of them was "the contemporary disenchantment with digital information systems and media gadgets." The Learn to Code meme suggests that disenchantment does not only revolve around the tools of the trade, but also around the trade itself. Coding, in the light of retraining, doesn't seem to be so emancipating.
Another understanding of post-digital Cramer highlighted has to do with the revival of old media. This might be a bit of a stretch, but what is more "old media", more 20th-century, than the idea of a workforce to be forged for the good of the nation? Of course, the Learn to Code narrative hints at the fact that jobs, skills and aspirations do not exist in a vacuum. However, due to a combination of disenchantment with programming and old-media labor rhetoric, coding emerges as a post-digital manifestation of capitalist realism, forcing graphic designers, journalists and coal miners alike to deal with their situations. All of them must go through mandatory updates, just like software. Is programming itself immune to this logic? Not really, it would seem, as the angelus novus of AI promises or threatens (depending on whom you ask) to automate the coder as well.

CODE TO LEARN

Ted Nelson once said: "The computer is as inhuman as we make it." The fact that coding is a cognitive activity does not make it intrinsically humane. It can also become a tedious, repetitive, industrialized exercise. This is what the jobs of a "new collar" workforce might look like, especially if such a workforce were a coercively retrained one. What then is coding beyond the not very empowering economic imperative of "learning to code"? Coding—whether it consists of writing HTML and CSS, posting a Javascript snippet on Github Gist, tweaking a Processing sketch, or publishing a Python module—can be not merely the content of a learning process, but its very medium. Thus, it can be a craft and a culture, or even better, a cultural meeting point. Who would we find there? A community of practice.

LEARNING WITH COMPUTERS
In a time when much is being said about the creativity of autonomous AI-powered machines, it is good to reconsider Licklider's notion of human-computer symbiosis. When you code, you instruct the computer to execute a more or less complex task, which is then immediately performed. You do not always know what to expect: the result might awe or disappoint you, allowing you to reorient or even redefine your initial goal. Part of the symbiosis is intrinsic to the language shared by user and machine—namely, code.
Creativity unfolds through this micro-iterative learning process: it is neither in the mind nor in the machine, but rather in the continuous scripted dialogue between the subject and their extensions. The computer is just one of those extensions, but a particularly powerful one, since it is, to use Alan Kay's term, a metamedium, that is, a medium capable of simulating all others.
As such the computer should be a thing that can be shaped and transformed. When it becomes less malleable, the computer is fixed within a stable media, which is perhaps more efficient, but also less surprising, less "creative." You do more but you learn less. Fundamentally, creativity is a question of time. Mostly of our daily activity with computers happens through hopefully speedy but ossified software. We use the computer in "speedrun mode." This is the paradox of creative coding: the coding part is supposed to make things faster, the creative part requires that things go slowly. According to permacomputing principles, one might say that Learn to Code is very "yang", and Code to Learn does also value the "yin": it "accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation."

LEARNING THROUGH COMPUTERS

Coding does not just manifest as a relationship between a user and a computer, but also between users through computers. Users exchanging techniques in real life or on Stack Overflow, appreciating each others' solutions, using coding as an excuse to just hang out, or building upon each other's tools. The input of this process is patience and a capacity for listening; the output is fun and a sense of belonging. Coding can also be a bridge linking us to past users. We see this in Ted Davis' assignment to recreate pioneering computer works or with the Re-Programmed Art Project, where a series of contemporary designers reinterpreted "analog" works of the Italian collective Gruppo T, active in the 1960s.

CODING AS A CRAFT
While Learn to Code turns coding into a resumé-ready skill, Code to Learn is about coding as a craft. My understanding of craft is wide-ranging: "a good job well done," as Sennett defines it. A craft is a savoir faire that is capable of stabilizing and consolidating one's identity. In a time when designers are urged to constantly decorate their bio with strategic labels, a craft is a reflective activity, in the sense that the crafted things and the tools for crafting are a reflection of their maker, who generally recognizes themselves in them. This is also true of coding. As Roberto Arista, creator of the Python for Designers course, puts it: Programming then can become a way to escape [the confinement of desktop publishing software], connecting different regions and patiently rebuilding the workshop within the tools that effectively destroyed these regions.
The craftsperson enters their own physical or digital workshop—a local hackerspace, a custom i3 setup, a DIY CMS—and feels at home. This is where they code and learn, learn and code.
This is where they can forget, for a while at least, if they are lucky, the pressures and economic necessities of daily life.
Without neglecting Learn to Code's stressful refrain of employability and professional obsolescence, Code to Learn helps by considering the coding activity in itself, and not merely as an inevitable destiny. Opting for one model over the other in a graphic design school also means determining how to teach programming. When I speak of coding in and of itself I do not mean it as an index of technical notions (variables, loops, etc.). That is precisely the instrumental reduction of the Learn to Code attitude. Rather, I mean it in a broader sense: coding as a social activity and a cultural domain. This is what Code to Learn is all about.

CODING AS A PLAYGROUND
Graphic Design ; Post-Digital Age ; Learn to Code vs. Code to Learn: Creative Coding Beyond the Economic Imperative
Essay by Silvio Lorusso Demian Conrad Onomatopee, 2022

LEARN TO CODE
In 2021, the economic imperative to train and retrain has never been so strong. After the pandemic's dramatic impact on artists' economies, a skepticism about "creative" work is emerging, portraying it as unproductive daydreaming, and a wholly unessential industry. The emphasis is now on hard labor and effectiveness. The fundamental idea of Learn to Code is that the ability to program is a historical necessity for people working at a useless or obsolete job, and that these people must serve the economic imperatives of capitalism. This servitude is referred to as "retraining". Perhaps it is useful at this point to briefly reiterate the distinction between coders and programmers. While programmers are recognized as having an acknowledged and relatively arcane expertise with a correspondingly high salary, coding is increasingly perceived as semi-skilled labor. The programmer belongs to a profession, the coder to a workforce.
Back to design. Currently, public recognition of graphic designers is not so different from that of journalists. Both are now perceived as entire professions that it would be good to automate once and for all. Jobs meant to become buttons. According to this scenario, coding emerges as a professional panacea linked to the rhetoric of skill obsolescence and employability. Coding becomes a skill, in the most reductive sense of the term: something to add to your CV, better if exploded into discrete units. HTML and CSS: good, JavaScript: basic.
The book Graphic Design Surveyed shows that US and UK students consider coding the third most useful skill to acquire (after networking and idea generation).
In 2014, German media theorist Florian Cramer dissected the various meanings of the term "post-digital". One of them was "the contemporary disenchantment with digital information systems and media gadgets." The Learn to Code meme suggests that disenchantment does not only revolve around the tools of the trade, but also around the trade itself. Coding, in the light of retraining, doesn't seem to be so emancipating.
Another understanding of post-digital Cramer highlighted has to do with the revival of old media. This might be a bit of a stretch, but what is more "old media", more 20th-century, than the idea of a workforce to be forged for the good of the nation? Of course, the Learn to Code narrative hints at the fact that jobs, skills and aspirations do not exist in a vacuum. However, due to a combination of disenchantment with programming and old-media labor rhetoric, coding emerges as a post-digital manifestation of capitalist realism, forcing graphic designers, journalists and coal miners alike to deal with their situations. All of them must go through mandatory updates, just like software. Is programming itself immune to this logic? Not really, it would seem, as the angelus novus of AI promises or threatens (depending on whom you ask) to automate the coder as well.

CODE TO LEARN

Ted Nelson once said: "The computer is as inhuman as we make it."

The fact that coding is a cognitive activity does not make it intrinsically humane. It can also become a tedious, repetitive, industrialized exercise. This is what the jobs of a "new collar" workforce might look like, especially if such a workforce were a coercively retrained one. What then is coding beyond the not very empowering economic imperative of "learning to code"? Coding—whether it consists of writing HTML and CSS, posting a Javascript snippet on Github Gist, tweaking a Processing sketch, or publishing a Python module—can be not merely the content of a learning process, but its very medium.

Thus, it can be a craft and a culture, or even better, a cultural meeting point. Who would we find there? A community of practice.

LEARNING WITH COMPUTERS
In a time when much is being said about the creativity of autonomous AI-powered machines, it is good to reconsider Licklider's notion of human-computer symbiosis. When you code, you instruct the computer to execute a more or less complex task, which is then immediately performed. You do not always know what to expect: the result might awe or disappoint you, allowing you to reorient or even redefine your initial goal. Part of the symbiosis is intrinsic to the language shared by user and machine—namely, code.
Creativity unfolds through this micro-iterative learning process: it is neither in the mind nor in the machine, but rather in the continuous scripted dialogue between the subject and their extensions. The computer is just one of those extensions, but a particularly powerful one, since it is, to use Alan Kay's term, a metamedium, that is, a medium capable of simulating all others.
As such the computer should be a thing that can be shaped and transformed. When it becomes less malleable, the computer is fixed within a stable media, which is perhaps more efficient, but also less surprising, less "creative." You do more but you learn less. Fundamentally, creativity is a question of time. Mostly of our daily activity with computers happens through hopefully speedy but ossified software. We use the computer in "speedrun mode." This is the paradox of creative coding: the coding part is supposed to make things faster, the creative part requires that things go slowly. According to permacomputing principles, one might say that Learn to Code is very "yang", and Code to Learn does also value the "yin": it "accepts the aspects that are beyond rational control and comprehension. Rationality gets supported by intuition. The relationship with the system is more bidirectional, emphasizing experimentation and observation."

LEARNING THROUGH COMPUTERS
Coding does not just manifest as a relationship between a user and a computer, but also between users through computers. Users exchanging techniques in real life or on Stack Overflow, appreciating each others' solutions, using coding as an excuse to just hang out, or building upon each other's tools.
The input of this process is patience and a capacity for listening; the output is fun and a sense of belonging. Coding can also be a bridge linking us to past users.
We see this in Ted Davis' assignment to recreate pioneering computer works or with the Re-Programmed Art Project, where a series of contemporary designers reinterpreted "analog" works of the Italian collective Gruppo T, active in the 1960s.

CODING AS A CRAFT
While Learn to Code turns coding into a resumé-ready skill, Code to Learn is about coding as a craft. My understanding of craft is wide-ranging: "a good job well done," as Sennett defines it. A craft is a savoir faire that is capable of stabilizing and consolidating one's identity.
In a time when designers are urged to constantly decorate their bio with strategic labels, a craft is a reflective activity, in the sense that the crafted things and the tools for crafting are a reflection of their maker, who generally recognizes themselves in them. This is also true of coding. As Roberto Arista, creator of the Python for Designers course, puts it: Programming then can become a way to escape [the confinement of desktop publishing software], connecting different regions and patiently rebuilding the workshop within the tools that effectively destroyed these regions.
The craftsperson enters their own physical or digital workshop—a local hackerspace, a custom i3 setup, a DIY CMS—and feels at home. This is where they code and learn, learn and code.
This is where they can forget, for a while at least, if they are lucky, the pressures and economic necessities of daily life.
Without neglecting Learn to Code's stressful refrain of employability and professional obsolescence, Code to Learn helps by considering the coding activity in itself, and not merely as an inevitable destiny. Opting for one model over the other in a graphic design school also means determining how to teach programming. When I speak of coding in and of itself I do not mean it as an index of technical notions (variables, loops, etc.). That is precisely the instrumental reduction of the Learn to Code attitude. Rather, I mean it in a broader sense: coding as a social activity and a cultural domain. This is what Code to Learn is all about.